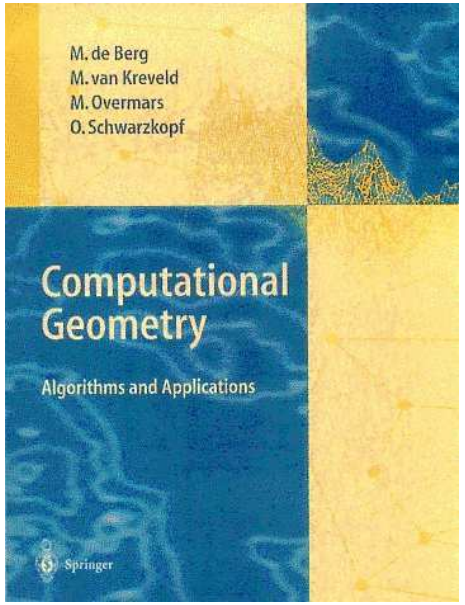
Geometrische Algorithmen

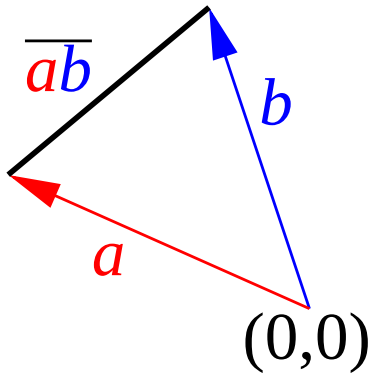
- Womit beschäftigen sich geom. Algorithmen?
- Schnitt von Strecken: Bentley-Ottmann-Algorithmus
- Weitere Beispiele

Quelle:

[Computational Geometry – Algorithms and Applications
de Berg, van Kreveld, Overmars, Schwartzkopf
Springer, 1997]



Elementare Geometrische Objekte



Punkte: $x \in \mathbb{R}^d$

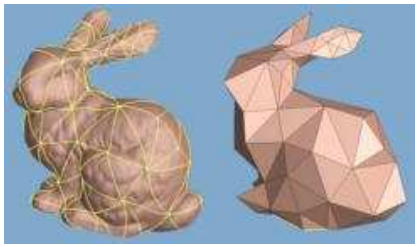
Strecken: $\overline{ab} := \{\alpha a + (1-\alpha)b : \alpha \in [0,1]\}$

uvam: Halbräume, Ebenen, Kurven,...

Dimension d:

1: Oft trivial. Gilt i. allg. nicht als geometrisches Problem

2: Geogr. Informationssysteme (GIS), Bildverarbeitung,...



3: Computergrafik, Simulationen,...

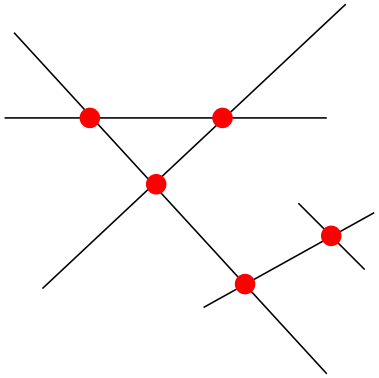
≥ 4: Optimierung, Datenbanken, maschinelles Lernen,...

curse of dimensionality!

n: Anzahl vorliegender Objekte

Typische Fragestellungen

- Schnittpunkte zwischen n Strecken



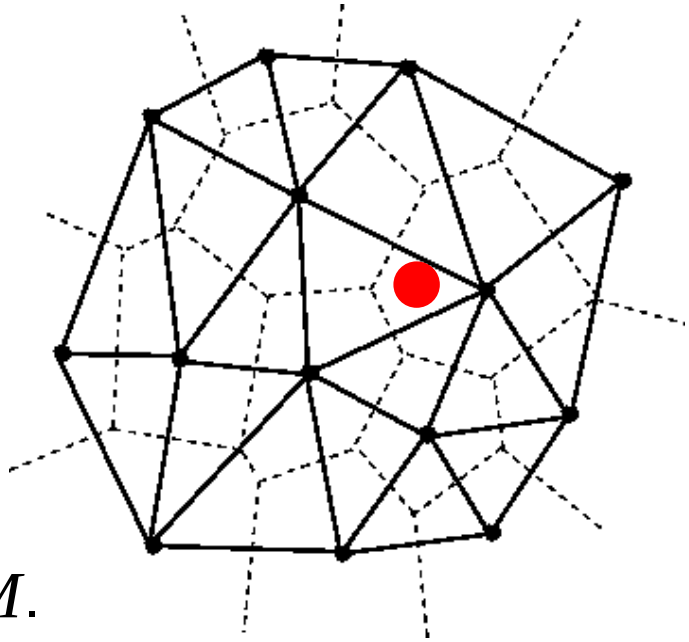
Typische Fragestellungen

- Schnittpunkte zwischen n Strecken
- Konvexe Hülle
- Triangulation von Punktmengen

  (2D, verallgemeinerbar)

  z.B. Delaunaytriangulierung:

  Kein Dreieck enthält weiteren Punkt



- Voronoi-Diagramme: Sei $x \in M \subseteq \mathbb{R}^d$.

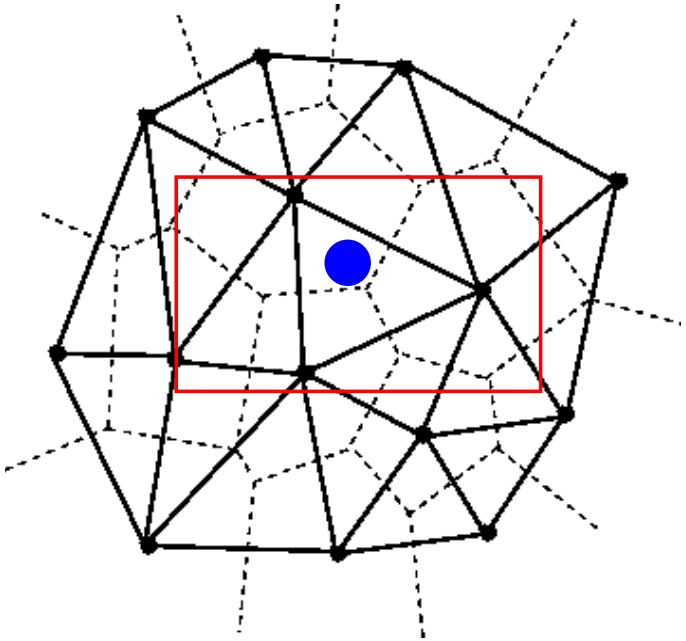
  $\forall y \in \mathbb{R}^d$ bestimme nächstes Element aus M.

  (Unterteilung von M in n Voronoizellen)
- Punklokalisierung: Geg. Unterteilung von R^d, $x \in R^d$:

  in welchem Teil liegt x ?

