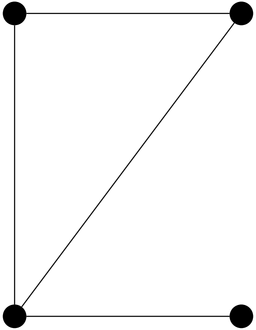
Threefold Problem Set — Partitions

Kalina Petrova
Based on slides by Martin Raszyk

December 2, 2020

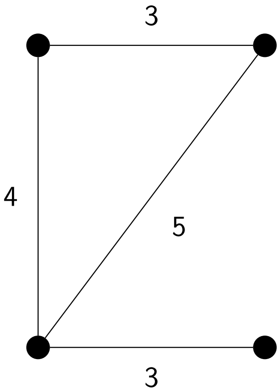
Introduction

Introduction



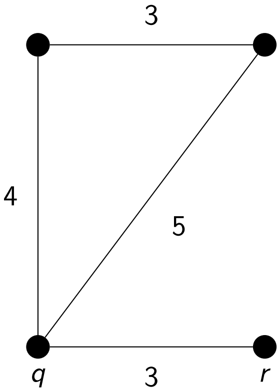
Introduction

- the length of an edge is the Euclidean distance between its endpoints



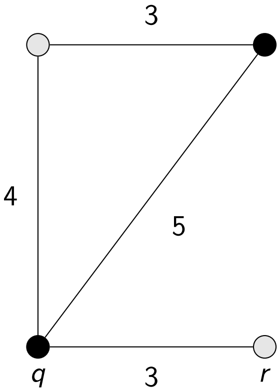
Introduction

- the length of an edge is the Euclidean distance between its endpoints



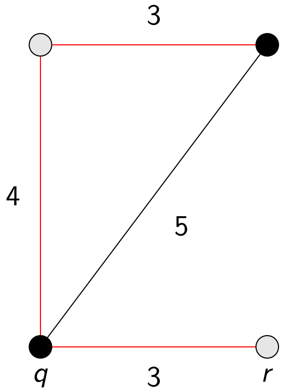
Introduction

- the length of an edge is the Euclidean distance between its endpoints
- a q-r partition is marked by shading



Introduction

- the length of an edge is the Euclidean distance between its endpoints
- a q-r partition is marked by shading
- edges crossing the partition are highlighted in red



Exercise: MiniSum

Edges between every pair of points whose distance is at most d

Output q-r partition

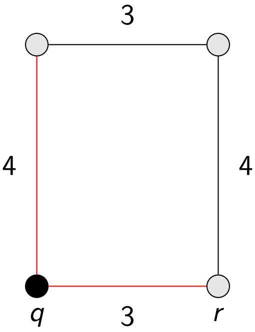
Goal minimize the **sum** of **squared lengths** of all edges crossing the partition

Exercise: MiniSum

Edges between every pair of points whose distance is at most d

Output q-r partition

Goal minimize the **sum** of **squared lengths** of all edges crossing the partition



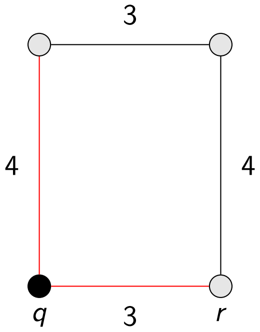
$\Sigma = 3^2 + 4^2 = 25$

Exercise: MiniSum

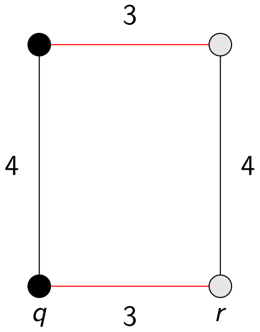
Edges between every pair of points whose distance is at most d

Output q-r partition

Goal minimize the **sum** of **squared lengths** of all edges crossing the partition



