Freie Universität Berlin

Master's Thesis

Combinatorics of Beacon-based Routing and Guarding in Three Dimensions

Jonas Cleve
jonas.cleve@fu-berlin.de

Submitted: March 23, 2017
Revised: March 31, 2017

Advisor:
Prof. Dr. Wolfgang Mulzer

Second Advisor:
Dr. Frank Hoffmann

Institut für Informatik
Freie Universität Berlin

Abstract

A beacon is a point-like object which can be enabled to exert a magnetic pull on other point-like objects in space. Those objects then move towards the beacon in a greedy fashion until they are either stuck at an obstacle or reach the beacon's location. Beacons placed inside three-dimensional polytopes can be used to route point-like objects from one location to another. A second use case is to cover a polytope such that every point-like object at an arbitrary location in the polytope can reach at least one of the beacons once the latter is activated.

The topic of beacon-based routing and guarding was introduced by Biro et al. [4] in 2011 and covered in detail by Biro in his PhD thesis [3]. Therein various aspects of beacons in the polygonal domain of two dimensions were studied.

In this thesis we provide the first results for beacon-based routing and guarding for three dimensions. We first define the setting for three dimensions and look at two-dimensional beacon-based routing which lays the groundwork for our three-dimensional approach. We then have a look at the complexity of certain three-dimensional routing and guarding problems for which a smallest set of beacons should be obtained. We show that some of the problems are at least as hard as their two-dimensional counterpart which makes them NP-hard and APX-hard.

For the problem of finding a smallest set of beacons to be able to route between any pair of points in a polytope we show that it is always sufficient and sometimes necessary to place $\lfloor (m+1)/3 \rfloor$ beacons, where m is the number of tetrahedra in a tetrahedral decomposition of the polytope.

Finally, we show that there exists a polytope which cannot be covered by placing a beacon at each vertex of the polytope.

Statement of authorship

I hereby declare that this thesis has been composed by myself and describes my own work unless otherwise acknowledged in the text. All verbatim quotations are marked as such and all sources of information have been specifically acknowledged. This thesis was neither submitted in any other application for a degree nor published in any form.

Berlin, March 23, 2017

Jonas Cleve

Acknowledgments

I was supported directly and indirectly by many people during the time I worked on this thesis.

First of all, I want to thank my advisor, Wolfgang Mulzer, for the topic suggestion and the supervision. He always took the time to answer my questions or to give me advice.

Furthermore, I would like to thank Nadja who not only shares an office with me but who also listened to my thoughts and problems and often helped me find solutions. I also want to thank Max for his suggestions on the structure of this work which improved it a lot.

Special thanks go to Franziska for her absolute support, especially whenever I seemed to have too much work in front of me to even begin—this means a lot to me.

Finally, I would like to thank my family which has always supported me in the decisions I have taken and which will continue to do so.

Contents

Chapter 1

Introduction and related work

1.1 Introduction

The topic of beacon-based routing and guarding was introduced by Biro et al. [4] in 2011 and covered in detail by Biro in his PhD thesis [3]. He looked at routing and guarding in (two-dimensional) polygonal domains. A beacon b is a point-like object that can be enabled and disabled and which, when enabled, exerts a *magnetic pull* on other point-like objects in space. Such an object p is then forced to move in a fashion which greedily minimizes its distance to b. If p reaches an obstacle which blocks the direct path from p to b, it will slide along the obstacle, still trying to greedily minimize its distance to b.

In general, two things can happen: Either p reaches b or it gets stuck, which means that it touches an obstacle and there is no movement along the obstacle which decreases the distance to b.

To *route* a point-like object p towards a location t *via beacons*, an implicit beacon at t is considered. Then the first beacon is enabled to attract p until it reaches the beacon's location. Subsequently the first beacon is disabled and the next one switched on. This procedure is repeated until the last (implicit) beacon at t is enabled and finally attracts p to its location. Note that in our setting every beacon must attract p until it reaches the beacon's location. Only then are we allowed to enable the next beacon. Thus, if p gets stuck it will be stuck there forever which is not a valid solution.

A set of beacons is said to *cover* some domain if and only if for every point p in the domain there is a beacon such that when the latter is enabled a point-like object at p reaches the beacon's location.

The main interest of this thesis lies in the combinatorics of the routing problem in three dimensions. We take the two-dimensional polygonal domain of Biro [3] and transfer it to the three-dimensional domain with polytopes. We are then mainly interested in the number of beacons needed to be able to route between any pair of points inside a polytope.

At first, in Chapter 2, we define the geometric and beacon-related terms for the remainder of the thesis. In Chapter 3, we look at the two-dimensional case for beacon-based routing and revise a part of the proof by Biro et al. [5]. We start with the three-dimensional domain in Chapter 4 where we show that (like in the two-dimensional case) some of the problems in beacon-based routing and coverage are NP-hard and APX-hard. The two-dimensional results from Chapter 3 are then applied to three dimensions in Chapter 5 where we prove a sharp bound for the number of beacons necessary to route within polytopes that have a tetrahedral decomposition. Finally, in Chapter 6 we show that for general polytopes it does not suffice to place a beacon at every vertex of the polytope to attract all points.

1.2 Related work

Two dimensions. The topic of beacon-based routing and coverage in two-dimensional polygons has been studied starting in 2011 especially by Biro in various publications [3, 4, 6] and very broadly in his PhD thesis [5]. In the latter he studied *attraction regions* (all points attracted by a beacon), *inverse attraction regions* (all points that can attract a specific point), *beacon kernels* (all points that attract all points inside a polygon), *beacon routing* (minimal number of beacons to route between any pair of points), and *beacon guarding* (minimal number of beacons to be able to attract all points).

In his work, he proved tight bounds of $\lfloor n/2 \rfloor - 1$ on the number of beacons necessary for routing in simple polygons with n vertices. To route within a polygon with h holes $\lfloor n/2 \rfloor - h - 1$ beacons are sometimes necessary and $\lfloor n/2 \rfloor + h - 1$ beacons are always sufficient. For routing in orthogonal polygons (all edges are parallel to the x- or y-axis) he can only show a smaller lower bound of $\lfloor n/4 \rfloor - 1$ beacons, resulting in loose bounds.

For beacon-based coverage of simple polygons and polygons with h holes he shows that $\lfloor 4n/13 \rfloor$ beacons are sometimes necessary, while $\lfloor (n+h)/3 \rfloor$ beacons are always sufficient to cover a polygon. For simple polygons $h = 0$ and thus the upper bound is $\lfloor n/3 \rfloor$. For the coverage of orthogonal polygons he shows an upper bound of $\lfloor n/4 \rfloor$ and a lower bound of $\lfloor (n+4)/8 \rfloor$.

In all of the previously mentioned cases with loose bounds, he conjectured that the lower bound is indeed also the upper bound.

Bae et al. [1] enhanced the former results by showing that $\lfloor n/6 \rfloor$ beacons are sometimes needed and always sufficient for beacon-based coverage in orthogonal polygons. They furthermore improved the bounds for routing in orthogonal polygons: The lower bound was improved slightly to $\lceil n/4 \rceil - 1$ while the upper bound could be reduced to $\lfloor (3n-4)/8 \rfloor - 1$.

Finally, Shermer [17] found that $\lfloor (n-4)/3 \rfloor$ beacons are always sufficient and sometimes necessary to route between any pair of points in an orthogonal polygon. An overview over all the best known results for routing and coverage in two dimensions can be found in Table 1.1.

Art gallery problems. The problem of beacon-based coverage is related to art gallery problems. It was introduced in 1973 by Victor Klee when he posed the question how

		Bound		
		Lower	Upper	Shown by
Routing	Simple polygons	$\lfloor \frac{n}{2} \rfloor - 1$	$\lfloor \frac{n}{2} \rfloor - 1$	Biro et al. [4]
	Polygons with holes	$\lfloor \frac{n}{2} \rfloor - h - 1$	$\lfloor \frac{n}{2} \rfloor + h - 1$	Biro [3]
	Orthogonal polygons	$\lfloor \frac{n-4}{3} \rfloor$	$\lfloor \frac{n-4}{3} \rfloor$	Shermer [17]
Coverage	Simple polygons	$\lfloor \frac{4n}{13} \rfloor$	$\lfloor \frac{n}{3} \rfloor$	Biro [3]
	Polygons with holes	$\lfloor \frac{4n}{13} \rfloor$	$\lfloor \frac{n+h}{3} \rfloor$	Biro [3]
	Orthogonal polygons	$\lfloor \frac{n}{6} \rfloor$	$\lfloor \frac{n}{6} \rfloor$	Bae et al. [1]

Table 1.1: The best known results in two dimensions.

many guards are needed to cover the interior of an art gallery with n walls. Chvátal [9] answered his question in 1975 by showing that $\lfloor n/3 \rfloor$ guards are always sufficient and sometimes necessary. His proof was later simplified by Fisk [10]. From then on, various aspects of the problem were studied, for example, orthogonal polygons, polygons with holes, guards which can move along a line segment, special polygon types, and exterior visibility. A good overview can be found in the book by O'Rourke [15].

However, in contrast to two-dimensional art gallery guarding, very few results are known for the three-dimensional case. The known results are mainly negative ones, i.e., lower bounds on the number of guards needed. O'Rourke [15] showed that there exists a polytope for which it is not sufficient to place a guard at all vertices to guard all interior points. Additionally, his construction for arbitrarily many vertices n needs $\Omega(n^{3/2})$ many guards. Later, Michael [14] introduced a different example which reaffirms both results by O'Rourke.

Chapter 2

Preliminaries

In this chapter we will give the relevant definitions used throughout the remaining work. We will also show some preliminary results that follow from the definitions.

2.1 Geometric terms

For the definition of geometric objects in three dimensions we mainly refer to Lee and Santos [11] who defined polytopes in general from a combinatorial point of view. Note that we have made some small changes to their original definitions to fit our setting.

Definition 2.1 ([cf. 11, p. 415, polytope]). A *convex polytope* P is the convex hull of a finite set V of points in $\mathbb{R}^k$ for some $k \in \mathbb{N}^{\geq 1}$. Its *dimension* $\dim(P)$ is the dimension of the affine hull of V.

Definition 2.2 ([cf. 11, pp. 415f., polytope]). A *face* of a convex polytope P is the set $P^f := \{x \in P \mid f(x) \geq f(y)\ \forall y \in P\}$ that maximizes a linear functional f.[a] The empty set and P are considered faces and every face is a convex polytope, of dimension ranging from -1 (empty set), 0 (vertices), 1 (edges), $2, \ldots$, to $d-1$ (facets), and $d = \dim(P)$.

[a] A linear functional f is a linear map from a vector space to its field of scalars, e.g., $f : \mathbb{R}^d \to \mathbb{R}$.

Definition 2.3 ([cf. 11, pp. 415f., polytope]). The *boundary* ∂P of a d-dimensional convex polytope P is the union of all its proper faces, that is, all faces with dimension $< d$. The *interior* $\operatorname{int}(P)$ is P without its boundary: $\operatorname{int}(P) := P \setminus \partial P$.

With this we have some knowledge about convex polytopes and their structure. We now combine several convex polytopes into polytopal complexes which will be the base for our definition of a polytope afterwards.

Definition 2.4 ([cf. 11, p. 416])**.** A *polytopal complex* is a finite, nonempty collection $S = \{P_1, \dots, P_l\}$ of convex polytopes in $\mathbb{R}^k$ such that $P_i \cap P_j$ is always a common face of both (possibly empty).

The *dimension* of S is the largest dimension of all $P_i \in S$, $\dim(S) := \max_{P_i \in S} \dim(P_i)$.

We want our polytopal complexes to have some special properties. For example, we do not want to allow a three-dimensional convex polytope with a single edge attached to its outside. Additionally, we want our polytopal complex to be connected. This means that there exists a path between each pair of points in the polytopal complex which is contained in the polytopal complex. These two constraints are covered in the following definitions.

Definition 2.5 ([cf. 11, p. 416])**.** A polytopal complex S is *pure* if all convex polytopes in S have the same dimension.

Definition 2.6. A polytopal complex S is *connected* if for every $P_i, P_j \in S$ and every $s \in P_i$ and $t \in P_j$ there is a path p from s to t which lies in the union of all convex polytopes, i.e., $p \subseteq \bigcup_{P_i \in S} P_i$.

We can now define what we mean with the term *polytope*, namely a pure, connected polytopal complex. Our definition of *connected* does not prevent two convex polytopes to only touch at a common vertex. However, given a polytopal complex with dimension d, we want to enforce that each pair of convex polytopes is either disjoint or that they share a common face of dimension $d - 1$. Thus we add some additional constraint in the following

Definition 2.7. A *polytope* S is a pure, connected polytopal complex of dimension d with the following additional constraint: For every $P_i \in S$ and every $P_j \in S$ with $j \neq i$ either $P_i \cap P_j = \emptyset$ or $\dim\left(P_i \cap P_j\right) = d - 1$.

We also refer to $P := \bigcup_{P_i \in S} P_i$ as the *polytope* P.

The presented definitions are general, that is, they work for arbitrary dimensions d. Since, as the title of this thesis suggests, we only talk about three dimensions, we restrict the definitions to three-dimensional polytopes in $\mathbb{R}^3$.

Additionally, whenever we talk about a polytope in the rest of this work we usually refer to it as a subset of $\mathbb{R}^3$ and not as a collection of convex polytopes. However, at least for three dimensions, one can always find a finite collection of convex polytopes which yields the polytope we are talking about. This is due to a result of Bern and Eppstein [2] which states that it is always possible to decompose a three-dimensional polytope into $\mathcal{O}\left(n^2\right)$ tetrahedra with the help of $\mathcal{O}\left(n^2\right)$ so-called Steiner points.

Even though it should be clear intuitively what constitutes the boundary and the interior of a polytope, we give a precise definition:

Definition 2.8. The *boundary* ∂P of a d-dimensional polytope P (with S being the collection of convex polytopes) is the union of the boundaries of its convex polytopes minus the interior of the shared faces of dimension $d-1$:

$$\partial P := \bigcup_{P_i \in S} \left(\partial P_i \setminus \bigcup_{B \in B_i} B \right), \text{ with}$$
$$B_i := \left\{ \operatorname{int}(P_i \cap P_j) \mid P_j \in S, j \neq i, \dim(P_i \cap P_j) = d-1 \right\}.$$

The *interior* $\operatorname{int}(P)$ is the polytope without its boundary, $\operatorname{int}(P) := P \setminus \partial P$.

Observation 2.9. The additional constraint introduced in Definition 2.7 implies that the interior of a polytope is always connected.

Observation 2.10. Note that our definition of a polytope does not impose any constraints on the number of *holes* or *cavities*. The two terms are interpreted as follows:

(i) A hole means a hole in the sense of holes in donuts. This means that, topologically speaking, the *genus* of the polytope's surface (or boundary as defined by Definition 2.8) can be arbitrarily high.

(ii) A cavity is interpreted like this: Given two three-dimensional polytopes P_1 and P_2 where P_2 is completely contained in the interior of P_1, i.e., $P_2 \subseteq \operatorname{int}(P_1)$. If we now look at the polytope $P := P_1 \setminus \operatorname{int}(P_2)$ we obtain a new polytope P with the cavity P_2.

Having a cavity also implies that the boundary ∂P is not connected. In our example $\partial P = \partial P_1 \cup \partial P_2$ and since $P_2 \subseteq \operatorname{int}(P_1)$ we see that ∂P_2 cannot touch ∂P_1.

2.2 Beacon terminology

In the following, we will define the characteristics of beacons and their effects. Many of the definitions encountered in this section are very similar, if not equal, to those given by Biro [3] for two dimensions. Where applicable, we indicate the relevant counterpart.

Definition 2.11 ([cf. 3, Definition 2.3.1])**.** A *beacon* b is placed at a point in a polytope P and can be enabled and disabled. When enabled, b exerts a pull on point-like objects in P. The objects then move to greedily minimize their Euclidean distance to b, while being constrained to remain in P.

The movement of a point-like object p under the influence of a beacon b is called the *attraction path* of p towards b.

Note 2.12. In the following when we mean that a point-like object placed at p moves towards a beacon b we will sometimes say that "a point p moves towards b". It should still be clear that the point itself does not move. We will also sometimes refer to a beacon as the point at which it is placed.

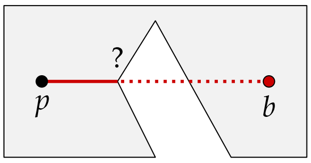


Figure 2.1: When b is enabled the point p first moves in a straight line. Then it is unclear whether it slides either upwards or downwards.

Observation 2.13. The attraction path is not well-defined in all cases. Let p be a point and b a beacon in P. If the attraction path of p towards b touches ∂P at an edge or a vertex it is sometimes possible for the movement to proceed in more than one way while still minimizing the distance to b.

Consider the situation in Figure 2.1 where the attraction path from p towards b could either slide upwards (and thus reach b) or slide downwards and get stuck. Since there is no "right" way to solve the situation we make the following

Assumption 2.14. If the attraction path of a point-like object p towards a beacon b is not well-defined and there is a possible attraction path which *does not* terminate in b, then p will take this path.