Datenstrukturen für Punktmengen

- nächsten Nachbarn berechnen
- Bereichsanfragen
- ...



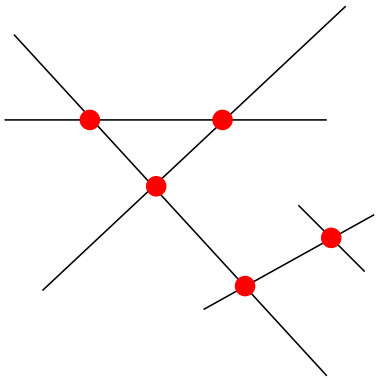
Mehr Fragestellungen

- Sichtbarkeitsberechnungen
- Lineare Programmierung
- Geometrische Versionen von Optimierungsproblemen
  - Kürzeste Wege, z.B.

    energieffiziente Kommunikation in Radionetzwerken
  - minimale Spannbäume

    reduzierbar auf Delaunay-Triangulierung + Graphalgorithmus
  - Matchings
  - Handlungsreisendenproblem
  - ...
- ...

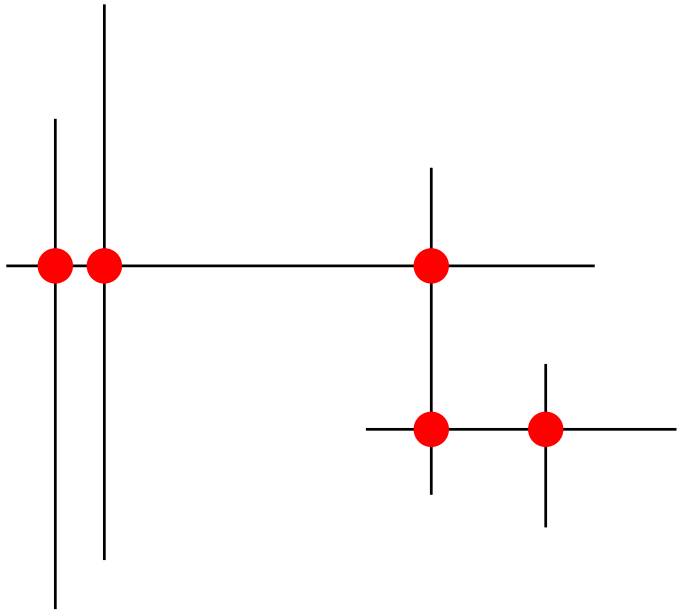
Streckenschnitt (line segment intersection)



Gegeben: $S = \{s_1, \ldots, s_n\}$, n Strecken

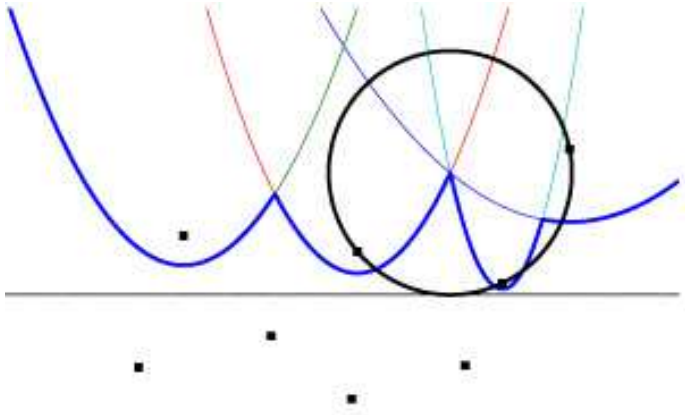
Gesucht: Schnittpunkte $\bigcup_{s,t \in S} s \cap t$

Zum Warmwerden: Orthogonaler Streckenschnitt – die Strecken sind parallel zur x- oder y-Achse



Streckenschnitt: Anwendungen

- Schaltungsentwurf: wo kreuzen sich Leiterbahnen?
- GIS: Strassenkreuzungen, Brücken,. . .
- Erweiterungen: z.B. Graphen benachbarter Strecken/Flächen aufbauen/verarbeiten
- Noch allgemeiner:
  Plane-Sweep-Algorithmen
  für andere Fragestellungen
  (z.B. Konstruktion von
  konvexen Hüllen oder
  Voronoi-diagrammen)



Streckenschnitt: Naiver Algorithmus

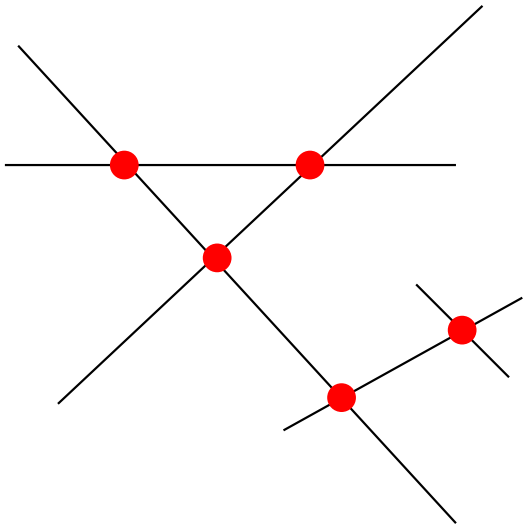
foreach $\{s,t\} \subseteq S$ **do**

  if $s \cap t \neq \emptyset$ **then**

    output $\{s,t\}$

Problem: Laufzeit $\Theta(n^2)$.

Zu langsam für große Datenmengen

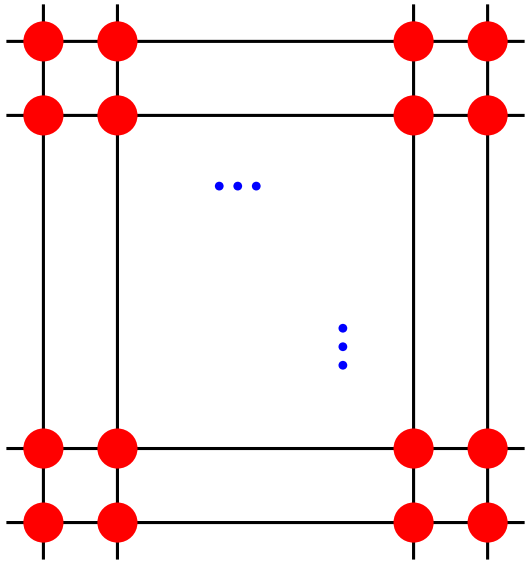


Streckenschnitt: Untere Schranke

$\Omega(n+k)$ mit k:= Anzahl ausgegebener Schnitte.

Vergleichsbasiert: $\Omega(n \log n + k)$ (Beweis: nicht hier)

Beobachtung $k = \Theta(n^2)$ ist möglich, aber reale Eingaben haben meist $k = \mathrm{O}(n)$.

