Bachelor's Assignment: Algorithms for finding the Euclidian distance between a point and a polytope with application to spanning tree polytopes

Bozhidar Petrov
Supervisor: Dr. Matthias Walter

July 2, 2020

UNIVERSITY OF TWENTE.

Abstract

The problem of determining the distance between a point and a polytope is investigated. Two ways of solving it are presented: a specialized algorithm by P. Wolfe and a generic quadratic programming algorithm. Both of these are then applied to spanning tree polytopes. The results of experiments show that the specialized algorithm is better suited for this problem.

Contents

1 Introduction

In this report we will investigate the problem of finding the distance of a point to a polytope. In order to define a polytope, we need several other definitions.

Definition 1 *Given a set of points $P = \{P_1, ..., P_m\} \subseteq \mathbb{R}^n$ define the **affine hull** of P as*

$$\text{aff}(P) = \Big\{X : X = \sum_{j=1}^{m} P_j w_j, \sum_{j=1}^{m} w_j = 1\Big\},$$

*that is, the set of all affine combinations of points of P. The **convex hull** of P is*

$$\text{conv}(P) = \Big\{X : X = \sum_{j=1}^{m} P_j w_j, \sum_{j=1}^{m} w_j = 1, w \geq 0\Big\}$$

with $w \in \mathbb{R}^m$ and w_i being the i-th element of w.

A convex hull is simply an affine hull with non-negative coefficients of the combinations. A polytope can then be defined in the following way:

Definition 2 *A polytope is the convex hull of a given set of points $P = \{P_1, ..., P_m\} \subseteq \mathbb{R}^n$.*

Suppose we want to find the distance from a point Y to such a polytope. The "nearest point problem" can then be defined mathematically as

$$\min |X - Y|^2 \text{ s.t. } X = \sum_{k=1}^{m} P_k w_k, \sum_{k=1}^{m} w_k = 1, w \geq 0$$

The constraints describe a point in the polytope and the objective describes its distance to the target point, which we are trying to minimize. Any problem of this type can be translated into a problem of finding the point of minimal norm in a polytope. For example, given a set of points $P = \{P_1, ..., P_m\}$ and a target point Y, would give a solution X. But the same problem with a set of points $P' = \{P_1 - Y, ..., P_m - Y\}$ with the origin as a target point would give solution $X - Y$. Thus, for simplicity, Y will be assumed to be 0 from now on. The objective of (1) can then be redefined as:

$$\min |X|^2 = X^T X \text{ s.t. } X = \sum_{k=1}^{m} P_k w_k, \sum_{k=1}^{m} w_k = 1, w \geq 0 \qquad (1)$$

It can easily be seen that this problem is a quadratic program (QP): quadratic objective function with linear constraints. There are many algorithms for solving such programs. We will first look at a specialized algorithm by P. Wolfe [1] that can solve the nearest point problem, then we will compare it to general QP-solving algorithms.
Then these algorithms will be applied to spanning tree polytopes to see if it performs better than a generic QP algorithm.

Definition 3 *The spanning tree polytope of a graph $G = (V, E)$ is the convex hull of vectors x^T for each spanning tree T of G, where x^T has an entry x_e^T corresponding to an edge $e \in E$ defined in the following way:*

$$x_e^T = \begin{cases} 1 & \textit{if } e \in T \\ 0 & \textit{otherwise} \end{cases} \tag{2}$$

Consider the complete graph on 3 vertices K_3 and label its edges e_1, e_2 and e_3. Clearly it has 3 spanning trees, T_1, T_2 and T_3, which means there are 3 points in the spanning tree polytope. Suppose T_1 consists of edges e_1 and e_2, then $x^{T_1} = (1, 1, 0)^T$ (here $(\cdot)^T$ denotes the transpose). The other two points can be defined in the same way. The resulting polytope would then be

$$P = \operatorname{conv}\left(\left\{\begin{pmatrix}1\\1\\0\end{pmatrix}, \begin{pmatrix}1\\0\\1\end{pmatrix}, \begin{pmatrix}0\\1\\1\end{pmatrix}\right\}\right)$$

Using spanning tree polytopes will allow us to take advantage of the properties of spanning trees, specifically Kruskal's algorithm for finding a minimum-cost spanning tree in a graph, which is much more efficient than what is used in Wolfe's algorithm to find new points.

2 Wolfe algorithm

Definitions

We are given a polytope in n dimensions and we want to find the point in the polytope with the smallest norm. We will start with a few definitions. First, we need a way to describe a point of the polytope in terms of the vertices of the polytope.

