Comparing Algorithms Approxmiating the Maximum Traveling Salesman Problem

by Feliks

Bachelor Thesis
Bachelor of Science
Major in Computer Science

First Supervisor: Prof. Dr. Lázló Kozma
Second Supervisor: Prof. Dr. Wolfgang Mulzer
Freie Universität Berlin, Institue for Computer Science

Date of Submission: September 10, 2025

Abstract

The approximation of the Travelling Salesperson Problem is a widely studied field where even simple approaches seem to promise good results. In this thesis, we examine two different approaches to approximating the Maximum TSP. The first, a Graph Patch Heuristic, attempts to find a solution by patching a maximum-weight cycle cover into a singular cycle. The second, called the Tunnelling Algorithm, builds on the idea of approximating the Euclidean norm with a metric defined by a polytope. Despite their drastically different methodologies, we find that both approaches reduce to a similar sub-problem and yield similar approximation ratios. While the Graph Patch Heuristic seems to outperform the Tunnelling Algorithm with larger input sizes and dimensions, we find that the Tunnelling Algorithm actually solves the Max TSP in polynomial time for common norms such as L_1 and L_∞. This thesis expands on some of the nuances of both algorithms, presenting them in a self-contained manner.

Contents

List of Notations

n	Input size of a problem
d	Dimensions of a space
$\mathbb{R}^d$	d dimensional space of rational numbers
dim	Doubling dimension
$dist(\cdot,\cdot)$	Distance function
G=(V,E)	A graph being defined by it's set of vertices and edges
$\rho(n)$	Approximation Ratio
$\rho(n)_{GPH}, \rho(n)_{TA}$	Approximation Ratio of GPH and Tunnelling Algorithm
$\rho(n)'$	$\rho(n)' = 1/\rho(n)$; Approximation ratio that is ≤ 1
δ	Relative Error
$Approx$	Value-result of approximation algorithm
OPT	Value of optimal solution
PTAS	Polynomial-time approximation scheme
APX	Class of approximation algorithms defined in introduction
$\mathcal{O}(\cdot)$	Standard Big O-Notation
$\mathcal{P}$	Set of cities
GPH	Graph Patching Heuristic
k_c	Number of cycles in a cycle cover
$Loss(e(a_1,b_1),e(a_2,b_2))$	Weight loss through patching two edges together
C_0	Initial maximum-weight cycle cover
C	Current cycle cover
Tunnelling Algorithm	Name for Algorithm approximating max TSP
Tunnelling Max TSP	A specified version of the Travelling Salesman Problem
$\mathcal{H}$	Set of halfspaces vectors defining a polytope
h	A halfspace vector
$\mathcal{T}, T, T'$	Set of tunnels
t	A tunnel
E'	A singular tour
E''	A partial tour $E'' \subset E'$; informally a guessed outline
F(c,t)	Access distance from city to front of tunnel defined as $t \cdot c$
B(c,t)	Access distance from city to back of tunnel defined as $-t \cdot c$
$x^B_{i,j}, x^F_{i,j}$	Decision variable for back/front tunnel for city c_j and tunnel t_i
f^B_i, f^F_i	Number used front and back tunnels t_i within a guessed outline
d_i	Number of visits of tunnel t_i
$f = 2 \cdot k$	Number of facets on a polytope
k	Number of vectors defining a polytope; number of tunnels
L_n, L_1, L_2, L_∞	A few L-norms
$\mathcal{B}$	Ball or unit ball
$\mathcal{B}_x : x \in \{1,2,l,g\}$	Different (unit) balls
λ	Shrink/Grow factor of a unit ball
$a \to b(c \to d)$	Equivalent to limit $lim_{c \to d} a = b$

1 Introduction

Since the relationship between the set of Problems in P, solvable in polynomial time, and Problems in NP, the set of problems where the solutions can be verified in polynomial time, is unclear, there exists great interest in finding approximation algorithms solving NP and NP-hard problems.

An approximation algorithm is supposed to guarantee a better runtime, with the trade-off that the solution may not be optimal. Therefore, there are at least two factors to compare approximation algorithms by, runtime and how close the solution is to the optimal solution. There are multiple ways to evaluate the accuracy of an algorithm, in some cases it can be an additive factor, but most commonly, the accuracy is measured with an approximation ratio $\rho(n)$, where the value of a solution $Approx$ provided by an approximation algorithm is within a factor of $\rho(n)$ to the actual optimal value OPT for any input size n. Conventionally $\rho(n) \geq 1$ by defining it as $\frac{OPT}{Approx}$ for maximization problems and $\frac{Approx}{OPT}$ for minimization problems, or generalizing to $max(\frac{OPT}{Approx}, \frac{Approx}{OPT}) \leq \rho(n)$[5, p. 1108]. Some authors, such as Shenmaier[3] have used a relative error δ, where $\delta = \frac{OPT-Approx}{OPT}$ to describe the accuracy of the approximation algorithm. The APX class is a category of algorithms that approximate an optimisation problem in NP in polynomial runtime with an approximation ratio that is constant.

The category of polynomial-time approximation schemes, short PTAS, is a category of algorithms to approximate a solution for an NP optimisation problem within a factor of $1 + \epsilon$ for minimization or $1 - \epsilon$ for maximization problems for $\epsilon > 0$ with a runtime that is polynomial in size n and dependant on ϵ and also allows to be exponential in ϵ. Since both the runtime and the accuracy depend on ϵ the algorithm can get arbitrarily close to the optimal solution, however the closeness to that solution will also impact the runtime of the algorithm. Note that the conventional definition of PTAS mentioned above may clash with the conventional definition of the approximation ratio, meaning in case of the approximation ratio for the PTAS we may use $\rho'(n) = 1/\rho(n)$.

Maximum TSP: Given a set $\mathcal{P}$ of n points/cities $c \in \mathbb{R}^d$ in a metric space $(X = \mathcal{P}, dist(\cdot, \cdot))$, implying that every distance between a pair of points is defined, a tour is a path that visits each point exactly once and ends on the same point where it started. This tour can be expressed as a graph $G = (V = \mathcal{P}, E)$ where G is connected and every point is exactly incident to two edges of E, where the weight of the edges are defined by the distance function $dist(\cdot, \cdot)$. The Maximum Travelling Salesman Problem is an optimisation Problem of finding a Graph G representing a tour over the given points $\mathcal{P}$ with the largest possible sum of all edge weights E.

Most commonly when referring to the TSP, it is assumed that we are in search for a minimum tour over cities that are either vertices on a graph or points in a metric space under Euclidean norm. It is well established that for either case it is well known to be NP-hard[6]. Additionally it has been so far proven that for any fixed dimension d and any L_p norm the problem remains NP-hard. Barvinok et al.[2] show in the paper from 2003 that even though max TSP is also NP-hard under Euclidean distance in dimension $d \geq 3$, it also is solvable in polynomial time for any dimension d for some L_p norms, as we will see in section 3.

The goal of this thesis is to review and explain two different algorithms in two different approximation classes and compare their results. Although arguing some of the more complex and optimized results may be out of the scope of this thesis, we will attempt to make the topic at hand as self-contained as possible. Section 2 will review an algorithm that approximates the max TSP by patching a maximum-weight cycle cover. However this thesis will mostly focus on the Tunnelling Algorithm in section 3 that finds a solution by approximating the unit ball of the Euclidean norm with a polytope, where we enumerate over how many possible outlines we can

guess and solving each one with a linear program.

Another big part of this thesis is the comparison in section 4 where compare the results of both the algorithms in the 2 dimensional Euclidean space.

1.1 Bipartite Matching

Before we start looking into the algorithms approximating TSP, we will look at the Bipartite Matching Problem. Interestingly, even though, at the time of writing, both approaches appeared to be drastically different, they both turn out to reduce to a bipartite matching problem. Therefore we dedicate this section to shortly define this problem as well as give an algorithm to solve the special case of it.

A Graph $G = (V \cup U, E)$ shall be called bipartite if V and U are disjunct and there is no $e(x, y) \in E$ s.t. x and y are in the same set. A matching M is a subset of E where for every vertex $x \in V \cup U$ there is no more than one edge $e \in M$ that is incident to x.

While there are many ways to specify this problem to adapt it to different application, in this thesis we will be particularly interested in the case where we can assume that $|V| = |U|$ and every $e \in E$ has a weight $w(e) \geq 0$. Further we want to find a minimum-weight matching, where we minimize $\sum_{e \in M} e$.

As this subproblem is not the main focus of this thesis we will look at a simplified version of solving it based on Munkres Hungarian Algorithm[15]. For further details we also direct the reader to text book Assignment Problems by Burkard, Dell'Amico, Martello[7, p.79].

As both sets are of size n we define a $n \times n$ matrix A with $a_{i,j}$ being equal to the cost of the edge between v_i and u_j. Our goal will be to process the matrix in such a way to create entries of value 0 at which point we know that we can use this entry as part of the matching.

Step 1: For every row i find an entry of minimum value and subtract it from every entry of this row. This will create at least one zero entry in each row.

Step 2: Check whether we can find a perfect matching by only using edges $e(v_i, u_j)$ for which $a_{i,j} = 0$. If not continue to step 3. This can be done by drawing horizontal lines and vertical lines through the matrix. If the minimal number of lines we can draw s.t. every zero entry is covered is less then n this means there is no matching yet.

Step 3: For every column j find an entry of minimum value and subtract it from every entry of this column.

Step 4: Do the same as step 2 checking whether there is a matching.

Step 5: Given we have the minimum number of horizontal and vertical lines covering our matrix, we want to find the minimum entry that is not covered, subtract it from every entry that is also not covered and add it to the entries that are covered by two lines. Again find a minimum number of lines that cover all 0 entries. Repeat this step till the minimum number of lines are exactly n

After the Algorithm finishes it's possible to select n zero values s.t. the set of edges represented by the entries are a perfect matching. While this implementation might not be optimal having been described as having a runtime of $\mathcal{O}(n^4)$, there is other interpretation and implementation of the Hungarian Algorithm solving the bipartite matching in $\mathcal{O}(n^3)$.