This assumption makes sure that we never rely on non-deterministic behavior.

Lemma 2.15. *Every attraction path is a polygonal chain, i.e., a connected series of line segments given by a series of points.*

Proof. Let p be a point and b be a beacon in the polytope P and let p be attracted by b. There are two possible movements for each part of the attraction path.

The first movement is the unconstrained movement. This is always the case if there exists an $\varepsilon > 0$ such that $p' = p + \varepsilon\overrightarrow{pb}$ and the line segment pp' is completely contained in P. Then p moves along the line segment pb until it either reaches b or its movement becomes constrained at a point $q \in \partial P$. In the first case (p, b) is the polygonal chain of the movement and in the second case (p, q) is a polygonal chain which is a prefix of the attraction path. The rest of the attraction path of p is the attraction path of q.

The second movement is the constrained movement. For this it is necessary that $p \in \partial P$ holds. Then there exists a two-dimensional facet F of P with $p \in F$. If there are multiple such facets choose the one along which p will move towards b, taking into account Assumption 2.14.

We now split the movement $\overrightarrow{pb}$ into two components $\overrightarrow{pb}_\perp$, which is orthogonal to F, and $\overrightarrow{pb}_\in$, which lies in F, such that $\overrightarrow{pb}_\perp + \overrightarrow{pb}_\in = \overrightarrow{pb}$. Since the movement along $\overrightarrow{pb}_\perp$ is blocked by F the point p moves along the remaining vector $\overrightarrow{pb}_\in$. This essentially is a projection of $\overrightarrow{pb}$ onto F. If $q = p + \overrightarrow{pb}_\in \in \operatorname{int}(F)$ the movement gets stuck at q since then $\overrightarrow{qb}_\in = \vec{0}$ and thus every point around q in F is further away from b. Then the movement of p is described by the polygonal chain (p, q).

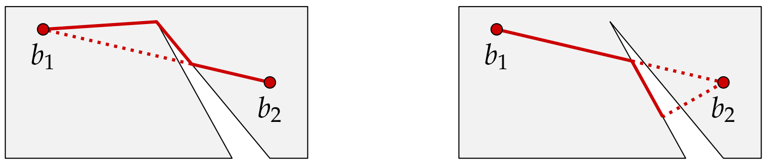


Figure 2.2: Attraction is not symmetric. In this two-dimensional example b_1 attracts b_2 (left) but b_2 does not attract b_1 (right).

If, on the other hand, $p + \overrightarrow{pb}_{\in} \notin \text{int}(F)$ there is a point $q \in \partial F$ with $q = p + r\overrightarrow{pb}_{\in}$ for some $r \in \mathbb{R}^{>0}$. The point p moves in a straight line towards q and thus (p, q) is the prefix of the attraction path of p towards b until reaching q. For the movement of q we obtain a polygonal chain by looking again an the two possible cases. □

Definition 2.16 ([cf. 3, Definition 2.3.2])**.** A beacon b *attracts* a point p if, while b is enabled, an point-like object starting at p moves so that it eventually reaches the location of b, that is, the Euclidean distance to b is 0. We also say that *p is attracted by b.*

Observation 2.17*.* Given a polytope P, a beacon $b \in P$ *attracts* a point $p \in P$ if and only if the *attraction path* of p towards b terminates at the location of b.

Note 2.18*.* One should be aware that the notion of *attraction* as defined here is different from the common use of the word. In common use we would say that b attracts p if p moves under the influence of b. With Definition 2.16 however, a beacon b only attracts p if p eventually reaches b.

Definition 2.19. A point p is a *dead point* with respect to a beacon b if p does not move when b is enabled. This means that its attraction path towards b consists only of p.

Observation 2.20*.* Attraction is a concept similar to visibility. We note that attraction is more powerful, i.e., for any point p the visibility region of p (all points p can see) is a subset of the attraction region of p (all points attracted by p). Additionally, as opposed to visibility, attraction is not symmetric. See Figure 2.2 for a two-dimensional example of two points b_1 and b_2 where b_1 attracts b_2 but is not attracted by it.

Different from visibility, the idea of attraction might lead to think about the case where multiple beacons are enabled. Here, the movement will depend on the strength of the attraction: One can think about strength which is relative to the distance (like magnets) or strength which is the same regardless of the distance. In the latter case the attracted point would try to move to the beacons' center of gravity. One could also think about enabling and disabling various beacons while a point is moving, which would complicate the point's movement a lot. To simplify our model for not needing to look at those more complex situations we make the following

Assumption 2.21. At all times, at most one beacon is enabled.

This assumption implies that, whenever we say that a beacon is enabled, it also means that all other beacons are disabled.

2.2.1 Routing

In the area of beacon-based *routing* we are interested in moving a point at a location inside a polytope to another location solely with the help of beacons. More specifically we are interested in finding minimum beacon paths between two given points, that is, a minimal number of beacons with which it is possible to route from the start to the target location. To do this, we need to define the term *beacon path* and what makes it a *minimum beacon path*.

Definition 2.22 ([cf. 3, Definition 6.1.1])**.** Given a polytope P and two points $s, t \in P$, a *beacon path* from s to t in P is a sequence of locations $s = b_0, b_1, b_2, \dots, b_k, b_{k+1} = t$ in P such that b_{i+1} attracts b_i for all $i = 0, 1, 2, \dots, k$. The *length* of such a path is k, i.e., the number of locations without s and t.

If P is clear from the context we just say: a *beacon path* from s to t.

Note 2.23. We would like to point out that $b_{k+1} = t$ is not considered an additional beacon in the sense that it is not counted in the number of beacons necessary to route s to t. Otherwise, to route from a fixed start s to all points in P, we would need to place a beacon at every possible destination t, resulting in uncountably many beacons.

The idea is to enable b_1 which attracts an object at the location of s. When the object reaches b_1, we activate b_2 and so on. In the end we activate t which attracts the object to its final location. From this definition of a beacon path we can extract the following assumptions about our model:

Assumption 2.24 ([3, Assumption 6.1.3])**.** Each beacon must lie inside the polytope.

Assumption 2.25 ([3, Assumption 6.1.4])**.** Each beacon may be activated at most once.

Assumption 2.26 ([3, Assumption 6.1.5])**.** The routed object must reach the activated beacon before the next beacon may be activated.

These assumptions were also made by Biro [3] to simplify the model. We chose to follow his path to make the results comparable. With this in mind, we can now give a definition of a *minimum beacon path*.

Definition 2.27 ([cf. 3, Definition 6.1.6])**.** Given a polytope P and two points $s, t \in P$, a *minimum beacon path* from s to t in P is a beacon path with minimum length among all beacon paths from s to t.

Definition 2.28 ([cf. 3, Definition 6.1.7])**.** Given a set B of beacon locations in a polytope P and two points $s, t \in P$, we say that *s is routed to t in B* if there exists a beacon path $s = b_0, b_1, b_2, \dots, b_k, b_{k+1} = t$ with $b_1, b_2, \dots, b_k \in B$.

2.2.2 Coverage

When looking at beacon-based *coverage* (sometimes also called *guarding* due to the similarity to art gallery guarding) the idea is to find a set of beacons B such that every point in the domain to be covered (in our case a polytope) is attracted by at least one of the beacons in B. We look at this more formally with the following

Definition 2.29. Given a polytope P, a set $B \subseteq P$ of beacon locations is said to *cover* the polytope P if for every point $p \in P$ there exists a beacon $b \in B$ such that p is attracted by b.

2.2.3 Relation between routing and coverage

Even though routing and coverage are similar, there are important differences. The most important difference is that routing is symmetric, i.e., we must be able to route s to t and also t to s, whereas coverage is asymmetric, i.e., every point must be attracted by some beacon but the beacons do not need to be attracted by them. If we look again at the two-dimensional example in Figure 2.2 we can see that b_1 covers the polygon but it is not possible to route between any pair of points with b_1 being the only beacon.

With this in mind it should be clear that a set of beacons able to route between any pair of points is also able to cover a polytope. This is formalized in the following

Lemma 2.30 (Routing implies coverage)**.** *Given a polytope P and a set of beacons $B \subseteq P$. If for all $s, t \in P$, s is routed to t in B then*

(i) if $B = \emptyset$ then for every point $p \in P$ the set $\{p\}$ covers P or

(ii) if $B \neq \emptyset$ then B covers P.

Proof. We show this individually for each case.

(i) If $B = \emptyset$ then for every pair of points $s, t \in P$ the target t attracts s. This means that every point t can attract every point s.

(ii) Assume $p \in P$ is a point not covered by B. Since p is not covered by B, no $b \in B$ can attract p. Then p cannot be routed to any of the $b \in B$, either. This is a contradiction. □

Corollary 2.31 (Non-coverage implies non-routing)**.** *Given a polytope P and a non-empty set of beacons $B \subseteq P$ which does not cover P. Then there exist $s, t \in P$ such that s is not routed to t in B.*

Chapter 3

Two-dimensional beacon-based routing revised

One of the first things studied by Biro et al. [4] after introducing the notion of beacon-based routing was the problem's combinatorial complexity. Here, one of the main questions is:

> Given a two-dimensional polygon P, how many beacons are needed to be able to route between any pair of points inside the polygon via beacons?

The main result to this question is the following theorem in which n is the number of vertices of P.

Theorem 3.1 (Theorem 1 by Biro et al. [5]). *Given a simple polygon P, $\lfloor \frac{n}{2} \rfloor - 1$ beacons are sometimes necessary and always sufficient to route between any pair of points in P.*

Here *simple* means that the edges of the polygon do not intersect, except for the vertices which are shared by exactly two consecutive edges.

The idea of their proof is to triangulate the polygon, yielding $n - 2$ triangles, and then successively remove at least two triangles by placing one beacon. They show that the beacon can always be placed so that it lies on the boundary of the remaining polygon and can always see the whole interior of the removed triangles. They can then conclude that at most $\lfloor n/2 \rfloor - 1$ beacons are needed and show a construction which always needs this many beacons.

3.1 Revision of the upper bound for simple polygons

While proving the upper bound of the above theorem they analyze different configurations of triangles to show that by placing one beacon at least two triangles can be removed. One of the cases is that of a triangle $\sigma_2 = \triangle BCD$ having two adjacent triangles $\sigma_1 = \triangle ABC$ and $\sigma_3 = \triangle CDF$ whose interior should be completely attracted by a beacon b positioned on the leftover line segment of σ_2, namely BD. See Figure 3.1

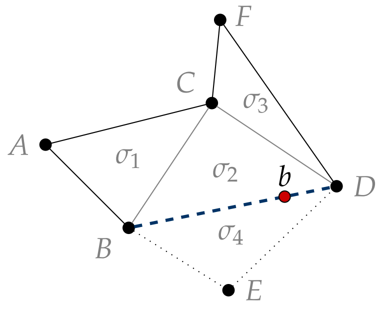


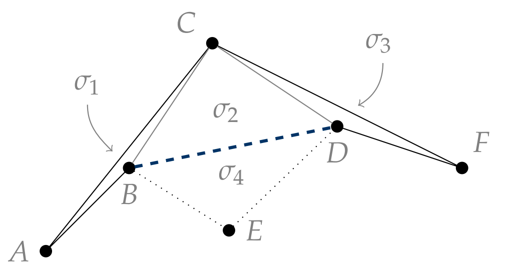
Figure 3.1: The situation as analyzed by Biro et al. [5]. Here b can be placed near D to fulfill the requirement of seeing every point inside $ABDFC$.

for an explanatory depiction. Note that the order of the names of the vertices is rather unusual but it was chosen to fit the original description. Then they state the following:

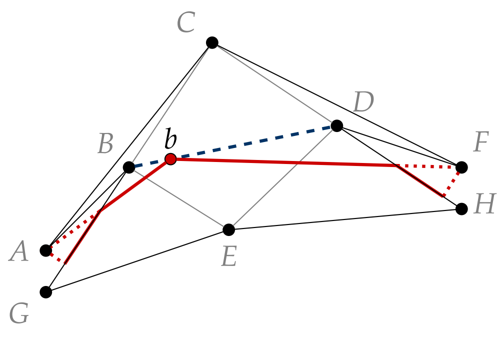
> The location b along BD is chosen so the pentagon $ABDFC$ is visible to b. This is always possible, by placing b on the correct side of lines CF and AC. Then, any point in triangles $\triangle ABC$, $\triangle BCD$, $\triangle CDF$ can be routed to or from b as b is *visible* to each point in those triangles. — Biro et al. [5, p. 2]

However, the constraint that b needs to lie to the right of AC and to the left of FC is not sufficient. Additionally, b needs to lie to the left of AB and the right of FD, that is, in the visibility cone of both triangles σ_1 and σ_3. Figure 3.2(a) shows an example where those stronger constraints cannot be fulfilled. Here, the line through B and D constrains the visibility of a beacon b on the line segment BD unless it is placed at B or D. This still means, however, that b cannot see the full pentagon.

Since we only need mutual attraction and not visibility, we might think that this problem is irrelevant here. In fact, we can always place b such that it can attract all



(a) In this situation b cannot be placed on BD to see the full pentagon.



(b) Wherever b is placed on the line segment BD, it cannot be attracted by both A and F.

Figure 3.2: It is not possible to place one beacon b on the line segment BD such that it attracts and is attracted by all points inside the pentagon $ABDFC$.

points inside the pentagon $ABDFC$ but the reverse does not hold. Look at Figure 3.2(b) for an example. If b is not placed at B we can position A very close to G such that A cannot attract b. The same holds true for F. Since b cannot be placed at B and D at the same time, it cannot be placed at all without violating the constraint.

Nevertheless, Theorem 3.1 still holds as we will show in the following lemma. We will show the complete upper bound and indicate where we introduced our new idea.

Lemma 3.2 (Two-dimensional upper bound). *Given a simple polygon P with $n \geq 2$ vertices, $\lfloor \frac{n}{2} \rfloor - 1$ beacons are always sufficient to route between any pair of points in P.*

Proof. First note that we have restricted n to be at least 2. Otherwise a the number of allowed beacons is negative. For $n = 2$ P is just a line segment. We do not ignore this case out of technical reasons—we will need the case where a polygon consists of just a line segment.

Triangulate P and look at the dual graph T of the triangulation: the triangles form the set of nodes and two vertices are connected if and only if the corresponding triangles share an edge in the triangulation. Because P is simple, T is a tree with $n - 2$ nodes and we require it to be rooted at an arbitrary leaf. The idea is to place a beacon which is included in at least three triangles and then remove two of them. By induction we will show that this is always possible.

Base case ($n \leq 4$). If $2 \leq n \leq 3$ then P is either a line segment, or a single triangle. In both cases no beacon is needed due to the convexity of P's shape.

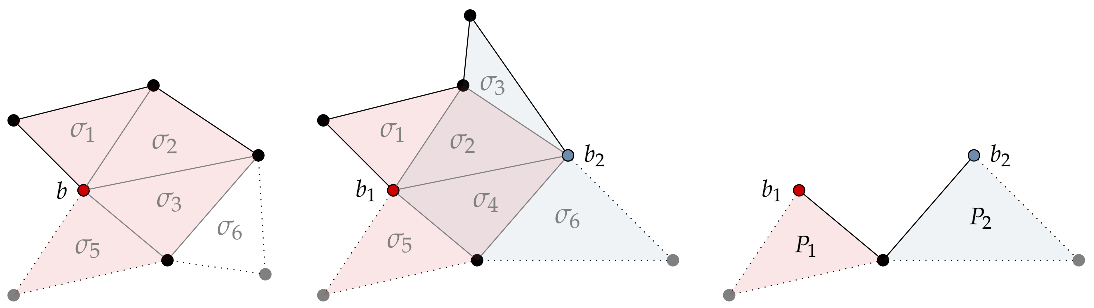
For $n = 4$ the polygon P consists of exactly two triangles which share a common edge. We are allowed to place one beacon which we place at an arbitrary point b along the shared edge. Then every point $p \in P$ can see b which means that they mutually attract each other. Then we can route from every point $s \in P$ to every point $t \in P$ via b.

Inductive step ($n > 4$). Assume that Lemma 3.2 holds for all polygons with strictly less than n vertices. Look at a triangle σ_1 which is a deepest leaf in T and whose parent triangle is called σ_2. We then distinguish between two cases:

(i) σ_2 has no other children. Let σ_3 be σ_2's parent triangle. Then σ_1, σ_2, and σ_3 share a common vertex at which we place a beacon b. This situation can be seen in Figure 3.3(a).

Afterwards we remove the triangles σ_1 and σ_2 which leaves a new simple polygon P' with $n' = n - 2$ vertices. By the induction hypothesis we know that we need at most $\lfloor \frac{n'}{2} \rfloor - 1 = \lfloor \frac{n-2}{2} \rfloor - 1 = \lfloor \frac{n}{2} \rfloor - 2$ beacons to route in P'. Together with the one beacon b we place this satisfies the boundary of $\lfloor \frac{n}{2} \rfloor - 1$.

We still need to show that we can route between any pair of points in P. By the induction hypothesis and because b is contained in σ_3 which remains in P' we can route between b and any point in P'. Additionally, every point in σ_1 or σ_2



(a) The beacon b covers at least three triangles: σ_1, σ_2, and σ_3.

