Kapitel 9
Algorithm. Geometrie

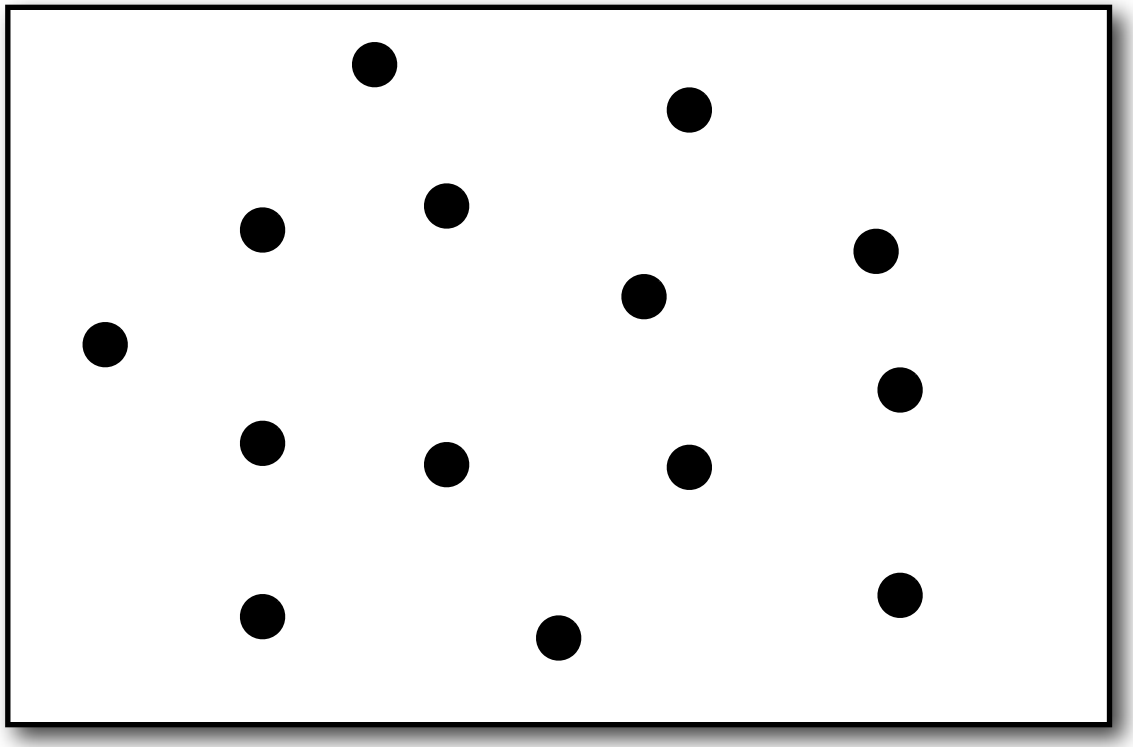
- Kürzeste Abstände
- Konvexe Hülle

Algorithmische Geometrie
Überblick

- Teilgebiet der Informatik, in dem es um die Entwicklung effizienter Algorithmen und die Bestimmung der algorithmischen Komplexität elementarer geometrischer Probleme geht

Algorithmische Geometrie

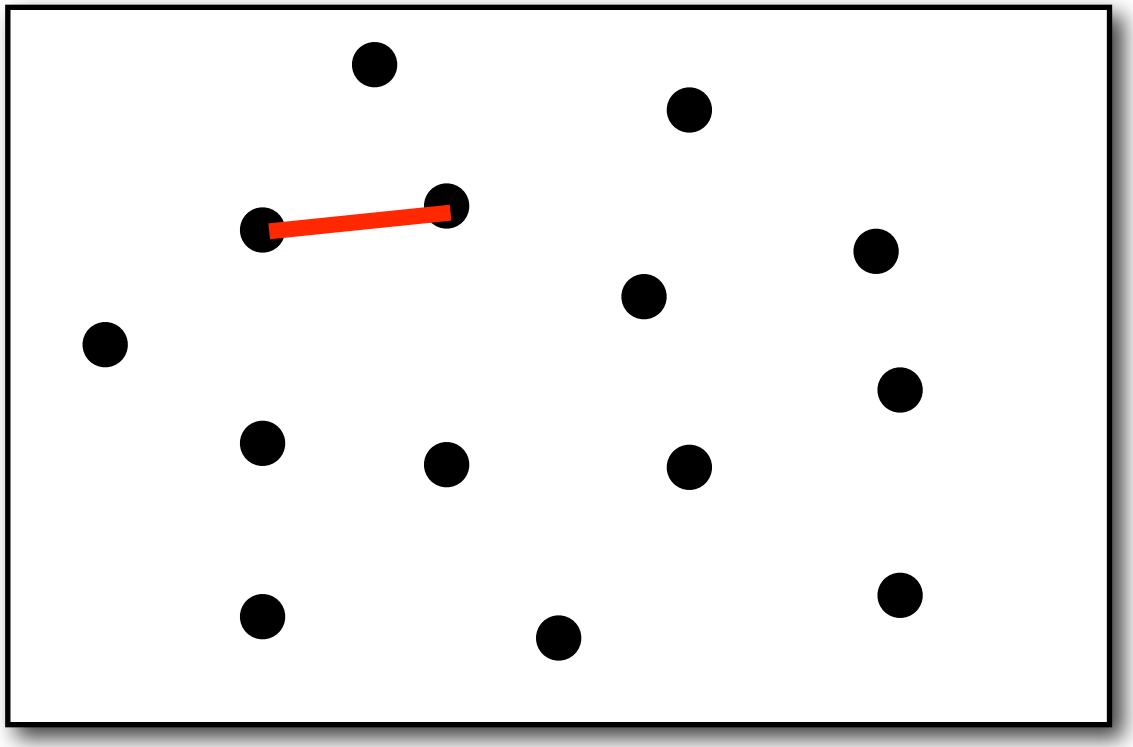
Kürzeste Abstände



- nearest neighbour problem - closest pair problem - closest points problem
- Anwendung z.B.: Radarbeobachtung, Flugsicherung

Algorithmische Geometrie

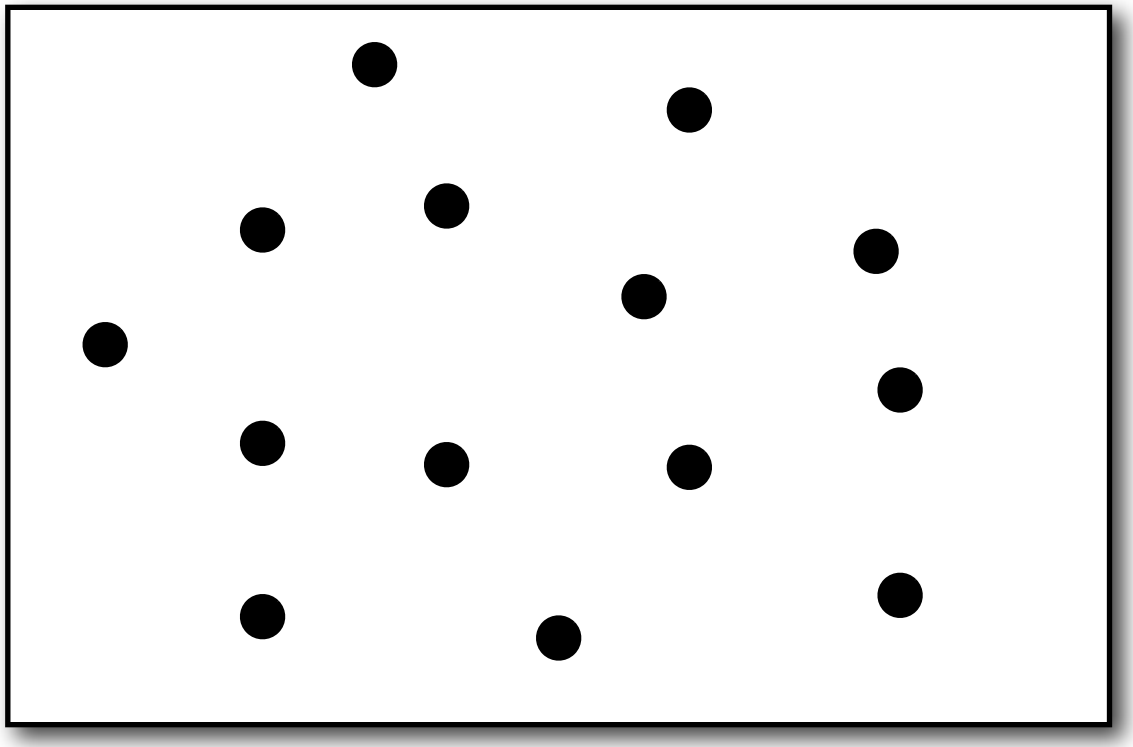
Kürzeste Abstände



- nearest neighbour problem - closest pair problem - closest points problem
- Anwendung z.B.: Radarbeobachtung, Flugsicherung

Algorithmische Geometrie

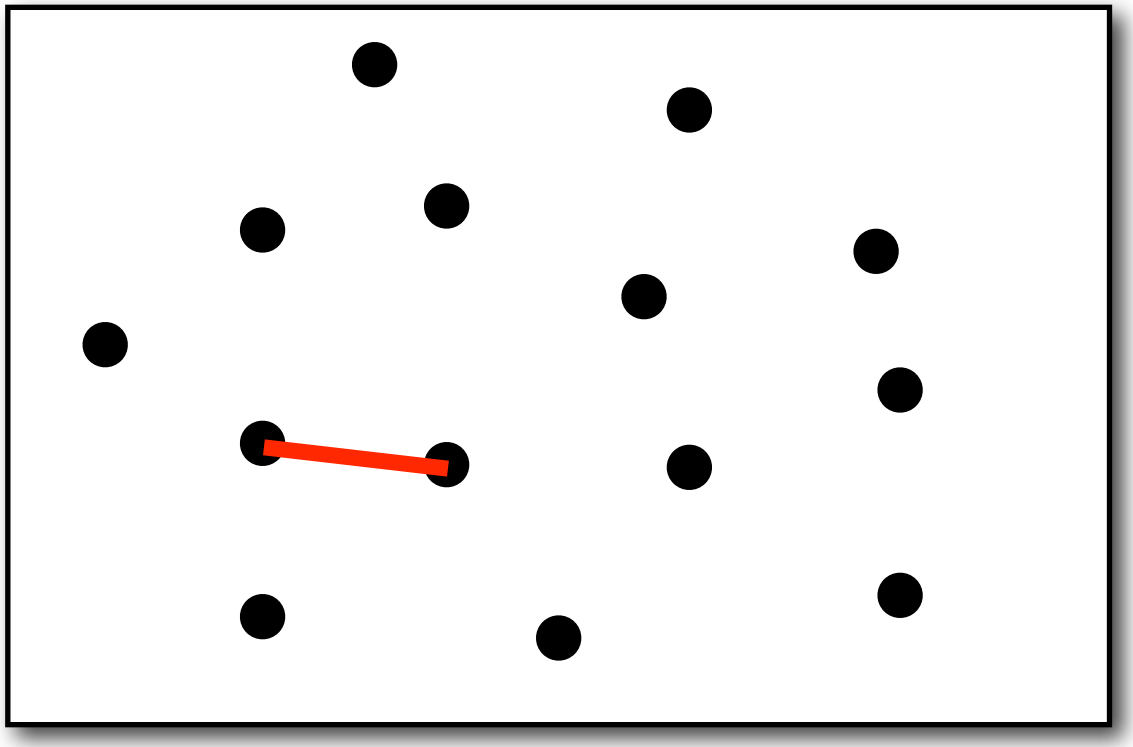
Kürzeste Abstände



- nearest neighbour problem - closest pair problem - closest points problem
- Anwendung z.B.: Radarbeobachtung, Flugsicherung

Algorithmische Geometrie

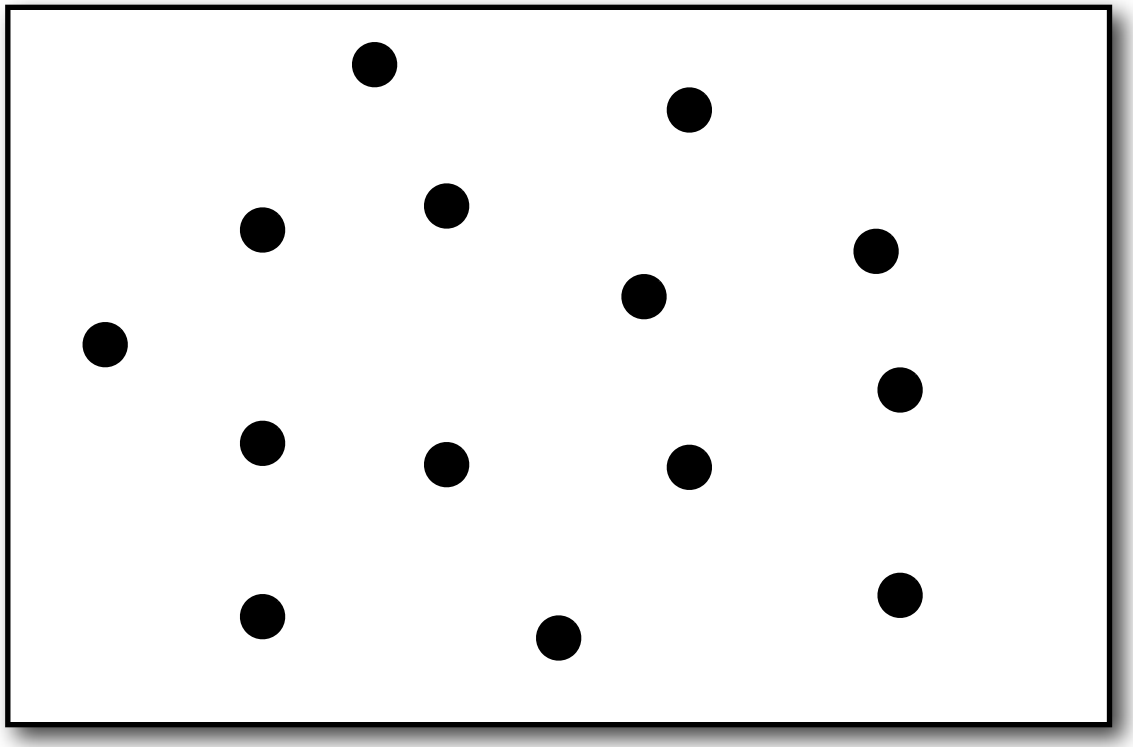
Kürzeste Abstände



- nearest neighbour problem - closest pair problem - closest points problem
- Anwendung z.B.: Radarbeobachtung, Flugsicherung

Algorithmische Geometrie

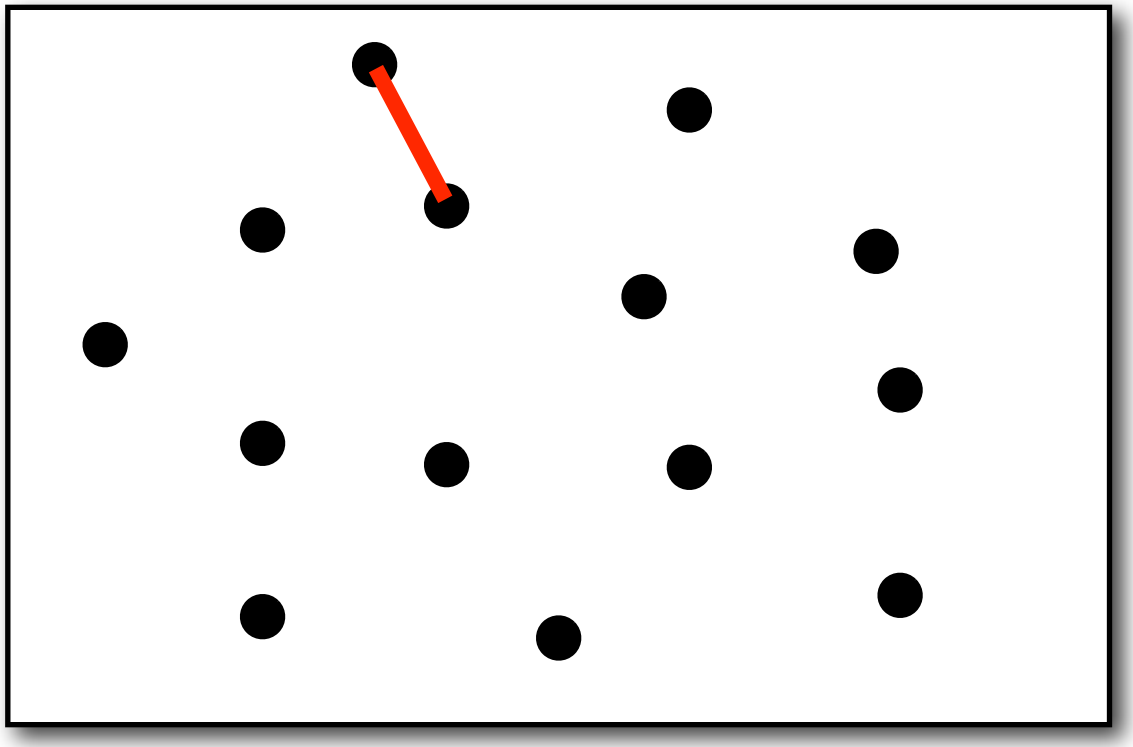
Kürzeste Abstände



- nearest neighbour problem - closest pair problem - closest points problem
- Anwendung z.B.: Radarbeobachtung, Flugsicherung

Algorithmische Geometrie

Kürzeste Abstände



- nearest neighbour problem - closest pair problem - closest points problem
- Anwendung z.B.: Radarbeobachtung, Flugsicherung

Algorithmische Geometrie
Kürzeste Abstände - Problemstellung

- Gegeben n Punkte $P_1=(x_1,y_1)$, ..., $P_n=(x_n,y_n)$, $x_i,y_i \in \mathbb{R}$, $n \geq 1$, in der Ebene: Man bestimme ein Paar von Punkten (x_j,y_j), (x_k,y_k), $j \neq k$, mit

$$d((x_j,y_j),(x_k,y_k)) \leq d((x_p,y_p),(x_q,y_q)) \text{ für alle } 1 \leq p < q \leq n$$

wobei Euklidische Metrik gilt:

$$d((x,y),(x',y')) = \sqrt{(x-x')^2+(y-y')^2}$$

Algorithmische Geometrie
Kürzeste Abstände - naive Lösung

- Brute-force-Methode:
  - bestimme den Abstand aller Punktepaare
  - gib den kleinsten Abstand aus
  - Laufzeit: $O(n^2)$
- Besserer (bester) Algorithmus: O(n log n) Laufzeit

Algorithmische Geometrie
algor. Paradigma - Plane sweeping

- zugrundliegendes universelles Muster:

Plane-sweeping

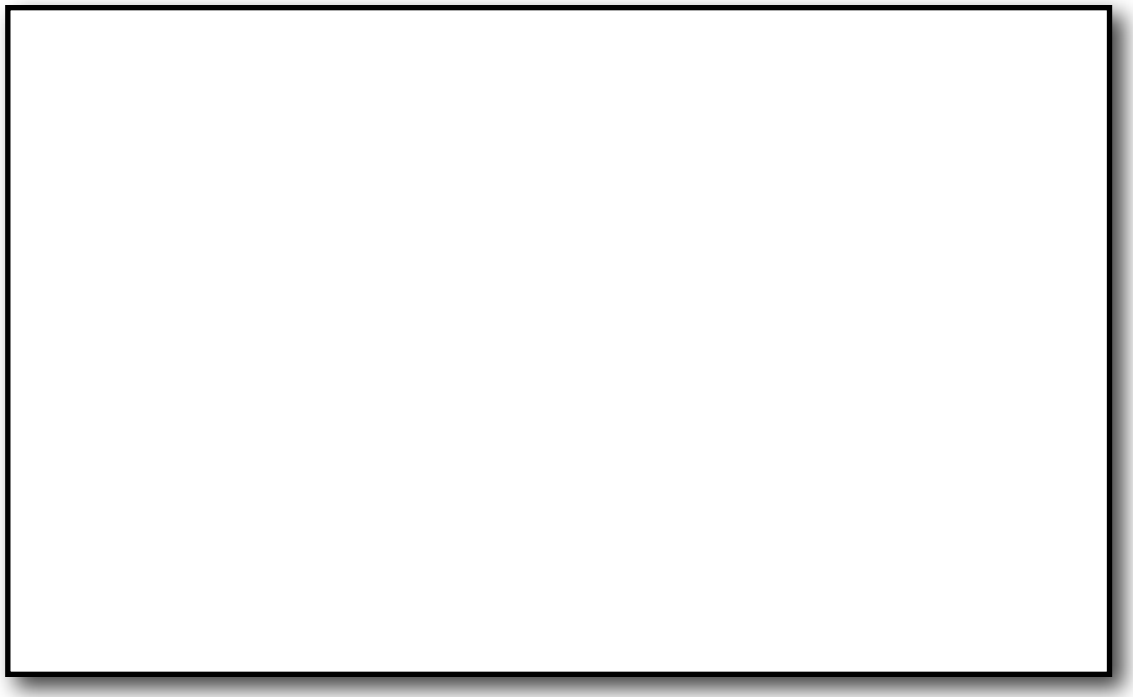
- Schema:

Algorithmische Geometrie
algor. Paradigma - Plane sweeping

- zugrundliegendes universelles Muster:

Plane-sweeping

- Schema:

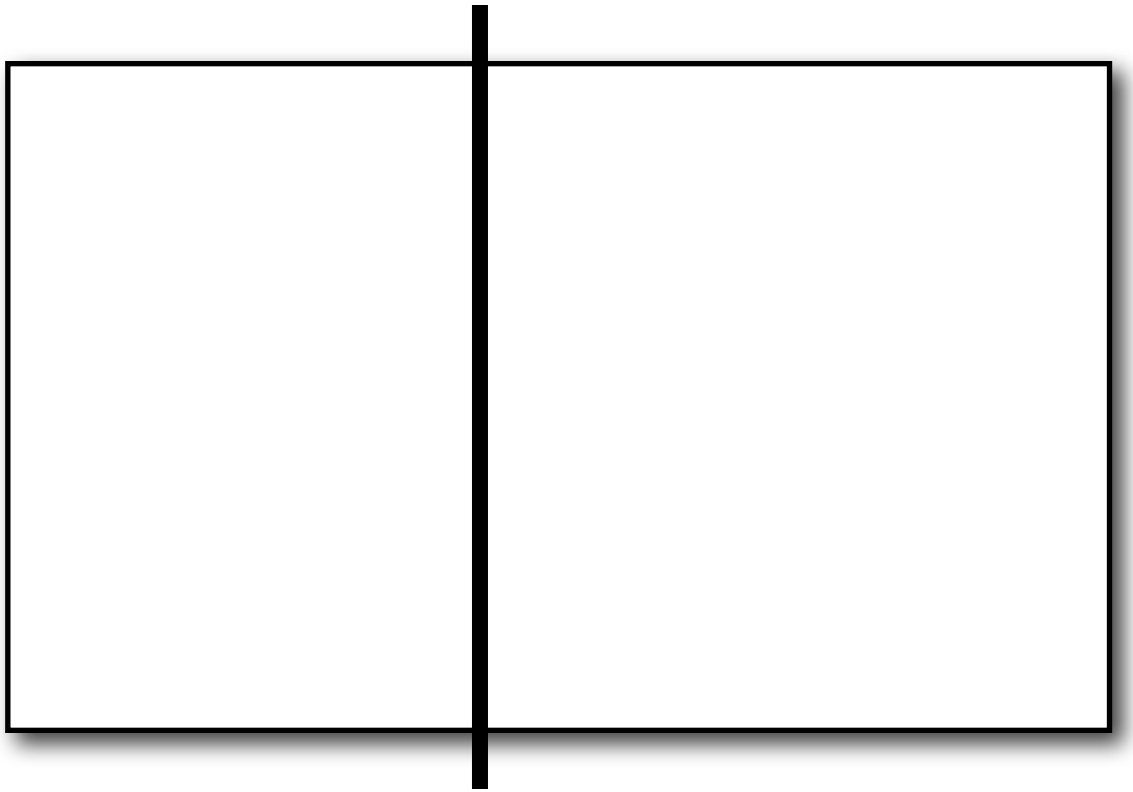


Algorithmische Geometrie
algor. Paradigma - Plane sweeping

- zugrundliegendes universelles Muster:

Plane-sweeping

- Schema:

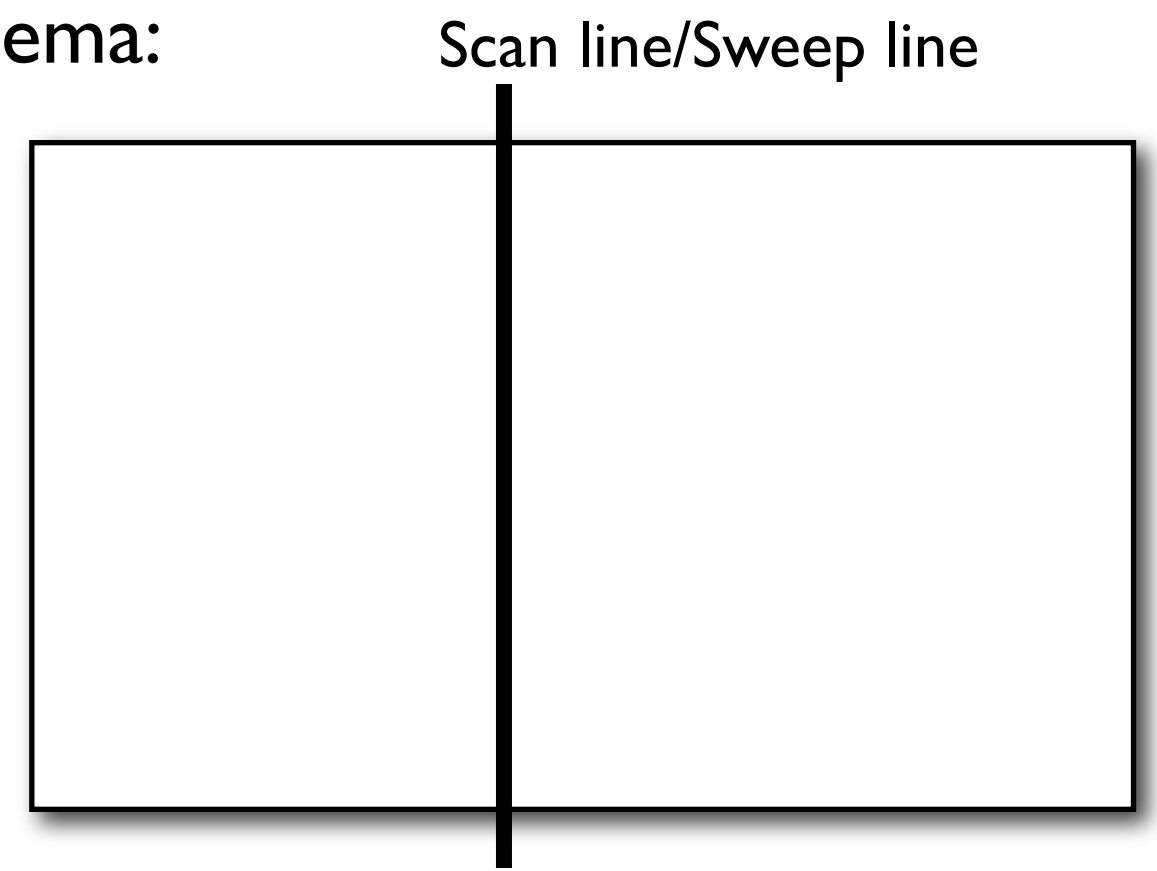


Algorithmische Geometrie
algor. Paradigma - Plane sweeping

- zugrundliegendes universelles Muster:

Plane-sweeping

- Schema:

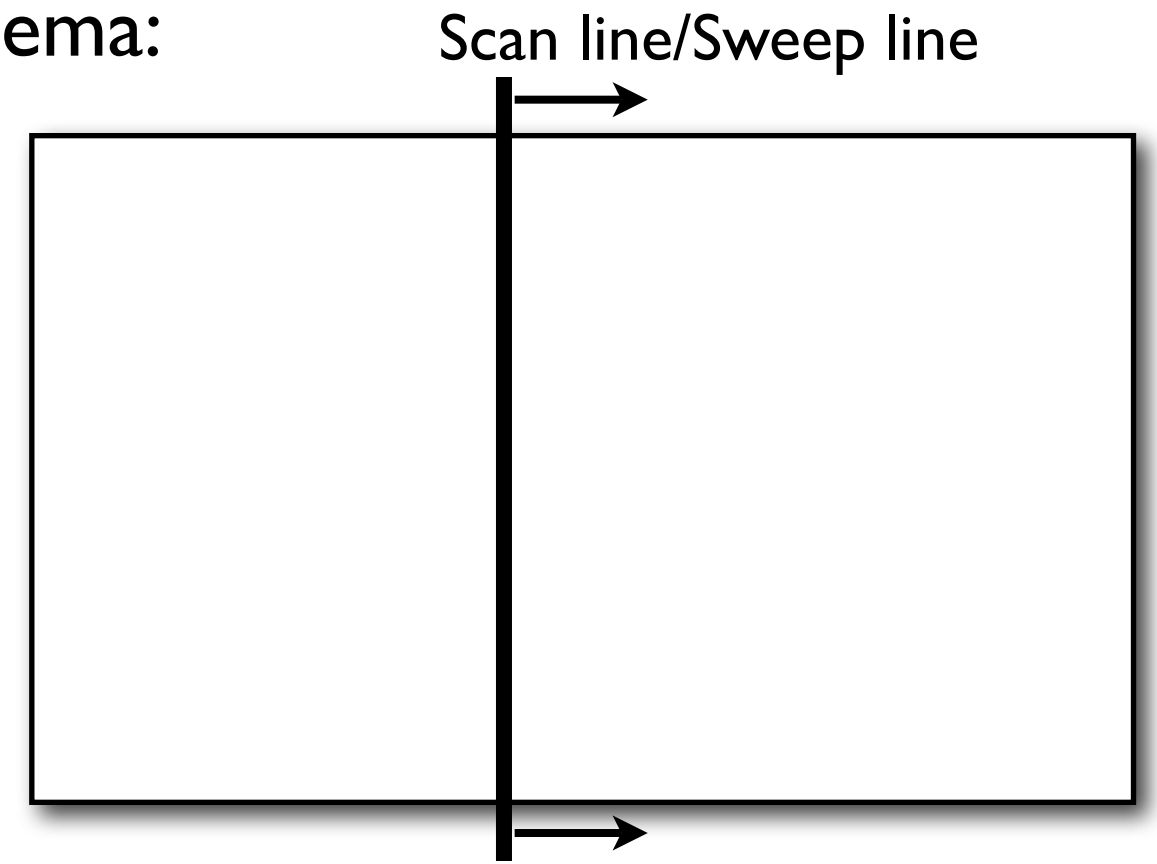


Algorithmische Geometrie
algor. Paradigma - Plane sweeping

- zugrundliegendes universelles Muster:

Plane-sweeping

- Schema:

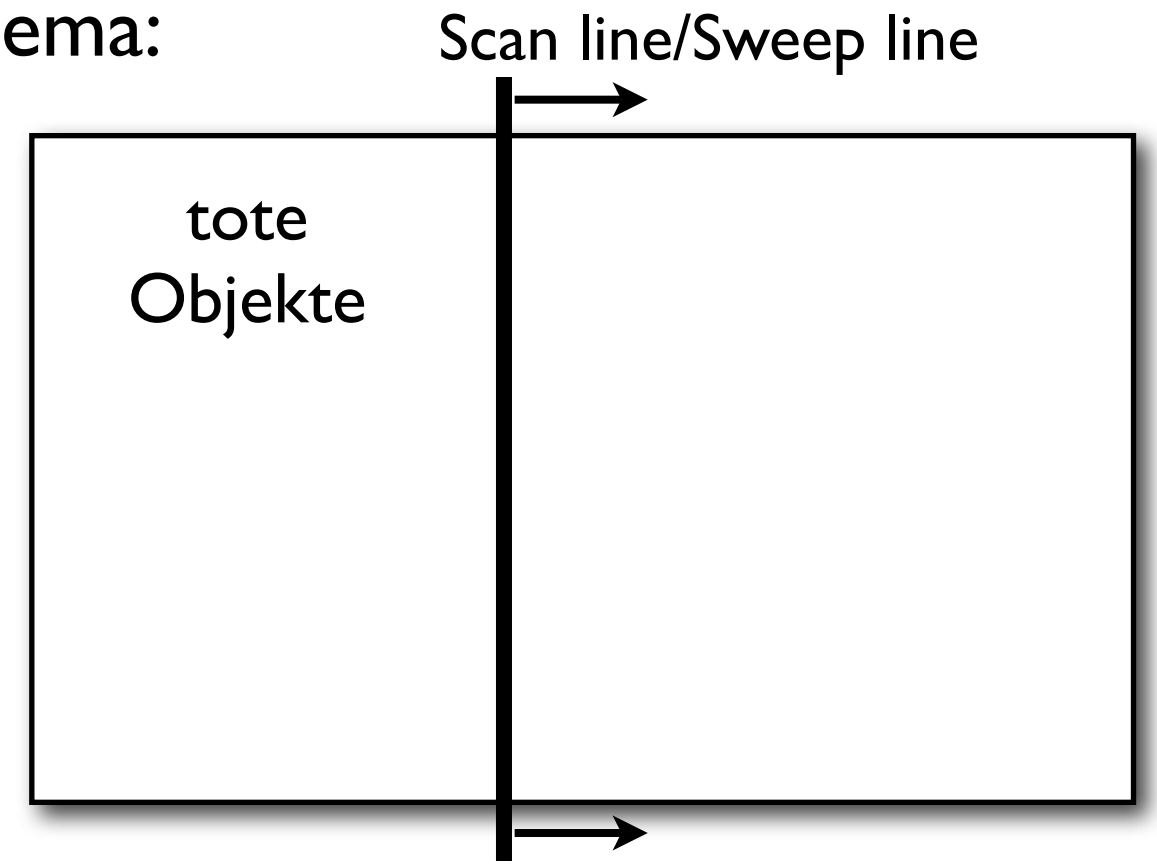


Algorithmische Geometrie
algor. Paradigma - Plane sweeping

- zugrundliegendes universelles Muster:

Plane-sweeping

- Schema:

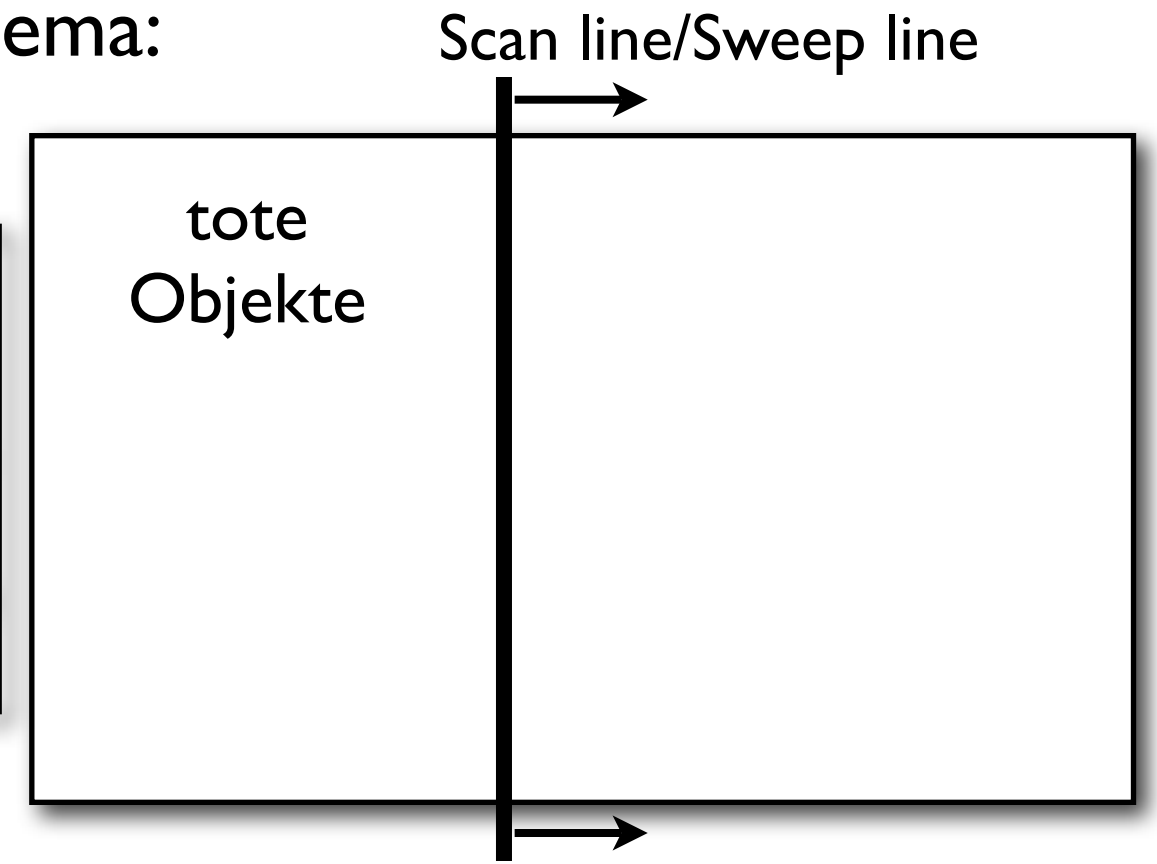


Algorithmische Geometrie
algor. Paradigma - Plane sweeping

- zugrundliegendes universelles Muster:

Plane-sweeping

- Schema:



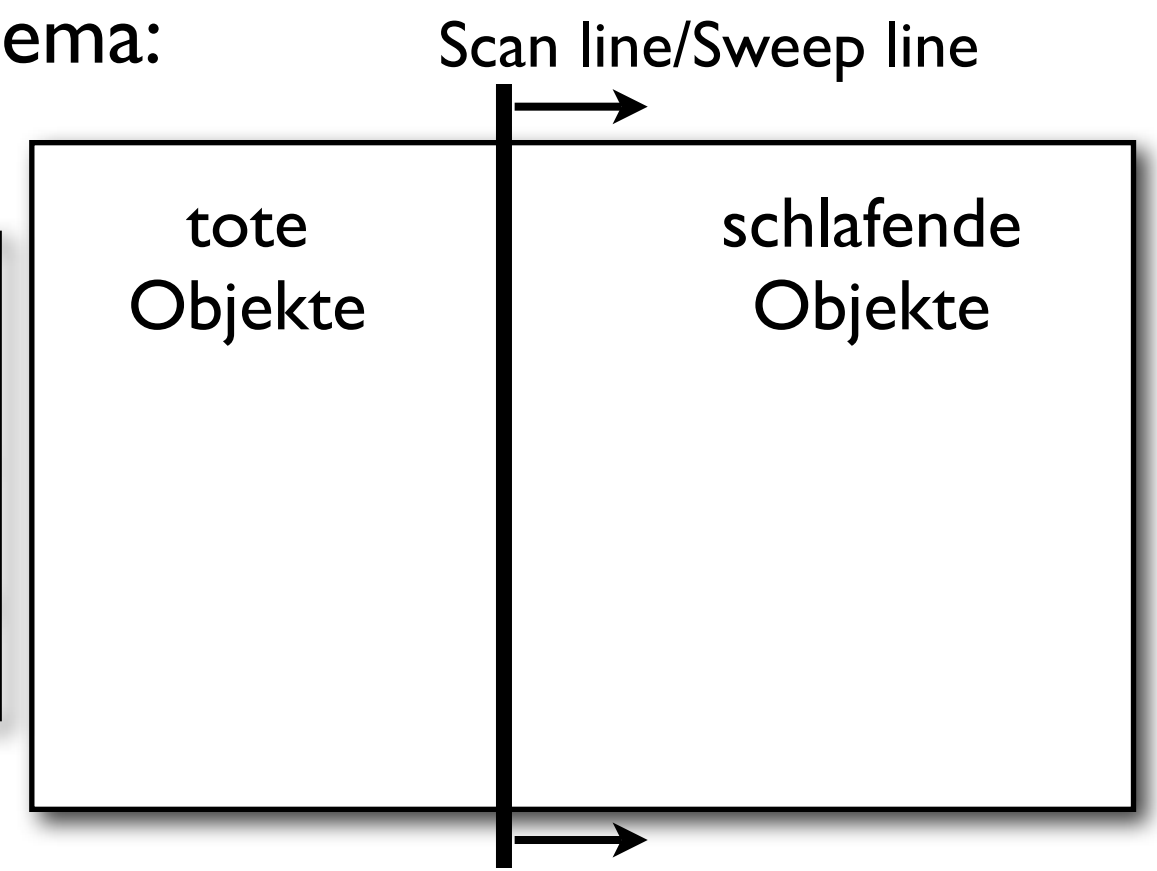
Für diesen Bereich liegt eine Lösung L vor, die alle toten Objekte erfaßt

Algorithmische Geometrie
algor. Paradigma - Plane sweeping

- zugrundliegendes universelles Muster:

Plane-sweeping

- Schema:



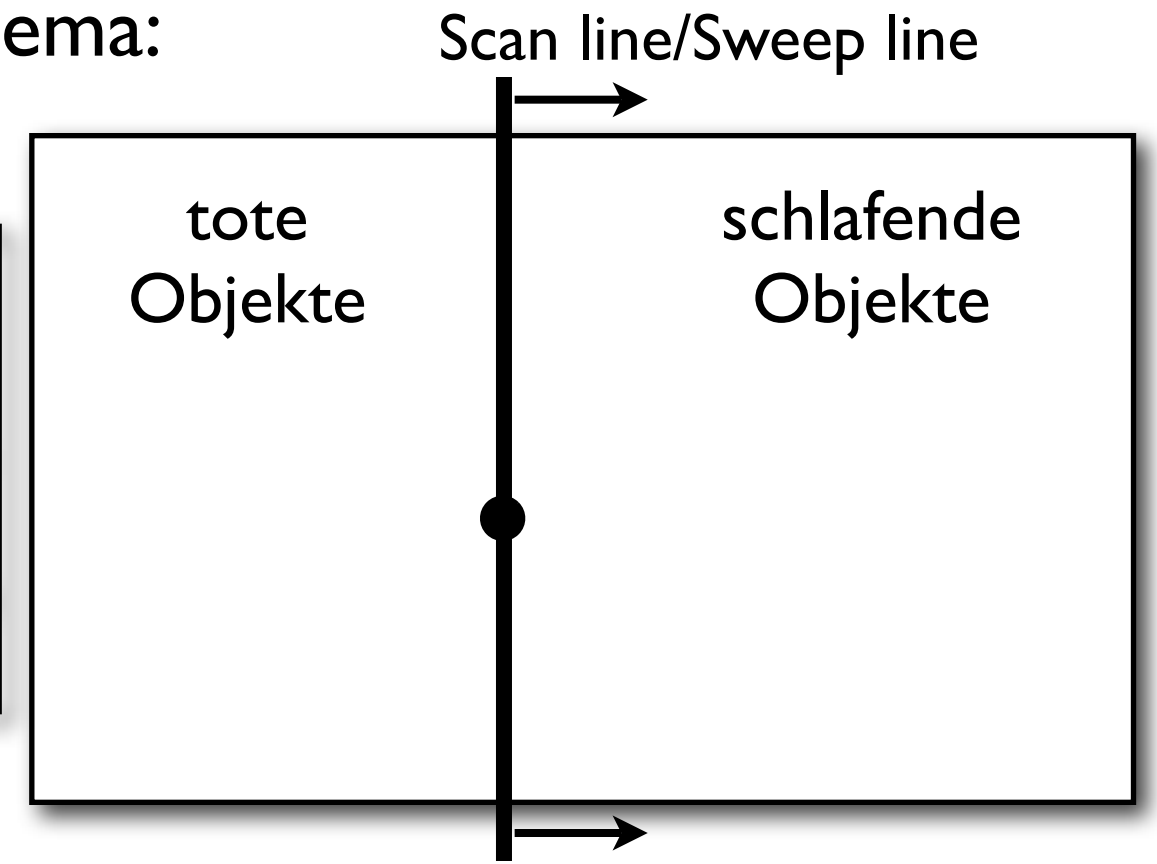
Für diesen Bereich liegt eine Lösung L vor, die alle toten Objekte erfaßt

Algorithmische Geometrie
algor. Paradigma - Plane sweeping

- zugrundliegendes universelles Muster:

Plane-sweeping

- Schema:



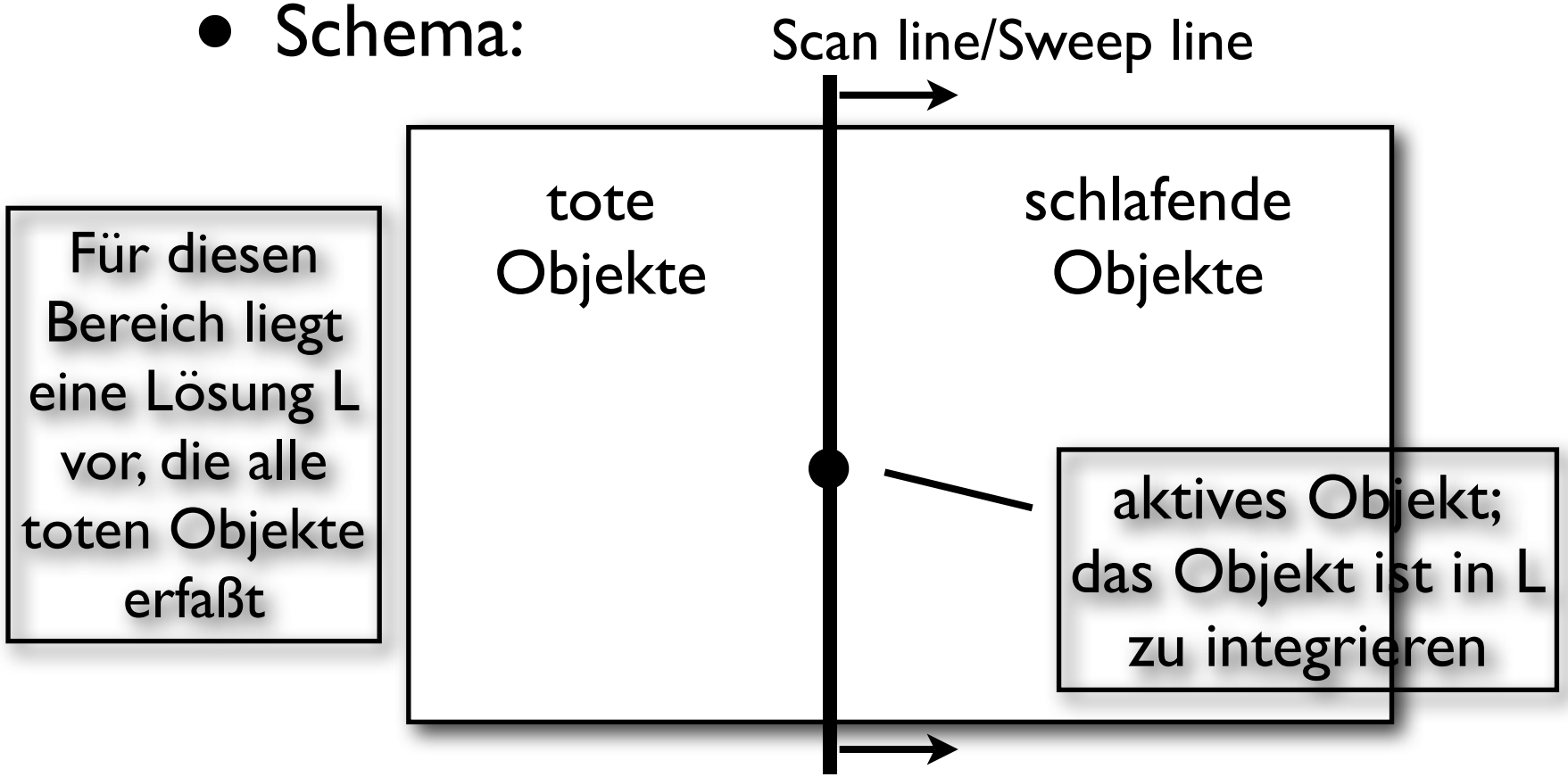
Für diesen Bereich liegt eine Lösung L vor, die alle toten Objekte erfaßt

Algorithmische Geometrie
algor. Paradigma - Plane sweeping

- zugrundliegendes universelles Muster:

Plane-sweeping

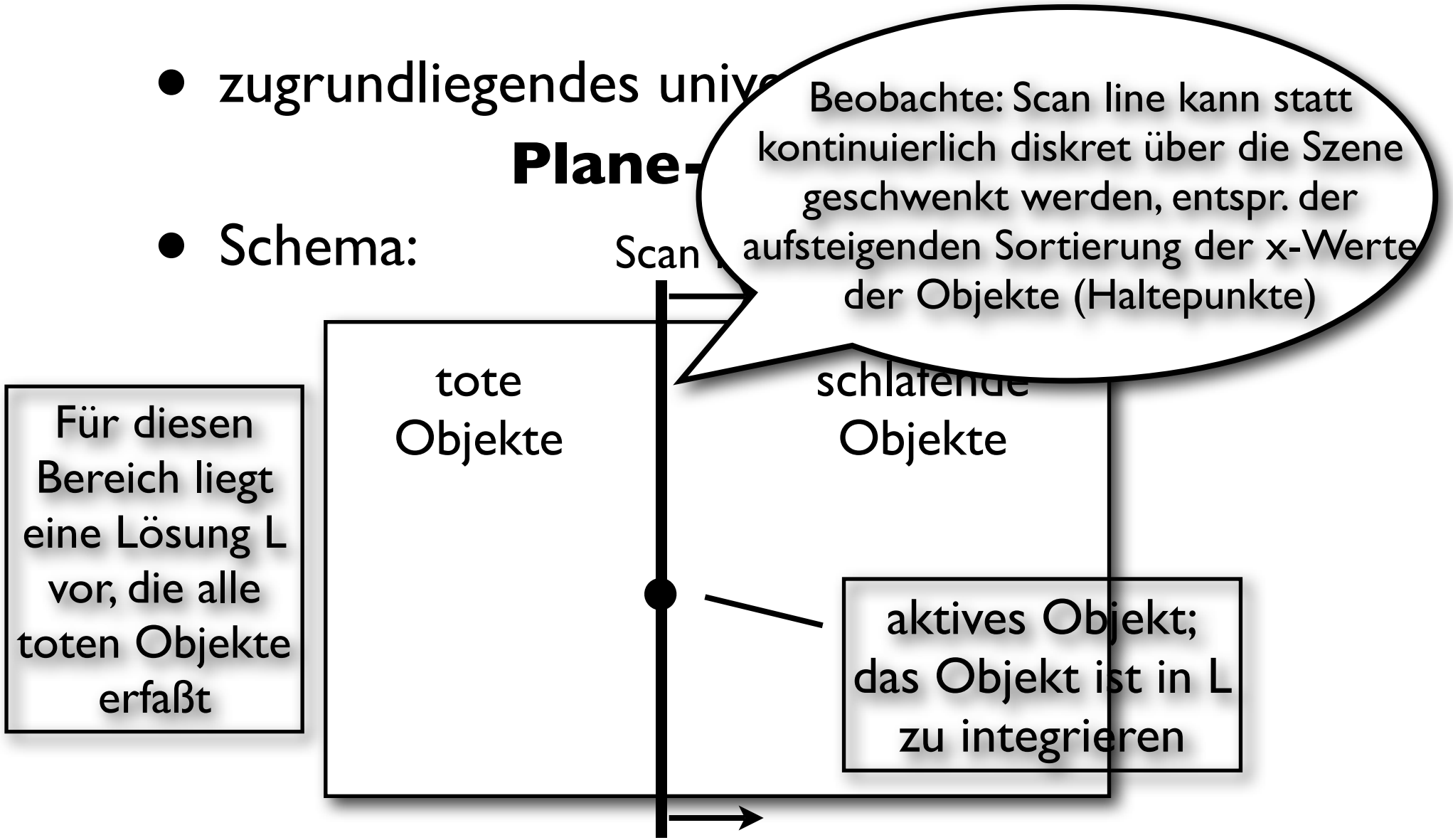
- Schema:



Algorithmische Geometrie

algor. Paradigma - Plane sweeping

- zugrundliegendes univ
- **Plane-**
- Schema:



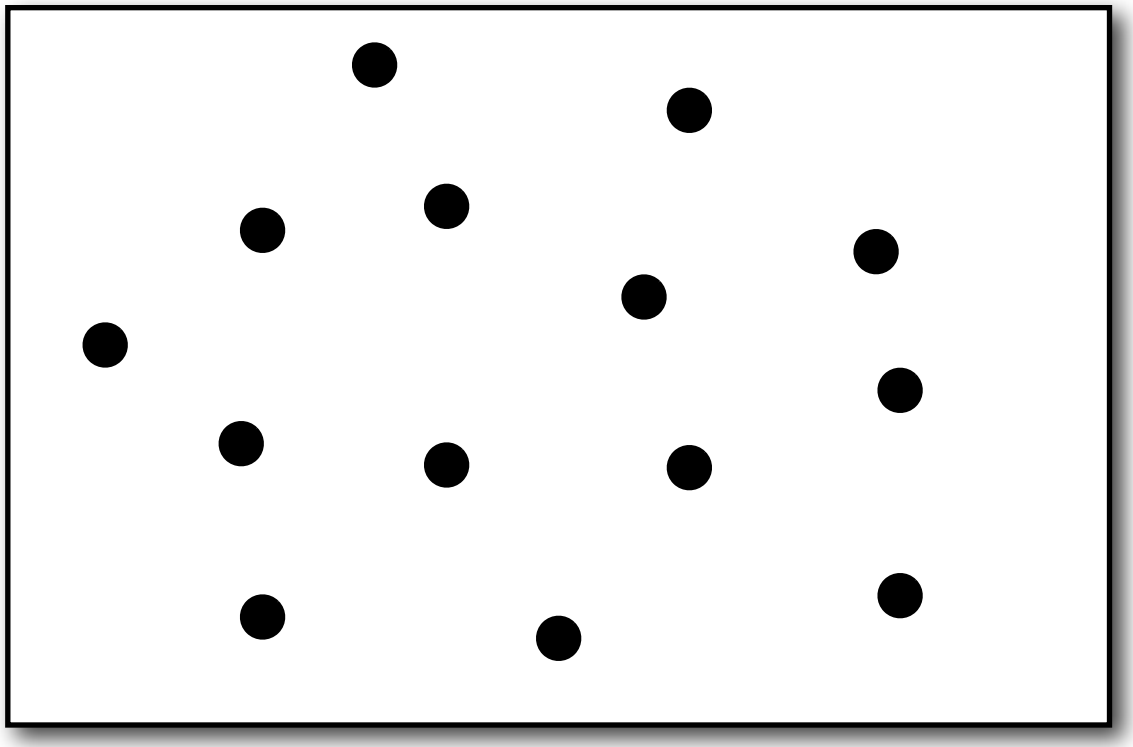
Algorithmische Geometrie
Kürzeste Abstände - Algorithm. Schema

```
Q:=Liste von Haltepunkten nach x-
Koordinate sortiert;
L:=∅;          {L: Lösungsmenge}
while Q≠∅ do
begin
    q:=first(Q);
    Q:=rest(Q);
    update(L,q)
end.
```

Algorithmische Geometrie

Kürzeste Abstände - Schemaanwendung

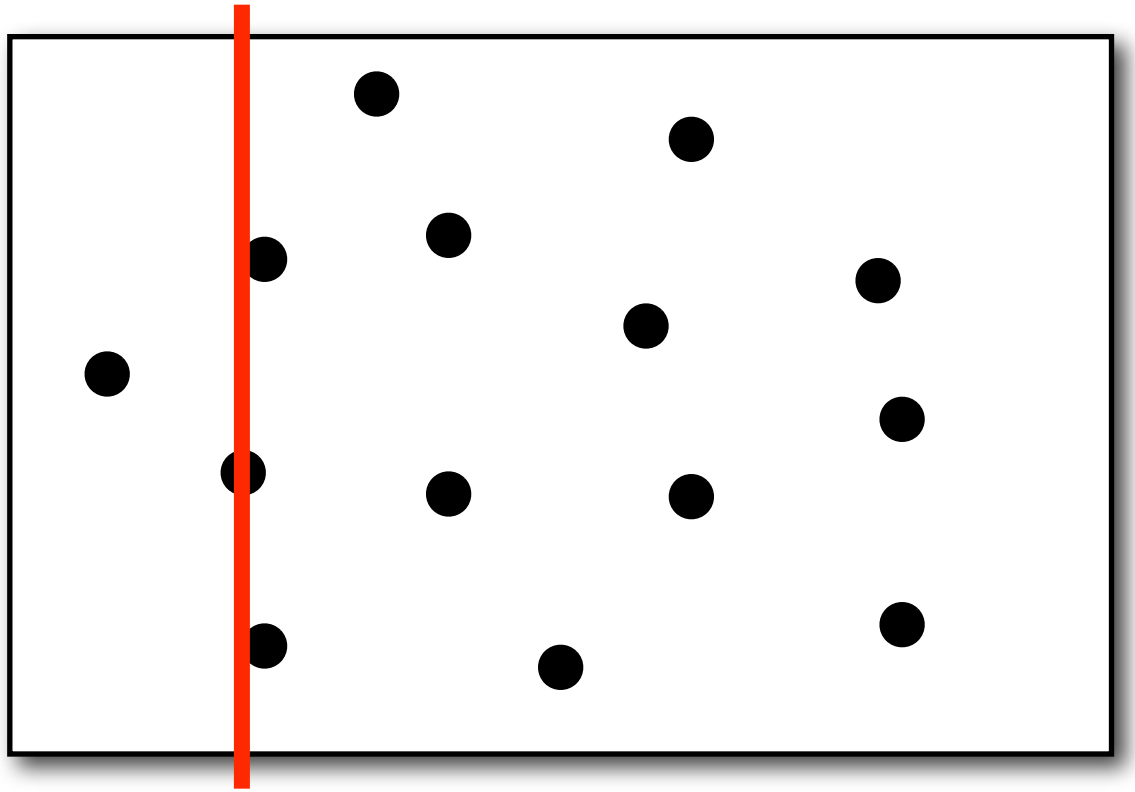
Anfangssituation



Algorithmische Geometrie

Kürzeste Abstände - Schemaanwendung

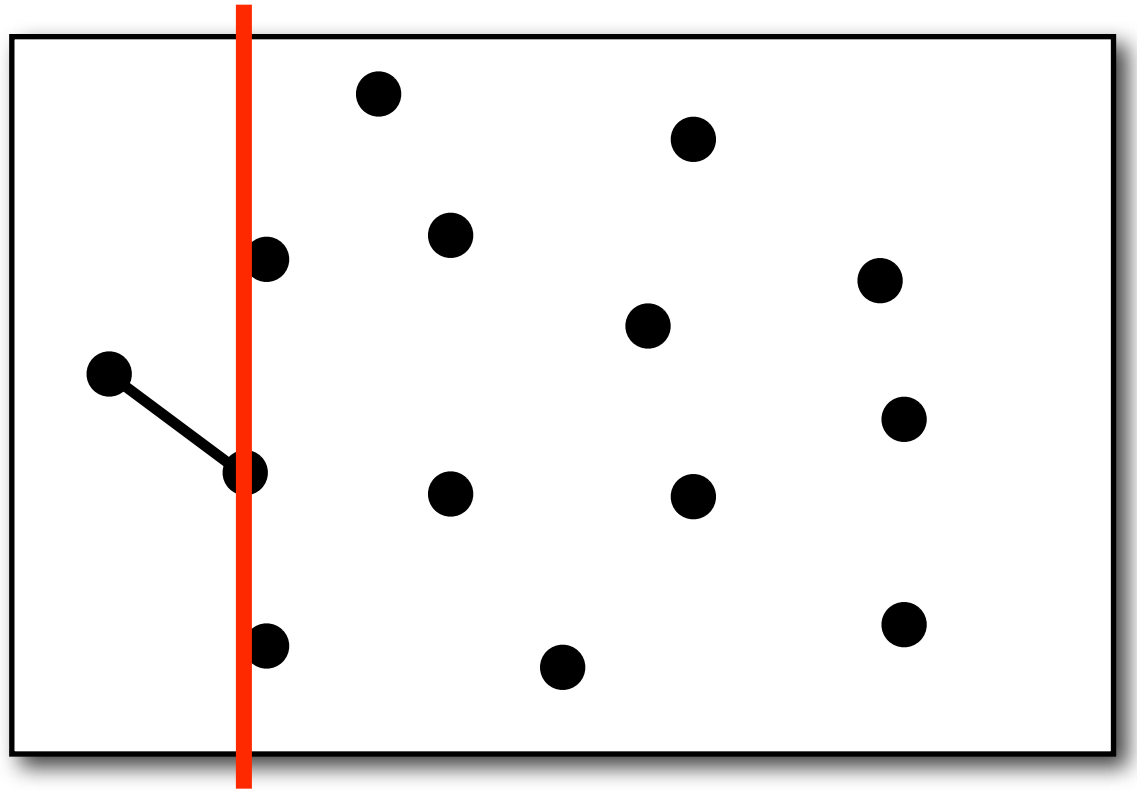
Anfangssituation



Algorithmische Geometrie

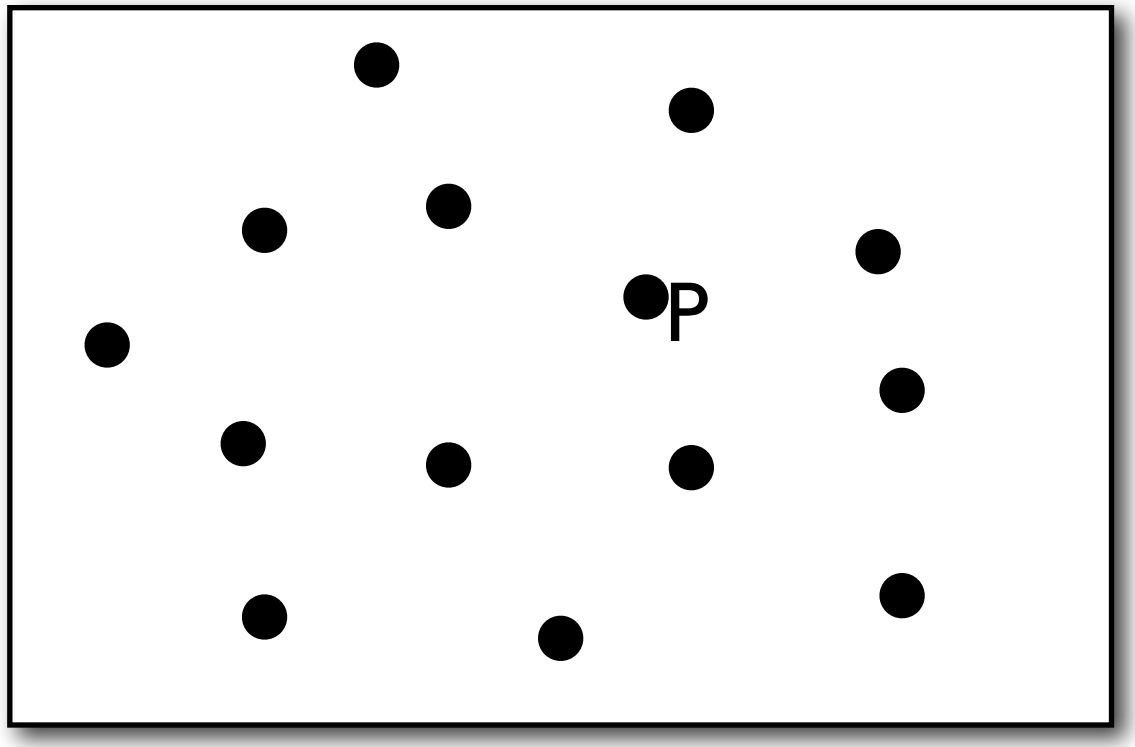
Kürzeste Abstände - Schemaanwendung

Anfangssituation



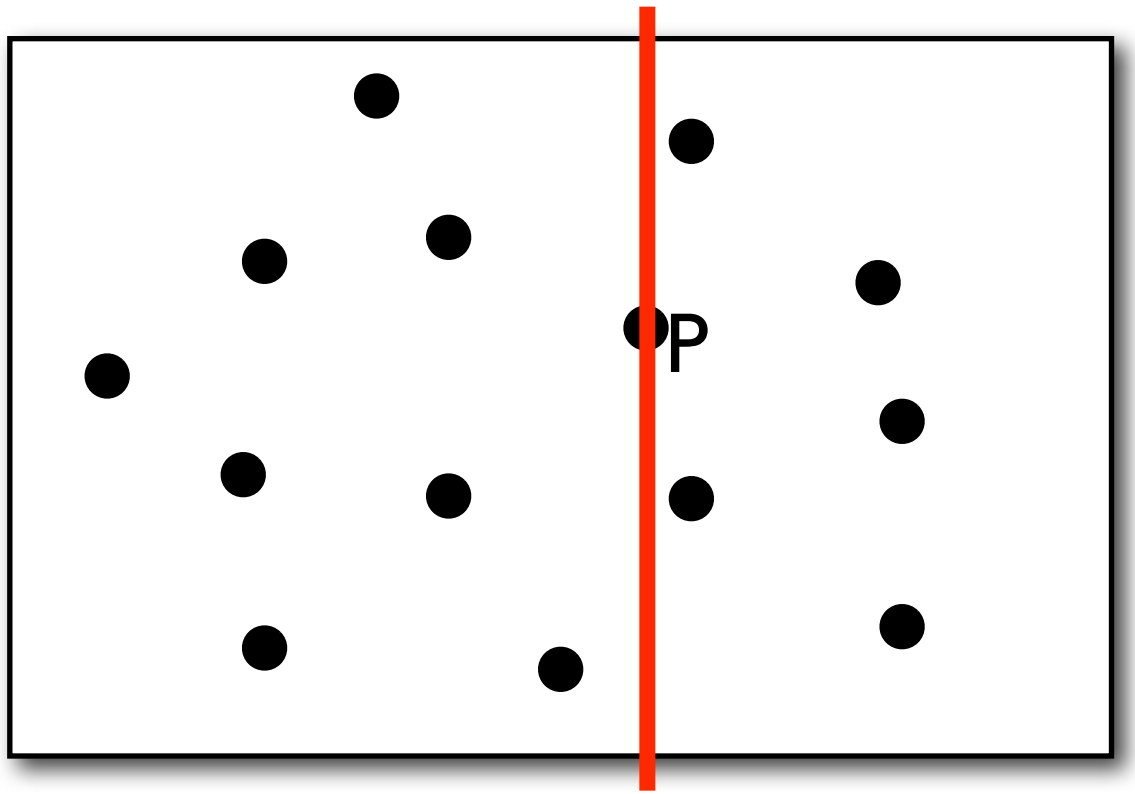
Algorithmische Geometrie
Kürzeste Abstände - Schemaanwendung

Zwischensituation



Algorithmische Geometrie
Kürzeste Abstände - Schemaanwendung

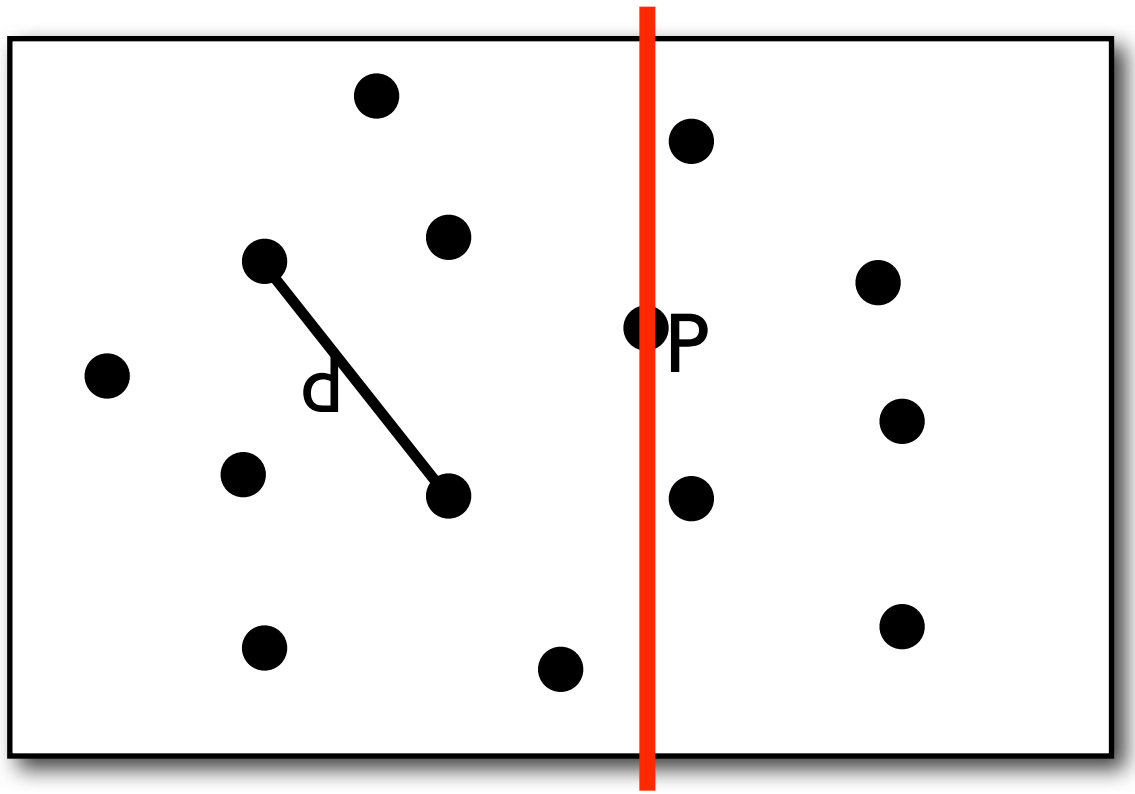
Zwischensituation



Algorithmische Geometrie

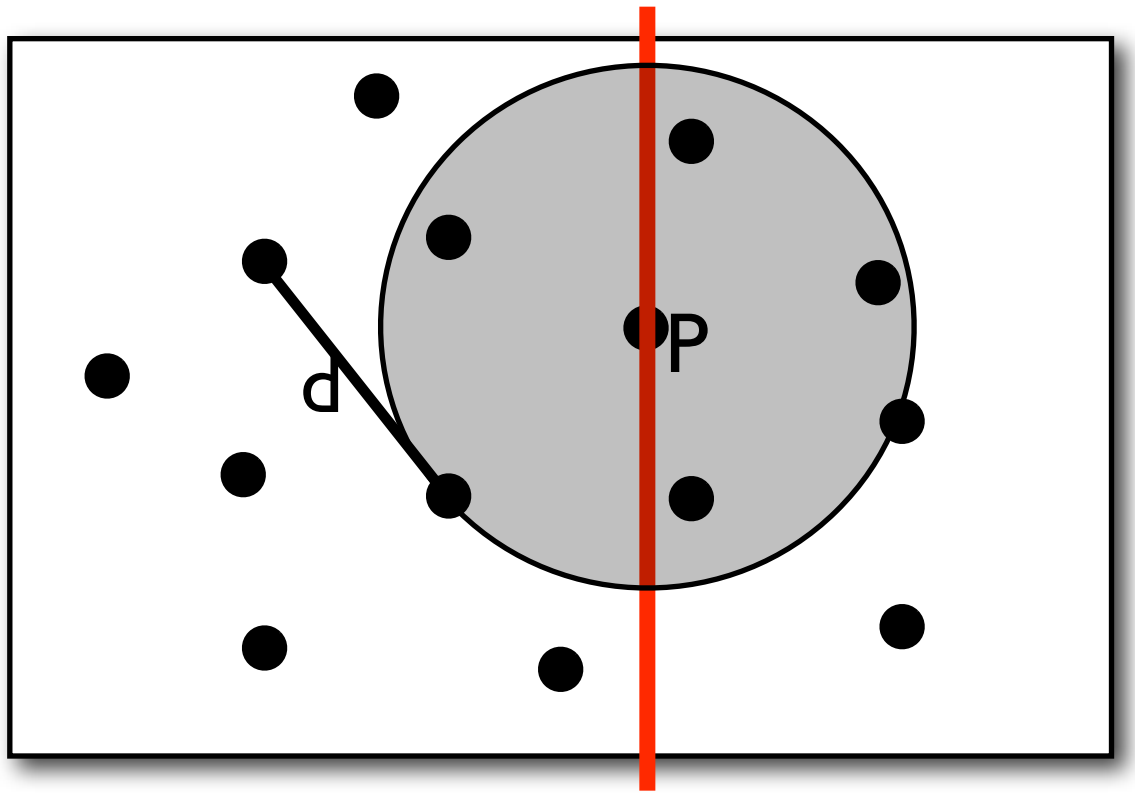
Kürzeste Abstände - Schemaanwendung

Zwischensituation



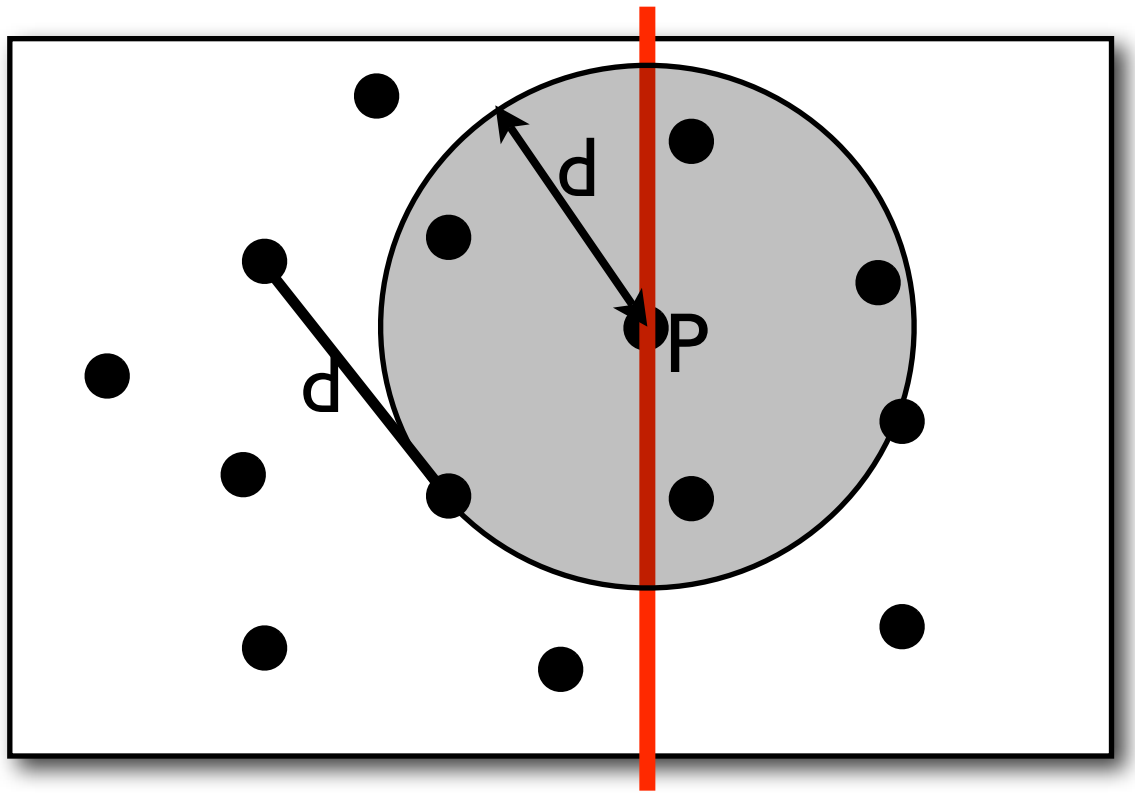
Algorithmische Geometrie
Kürzeste Abstände - Schemaanwendung