$\Sigma = 3^2 + 4^2 = 25$



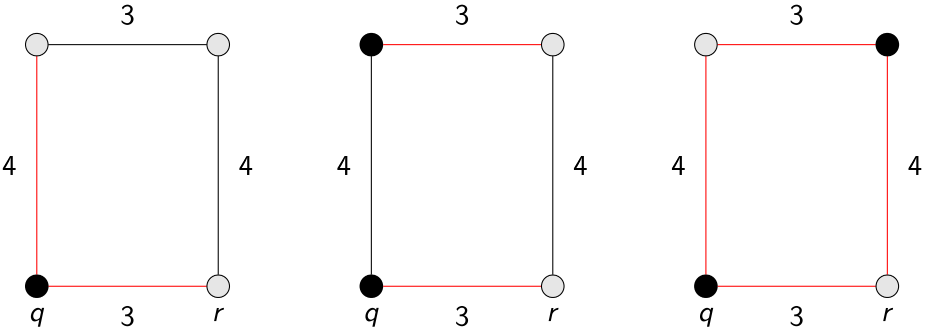
$\Sigma = 3^2 + 3^2 = 18$

Exercise: MiniSum

Edges between every pair of points whose distance is at most d

Output q-r partition

Goal minimize the **sum** of **squared lengths** of all edges crossing the partition



$\Sigma = 3^2 + 4^2 = 25$ $\Sigma = 3^2 + 3^2 = 18$ $\Sigma = 3^2 + 4^2 + 3^2 + 4^2 = 50$

Exercise: MiniSum — Solution

1. We need to find a minimum q-r cut $(S, P \setminus S)$ in the undirected weighted graph where an edge weight is its squared length.

Exercise: MiniSum — Solution

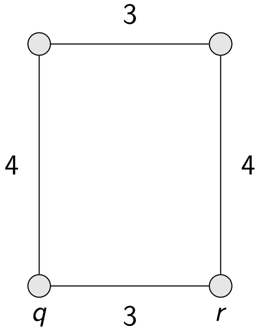
1. We need to find a minimum q-r cut $(S, P \setminus S)$ in the undirected weighted graph where an edge weight is its squared length.
2. This can be done using the max-flow min-cut theorem.

Exercise: MiniSum — Solution

1. We need to find a minimum q-r cut $(S, P \setminus S)$ in the undirected weighted graph where an edge weight is its squared length.
2. This can be done using the max-flow min-cut theorem.
3. Set S can be obtained by BFS from q in the flow network ignoring saturated edges.

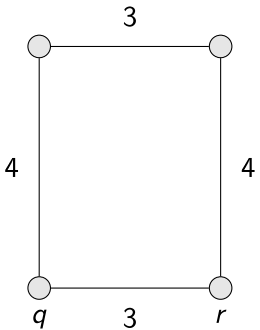
```
vis[q] = true;
Q.push(q);
while (!Q.empty()) {
  const int u = Q.front();
  Q.pop();
  out_edge_it ebeg, eend;
  for (boost::tie(ebeg, eend) = boost::out_edges(u, G); ebeg != eend; ++ebeg) {
    const int v = boost::target(*ebeg, G);
    if (rc_map[*ebeg] == 0 || vis[v]) continue;
    vis[v] = true;
    Q.push(v);
  }
}
```

Exercise: MiniSum — Illustration of the solution

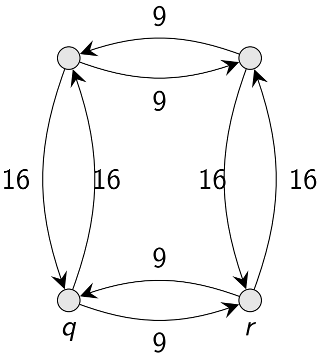


Example with $d = 4$

Exercise: MiniSum — Illustration of the solution

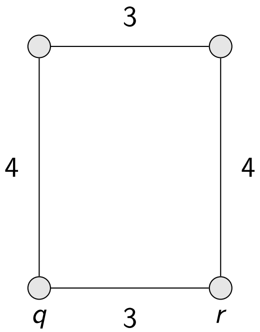


Example with $d = 4$

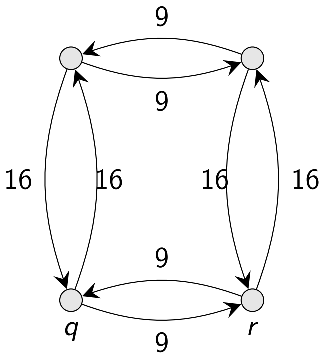


Flow network

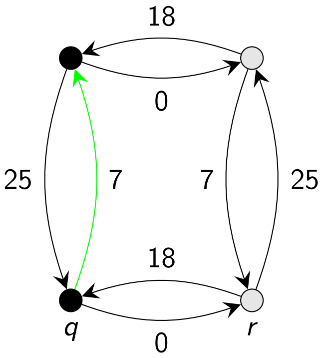
Exercise: MiniSum — Illustration of the solution



Example with $d = 4$



Flow network



Maximum flow $= 18$, residual graph shown

Exercise: MiniSum — Time complexity

1. Push-relabel maximum flow algorithm takes time $O(n^3)$ in the worst case.

Exercise: MiniSum — Time complexity

1. Push-relabel maximum flow algorithm takes time $O(n^3)$ in the worst case.
2. BFS takes time $O(n^2)$ as there are up to $O(n^2)$ edges in the graph.