(b) The two beacons b_1 and b_2 cover σ_1, σ_2, σ_3, and σ_4 and both neighbors of σ_4.

(c) After removing σ_1 to σ_4 two (possibly empty) polygons P_1 and P_2 remain.

Figure 3.3: The two possible configurations in the inductive step are shown in (a) and (b). In (c) we can see the situation of (b) after removing the triangles.

sees b due to convexity and can thus attract and be attracted by it. Hence, we can route between any pair of points via b.

(ii) σ_2 has a second child σ_3. This is the erroneous case of the original proof by Biro et al. [5].

Let σ_4 be σ_2's parent triangle. We know that σ_3 is a leaf in T because σ_1 is a deepest one. Look at Figure 3.3(b) for a depiction of the situation.

Instead of placing one beacon and removing three triangles, as suggested in the original proof, we will place two beacons and remove four triangles. Beacon b_1 is placed at the common vertex of σ_1, σ_2, and σ_4 (marked red) and b_2 at the common vertex of σ_3, σ_2, and σ_4 (marked blue). Note that if σ_4 has additional neighbors σ_5 or σ_6 they are also covered by one of the beacons.

We now remove σ_1 to σ_4 which leaves two polygons P_1 and P_2 which share exactly one vertex. If either polygon is "non-existent" it still consists of the line segment between the incident beacon and the shared vertex. This will be important in the next step.

Let $n_1 \geq 2$ and $n_2 \geq 2$ be the number of vertices of P_1 and P_2, respectively. Here, it is important that a "non-existent" polygon still has two vertices. We know that $n_1 + n_2 = n - 2$ since we have removed three vertices and P_1 and P_2 share one vertex which needs to be counted twice.

Because $n_1 < n$ and $n_2 < n$ we can apply the induction hypothesis individually to P_1 and P_2 which tells us that we need at most $k_1 = \lfloor \frac{n_1}{2} \rfloor - 1$ and $k_2 = \lfloor \frac{n_2}{2} \rfloor - 1$ beacons. To remove the four triangles we have placed two additional beacons which means that in total we never place more than $k_1 + k_2 + 2$ beacons. It remains to show that this number is never larger than the claimed upper bound of $\lfloor \frac{n}{2} \rfloor - 1$:

$$\begin{aligned}
& & k_1 + k_2 + 2 &\leq \left\lfloor \frac{n}{2} \right\rfloor - 1 \\
&\Leftrightarrow & \left\lfloor \frac{n_1}{2} \right\rfloor - 1 + \left\lfloor \frac{n_2}{2} \right\rfloor - 1 + 2 &\leq \left\lfloor \frac{n}{2} \right\rfloor - 1 \\
&\Leftrightarrow & \left\lfloor \frac{n_1}{2} \right\rfloor + \left\lfloor \frac{n_2}{2} \right\rfloor &\leq \left\lfloor \frac{n}{2} \right\rfloor - 1 \\
&\Leftrightarrow & \left\lfloor \frac{n_1}{2} \right\rfloor + \left\lfloor \frac{n_2}{2} \right\rfloor &\leq \left\lfloor \frac{n_1 + n_2 + 2}{2} \right\rfloor - 1 \\
&\Leftrightarrow & \left\lfloor \frac{n_1}{2} \right\rfloor + \left\lfloor \frac{n_2}{2} \right\rfloor &\leq \left\lfloor \frac{n_1 + n_2}{2} \right\rfloor
\end{aligned}$$

The last inequality trivially holds, showing that we never place too many beacons.

We still need to show that we can route between any pair of points in the original polygon P. By the induction hypothesis and because of the placement of b_1 and b_2 on the boundary of P_1 and P_2 we know that we can route between b_1 and any point in P_1 and between b_2 and any point in P_2. Additionally, b_1 and b_2 can see and attract each other, since they share an edge of the triangulation of P. Furthermore, all four removed triangles either contain b_1 or b_2 and thus every point in the removed tetrahedra can attract and be attracted by one of the two beacons. As a result, we can route every point $p \in P$ to and from b_1 or b_2 and between b_1 and b_2 which enables us to route between all pairs of points in P. □

3.2 A different lower bound for simple polygons

As stated in Theorem 3.1 and shown by Biro et al. [5], $\lfloor \frac{n}{2} \rfloor - 1$ is not only an upper but also a lower bound for the necessary number of beacons in simple polygons. We will prove this lower bound in a different way which will be extended to the three-dimensional case later on in Section 5.3. The idea for this construction is similar to the one used by Shermer [17] for the lower bound for beacon-based routing in orthogonal polygons.

We first show the construction of a class of spiral-shaped polygons for which we will then show that $\lfloor \frac{n}{2} \rfloor - 1$ beacons are needed for a specific pair of points.

Definition 3.3. For every $c \in \mathbb{N}^{\geq 1}$ and some small $0 < \delta < 1$ a *c-corner spiral polygon* is a simple polygon with $n = 2c + 2$ vertices. These vertices, given in counterclockwise order, are called s, q_1, q_2, …, q_c, t, r_c, r_{c-1}, …, r_1 and their coordinates are given in polar notation as follows:

- $s = (1; 0\pi)$, $t = \left(\left\lfloor \frac{c+1}{3} \right\rfloor + 1; (c+1) \cdot \frac{2}{3}\pi\right)$,
- $q_k = \left(\left\lfloor \frac{k}{3} \right\rfloor + 1 + \delta; k \cdot \frac{2}{3}\pi\right)$ for all $1 \leq k \leq c$, and
- $r_k = \left(\left\lfloor \frac{k}{3} \right\rfloor + 1; k \cdot \frac{2}{3}\pi\right)$ for all $1 \leq k \leq c$.

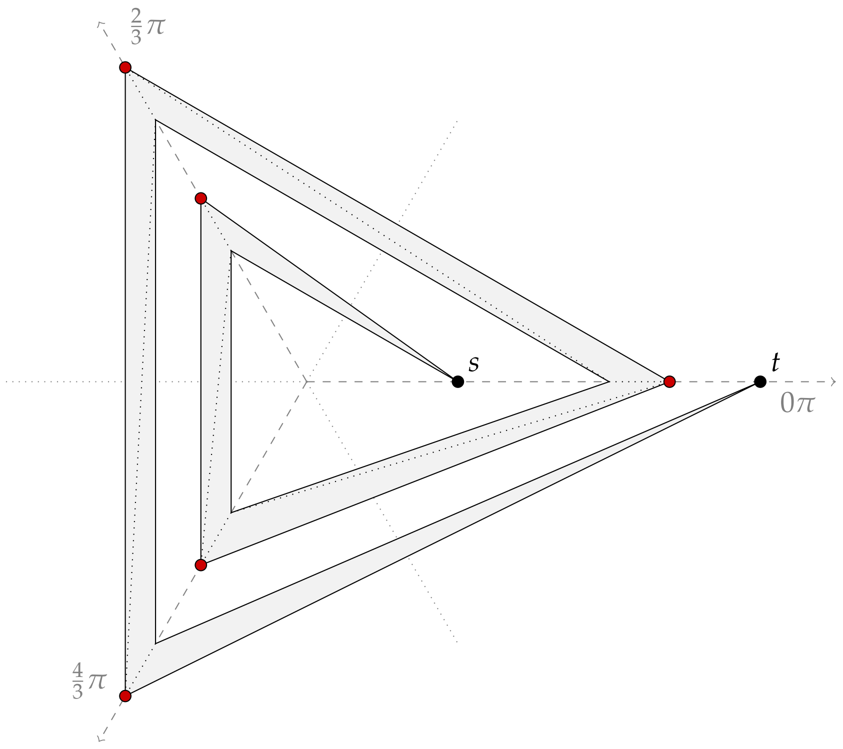


Figure 3.4: A 5-corner spiral polygon with $\delta = 0.4$ for which five beacons (marked in red) are necessary to route from s to t.

The two vertices r_k and q_k form the k-th *corner*. The $c-1$ trapezoids $\triangle r_k q_k q_{k+1} r_{k+1}$ for all $1 \leq k < c$ and the two triangles $\triangle s r_1 q_1$ and $\triangle t r_c q_c$ are each called *hallway*.

An example for $c = 5$ can be seen in Figure 3.4. There are five corners and we have already placed five beacons to be able to route from s to t.

Lemma 3.4 (Two-dimensional lower bound)**.** *Given a c-corner spiral polygon. Then c beacons are necessary to route from s to t.*

Proof. To show that we need c beacons we introduce some additional notational conventions as depicted in Figure 3.5. The interior angle at each (reflex) vertex r_k is called α_k. We split each hallway into two parts. To achieve this, three rays starting at the origin with the angles $\frac{1}{3}\pi$, π, and $\frac{5}{3}\pi$ (the dotted rays in Figure 3.4) are drawn. Every hallway is divided by exactly one of the three rays and the intersection points with the polygon's boundary are called a_k and b_k: a_k is the intersection of one of the rays with the edge $r_{k-1}r_k$ and b_k with the edge $q_{k-1}q_k$.

We observe that, due to the triangular shape, the angle α_k is always strictly less than $90°$, for every $1 \leq k \leq c$.

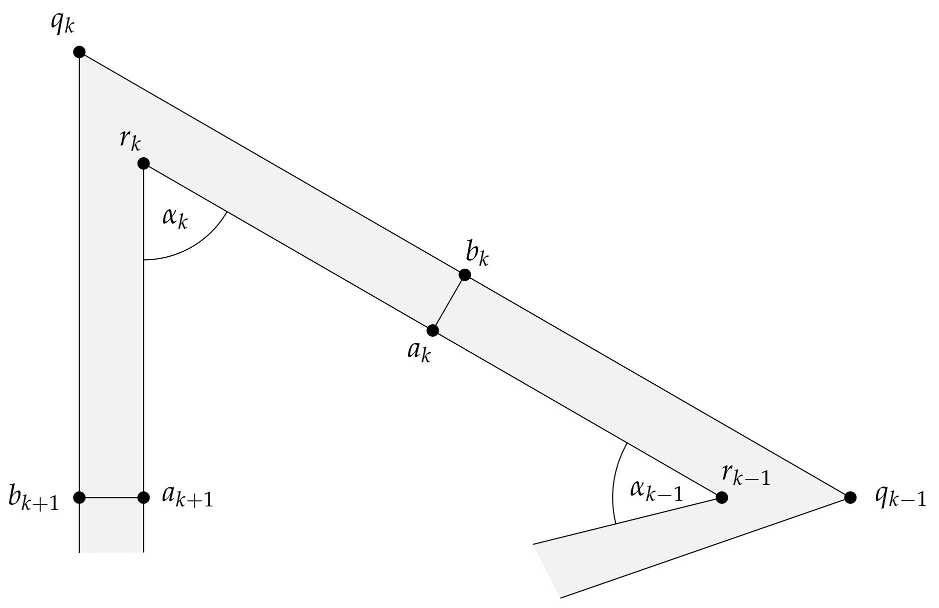


Figure 3.5: Notation for various parts on the triangular spiral.

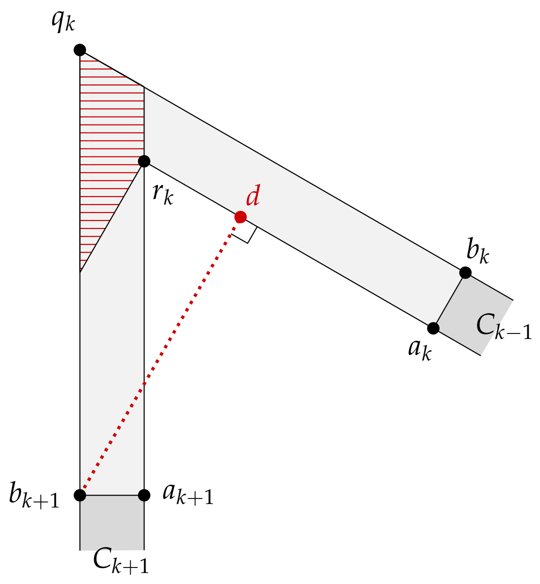


Figure 3.6: The complete corner C_k in lighter gray. It is not possible to route through this corner from C_{k-1} to C_{k+1} without a beacon inside the marked region. If b_{k+1} was a beacon, points from C_{k-1} would not travel further than d.

We now divide our polygon into $c+2$ subpolygons C_0 to C_{c+1} at each pair a_k and b_k. This subdivision results in two triangles C_0 and C_{c+1} which contain s and t, respectively, and c subpolygons C_1 to C_c which are called *complete corners* and which all have the same structure. We show that every such complete corner needs to contain at least one beacon to be able to route from s to t. We look at one complete corner C_k, $1 \leq k \leq c$, shown in Figure 3.6.

We want to route from C_{k-1} to C_{k+1}. To route any point from C_{k-1} (which all lie to the right of $a_k b_k$) towards b_{k+1} there has to be a beacon such that the shortest path of this beacon to the line segment $r_k a_k$ ends in r_k. Otherwise, an attracted point will get stuck on $r_k a_k$ because the shortest path ends somewhere on $r_k a_k$ (excluding r_k itself). In Figure 3.6 we see the case where b_{k+1} is a beacon. Here d is a dead point with respect to b_{k+1} and no point from C_{k-1} will travel further into C_k when attracted by b_{k+1}.

The marked region in Figure 3.6 is the region in which every point can attract at least all points on the line segment $a_k b_k$. Additionally, every point in the region can be attracted by a beacon somewhere in the first part of C_{k+1}, i.e., the region to the right of $b_{k+1} a_{k+1}$ and to the right of the line trough $r_k a_{k+1}$.

On the other hand, there is no better option than b_{k+1} for a beacon position outside of C_k. All other points in C_{k+1} lie to the right of the line through $b_{k+1} d$ and hence their respective dead point on $r_k r_{k-1}$ lies further away from r_k.

We can see that if the length of the hallways (the distance between r_k and r_{k+1}) is sufficiently large compared to their width (the distance between a_k and b_k) it is never possible for a point outside of C_k to be inside the marked region. Therefore it is not possible to route from somewhere inside C_{k-1} to somewhere inside C_{k+1} without an additional beacon inside C_k. $\square$

Chapter 4

Complexity of three-dimensional beacon-based routing and coverage

The primary goal of beacon-based routing in polytopes is to find a minimum set B of beacons such that we only need beacons from B to route between all pairs of points, from all points to one end point, from one start point to all end points, or between to specific points. For coverage in polytopes we want to find a minimum set B such that the whole polytope is covered by B. In this chapter, we first define the problems whose complexity we want to determine. We then explain briefly how Biro [3] showed hardness for a number of similarly defined problems in two dimensions. Afterwards, we show that the respective three-dimensional problems are NP-hard and APX-hard with a reduction from their two-dimensional counterparts.

4.1 The problems

We first define the problem of finding minimum beacon paths and its variations. The most interesting problems are to find a minimum set of beacons to route between all pairs of points, to route a specific start point to all points, to route all points to a specific end point, and to find a minimum beacon path to route from a known start point to a known end point.

Problem 4.1 ([cf. 3, Problem 6.1.8])**.** Given a three-dimensional polytope P, finding a minimum set B of beacons such that for every pair of points $s, t \in P$, there exists a beacon path from s to t in B is called the Three-Dimensional All-Pair Beacon-Routing Problem or the 3D-All-Pair Problem.

Problem 4.2 ([cf. 3, Problem 6.1.9])**.** Given a three-dimensional polytope P and a point $s \in P$, finding a minimum set B of beacons such that for every point $t \in P$, there exists a beacon path from s to t in B is called the Three-Dimensional All-Sink Beacon-Routing Problem or the 3D-All-Sink Problem.

Problem 4.3 ([cf. 3, Problem 6.1.10])**.** Given a three-dimensional polytope P and a point $t \in P$, finding a minimum set B of beacons such that for every point $s \in P$, there exists a beacon path from s to t in B is called the Three-Dimensional All-Source Beacon-Routing Problem or the 3D-All-Source Problem.

Problem 4.4 ([cf. 3, Problem 6.1.11])**.** Given a three-dimensional polytope P and a pair of points $s, t \in P$, finding a minimum beacon path is called the Three-Dimensional Given-Pair Beacon-Routing Problem or the 3D-Given-Pair Problem.

For beacon coverage there is mainly one problem—namely, to find a minimum set of beacons such that every point in the polytope is attracted by at least one beacon, or more formally:

Problem 4.5 ([cf. 3, Section 7.1])**.** Given a three-dimensional polytope P, finding a minimum set B of beacons such that for every point $p \in P$, there exists a beacon $b \in B$ such that b attracts p is called the Three-Dimensional Beacon-Coverage Problem or the 3D-Coverage Problem.