2 Greedy Patch Heuristic

We will start with the simpler one of the two algorithms that finds a tour by first finding a maximum-weight cycle cover and then combining it into one cycle creating a tour. The results of this section are mainly focused on the paper by Shenmaier(2022)[3], where he describes an algorithm for which he estimates a constant

approximation ratio, but the more interesting result is that he proves a relative error that improves with input size. The algorithm itself is a modified version of the greedy Karp-Steel algorithm where Shenmaier[3] references papers that analyse such greedy Karp-Steel algorithm in the context of Minimum TSP. For more exact steps of the algorithm we will mainly use details from a 1999 paper by Clover et al.[8], where we will of course modify the proposed algorithm to work for the Maximum TSP. Additionally the Hungarian Algorithm from Section 1.1 will find its use in this algorithm. Tracking down the references of various papers discussing such a patching algorithm we can find that the idea dates back to at leasts a 1976 paper by Karp[9], as well as another paper we consider from 1985 by Karp and Steel[10].

There is another paper by Shenmaier(2021)[4] where Shenmaier provides a similar algorithm slightly modified to allow it to be a PTAS. In this thesis however it was decided to instead look at results of the 2022 paper [3], firstly for the sake of simplicity of the algorithm but also to use this algorithm as an example of an APX algorithm. Additionally, a point of interest would be how to compare an algorithm whose approximation ratio is dependant on the input size to a PTAS, as is the algorithm of section 3.

2.1 The Algorithm

The Greedy Patch Heuristic, as called by Shenmaier[3], can ultimately by summed up in two steps.

1. Find a maximum-weight cycle cover $C = C_0$
2. While there is more then one cycle in C find two edges part of different cycles and patch them such that the loss of weight by the patching is minimized

In his paper, Shenmaier[3] assumes that the set of cities is provided in a metric space, where the distance function satisfies the metric axioms, which obviously also applies to the metric induced by the Euclidean norm. Additionally, he provides the definition of doubling dimension where $dim \geq 0$ is the smallest number for which in the metric space of doubling dimension dim any ball can be covered by 2^{dim} balls of half the radius.

As the goal of this thesis is primarily to compare the algorithms that approximate the Max TSP under Euclidean norm, we can draw a connection between a metric space of doubling dimension and the Euclidean space. We can easily show that for the 1-dimensional Euclidean space we need 3 balls two cover a ball of twice the radius, therefore resulting in a doubling dimension of $dim_1 = log(3)$. For the 2-dimensional Euclidean space we need 7 balls (as seen in figure 1), resulting in a doubling dimension of $dim_2 = log(7)$. We can imagine by adding another dimension we will need at most the squared number of balls compared to the previous dimension. Therefore the relationship between the number of balls is exponential in d but also exponential in dim as per definition. Therefore the doubling dimension is at the very least $dim \in \mathcal{O}(d)$, but commonly assumed to be $dim \in \theta(d)$[12; 13].

2.2 Maximum-Weight Cycle Cover

A cycle cover is a subset of edges of a given graph where each city of the set of points is incident to exactly to edges, the difference to a Hamiltonian Cycle being that a cycle cover does not have to be connected, resulting in a subset of edges that can form more than one cycle. Therefore a maximum-weight cycle cover aims to maximize the weight of all the edges part of such a cycle cover. Such a maximum-weight cycle cover can be found efficiently as we will see in this section.

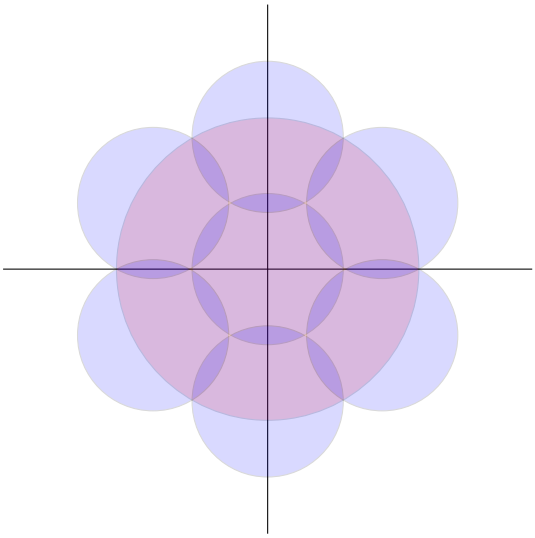

Figure 1: Covering a ball with 7 balls of half it's radius in the plane

While Shenmaier[3] cites 1983's Gabow[11] we can also instead solve the maximum-weight cycle cover with the Hungarian Algorithm as mentioned in section 1.1. For this we can simply split every $c_i \in \mathcal{P}$ into two vertices u_i, w_i. We want to make this graph bipartite, therefore there are no edges between u_i and u_j, as well as no edges between w_i and w_j for $i, j \in \{1, 2, \ldots, n\}$. The graph shall have edges between u_i, w_j for $i \neq j; i, j \in \{1, 2, \ldots, n\}$. Since the Hungarian Algorithm aims to minimize and our goal is to maximize, we need to modify the edge weights between cities, by finding the maximum distance $dist_{max}$ and defining the weights as $w(e(u_i, w_j)) = dist_{max} - dist(c_i, c_j)$. Another modification we make is that the entries between u_i and w_i stay at infinity, or a dummy value. Meaning when we do subtraction steps as described previously the dummy values will be ignored. This is to avoid making cycles made of a single city.

Informally we can interpret this setup as all the cities on both sides of the graph where every u_i vertex is searching to be matched with a w_j vertex where if this edge ends up in the matching it means that for the maximum-weight cycle cover the next city after c_i is c_j. Further the follower of city c_j will be equivalent to the edge in the resulting matching that is incident to u_j. And we stop u_i matching with w_j if $j = i$.

We can easily see that both partitions of the bipartite graph have exactly n vertices. Therefore the Hungarian Algorithm gives us a result in $\mathcal{O}(n^3)$ time. The resulting matching will translate to a set of edges where every city has an outgoing and an ingoing edge, therefore making it exactly incident to two edges. However we don't guarantee that the graph induced by the edges of the matching is connected.

It is important to mention that for the analysis Shenmaier[3] assumes that a cycle shall have at least 3 vertices by definition. Our method however allows for any cycle to consist of 2 vertices. This of course impacts what the upper bound for the number of cycles k_c is. With the method presented in this thesis $k_c \leq n/2$, while with Shenmaiers assumption we get a better upper bound of $k_c \leq n/3$. We will later mention this again when it comes to the discussion of the resulting approximation ratio.

2.3 Patching

This step is not explicitly described by Shenmaier[3] but we will instead explain it as it is described by Glover et al.[8]. The latter is a paper that, describes this algorithm in the context of Min TSP as well as reviews

some computational experiments. For this thesis we will leave out some minor optimisation steps, for the sake of simplicity as well as modify it to work on Max TSP.

The difficult part of the patching step is of course to find the optimal edges to patch to minimize loss of weight. Assuming that $a_1, b_1, a_2, b_2 \in \mathcal{P}$ where the edges $e(a_1, b_1)$ and $e(a_2, b_2)$ are in different cycles but within one cycle cover, we replace $e(a_1, b_1), e(a_2, b_2)$ by either $e(a_1, b_2), e(a_2, b_1)$ or $e(a_1, a_2), e(b_1, b_2)$ depending on which of these sets maximizes the weight and minimizes the loss. The loss by this patching shall be defined as:

$$Loss(e(a_1, b_1), e(a_2, b_2)) = dist(a_1, b_1) + dist(a_2, b_2) - max\{dist(a_1, b_2) + dist(a_2, b_1), dist(a_1, a_2) + dist(b_1, b_2)\}$$

As we have previously discussed how to find a cycle cover, we can use the data from the matching by saving what city is next relative to what city, meaning $next(c_i) = c_j$ if $e(u_i, w_j)$ was in the resulting matching. Additionally we can easily find what city belongs to what cycle in $\mathcal{O}(n)$ space as well as time, by just following the edges part of the cycle cover, switching as soon as we visit a city that already has been visited. Glover et al.[8] describe an efficient way of keeping track of what edges to patch by using an $n \times n$ matrix W, where $w_{i,j}$ represents the loss between patching the $e(c_i, next(c_i))$ and $e(c_j, next(c_j))$. We can set up such matrix in $\mathcal{O}(n^2)$ as for every entry $w_{i,j}$ we either set the distance to infinity if c_i and c_j are part of the same cycle or we calculate the loss function in $\mathcal{O}(1)$.

After every patch we have to update the the loss matrix with an upper bound of $\mathcal{O}(n^2)$. Here we need to set the loss to infinity for every edge that now share the same cycle, as well as update the loss for at most $\mathcal{O}(n^2)$ entries. While it seems that the loss entries that we have to update maybe done in $\mathcal{O}(n)$ since we only need to touch costs for entries of where at least one of the cities was involved in the patching, it may become complicated when we patch for instance $e(c_i, next(c_i))$ and $e(c_j, next(c_j))$ with $e(c_i, c_j)$ and $e(next(c_i), next(c_j))$. Therefore having to set $next(c_i) = c_j$ which will cause us to have to change the direction of the old cycle of c_j. Thus we will be conservative by estimating that we have to update at most $\mathcal{O}(n^2)$ entries.

As mentioned in the previous section we have a maximum number of cycles in the cycle cover using our method of $k_c \leq n/2$ or $k_c \leq n/3$, as described by Shenmaier[3]. Concluding that we have to perform k_c patches, where the runtime of every patch is $\mathcal{O}(n^2)$, altogether resulting in a runtime of $\mathcal{O}(n^3)$ of the patching step.

2.4 Results

Ultimately, as the runtime of the maximum-weight cycle cover is $\mathcal{O}(n^3)$ and the runtime of patching the cycle cover is $\mathcal{O}(n^3)$, the GPH has a runtime of $\mathcal{O}(n^3)$.

Now that we know how the algorithm works, we want to look at the approximation ratio. Obviously we can imagine that given a Hamiltonian cycle that solves the TSP such a tour is also a cycle cover of a given graph, therefore a cycle cover being a relaxation of a Hamiltonian cycle, we can easily see that the the weight of a maximum-weight cycle cover is an upper bound to the Hamiltonian cycle. Shenmaier[3] gives us a worse case of the approximation ratio by approximating how much weight is lost to the cycle cover, without mentioning how tight the weight of the maximum-weight cycle cover may be compared to the optimal tour.