Zwischensituation



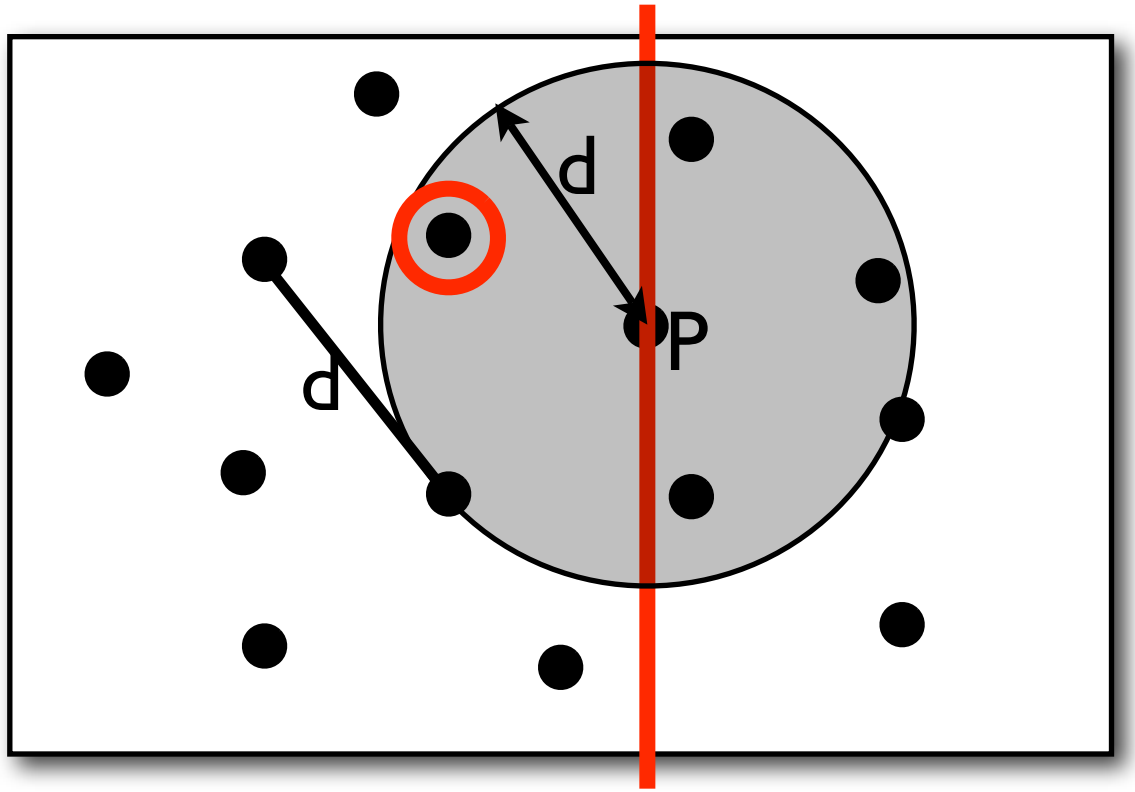
Algorithmische Geometrie
Kürzeste Abstände - Schemaanwendung

Zwischensituation



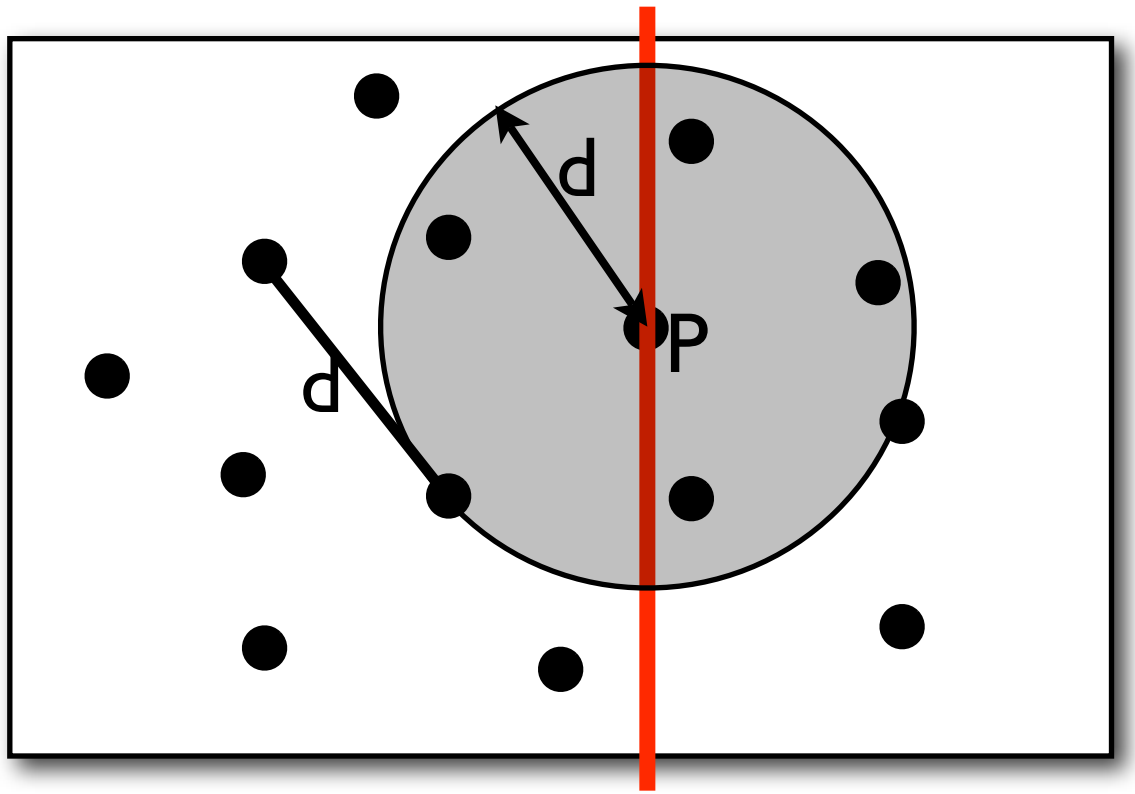
Algorithmische Geometrie
Kürzeste Abstände - Schemaanwendung

Zwischensituation



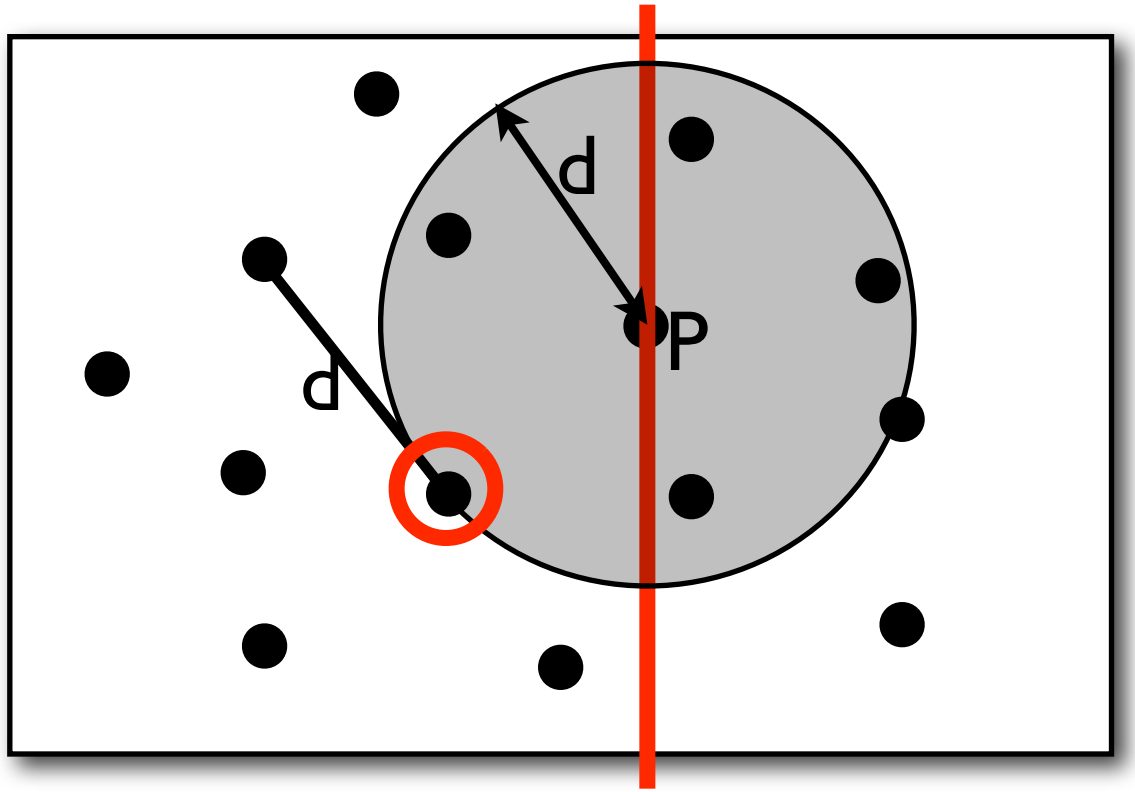
Algorithmische Geometrie
Kürzeste Abstände - Schemaanwendung

Zwischensituation



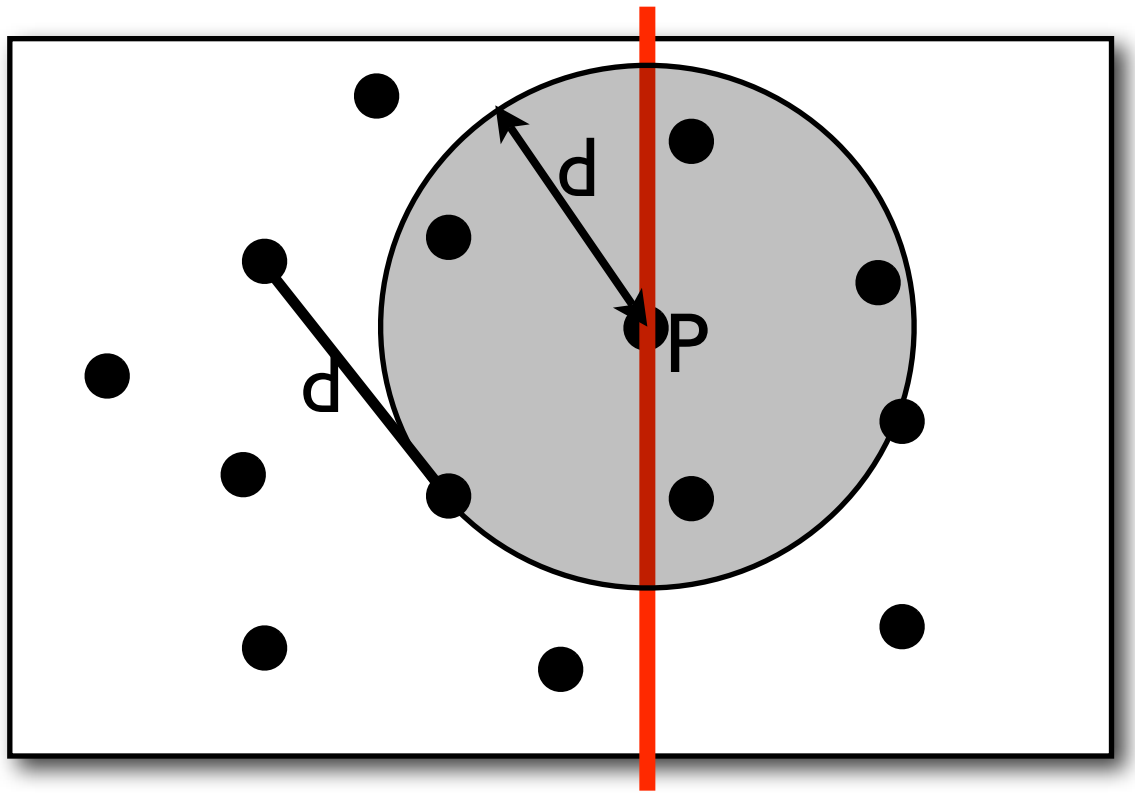
Algorithmische Geometrie
Kürzeste Abstände - Schemaanwendung

Zwischensituation



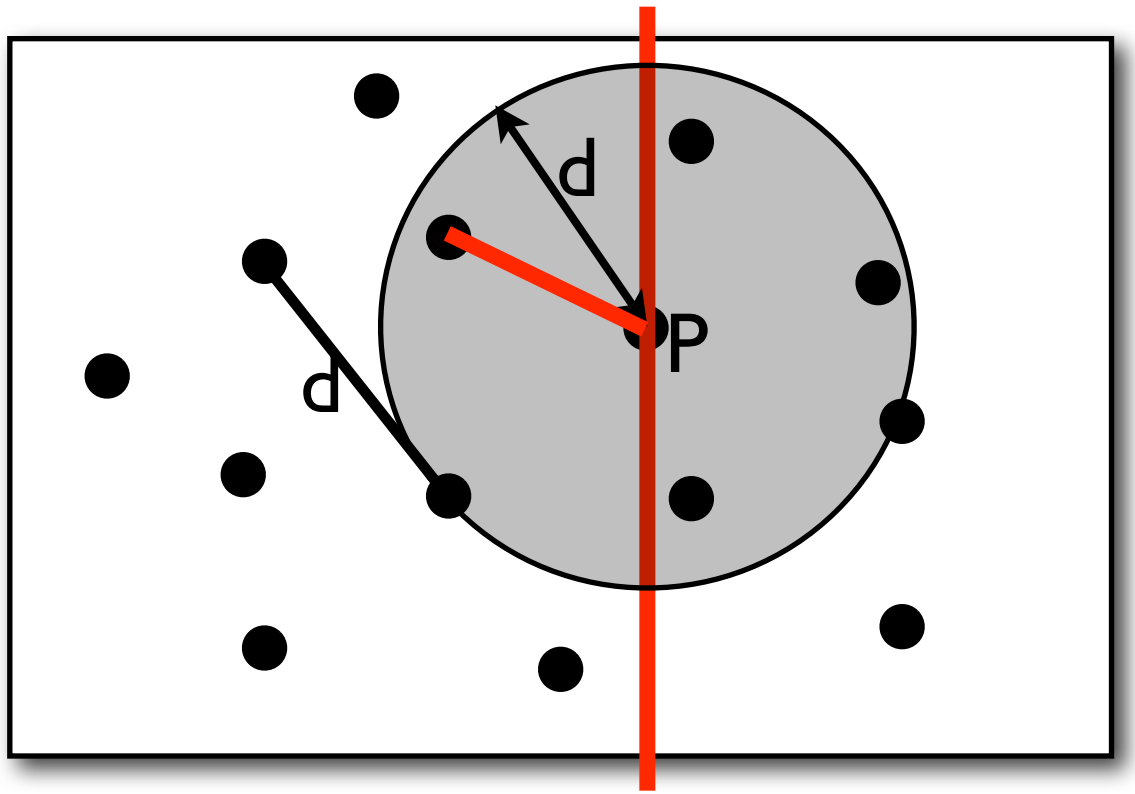
Algorithmische Geometrie
Kürzeste Abstände - Schemaanwendung

Zwischensituation



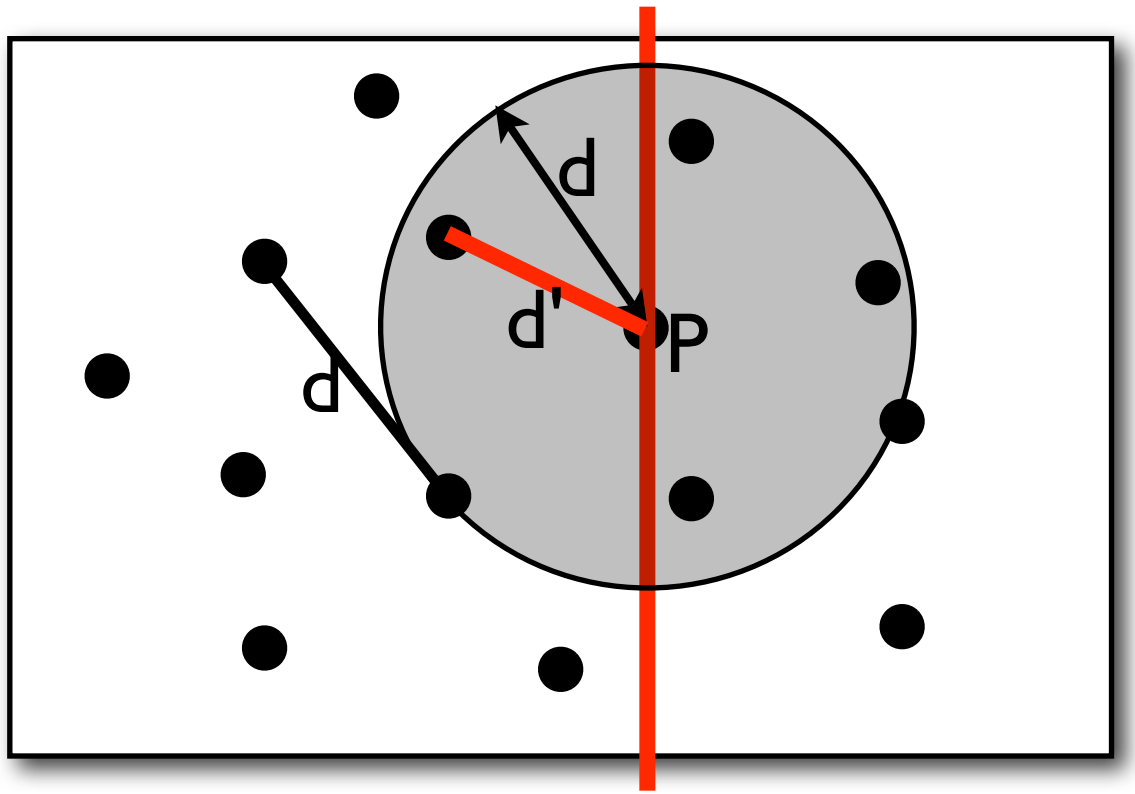
Algorithmische Geometrie
Kürzeste Abstände - Schemaanwendung

Zwischensituation



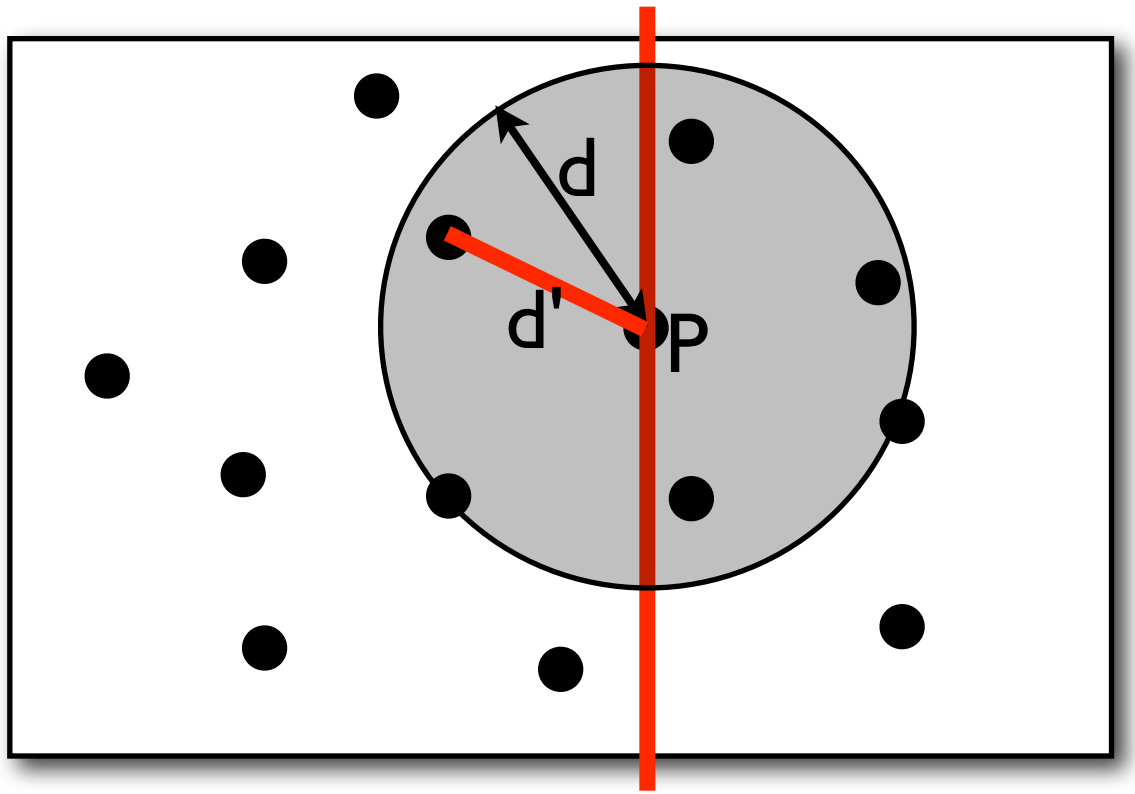
Algorithmische Geometrie
Kürzeste Abstände - Schemaanwendung

Zwischensituation



Algorithmische Geometrie
Kürzeste Abstände - Schemaanwendung

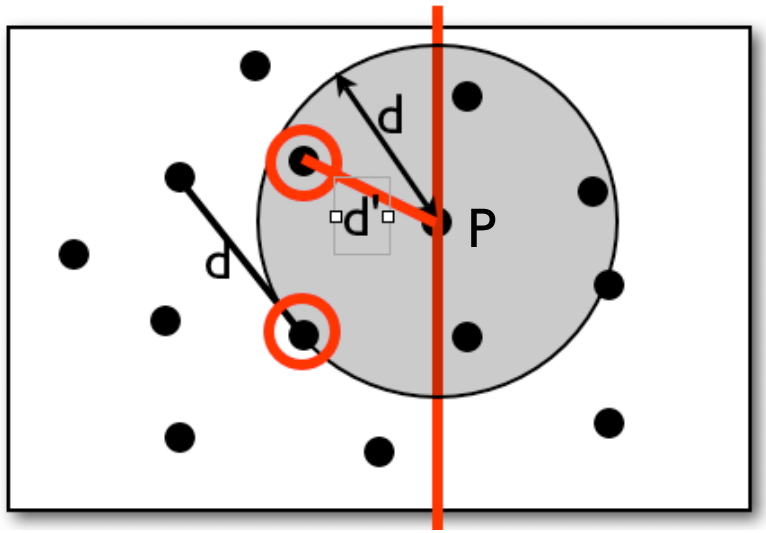
Zwischensituation



Algorithmische Geometrie
Kürzeste Abstände - Schemaanwendung

Zwischensituation (Vorgehen):

- Prüfe, ob P zu einem neuen Abstand führt
- Das ist nur dann der Fall, wenn es im Halbkreis um P mit Radius d Knoten gibt
- Prüfe alle Knoten im Halbkreis durch?
- Problem: Wieviele Punkte können im Halbkreis liegen?



Algorithmische Geometrie

Kürzeste Abstände - Laufzeitanalyse

```
Q:=Liste von Haltepunkten nach x-Koordinate sortiert;
L:=∅;        {L: Lösungsmenge}
while Q≠∅ do
begin
    q:=first(Q);
    Q:=rest(Q);
    update(L,q)
end.
```

Algorithmische Geometrie

Kürzeste Abstände - Laufzeitanalyse

```
Q:=Liste von Haltepunkten nach x-Koordinate sortiert;   O(n log n)
L:=∅;          {L: Lösungsmenge}
while Q≠∅ do
begin
    q:=first(Q);
    Q:=rest(Q);
    update(L,q)
end.
```

Algorithmische Geometrie

Kürzeste Abstände - Laufzeitanalyse

Q:=Liste von Haltepunkten nach x-Koordinate sortiert; O(n log n)

L:=∅; {L: Lösungsmenge}

while Q≠∅ do O(n) Durchläufe

begin

q:=first(Q);

Q:=rest(Q);

update(L,q)

end.