4.2 The idea of hardness in two dimensions

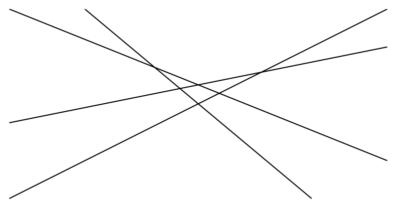
If we change the definitions of Problems 4.1 to 4.5 such that P is a polygon instead of a polytope P we obtain the corresponding definitions of the two-dimensional problems as defined by Biro [3]. He has shown that those two-dimensional problems, except for the two-dimensional Given-Pair Beacon-Routing Problem, are NP-hard and APX-hard by reducing from the Minimum Line Covering Problem. This problem was shown to be NP-hard by Megiddo and Tamir [13] and APX-hard by Brodén et al. [7] and it is defined as:

Problem 4.6 ([3, Problem 6.2.1])**.** Given a set of lines in the plane, the Minimum Line Covering Problem is finding the smallest set of points so that there is at least one point on each line.

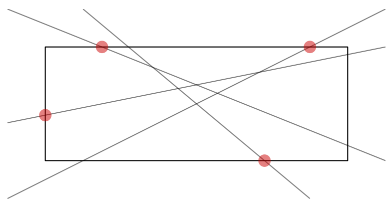
Biro [3] showed the hardness for all of the four problems individually but we will only present the idea of their reduction[1]: Given an instance of the Minimum Line Covering Problem, that is, an arrangement of lines in the plane, we construct a rectangle P, called *spike box*, such that all intersections of the lines are contained inside the rectangle. An example of the construction of P is shown in Figure 4.1. After certain modifications which we will describe subsequently, P will be the input polygon for the beacon problems.

Every line of the line arrangement leaves P at two locations. We pick one location arbitrarily and construct an additional gadget at this location The type of gadget

[1]For the complete proofs refer to Theorems 6.2.2, 6.2.3, 6.2.4, and 7.2.1 of Biro [3].



(a) An input for the Minimum Line Covering Problem (Problem 4.6).



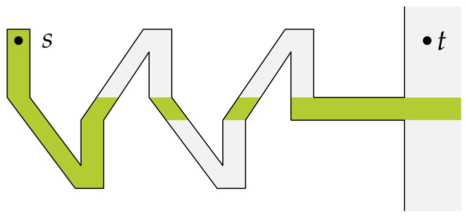
(b) The spike box which contains all intersections. Additional gadgets are constructed at the marked points.

Figure 4.1: Given a line arrangement for the Minimum Line Covering Problem it is transformed by first constructing a rectangular box which contains all intersection points. Then additional gadgets are added at one side where each line leaves the box.

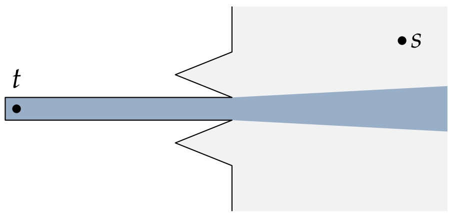
depends on the problem for which we want to show hardness and two of the three gadgets used can be seen in Figure 4.2.

The first *zigzag gadget* ensures that only beacons located inside the marked area can attract the point s. The area consists of the part inside the gadget and an arbitrarily thin region around the original line. For the two-dimensional All-Source Problem we place t somewhere in the rectangle without being near the lines and construct the zigzag gadget for every line. This prevents t from attracting the point s for every constructed gadget and thus a smallest beacon set necessarily has at least one beacon inside each marked region. Every beacon then corresponds to a point on the line of the original instance of the Minimum Line Covering Problem.

The second *inverted arrow gadget* is constructed for the opposite case. It guarantees that t can only attract points inside the marked area. As is visible in Figure 4.2(b), the area widens slightly due to the non-zero angle between t and the points at which the gadget is connected to the spike box. This problem is mitigated by choosing an arbitrarily small width for the gadget. For the two-dimensional All-Sink Problem



(a) A *zigzag gadget* to prevent t from attracting s if t is not placed anywhere inside the green area.



(b) An *inverted arrow gadget* to prevent t from attracting anything outside the blue attraction region.

Figure 4.2: The two gadgets used for reducing the two-dimensional All-Source Problem (left) and All-Sink Problem (right). In both cases an additional beacon is needed to route s to t.

we construct the *inverted arrow gadget* for every line and place s inside the spike box without being near any marked region. Then, with the same argument as before, we need at least one beacon in every marked region to be able to route s to all t for every gadget—and once again every beacon then corresponds to a point on a line in the Minimum Line Covering Problem. The two-dimensional All-Pair Problem can be treated with the same strategy as the two-dimensional All-Sink Problem.

To show hardness of the two-dimensional Coverage Problem a third gadget is used by Biro [3]. However, it should be sufficient to reuse the zigzag gadget as this gadget only allows beacons on the original line to attract all points inside the gadget. Since all points must be covered there is at least one beacon on every line and hence the position of the beacons give us a solution for the original problem.

4.3 Hardness in three dimensions

We will now show that the corresponding three-dimensional problems (Problems 4.1 to 4.3 and 4.5) are all at least as hard as their two-dimensional versions and thus both NP-hard and APX-hard. We will reduce the two-dimensional problems to the three-dimensional ones to prove their hardness, as shown in the following

Theorem 4.7. *The* 3D-All-Pair Problem*, the* 3D-All-Sink Problem*, the* 3D-All-Source Problem*, and the* 3D-Coverage Problem *are all* NP*-hard and* APX*-hard.*

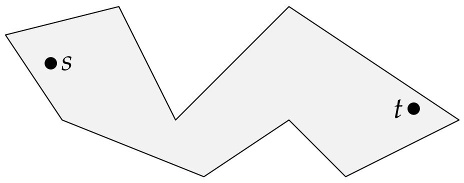
Proof. We will reduce from the respective two-dimensional problems which were already shown to be NP-hard and APX-hard. We first define that a beacon set B is *sufficient* if and only if B is a solution according to the problem definition without the constraint of being a minimum set.

The input for the respective two-dimensional problems consists of a polygon together with an optional target or start point. We denote those inputs as (P), (P,t), and (P,s). We transform it into inputs (P^*), (P^*,t), and (P^*,s) where P^* is a polytope according to the following construction:

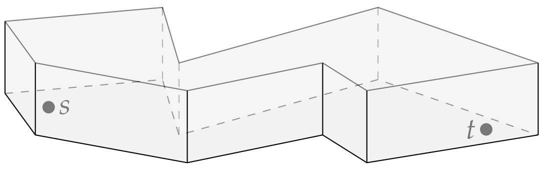
Suppose all vertices of P lie in the xy-plane, i.e., their z-coordinate is zero. We take a copy P' of P and set all of its z-coordinates to one. We then connect every vertex of P with the respective vertex of P'. This yields a polytope P^*. An example of the transformation can be seen in Figure 4.3.

Let B^* be a beacon set for the transformed three-dimensional problem and let $B = \{(x,y) \mid (x,y,z) \in B^*\}$ the projection of the solution to the xy-plane. We show that B is a sufficient solution for the two-dimensional problem if and only if B^* is a sufficient solution for the three-dimensional problem. We note that since $B^* \subseteq P^*$ also $B \subseteq P$ must hold.

We observe that the movement along the z-direction inside P^* is never obstructed if the z-coordinate lies in the interval $[0,1]$—if the z-coordinate is outside of this interval the corresponding point lies outside of P^*. Due to the fact that every cross section of P^* parallel to the xy-plane yields the same polygon P, we can split the movement of a



(a) A polygon P with two points. Depending on the problem, either s or t or both are missing.



(b) The result P^* of the transformation. The z-coordinates of s and t are set to zero.

Figure 4.3: The transformation of a two-dimensional instance (a) into a three-dimensional one (b).

point p under the influence of a beacon b into its xy-movement and its z-movement. Those two movements are independent and the latter linearly moves from z_p to z_b. The result is that the z-coordinate of the beacons in B^* is not important and can be set to zero for all of them. This essentially yields the set B as defined before and thus B is sufficient for the polygon P if and only if B^* is sufficient for the polytope P^*.

We now show that B^* is a minimum beacon set if and only if B is a minimum beacon set: Suppose B^* is a minimum beacon set and sufficient for the three-dimensional problem. Furthermore assume that there exists a beacon set O with $|O| < |B|$ which is sufficient for the two-dimensional problem. Then $O^* = \{(x, y, 0) \mid (x, y) \in O\}$ is sufficient for the three-dimensional problem. Because $|O^*| = |O| < |B| = |B^*|$ holds, B^* is not a minimum beacon set. This directly contradicts our assumption that B^* is a minimum beacon set.

On the other hand, if B^* is not a minimum beacon set then a sufficient and strictly smaller beacon set O^* exists which can be transformed into a sufficient two-dimensional beacon set O as before. Then O is smaller than B and hence B is not a minimum beacon set either. □

As a result of the theorem and with the assumption of P ≠ NP, no polynomial-time algorithm to obtain an optimal solution can exist for any of the four three-dimensional problems. Additionally, with the same assumption, no polynomial-time approximation scheme (PTAS) exists to find an approximate solution.

Regarding the 3D-Given-Pair Problem (Problem 4.4), we do not show any hardness. For the respective two-dimensional problem Biro [3] shows that there exists a PTAS and gives an algorithm to compute an optimal solution whose complexity is unknown. He uses techniques which were out of scope of this thesis to obtain those results and thus we have no similar results for three dimensions.

Some more details about the PTAS and the exact algorithm are given in Section 7.1 where we describe the open problems.

Chapter 5

Beacon-based routing in polytopes with a tetrahedral decomposition

In this chapter, we want to look at the combinatorial aspects of the routing problems to show upper and lower bounds. The first idea when tackling the problem is to transfer the two-dimensional approach by Biro [3] to three dimensions. In two dimensions the main idea is to look at a triangulation of the polygon. In our case, we will look at the three-dimensional analogon, the decomposition of a polytope into tetrahedra.

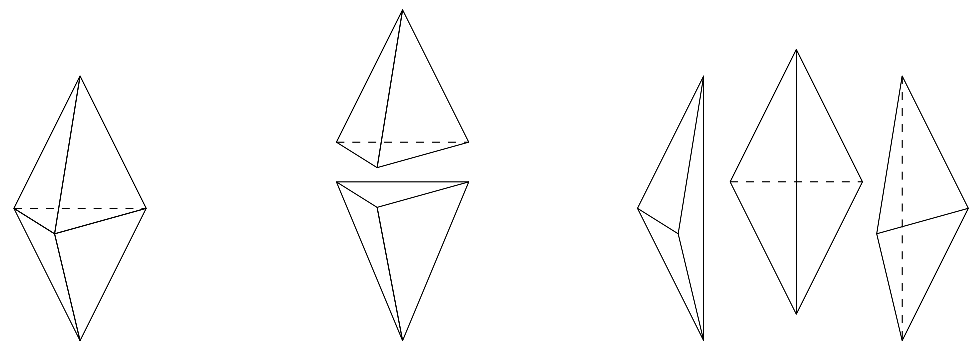
5.1 Preliminary thoughts on tetrahedral decompositions

Lennes [12] has shown in 1911 that a polyhedron cannot, in general, be decomposed into tetrahedra if no additional vertices are allowed. The problem of deciding whether such a decomposition exists is, in fact, NP-complete as shown by Ruppert and Seidel [16].

In the two-dimensional case, every simple polygon with n vertices has a triangulation with exactly $n-2$ triangles; a polygon with h holes has a triangulation of $n-2+2h$ triangles. In contrast, for three dimensions the number of tetrahedra in a tetrahedral decomposition is not directly related to the number of vertices. Chazelle [8] showed that for arbitrary n there exists a polytope with $\Theta(n)$ vertices for which at least $\Omega(n^2)$ convex parts are needed to decompose it. Naturally, this is also a worst-case lower bound on the number of tetrahedra. On the other hand, Bern and Eppstein [2, Theorem 13] show that any polytope can be triangulated with $\mathcal{O}(n^2)$ tetrahedra with the help of $\mathcal{O}(n^2)$ so-called Steiner points. Furthermore, it is clear that every tetrahedral decomposition consists of at least $n-3$ tetrahedra.

Additionally, the same polytope can have different tetrahedral decompositions with different numbers of tetrahedra. One such polytope can be found in Figure 5.1. Due to this, we will prove bounds on the number of beacons needed for routing relative to the number of tetrahedra m rather than the number of vertices n.

To successfully apply the ideas for two dimensions to three dimension we need some preliminary definitions and lemmas.


(a) A triangular bipyramid.

(b) Decomposition into two tetrahedra.

(c) Decomposition into three tetrahedra.

Figure 5.1: The number of tetrahedra in a polytope's tetrahedral decomposition is not unique: The bipyramid in (a) can be decomposed (b) into the upper and lower tetrahedron or (c) into three tetrahedra. (Source: [16, p. 228])

Definition 5.1 (Dual graph of tetrahedral decompositions)**.** Given a polytope with a tetrahedral decomposition $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ into k tetrahedra. Its *dual graph* is an undirected graph $D(\Sigma) = (V, E)$ where

(i) $V = \{\sigma_1, \ldots, \sigma_k\}$ and

(ii) $E = \left\{\{\sigma_1, \sigma_2\} \in \binom{V}{2} \;\middle|\; \sigma_1 \text{ and } \sigma_2 \text{ share exactly one triangular facet}\right\}$.

Observation 5.2. It is a fact that the dual graph of a triangulation of a two-dimensional simple polygon is always a tree with a constant maximum degree of 3. This is not true for tetrahedral decompositions in three dimensions. An example is the decomposition in Figure 5.1(c) whose dual graph is exactly K_3. Nevertheless, we can observe that each node in the latter dual graph has at most 4 neighbors—one for each facet of the tetrahedron.

Lemma 5.3. *Given a tetrahedral decomposition Σ of a polytope together with its dual graph $D(\Sigma)$ and a subset $S \subseteq \Sigma$ of tetrahedra from the decomposition whose induced subgraph $D(S)$ of $D(\Sigma)$ is connected. Then*

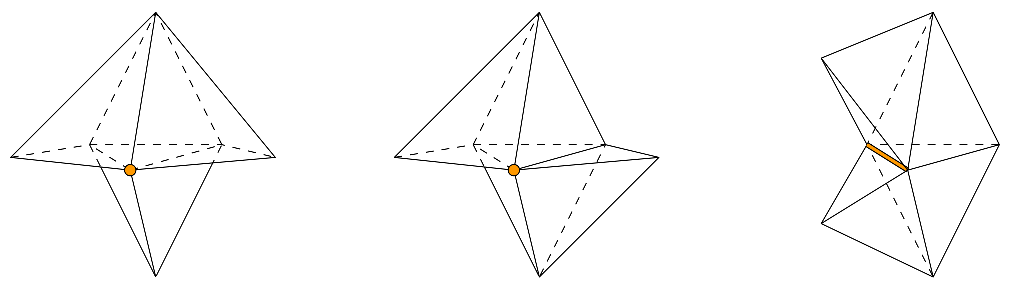
(i) $|S| = 2$ implies that the tetrahedra in S share one triangular facet,

(ii) $|S| = 3$ implies that the tetrahedra in S share one edge, and

(iii) $|S| = 4$ implies that the tetrahedra in S share at least one vertex.

Proof. We show this seperately for every case.

(i) This follows directly from Definition 5.1.


(a) One tetrahedron in the center has all other tetrahedra as neighbors.

(b) Two tetrahedra with one and two tetrahedra with two neighbors.

(c) In this configuration all four tetrahedra share one edge.

Figure 5.2: A polytope with a tetrahedral decomposition of four tetrahedra is in one of those three configurations. The shared vertex or edge is marked in orange.

(ii) In a connected graph of three nodes there is one node neighboring the other two. By Definition 5.1 the dual tetrahedron shares one facet with each of the other tetrahedra. In a tetrahedron every pair of facets shares one edge. Thus all three tetrahedra share this edge.

(iii) By case (ii) there is a subset of three (connected) tetrahedra that shares one edge e. This edge is therefore part of every of the three tetrahedra. By Definition 5.1, the fourth tetrahedron shares a facet f with at least one of the other three (called σ). Since f contains three and e two vertices of σ they share at least one vertex. A depiction of the possible configurations of four tetrahedra can be seen in Figure 5.2. □

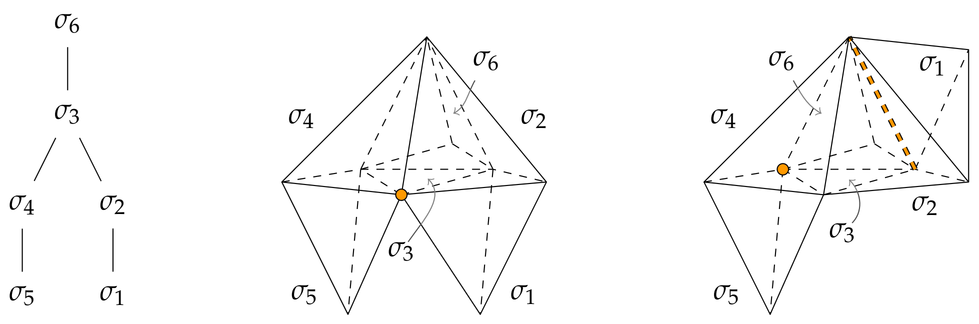
We now show that if the dual graph of a tetrahedral decomposition with six tetrahedra is in a very specific configuration, we have more information about the shared objects between those tetrahedra than what we already know from Lemma 5.3. This knowledge will be needed in Section 5.2.2.