Definition 4 *Let $P \subseteq \mathbb{R}^n$ be a set of m points and $X \in \mathbb{R}^n$. The vector $w = (w_1, ..., w_m)$ describes the **barycentric coordinates** of X in terms of P. Thus $X = \sum_{k=1}^{m} P_k w_k$.*

For an arbitrary set P, these coordinates may not be unique. In order for them to be unique, the set must satisfy a certain property.

Definition 5 *A set of points Q is **affinely independent** if none of the points in Q are in the affine hull of the remaining points.*

Throughout the algorithm we will always work with a subset of points Q of P, which has the property that the points are affinely independent. This affine independence means that the barycentric coordinates of X in Q are always unique and Q can have at most $n + 1$ points. As an example, in $\mathbb{R}^2$ any point can be uniquely described in terms of 3 points, given they are affinely independent (not on the same line). But given 4 such points, there is no unique way to describe the target in terms of them.

Definition 6 *We define the point that minimizes $|X|$ over the affine hull of a set of points Q as* **near**(Q)*. We call a set Q a* ***corral*** *if* $\mathrm{near}(Q)$ *is in the convex hull of Q.*

The method we will use to determine if a point minimizes over the whole polytope is to see whether it "separates" the polytope from the origin. For this we need to define a certain hyperplane for every point:

Definition 7 *We define* **hyp**(X) *to be the hyperplane* $\{x : X^T x = |X|^2\}$.

This hyperplane is essentially the hyperplane going through the respective point, that is orthogonal to the line from origin to the point.
Lastly, we will need to know what a "face" of a polytope is.

Definition 8 *A* ***face*** *F of a polytope* $\mathrm{conv}(Q)$ *is conv(Q') where Q' is a proper subset of Q.*

Geometric description

Now we are ready to state the algorithm. In short, the algorithm works by picking different subsets Q of P and minimizing over them, until the minimum of the whole polytope is found. The algorithm consists of major and minor cycles, where the minor cycles are always part of a major cycle. A major cycle starts with a corral Q and finds a new point to adjoin to it. If the new Q is still a corral, the major cycle ends and a new one starts. If it is not, then a minor cycle starts, where points are removed from Q until it is a corral again. Here is the algorithm described geometrically in more detail:

Algorithm 1 Wolfe algorithm geometric description

```
 1: $X \leftarrow$ point of smallest norm in $P$                                ▷ Step 0
 2: $Q \leftarrow \{X\}$
 3: while true do                                                              ▷ Step 1
 4:     start major cycle
 5:     if $X = 0$ or if hyp($X$) separates $P$ from the origin then
 6:         stop, $X$ is the solution
 7:     else
 8:         add to $Q$ point on near side of hyp($X$) with largest distance from it
 9:         while true do                                                      ▷ Step 2
10:             $Y \leftarrow$ point of smallest norm in aff($Q$)
11:             if $Y$ is in the relative interior of conv($Q$) then
12:                 $X \leftarrow Y$
13:                 start next major cycle
14:             else                                                           ▷ Step 3
15:                 start minor cycle
16:                 $Z \leftarrow$ the nearest point to $Y$ on $\overline{XY}$ that is still in conv($Q$)
17:                 $F \leftarrow$ the face of conv($Q$) on which $Z$ lies
18:                 delete from $Q$ a point not in $F$
19:                 $X \leftarrow Z$
20:             end if
21:         end while
22:     end if
23: end while
```

We will now trace the steps of the algorithm on a two-dimensional example (Figure 1).

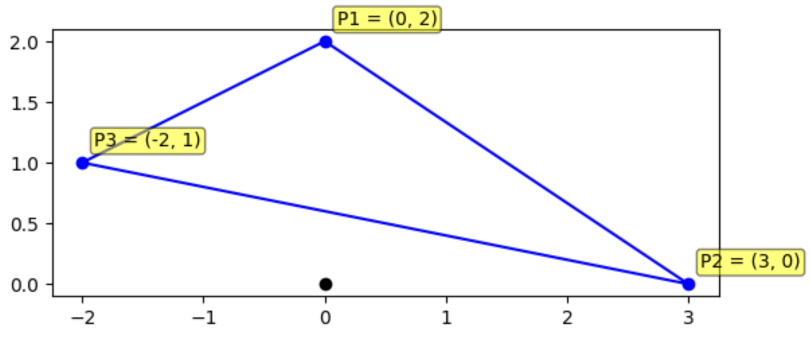

Figure 1: Example polytope in 2 dimensions

In step 0 we choose the point of smallest norm, which is P_0 in this case. That becomes the first position of X (Figure 2).