Exercise: MiniSum — Time complexity

1. Push-relabel maximum flow algorithm takes time $O(n^3)$ in the worst case.
2. BFS takes time $O(n^2)$ as there are up to $O(n^2)$ edges in the graph.
3. The overall time complexity is thus $O(n^3)$.

Exercise: MiniSum — Time complexity

1. Push-relabel maximum flow algorithm takes time $O(n^3)$ in the worst case.
2. BFS takes time $O(n^2)$ as there are up to $O(n^2)$ edges in the graph.
3. The overall time complexity is thus $O(n^3)$.

Rule of thumb for push-relabel maximum flow algorithm:

Here we have $2 \leq n \leq 500$. ✓

Exercise: MiniLength

Edges between every pair of points

Output q-r partition

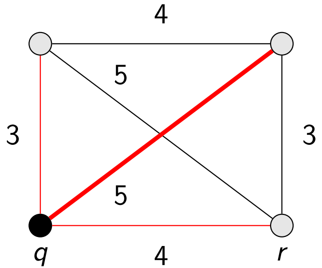
Goal minimize the **maximum length** of an edge crossing the partition

Exercise: MiniLength

Edges between every pair of points

Output q-r partition

Goal minimize the **maximum length** of an edge crossing the partition

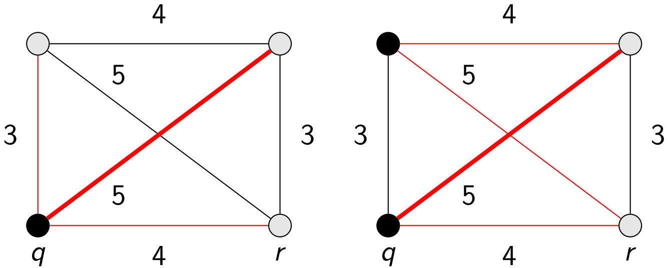


Exercise: MiniLength

Edges between every pair of points

Output q-r partition

Goal minimize the **maximum length** of an edge crossing the partition

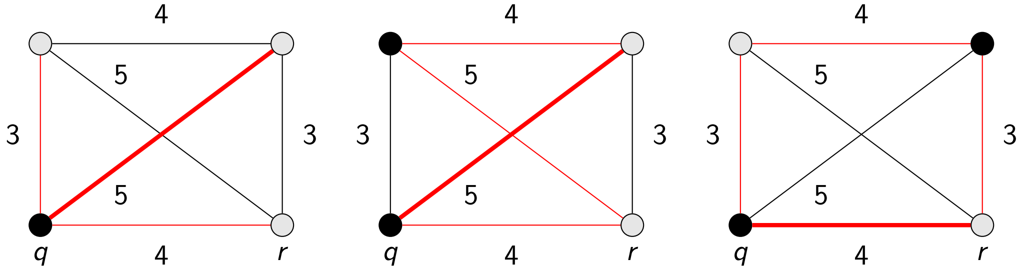


Exercise: MiniLength

Edges between every pair of points

Output q-r partition

Goal minimize the **maximum length** of an edge crossing the partition



Exercise: MiniLength — Observation

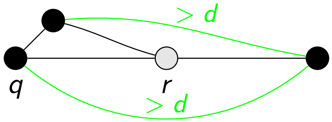
Lemma

For any length d, there is a q-r partition only crossed by edges of length at most d $\iff$ the set S of vertices reachable from q using only edges (strictly) longer than d does not contain r.

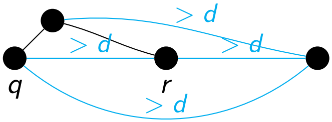
Exercise: MiniLength — Observation

Lemma

For any length d, there is a q-r partition only crossed by edges of length at most d $\iff$ the set S of vertices reachable from q using only edges (strictly) longer than d does not contain r.