Lemma 5.4. *Given a tetrahedral decomposition of six tetrahedra with the dual graph as depicted in Figure 5.3(a). Then at least one of the following holds:*

(i) σ_1 *to* σ_5 *share a common vertex, or*

(ii) σ_3, σ_4, σ_5, *and* σ_6 *share a common vertex* v; σ_1, σ_2, σ_3, *and* σ_6 *share a common edge* e; *and* $v \cap e = \emptyset$.

Proof. We first define $S_1 = \{\sigma_3, \sigma_4, \sigma_5, \sigma_6\}$ and $S_2 = \{\sigma_1, \sigma_2, \sigma_3, \sigma_6\}$. We observe that by Lemma 5.3 each set shares at least a vertex, but can also share an edge. We distinguish the cases by the shared geometric object:



(a) The dual graph of the tetrahedral decomposition.

(b) All tetrahedra but the rearmost tetrahedron σ_6 share one common vertex, here marked in orange.

(c) The four tetrahedra on the left share a common vertex while the right four tetrahedra share a common edge.

Figure 5.3: One tetrahedron σ_6 with a subtree of five tetrahedra. Subfigures (b) and (c) depict configurations that satisfy cases (i) and (ii) of Lemma 5.4, respectively.

- If both S_1 and S_2 each share an edge, case (i) holds. Each such edge needs to be part of the triangular facet which connects σ_3 and σ_6. Thus both edges share a common vertex.
- If just one of the two sets shares an edge e and the other shares only a vertex v there are two trivial cases: If $v \cap e = \emptyset$ then case (ii) is true—see Figure 5.3(c) for an example. On the other hand, if $v \cap e = v$ then case (i) holds.
- The last case is the one in which each of the sets shares only a vertex. The situation can be seen in Figure 5.3(b). First look at the vertex v of σ_3 not contained in the facet shared by σ_3 and σ_6, i.e., $v \notin \sigma_3 \cap \sigma_6$. In the figure v is marked in orange. We observe that then all neighbors of σ_3 except σ_6 contain v.

  Thus, σ_2 contains v and three of its four facets are incident to v. One of the facets is the shared facet with σ_3, but the other two are where σ_1 could be placed. σ_1 cannot be located at the fourth facet of σ_2, since it would then share an edge with σ_3 and σ_6 which is covered in the previous cases. Therefore, σ_1 contains v and with the same argument the same holds true for σ_5. Then case (i) holds. □

5.2 The upper bound

After the preparatory work, we can now show an upper bound on the number of beacons needed to route within a polytope with a tetrahedral decomposition. The idea of the proof is based on the proof by Biro et al. [4] for (two-dimensional) polygons. We want to show the following

Hypothesis 5.5. *Given a polytope P with a tetrahedral decomposition Σ with $m = |\Sigma|$ tetrahedra. Then it is always sufficient to place $\lfloor \frac{m+1}{3} \rfloor$ beacons to route between any pair of points in P.*

Since the proof is very long and consists of many cases we split it up into various lemmas and finally combine their results in Theorem 5.10.

5.2.1 Preparation

Given the polytope P and a tetrahedral decomposition Σ with $m = |\Sigma|$ tetrahedra, we look at the dual graph $D(\Sigma)$ of the tetrahedral decomposition. For the rest of the section we want the dual graph to be a tree. This is possible by looking at a spanning tree T of $D(\Sigma)$ rooted at some arbitrary leaf node.

In the following lemmas, we will place beacons depending only on the neighborhood relation between tetrahedra. If T is missing some edge $\{u, v\}$ from $D(\Sigma)$ we "forget" that tetrahedra u and v are neighbors, i.e., share a common facet. We have less information about a tetrahedron's neighborhood and thus we might place more beacons than needed—but never less.

Note 5.6. In the following we will refer to nodes of T as well as their corresponding tetrahedron with σ_i. It should be clear from the context when the node and when the tetrahedron is meant—if not, it is indicated.

The main idea of the proof is as follows: In a recursive way we are going to place a beacon and remove tetrahedra until no tetrahedra are left. As will be shown, for every beacon we can remove at least three tetrahedra which yields the claimed upper bound. We will show this by induction and start with the base case:

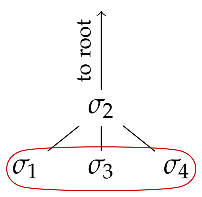
Lemma 5.7 (Base case). *Given a polytope P with a tetrahedral decomposition Σ with $m = |\Sigma| \leq 4$ tetrahedra. Then it is always sufficient to place $\lfloor \frac{m+1}{3} \rfloor$ beacons to route between any pair of points in P.*

Proof. If $m = 1$ then P is a tetrahedron and due to convexity no beacon is needed.

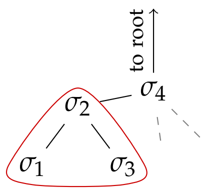
If $2 \leq m \leq 4$ we can apply Lemma 5.3 which shows that all tetrahedra share at least one common vertex v. We place the only beacon we are allowed to place at v. Then v is contained in every tetrahedron and thus, by convexity, every point in P can attract and be attracted by a beacon at v. □

We can now proceed with the inductive step, that is, polytopes with a tetrahedral decomposition of $m > 4$ tetrahedra. Our goal is to place k beacons which are contained in at least $3k + 1$ tetrahedra and can therefore mutually attract all points in those tetrahedra. Afterwards, we will remove at least $3k$ tetrahedra, leaving a polytope with a tetrahedral decomposition of strictly less than m tetrahedra, to which we can apply the induction hypothesis. We then need to show how to route between the smaller polytope and the removed tetrahedra.

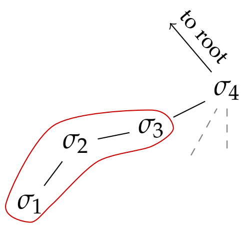
To do this, we look at a deepest leaf σ_1 of the spanning tree T. If multiple leaves with the same depth exist we choose the one whose parent σ_2 has the largest number of children, breaking ties arbitrary. In Figure 5.4 we can see different cases how the part of T which contains σ_1 and σ_2 might look like. We first concentrate on Figures 5.4(a)



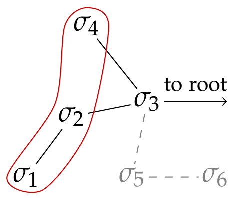
(a) Remove σ_1, σ_3, and σ_4 by placing a beacon where all four tetrahedra meet.



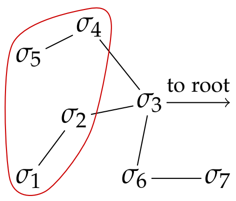
(b) Remove σ_1, σ_2, and σ_3 by placing a beacon where all four tetrahedra meet.



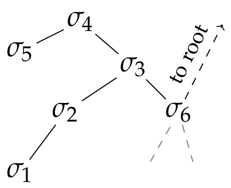
(c) Remove σ_1, σ_2, and σ_3 by placing a beacon where all four tetrahedra meet.



(d) Remove σ_1, σ_2, and σ_4 by placing a beacon where all four tetrahedra meet.



(e) Remove σ_1, σ_2, σ_4, and σ_5 by placing a beacon where σ_1 to σ_5 meet.



(f) The number and configuration of σ_6's children needs to be looked at.

Figure 5.4: The possible configurations in the first part of the inductive step.

to 5.4(e) and show for them the first part of the inductive step. Note that in all five cases there needs to be at least one additional root node—either because we have strictly more than four tetrahedra or because the tree is required to be rooted at a leaf node. The second part of the inductive step, namely Figure 5.4(f) will be dealt with in Section 5.2.2.

Lemma 5.8 (Inductive step I). *Given a polytope P with a tetrahedral decomposition Σ with $m = |\Sigma| > 4$ tetrahedra and a spanning tree T of its dual graph $D(\Sigma)$ rooted at some arbitrary leaf node. Let σ_1 be a deepest leaf of T with the maximum number of siblings and let σ_2 be its parent. Assume furthermore that any of the following conditions holds:*

(i) σ_2 has three children σ_1, σ_3, and σ_4 (see Figure 5.4(a)),

(ii) σ_2 has two children σ_1 and σ_3 and a parent σ_4 (see Figure 5.4(b)),

(iii) σ_2 has one child σ_1 and is the only child of its parent σ_3 whose parent is σ_4 (see Figure 5.4(c)),

(iv) σ_2 has one child σ_1 and its parent σ_3 has two or three children of which one, σ_4, is a leaf (Figure 5.4(d)), or

(v) σ_2 has one child σ_1 and its parent σ_3 has three children each of which has a single leaf child (Figure 5.4(e)).

Then we can place a beacon b at a vertex of σ_1 which is contained in at least four tetrahedra. We can then remove at least three tetrahedra containing b without violating the tree structure of T and while there is at least one tetrahedron left in T which contains b.

Proof. We show this individually for the conditions.

(i)–(iv) In all those cases the induced subgraph of the nodes σ_1, σ_2, σ_3, and σ_4 is connected. We can then see with Lemma 5.3(iii) that the four tetrahedra share at least one vertex at which b is placed.

After that we either remove σ_1, σ_3, and σ_4 (case (i)); σ_1, σ_2, and σ_3 (cases (ii) and (iii)); or σ_1, σ_2, and σ_4 (case (iv)). In all of those cases only leaves or inner nodes with all their children are removed which means that the tree structure of T is preserved. Additionally, we only remove three of the four tetrahedra that contain b, thus, one of them remains in T.

(v) Looking at Figure 5.4(e) we see that we have three different sets, each containing σ_3, a child σ_i of σ_3, and σ_i's child: $\{\sigma_1, \sigma_2, \sigma_3\}$, $\{\sigma_5, \sigma_4, \sigma_3\}$, and $\{\sigma_7, \sigma_6, \sigma_3\}$.

When applying Lemma 5.3(ii), we see that each set shares one edge, giving us three edges of σ_3. Since at most two edges in any tetrahedron can be disjoint, at least two of the given edges must share a common vertex. Without loss of generality let these be the edges shared by $\{\sigma_1, \sigma_2, \sigma_3\}$ and $\{\sigma_5, \sigma_4, \sigma_3\}$. We can then place b at the shared vertex and afterwards remove σ_1, σ_2, σ_4, and σ_5. The beacon b is also contained in σ_3 which remains in T. □

5.2.2 Special cases in the inductive step

Until now, we have ignored the configuration in Figure 5.4(f). The problem here is that to remove the tetrahedra σ_1 to σ_5 we need to place two beacons. Placing two beacons but only removing five tetrahedra violates our assumption that we can always remove at least $3k$ tetrahedra by placing k beacons. If we removed σ_6 and σ_6 had additional children then T would no longer be connected which also leads to a non-provable situation. Thus, we need to look at the number and different configurations of the (additional) children of σ_6.

Since there are many different configurations of σ_6's children (and their subtrees) we decided to use a brute force approach to generate all cases we need to look at. To do this, we wrote a small program in Python which generated all trees of maximum depth 3 where every inner node has at most three children. It then walks through the tree and applies Lemma 5.8 exhaustively to remove nodes and subtrees. Afterwards, the trees are normalized to prevent duplicate trees where only the order of the children is different. The code can be found in Appendix B.

The result of the program consists of 9 different trees with a depth of 3. A depiction of all those trees can be found in Figure 5.5. Note that in all cases but Figure 5.5(a) σ_6 cannot be the root node since we required T to be rooted at a leaf node. This means

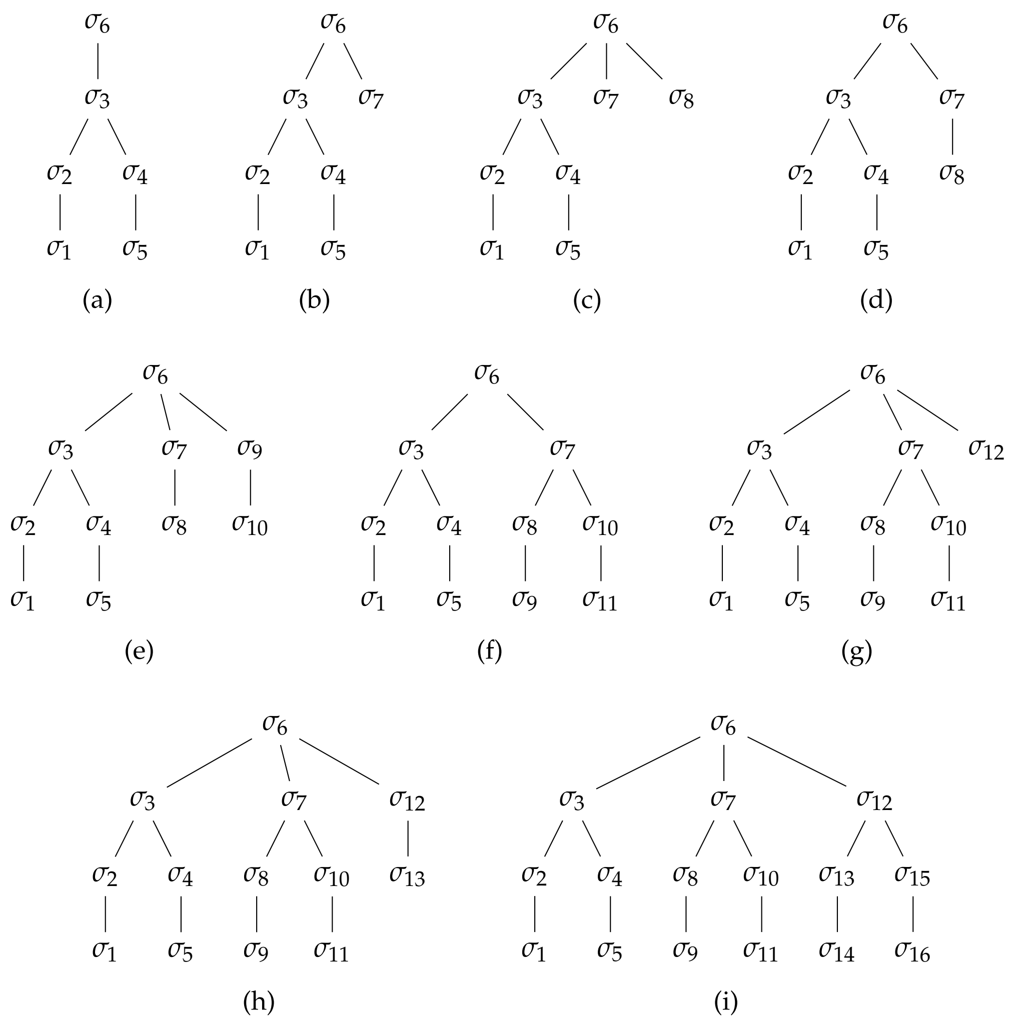


Figure 5.5: All "nontrivial" configurations of children of σ_6. The tree in (a) is a subtree of all configurations. In all cases σ_6 has no other children than the ones shown here. Furthermore by the requirement that T is rooted at a leaf node, σ_6 needs to have an additional parent (except for case (a)).

that when removing σ_6 we need to place a beacon at a common vertex of σ_6 and its parent. We show that we can always place beacons and remove sufficient tetrahedra with the following

Lemma 5.9 (Inductive step II)**.** *Given a polytope P with a tetrahedral decomposition Σ with $m = |\Sigma| > 4$ tetrahedra and a spanning tree T of its dual graph $D(\Sigma)$ rooted at some arbitrary leaf node. Let any of the cases shown in Figure 5.5 be a subtree of T.*

Then we can place k beacons which are contained in at least $3k + 1$ tetrahedra. Additionally, every beacon is contained in an edge together with at least one other beacon and the graph of *the beacons and the edges they are contained in is connected. We can then remove at least $3k$ tetrahedra, each of which contains a beacon, without violating the tree structure of T. After removal there is at least one tetrahedron left in T which contains one of the beacons.*

Proof. We show this individually for each of the nine cases.

Note that whenever we say that two beacons b_1 and b_2 *share an edge* or that b_1 and b_2 are *neighbors* we mean that there exists an edge e in the tetrahedral decomposition such that e is defined by the vertices v_1 and v_2 at which the beacons are placed: $e = \{v_1, v_2\}$.

(a) Looking at Figure 5.5(a) we know from Lemma 5.3(iii) that both of the sets $\{\sigma_1, \sigma_2, \sigma_3, \sigma_6\}$ and $\{\sigma_6, \sigma_3, \sigma_4, \sigma_5\}$ each share one vertex v_1 and v_2, respectively. We place one beacon at v_1 and one beacon at v_2. If $v_1 = v_2$ we arbitrarily place the second beacon at one of the three other vertices of σ_6.

If σ_6 has a parent tetrahedron the shared facet contains three vertices of σ_6 and thus at least one of the two beacons. Thus, we can remove all tetrahedra σ_1 to σ_6 after placing the two beacons. By placing $k = 2$ beacons we can remove $6 = 3k$ tetrahedra.

(b) In Figure 5.5(b) we have the same situation as before except for the additional σ_7. We place the $k = 2$ beacons as in case (a) and observe that σ_6 contains two beacons. With the same argument as for σ_6's parent, σ_7 contains at least one beacon.