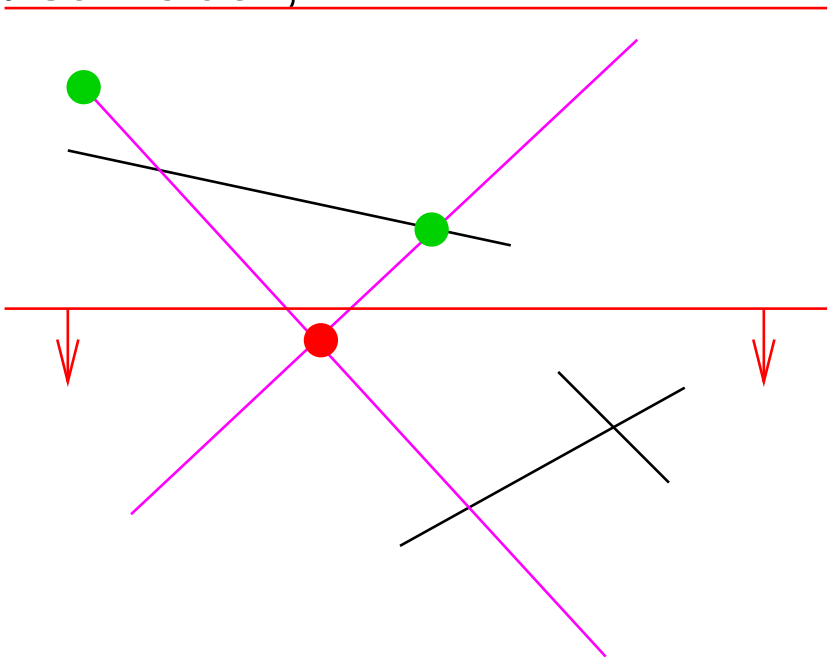


Idee: Plane-Sweep-Algorithmen

(Waagereche) Sweep-Line ℓ läuft von oben nach unten.

Invariante: Schnittpunkte oberhalb von ℓ wurden korrekt ausgegeben

Beobachtung: Nur Segmente, die sich mit ℓ schneiden,
müssen neu ausgegeben werden.



Plane-Sweep für orth. Streckenschnitt

$T = \langle\rangle$: SortedSequence **of** Segment

invariant T stores the vertical segments intersecting ℓ

Q:= sort$(\langle (y,s) : \exists$hor seg. @ y or $\exists$vert. seg. starting/ending @$y \rangle)$

  //tie breaking: vert. starting events first, vert. finishing events last

foreach $(y,s) \in Q$ in descending order **do**

  if vertical segment s starts at y **then** T.insert(s)

  else if vertical segment s ends at y **then** T.remove(s)

  else //we have a horizontal segment $s = \overline{(x_1,y)(x_2,y)}$

    foreach $t = \overline{(x,y_1)(x,y_2)} \in T$ with $x \in [x_1,x_2]$ **do**

      output $\{s,t\}$

    handle horizontal segments on ℓ// interval intersection problem

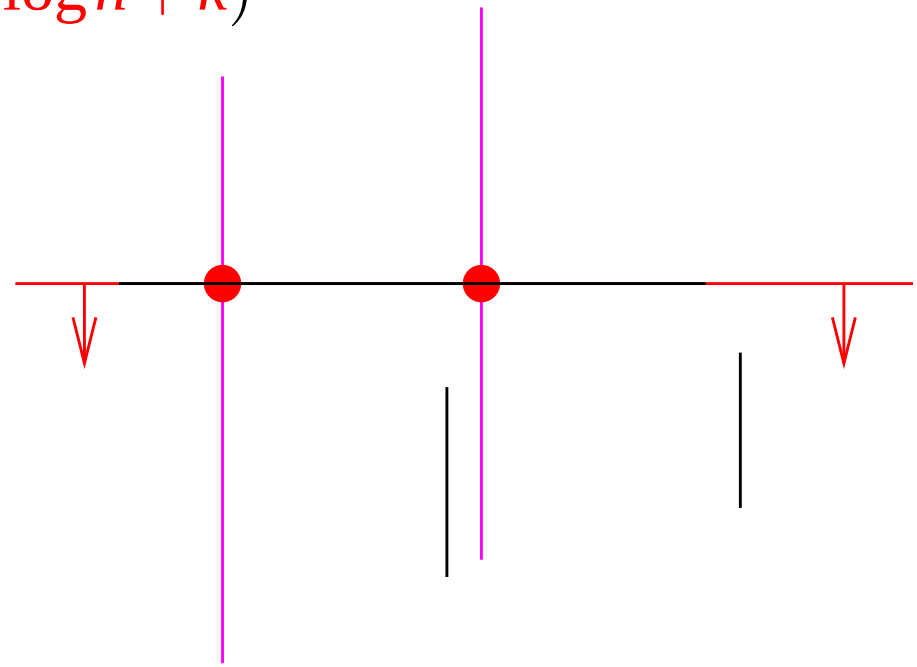
Analyse orth. Streckenschnitt

insert: $O(\log n)$ $(\leq n\times)$

remove: $O(\log n)$ $(\leq n\times)$

rangeQuery: $O(\log n + k_s)$, k_s Schnitte mit hor. Segment s

Insgesamt: $O(n\log n + \sum_s k_s) = O(n\log n + k)$



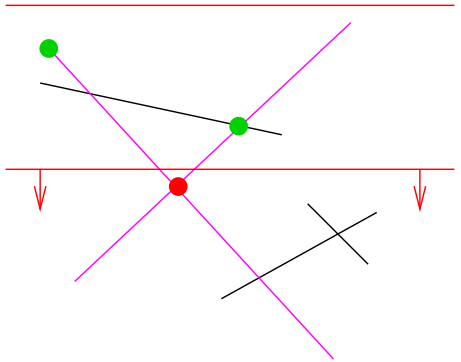
Verallgemeinerung – aber erstmal “nicht ganz”

Annahme zunächst:

Allgemeine Lage, d.h. hier

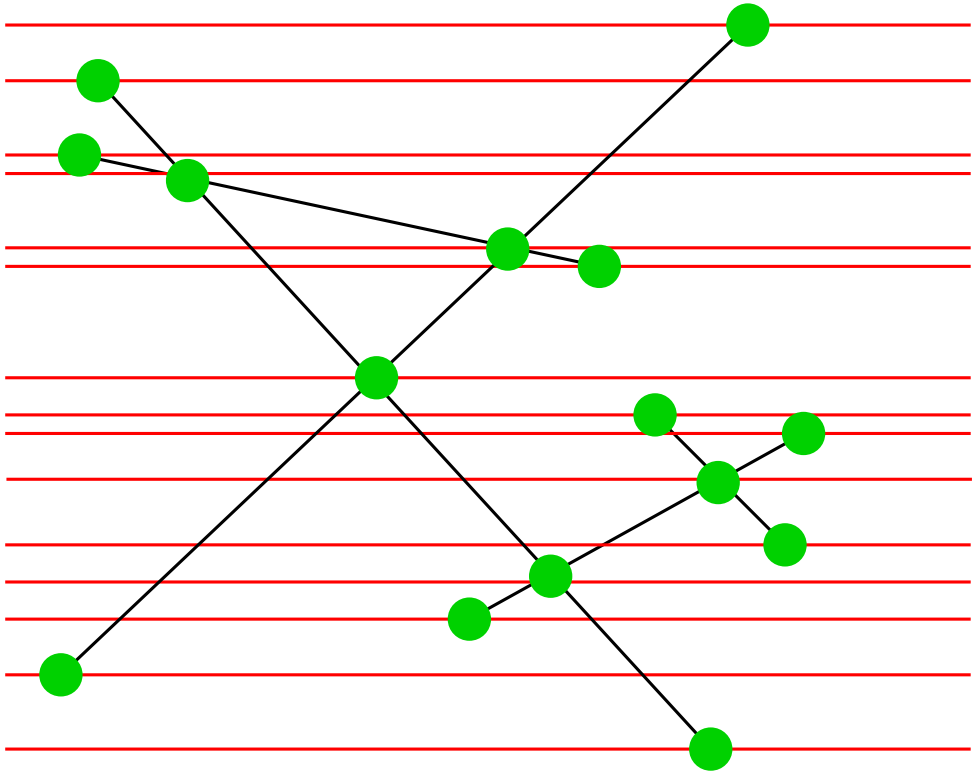
- Keine horizontalen Strecken
- Keine Überlappungen
- Schnittpunkte jeweils im Inneren von genau zwei Strecken

Beobachtung: kleine zuf. Perturbationen produzieren allg. Lage.



Verallgemeinerung – Grundidee

- Plane-Sweep mit Sweep-Line ℓ
- Status T := nach x geordnete Folge der ℓ schneidenden Strecken
- Ereignis:= Statusänderung
  - Startpunkte
  - Endpunkte
  - Schnittpunkte
- Schnitttest nur für Segmente, die an einem Ereignispunkt in T benachbart sind.



Verallgemeinerung – Korrektheit

Lemma:

$s \cap t = \{(x,y)\} \longrightarrow \exists$ Ereignis : s,t werden Nachbarn auf ℓ

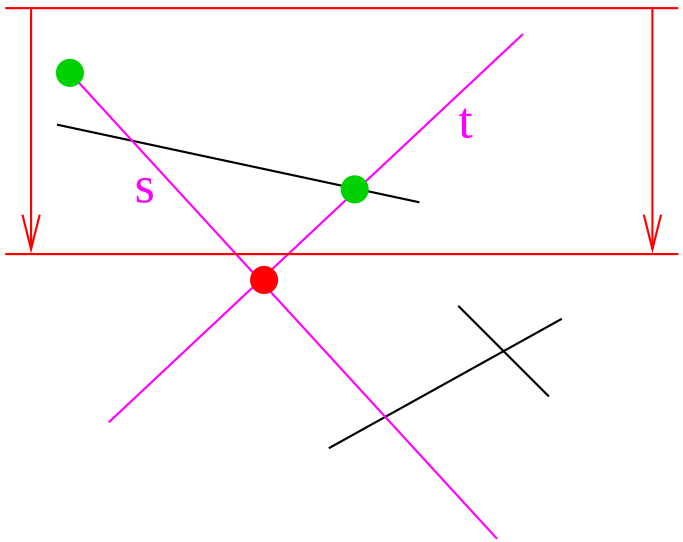
Beweis:

Anfangs : $T = \langle\rangle \longrightarrow s,t$ sind nicht in ℓ benachbart.

@$y+\varepsilon$: s,t sind in ℓ benachbart.

$\longrightarrow$

$\exists$ Ereignis bei dem s und t Nachbarn werden.



Verallgemeinerung – Implementierung

$T = \langle\rangle$: SortedSequence **of** Segment

invariant T stores the relative order of the seqments intersecting ℓ

Q : MaxPriorityQueue

$Q := Q \cup \left\{ (y, \text{start}, s) : s = \overline{(x,y)(x',y')} \in S, y \leq y' \right\}$ // $O(n \log n)$

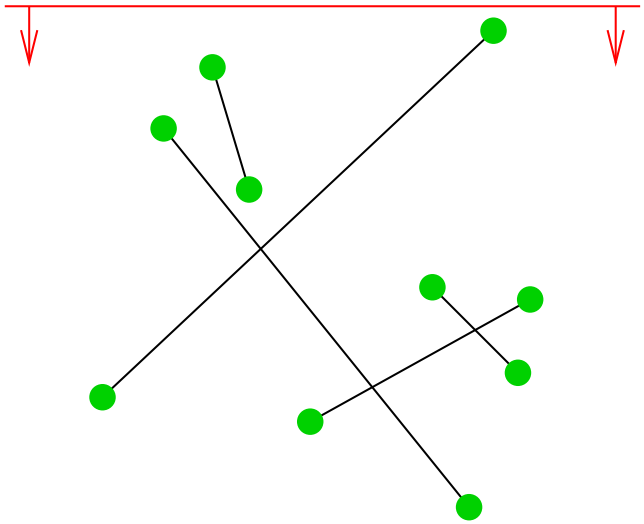
$Q := Q \cup \left\{ (y, \text{finish}, s) : s = \overline{(x,y)(x',y')} \in S, y \geq y' \right\}$ // $O(n \log n)$

while $Q \neq \emptyset$ **do**

$(y, \text{type}, s) := Q.\text{deleteMin}$

// $O((n+k)\log n)$

handleEvent$(y, \text{type}, s, T, Q)$



handleEvent$(y, \text{start}, s, T, Q)$ // $n\times$

h:= T.insert(s) // $O(\log n)$

prev:= pred(h) // $O(1)$

next:= succ(h) // $O(1)$

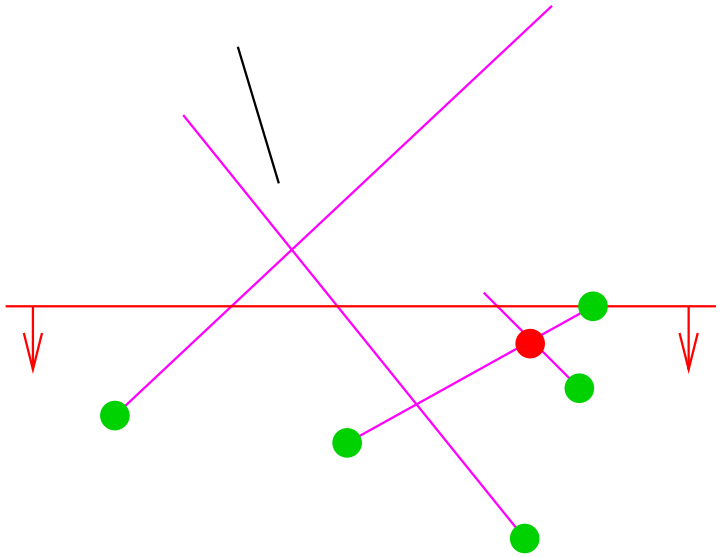
findNewEvent(prev, h)

findNewEvent(h, next)