Case 1, $r \notin S$



Case 2, $r \in S$

Proof.

1. If $r \notin S$, then $(S, P \setminus S)$ is a q-r partition and all crossing edges have length $\leq d$.
2. If $r \in S$, then there exists a q-r path using only edges strictly longer than d.
   Any q-r partition has to be crossed by an edge from this path.

Exercise: MiniLength — Solution

1. Do a binary search on the correct answer d_{min}.

Exercise: MiniLength — Solution

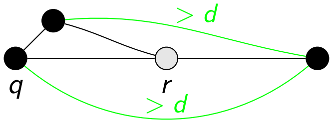
1. Do a binary search on the correct answer d_{min}.
2. For each guess d for the correct answer d_{min}, to determine whether $d_{min} \leq d$:

Exercise: MiniLength — Solution

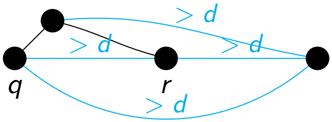
1. Do a binary search on the correct answer d_{min}.
2. For each guess d for the correct answer d_{min}, to determine whether $d_{min} \leq d$:
   2.1 Build a graph including only edges strictly longer than d.

Exercise: MiniLength — Solution

1. Do a binary search on the correct answer d_{min}.
2. For each guess d for the correct answer d_{min}, to determine whether $d_{min} \leq d$:
   2.1 Build a graph including only edges strictly longer than d.
   2.2 If there is a path from q to r, then $d_{min} > d$, otherwise $d_{min} \leq d$.



Case 1, $r \notin S$



Case 2, $r \in S$

Exercise: MiniLength — Time complexity

1. For each guess d for the correct answer d_{min}, a breadth-first search from q takes $O(n^2)$ time as there are $O(n^2)$ edges.

Exercise: MiniLength — Time complexity

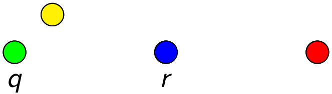
1. For each guess d for the correct answer d_{min}, a breadth-first search from q takes $O(n^2)$ time as there are $O(n^2)$ edges.
2. We repeat this step $O(\log n)$ times within the binary search.

Exercise: MiniLength — Time complexity

1. For each guess d for the correct answer d_{min}, a breadth-first search from q takes $O(n^2)$ time as there are $O(n^2)$ edges.
2. We repeat this step $O(\log n)$ times within the binary search.
3. Thus, the total time complexity is $O(n^2 \log n)$.

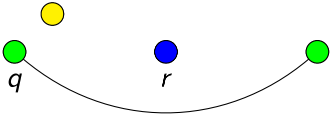
Exercise: MiniLength — Alternative solution

1. Using a Union-Find data structure, add edges starting from the longest until q and r belong to the same connected component (after adding some edge e).
2. The optimal partition is obtained by taking vertices reachable from q using only edges longer than e.



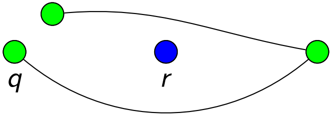
Exercise: MiniLength — Alternative solution

1. Using a Union-Find data structure, add edges starting from the longest until q and r belong to the same connected component (after adding some edge e).
2. The optimal partition is obtained by taking vertices reachable from q using only edges longer than e.



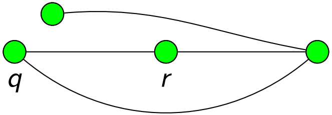
Exercise: MiniLength — Alternative solution

1. Using a Union-Find data structure, add edges starting from the longest until q and r belong to the same connected component (after adding some edge e).
2. The optimal partition is obtained by taking vertices reachable from q using only edges longer than e.



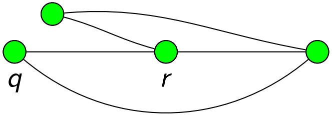
Exercise: MiniLength — Alternative solution

1. Using a Union-Find data structure, add edges starting from the longest until q and r belong to the same connected component (after adding some edge e).
2. The optimal partition is obtained by taking vertices reachable from q using only edges longer than e.



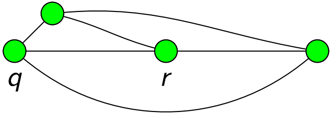
Exercise: MiniLength — Alternative solution

1. Using a Union-Find data structure, add edges starting from the longest until q and r belong to the same connected component (after adding some edge e).
2. The optimal partition is obtained by taking vertices reachable from q using only edges longer than e.