Hence, we can remove all tetrahedra σ_1 to σ_7. By placing $k = 2$ beacons we can remove $7 > 3k$ tetrahedra.

(c) For Figure 5.5(c) we apply the same argument as for case (b) but remove all tetrahedra σ_1 to σ_8. By placing $k = 2$ beacons we can remove $8 > 3k$ tetrahedra.

(d) For Figure 5.5(d) we first see with Lemma 5.3(ii) that the set $\{\sigma_6, \sigma_7, \sigma_8\}$ shares an edge e. After applying Lemma 5.4 to σ_1 to σ_6 we have two cases to consider.

In the first case σ_1 to σ_5 share a vertex v. As it is a vertex of σ_3 (and σ_3 is neighbor of σ_6) three of its neighboring vertices are vertices of σ_6. The edge e then contains at least one of those three neighbors (call it w). We place one beacon at v and one at w. Since they are neighbors they share the edge vw.

In the second case we have a vertex v and an edge e', both inside σ_6 and disjoint. This covers three vertices of σ_6 and therefore the edge e shares at least one vertex with v or e'. We can now choose two vertices of σ_6 for the beacons such that v, e', and e each contain at least one. Since they are both vertices of σ_6 they are neighbors.

In both cases we have placed $k = 2$ beacons. We then remove σ_1 to σ_5, σ_7, and σ_8 which are $7 > 3k$ tetrahedra.

(e) When Lemma 5.3(ii) is applied to the situation in Figure 5.5(e) we see that $\{\sigma_6, \sigma_7, \sigma_8\}$ and $\{\sigma_6, \sigma_9, \sigma_{10}\}$ share edges e_1 and e_2, respectively. We also apply Lemma 5.4 to σ_1 to σ_6 which leaves us again with two cases.

In the first case σ_1 to σ_5 share a vertex v. We choose v and two additional vertices of σ_6 as beacons such that both edges e_1 and e_2 contain at least one beacon. With the same argument as in case (d) v is neighbor to at least one of the other beacons and the other two beacons are neighbors because they are both vertices of σ_6.

In the second case we have a vertex v and an edge e, both inside σ_6 and disjoint. We choose v and two additional vertices of σ_6 as beacons such that all edges e, e_1, and e_2 contain at least one beacon. Obviously, all three beacons are neighbors.

In all cases we place $k = 3$ beacons such that every tetrahedron contains at least one. We then remove all tetrahedra but σ_6, which removes $9 = 3k$ tetrahedra.

(f) For Figure 5.5(f) we apply Lemma 5.4 independently to σ_1 to σ_6 and σ_6 to σ_{11}.

In the first case both σ_1 to σ_5 and σ_7 to σ_{11} share a vertex, which we call v_1 and v_2. With the same argument as in case (d) three neighboring vertices of both v_1 and v_2 are vertices of σ_6. Therefore, at least one vertex of σ_6 is neighbor to both; we call it v. We then place three beacons at v, v_1, and v_2.

In the second case, without loss of generality, σ_1 to σ_5 share a vertex v_1 and σ_6 to σ_{11} have a vertex v_2 and an edge e in σ_6, with $v_2 \cap e = \emptyset$. Then at least one of the three vertices of σ_6 which are neighbors of v_1 is covered by v_2 or e. We place one beacon at v_1, at v_2, and at one of the vertices of e.

In the third case, σ_1 to σ_6 have a vertex v_1 and an edge e_1 in σ_6 and σ_6 to σ_{11} have a vertex v_2 and an edge e_2 in σ_6. We place three beacons at vertices of σ_6 such that v_1, v_2, e_1, and e_2 contain at least one beacon.

In all cases we place $k = 3$ beacons such that every tetrahedron contains at least one. As before, every beacon has at least one other beacon as a neighbor. We then remove all tetrahedra but σ_6, which removes $10 > 3k$ tetrahedra.

(g) The situation in Figure 5.5(g) is similar to case (f) for Figure 5.5(f). Thus we only describe where we need to take some additional care.

In the first case, v can be placed at two vertices. Here we chose the vertex which is contained in σ_{12}. This is always possible since σ_{12} contains three of the four vertices of σ_6

In the the second case, we do not place the beacon at an arbitrary vertex of edge e but at the vertex which is contained in σ_{12}. The same argument as before applies.

In the third case there is at least one beacon contained in σ_{12} because all three beacons are placed at vertices of σ_6.

By placing $k = 3$ beacons we can then remove $11 > 3k$ tetrahedra: all but σ_6.

(h) The situation in Figure 5.5(h) can be handled similar to the situation in Figure 5.5(f) and thus we first apply case (f). By Lemma 5.3(ii) $\{\sigma_6, \sigma_{12}, \sigma_{13}\}$ shares an edge e'. If e' is not covered by one of the three beacons placed by the rules of case (f), we place one additional beacon at one of the vertices of e'.

By placing either $k = 3$ or $k = 4$ beacons we remove $12 \geq 3k$ tetrahedra: all but σ_6.

(i) For Figure 5.5(i) let $S_1 = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6\}$, $S_2 = \{\sigma_6, \sigma_7, \sigma_8, \sigma_9, \sigma_{10}, \sigma_{11}\}$, and $S_3 = \{\sigma_6, \sigma_{12}, \sigma_{13}, \sigma_{14}, \sigma_{15}, \sigma_{16}\}$. Furthermore let $S_1' = S_1 \setminus \{\sigma_6\}$, $S_2' = S_2 \setminus \{\sigma_6\}$, and $S_3' = S_3 \setminus \{\sigma_6\}$. We then apply Lemma 5.4 independently to S_1, S_2, and S_3.

In the first case, S_1', S_2', and S_3' each share a vertex, which we call v_1, v_2, and v_3. We place beacons at those three vertices. Each such vertex has three neighbors which are vertices of σ_6 with the argument of case (d). Thus, they all have one common neighbor vertex in σ_6 at which we place a fourth beacon.

Without loss of generality, in the second case, S_1' and S_2' each share a common vertex v_1 and v_2 and S_3 has a vertex v_3 and an edge e_3, disjoint and in σ_6. We choose v_1, v_2, and v_3 and one of the vertices of e as beacon locations. The latter two are vertices of σ_6 and thus v_1 and v_2 have a neighboring beacon in σ_6.

In the third case, without loss of generality, S_1' shares a common vertex v_1 and S_2 and S_3 each have a vertex v_2 and v_3 and an edge e_2 and e_3, all four in σ_6. We place a beacon at v_1 and three beacons at vertices of σ_6 such that v_2, v_3, and both edges e_1 and e_2 contain at least one beacon. Since three neighboring vertices of v_1 are vertices of σ_6 the beacon at v_1 has at least one beacon neighbor in σ_6.

In the fourth case, all S_1, S_2, and S_3 each have one vertex and one edge in σ_6. We place one beacon at the resulting vertex of each set. This also covers all edges. Hence, we only need three beacons for all tetrahedra.

In all cases, we place $k \leq 4$ beacons to remove $15 > 3k$ tetrahedra (all but σ_6). □

5.2.3 Conclusion

We can now restate Hypothesis 5.5 as a theorem:

Theorem 5.10 (Upper bound). *Given a polytope P with a tetrahedral decomposition Σ with $m = |\Sigma|$ tetrahedra. Then it is always sufficient to place $\left\lfloor \frac{m+1}{3} \right\rfloor$ beacons to route between any pair of points in P.*

Proof. We show this by induction. The base case is shown by Lemma 5.7. We assume that the induction hypothesis (Hypothesis 5.5) holds for all polytopes with a tetrahedral decomposition with strictly less than m tetrahedra. We then show that it also holds for tetrahedral decompositions Σ with exactly m tetrahedra.

Look at the spanning tree T of the dual graph $D(\Sigma)$ of the tetrahedral decomposition Σ which is rooted at an arbitrary leaf node. Let σ_1 be a deepest leaf node and if σ_1 is not unique choose the one with the largest number of siblings, breaking ties arbitrary. By the brute force argument of the program in Appendix B σ_1 is in one of the configurations in Figures 5.4 and 5.5. In both cases we can apply Lemmas 5.8 and 5.9, respectively. We then know at least the following:

(i) We have placed $k \geq 1$ beacons and removed at least $3k$ tetrahedra.

(ii) Every removed tetrahedron contains at least one beacon.

(iii) Every placed beacon has at least one of the other beacons as a direct neighbor and all beacons are connected by edges.

(iv) There is at least one beacon b contained in the remaining polytope P'.

From (i) it follows that the new polytope P' has a tetrahedral decomposition of $m' \leq m - 3k$ tetrahedra. We can then apply the induction hypothesis for P'. Thus we only need to place $k' = \lfloor \frac{m'+1}{3} \rfloor \leq \lfloor \frac{m-3k+1}{3} \rfloor = \lfloor \frac{m+1}{3} \rfloor - k$ beacons in P' to route between any pair of points in P'. Since $k' + k = \lfloor \frac{m+1}{3} \rfloor$ we never place more beacons than we are allowed.

From the induction hypothesis and (iv) we conclude that we are especially able to route from any point in P' to the beacon b and vice versa, since b is contained in P'. With (ii) we know that for every point p in the removed tetrahedra there is a beacon b' such that p attracts b' and b' attracts p. Finally, with (iii) we know that we can route between all beacons we have placed. This especially means that we can route from every beacon to the beacon b which is inside P' and vice versa.

This completes the inductive step and thus, by induction, we have proved the theorem. □

5.3 The lower bound

We now want to show a lower bound for the number of beacons needed to route within polytopes with a tetrahedral decomposition. For this we use a construction very similar to the one used in Section 3.2 which was defined in Definition 3.3.

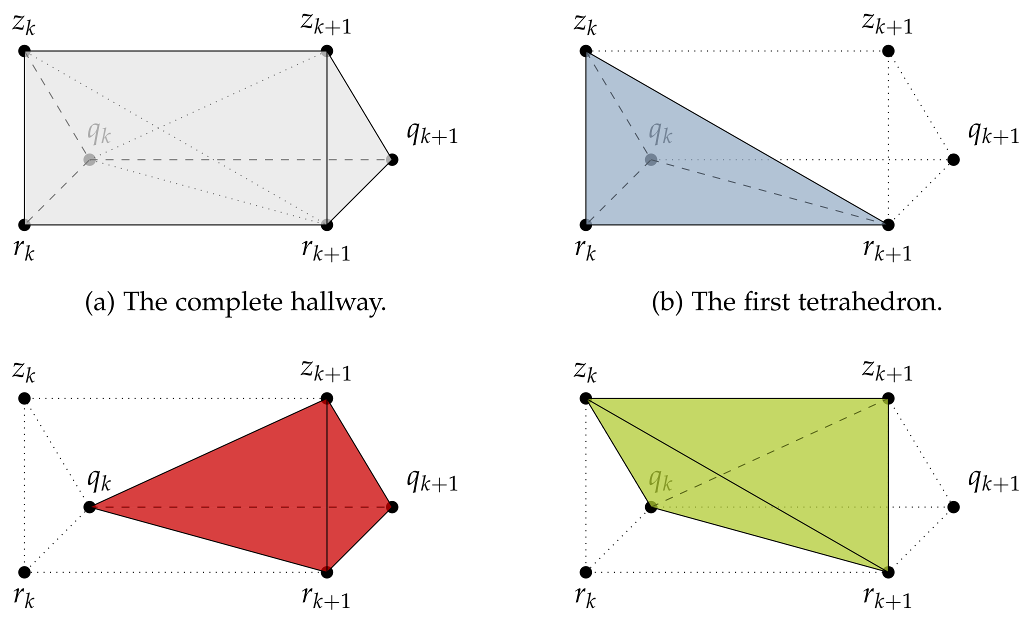
Definition 5.11. For every $c \in \mathbb{N}^{\geq 1}$ and some small $0 < \delta < 1$ a *c-corner spiral polytope* is a polytope with $n = 3c + 2$ vertices. These vertices are s and t as well as q_k, r_k, and z_k for all $1 \leq k \leq c$. The coordinates of s, t, q_k, and r_k are the same as in Definition 3.3 with their z-coordinate set to 0. The z_k are positioned above r_k or more formally $z_k := r_k + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ for all $1 \leq k \leq c$.

The edges and facets of the polytope are given by the tetrahedral decomposition:

- The start and end tetrahedra are formed by $\triangle r_1 q_1 z_1 s$ and $\triangle r_c q_c z_c t$.
- The *hallway* between two triangles $\triangle r_k q_k z_k$ and $\triangle r_{k+1} q_{k+1} z_{k+1}$ consists of the three tetrahedra $\triangle r_k q_k z_k r_{k+1}$, $\triangle r_{k+1} q_{k+1} z_{k+1} q_k$, and $\triangle q_k z_k r_{k+1} z_{k+1}$.

The three vertices r_k, q_k, and z_k form the k-th *corner*.

Figure 5.6 shows how the hallway is constructed out of the three described tetrahedra.



(a) The complete hallway. (b) The first tetrahedron.

(c) The second tetrahedron. (d) The third tetrahedron.

Figure 5.6: The three-dimensional hallway (a) consists of three tetrahedra (b) to (d). Thus three vertices are needed for one additional hallway.

Observation 5.12. The smallest c-corner spiral polytope with $c = 1$ consists of exactly two tetrahedra. For greater c we add exactly $c - 1$ hallways, each consisting of three tetrahedra. This means that a c-corner spiral polytope has a tetrahedral decomposition with $m = 3c - 1$ tetrahedra. It follows from Definition 5.11 that the number of tetrahedra relative to the number of vertices is $m = 3 \cdot \frac{n-2}{3} - 1 = n - 3$.

We will now show that this class of polytopes is a worst-case example for the number of beacons needed to route between a specific pair of points.

Lemma 5.13 (Lower bound)**.** *Given a c-corner spiral polytope P. Then c beacons are necessary to route from s to t.*

Proof. We project P onto the xy-plane which results in a c-corner spiral polygon P' due to the construction of the c-corner spiral polytope. To P' we can apply Lemma 3.4 where we showed that c beacons are sometimes necessary to route in P'. In the proof, we showed that a beacon is needed in an area around each of the c corners.

As opposed to the polygon, the movement in the polytope is not constrained to the xy-plane. Additionally, beacons can be placed at locations which do not lie in the xy-plane. We need to show that this does not change the situation in a way so that less than c beacons are necessary.

The argument here is similar to the one in our proof of Theorem 4.7. First, note that, due to the construction in Definition 5.11, every cross section of the polytope parallel to the xy-plane yields a c-corner spiral polygon with different widths δ. For every such cross section, Lemma 3.4 tells us that to route only in this cross section, c beacons are needed.

Additionally, the hallway's inner boundary $r_k z_k r_{k+1} z_{k+1}$ is perpendicular to the xy-plane. This means that the movement of all points p which are attracted by a beacon b can be split into a xy-movement and a z-movement because the z-coordinate is not important for any movement along the inner boundary. Since each hallway is convex there is no other movement of a point p attracted by a beacon b which is constrained by the polytope's boundary ∂P. We can then only look at the xy-movement which again yields a two-dimensional situation to which Lemma 3.4 can be applied. □

5.4 The sharp bound

Theorem 5.14. *Given a polytope P for which a tetrahedral decomposition with m tetrahedra exists. Then it is always sufficient and sometimes necessary to place $\lfloor \frac{m+1}{3} \rfloor$ beacons to route between any pair of points in P.*

Proof. In Theorem 5.10 we have shown that $\lfloor \frac{m+1}{3} \rfloor$ is an upper bound.

We now show that for a given m we can always construct a polytope which needs exactly $\lfloor \frac{m+1}{3} \rfloor$ beacons. With $c = \lfloor \frac{m+1}{3} \rfloor$ we construct a c-corner spiral polytope which, by Observation 5.12, consists of $m' = 3c - 1 \leq m$ tetrahedra and for which, by Lemma 5.13, c beacons are sometimes necessary. If $m' < m$ we need to add one or two additional tetrahedra such that we obtain a polytope which consists of m tetrahedra. We do this by adding one or two tetrahedra to the tetrahedron which contains t, the same way the hallway of the spiral polytope is constructed. We first add the tetrahedron in Figure 5.6(d) and then (optionally) the tetrahedron in Figure 5.6(c). This way the obtained polytope is always connected by shared facets, as required by our definition.

We have shown how to construct a polytope with a tetrahedral decomposition of m tetrahedra for which $c = \lfloor \frac{m+1}{3} \rfloor$ beacons are sometimes necessary to route between a specific pair of points. This matches the upper bound and thus completes the proof. □

Chapter 6

General polytopes

As we have already seen in Chapter 5, there exist polytopes which cannot be triangulated easily, if at all. Nevertheless, when additional so-called Steiner points are allowed to be placed, a tetrahedral decomposition can be obtained for every polytope [2].

6.1 One beacon at each vertex does not cover the polytope

In his book *Art Gallery Theorems and Algorithms* O'Rourke [15, pp. 255f.] presents a class of polytopes for three-dimensional art gallery problems which, among others, fulfill the following property: Placing a guard at every vertex does not cover the whole interior of the polytope.