For this Shenmaier[3] argues that, given the cycle cover C, there will always be a patching between the lightest edge $\tau = e(a_1, b_1)$ and any other edge $e(a_2, b_2)$ where the weight lost is at most $w(\tau) \geq Loss(e(a_1, b_1), e(a_2, b_2))$. Further as we assume that τ is the lightest edge the weight of such edge is at most $w(\tau) \leq w(C)/n$. Therefore $w(C)/n \geq w(\tau) \geq Loss(e(a_1, b_1), e(a_2, b_2))$. Shenmaier argues this by saying it's easily implied through triangle inequality, however we will expand on this proof.

Proof: Assuming $\tau = e(a_1, b_1)$ is the lightest edge in C, we can easily see that $\tau \leq w(C)/n$, as it is at most equal to the weight of every other edge. Given any edge $e(a_2, b_2)$ that is part of the cycle cover but not the same cycle as τ, in a patching we have the option to replace $e(a_1, b_1)$ and $e(a_2, b_2)$ with either the set $e(a_1, b_2), e(a_2, b_1)$ or $e(a_1, a_2), e(b_1, b_2)$. One of the two edge sets is either equal or greater in sum compared to the other set and will therefore take effect on the calculation of the loss. Assuming $w(e(a_1, a_2)) + w(e(b_1, b_2)) \geq w(e(a_1, b_2)) + w(e(a_2, b_1))$ we have a loss of:

$$Loss(\tau, e(a_2, b_2)) = \tau + w(e(a_2, b_2)) - w(e(a_1, a_2)) - w(e(b_1, b_2))$$

In this instance we differentiate between two cases. As $w(e(a_1, a_2))+w(e(b_1, b_2)) \geq w(e(a_1, b_2))+w(e(a_2, b_1))$ one of the edge of the left pair has to be at least equal or larger then one on the right pair. The two cases we have is either case 1 $w(e(a_1, a_2)) \geq w(e(a_2, b_1))$ or case 2 $w(e(b_1, b_2)) \geq w(e(a_1, b_2))$. There is no third case, since if we negate the conditions of case 1 and case 2 we arrive at a contradiction. As, when $w(e(a_1, a_2)) < w(e(a_2, b_1))$ and $w(e(b_1, b_2)) < w(e(a_1, b_2))$ hold true then this implies that $w(e(a_1, a_2)) + w(e(b_1, b_2)) < w(e(a_1, b_2)) + w(e(a_2, b_1))$, which is a contradiction.

It's left to show that for both case 1 and case 2 $Loss(\tau, e(a_2, b_2)) \leq w(\tau)$. For this we will use the fact that the edge weights are defined by a metric, therefore the triangle inequality applies. Case 1:

$$\begin{aligned}
&\textit{assuming: } w(e(a_1, a_2)) \geq w(e(a_2, b_1)) \\
&\textit{assuming: } w(e(a_2, b_2)) \leq w(e(b_1, b_2) + w(e(a_2, b_1)
\end{aligned}$$

$$\begin{aligned}
Loss(\tau, e(a_2, b_2)) &= \tau + w(e(a_2, b_2)) - w(e(a_1, a_2)) - w(e(b_1, b_2)) \\
&\leq \tau + w(e(b_1, b_2) + w(e(a_2, b_1)) - w(e(a_1, a_2)) - w(e(b_1, b_2)) \\
&= \tau + w(e(a_2, b_1)) - w(e(a_1, a_2)) \\
&\leq \tau \\
\implies Loss(\tau, e(a_2, b_2)) &\leq \tau
\end{aligned}$$

The first assumption is of course derived by the definition of the case and the second one is obviously implied by the triangle inequality. First we use the Triangle inequality and at the final step the assumption given by the case. Case 2 can easily be performed symmetrically to case 1.

$$\begin{aligned}
&\textit{assuming: } w(e(b_1, b_2)) \geq w(e(a_1, b_2)) \\
&\textit{assuming: } w(e(a_2, b_2)) \leq w(e(a_1, a_2) + w(e(a_1, b_2)
\end{aligned}$$

$$\begin{aligned}
Loss(\tau, e(a_2, b_2)) &\leq \tau + w(e(a_1, a_2) + w(e(a_1, b_2)) - w(e(a_1, a_2)) - w(e(b_1, b_2)) \\
&= \tau + w(e(a_1, b_2)) - w(e(b_1, b_2)) \leq \tau
\end{aligned}$$

Clearly we can do the same proof if we would assume that the other patching would cause less loss. This concludes the proof of $Loss(\tau, e) \leq w(C)/n$. □

With this lemma proven, Shenmaier[3] gives us an argument for a constant approximation ratio of this algorithm. While he also proves it, we would like to expand on this step.

Given we have a maximum-weight cycle cover C_0 with k_c number of cycles. We perform $k_c - 1$ number of patching operations after which the cycle cover is also a Hamiltonian Cycle. Let's call $w(C_{k_c-1})$ the weight of the final cycle cover after $k_c - 1$ patches. We can observe that we can calculate the weight of the cycle cover C_1 after 1 patching step compared to the weight of C_0 as we know how much weight we lose at most.

$$w(C_1) \geq w(C_0) - w(C_0)/n$$
$$w(C_1) \geq w(C_0)(1 - 1/n)$$

we can generalize this with:

$$w(C_i) \geq w(C_{i-1})(1 - 1/n), \quad i = 1, \ldots, k_c - 1$$

This mean for every patching the weight of the next cycle cover is at most the previous weight multiplied by $1 - 1/n$. Thus:

$$w(C_{k_c}) \geq w(C_0)(1 - 1/n)^{k_c-1}$$
$$w(C_{k_c})/w(C_0) \geq (1 - 1/n)^{k_c-1}$$

We recall that the definition of the approximation ratio is $OPT/Approx$. In this case $Approx = w(C_{k_c})$ as it is the weight of the Hamilton Cycle and therefore an approximation for a Max TSP tour. While $w(C_0)$ is not the optimal value of a solution for Max TSP, but it is an upper bound, meaning that the approximation ratio will be at least as good.

$$1/\rho(n) \geq w(C_{k_c})/w(C_0) \geq (1 - 1/n)^{k_c-1}$$

We will use the assumption made by Shenmaier[3] that $k_c \leq n/3$ because the maximum-weight cycle cover algorithm he uses allows to find cycles with minimum 3 vertices in one cycle. Note that with our algorithm $k_c \leq n/2$, as our method allows for a cycle to have minimum 2 vertices, we will however use the better result to work with.

Additionally Shenmaier[3] simplifies this equation by using an estimate for the Euler Number, as:

$$lim_{n\to\infty}(1 + k/n)^n = e^k$$

From this we can assume that $(1 + k/n)^n \leq e^k$ allowing us to do this estimation:

$$
\begin{aligned}
1/\rho(n) &\geq (1-1/n)^{k_c-1} \geq (1-1/n)^{k_c} \\
&= (1+(-1)/n)^{k_c} \\
&\geq (1+(-1)/n)^{n/3} = ((1+(-1)/n)^n)^{1/3} \\
&\geq (e^{-1})^{1/3} \\
&\geq e^{-1/3} \\
\implies \rho(n) &\leq e^{1/3}
\end{aligned}
$$

Finally giving us an upper bound for the approximation ratio of $e^{1/3} \approx 1.3956$.

Another interesting result of Shenmaier[3] is that he proves a relative error that is dependant on the doubling dimension dim as well as the input size n. While it would be interesting to fully explain the proof, this is however out of the scope of this thesis, as the proof seems to quite lengthy an abstract. Nevertheless we will want to compare the GPH and the Tunnelling Algorithm with this relative error in mind.

Summed up the argument for the relative error Shenmaier[3] gives is establishing a ball dependant on a variable ρ around the a vertex of the initial shortest edge. Continuing, by defining 3 types of patches, where the first patch type is a patch of cycles at least one of which is far relative to the ball. The second type patches cycles that are both considered near with a loss of a certain threshold. And the third type covers the rest, meaning a patching between cycles that are considered near while surpassing the threshold for the second case.

As the ball as well as the definition of near and far are defined based on two different variables, Shenmaier[3] arrives at a conclusion having argued the loss caused by every type of patching and provides us with an relative error of the GPH by selecting a clever value of the open variables, ultimately dependant on the doubling dimension dim.

The final relative error is $\delta = \frac{7/3}{n^{1/(2dim+1)}}$. We want to make one more step, where we calculate the approximation ratio based on this given relative error as following relation ship holds true $\delta = \frac{OPT-Aprox}{OPT}$, and therefore

$$
\begin{aligned}
\delta &= \frac{OPT - Approx}{OPT} \\
OPT \cdot \delta &= OPT - Approx \\
OPT \cdot (\delta - 1) &= -Approx \\
1 - \delta &= \frac{Approx}{OPT} \\
\frac{OPT}{Approx} &= (1-\delta)^{-1} \\
\rho(n) &= (1-\delta)^{-1}
\end{aligned}
$$

Therefore we can get an approximation ratio of $\rho(n)_{GPH} = (1 - \frac{7/3}{n^{1/(2dim+1)}})^{-1}$. We can observe that $\delta \to 0 (n \to \infty)$ and therefore as n approaches infinity the approximation ratio get closer to on meaning it improves. Of course the curse of dimensionality holds true also for this algorithm, as $dim \in \theta(d)$, as doubling dimension increases so does the approximation ratio deteriorate as the relative error starts approaching 0 slower

and therefore the approximation ratio also approaches 1 slower as a consequence. More practical discussion will be done in section 4.

3 Tunnelling Algorithm

This section will present an approach where we will solve the Max TSP under a special norm called the polyhedral norm which will be explained in the coming sections. We will see that a solution of the Max TSP under this norm will also be a reasonably good solution for the Max TSP under Euclidean norm.

Further we will break down the problem of Max TSP under polyhedral norm to a Tunnelling Max TSP for which we will present two solution, where Woeginger[1] will give us a more simplified approach, whereas Barvinok et al.[2] will be more complicated but will also provide us with a better runtime.

Within this thesis we will refer to the algorithm described by Barvinok et al.[2] as the Tunnelling Algorithm, promising a runtime of $\mathcal{O}(n^{2k-2}log(n))$. This should not be confused with the special case of TSP called the Tunnelling Max TSP, which will be solved by the Tunnelling Algorithm.