Exercise: MiniLength — Alternative solution

1. Using a Union-Find data structure, add edges starting from the longest until q and r belong to the same connected component (after adding some edge e).
2. The optimal partition is obtained by taking vertices reachable from q using only edges longer than e.



Exercise: MiniLength — Time complexity of alternative solution

1. Sorting the edges takes time $O(n^2 \log n)$ as there are $\Theta(n^2)$ edges.

Exercise: MiniLength — Time complexity of alternative solution

1. Sorting the edges takes time $O(n^2 \log n)$ as there are $\Theta(n^2)$ edges.
2. Processing the edges and maintaining the connected components takes time $O(n^2 \cdot \alpha(n))$, where $\alpha(n)$ is the (extremely slowly growing) inverse Ackermann function, by using a Union-Find data structure.

Exercise: MiniLength — Time complexity of alternative solution

1. Sorting the edges takes time $O(n^2 \log n)$ as there are $\Theta(n^2)$ edges.
2. Processing the edges and maintaining the connected components takes time $O(n^2 \cdot \alpha(n))$, where $\alpha(n)$ is the (extremely slowly growing) inverse Ackermann function, by using a Union-Find data structure.
3. The overall time complexity is thus $O(n^2 \log n)$.

WARNING

The following problem is more **difficult** and may need **additional ideas**!

Exercise: MiniDist

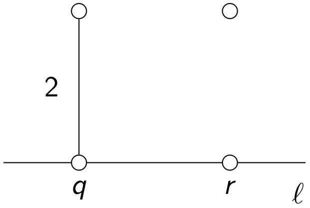
Output a line separating q and r

Goal minimize the **maximum Euclidean distance** of a point to the line

Exercise: MiniDist

Output a line separating q and r

Goal minimize the **maximum Euclidean distance** of a point to the line

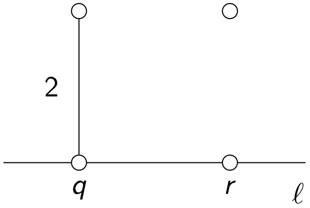


not optimal

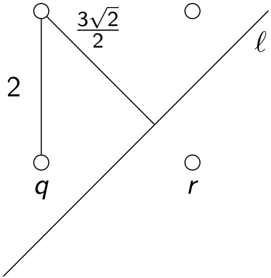
Exercise: MiniDist

Output a line separating q and r

Goal minimize the **maximum Euclidean distance** of a point to the line



not optimal

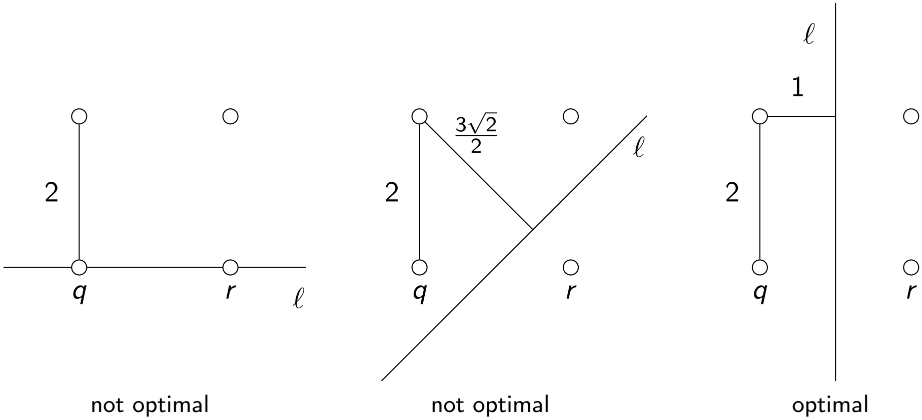


not optimal

Exercise: MiniDist

Output a line separating q and r

Goal minimize the **maximum Euclidean distance** of a point to the line



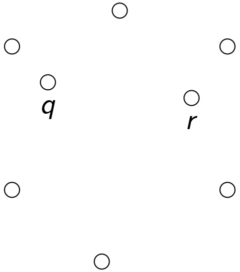
Exercise: MiniDist — Solution outline

1. For a fixed direction of the line, find the optimal line with that direction.

Exercise: MiniDist — Solution outline

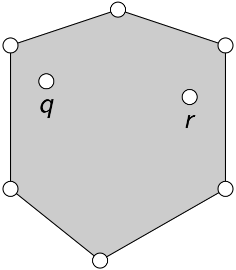
1. For a fixed direction of the line, find the optimal line with that direction.
2. Iterate through interesting directions of the line which are guaranteed to contain the direction of the optimal line.