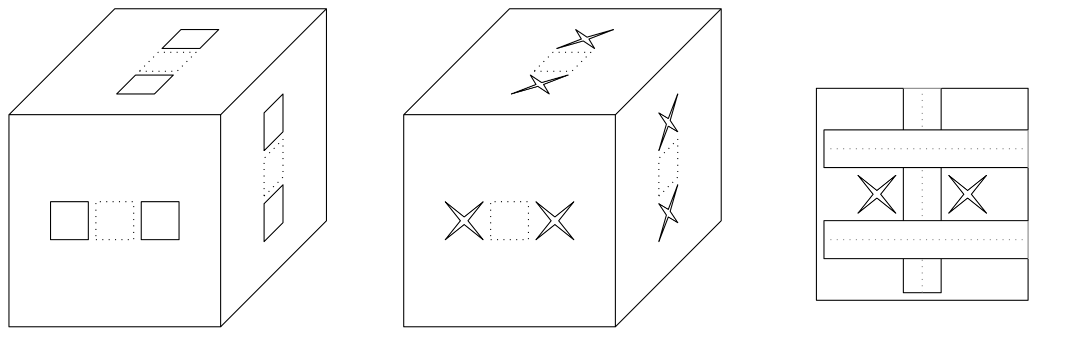
The example is constructed with visibility and not attraction in mind. In fact, placing a beacon at every vertex of the described polytope is indeed sufficient to attract any point and route between any pair of points. However, a modification of the construction yields the following

Lemma 6.1. *There exists a polytope P for which it is not sufficient to place a beacon at every vertex of P to attract any point $p \in P$.*

Proof. We construct a polytope similar to the one presented by O'Rourke, starting with an axis-aligned cube C of side length $L > 5$. We then locate an axis-aligned unit cube in the center of C, i.e., both centers of mass are located in the same point. We now project the unit cube onto the front, top, and right facet of C.

For every of these three facets we take two additional unit squares, place them on the projection and then translate them by $1 + \varepsilon$, for some $0 < \varepsilon \ll 1$, along one axis, such that we obtain the situation depicted by Figure 6.1(a). Here the unit cube's projections are marked with a dotted line while the additional squares are solid.

The solid squares are now distorted in the following fashion: Place a vertex at the center of each edge and move it $\frac{1}{2} - \delta$ units towards the square's center, for some $0 < \delta \ll 1$. This results in star-shaped polygons, as seen in Figure 6.1(b).



(a) The six solid unit squares do not touch when pushed into the interior.

(b) The squares are distorted in a way such that we obtain star-shaped polygons.

(c) A frontal cross section through the cube's *center*.

Figure 6.1: The cube C with side length L. The dotted unit squares are projections of a unit cube located in C's center. The solid shapes in (a) and (b) are pushed into the interior until they nearly touch the opposite facet. This can also be seen in the cross section (c).

We choose some $0 < \sigma \ll 1$ and then push those polygons into the interior of C by $L - \sigma$ units, resulting in star prisms that nearly touch the opposite facet. The cross section in Figure 6.1(c) shows the result. In Figure 6.2 we can see how all the parameters play together.

We can now place a point c in the center of C. On all six sides c is confined by the six star prisms. We first show that c cannot be attracted by the vertices of those prisms. Afterwards we show that the corner vertices of C can also not attract c.

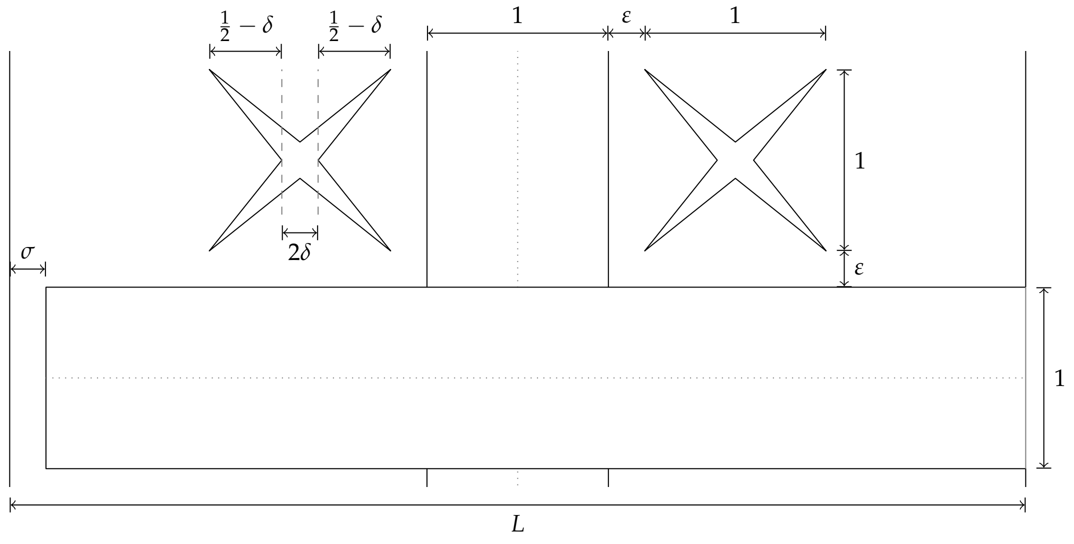
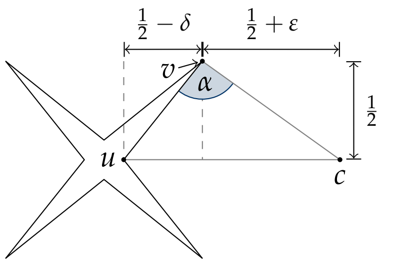
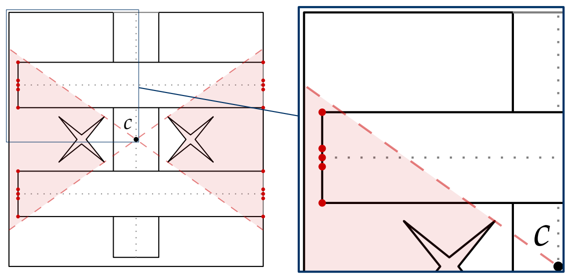


Figure 6.2: A part of Figure 6.1(c) to show the parameters L, ε, δ, and σ.



(a) Parameters ε and δ have to be chosen such that $\alpha > 90°$ holds.



(b) All points in the shaded area, including the red ones, cannot attract the point c.

Figure 6.3: If we choose ε and δ such that $\alpha > 90°$ (see (a)) no point invisible to c can attract it. With this we can prevent the vertices of the star prisms (whose projections are marked in red in (b)) from attracting c.

Star prism vertices cannot attract c. The cross section in Figure 6.3(b) shows in red the area in which points are not visible from c. Our goal is to show that this is also the area of the points that cannot attract c. To fulfill this goal we first look at c together with one of the star prisms in Figure 6.3(a). In the following, we will use the notation of this figure for the points v and u which are the top right and middle right points, respectively, of the left star polygon. Whenever we refer to the angle α we also mean the angle from this figure.

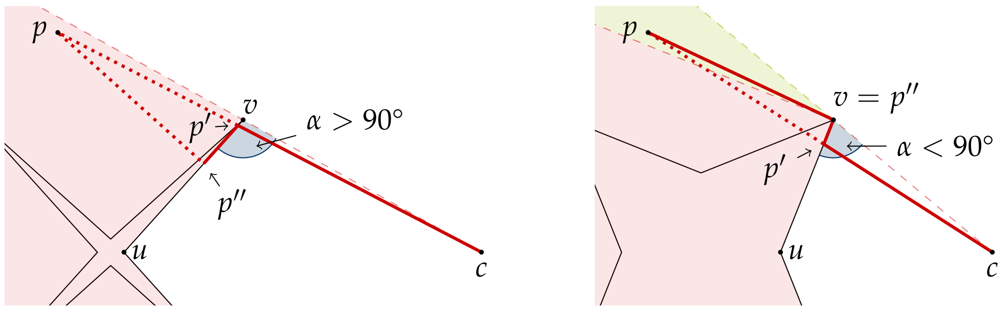
We will concentrate on the points to the right of the line cu; due to symmetry the same argument applies for points to the left of this line. Our claim is that if the angle α from Figure 6.3(a) is strictly smaller than 90°, then no point invisible from c is able to attract c.

We choose an arbitrary point p that lies to the left of the line trough cv and to the right or on the line trough cu. Due to this constraint the line segment pc has to intersect the edge uv at some point $p' \in uv$. Additionally, the shortest line segment from p to uv hits uv at some point $p'' \in uv$.

The movement of c under the influence of p is the following: First c moves in a straight line until it reaches point p'. It will then slide along the edge uv until reaching p''. At this point there are two possibilities which are both depicted in Figure 6.4: If $p'' \neq v$ we are either stuck on the edge uv or at the vertex u, in both cases p is unable to attract c further. See Figure 6.4(a) for an example of this situation. If, however, $p'' = v$ our point will continue moving towards p. There is an example of the movement in Figure 6.4(b).

To prevent the latter, we have to make sure that the shortest line segments to uv of all points not visible from c do not end in the vertex v. This is the case if the angle α is strictly larger than 90°. Then, for any observed point p, the angle $\sphericalangle pvu$ is strictly less than 90° and the shortest lines segment to uv does not terminate at v. We only have to check whether it is possible to chose ε and δ as needed.

We look at Figure 6.3(a) and notice that the triangle $\triangle cuv$ can be decomposed into two right triangles along the dashed line. Each triangle contains a part of α which



(a) Here ε is large, which moves c to the right, and δ is small, which moves u to the left. Both movements increase the angle α.

(b) Here ε is small, moving c to the left, and δ is big, moving u to the right. Both movements decrease α.

Figure 6.4: Two different configurations of the parameters ε and δ, as depicted in Figure 6.3(a). In both figures p is the same point which is not visible by c. We can see that in (a) p cannot attract c since the shortest path from p to uv terminates on the edge. In (b) p can attract c since the shortest path to uv terminates in v.

we call α_1 and α_2 and for which $\alpha = \alpha_1 + \alpha_2$ holds. Since we know the lengths of the catheti of both triangles we can compute α depending on ε and δ:

$$\begin{aligned}\alpha = \alpha_1 + \alpha_2 &= \arctan\left(\frac{\frac{1}{2}+\varepsilon}{\frac{1}{2}}\right) + \arctan\left(\frac{\frac{1}{2}-\delta}{\frac{1}{2}}\right)\\ &= \arctan(1+2\varepsilon) + \arctan(1-2\delta).\end{aligned}$$

Since we want α to be strictly larger than 90° it follows that

$$90° < \arctan(1+2\varepsilon) + \arctan(1-2\delta) \tag{6.1}$$

must hold true. For any right triangle with catheti of lengths $x > 0$ and $y > 0$ we know that

$$90° = \arctan\left(\frac{y}{x}\right) + \arctan\left(\frac{x}{y}\right)$$

holds, because the two non-right angles sum up to exactly 90°. Additionally, arctan is monotonically increasing and thus, for some $\mu > 0$, we can also write

$$90° < \arctan\left(\frac{y}{x}\right) + \arctan\left(\frac{x}{y} + \mu\right). \tag{6.2}$$

If we find values for ε and δ such that they fulfill the requirement

$$\frac{y}{x} = 1 + 2\varepsilon \text{ and } \frac{x}{y} + \mu = 1 - 2\delta$$

for all $x > 0$ and $y > 0$ we can conclude that Equation (6.1) must be true for those values since Equation (6.2) holds. With

$$\begin{aligned} & \frac{x}{y} < \frac{x}{y} + \mu \\ \Leftrightarrow \quad & \frac{1}{1+2\varepsilon} < 1 - 2\delta \\ \Leftrightarrow \quad & 2\delta < 1 - \frac{1}{1+2\varepsilon} \\ \Leftrightarrow \quad & \delta < \frac{1}{2}\left(\frac{1+2\varepsilon}{1+2\varepsilon} - \frac{1}{1+2\varepsilon}\right) \\ \Leftrightarrow \quad & \delta < \frac{\varepsilon}{1+2\varepsilon} \end{aligned} \tag{6.3}$$

we see that the latter inequality needs to hold in order for α to be strictly larger than $90°$. This is always possible since ε is positive. When choosing ε and δ according to Equation (6.3) no point invisible from c can attract c.

In order to make the vertices of the star prisms (marked red in Figure 6.3(b)) invisible from c we have two options. We either choose a small value for ε (and thereby δ), which increases the angle of the area of invisibility. We can also choose a large value for L, the side length of C. By doing so we can finally obtain the situation as depicted in Figure 6.3(b).

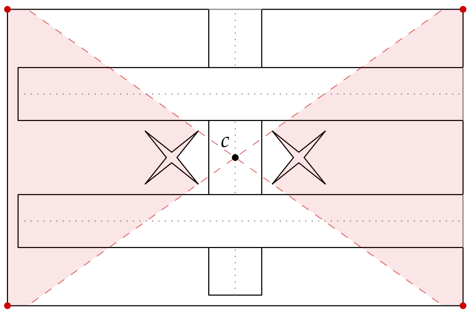
With this, we have shown that no point, which lies in the same xz-plane as c and is not visible from c, can attract c. Unfortunately, most star prism vertices w do not lie in the same plane as c, that is, $y_w \neq y_c$. However, since the star prism covers all y-coordinates between y_w and y_c and since there are no obstacles which can inhibit a y-movement from y_c to y_w, we can split the movement of c under the influence of w into the xz- and y-movement. Since the y-movement is monotonic and the star prism covers all corresponding y-coordinates we can safely ignore it. We are then left with the xz-movement which we have just analyzed.

Due to symmetry the same argument holds for the top and lateral cross sections through C's center. The only vertices remaining are the eight corners of C.

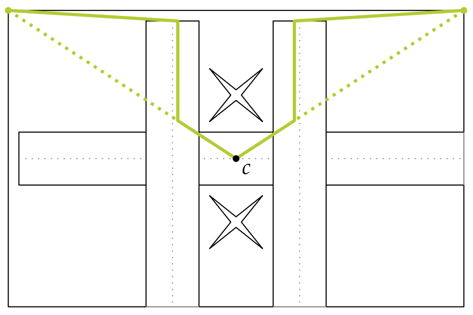
Corner vertices cannot attract c. Due to the symmetric construction and the placement of c in the exact center of C, c is able to see all eight vertices of the outer cube. This means, of course, that they can also see and hence attract c.

However, by enlarging the length of one side of C we can hide the vertices behind a star prism the same way as done before. A cross section of the resulting polytope is shown in Figure 6.5(a). This may look as if we already succeeded.

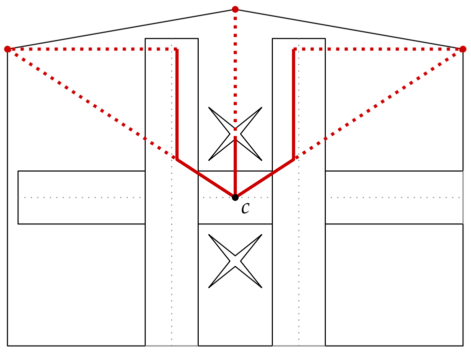
While looking at a cross section of C from the top, as in Figure 6.5(b), we notice that the four corners of the side which is not touched by the star prisms can still attract c. First c will move straight until it hits a star polygon. It will then slide upwards until it reaches the end of the star prism. There it is able to see and be attracted by the corner vertex.



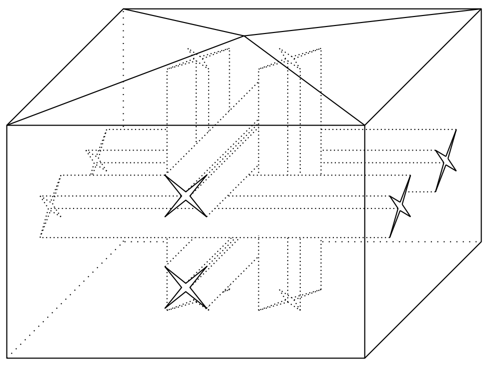
(a) The *frontal* cross section after stretching C to the left and right. If all eight corner vertices of C were in the same plane as c they could not attract c.



(b) A cross section *while looking at the top*. Since the star prisms do not touch both the lower and upper facet, the upper vertices of the cube can still attract c.



(c) We add a *roof* to prevent the attraction visible in (b). The roof node itself cannot attract c either.



(d) The complete three-dimensional construction. No vertex can now attract the point in its center. The box was rotated once towards the reader.

Figure 6.5: By (a) enlarging one side of C and (c) adding a roof which enables us to push one pair of prisms more into C we obtain the result (d) which inhibits the attraction of c by corner vertices, as in (b).

The solution to this problem could either consist in pushing the star prisms even further, eventually leading to a polytope with holes. Since we want to show the result for polytopes without holes, we opt for a second solution. It consists of pulling one facet of C outwards at its center point which constructs a *roof*. We can then push the star prisms further into C such that they are higher than the corner vertices, without touching the roof. The cross section in Figure 6.5(c) shows that then neither the newly added roof point nor the corner vertices can attract c.

The complete construction is shown in Figure 6.5(d). It prevents any of its vertices to attract c and thus completes the proof. □

With the result of Corollary 2.31 it follows that placing a beacon at every vertex is also not sufficient to route between any pair of points in the constructed polytope.