Algorithmische Geometrie

Kürzeste Abstände - Laufzeitanalyse

```
Q:=Liste von Haltepunkten nach x-Koordinate sortiert;   O(n log n)
L:=∅;          {L: Lösungsmenge}
while Q≠∅ do   O(n) Durchläufe
begin
    q:=first(Q);   O(1)
    Q:=rest(Q);
    update(L,q)
end.
```

Algorithmische Geometrie

Kürzeste Abstände - Laufzeitanalyse

Q:=Liste von Haltepunkten nach x-Koordinate sortiert; O(n log n)

L:=∅; {L: Lösungsmenge}

while Q≠∅ do O(n) Durchläufe

begin

q:=first(Q); O(1)

Q:=rest(Q); O(1)

update(L,q)

end.

Algorithmische Geometrie

Kürzeste Abstände - Laufzeitanalyse

Q:=Liste von Haltepunkten nach x-Koordinate sortiert; O(n log n)

L:=∅; {L: Lösungsmenge}

while Q≠∅ do O(n) Durchläufe

begin

q:=first(Q); O(1)

Q:=rest(Q); O(1)

update(L,q) ←

end.

Überprüfe alle Punkte im Halbkreis um P auf ihren Abstand zu P;
zeige: im Halbkreis max. 4 Punkte.
⇒ O(1)

Algorithmische Geometrie

Kürzeste Abstände - Laufzeitanalyse

```
Q:=Liste von Haltepunkten nach x-Koordinate sortiert;
L:=∅;          {L: Lösungsmenge}
while Q≠∅ do
begin
    q:=first(Q);
    Q:=rest(Q);
    update(L,q)
end.
```

O(n log n)

O(n) Durchläufe

Insgesamt: O(n log n)

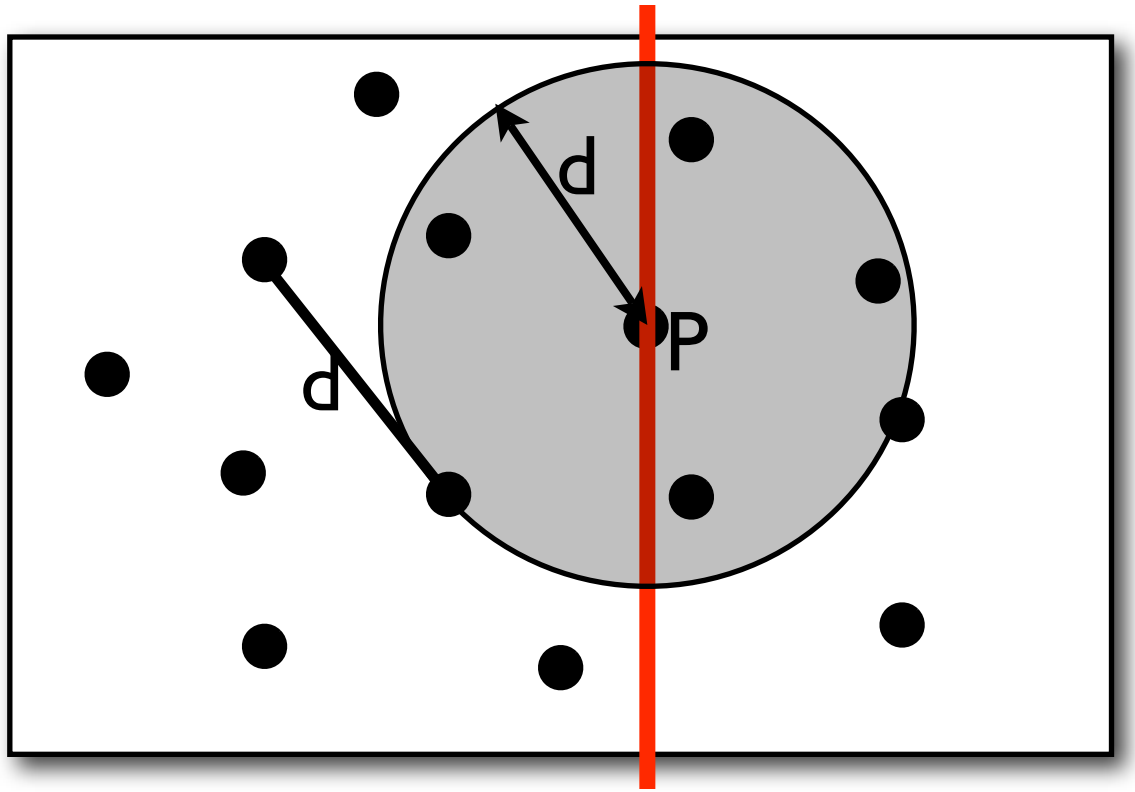
O(1)

O(1)

Überprüfe alle Punkte im Halbkreis um P auf ihren Abstand zu P;
zeige: im Halbkreis max. 4 Punkte.
⇒ O(1)

Algorithmische Geometrie

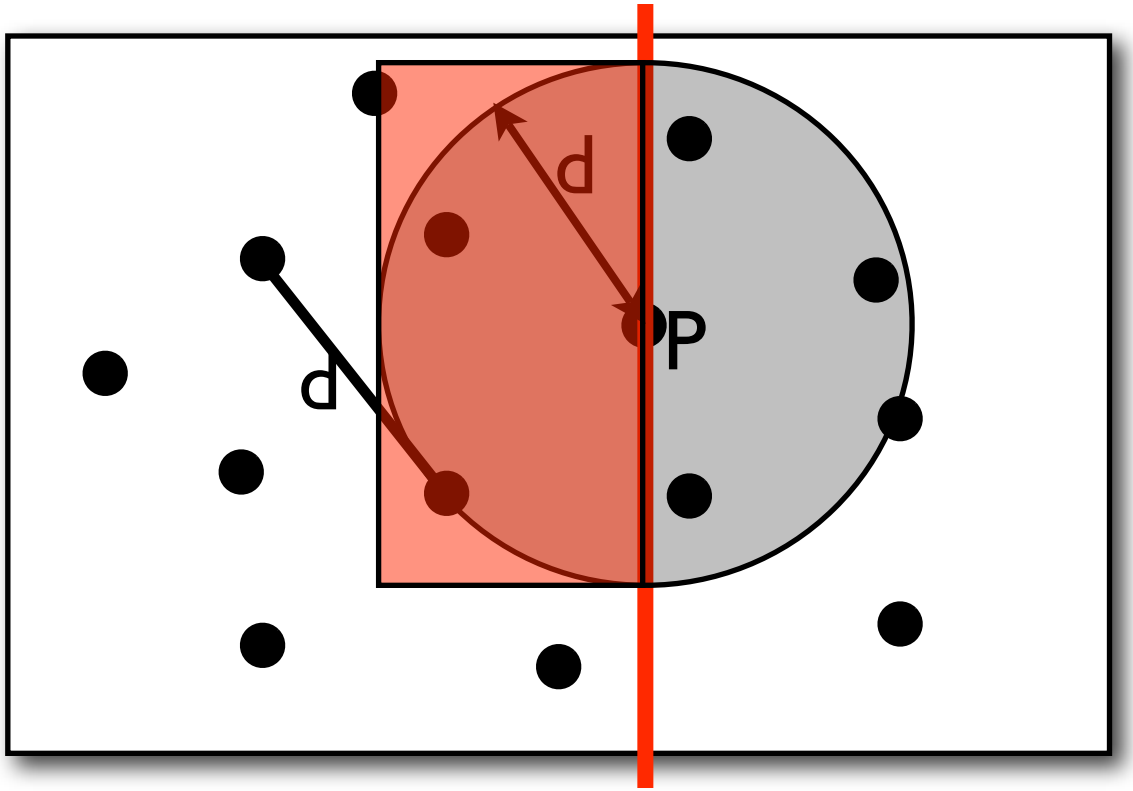
Kürzeste Abstände - Halbkreisanalyse



Halbkreisanfragen sind aufwendig. Daher:
Erweitere den Halbkreis mit Radius d
auf ein Rechteck mit Kantenlängen d und 2d.
Zeige: Rechteck enthält max. 10 Punkte.

Algorithmische Geometrie

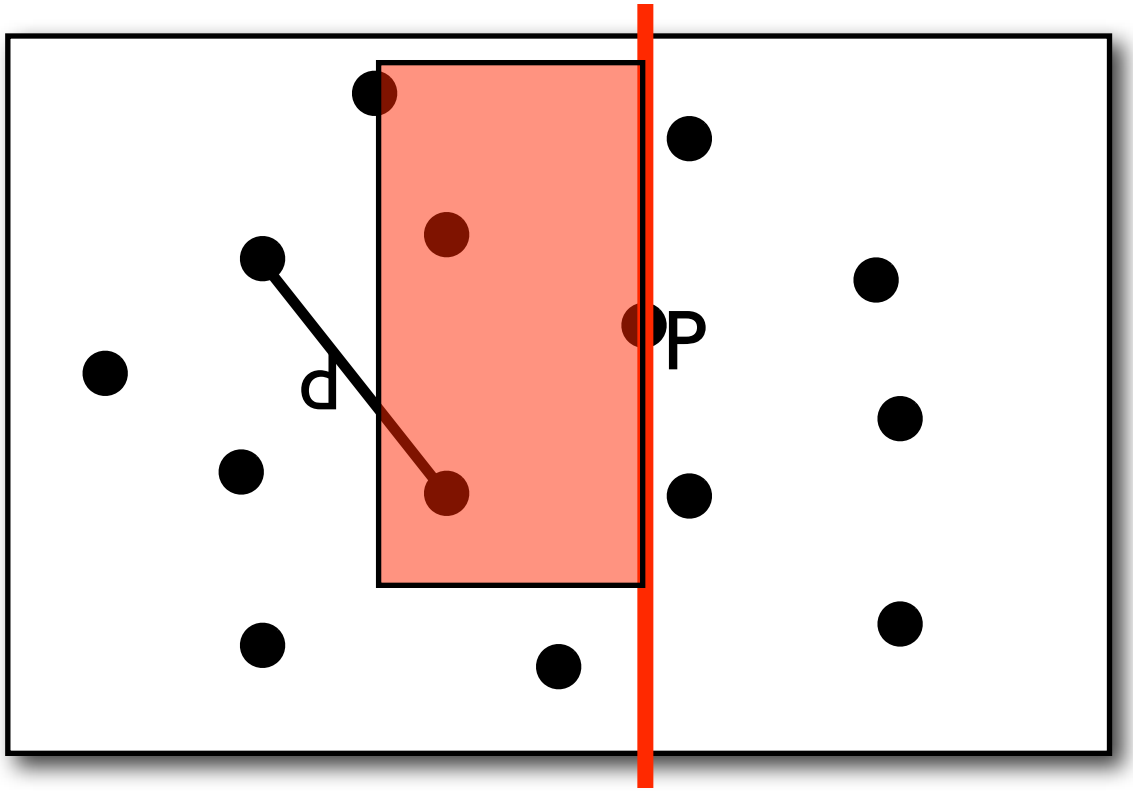
Kürzeste Abstände - Halbkreisanalyse



Halbkreisanfragen sind aufwendig. Daher:
Erweitere den Halbkreis mit Radius d
auf ein Rechteck mit Kantenlängen d und 2d.
Zeige: Rechteck enthält max. 10 Punkte.

Algorithmische Geometrie

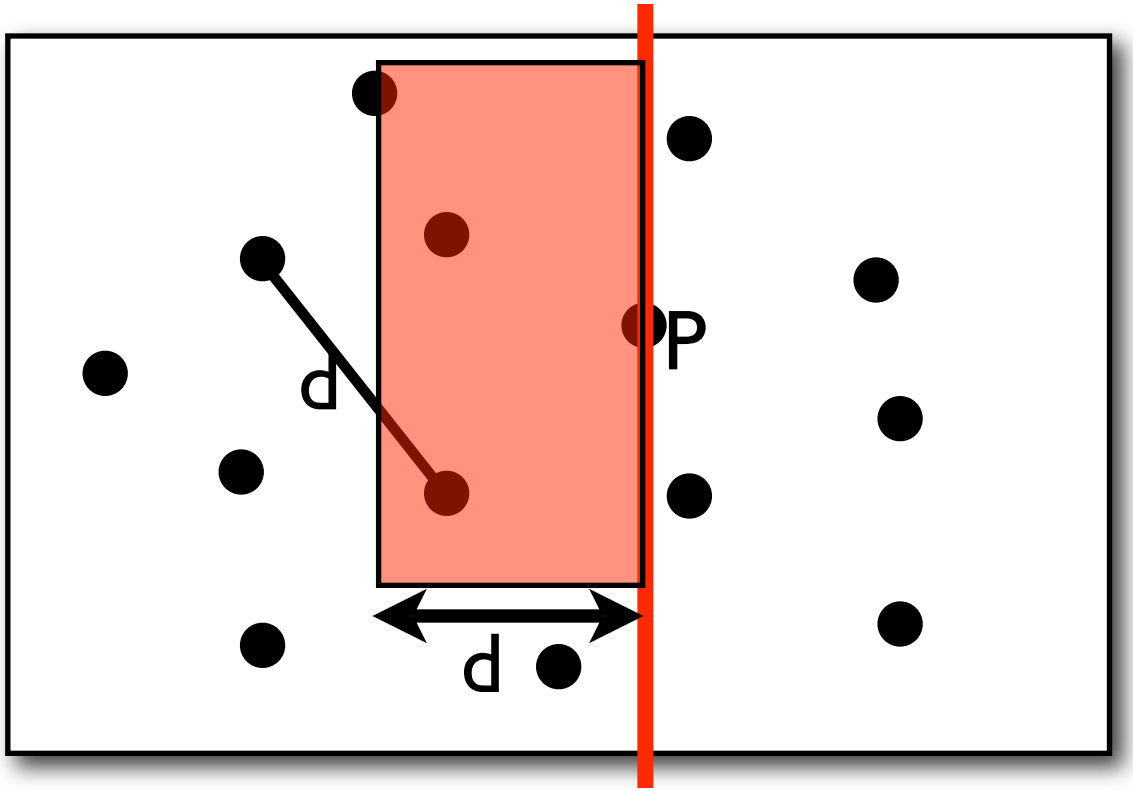
Kürzeste Abstände - Halbkreisanalyse



Halbkreisanfragen sind aufwendig. Daher:
Erweitere den Halbkreis mit Radius d
auf ein Rechteck mit Kantenlängen d und 2d.
Zeige: Rechteck enthält max. 10 Punkte.

Algorithmische Geometrie

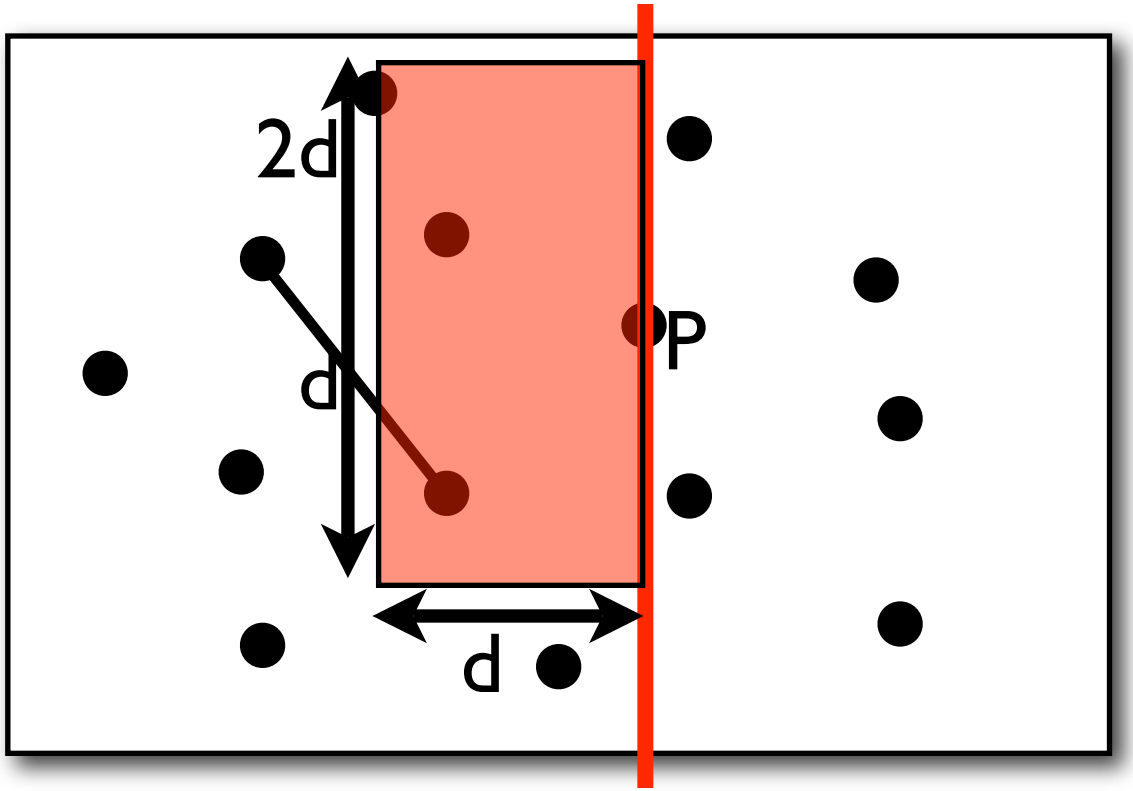
Kürzeste Abstände - Halbkreisanalyse



Halbkreisanfragen sind aufwendig. Daher:
Erweitere den Halbkreis mit Radius d
auf ein Rechteck mit Kantenlängen d und 2d.
Zeige: Rechteck enthält max. 10 Punkte.

Algorithmische Geometrie

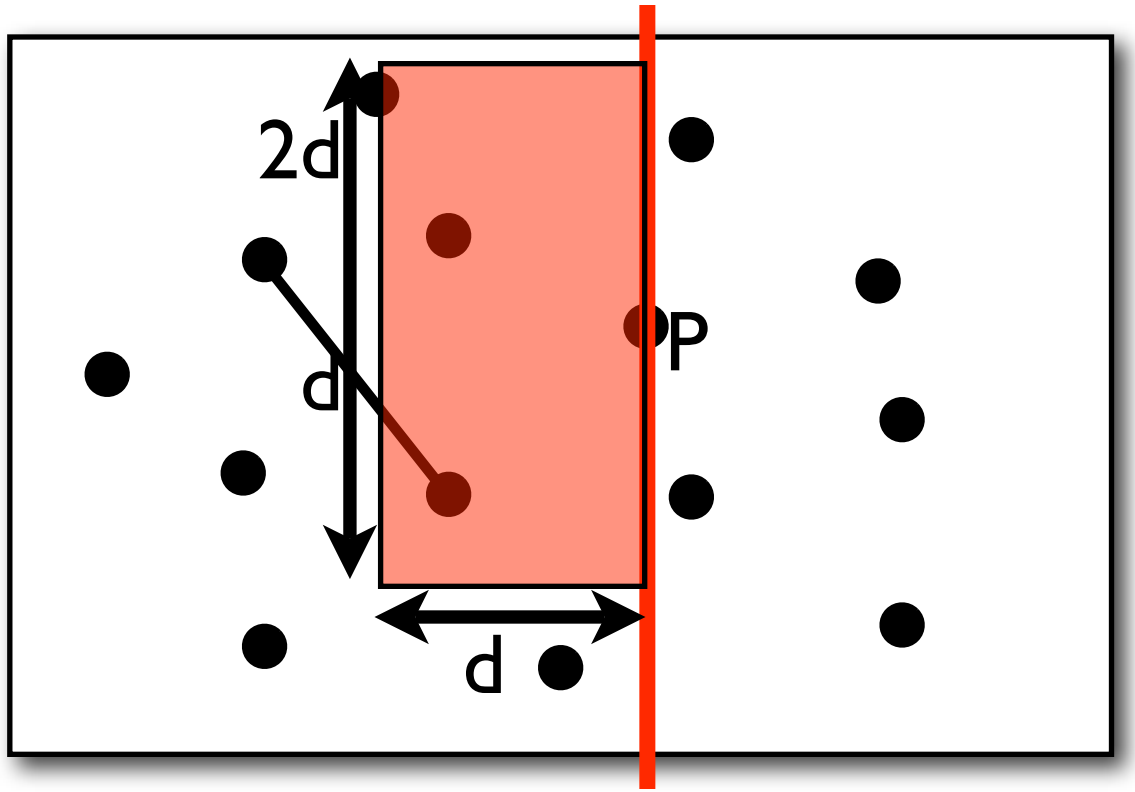
Kürzeste Abstände - Halbkreisanalyse



Halbkreisanfragen sind aufwendig. Daher:
Erweitere den Halbkreis mit Radius d
auf ein Rechteck mit Kantenlängen d und 2d.
Zeige: Rechteck enthält max. 10 Punkte.

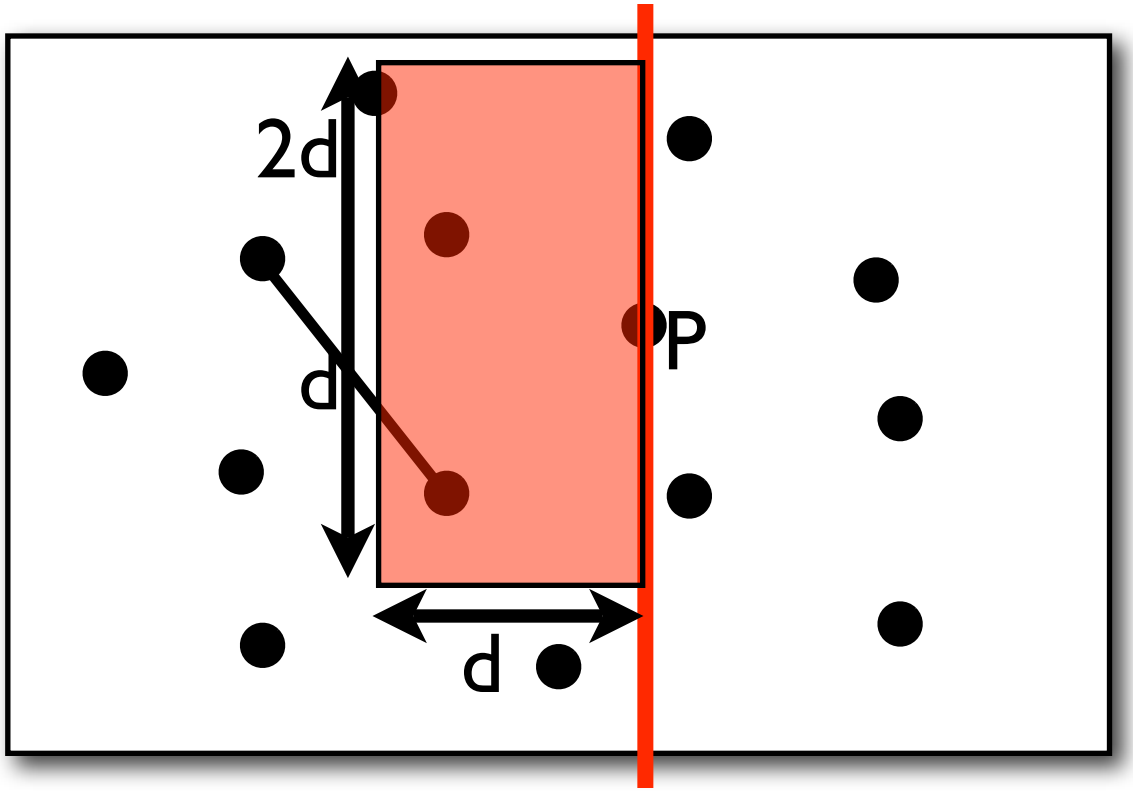
Algorithmische Geometrie

Kürzeste Abstände - Halbkreisanalyse



Algorithmische Geometrie

Kürzeste Abstände - Halbkreisanalyse



Anfrage an Rechteck:
Vergleiche nur x- und y-Koordinaten der Punkte mit denen von P.

Algorithmische Geometrie
Kürzeste Abstände - Lemma

Lemma:

Sei $d>0$ und S eine Menge von Punkten in der Ebene, die paarweise Abstand $\geq d$ besitzen. Dann enthält ein Rechteck mit Kantenlängen d und 2d höchstens 10 Punkte aus S.

Algorithmische Geometrie
Kürzeste Abstände - Lemmabeweis

Beweis:

Nach Voraussetzung sind die offenen Kreise um $p \in S$ mit Radius d/2 paarweise disjunkt.

Liegt p in einem Rechteck R mit Kantenlänge d und 2d, so ist mindestens 1/4 der offenen Kreisscheibe in R enthalten. Durch Flächenberechnung ergibt sich, daß R höchstens

$$\frac{\text{Fläche}(R)}{\text{Fläche}(\text{Viertelkreis})} = \frac{2d^2}{(1/4)\pi(d/2)^2} = \frac{32}{\pi} < 11$$

viele Punkte von S enthalten kann.