Exercise: MiniDist — Optimal line with fixed direction



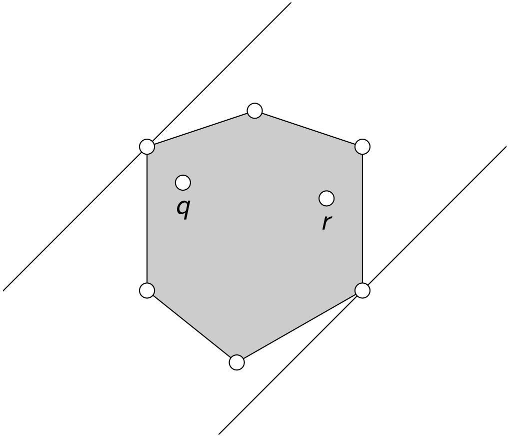
Given n points.

Exercise: MiniDist — Optimal line with fixed direction



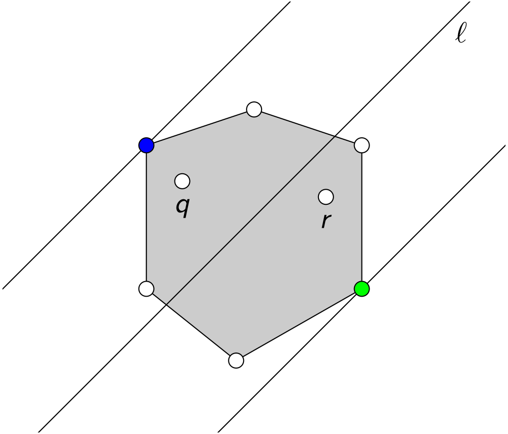
Compute their convex hull.

Exercise: MiniDist — Optimal line with fixed direction

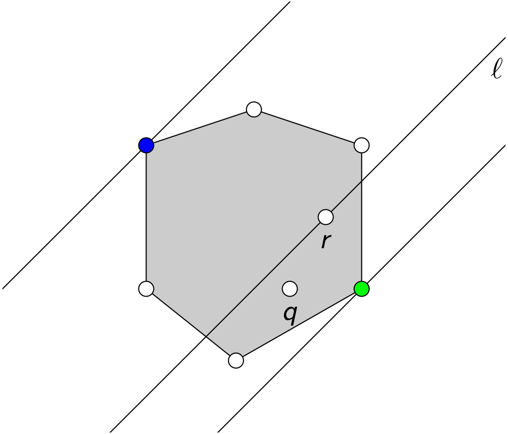


Find lines touching the convex hull.

Exercise: MiniDist — Optimal line with fixed direction

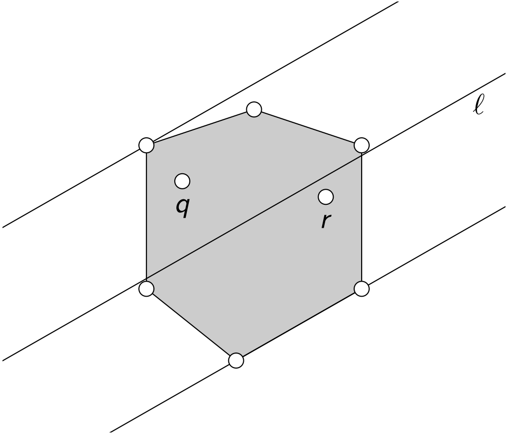


Optimal line: belt bisector separating q and r

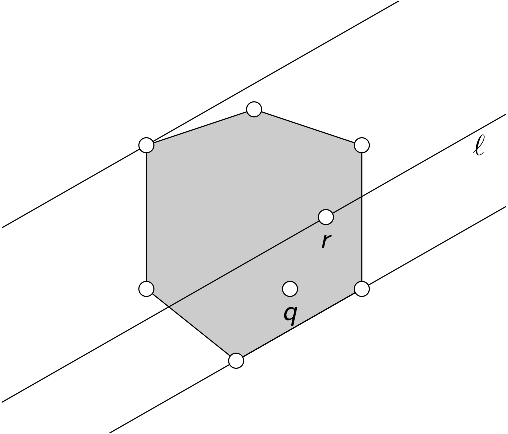


or a line through q or r.