3.1 The Approximation Idea

It is well known that any norm in a normed space can be represented by a unit ball $\mathcal{B}$. Looking at the L_2 or commonly known as the Euclidean norm we can calculate a distance between two points $x, y \in R^d$ with $dist_{L_2}(x, y) = ||x - y||_2 = (\sum_i (x_i - y_i)^2)^{1/2}$. We can also geometrically deduct this distance through the unit ball $\mathcal{B}_{L_2}$, a circle with a radius of 1 unit length, by centring it on x. The distance from x to y will be the smallest possible factor λ with which we shrink or expand the unit ball such that y lies within the unit ball (see figure 2).

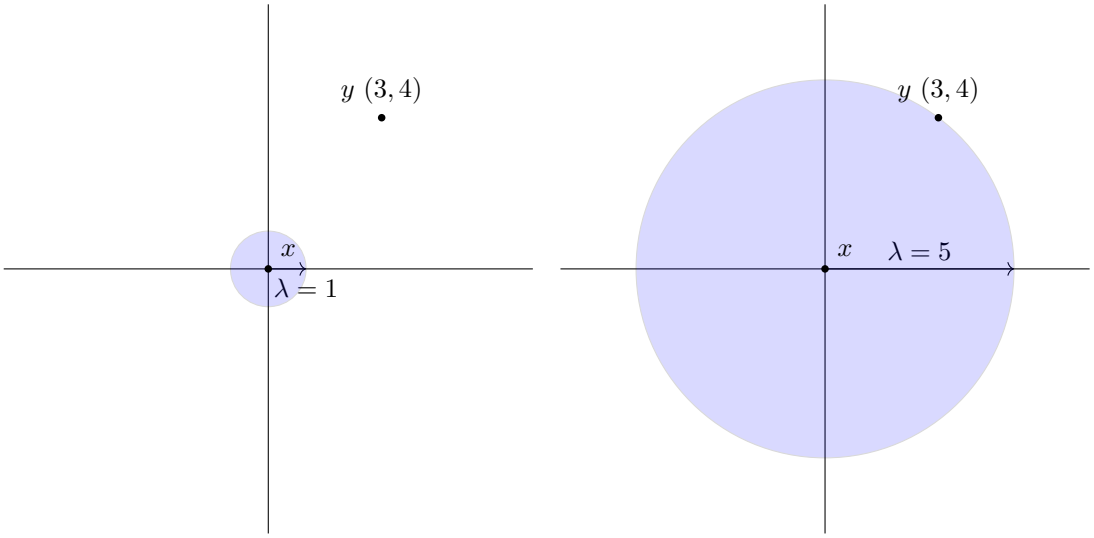

Figure 2: Visualization of a Euclidean unit ball where distance is 5 between x and y.

The central idea of the Tunnelling Algorithm is to approximate the unit ball $\mathcal{B}_{L_2}$ of the Euclidean norm by an appropriately chosen polytope. The algorithm will be able to find a correct solution in polynomial time for the given polyhedral norm, therefore giving us an approximation for the optimal solution under the Euclidean norm.

Woeginger[1] presents a crucial observation that ties the choice of the approximating unit ball to the approximation ration. We will come back to the in Section 4. Summed up we arrive at:

Crucial Observation 1: Given two unit balls $\mathcal{B}_1$ and $\mathcal{B}_2$ where $(1-\epsilon)\mathcal{B}_2 \subseteq \mathcal{B}_1 \subseteq (1+\epsilon)\mathcal{B}_2$. Assuming that a geometric maximum optimisation problem under $\mathcal{B}_2$ can be solved in polynomial time, then the optimisation problem under $\mathcal{B}_1$ has an polynomial time approximation algorithm with the worst case guarantee $(1-\epsilon)/(1+\epsilon)$.

The algorithm will demonstrate that, although max TSP is not solvable in polynomial time in any fixed dimension under Euclidean norm, we can find a solution for a norm with a polytope as a unit ball under any fixed dimension. This type of norm is called **polyhedral**. Although not necessarily a characteristic of polytopes, the unit ball of a norm is required to be point symmetric and therefore we can derive that the number of facets f will be divisible by 2. Furthermore we can represent such a polytope by a set $\mathcal{H}$ of size $k = f/2$ halfspaces with $\mathcal{H} = \{h_1, \dots h_k\}$ where x is inside the polytope when the following formula holds true for every $i \in \{1, 2, \dots, k\}$:

$$h_i \cdot x \leq \ 1$$
$$h_i \cdot x \geq -1$$

We shall refer to the vectors $h_i \in \mathcal{H}$ as halfspace vectors, as they each define two halfspaces as per the formula above. Obviously a halfspace vector is orthogonal to the halfspace it defines.

Of the norms that fall within the polyhedral norms, we have most notably the L_1 and L_{max} norm or commonly known as Manhattan/Rectilinear and the Uniform/Supremum norm, which in 2 dimensions are represented by two different squares, only difference being that the unit ball of the Rectilinear norm is rotated by 45 degrees. The halfspaces then are represented by the vectors {(1,1),(-1,1)} for the Rectilinear and {(1,0),(0,1)} for the Supremum norm respectively (see figure 3 and 4). In contrast to the previous figure the blue area now signifies the area that is not included in the halfspace cross-section in the first 3 depictions.

We can confirm the correctness of these norms by doing some simple examples. In the case of the Supremum norm each vector just yields one component while the others are nullified. In the case of the Rectilinear norm, if both components are either negative or positive the distance would be the result of the absolute number of the sum of components, while the halfspace vector $(-1, 1)$ covers the case where one component is negative and one is positive.

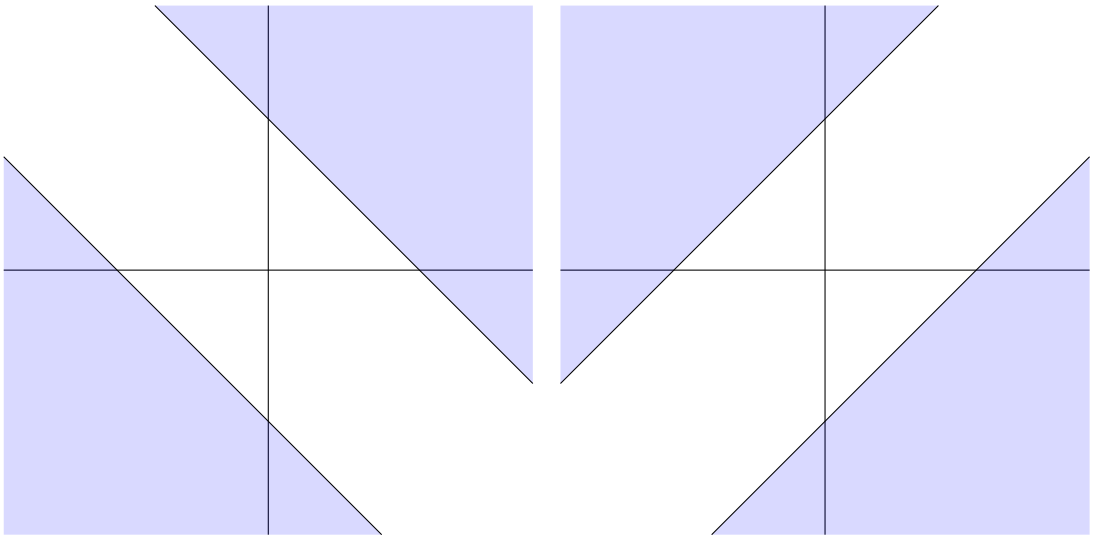

Figure 3: Left: two halfspaces created by the vector (1,1), Right: two halfspaces created by the vector (-1,1)

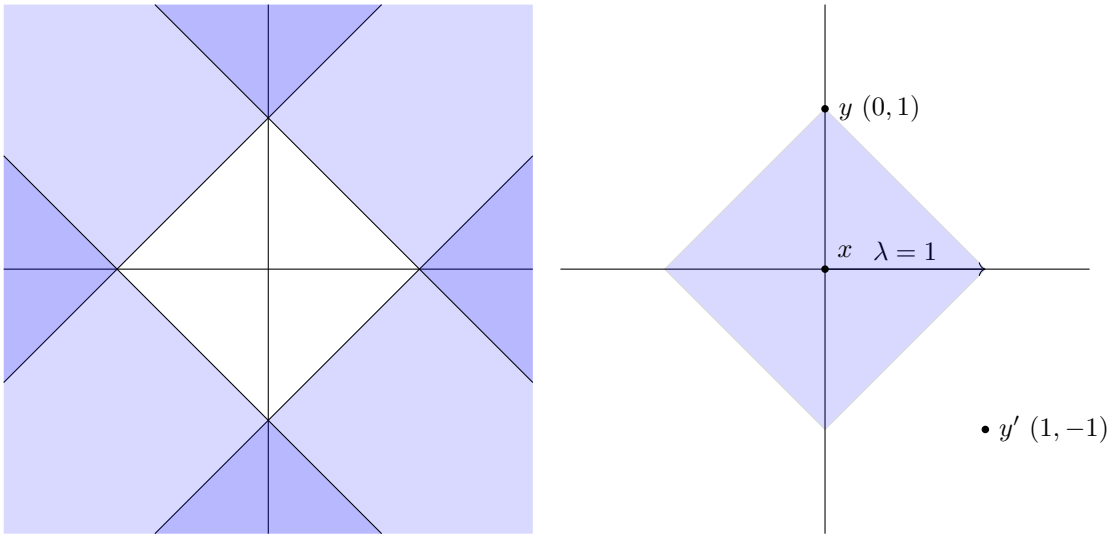

Figure 4: Left: the unit ball created by both vectors, Right: Visualizing distance between x and y

As previously observed the distance between two points $dist(x, y)$ based on a norms unit ball is the smallest λ for which y is within the unit ball that is centred on x. Woeginger[1] as well as Barvinok et al.[2] provide us some useful equivalences for the distance under the polyhedral norm:

$$
\begin{aligned}
dist(x,y) = & min\{\lambda : y \in x + \lambda\mathcal{B}\} \\
& min\{\lambda : y - x \in \lambda\mathcal{B}\} \\
& min\{\lambda : |h_i(y-x)| \leq \lambda : 1 \leq i \leq k\} \\
& max\{|h_i(x-y)| : 1 \leq i \leq k\} \\
& max\{h_i(x-y),\ h_i(y-x) : 1 \leq i \leq k\} \\
& max\{h_i \cdot x - h_i \cdot y,\ h_i \cdot y - h_i \cdot x : 1 \leq i \leq k\} \qquad (1)
\end{aligned}
$$