Algorithmische Geometrie
Kürzeste Abstände - Ergebnis

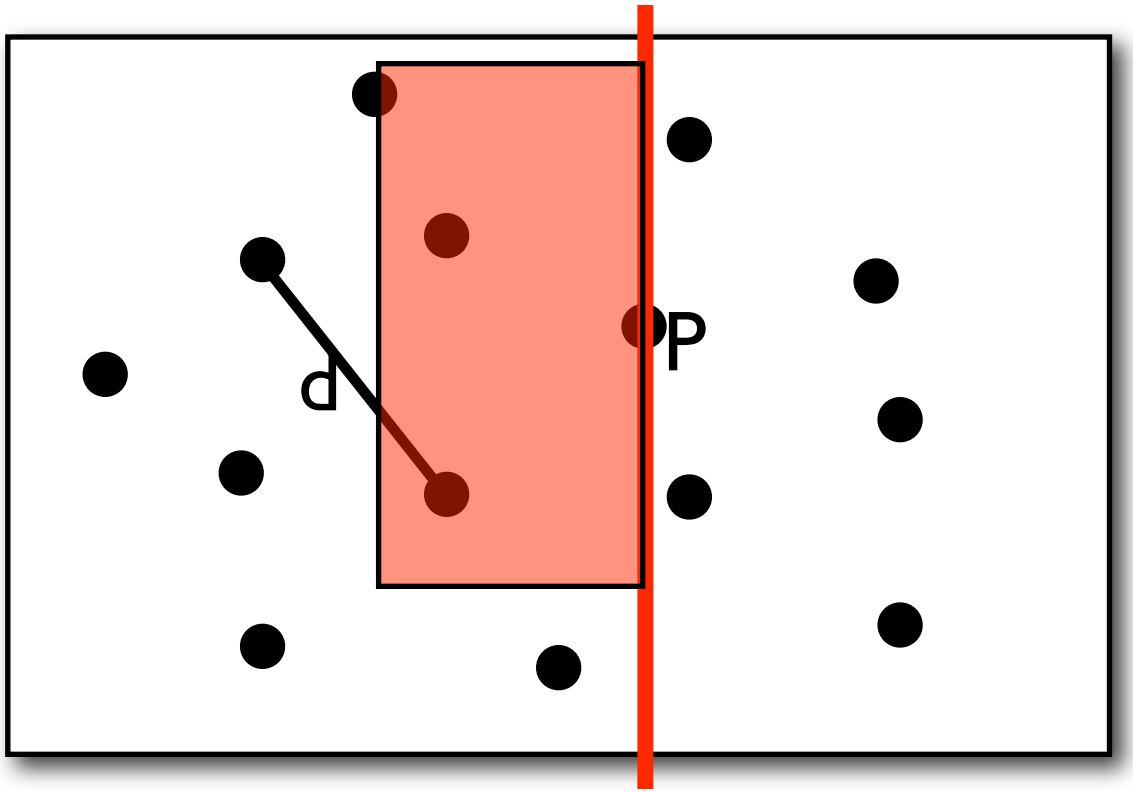
- Ergebnis:
  - update(L,q) bei Hinzunahme eines neuen Punkts benötigt O(1) Zeit (bei geeigneter Implementierung).
  - Gesamtalgorithmus: O(n log n) Zeit.

Algorithmische Geometrie
Kürzeste Abstände - Animation

- http://www.cs.mcgill.ca/~cs251/ClosestPair/ClosestPairAP.html
- Zugriff: 06.07.2007