Exercise: MiniDist — Interesting directions

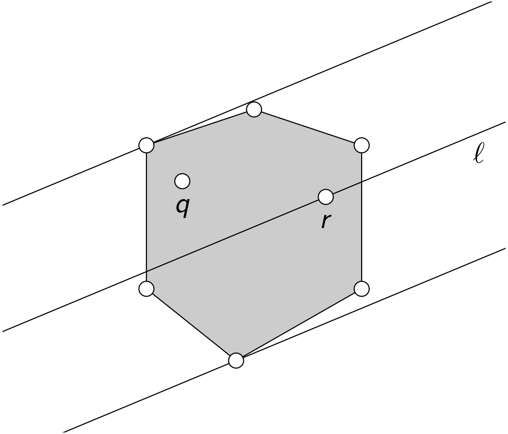


Line coincides with a side.



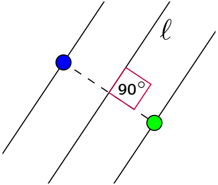
Line coincides with a side.

Exercise: MiniDist — Interesting directions

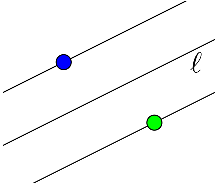


Belt bisector passes through q or r.

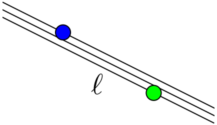
Exercise: MiniDist — Why are these the only interesting directions?



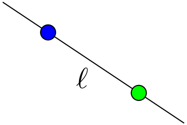
The worst direction



Rotate the lines

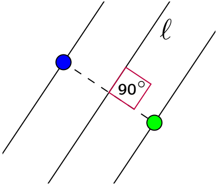


An almost best direction

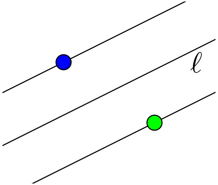


The best direction

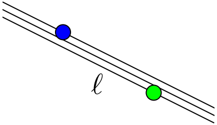
Exercise: MiniDist — Why are these the only interesting directions?



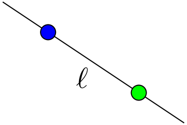
The worst direction



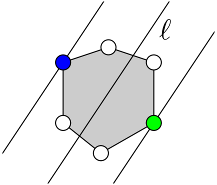
Rotate the lines



An almost best direction

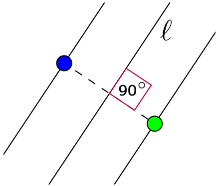


The best direction

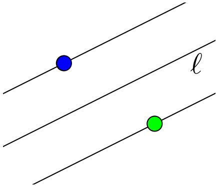


Back to many points
– The worst direction

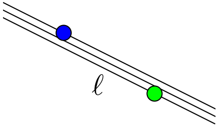
Exercise: MiniDist — Why are these the only interesting directions?



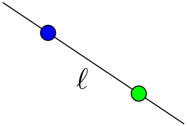
The worst direction



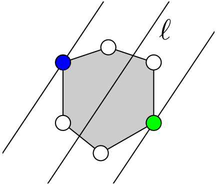
Rotate the lines



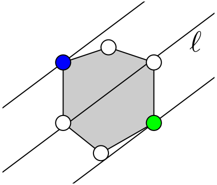
An almost best direction



The best direction

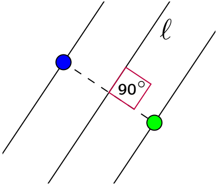


Back to many points – The worst direction

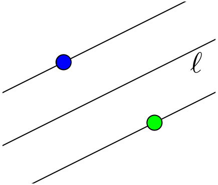


Rotate and improve

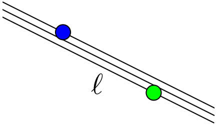
Exercise: MiniDist — Why are these the only interesting directions?