Chapter 7 Conclusion and open problems

Starting with the two-dimensional foundation laid out by Biro [3] and Biro et al. [4, 5, 6] we have transferred the idea of beacon-based routing and coverage from the two-dimensional polygonal domain to the three-dimensional polytopal one. We have shown that several problems (in particular the 3D-All-Pair Problem, the 3D-All-Sink Problem, the 3D-All-Source Problem and the 3D-Coverage Problem as defined in Chapter 4) are NP-hard and APX-hard by reducing to the respective two-dimensional problems.

Nevertheless, we have shown a tight worst-case bound of $\lfloor (m+1)/3 \rfloor$ for the number of beacons needed to route within polytopes with a tetrahedral decomposition of m tetrahedra. This bound depends on the number of tetrahedra because the number of tetrahedra needed to decompose a polytope can vary widely. Relative to the number of vertices n this bound can be as low as $\lfloor (n-2)/3 \rfloor$ but never bigger than $\mathcal{O}(n^2)$.

Since every polytope can be decomposed into $\mathcal{O}(n^2)$ tetrahedra if additional Steiner points are allowed (see Section 5.1) the latter bound is also an upper bound for general polytopes. For general polytopes we have also shown that it is not sufficient to place a beacon at every vertex of the polytope to cover it.

7.1 Open problems

In his PhD thesis Biro [3] looked at various subtopics in the beacon model which were not covered in this work but might be interesting to look at. These subtopics are

- *attraction regions:* computing all points which are either attracted by a specific beacon or by beacons in a specific subset of the polytope,
- *inverse attraction regions:* all points that can attract a specific point or a specific subset of the polytope, and
- *beacon kernels:* the set of all points in a polytope such that a beacon at any such point covers P completely.

There are also some more specific problems which are interesting to look at. We had a look at some of the problems but either decided that it would be out of scope or we tried to tackle the problem with different attempts but were not successful.

Open question 7.1. In Section 5.3 we have shown a lower bound of $\Omega(n)$ for the number of beacons necessary to route within polytopes with a tetrahedral decomposition. Can a larger lower bound be shown for the number of beacons needed to cover or route within general polytopes?

The construction by O'Rourke [15], which we used in Section 6.1, is actually a class of polytopes for arbitrary large numbers of vertices n. The construction leads to a lower bound of $\Omega(n^{3/2})$ for the number of guards necessary to cover the polytope. We tried to modify the construction to suit the needs of beacon-based guarding but did not succeed.

Open question 7.2. What is the complexity of finding a shortest beacon path between two given points, i.e., the complexity of the 3D-Given-Pair Problem?

This question was briefly mentioned at the end of Chapter 4. For two dimensions Biro [3] first presents a 2-approximation with the following idea:

Triangulate the polygon and construct a directed graph whose vertices are the triangles and whose edges are given as follows. For every triangle σ calculate its inverse attraction region $\mathrm{IA}(\sigma)$, i.e., all points which can attract at least one point in the triangle. For every other triangle τ add an directed edge from τ to σ in the graph if $\tau \cap \mathrm{IA}(\sigma) \neq \emptyset$, that is, there is a point $b \in \tau$ and a point $p \in \sigma$ such that b attracts p. Then a shortest path in the directed graph from the triangle which contains t to the triangle containing s is found. For every triangle in the shortest path two beacons are placed, one that attracts a point in the previous triangle and one that is attracted by a beacon in the next one. He then shows that the number of beacons needed is always as high as the number of triangles in the shortest path, yielding a 2-approximation.

By calculating not only the attraction region of triangles but the attraction region of the attraction region iteratively the algorithms becomes more precise which results in a PTAS.

The idea of the exact algorithm is quite obvious: Given s, t calculate the inverse attraction region $A_1 = \mathrm{IA}(s)$. Iteratively compute $A_{i+1} = \mathrm{IA}(A_i)$ until $t \in A_k$ for some $k \in \mathbb{N}$. Then place a beacon b_k at t and choose an arbitrary point of IA_{k-1} which is attracted by b_k. Place the beacon b_{k-1} at this point and iterate until b_1 is placed.

The regions A_i are represented as polygons and for their size (the number of vertices) $m_i = |A_i|$ the best known bound is quadratic in the input region and the size of the polygon: $m_i \in \mathcal{O}(m_{i-1}^2 n^2)$ with $n = |P|$. Thus, the running time of the algorithm could be exponential in the number of steps k and since sometimes n beacons are needed it could be exponential in the size of the polygon.

Appendix A

Bibliography

[1] Sang Won Bae, Chan-Su Shin, and Antoine Vigneron. “Improved Bounds for Beacon-Based Coverage and Routing in Simple Rectilinear Polygons”. In: *arXiv preprint arXiv:1505.05106* (2015). URL: `http://arxiv.org/abs/1505.05106` (visited on 07/14/2016) (cit. on pp. 2, 3).

[2] Marshall Bern and David Eppstein. “Mesh Generation and Optimal Triangulation”. In: *Computing in Euclidean geometry* 4 (1995), pp. 47–123. URL: `http://www.wias-berlin.de/people/si/course/files/BernEpp92-OptimalMeshGen.pdf` (visited on 01/31/2017) (cit. on pp. 6, 27, 41).

[3] Michael Biro. “Beacon-Based Routing and Guarding”. PhD thesis. State University of New York at Stony Brook, 2013. URL: `http://gradworks.umi.com/35/68/3568472.html` (visited on 07/14/2016) (cit. on pp. iii, 1–3, 7, 9, 10, 21, 22, 24, 25, 27, 47, 48).

[4] Michael Biro, Jie Gao, Justin Iwerks, Irina Kostitsyna, and Joseph S. B. Mitchell. “Beacon-Based Routing and Coverage”. In: *21st Fall Workshop on Computational Geometry (FWCG 2011)*. 2011. URL: `https://www.researchgate.net/profile/Jie_Gao10/publication/268030595_Beacon_based_routing_and_coverage/links/54d393f90cf25017918254ad.pdf` (visited on 06/08/2016) (cit. on pp. iii, 1–3, 13, 30, 47).

[5] Michael Biro, Jie Gao, Justin Iwerks, Irina Kostitsyna, and Joseph S. B. Mitchell. “Combinatorics of Beacon-Based Routing and Coverage”. In: *Proceedings of the 25th Canadian Conference on Computational Geometry (CCCG 2013)*. Vol. 1. 2013, p. 3. URL: `http://cccg.ca/proceedings/2013/papers/paper_74.pdf` (visited on 06/08/2016) (cit. on pp. 2, 13, 14, 16, 17, 47).

[6] Michael Biro, Justin Iwerks, Irina Kostitsyna, and Joseph S. B. Mitchell. “Beacon-Based Algorithms for Geometric Routing”. In: *Algorithms and Data Structures*. Springer, 2013, pp. 158–169. URL: `http://link.springer.com/chapter/10.1007/978-3-642-40104-6_14` (visited on 06/08/2016) (cit. on pp. 2, 47).

[7] Björn Brodén, Mikael Hammar, and Bengt J. Nilsson. "Guarding Lines and 2-Link Polygons Is APX-Hard". In: *Proceedings of the 13th Canadian Conference on Computational Geometry (CCCG 2001)*. 2001, pp. 45–48. URL: http://www.cccg.ca/proceedings/2001/mikael-2351.ps.gz (visited on 02/27/2017) (cit. on p. 22).

[8] Bernard Chazelle. "Convex Partitions of Polyhedra: A Lower Bound and Worst-Case Optimal Algorithm". In: *SIAM Journal on Computing* 13.3 (1984), pp. 488–507. URL: http://epubs.siam.org/doi/abs/10.1137/0213031 (visited on 11/07/2016) (cit. on p. 27).

[9] V. Chvátal. "A Combinatorial Theorem in Plane Geometry". In: *Journal of Combinatorial Theory, Series B* 18.1 (1975), pp. 39–41. DOI: 10.1016/0095-8956(75)90061-1 (cit. on p. 3).

[10] Steve Fisk. "A Short Proof of Chvátal's Watchman Theorem". In: *Journal of Combinatorial Theory, Series B* 24.3 (1978), p. 374. DOI: 10.1016/0095-8956(78)90059-X (cit. on p. 3).

[11] Carl W. Lee and Francisco Santos. "16: Subdivisions and Triangulations of Polytopes". In: *Handbook of Discrete and Computational Geometry*. Ed. by Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth. 3rd edition. 2017, pp. 415–447. URL: http://www.csun.edu/~ctoth/Handbook/chap16.pdf (visited on 12/15/2016). Pre-published (cit. on pp. 5, 6).

[12] N. J. Lennes. "Theorems on the Simple Finite Polygon and Polyhedron". In: *American Journal of Mathematics* 33 (1/4 Jan. 1911), p. 37. ISSN: 00029327. DOI: 10.2307/2369986 (cit. on p. 27).

[13] Nimrod Megiddo and Arie Tamir. "On the Complexity of Locating Linear Facilities in the Plane". In: *Operations research letters* 1.5 (1982), pp. 194–197. DOI: 10.1016/0167-6377(82)90039-6 (cit. on p. 22).

[14] T. S. Michael. *How to Guard an Art Gallery and Other Discrete Mathematical Adventures*. Baltimore: Johns Hopkins University Press, 2009. 257 pp. ISBN: 978-0-8018-9298-1 (cit. on p. 3).

[15] Joseph O'Rourke. *Art Gallery Theorems and Algorithms*. The international series of monographs on computer science 3. New York, NY: Oxford Univ. Press, 1987. 282 pp. ISBN: 978-0-19-503965-8 (cit. on pp. 3, 41, 48).

[16] Jim Ruppert and Raimund Seidel. "On the Difficulty of Triangulating Three-Dimensional Nonconvex Polyhedra". In: *Discrete & Computational Geometry* 7.3 (1992), pp. 227–253. DOI: 10.1007/BF02187840 (cit. on pp. 27, 28).

[17] Thomas C. Shermer. "A Combinatorial Bound for Beacon-Based Routing in Orthogonal Polygons". In: *arXiv preprint arXiv:1507.03509* (2015). URL: http://arxiv.org/abs/1507.03509 (visited on 07/14/2016) (cit. on pp. 2, 3, 17).

Appendix B

Program code

Here we include the program code used to generate all possible trees of maximum depth four used in section Section 5.2.2.

```python
#!/usr/bin/env python3
"""Generate all dual graph configurations we need to look at."""

from itertools import chain, product

from graphviz import Digraph


################################################################################
# Tree structure.
################################################################################
class Node:
    """A tree structure which allows pruning of unneeded subtrees."""

    # ==========================================================================
    # General tree structure.
    # ==========================================================================

    def __init__(self):
        """A new node is simply a leaf."""
        self.nodes = []

    def add(self, n=1):
        """Append n additional children and return self."""
        for _ in range(n):
            self.nodes.append(Node())
        return self

    def append(self, node):
        """Append a node or an iterable of nodes and return self."""
        try:
            for n in node:
                self.nodes.append(n)
        except TypeError:
            self.nodes.append(node)
        return self

    def is_leaf(self):
        """Return whether this node is a leaf, i.e., has no children."""
        return not self.nodes

    # ========================================================================
    # Graphviz.
    # ========================================================================

    def to_dot(self, graph=None, prefix=''):
        """Return a Graphviz representation of the tree."""
        if graph is None:
            graph = Digraph()
        self._dot_recursion(graph, 1, prefix)
        return graph

    def _dot_recursion(self, graph, current, prefix=''):
        """Recursively create Graphviz tree."""
        graph.node(prefix + str(current), label=str(current))
        this_number = current
        current = current + 1
        for child in self.nodes:
            current, child_number = child._dot_recursion(graph, current,
                                                         prefix)
            graph.edge(prefix + str(this_number), prefix + str(child_number))
        return current, this_number

    # ========================================================================
    # Pruning of "easy" cases.
    # ========================================================================

    def prune(self):
        """Remove subtrees that are easily removed."""
        # First try to remove subtrees.
        if self._prune() is None:
            return None

        # Call prune() for all children and filter out children that were.
        # removed
        self.nodes = list(filter(lambda x: x is not None,
                                 map(Node.prune, self.nodes)))

        # Sort children after pruning to have a canonical structure.
        self.nodes.sort()

        # Try pruning easy subtrees again. Maybe pruning the children created a
        # prunable configuration again.
        return self._prune()

    def _prune(self):
        """Remove subtrees that are easily removed."""
        if len(self.nodes) == 3:
            if all(n.is_leaf() for n in self.nodes):
                # Case (i): Figure 5.4(a): This is s2
                #   Three children that are leaf nodes: Remove all of them.
                self.nodes = []
            elif all(len(n.nodes) == 1 and n.nodes[0].is_leaf()
                     for n in self.nodes):
                # Case (iii)(3): Figure 5.4(e): This is s3
                #   Three children with one child leaf each: Remove two
                #                                            children.
                self.nodes.pop()
                self.nodes.pop()
        if len(self.nodes) == 2:
            if all(n.is_leaf() for n in self.nodes):
```

```python
                # Case (ii): Figure 5.4(b): This is s2
                #   Two children that are leaf nodes: Remove both including the
                #                                     parent node.
                return None
        if len(self.nodes) == 1:
            if len(self.nodes[0].nodes) == 1:
                if self.nodes[0].nodes[0].is_leaf():
                    # Case (iii)(1): Figure 5.4(c): This is s3
                    #   A chain of three nodes: Remove all of them.
                    return None
        if len(self.nodes) >= 2:
            leaves = [n for n in self.nodes if n.is_leaf()]
            leaves2 = [n for n in self.nodes if len(n.nodes) == 1 and
                       n.nodes[0].is_leaf()]
            if leaves and leaves2:
                # Case (iii)(2): Figure 5.4(d): This is s3
                #   One leaf child and one child with a single leaf child:
                #       Remove both children.
                self.nodes.remove(leaves[0])
                self.nodes.remove(leaves2[0])

        # Return self to indicate that the node itself is not to be removed.
        return self

    # ==========================================================================
    # Make trees comparable.
    # ==========================================================================

    def __eq__(self, other):
        """
        Compare equality of two nodes.

        Two nodes are equal if they have the same number of children and every
        child is equal to the respective child of the other node.
        """
        if other is None:
            return False
        if len(other.nodes) != len(self.nodes):
            return False
        for this, that in zip(self.nodes, other.nodes):
            if this != that:
                return False
        return True

    def __lt__(self, other):
        """
        Compare whether a node is smaller than another node.

        A node is smaller then another node if it has more direct children or
        if any of the children is smaller than the respective other child.
        """
        if len(self.nodes) != len(other.nodes):
            return len(self.nodes) > len(other.nodes)

        for this, that in zip(self.nodes, other.nodes):
            if this < that:
                return True
            if that < this:
                return False

        return True

    # ==========================================================================
    # String representation and hash value for uniqueness.
    # ==========================================================================

    def __str__(self):
        """Generate a bracket term representing the tree."""
        return '(' + ''.join(str(n) for n in self.nodes) + ')'

    def __repr__(self):
        """Terminal representation."""
        return str(self)

    def __hash__(self):
        """Hash value for uniqueness."""
        return hash(str(self))


################################################################################
# Generate all trees with certain maximum depth.
################################################################################
def all_trees(depth):
    """
    Yield all trees with a given maximum depth.

    The trees are created recursively by appending combinations of trees of
    depth-1 to a node.
    """
    if depth == 1:
        # Create a node with 0, 1, 2, and 3 children.
        for i in range(4):
            yield Node().add(i)
    else:
        # Append 0, 1, 2, or 3 children.
        for number_of_children in range(4):
            # Create as many iterators of the next lower depth as there should
            # be children appended.
            next_level_iterators = []
            for _ in range(number_of_children):
                next_level_iterators.append(all_trees(depth - 1))

            # Combine all possible combinations of the iterators and add them
            # to a new node.
            for subtrees in product(*next_level_iterators):
                yield Node().append(subtrees)


def iterator_len(iterator):
    """
    Return the number of elements in an iterator.

    The iterator is consumed by calling this function.
    """
    length = 0
    for _ in iterator:
        length += 1
    return length


################################################################################
# Main program.
################################################################################
if __name__ == '__main__':
    # The maximum depth of the tree is 3
    depth = 3
    number_of_combinations = iterator_len(all_trees(depth))
    # Start with the first tree
    current = 1

    # A container for all distinct non-prunable trees
    trees = set()

    # Iterate through all different trees of maximum depth
    for tree in all_trees(depth):
        # Prune "easy" cases
        tree = tree.prune()
        # Add tree to set of trees if it was not pruned completely
        if tree is not None:
            trees.add(tree)

        # Debug output
        print('\r{percent:.2f}% ({current} / {all}) - trees: {trees}'
              .format(percent=100 * current / number_of_combinations,
                      current=current,
                      all=number_of_combinations,
                      trees=len(trees)),
              end='', flush=True)
        current += 1

    # Sum up the number of trees
    print()
    print(len(trees), 'trees')

    # Create a document with all non-prunable trees
    g = Digraph()
    prefix = 1
    for tree in trees:
        tree.to_dot(g, str(prefix) + '_')
        prefix += 1
    g.render('trees')
```