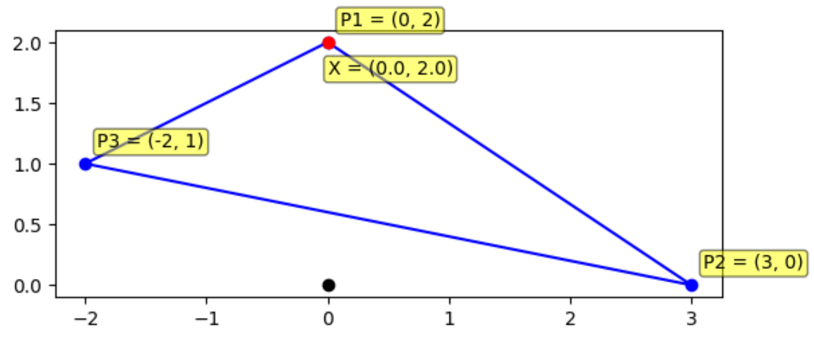

Figure 2: Polytope after the first major cycle

Next, we must find the point farthest away from hyp(X). In this case that is P_2, so our set Q now consists of P_1 and P_2. The new X is the point of smallest norm in the convex hull of those two points, which is the line between them. The position of X is updated (Figure 3).

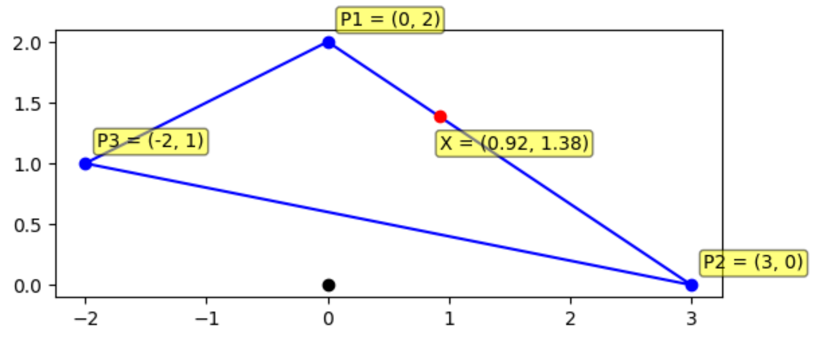

Figure 3: Polytope after the second major cycle

The only point that can be added to Q now is P_3. $Y = \text{near}(Q)$ of the new Q is then the origin, which is outside of the convex hull of Q. This means Q is not a corral anymore and a minor cycle starts. We see that the closest point

to Y (the origin) on the line between X and Y, that is still in the convex hull of Q (the polytope) lies on the line between P_2 and P_3. Thus F is defined by $\{P_2, P_3\}$, so P_1 is discarded from Q.
Recalculating X for the new $Q = \{P_2, P_3\}$ gives us the next position of X (Figure 4). It can be seen that hyp(X) separates the polytope from the origin, which means we have arrived at the solution.

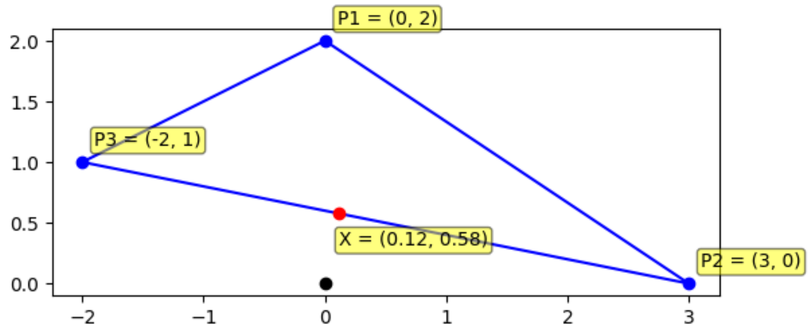

Figure 4: Solution

Algebraic description

Now we will define the algorithm algebraically. In order to find the point of minimum norm over aff(Q), we have to solve the problem

$$\min |X|^2 = w^T Q^T Q w$$
$$\text{s.t. } e^T w = 1.$$

The solution to these is given by the system

$$e^T w = 1$$

$$e\lambda + Q^T Q w = 0$$

[1]. However, building these equations every major and minor cycle is computationally expensive, so Wolfe proposes keeping them in the form of an upper triangular matrix R which is updated every time Q changes.
Another note about implementation is that we will not keep track of Q explicitly, instead only the set of indices of the points of Q in P are kept.
Next the algorithm will be described in detail (2), using the following variables:

- P: given set of points
- S: set of indices of Q

- X: point near(Q) with Q corral
- w: barycentric coordinates of X in terms of Q
- v: new candidate for w; barycentric coordinates of Y
- R: matrix representing the equations

Algorithm 2 Wolfe algorithm algebraic description

```
1:  $J \leftarrow$ index of point of smallest norm in $P$                      ▷ Step 0
2:  add $J$ to $S$
3:  $R \leftarrow [\sqrt{1+|P_J|^2}]$
4:  while true do                                                               ▷ Step 1
5:      if $X^T P_J > X^T X$ then                                               ▷ 1a
6:          stop, $X$ is the solution
7:      else
8:          $J \leftarrow$ index of point with smallest dot product with $X$       ▷ 1b
9:          $r \leftarrow$ solve $R^T r = e + P[S]^T P_J$                          ▷ 1c
10:         $\rho \leftarrow \sqrt{1+|P_J|^2-|r|^2}$
11:         adjoin to $R$ on the right $(r\rho)^T$
12:         add $J$ to $S$                                                      ▷ 1d
13:         while true do                                                       ▷ Step 2
14:             $v \leftarrow$ the solution $u$ of $R^T\bar{u} = e$ and $Ru = \bar{u}$   ▷ 2a
15:             divide $v$ by $|\bar{u}|^2$
16:             if $v > 0$ then                                                 ▷ 2b
17:                 $X \leftarrow Y$
18:                 break
19:             else                                                            ▷ Step 3
20:                 POS $\leftarrow$ set of indices $i$ with $w_i > v_i$           ▷ 3a
21:                 $\theta \leftarrow$ min(1,min($w_i/(w_i - v_i)$), $i \in$ POS)    ▷ 3b
22:                 replace $w$ by $(1-\theta)w + \theta v$                         ▷ 3c
23:                 delete from $w$ a zero component and its index from $S$      ▷ 3d
24:                 delete column of $R$ and restore to upper triangular        ▷ 3e
25:             end if
26:         end while
27:     end if
28: end while
```

This formulation closely follows the geometric description in Algorithm 1. Step 0 consists of picking a starting point and setting up the matrix of equations R. Step 1a checks if the current X is the solution. Step 1b finds the point with the smallest dot product with the current X, which is equivalent to the step of finding the point on the near side of hyp(X) which is farthest away from it. Step 1c is concerned with updating the matrix R with the newly added point. Step 2 solves the equations of R and checks if the solution is in the convex hull of Q by looking at the barycentric coordinates.

Step 3 is about deciding which point to remove from Q (in this case S, which describes Q). We do not actually calculate that point here, instead we look at its barycentric coordinates, specifically which ones become 0. The ones that stay positive correspond to the points of the face the new Z lies on, thus any one of the others can be discarded.
There are some differences between this implementation and the one proposed by Wolfe. In step 2a the solution to the equations needs to be divided by a factor in order to be correct. Furthermore, in step 3c, the factors in front of w and v have been switched. These mistakes were discovered by experimenting with the algorithm.
The operations on R will not be explained in this paper, they can be found in Wolfe's paper. [1]. It should, however, be noted that the order of the steps given by Wolfe is incorrect: 1c needs to happen before 1d as otherwise the size of S makes the dimensions of the matrices incompatible.
Some features need to be implemented to avoid numerical issues, like a "disaster check" that makes sure that in step 1 the newly added point is not already in Q, which can happen if the polytope contains points that are very close to each other. These have been omitted here for readability and can also be found in Wolfe's paper [1].

Computational complexity

We will now determine the worst-case complexity of the Wolfe algorithm. First, a few observations.
The size in S (the number of points in Q) is bounded by $n+1$ since having more points means the set is not affinely independent. This will result in near(Q) being at the origin, which is either the solution or will trigger a minor cycle, decreasing the size of Q.
The number of major cycles is only bounded by the possible number of corrals. A corral cannot be considered more than once because the norm of X can only decrease with each cycle [1].
The complexities of the individual steps can be found in appendix A. In short, the worst-case complexity of a single major cycle is polynomial in n, the number of dimensions of the polytope, and linear in m, the number of points. The number of points is typically much larger than the number of dimensions, and this dependence on m only occurs in one step of the algorithm. This makes it an interesting topic when it comes to improving this algorithm and will be addressed when working with spanning tree polytopes.
We have now determined the complexity of a single major cycle, but we still need to have a look at the total number of major cycles. As mentioned before, its upper bound is the number of possible corrals of the polytope, which is very high: $\sum_{i=1}^{n+1} \binom{m}{i}$, order $O(m^m)$. However, in practice the number of major cycles is much lower than that (see Section 5).