Procedure findNewEvent(s,t) // $O(1+\log n)$

if $*s$ and $*t$ intersect at $y' > y$ **then**

Q.insert$((y', \text{intersection}, (s,t)))$



handleEvent$(y, \text{finish}, s, T, Q)$ // $n\times$

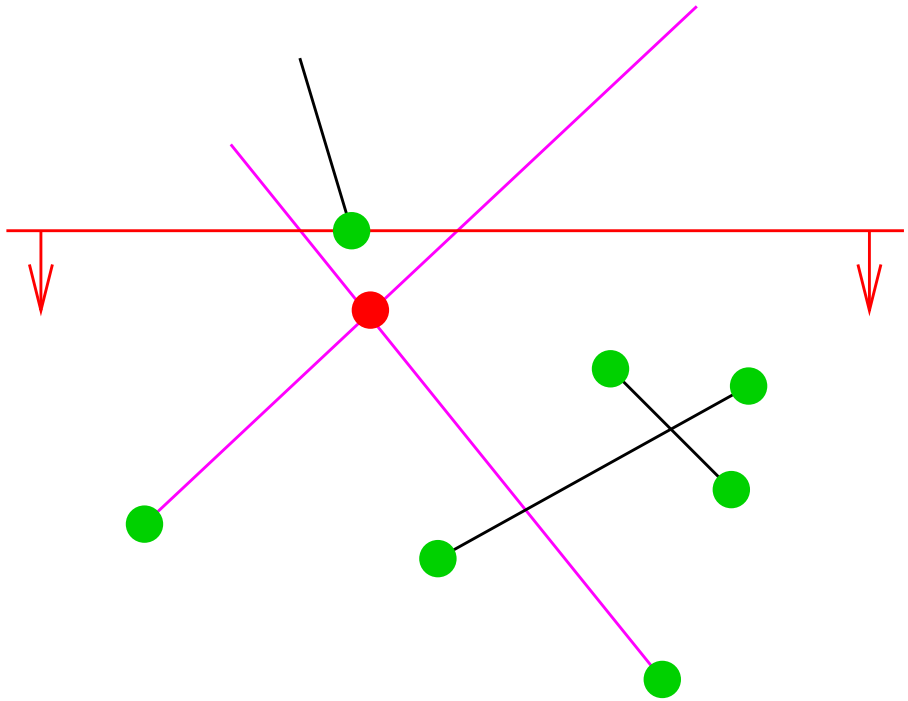
h:= T.locate(s) // $O(n \log n)$

prev:= pred(h) // $O(n)$

next:= succ(h) // $O(n)$

T.remove(s) // $O(n)$

findNewEvent(prev, next)



handleEvent$(y, \text{intersection}, (a,b), T, Q)$ // $k\times$

output $(*s \cap *t)$ // $O(k)$

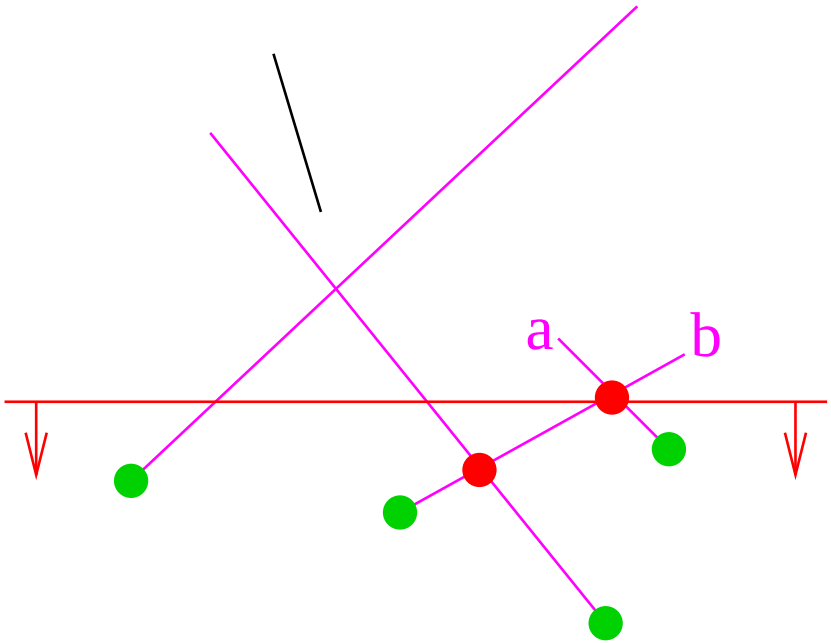
T.swap(a,b) // $O(k)$

prev:= pred(b) // $O(k)$

next:= succ(a) // $O(k)$

findNewEvent(prev, b)

findNewEvent(a, next)



Verallgemeinerung – Analyse

Insgesamt: $O((n+k)\log n)$

Konstante Faktoren falls $k \gg n$:

$(n+k)\times$ insert, deleteMin

in Prioritätsliste.

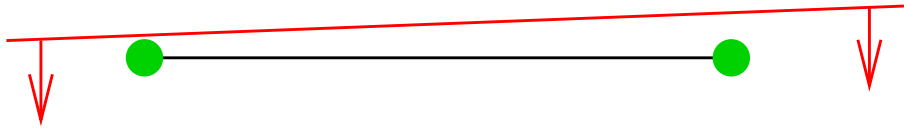
Verallgemeinerung – jetzt (fast) wirklich

Verbleibende Annahme: Keine Überlappungen

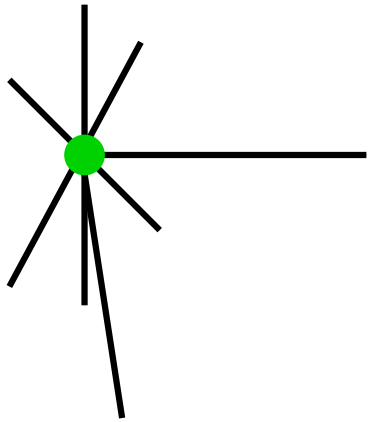
Ordnung für Q : $(x,y) \prec (x',y') \Leftrightarrow y > y' \vee y = y' \wedge x < x'$

(verquere lexikographische Ordnung)

Interpretation: infinitesimal ansteigende Sweep-Line



Q speichert Mengen von Ereignissen mit gleichem (x,y)



handleEvent$(p = (x,y))$

U:= segments starting at p // from Q

C:= segments with p in their interior // from T

L:= segments finishing at p // from Q

if $|U| + |C| + |L| \geq 2$ **then** report intersection @ p

T.remove$(L \cup C)$

T.insert$(C \cup U)$ such that order just below p is correct

if $U \cup C = \emptyset$ **then**

findNewEvent$(T$.findPred$(p), T$.findSucc$(p), p)$

else

findNewEvent$(T$.findPred$(p), T$.findLeftmost$(p), p)$

findNewEvent$(T$.findRightmost$(p), T$.findSucc$(p), p)$

findNewEvent(s,t,p)

if s and t intersect at a point $p' \succ p$ **then**

if $p' \notin Q$ **then** Q.insert(p')

Überlappungen finden

Für jede Strecke s berechne die Gerade $g(s)$, auf der s liegt

Sortiere S nach $g(s)$

1D Überlappungsproblem für jede auftretende Gerade.