3.2 The Tunnelling Max TSP

The Tunnelling Max TSP is a specified version of the max TSP, where the goal is to find the highest weighted cycle that visits every city, however each transition from one city to the next city has to pass through one of k tunnels. The weight of the cycle will be defined by the distance between city and tunnel, while the tunnel itself does not add any weight. This means when finding a path between two cities c and c' through the tunnel t we have two options based on whether we first go through the front or the back exiting on the opposite side. Since we're primarily interested in maximizing the weight we can establish that the distance between those two cities through the tunnel t shall be:

$$max\{F(c,t) + B(c',t),\ F(c',t) + B(c,t)\}$$

Further, we can consider every other tunnel in the set of tunnels, where the largest transition through a singular tunnel between city c and c' shall be:

$$max\{F(c,t_i) + B(c',t_i),\ F(c',t_i) + B(c,t_i) : 1 \leq i \leq k\} \qquad (2)$$

We can establish a connection between the Tunnelling Max TSP and the Max TSP under polyhedral norm when we set the tunnel distances to $F(c,t) = c \cdot t$ and $B(c,t) = -c \cdot t$. Therefore (2) can be further simplified to:

$$max\{t_i \cdot c - t_i \cdot c',\ t_i \cdot c' - t_i \cdot c : 1 \leq i \leq k\} \qquad (3)$$

It's important to note that equation 1 and equation 3 signify two different things, even though we end up with a similar looking formula. While equation 1 gives a definition of what a distances is under the polyhedral norm, equation 2 and equation 3 gives us the largest distance that we can choose between two cities under the Tunnelling Max TSP. But we can use the equality of equation 1 and equation 3, assuming that the set of Tunnels $\mathcal{T}$ is equal to the set of halfspace vectors $\mathcal{H}$, that define the polyhedral norm, to argue that an optimal tour for the Tunnelling Max TSP is also an optimal solution for the Max TSP under the polyhedral norm. While Barvinok[2] and Woeginger[1] don't go further into the equivalence of the problems, we will provde an argument for completeness sake:

Proof: Given we have every permutation possible of the cities provided, we could evaluate every maximum tour under every permutation of cities to determine the permutation that yields the longest tour. Under a singular permutation the weight contributed to the tour by transitioning from c to c' under Tunnelling Max TSP is provided by equation 3 meaning there's at least one tunnel t that provides maximum weight to the tour. In terms of Max TSP under polyhedral norm defined by $\mathcal{H}$, since equation 3 and equation 1 are equal, the

maximum tunnel t would translate to the maximum halfspace vector h that defines the distance between c and c'. Therefore for every pair of cities that follow each other, the transitioning from one to the next contributes the same weight for both problems. Further implying that summing over every maximum distance of every neighbouring city we receive the same tour length for any permutation. Meaning that the problem boils down to finding a permutation that induces a maximum tour compared to every other permutation under any of the two problems. When the tour is found the weight of the tour will be an optimal value for both problems. □

This raises a questions - why do we even equate the solution of the Max TSP under polyhedral norm to the Tunnelling Max TSP, even though our interest lies on solving the Max TSP. While Woeginger[1] doesn't even define the Tunnelling TSP, Barvinok et al.[2] don't really argue why this step is necessary. One could say that the main advantage of continuing, by giving an algorithm that solves the Tunnelling Max TSP, is that the algorithm that will be presented, will attempt to find an optimal tour by choosing the correct tunnels. It could be seen as more logical to think of it as choosing tunnels instead of choosing halfspace vectors, where it seems counter intuitive to choose vectors, where the distance between cities is defined through a vector that maximizes an equation and is not chosen.

3.3 The Algorithm

We can now imagine that we have a bipartite graph $G = (V = \mathcal{P} \cup \mathcal{T}, E)$. On one side of the graph are the cities $\mathcal{P}, |\mathcal{P}| = n$ while on the other side are the Tunnels $\mathcal{T}, |\mathcal{T}| = k$. As previously mentioned that a Tunnel can be accessed by a city either at the front or back, meaning that for the edges of this graph we have from every city to every Tunnel exactly 4 edges, 2 edges to the front and back each. The reason being that a city could be reached for instance by exiting through the front of a tunnel and continuing the tour accessing the front of the same tunnel again, therefore making it necessary to have two edges that signify accessing the front to be able to represent every possible tour under the Tunnelling TSP. Therefore, using Barvinoks et al.[2] notation, the set of edges can be defined as $E = \{e_i[c, t, X] : i \in \{1, 2\}, c \in \mathcal{P}, t \in \mathcal{T}, X \in \{B, F\}\}$ and the weights defined, as previously mentioned, by $w(e_i[c, t, F]) = F(c, t)$, $w(e_i[c, t, B]) = B(c, t)$.

Further, Barvinok[2] provides us 3 conditions for a subset $E' \subset E$ to be a tour, where the length of the tour would be $\sum_{e \in E'} w(e)$.

($T1$) *Every city is incident to exactly two edges in E'*

($T2$) *For each tunnel t there's an equal number of edges of type F(x,t) and B(x,t)*

($T3$) *The set E' is connected*

The idea of the algorithm is to attempt to guess such an outline, and attempt to get the maximum solution for each outline, ultimately finding the maximum solution of all possible outlines. The important thing to consider is that a PTAS should have a runtime that is polynomial in input size n but can be exponential in ϵ or in this case in k.

3.4 Woegingers approach

Woeginger[1] approaches this problem by pointing out that there are n transitions between two cities and therefore also n transitions from one tunnel to the next. Guessing the outline by choosing a city order would result in $(n-1)!$ outlines and therefore would be exponential in n due to the Stirling Approximation. Therefore

instead we guess transitions between one tunnel to the next that accounts for k^2 possibilities. Between both tunnels we have 4 combinations of 'back' and 'font' tunnels. Leaving us with $4k^2$ possibilities for one transition and therefore $\mathcal{O}(n^{4k^2})$ outlines.

Given a guessed outline that consists of n tunnel transitions we need to check whether this outline is connected in $\mathcal{O}(n)$, as well as the number of back and front connections for every tunnel are correct in $\mathcal{O}(k)$. We now want to find a city to every tunnel transition. We can easily formulate this into either linear program or a maximum-weight bipartite matching problem, where in the latter case we have discussed the Hungarian Algorithm to solve such problem. Note that the Hungarian Algorithm maximizes therefore we have to find a maximum value of the entire matrix and adjust the edge weights accordingly - let's call this maximum value simply max. A transition from the X side of tunnel t to the X' side of tunnel t', where $X, X' \in \{F, B\}$ shall be connected to every city $c \in \mathcal{P}$, through the edge $e[t, X, t', X', c]$, with the weight of this edge being $w(e[t, X, t', X', c]) = max - (X(c, t) + X'(c, t'))$. Therefore V shall be the set of all tunnel transitions, while U is every city $c \in \mathcal{P}$. The edges are $E \subseteq V \times U$, where V and U are obviously disjunct. As we have n tunnel transitions and n cities this problem is solvable in $\mathcal{O}(n^3)$ through the Hungarian Algorithm described in section 1.1. Concluding that we have $\mathcal{O}(n^{4k^2})$ guessed outlines that can be solved by the Hungarian Algorithm, therefore the runtime of this approach to find a solution to the Tunnelling Max TSP and therefore the Max TSP under polyhedral norm with k facets is $\mathcal{O}(n^{4k^2+3})$.

Within Woegingers paper[1] it was clear that the idea was to simplify the Algorithm of Barvinok et al.[2] which promises a better runtime of $\mathcal{O}(n^{2k-2}log(n))$. Despite the complexity of that paper relative to me I decided to try to explain the results given by Barvinok et al. and therefore this subsection was kept rather short.

3.5 A Faster Solution

As we've previously noticed the largest contributing factor to the runtime is the number of guessed outlines. Barvinok et al.[2] gives us 5 characteristic of this outline that multiplied give us an upper bound the number of all outlines of this type.

1.Given a valid tour E' there's a set of tunnels that actually get visited within the outline $T' \subset T$ with $|T'| = p \leq k$. . Guessing the tunnel we use in the outline we have $2^k - 1$ options, since for every tunnel we have the choice whether it's part of the outline or not, but we don't allow $p = 0$. Further we want to also choose a connected subgraph that spans every tunnel in T' with $2p - 2$ edges, since to go from one tunnel to the next you have to use two edges, while two tunnels of the subgraph will only have one edge.

2. The set of edges used to span every Tunnel in T' shall be called $E'' \subset E'$. We can count different types of arrangements of spanning trees through Cayley's formula[14] where for p vertices there are p^{p-2} different type of spanning trees.

3.We now assign a city to every city in the spanning tree induced by E'' resulting in $\mathcal{O}(n^{p-1})$, since to span p tunnels we only need $p - 1$ cities.

4.Considering that every edge of E'' can be one of 2 front or 2 back types there's an estimate of $\mathcal{O}(4^{2p-2})$ possible arrangements.

5.Considering that (T2) applies we know that for every tunnel we use, there should be an equal number of front and back edges that are used in the tour. One more thing that we guess about an outline is the number of visits of a tunnel giving us a set $\mathcal{D} = \{d_1, d_2, \ldots, d_p\}$, the sum of which is $\sum d_i = n$. Barvinok et al. uses a trick here where d_p shall remain undefined as well as d_{p-1} being linearly dependant on the choice of d_p. Meaning the guessed outline should have $d_1, d_2, \ldots, d_{p-2}$ defined but to actually solve the outline we will set the d_p

afterwards. For the $p-2$ variables we have $\mathcal{O}(n^{p-2})$ possibilities.