3 Formulation as a quadratic program

We will now look at how to formulate and solve the nearest point problem as a quadratic program (QP) and what types of algorithms to solve such a program exist.
The general form of a quadratic program is

$$\min_x\{\frac{1}{2}x^TQx + q^Tx\}$$

$$\text{s.t. } Ax = a$$

$$Bx \leq b$$

$$x \geq 0$$

[2]. Unlike the nearest point problem defined in Section 1, a general QP may have both equality and inequality constraints. The general form of the nearest point problem would be:

$$\min \frac{1}{2}x^TPx \text{ s.t. } e^Tx = 1 \text{ and } x \geq 0 \tag{3}$$

with e being a vector of 1's of size m and P being the set of points in matrix form (each column represents a point).

It is interesting to look into what type of generic QP algorithm is best suited for our problem. Most popular QP algorithms fall in one of two categories: active-set (AS) and interior-point (IP) [3].
Active-set: Move along the boundary of the feasible region, always maintaining a set of vertices as an "active set". Clearly the Wolfe algorithm falls in this category. These algorithms solve many sub-problems of the main problem, for different sets of vertices. This results in relatively many iterations, but smaller sub-problems. A disadvantage to these is that they slow down close to solution [3]. Another disadvantage is that the system to be solved changes each iteration and sometimes needs to be derived again from scratch [3].
Interior-point: These algorithms traverse the inside of the feasible region. They use fewer iterations to get to the solution but each iteration considers entire problem and takes longer as a consequence [3]. This might make these algorithms less suitable for larger problems. An advantage of IP could be that the system to be solved remains the same between steps [3].
Both of these types of algorithms could be applied to the nearest point problem. The update procedure for R given in the paper does not involve formulating the equations from scratch, so it should not slow down the computation as much as other AS methods might. Later this will be investigated through experiments to show if the Wolfe algorithm performs better than an IP algorithm in a specific context.
As far as computational complexity, neither type of method can provide a good upper bound for the number of iterations. Like Wolfe's algorithm, AS methods

are only bounded by the number of possible combinations of points, which is very large, whereas IP methods do not provide any upper bound. Based on these observations, we will not research the topic of computational complexity any further. Instead, we will use computational experiments to see which algorithm performs better.

4 Application to spanning tree polytopes

The Wolfe algorithm can be applied to a certain class of polytopes: spanning tree polytopes. This extends research that has been done on spanning tree polytopes, where the goal is to find the barycentric coordinates of a point that is known to be inside the polytope [4].

Minimum spanning tree formulation

The number of spanning trees of a graph is very large, but if we use Wolfe's algorithm we never use all vertices of the polytope except in step 0 and step 1. In step 1, we loop through all points of P in order to find the one that has the smallest dot product with the current X. However, we can avoid this rather expensive step by using the properties of spanning trees. In this case we are looking for a minimum-cost spanning tree, where the weights are the coordinates of X. This can be done without considering all points the polytope by using Kruskal's or Prim's algorithm, which are both very efficient. For step 0, finding the point of smallest norm, we can do the same with weight 1 for every edge.
This formulation requires some changes in Wolfe's algorithm to accommodate the extra features. First of all, we do not keep a full list of points P anymore, nor the set of indices S, but instead only the set Q. Furthermore, the "disaster check" is no longer necessary since the points of the polytope cannot be very close together because of the definition of the spanning tree polytope.
In experiments we will only work with complete graphs for simplicity. Removing an edge from a complete graph would be equivalent to forcing its respective coordinate in each spanning tree/vertex of the polytope to 0, which is effectively a projection of the polytope on a lower dimension.