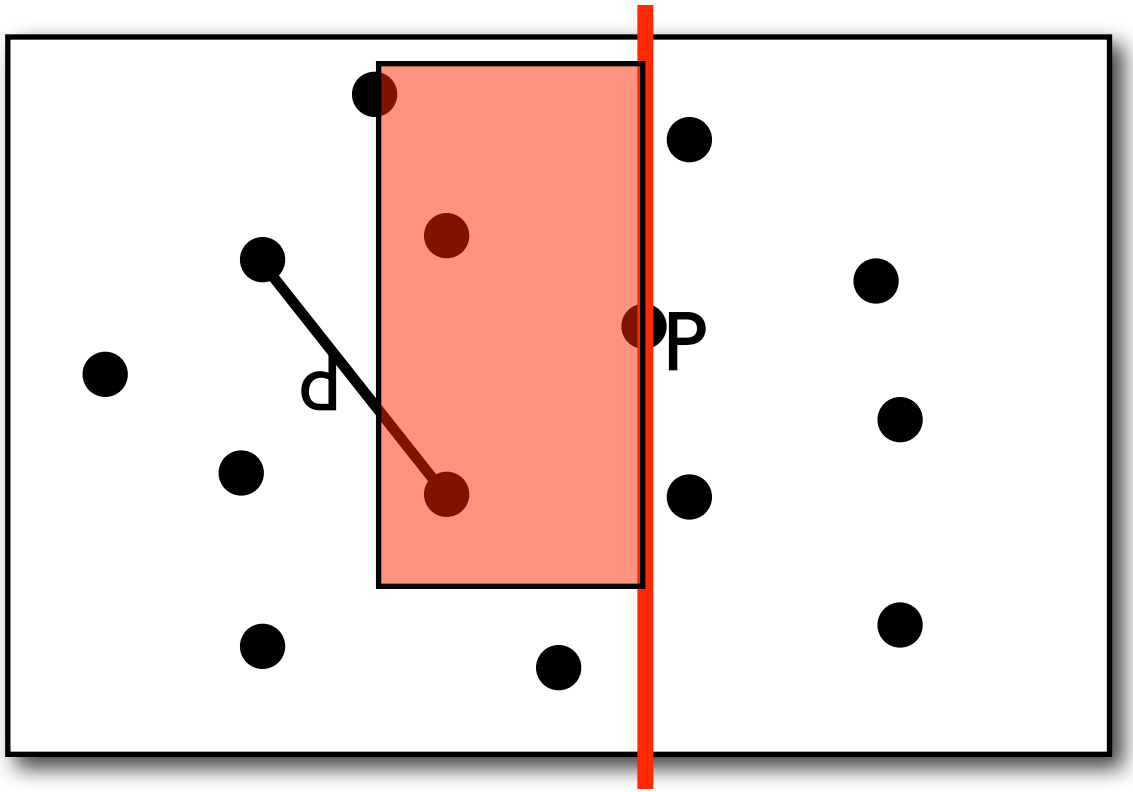
Algorithmische Geometrie

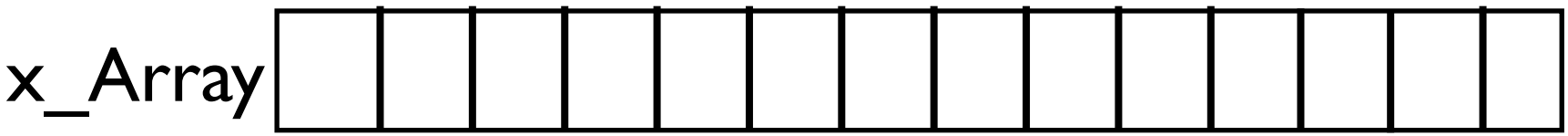
Kürzeste Abstände - Implementierung



Algorithmische Geometrie

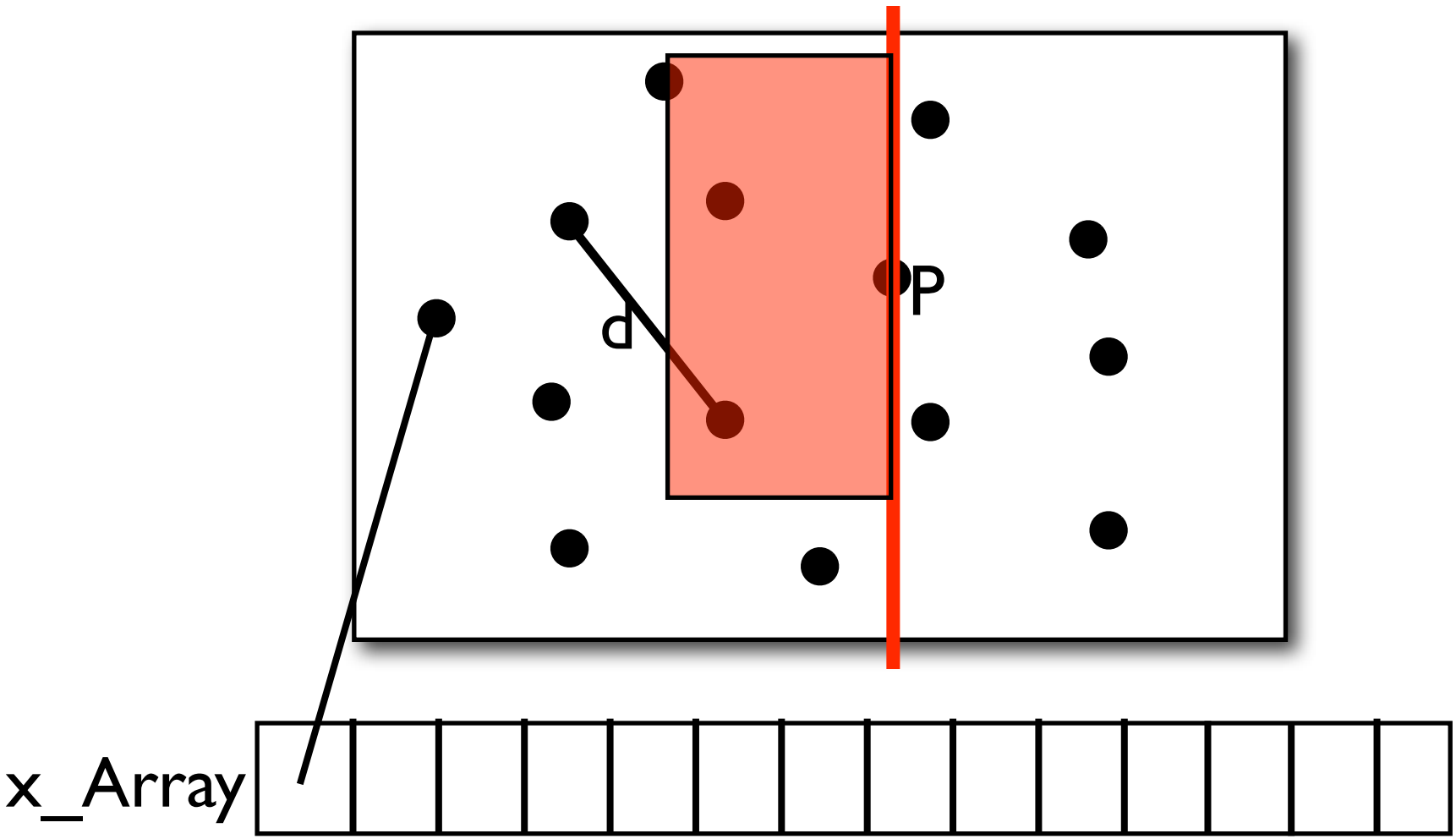
Kürzeste Abstände - Implementierung





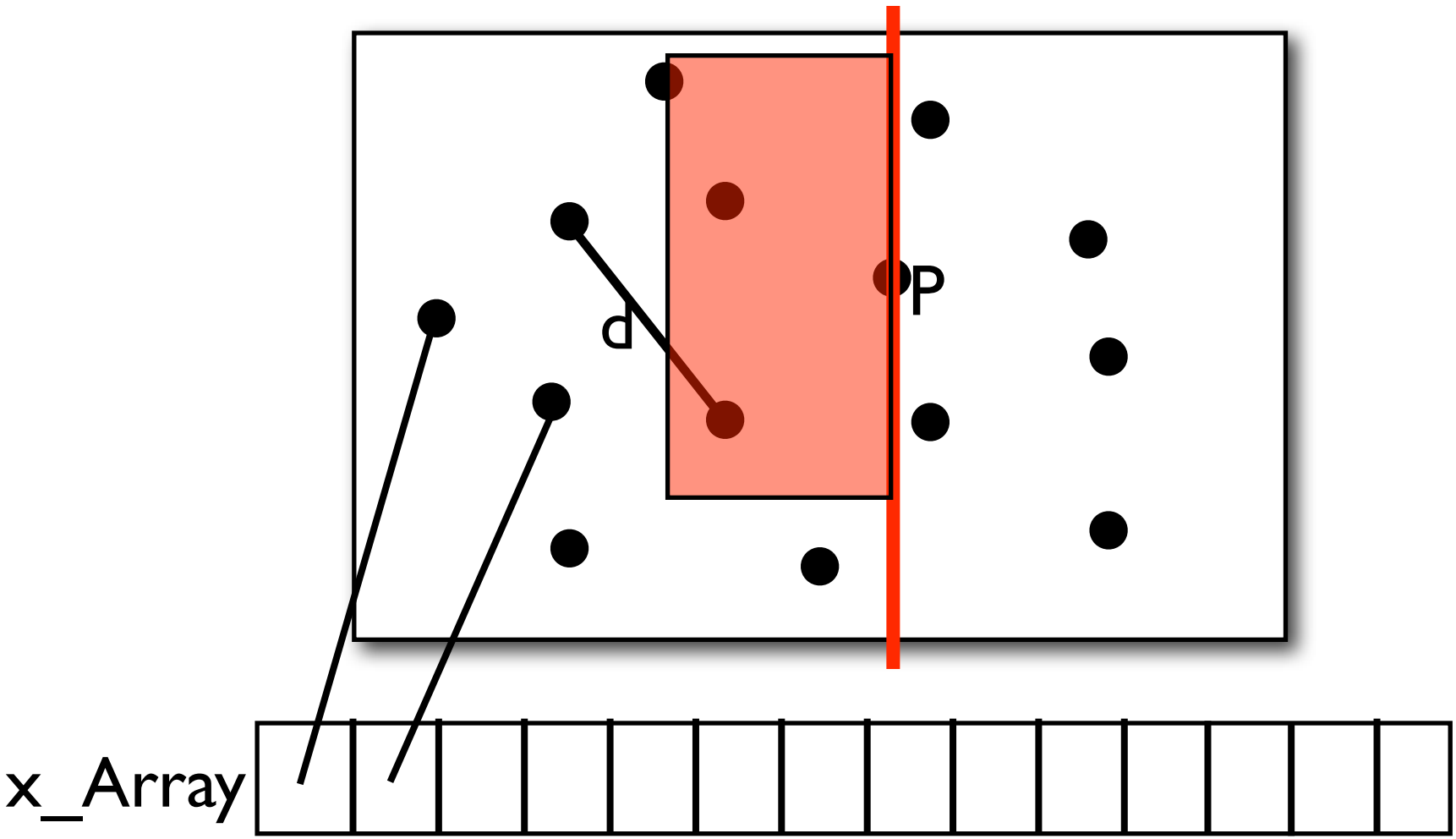
Algorithmische Geometrie

Kürzeste Abstände - Implementierung



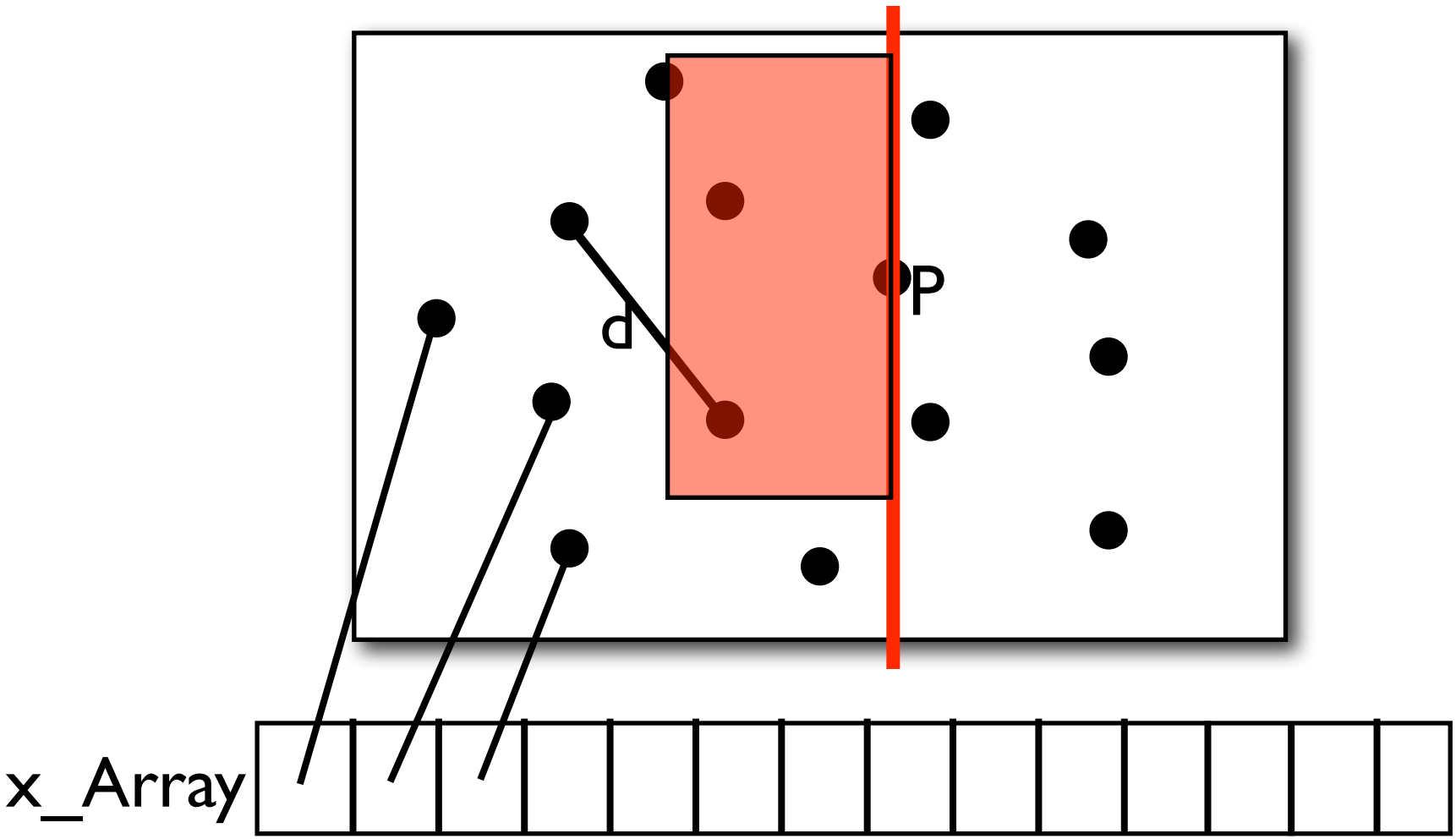
Algorithmische Geometrie

Kürzeste Abstände - Implementierung



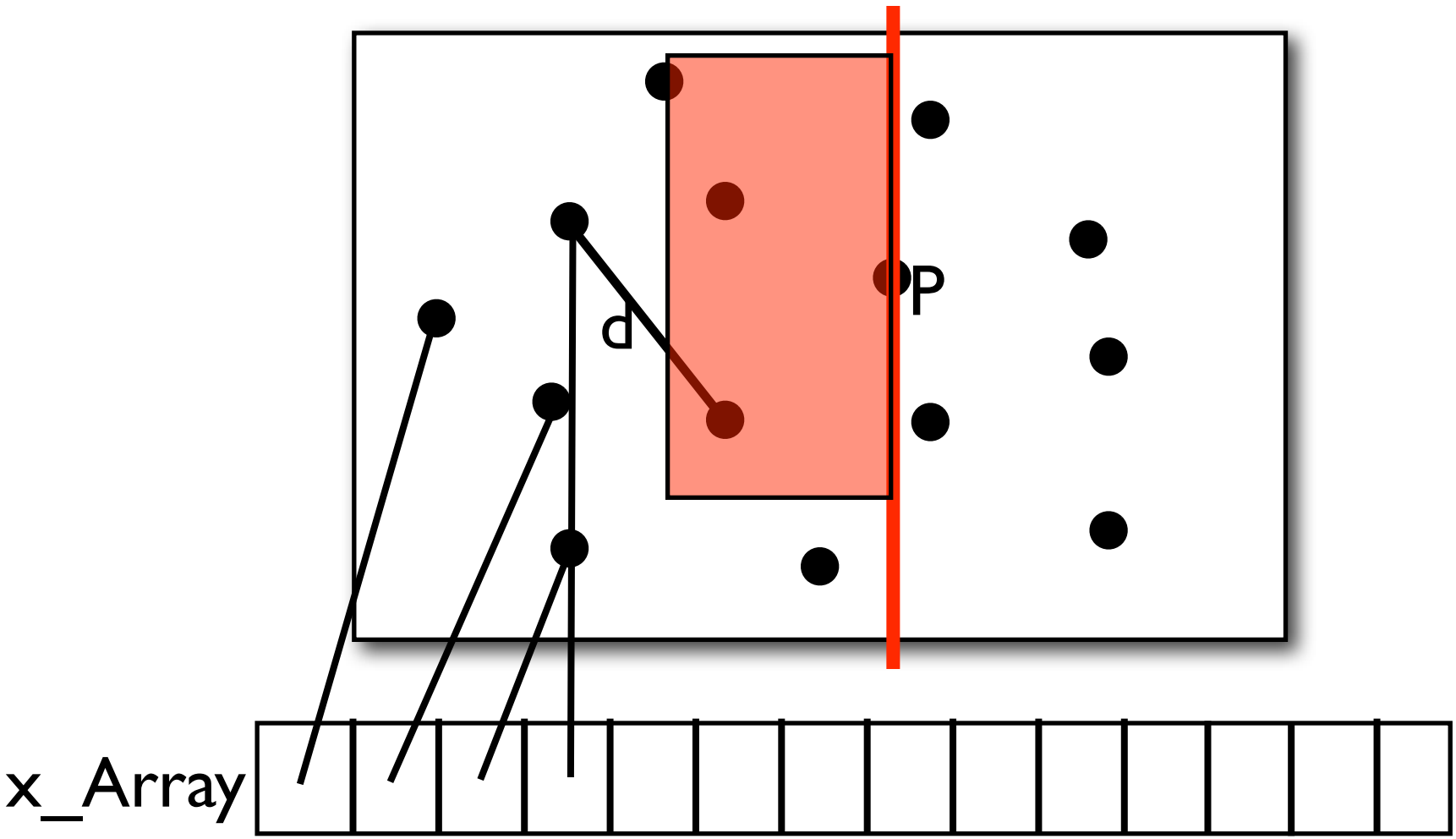
Algorithmische Geometrie

Kürzeste Abstände - Implementierung



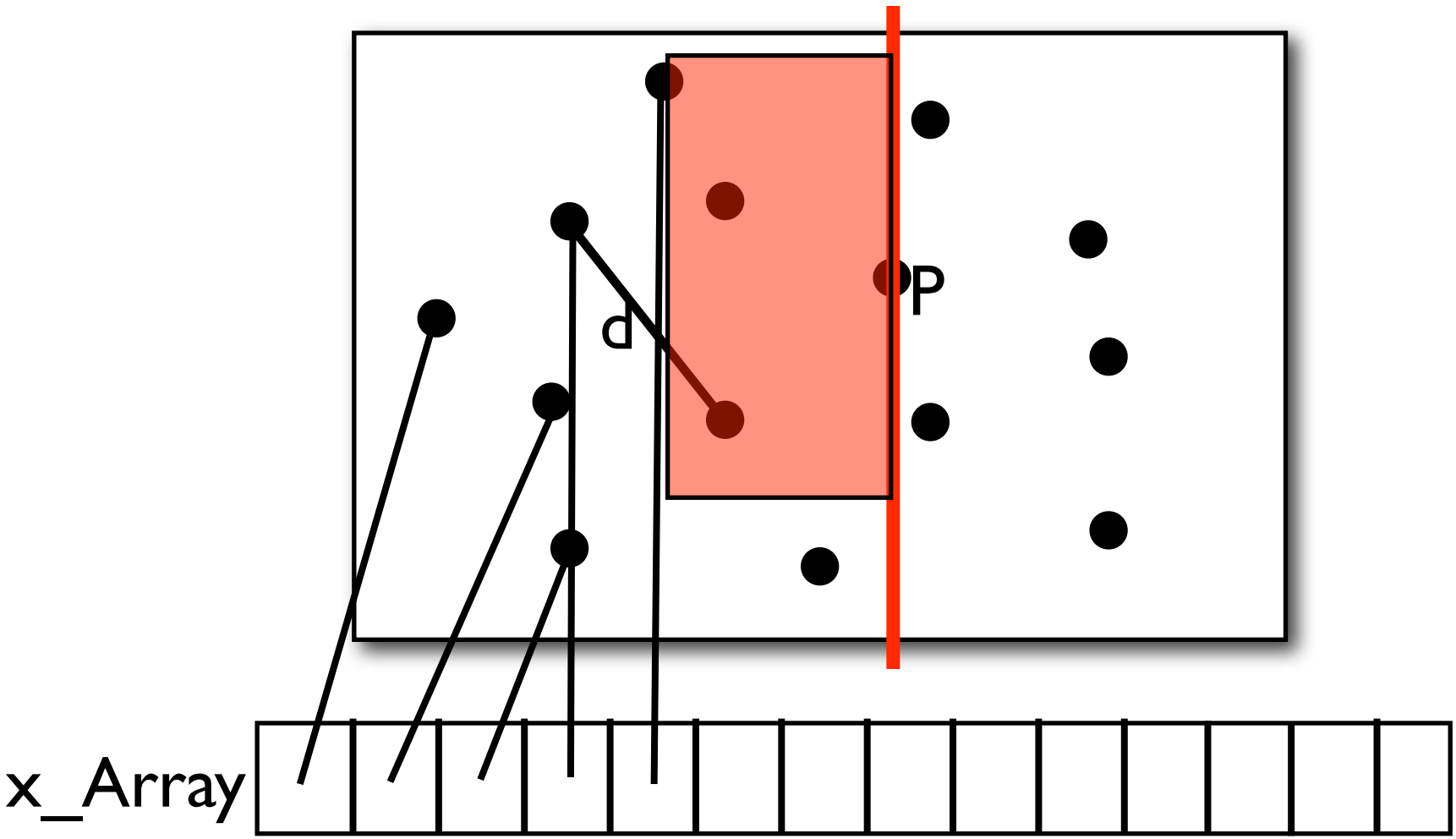
Algorithmische Geometrie

Kürzeste Abstände - Implementierung



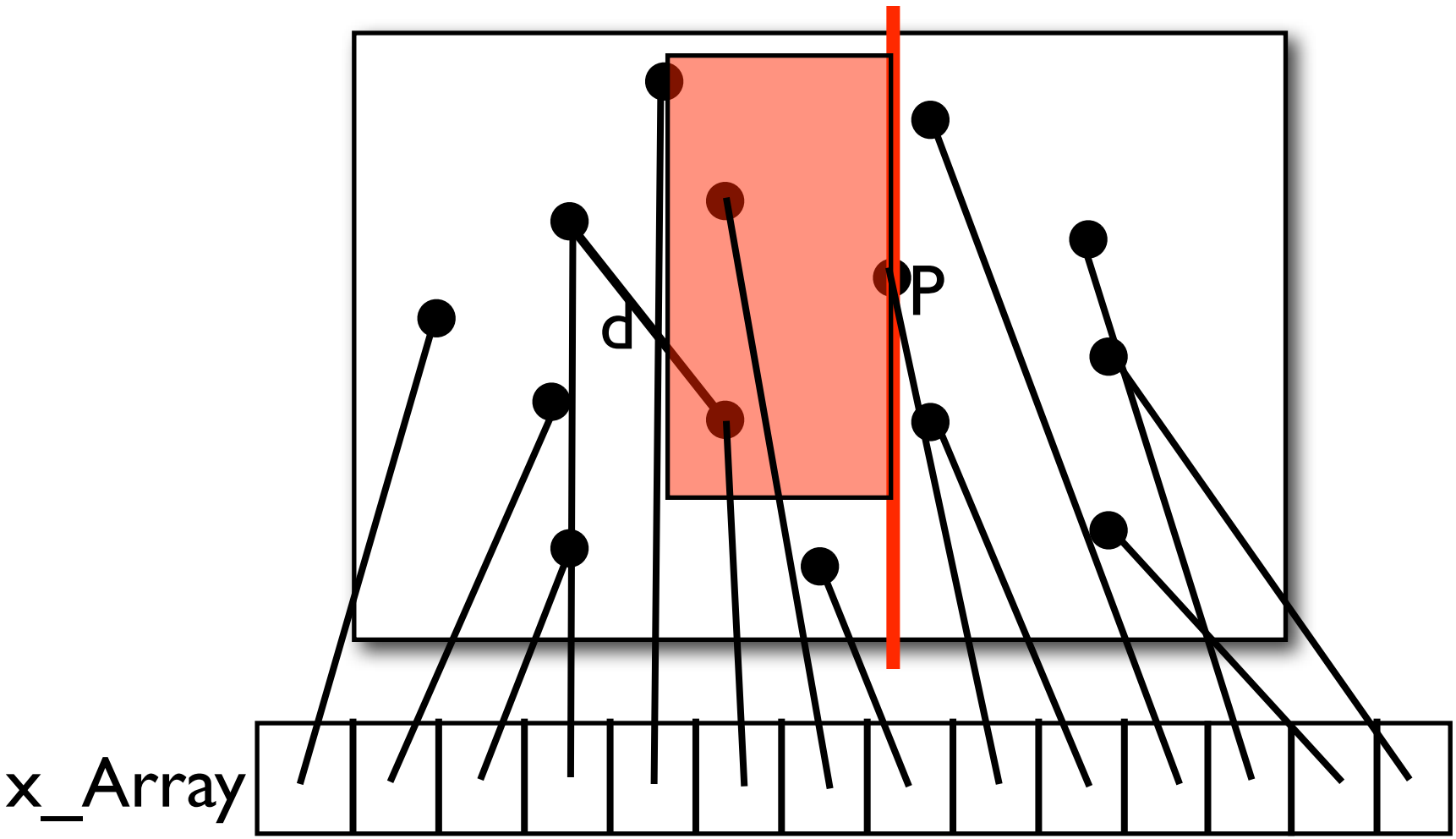
Algorithmische Geometrie

Kürzeste Abstände - Implementierung



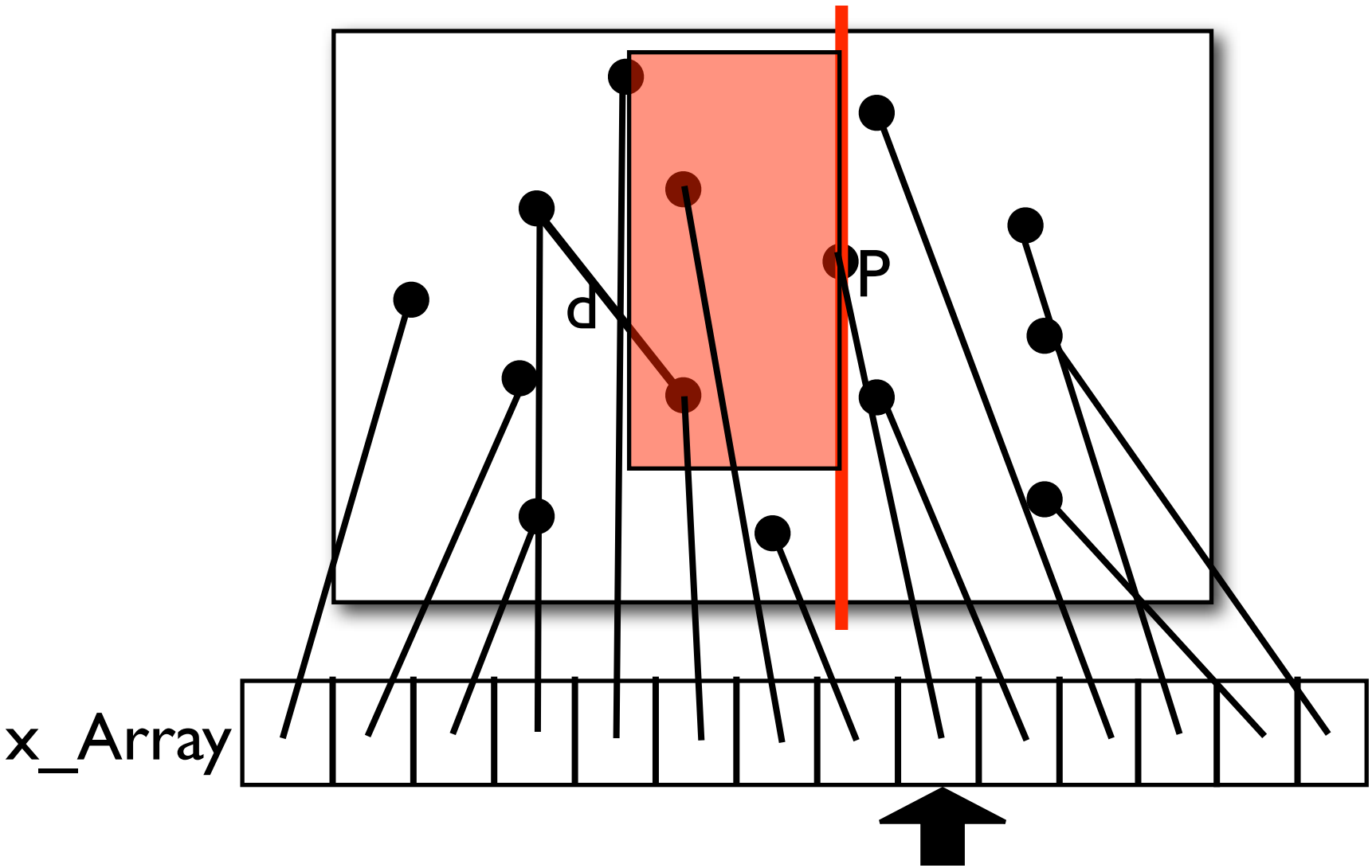
Algorithmische Geometrie

Kürzeste Abstände - Implementierung



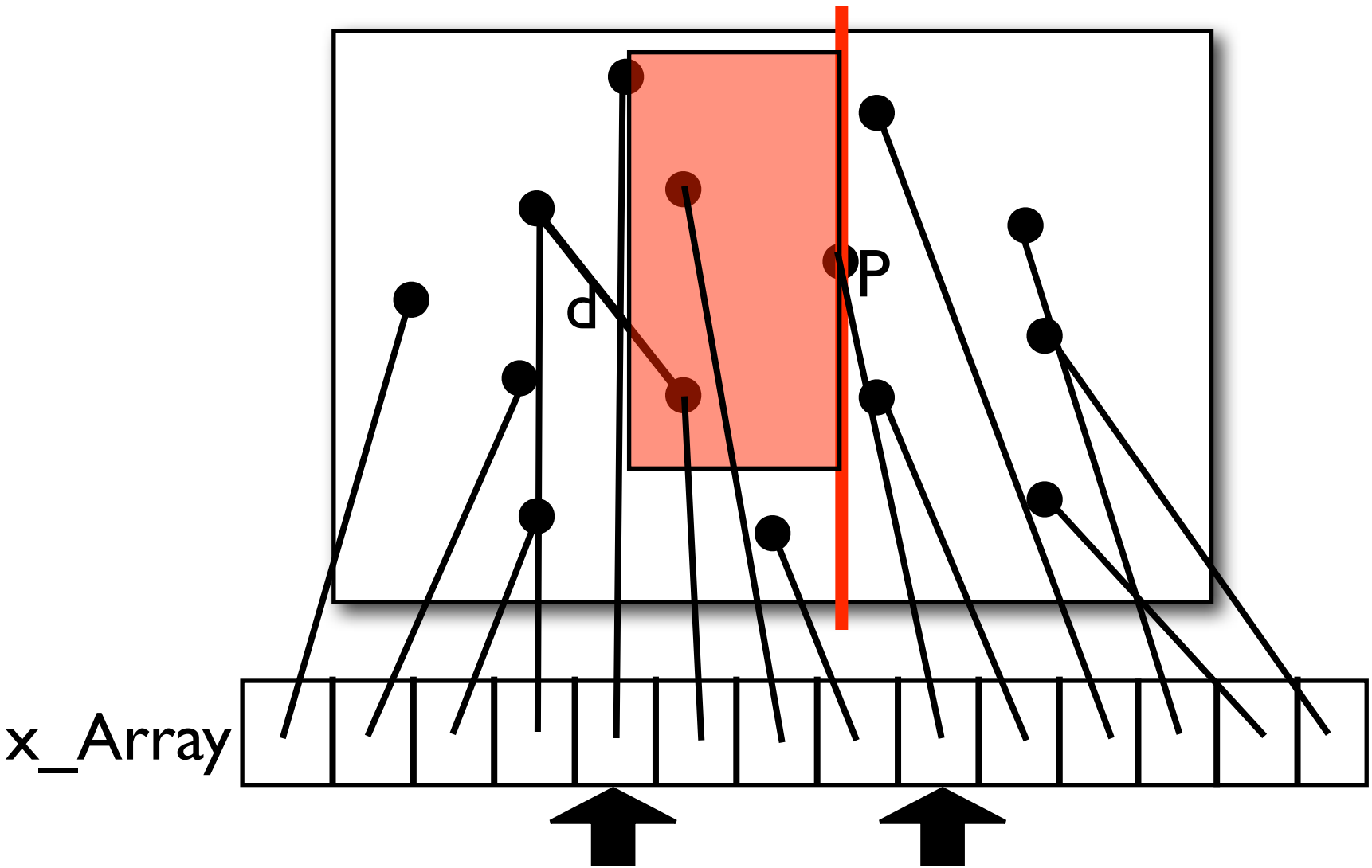
Algorithmische Geometrie

Kürzeste Abstände - Implementierung



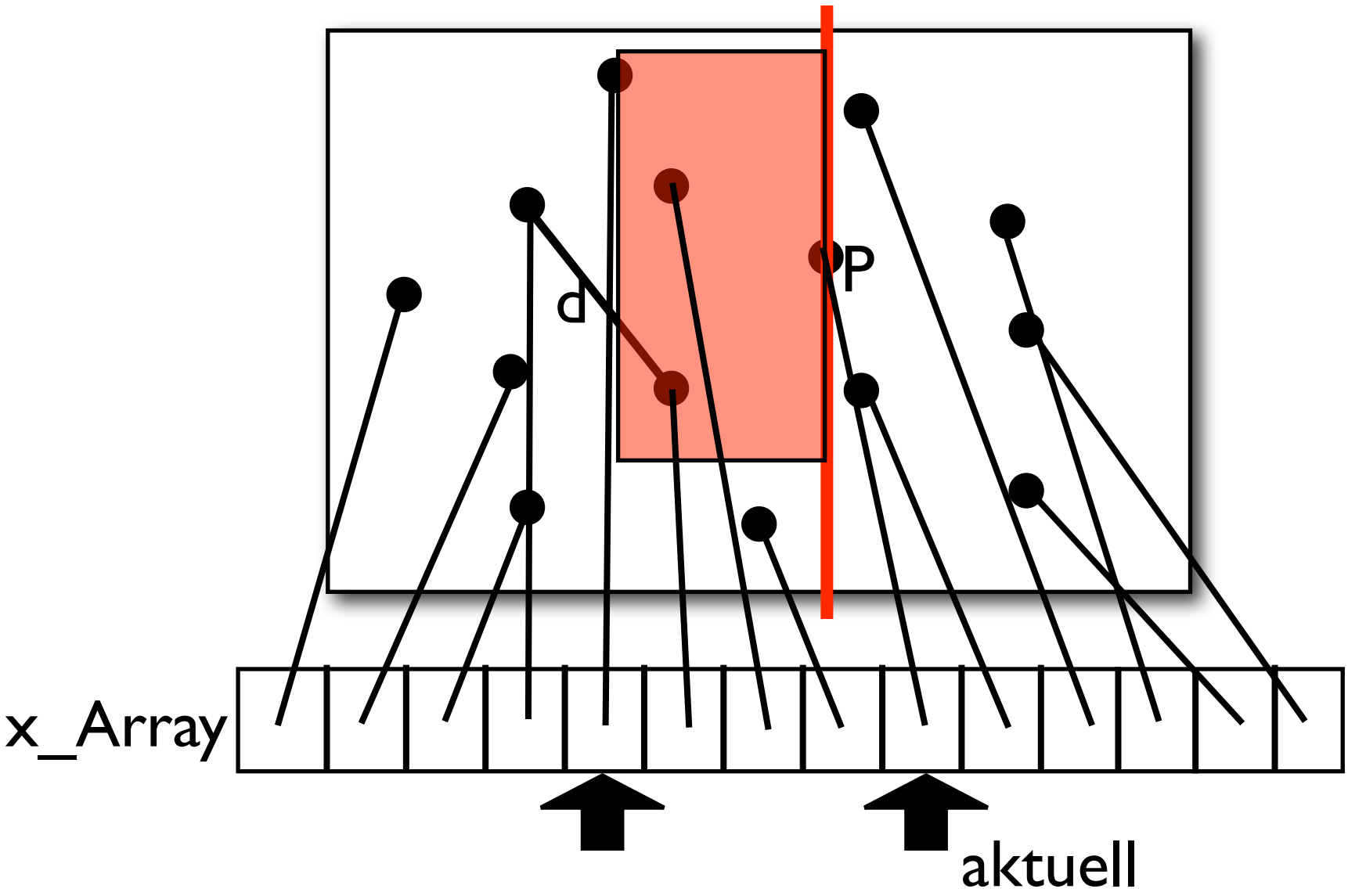
Algorithmische Geometrie

Kürzeste Abstände - Implementierung



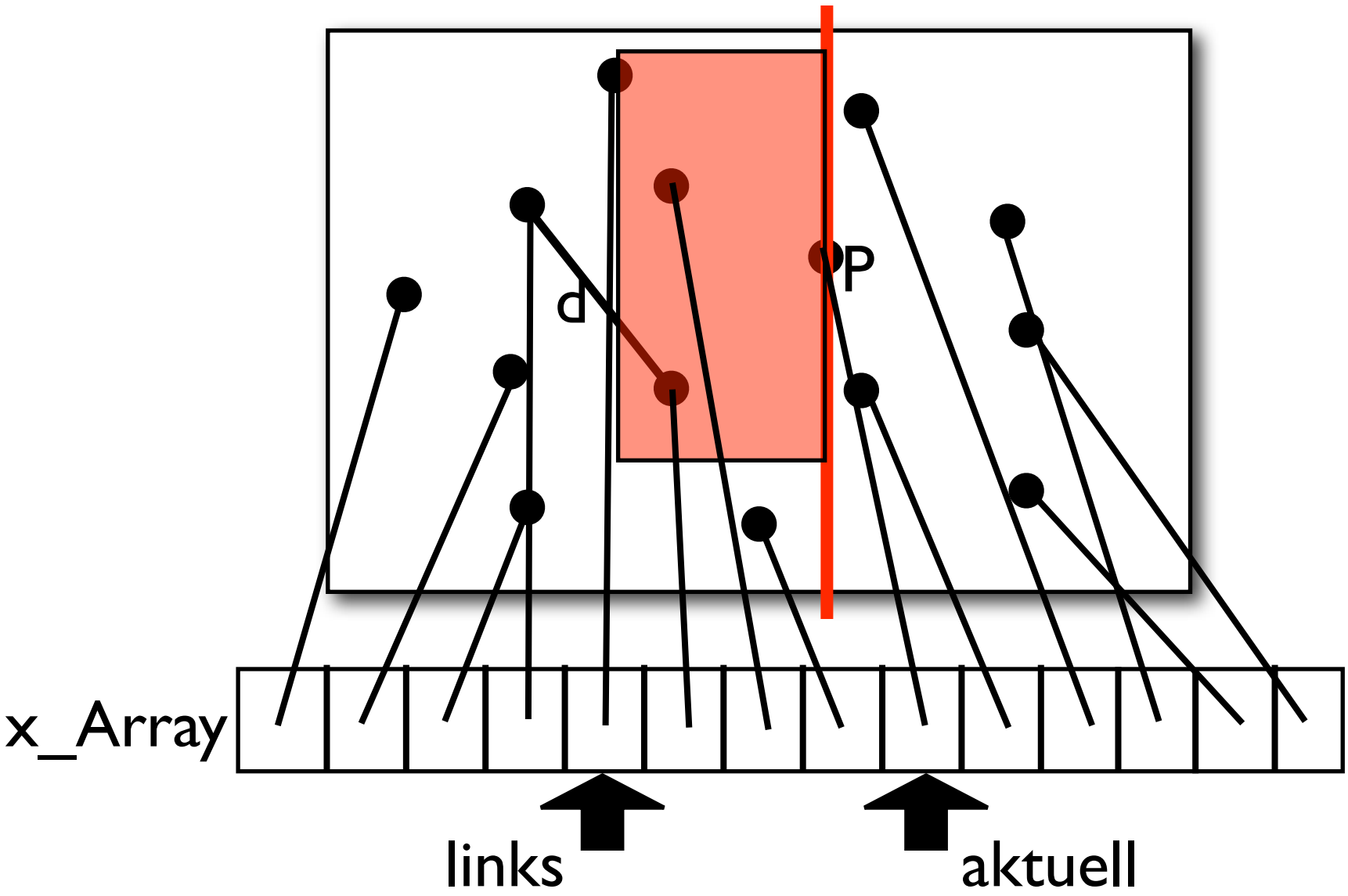
Algorithmische Geometrie

Kürzeste Abstände - Implementierung



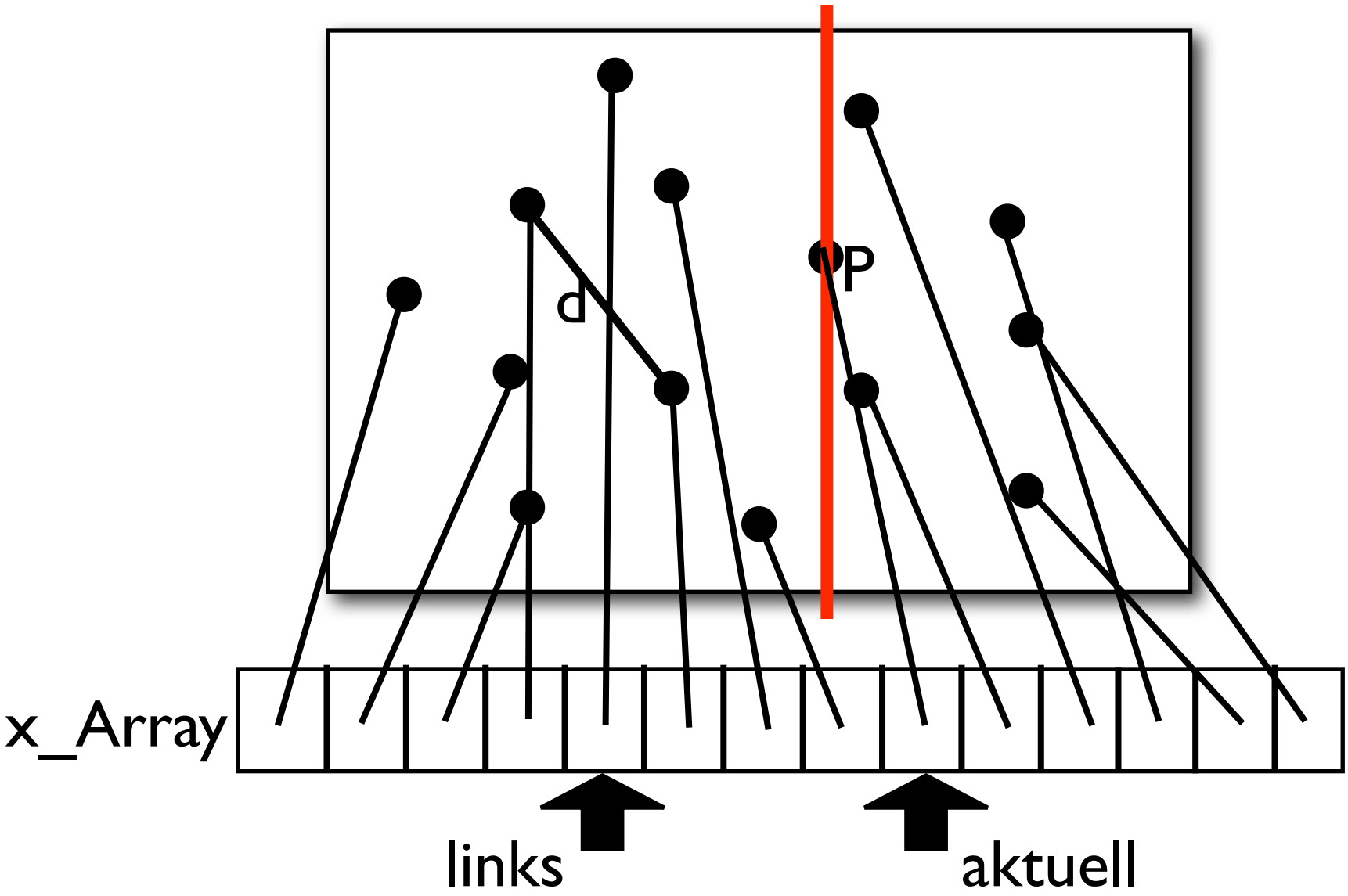
Algorithmische Geometrie

Kürzeste Abstände - Implementierung



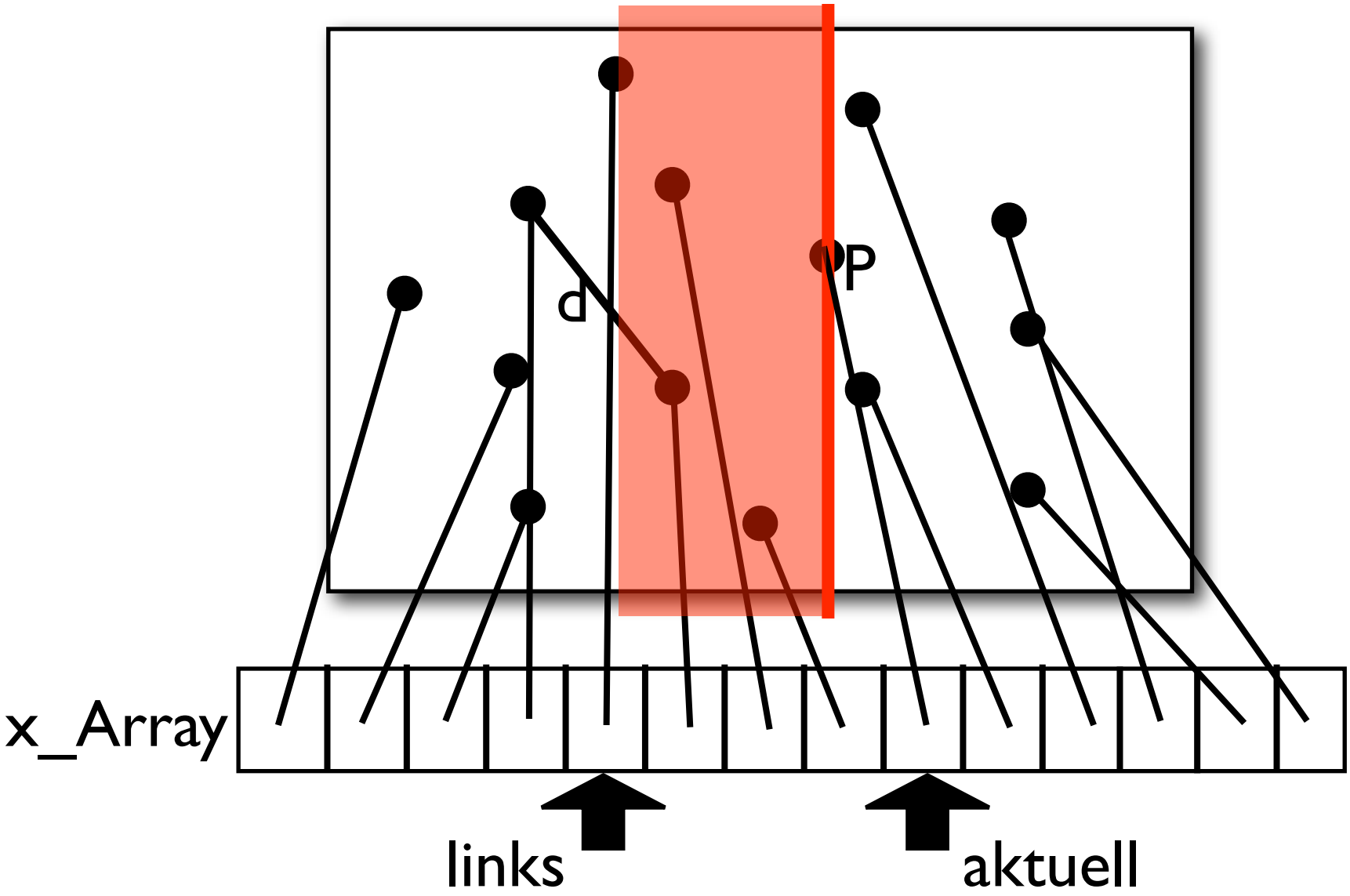
Algorithmische Geometrie

Kürzeste Abstände - Implementierung



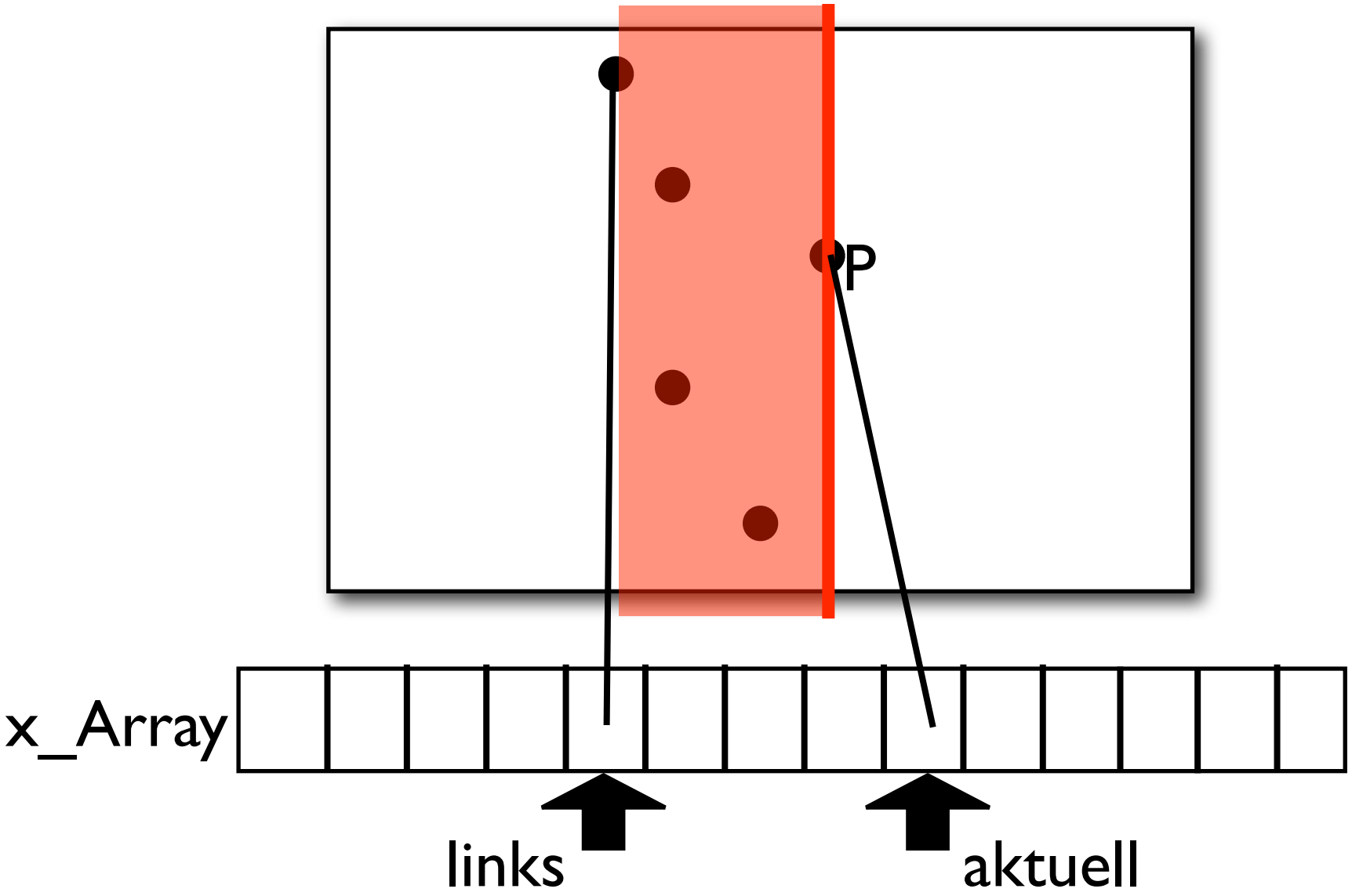
Algorithmische Geometrie

Kürzeste Abstände - Implementierung



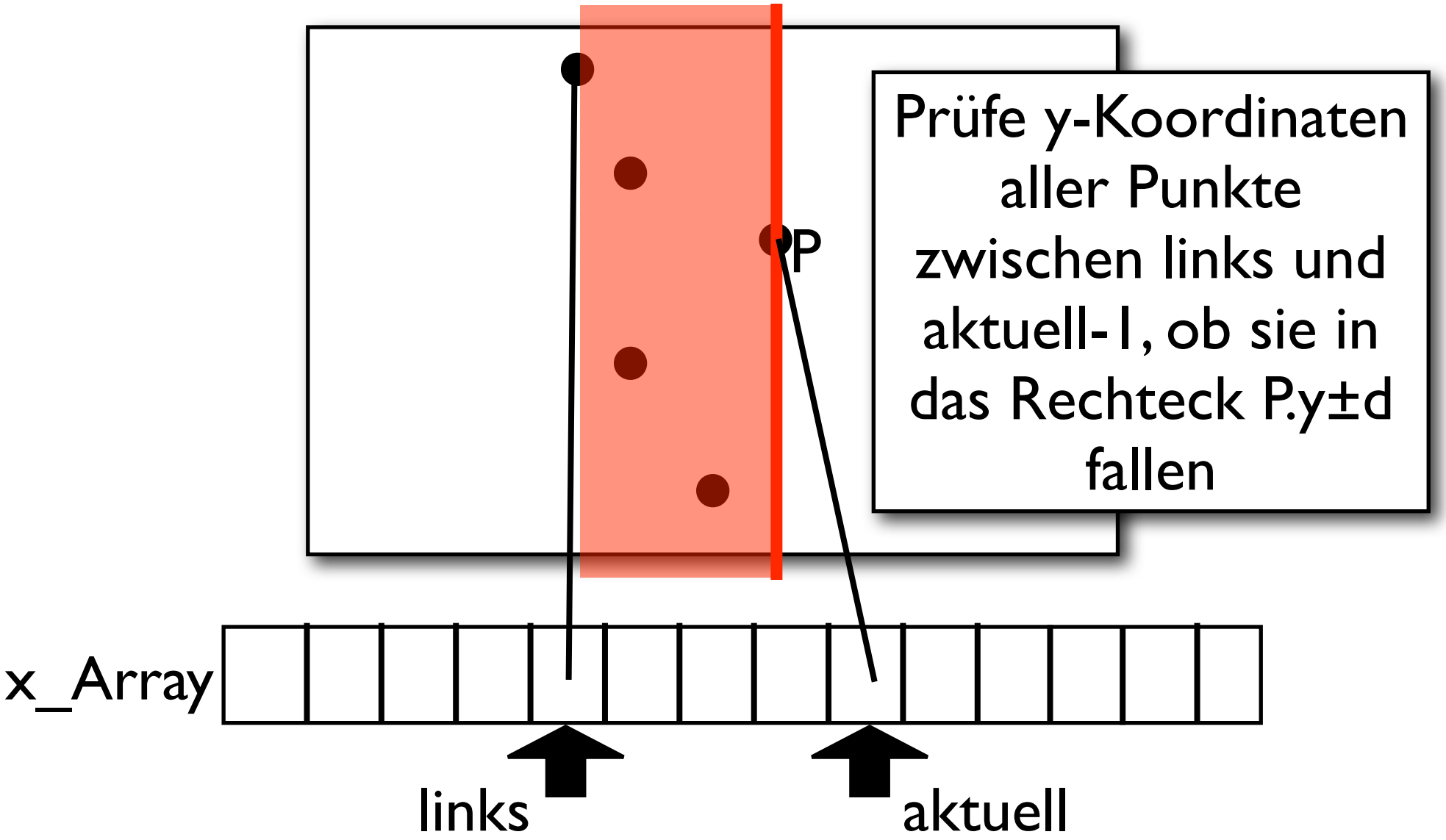
Algorithmische Geometrie

Kürzeste Abstände - Implementierung



Algorithmische Geometrie

Kürzeste Abstände - Implementierung



Algorithmische Geometrie
Kürzeste Abstände - Implementierung

- Anlegen einer Datenstruktur Y für die Punkte zwischen “links” und “aktuell”, sortiert nach y-Koordinate
- Datenstruktur wird dynamisch immer wieder mit den relevanten Punkten bestückt
- effizientes Einfügen, Löschen erforderlich
- effiziente **Bereichssuche** (*range query*) der Art: liefere alle Punkte P mit

  x_array[aktuell].y-d ≤ P.y ≤ x_array[aktuell].y+d