To find out how many of these guessed outlines we have, we simply multiply the 5 characteristics listed above assuming that worse case $p = k$.

$$2^k \cdot k^{k-2} \cdot n^{k-1} \cdot 4^{2k-2} \cdot n^{k-2} \in \mathcal{O}(n^{2k-3})$$

Ultimately we get a guessed outline with a set of tunnels in use $T' = \{t_1, \ldots, t_p\}$, as well as $2(p-1)$ edges and $p-1$ cities used to span the tunnels. Additionally the number of usages of each tunnel up to t_{p-2} is defined. The next step is to assume that all of these variables are given and we want to construct an optimal maximum tour that has these preset characteristics. Mainly, the variables that are open are the edges that connect the rest of the cities to the tunnels s.t. we full fill (T1)-(T3)

As per characteristic 5 Barvinok et al. describes that d_p will be left unset and as well as d_{p-1} that will linearly depend on d_p since we know the entire sum of $\sum d_i$ to be equal to n. This means while we can assume that the variables stated above are given within this guessed outline we can choose a $d_p \in \{1, 2, \ldots, n\}$. This means that within one guessed outline we have at most $\mathcal{O}(n)$ sub-problems, the maximum of which we want to find.

Barvinok et al.[2] do explain the 5 characteristics as above however they gives us immediately the problem formulated in a linear optimisation problem. In this thesis we would like to go more in-depth and understand what the decision variables mean as well as the constraints.

Firstly the decisions variables are $x^B_{i,j}$ and $x^F_{i,j}$ with $1 \leq i \leq p$ and $p \leq j \leq n$. The setting of these decision variables mean whether the front edge or back edge between tunnel i and city j is part of the locally optimal tour E'. Therefore our first two constraints are of course the non-negativity constraint, since we can either choose to have 0,1 or 2 tunnels of this type.

$$\begin{aligned} x^B_{i,j} &\geq 0 \quad , i = 1, \ldots, p, \ j = p, \ldots, n \\ x^F_{i,j} &\geq 0 \quad , i = 1, \ldots, p, \ j = p, \ldots, n \end{aligned}$$

Due to (T1) each city can be only incident to two edges meaning we get the next set of constraints. Note that this is not a singular constraint but $n-p+1$ constraints. Since for the first $p-1$ cities the edges are already set as per the guessed outline, we want to make sure for the rest of the cities the amount of edges incident to them is at exactly two.

$$\sum_{i=1}^{p} (x^B_{i,j} + x^F_{i,j}) = 2, \quad j = p, \ldots, n$$

We shall define another two minor variables f^F_i and f^B_i with $i \in \{1, 2, \ldots, p\}$, that stand for the the number of front and back edges of tunnel t_i that are already in the guessed outline E''. We can define them as such:

$$\begin{aligned} f^F_i &= |\{e(c, t, X) \in E'' : X = F, t = t_i, c \in \mathcal{P}\}| \\ f^B_i &= |\{e(c, t, X) \in E'' : X = B, t = t_i, c \in \mathcal{P}\}| \end{aligned}$$

And therefore our last two sets of constraints are:

$$\sum_{j=p}^{n} x_{i,j}^{B} = d_i - f_i^{B}, \quad i = 1, \ldots, p$$
$$\sum_{j=p}^{n} x_{i,j}^{F} = d_i - f_i^{F}, \quad i = 1, \ldots, p$$

These set of constraints can be explained by understanding both side separately as well as paying attention to the fact that we sum over j while for every i there is a separate constraint. This means, given an arbitrary i within the range of 1 to p, considering the first set of constraint where we look at the back tunnels, we subtract the number of back tunnels already in the guessed outline f_i^B from the number of tunnel visits set by the guessed outline d_i. This should be equal to the sum over the decision variables that dictate whether a back edge from any city, that is not already in the guess outline, $c_j, j = p, \ldots, n$ to this one arbitrary tunnel t_i. In simple terms we can draw only as many back edges from tunnel i to any city j that is dictate by the preset visits minus the already used edges.

It should be emphasized that the number d_i is not to be confused with number of incident edges to t_i but is instead the number of visits to that tunnel, meaning that there are exactly d_i front edges as well as d_i back edges incident to t_i as we can derive from (T2).

Additionally, there's a simplification we make here compared to Barvinok et al. where, as we previously discussed, d_p is a variable that we choose instead of being given and d_{p-1} being linearly dependant on the choice of d_p. Therefore Barvinok et al. choose to give two separate constraints for $i = p - 1$. However we can simply argue that the constraints mentioned above include the case $i = p - 1$, simply by calculating d_{p-1} beforehand:

$$d_{p-1} = n - d_p - \sum_{i=1}^{p-2} d_i$$

Finally we can construct the objective function that we want to maximize.

$$g(d_p) = max(\sum_{i=1}^{p} \sum_{j=p}^{n} (B(c_j, t_i) x_{i,j}^{B} + F(c_j, t_i) x_{i,j}^{F}))$$

$g(d_p)$ can be simply looked as a function dependant on d_p where setting a d_p will give us the complete optimisation problem. We sum over every decision variable $x_{i,j}^B, x_{i,j}^F$ multiplied by their respective weights. Meaning if for instance $x_{i,j}^B = 1$ by solving a problem, we can interpret this decision as the optimal solution of a given guessed outline being one where we choose exactly one edge $e(c_j, t_i, B)$, further implying that the weight contributed to the objective function would be $w(e(c_j, t_i, B)) \cdot x_{i,j}^B = B(c_j, t_i) \cdot x_{i,j}^B$.

While this can be formulated as a as a maximum-weight bipartite matching with $n - p + 1$ remaining cities as sources and $2p$ destinations being the front and back access to every tunnel in use, it can not be easily solved by the Hungarian Algorithm. Solving this as a linear program we of course have the option of the Simplex algorithm but it is unclear how it will behave and under what runtime. Barvinok et al.[2] seem to argue that as the number of destinations is in $\mathcal{O}(p)$ and we can see p as a fixed parameter $p \leq k$ we can use this structure to solve such a linear program for every $\mathcal{O}(n)$ cases of d_p in $\mathcal{O}(n)$ time. Another argument Barvinok et al. uses that going over every case of d_p we can instead of going linearly through it, it's possible to do a binary search over the optimal solution for the values of d_p. Sadly the full explanation to optimally solve this linear program is out of the scope of this thesis, however this seems to be a rather small factor as the biggest factor contributing

to the runtime is the enumeration of guessed outlines.

Concluding that with $\mathcal{O}(n^{2k-3})$ cases we go through at most n values with the help of binary search and solving each linear program in $\mathcal{O}(n)$, we arrive at the final runtime of $\mathcal{O}(n^{2k-2}log(n))$.

4 Comparing Tunnelling Algorithm to GPH

Now that we have a good understanding of how the GPH as well as the Tunnelling Algorithm function as well as having an understanding of their runtime and approximation ratio, we can try to understand how they compare.

Recall the two classes of approximation algorithms in the introduction. Given that an algorithm has a multiplicative approximation ratio $\rho(n)$ we classify the algorithm as APX if the factor is constant, and if we can make a stronger assertion that the approximation ratio and the runtime depend on a factor ϵ where with a larger factor the solution improves but the runtime degrades, we classify such an algorithm as Polynomial-Time Approximation Scheme or PTAS. Further, the interesting thing about both the algorithms presented is that the relationship of the algorithms to their respective class is not obvious just looking at the results.

The main contribution of this thesis will be to compare the algorithms in a more practical scenario. We notice that certain parts of both algorithms are defined in a very broad way, where doubling dimension is a generalization for higher dimensional spaces as well as the Tunnelling Algorithm being dependant on the dimension d and the choice of polytope to represent the polyhedral norm. To simplify the comparison we will first make the decision to compare the algorithms within the plane $\mathbb{R}^d$ where $d = 2$. As stated previously the goal is to see how the algorithms approximate the Max TSP under Euclidean norm.

4.1 GPH

In the case of the GPH we have found two upper bounds for the approximation ratio.

$$\rho(n)_{GPH} \leq e^{1/3} \approx 1.3956$$
$$\rho(n)_{GPH} \leq (1 - \frac{7/3}{n^{1/(2dim+1)}})^{-1}$$

As mentioned previously, it is well known that within 2 dimensional Euclidean space we can cover any ball with 7 balls of half it's radius as seen in figure 1, meaning we can derive the doubling dimension by calculating $7 = 2^{dim}$ and therefore $dim = log_2(7) \approx 2.8$. Therefore the approximation ration of the GPH in $\mathbb{R}^2$ has an upper bound of:

$$\rho(n)_{GPH} \leq (1 - \frac{7/3}{n^{1/6.6}})^{-1}$$

A key observation is that the approximation ratio is dependant on the input size n, which is quite unusual. Since the APX class requires the approximation ratio to be a constant, the GPH is not strictly in APX, however given the limit of the approximation ratio we know that with higher input size the ratio improves since it moves closer to 1, therefore the upper bound of $\rho(n)_{GPH} \leq e^{1/3}$ always applies as that the approximation ratio seems to be better then a strictly constant approximation ratio.

We can combine both the upper bounds for the approximation ratio, as Shenmaier[3] asserts that for $n^{1/(2dim+1)} < 8$ the relative error becomes $\delta \leq 1 - e^{-1/3}$ which is a approximation ratio of $\rho(n)_{GPH} \leq e^{1/3}$.

This assertion is however not a minor one as:

$$\begin{aligned} n^{1/(2dim+1)} &< 8 \\ n^{1/(2\cdot 2.8+1)} &< 8 \\ n &< 8^{6.6} \approx 9.128 \times 10^5 \end{aligned}$$

Meaning for an input size $n < 9.128 \times 10^5$ we can assume an upper bound for the approximation ratio of $\rho(n) \leq e^{1/3}$, while for any $n \geq 9.128$ the approximation ratio seems to improve relative to input size.

4.2 Tunnelling Algorithm

The case for the Tunnelling Algorithm is however a little more difficult. Since a polytope is a broad definition of a geometrical shape with $f = 2k$ facets, we can simplify to having a choice of having a polyhedral norm with a polygon, a two dimensional polytope, as a unit ball. As we know, the runtime suggested by Barvinok et al.[2] is $\mathcal{O}(n^{2k-2}log(n))$. However the approximation ratio is derived by choice of polytope, or in this case polygon.

Given we made such a choice we can calculate an approximation ratio but there is no clear formula that ties k directly to the approximation ratio. While providing a more broader equation that accounts for higher dimensional polytopes might be out of reach for this thesis, we can make some calculation that will apply in the two dimensional space.