Compact formulation

There is another formulation involving spanning tree polytopes we can investigate. We call this the compact formulation [5]. We want to solve the problem as a QP, but the size of the problem would be too large in the original space, given in [6]. The number of constraints in this new formulation is $O(n^3)$.
Let $G = (V, E)$ be a graph and $D = (V, A)$ its corresponding bi-directed graph (for each edge of G, D has an arc in each direction). Let $r \in V$ be an arbitrary root node.
We define a polytope $Q = \{(x, f) \text{ s.t. } (4), (5), (6) \text{ and } (7) \text{ are satisfied}\}$ with

$$\sum_{(i,j)\in A} f_{it}^k - \sum_{(t,j)\in A} f_{tj}^k = \begin{cases} -1, & t = 0 \\ 0, & t \neq 0 \wedge t \neq k, \quad \forall k \in V \setminus \{r\} \\ 1, & t = k \end{cases} \tag{4}$$

for all $k, v \in V$ with $k \neq r$.

$$f_{v,w}^k \geq 0 \tag{5}$$

for all $k \in V\backslash\{r\}$ and all $(v, w) \in A$.

$$f_{v,w}^k + f_{w,v}^l \leq x_{\{v,w\}} \leq 1 \tag{6}$$

for all $\{v, w\} \in E$ and all $k, l \in V\backslash\{r\}$.

$$\sum_{e\in E} x_e = |V| - 1 \tag{7}$$

The projection of Q onto x is the spanning tree polytope $P = \text{conv}(\{x^T : T \text{ is a spanning tree}\})$ [7]. This formulation lets us solve the nearest point problem as a QP.

5 Experiments

In this section the two methods for solving the nearest point problem on a spanning tree polytope will be compared through experiments. The source code can be found on a Git repository [8]. As mentioned before, we will only use complete graphs to generate spanning tree polytopes for simplicity. We will generate random target points, with each coordinate uniformly distributed on (-1, 0) to guarantee the target point will not be very close to the polytope. For solving the QP of the compact formulation we will use the Gurobi module in Python [9], which uses an interior-point method. Because the Wolfe algorithm is an active-set algorithm and Gurobi is interior-point, it does not make sense to compare number of iterations and time per iteration. Instead the total run time can be compared.
While using the precision values recommended by the author, running the implementation of the Wolfe algorithm for such large input polytopes leads to numerical errors. There are two types of problems: sometimes the algorithm does not terminate, sometimes in step 3 the list POS is empty. Neither of these should occur in theory, but experiments show that both occur when the algorithm is already very close the solution (this is seen by comparing with the solution of Gurobi). Increasing the precision of the tolerance of v to be considered positive in step 2 of the algorithm from 10^{-10} to 10^{-15} fixes both of these issues. These errors can easily be explained: the precision values proposed by Wolfe were based on hardware from the 1970's, which presumably worked differently and was not applicable to such large problems.

Difference in solutions

Naturally the solutions of both algorithms deviate slightly. The table in appendix B shows how that difference changes with the number of nodes in the graph. An interesting observation is that the solution of Wolfe is always slightly better, by about 10^{-9}. This can be explained with the accuracy of the algorithms: our Wolfe implementation uses precision of 10^{-10}, whereas Gurobi uses 10^{-6}. This difference also does not seem to increase with the size of the graph (except for one outlier).

Comparison of run times

We would like to compare the run times of both algorithms. The results are shown in Figure 5.

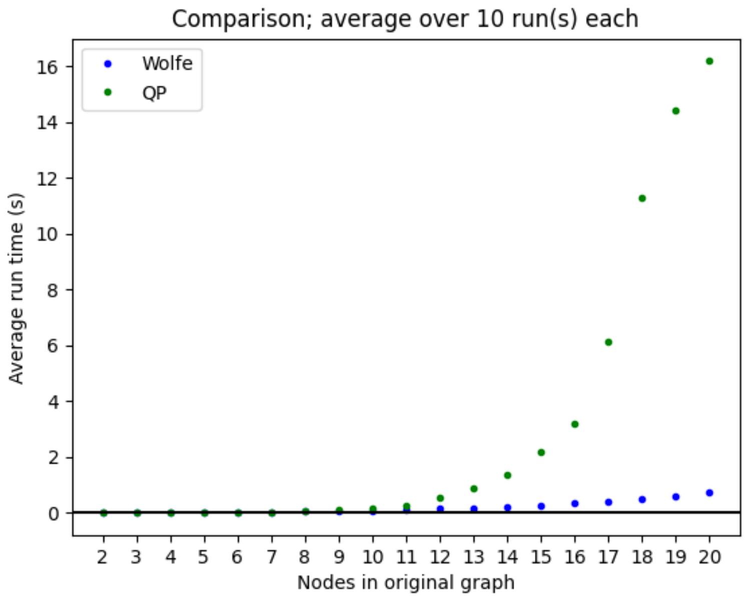

Figure 5: Comparison of both algorithms' performance on the spanning tree polytope of a complete graph and a randomly generated target point. The same target points were used for both algorithms. The horizontal axis shows the number of nodes of the original complete graph, from which the spanning tree polytope is generated. For each point in the graph, the average of the results of 10 runs of the original graph are used.