Platzverbrauch

Im Moment: $\Theta(n+k)$

Reduktion auf $\mathrm{O}(n)$:

lösche Schnittpunkte zwischen nicht benachbarten Strecken aus T.

Die werden ohnehin wieder eingefügt wenn sie wieder benachbart werden.

Mehr Linienschnitt

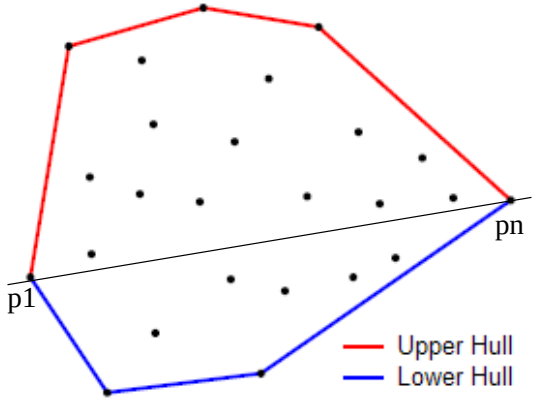
- [Bentley Ottmann 1979] Zeit $O((n+k)\log n)$
- [Chazelle Edelsbrunner 1988] Zeit $O(n\log n+k)$
- [Pach Sharir 1991] Zeit $O((n+k)\log n)$, Platz $O(n)$
- [Mulmuley 1988] erwartete Zeit $O(n\log n+k)$, Platz $O(n)$
- [Balaban 1995] Zeit $O(n\log n+k)$, Platz $O(n)$

2D Konvexe Hülle

Gegeben: Menge $P = \{p_1, \ldots, p_n\}$ von Punkten in $\mathbb{R}^2$

Gesucht: Konvexes Polygon K mit Eckpunkten aus P und $P \subseteq K$.

Wir geben einen einfachen Algorithmus, der in Zeit $O(\text{sort}(n))$ läuft.



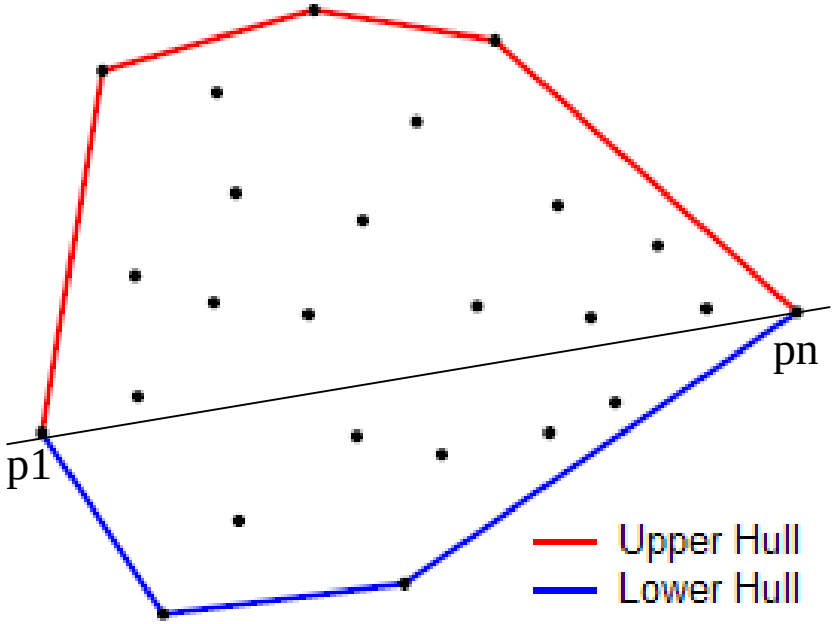
Konvexe Hülle

sortiere P lexikographisch, d.h., ab jetzt

$p_1 < p_2 < \cdots < p_n$

OBdA:

Wir berechnen nur die obere Hülle von Punkten oberhalb von $\overline{p_1 p_n}$



Graham's Scan [Graham 1972, Andrew 1979]

Function upperHull$(p_1, \ldots, p_n)$

$L = \langle p_n, p_1, p_2 \rangle$: Stack **of** Point

invariant L is the upper hull of $\langle p_1, \ldots, p_i \rangle$

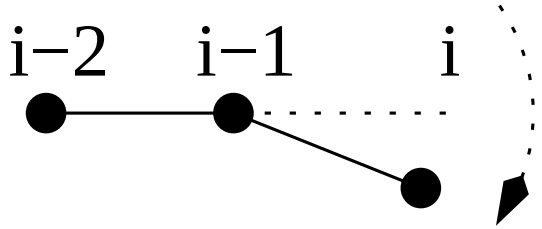
for $i := 3$ **to** n **do**

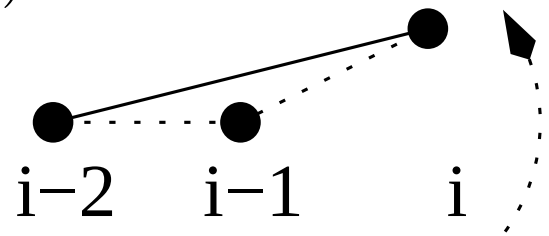
while $\neg$rightTurn$(L.\text{secondButlast}, L.\text{last}, p_i)$ **do**