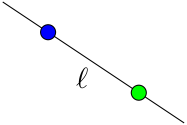
The worst direction



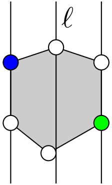
Rotate the lines



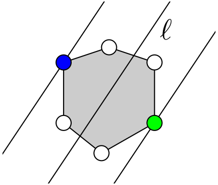
An almost best direction



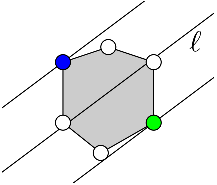
The best direction



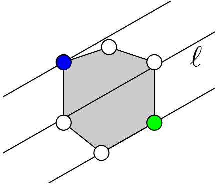
Can't rotate more than this



Back to many points – The worst direction

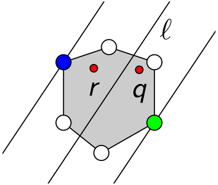


Rotate and improve

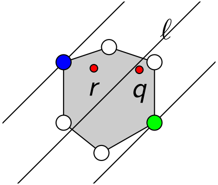


Can't rotate more than this

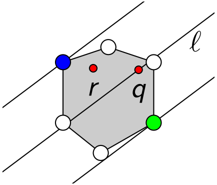
Exercise: MiniDist — Why are these the only interesting directions?



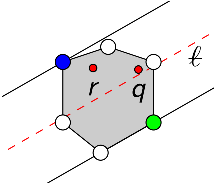
Reintroduce q and r – The worst direction



Rotate, use belt bisector

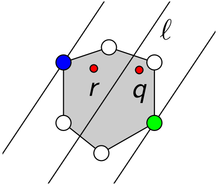


The best we can do while still using a belt bisector

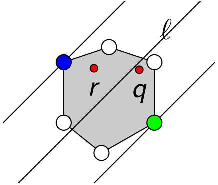


Belt bisector (dashed red line) not allowed as ℓ

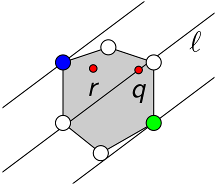
Exercise: MiniDist — Why are these the only interesting directions?



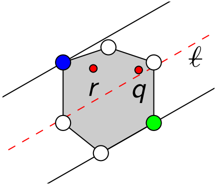
Reintroduce q and r – The worst direction



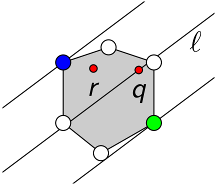
Rotate, use belt bisector



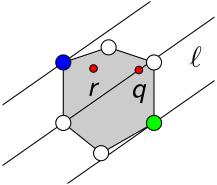
The best we can do while still using a belt bisector



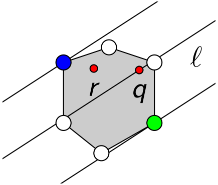
Belt bisector (dashed red line) not allowed as ℓ



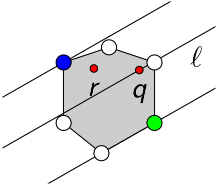
Belt bisector touches q



Rotate ℓ around q



Keep rotating ℓ around q



Until we hit a side

Exercise: MiniDist — Convex Hull by Delaunay Triangulation

```
typedef CGAL::Exact_predicates_exact_constructions_kernel K;
typedef CGAL::Point_2<K> Point;
typedef CGAL::Delaunay_triangulation_2<K> Triangulation;

std::vector<Point> convex_hull(const std::vector<Point> &pts) {
  Triangulation t;
  t.insert(pts.begin(), pts.end());

  Triangulation::Vertex_circulator v = t.incident_vertices(t.infinite_vertex());
  std::vector<Point> hull;
  do {
      hull.push_back(v->point());
  } while (++v != t.incident_vertices(t.infinite_vertex()));

  return hull;
}
```

Exercise: MiniDist — Time complexity

1. Computing convex hull (by using Delaunay triangulation) takes time $O(n \log n)$.

Exercise: MiniDist — Time complexity

1. Computing convex hull (by using Delaunay triangulation) takes time $O(n \log n)$.
2. Finding the optimal line with a fixed direction takes time $O(1)$.

Exercise: MiniDist — Time complexity

1. Computing convex hull (by using Delaunay triangulation) takes time $O(n \log n)$.
2. Finding the optimal line with a fixed direction takes time $O(1)$.
3. There are $O(n)$ interesting directions.

Exercise: MiniDist — Time complexity

1. Computing convex hull (by using Delaunay triangulation) takes time $O(n \log n)$.
2. Finding the optimal line with a fixed direction takes time $O(1)$.
3. There are $O(n)$ interesting directions.
4. The overall time complexity is thus $O(n \log n)$.