It can clearly be seen that the QP implementation takes longer. This fact, combined with the quality of the solution discussed previously, leads us to the conclusion that the Wolfe algorithm is better suited for the nearest point problem, at least when it comes to spanning tree polytopes.

Further experiments on Wolfe algorithm

We would like to further analyze the behaviour of the Wolfe algorithm for larger problems. Since solving the problem as a QP takes a long time for larger problems, we exclude it at this stage. The results are shown in Figure 6

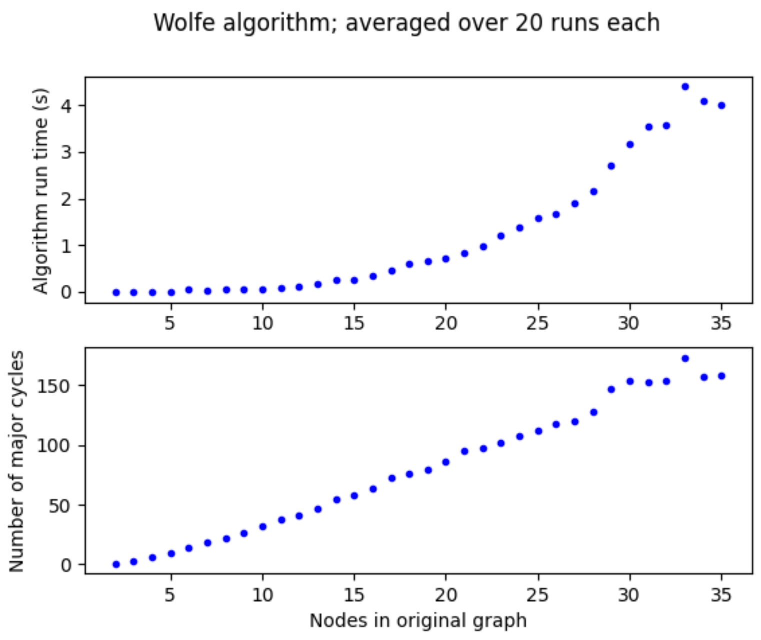

Figure 6: Performance of Wolfe algorithm on spanning tree polytopes of complete graphs. The horizontal axis shows how many nodes the complete graph has, that the spanning tree polytope was generated from.

It should be kept in mind that the number of points of the polytope is n^{n-2} with n being the number of nodes of the original graph. A complete graph of 30 vertices has $35^{35-2} \approx 10^{50}$, which shows that the Wolfe algorithm works fine on larger problems. The second plot shows that the number of major cycles increases roughly linearly with the number of nodes.
It is also interesting to see how the size of Q develops over the course of an iteration of the algorithm. The result of several such iterations is shown in Figure 7.

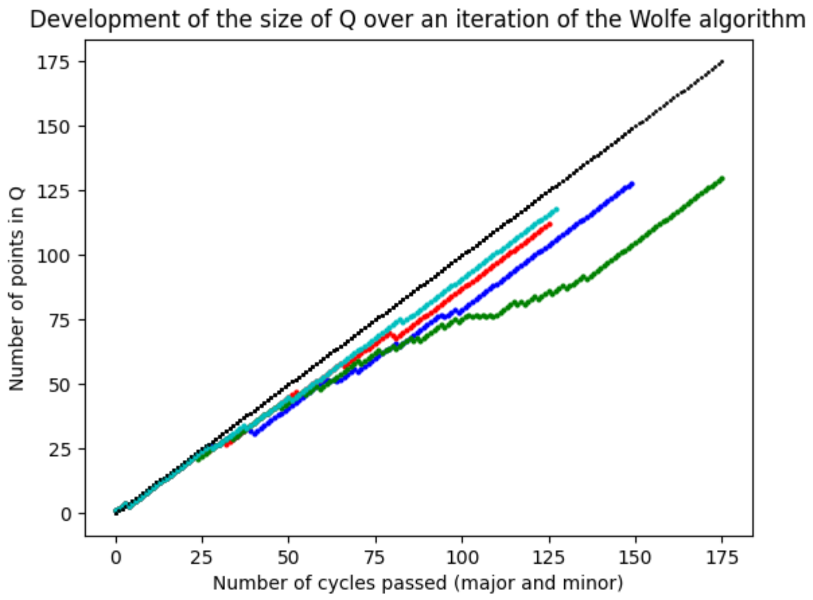

Figure 7: Development of the size of the active set (Q) over one iteration of the Wolfe algorithm. The black line (number of cycles=number of points) is for comparison.