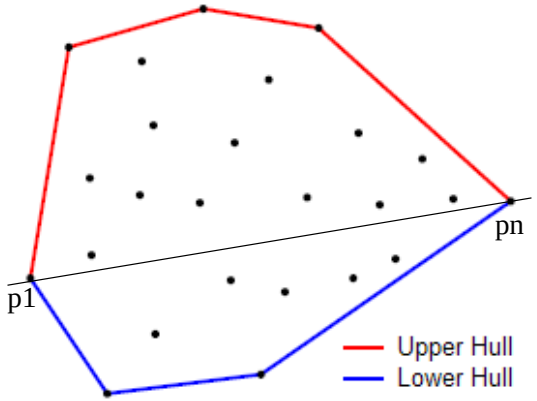
L.pop

$L := L \circ \langle p_i \rangle$

return L

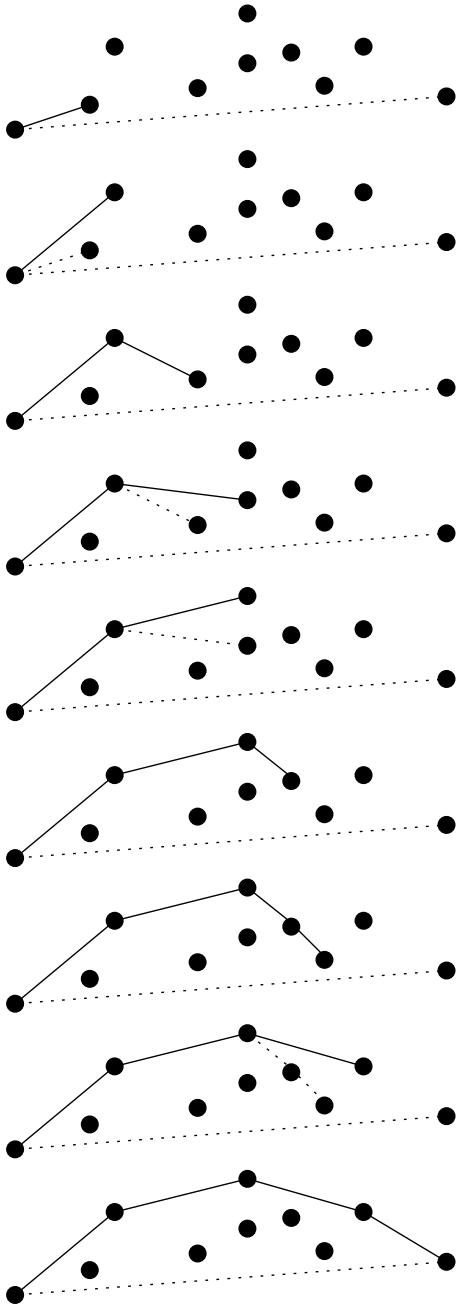






Graham's Scan – Beispiel

Function upperHull$(p_1, \ldots, p_n)$
  $L = \langle p_n, p_1, p_2 \rangle$: Stack **of** Point
  invariant L is the upper hull of $\langle p_1, \ldots, p_i \rangle$
  for $i := 3$ **to** n **do**
    while ¬rightTurn$(L$.secondButlast, L.last, $p_i)$ **do**
      L.pop
    $L := L \circ \langle p_i \rangle$
  return L



Graham's Scan – Analyse

Function upperHull$(p_1, \ldots, p_n)$

$L = \langle p_n, p_1, p_2 \rangle$: Stack **of** Point

invariant L is the upper hull of $\langle p_1, \ldots, p_i \rangle$

for $i := 3$ **to** n **do**

while $\neg$rightTurn$(L$.secondButlast$, L$.last$, p_i)$ **do**

L.pop

$L := L \circ \langle p_i \rangle$

return L

Sortieren $+\mathrm{O}(n)$

Wieviele Iterationen der While-Schleife insgesamt?

3D Konvexe Hülle

Geht in Zeit $O(n \log n)$ [Preparata Hong 1977]

Konvexe Hülle, $d \geq 4$

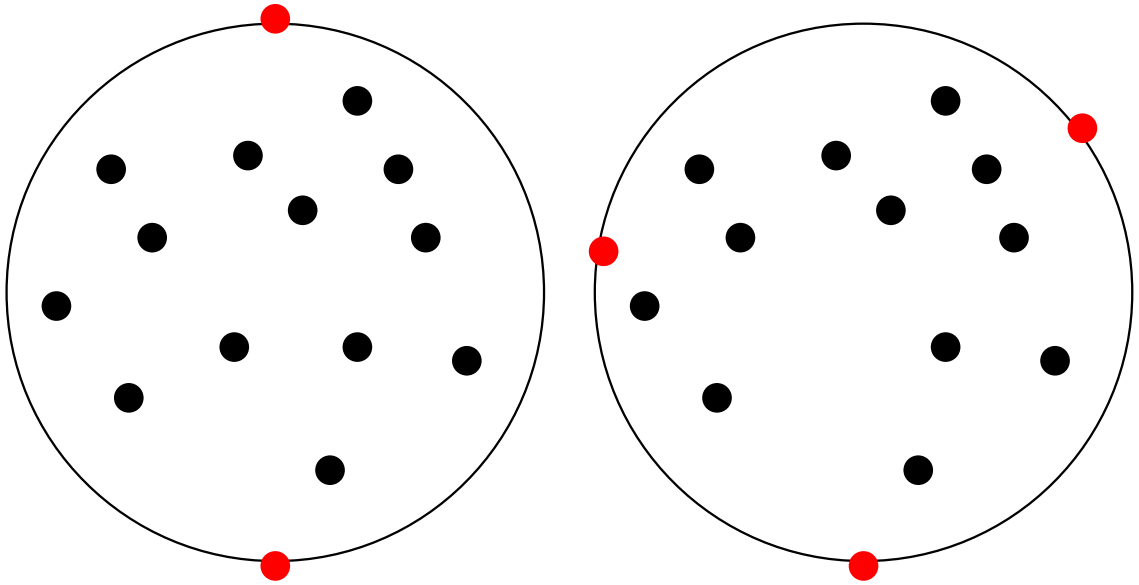
Ausgabekomplexität $O\left(n^{\lfloor d/2 \rfloor}\right)$

Kleinste einschließende Kugel

Gegeben: Menge $P = \{p_1, \ldots, p_n\}$ von Punkten in $\mathbb{R}^d$

Gesucht: Kugel K mit minimalem Radius, so dass $P \subseteq K$.

Wir geben einen einfachen Algorithmus, der in erwarteter Zeit $O(n)$ läuft. [Welzl 1991].



Function smallestEnclosingBallWithPoints(P, Q)

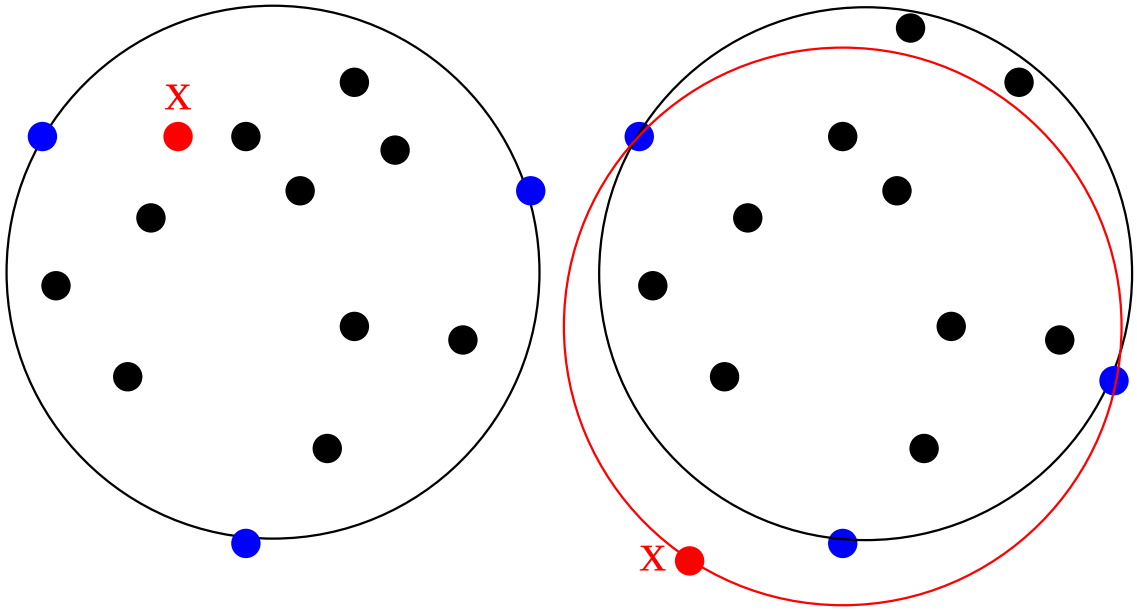
if $|P| = 1 \vee |Q| = d+1$ **then return** ball(Q)