Firstly let's understand what the crucial observation 1 requires and what approximation ratio it provides, given the prerequisites. Assuming we want to approximate the Euclidean unit ball, a circle in the plane, with a polygon. For the figures below(see figure 5) I will provide an example where the polygon is a simple square. The most optimal way to construct this polygon would be to have each side be equal in length as well as having each inner angle be equal, therefore being a regular polygon. Note that as previously mentioned we only allow for polygons with a number of facets divisible by 2, as a consequence of having it to be point symmetrical to classify the polyhedral norm as a norm.

Given we have a unit circle $\mathcal{B}_1$ with the radius of 1 unit we want to find a polygon $\mathcal{B}_2$ that will approximate the circle, where $(1-\epsilon)\mathcal{B}_2 \subseteq \mathcal{B}_1 l \subseteq (1+\epsilon)\mathcal{B}_2$. Let's define $(1-\epsilon)\mathcal{B}_2 = \mathcal{B}_l$ and $(1+\epsilon)\mathcal{B}_2 = \mathcal{B}_g$, as in g for greater and l for lesser. We can imagine both $\mathcal{B}_l$ and $\mathcal{B}_g$ as shrunken and expanded versions of $\mathcal{B}_2$ respectively, meaning that the inner angles do not change as well as the number or placement of faces.

We can start by constructing $\mathcal{B}_g$ and $\mathcal{B}_l$ where for $\mathcal{B}_g$ the best polygon we can construct will be one where each side of the polygon is a tangent to $\mathcal{B}_1$. Drawing a line s_g between the distance of the centre and an arbitrary tangent point, we know that the length of s_g will be $s_g = 1$, as it is equal to the radius of $\mathcal{B}_1$.

To determine the growth and shrink factor and therefore ϵ and $\rho(n)$ we can simply determine the same line s_l from centre to a tangent point for $\mathcal{B}_l$. In case of $\mathcal{B}_l$, we construct it so the circle passes through all the vertices of the polygon. We know that s_l is perpendicular to the side of the polygon and that the distance from the centre to edge is 1 according to the radius of the circle that passes through it. Additionally it is well know that the the interior angle of a regular polygon with f facets is:

$$\alpha = (f-2)\cdot 180^\circ / f = 180^\circ(2k-2)/2k = 180^\circ(k-1)/k$$

We think of the triangle that is created by a side of the polygon and two lines from the centre to each of

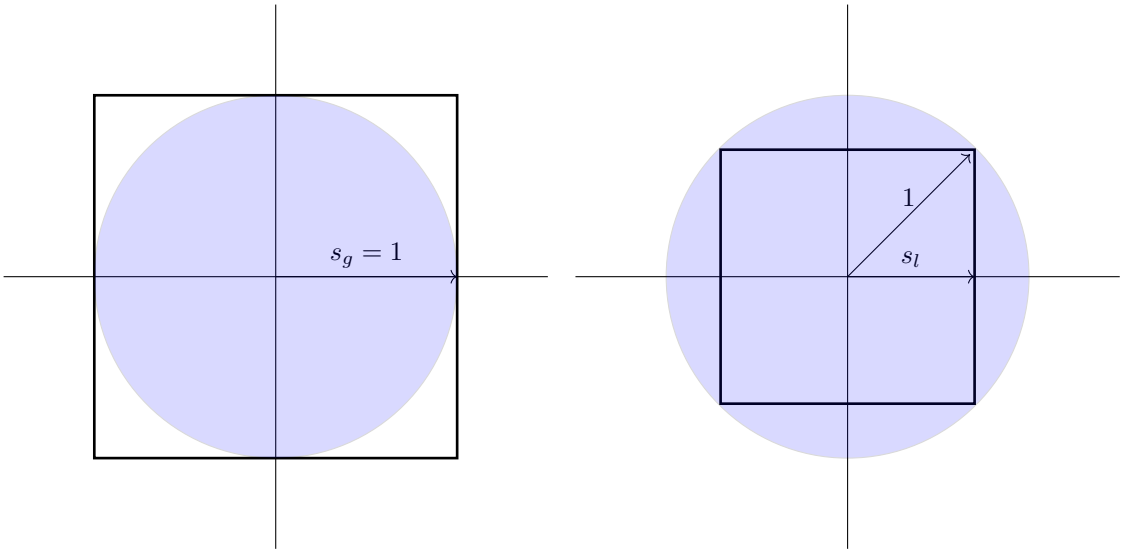

Figure 5: Left: $\mathcal{B}_g$; Right: $\mathcal{B}_l$; Blue circle represents the unit ball of the Euclidean norm

the ends of the side of the polygon as pictures in figure 6. (These calculations work on a broad spectrum of polygons and therefore the figure is only representative, in reality the angles would obviously differ based on the number of faces, as well as the faces to the side of the triangle are only representative to how the other sides would connect.)

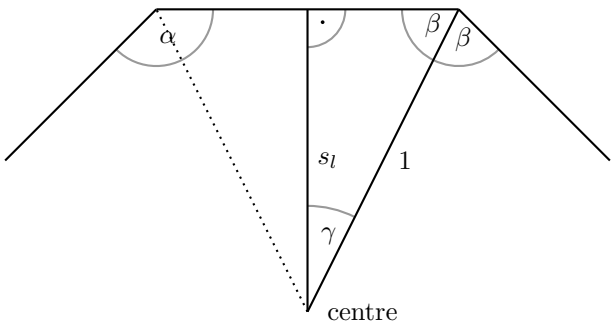

Figure 6: Triangle within the polygon

Within the triangle the angle is exactly halved therefore:

$$\beta = \alpha/2 = 90°(k-1)/k$$

And the last angle of the triangle is:

$$\gamma = 180° - 90° - \beta = 90° - 90°(k-1)/k = 90°/k$$

Therefore we can easily find that:

$$cos(\gamma) = \frac{s_l}{1}$$
$$s_l = cos(\frac{90^\circ}{k})$$

Let's call the line that starts from the centre and meets the middle of a side of the polygon $\mathcal{B}_2$ the original s. As we now know the length of the upscaled s_g as well as the down scaled s_l we can now derive the approximation ratio since we know from the crucial observation 1 that given that the Ball $\mathcal{B}_2$ can be upscaled by $(1+\epsilon)$ to include $\mathcal{B}_1$ and downscaled with $(1-\epsilon)$ to be inscribed in $\mathcal{B}_1$ the resulting approximation algorithm will have an approximation ratio $\rho'(n)$ of worst case $\frac{(1-\epsilon)}{(1+\epsilon)}$.

Note that we call this approximation ratio $\rho'(n)$, instead of $\rho(n)$, as we will later find out that Woegingers[1] definition assumes that the approximation ratio is what percentage the approximated value is to the optimal value. This may be seen as more intuitive, however we choose to follow a different definition. Furthermore worst case approximation ratio will be interpreted as $\rho'(n) \geq \textit{worst-case}$ or $\rho(n) \leq 1/(\textit{worst-case})$.

$$(1-\epsilon)s = s_l$$
$$(1+\epsilon)s = s_g$$

$$(1-\epsilon)s = cos(\frac{90^\circ}{k})$$
$$(1+\epsilon)s = 1$$

$$\Rightarrow s = \frac{1}{1+\epsilon}$$

$$(1-\epsilon)s = cos(\frac{90^\circ}{k})$$
$$\Rightarrow (1-\epsilon)\frac{1}{1+\epsilon} = cos(\frac{90^\circ}{k})$$
$$\Rightarrow \rho'(n)_T \geq cos(\frac{90^\circ}{k}); \quad since\ \rho'(n) \geq (1-\epsilon)/(1+\epsilon)$$

Since our goal was finding out the approximation ratio in the first place, there is no interest in determining the exact size of s, we can however imagine that it will be somewhere in between the up and downscaled versions of it.

Therefore we have established an approximation ratio. Looking at the function however, we notice that looking at values $k \geq 2$ the function approaches 1, as expected, however it approaches it from the bottom, justifying our differentiation between $\rho(n)$ and $\rho'(n)$.

We can imagine that the more facets we use to approximate the Euclidean unit ball the closer our algorithm comes to the optimal solution which is confirmed by calculating the limits to the approximation ratio where:

$$lim_{k\to\infty} 1/cos(90^\circ/k) = 1/cos(0) = 1$$

Looking at the graph of this function in figure 7 we can see that, as the limits predict, we start approaching 1 very quickly. I chose to omit results for values $k < 0.5$ due to the function being difficult to portray, but additionally, since the minimum amount of facets a polygon that is also point symmetrical can have is $f = 2k = 4$, the evaluation of the graph for $k < 2$ is not relevant. We can imagine that approaching 0 the graph starts to oscillate at an increasingly faster pace and the limit will become undefined as is common for limits of trigonometric functions.

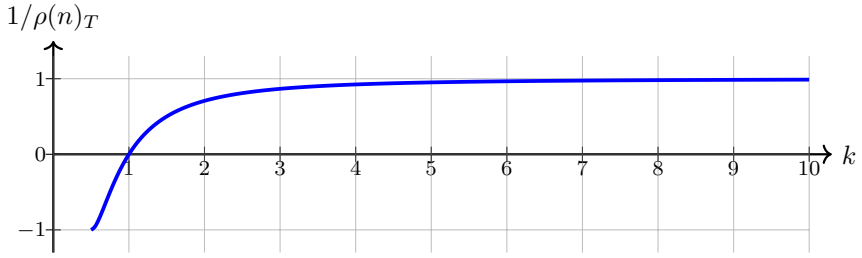

Figure 7: Note that the y-Axis is $\rho'(n) = 1/\rho(n)$.

Concluding, the Tunnelling Algorithm has a runtime of $\mathcal{O}(n^{2k-2}log(n))$, where in the plane we can chose a polygon with $f = 2k$ facets s.t. the approximation ratio has an upper bound of $\rho(n)_T \leq 1/cos(\frac{90^\circ}{k})$

4.3 Comparison for 2-Dimensional Space

Now that we have some more concrete equations to work with and a more exact upper bounds for the approximation ratios for both algorithms we can make a more direct comparison. It's crucial to notice that for both algorithm we have provided worse case approximation ratios, where it's also unclear how tight these approximation worse cases are.