Clearly, the curve for any run of the algorithm cannot be above the black line, since at most one point can be added to Q per cycle. But the plot shows that the actual results from the runs are not very far from the black line either, meaning there are not too many minor cycles. This further shows that the algorithm performs well.

6 Conclusion

We have seen how the nearest point problem can be solved both by a specialized algorithm and as a standard quadratic program. Furthermore, both can be applied on spanning tree polytopes. Experiments show that the algorithm of Wolfe performs better, both in terms of run time and solution quality, on spanning tree polytopes. It can also be applied to very large problems without sacrificing accuracy.
Some possible points of improvement for the algorithm of Wolfe are given in his paper [1]. The main point which addressed in this paper is the inefficiency of cycling through all vertices of the polytope to find a new one every major cycle. In this implementation we choose the vertex farthest away from hyp(X) on the side of the origin. Instead one could, for example, find which vertex would result

in the smallest value of $|X|$ and then add it to the active set. This could result in less major cycles.
The choice to apply the algorithm to spanning tree polytopes was made to use the properties of spanning trees. But naturally it can also be applied to other kinds of polytopes to investigate its performance.
As a further direction for experiments, one could use exact arithmetic to properly assess the quality of the solution.

References

[1] P. Wolfe, "Finding the nearest point in a polytope," *Mathematical Programming*, vol. 11, pp. 128–149, Dec 1976.

[2] T. R. Kruth, "Interior-point algorithms for quadratic programming," Master's thesis, Technical University of Denmark, DTU, DK-2800 Kgs. Lyngby, Denmark, 2008.

[3] A. Geletu, "Lecture slides: Quadratic programming problems - a review on algorithms and applications (active-set and interior point methods)."

[4] D. P. Williamson, "An experimental evaluation of the best-of-many christofides' algorithm for the traveling salesman problem," 2015.

[5] A. M. Chwatal and G. R. Raidl, "Solving the minimum label spanning tree problem by mathematical programming techniques," *Advances in Operations Research*, vol. 2011, Jun 2011.

[6] J. Edmonds, "Matroids and the greedy algorithm," *Mathematical programming*, vol. 1, no. 1, pp. 127–136, 1971.

[7] R. Wong, "Integer programming formulations of the traveling salesman problem," *In Proceedings of 1980 IEEE International Conference on Circuits and Computers*, p. 149–152, 1980.

[8] B. Petrov, "Bachelor assignment git repository."

[9] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2020.

A Computational complexity of Wolfe algorithm

Step	Description	Complexity (O(...))	Total
0	Find point in P of minimal norm, create R	mn^2	mn^2
1a	Multiply $P[S]w$	n^3	$n^2(m+n)$
1b	Find point in P of minimal dot product with X	mn^2	
1c	Maximum norm in S	n^3	
1d	Check if J in S	n	
1d-1	Solve R, first solve matrix multiplication.	n^3	
1d-2	Update R	n	
1e	Add S to J	-	
2a	Solve R twice	n^3	n^3
2b	Check if $v > 0$	n	
3a	Find indices of S	n	n^3
3b	Set θ based on indices of S	n	
3c	Replace w	n	
3d	Change elements of w	n	
3e	Delete a zero component	n	
3f	Delete column of R	-	
3g	Iterate over rows of R, doing row operations for each	n^3	

B Difference of solutions of Wolfe algorithm and QP solver

Size of graph	Difference in solutions (Wolfe minis Gurobi)
2	0.000000e+00
3	-8.063830e-10
4	-1.066055e-10
5	-1.140024e-09
6	-2.267790e-09
7	-1.232024e-09
8	-7.573072e-10
9	-7.359644e-10
10	-1.912691e-09
11	-9.493895e-10
12	-1.672143e-09
13	-6.254124e-10
14	-1.870507e-09
15	-1.173162e-09
16	-3.471272e-10
17	-2.154771e-08
18	-2.017024e-08
19	-4.075792e-04
20	-5.558251e-08