pick random $x \in P$

B:= smallestEnclosingBallWithPoints$(P \setminus \{x\}, Q)$

if $x \in B$ **then return** B

return smallestEnclosingBallWithPoints$(P, Q \cup \{x\})$



Kleinste einschließende Kugel – Korrektheit

Function smallestEnclosingBallWithPoints(P, Q)

if $|P| = 1 \vee |Q| = d+1$ **then return** ball(Q)

pick random $x \in P$

B:= smallestEnclosingBallWithPoints$(P \setminus \{x\}, Q)$

if $x \in B$ **then return** B

return smallestEnclosingBallWithPoints$(P, Q \cup \{x\})$

z.Z.: $x \notin B \rightarrow x$ ist auf dem Rand von sEB(P)

Wir zeigen Kontraposition:

x nicht auf dem Rand von sEB(P)

$\longrightarrow$ sEB(P) = sEB$(P \setminus \{x\}) = B$ z.Z.: sEBs sind eindeutig!

Also $x \in B$

Lemma: sEB(P) ist eindeutig bestimmt.

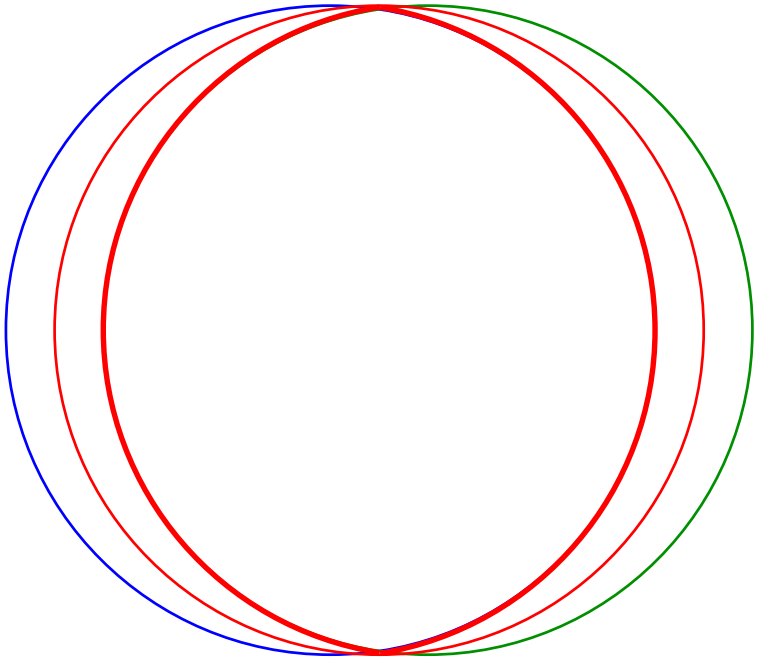
Beweis: Annahme, $\exists$sEBs $B_1 \neq B_2$

$\longrightarrow P \subseteq B_1 \wedge P \subseteq B_2$

$\longrightarrow P \subseteq B_1 \cap B_2 \subseteq \text{sEB}(B_1 \cap B_2) =: B$

Aber dann ist radius$(B) <$ radius(B_1)

Widerspruch zur Annahme, dass B_1 eine sEB ist. $\square$



Kleinste einschließende Kugel – Analyse

Wir zählen die erwartete Anzahl der Tests $x \in B$, $T(p,q)$.

$$T(p,d+1) = T(1,p) = 0 \qquad \text{Basis der Rekurrenz}$$

$$\begin{aligned} T(p,q) &\leq 1 + T(p-1,q) + \mathbb{P}[x \notin B]\, T(p,q+1) \\ &\leq 1 + T(p-1,q) + \frac{d+1-q}{p} T(p,q+1) \end{aligned}$$

Kleinste einschließende Kugel – Analyse, $d = 2$

$T(p, d+1) = T(1, p) = 0$

$$T(p,q) \leq 1 + T(p-1,q) + \frac{d+1-q}{p} T(p,q+1)$$

$$T(p,2) \leq 1 + T(p-1,2) + \frac{1}{p} T(p,3) \leq 1 + T(p-1,2) \leq p$$

$$T(p,1) \leq 1 + T(p-1,1) + \frac{2}{p} T(p,2)$$

$$\leq 1 + T(p-1,1) + \frac{2}{p} p = 3 + T(p-1,1) \leq 3p$$

$$T(p,0) \leq 1 + T(p-1,0) + \frac{3}{p} T(p,1)$$

$$\leq 1 + T(p-1,0) + \frac{3}{p} 3p = 10 + T(p-1,0) \leq 10p$$

Kleinste einschließende Kugel – Analyse

d	$T(p,0)$
1	$3n$
2	$10n$
3	$41n$
4	$206n$

Allgemein $T(p,0) \geq d!n$

Ähnliche Randomisierte Linearzeitalgorithmen

- Lineare Programmierung mit konstantem d [Seidel 1991]
- Kleinster einschließender Ellipsoid, Kreisring,. . .
- Support-Vector-Machines (maschinelles Lernen)
- Alles wo (LP-type problem [Sharir Welzl 1992])
  - $O(1)$ Objekte das Optimum festlegen
  - Objekt x hinzufügen

    $\longrightarrow$ Lösung bleibt gleich oder ist an Lösungsdef. beteiligt

14 Onlinealgorithmen

15 Exponentialzeit- und Fixed-Parameter-Algorithmen

16 Randomisierte Algorithmen und average-case Analyse

17 Kombinatorische Optimierung

18 Parallele Algorithmen

19 Algorithmen für Speicherhierarchien

20 Algorithmische Spieltheorie

21 Schedulingalgorithmen

22 Algorithmen für planare Graphen und Graph-Drawing

23 Succinct Data Structures

24 Algorithmische Lerntheorie, Data Mining, Datenbanken, Information Retrieval