Comparing the runtime, the smallest k we can choose for the Tunnelling Algorithm in the plane would be $k = 2$ which is also convenient, since this gives us a runtime of $\mathcal{O}(n^{2\cdot 2-2}log(n)) = \mathcal{O}(n^2log(n))$ meaning that it is just slightly better than the GPH runtime of $\mathcal{O}(n^3)$. Meaning the resulting approximation ratios would be:

$$\rho(n)_T \leq 1/cos(90^\circ/2) \approx 1.412$$
$$\rho(n)_{GPH} \leq 1/(1 - \frac{7}{3n^{1/6.6}})$$
$$\rho(n)_{GPH} \leq e^{1/3} \approx 1.3956$$

As we can see the constant upper bound of the approximation ratio of the GPH is already better then the upper bound for the Tunnelling Algorithm. Doing a comparison between the Tunnelling Algorithms upper bound and the upper bound of the GPH dependant on the input size n we can calculate at what input size $n \geq n'$ the upper bound of the GPH becomes better:

$$
\begin{aligned}
&\rho(n)_T = 1/cos(90°/2) \approx 1.412 \\
&\rho(n)_{GPH} = 1/(1 - \frac{7}{3n^{1/6.6}}) \\
\Rightarrow &1/cos(90°/2) = 1/(1 - \frac{7}{3n'^{1/6.6}}) \\
\Rightarrow &cos(90°/2) = 1 - \frac{7}{3n'^{1/6.6}} \\
\Rightarrow &1 - cos(90°/2) = \frac{7}{3n'^{1/6.6}} \\
\Rightarrow &n'^{1/6.6} = \frac{7}{3 - 3cos(90°/2)} \\
\Rightarrow &n' = (\frac{7}{3 - 3cos(90°/2)})^{6.6} \\
\Rightarrow &n' \approx 8.879 \times 10^5
\end{aligned}
$$

As we have discussed previously in section 4.1 for $n \leq 9.128 \times 10^5$ the constant upper bound is a better upper bound for the GPH. Despite the GPH upper bound being smaller then the upper bound of the Tunnelling Algorithm it is important to say that the Tunnelling Algorithm delivers a slightly better runtime of $\mathcal{O}(n^2 log(n))$ compared to $\mathcal{O}(n^3)$ and the upper bound being very close as we can interpret that the optimal solution is within a factor of roughly 1.4 to the approximated solution, where only for values $n \geq 9.128 \times 10^5$ the upper bound of the GPH starts to improve whereas the the upper bound of the Tunnelling Algorithm stays put.

Obviously there is still an option to just increase the number of facets for the polyhedral norm, therefore achieving a better ratio and therefore requiring the GPH to have an even higher input size to match the approximation ratio.

We can start by doing an example where we calculate n' in the case of $k = 3$:

$$
\begin{aligned}
n' &= (\frac{7}{3 - 3cos(90°/3)})^{6.6} \\
\Rightarrow n' &\approx 1.5499 \times 10^8
\end{aligned}
$$

While it seems like the n' has increased by a factor of roughly 10^2, one could falsely conclude that the relationship between n' and k is an exponential one, however that is not true. To examine the relationship between these two, we first imagine the following function, the growth rate of which we want to figure out:

$$
n' = f(k) = (\frac{7}{3 - 3cos(90°/k)})^{6.6} \quad for\ k \geq 2
$$

Firstly we need to convert the cosine of degree to radiant cosine meaning $cos_d(90°/x) = cos_r(\frac{90°}{k}\frac{\pi}{180°})$, leaving us with

$$
f(k) = (\frac{7}{3 - 3cos(\pi/2k)})^{6.6} \quad for\ k \geq 2
$$

This gives us the opportunity to approximate the radiant cosine with the help of Taylor Series, where it is well known that we can approximate cosine with:

$$cos_r(x) = 1 - x^2/2! + x^4/4! - x^6/6! + \dots$$

Since, with a growing k the term within the cosine becomes smaller and is generally $\pi/2k \leq 1$ for $k \geq 2$, we can imagine that the $1 - x^2/2!$ dominates the term and gives us a reasonably accurate approximation to determine the growth of the function, leaving us with:

$$\begin{aligned}
f(k) &\in \mathcal{O}(\frac{7}{3 - 3(1 - (\pi/2k)^2/2)})^{6.6} \\
&= (\frac{7}{3 - 3 + (\pi/2k)^2/2})^{6.6} \\
&= (\frac{7}{(\pi/2k)^2/2})^{6.6} \\
&= (\frac{7}{\pi^2/2k^2})^{6.6} \\
&= (\frac{14k^2}{\pi^2})^{6.6} \\
&= \frac{14^{6.6}k^{13.2}}{\pi^{13.2}} \\
&= \frac{14^{6.6}}{\pi^{13.2}}k^{13.2} \in \mathcal{O}(k^{14}) \\
\Rightarrow n' &\in \mathcal{O}(k^{14})
\end{aligned}$$

Since the runtime of the Tunnelling Algorithm is $\mathcal{O}(n^{2k-2}log(n))$ and the runtime of the GPH is $\mathcal{O}(n^3)$ independent of a k. We can see that with a larger k the runtime of the Tunnelling Algorithm grows exponentially while the n' which is the input size for which the GPH with an input of $n \geq n'$ gets a better worst case approximation ratio only grows polynomially.

Concluding this section, we can say that for $k = 2$ the Tunnelling Algorithm and GPH have a similar approximation ratio for a small n, while the Tunnelling Algorithm has a slightly better runtime. Choosing a larger number of facets k the growth of the Tunnelling Algorithms runtime is worse compared to the growth of n' relative to k for which the GPH starts to out perform the Tunnelling Algorithm.

4.4 Higher Dimensional Scaling

While, as discussed in section 2.1, it is well known that the relationship between doubling dimension and the d dimensional Euclidean space is $dim \in \theta(d)$, generalizing the Tunnelling Algorithm for higher dimension poses some difficulties. For the case of the GPH it's an obvious observation that a higher doubling dimension causes the approximation ratio to approach 1 slower than before. For simplicity we can just make the calculation with the relative error.

$$lim_{n\to\infty}\frac{\frac{7}{3}n^{-1/2(dim+1)+1}}{\frac{7}{3}n^{-1/2(dim)+1}}$$
$$lim_{n\to\infty}\frac{n^{-1/2(dim+1)+1}}{n^{-1/2(dim)+1}}$$
$$lim_{n\to\infty}n^{(-1/2dim+3)+(1/2dim+1)} = 0 \quad for\ dim > 0$$

As for the Tunnelling Algorithm it becomes more difficult to give an strong assertion. As usual the Tunnelling Algorithm suffers from a curse of dimensionality as a d dimensional Euclidean Space $\mathbb{R}^d$ we will need at least $k = d$ halfspace vectors to complete a polytope of d dimension. Meaning that the runtime of the Tunnelling Algorithm increases exponentially relative to the dimension d.

The same calculation for α as in the 2-dimensional calculation can be made in 3 dimensions with a regular polyhedron, the only difference being that since the calculation is made on two dimensional space and the third dimension does not impact the angles, the angles that we're calculating stays the same since we can look at a cube from one side and calculating the angles as we did in two dimensions. Therefore the approximation ratio stays $\rho(n) = 1/cos(90°/2) = 1/cos(45°) \approx 1.412$.

Without proof, we can make an educated guess that given a d dimensional euclidean space we can choose $k = d$ as the minimal number of halfspace vectors. Furthermore we can choose $\mathcal{H}$ as the canonical basis vectors such that the approximation ratio will stay at $\rho(n)_T = 1/cos(45°) \approx 1.4142$. Additionally it may be that this upper bound of the approximation ratio holds true as long as the polytope induced by $\mathcal{H}$ is regular.

However since the dimensionality impacts the Tunnelling Algorithm by increasing it's runtime while in the case of GPH the dimension mainly impacts the approximation ratio it becomes difficult to make a direct comparison.

References

[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 .

[4] V.V. Shenmaier(2021). Efficient PTAS for the maximum traveling salesman problem in a metric space of fixed doubling dimension. Optimization Letters, Volume 16, Issue 7, Springer Science and Business Media LLC. ISSN:1862-4480. DOI: 10.1007/s11590-021-01769-2. arXiv:2012.08379

[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein(2003). Introduction To Algorithms Third Edition. ISBN-13 9780262533058. MIT Press 9780262533058.

[6] C.H. Papadimitriou(1977). The Euclidean travelling salesman problem is NP-complete. Theoretical Computer Science, Volume 4, Issue 3, 237-244. ISSN 0304-3975. DOI: 10.1016/0304-3975(77)90012-3. Sciencedirect: 0304397577900123.

[7] R.E. Burkard, M. Dell'Amico, S. Martello(2009). Assignment Problems - Revised Reprint. Published by Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104). ISBN 978-1-611972-22-1 (revised reprint).

[8] F. Glover, G. Gutin, A. Yeo, A. Zverovich(2001). Construction heuristics for the asymmetric TSP. European Journal of Operational Research, Volume 129, Issue 3, Pages 555-568. ISSN 0377-2217. DOI: 10.1016/S0377-2217(99)00468-3. Sciencedirect: S0377221799004683.

[9] R.M. Karp(1976). The Probabilistic Analysis of some Combinational Search Algorithms. EECS Department, University of California, Berkley. Technical Report No. UCB/ERL M581.

[10] R.M. Karp, J.M. Steele(1985). Probabilistic analysis of heuristics. E.L. Lawler et al. (Eds.), The Traveling Salesman Problem, Wiley, New York, pp. 181-205.

[11] H.N. Gabow(1983). An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. Association for Computing Machinery, Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, pages 448–456, STOC 1983. ISBN: 0897910990. DOI: 10.1145/800061.808776. ACM: 10.1145/800061.808776

[12] A. Gupta, R. Krauthgamer, and J. R. Lee(2003). Bounded geometries, fractals, and low-distortion embeddings. 44th Annual IEEE Symposium on Foundations of Computer Science. Cambridge, MA, USA, 2003, pp. 534-543. doi: 10.1109/SFCS.2003.1238226.
IEEE Symposium on Foundations of Computer Science, 534–543

[13] Doubling space.(2025, September 8) In Wikipedia

[14] A. Cayley. "A theorem on trees". Quarterly Journal on Mathematics, Volume 23, 376-378.

[15] James Munkres(1957).Algorithms for the Assignment and Transportation Problems. Journal of the Society for Industrial and Applied Mathematics 5, no. 1, 32-38. JSTOR.