- Implementierung von Y: z.B. als AVL-Baum mit linearer Verkettung der Knoten (|Y|≤10)

Algorithmische Geometrie
Kürzeste Abstände - Algorithmus

Initialisierung:

Sortiere Punkte aufsteigend nach x-Koordinate in x_Array;

füge P_1, P_2 in Y ein;

d:=d(P_1, P_2);

links:=1; aktuell:=3;

Algorithmische Geometrie
Kürzeste Abstände - Algorithmus

Schleife:

```
while aktuell≤n do
    if x_array[links].x+d≤x_array[aktuell].x then
        entferne Punkt x_array[links] aus Y;
        links:=links+1
    else
        füge Punkt x_array[aktuell] in Y ein;
        aktuell:=aktuell+1;
        d:=Min_d(Y,x_array[aktuell],d)

write(d).
```

Algorithmische Geometrie
Kürzeste Abstände - Algorithmus

Schleife:

```
while aktuell≤n do
    if x_array[links].x+d≤x_array[aktuell].x then
        entferne Punkt x_array[links] aus Y;
        links:=links+1
    else
        füge Punkt x_array[aktuell] in Y ein;
        aktuell:=aktuell+1;
        d:=Min_d(Y,x_array[aktuell],d)

write(d).
```

range query mit Laufzeit O(log n)

Algorithmische Geometrie
Konvexe Hülle - Problemstellung

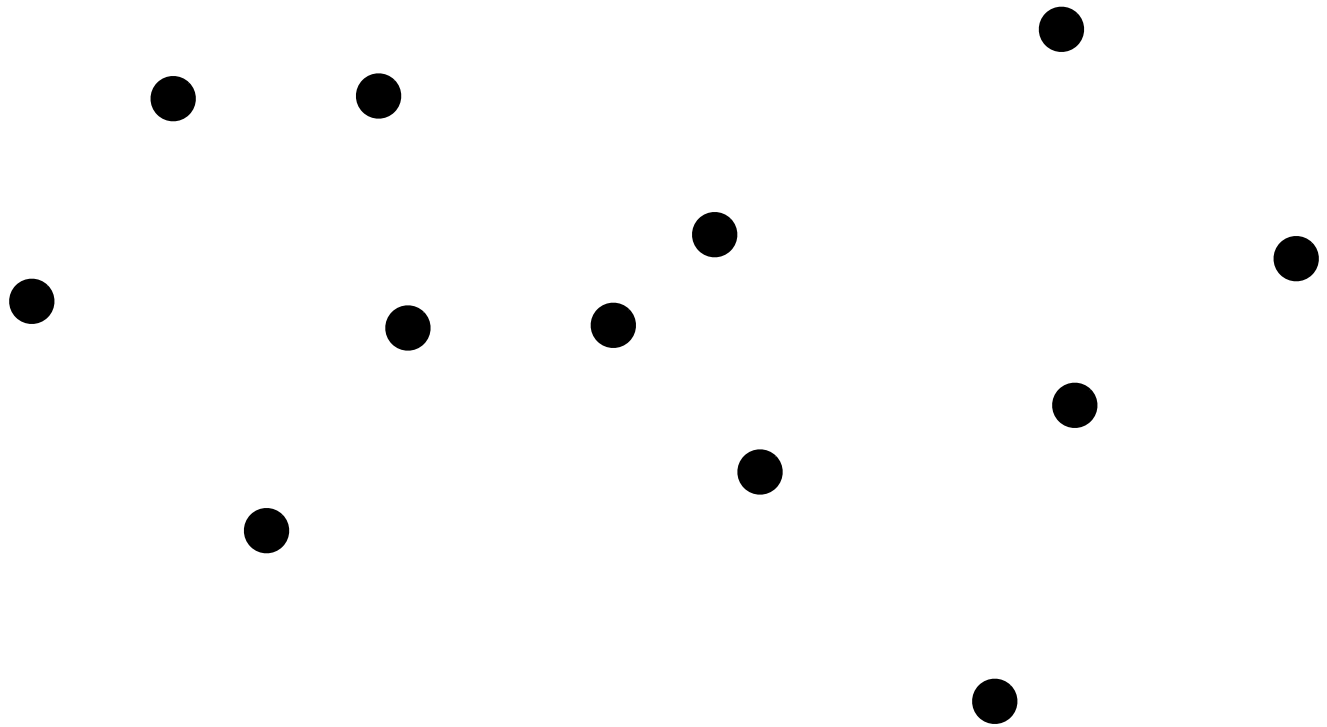
- Gegeben: n Punkte im d-dimensionalen Raum (hier nur d=2)
- Gesucht: kleinste konvexe Hülle, die alle Punkte enthält (Ausgabe der zur Hülle gehörenden Punkte im Uhrzeigersinn)
- Eselbrücke: kürzester Zaun um Goldminen

Algorithmische Geometrie

Konvexe Hülle - Beispiel

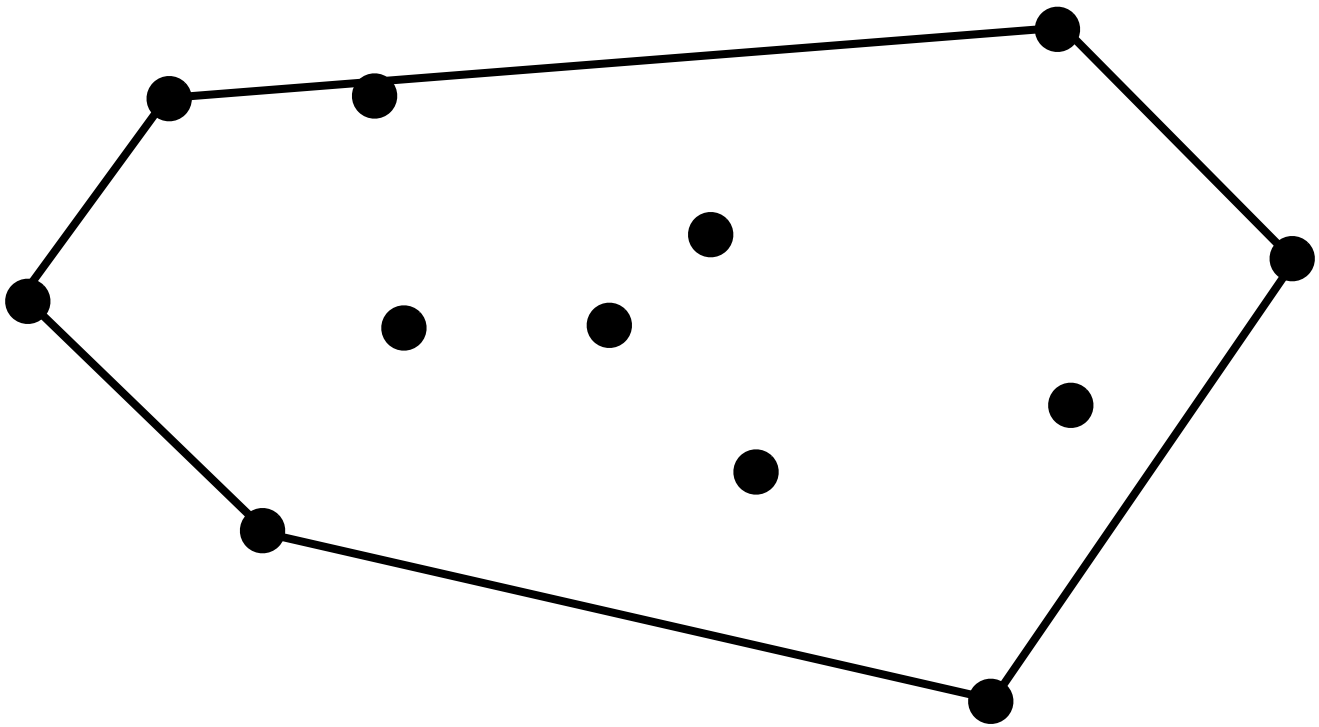
Algorithmische Geometrie

Konvexe Hülle - Beispiel



Algorithmische Geometrie

Konvexe Hülle - Beispiel



Algorithmische Geometrie
Konvexe Hülle - Konvexität

Definition

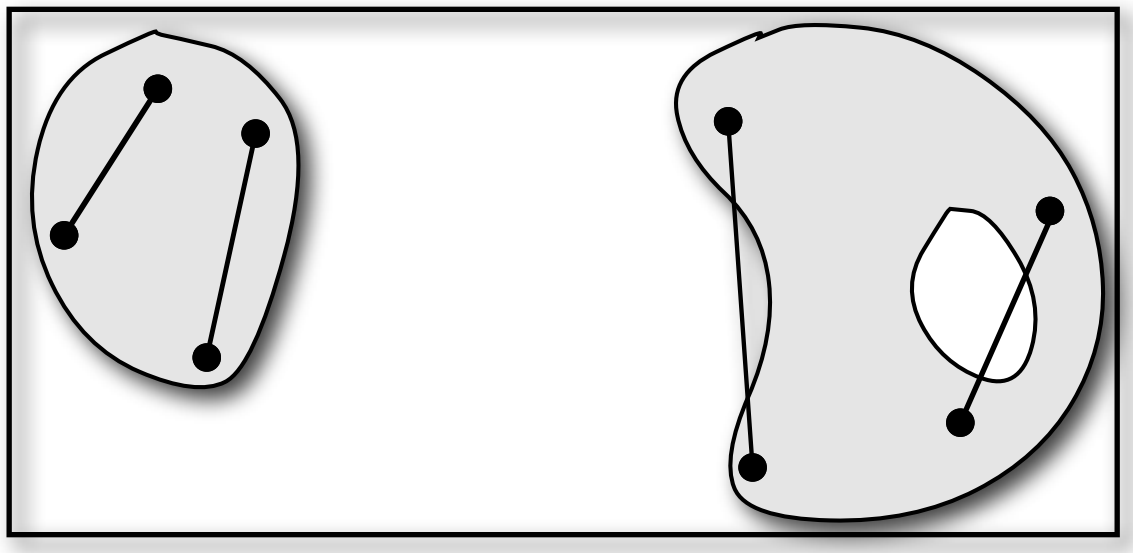
Eine Menge $M \subseteq \mathbb{R}^n$ heißt **konvex**, wenn mit je zwei Punkten $x,y \in M$ die gesamte Strecke von x nach y in M liegt.

Algorithmische Geometrie
Konvexe Hülle - Konvexität

Definition

Eine Menge $M \subseteq \mathbb{R}^n$ heißt **konvex**, wenn mit je zwei Punkten $x,y \in M$ die gesamte Strecke von x nach y in M liegt.

konvex



nicht konvex

Algorithmische Geometrie
Konvexe Hülle - Konvexität

Definition

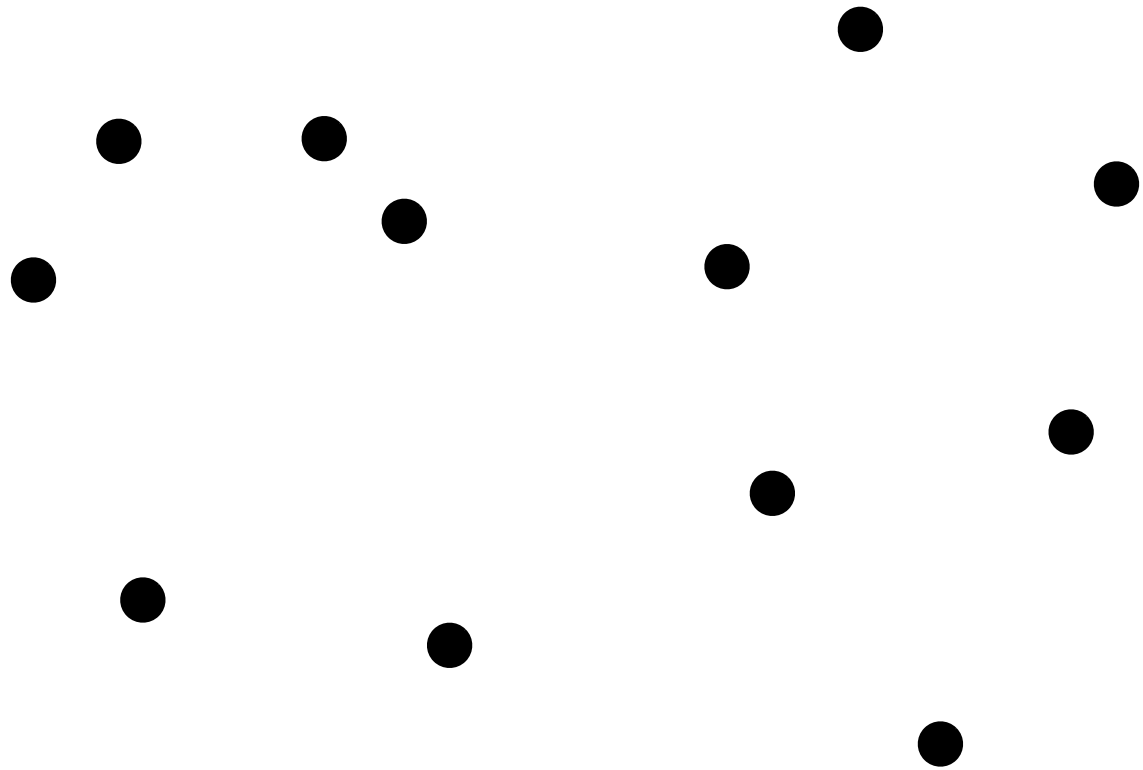
Die **konvexe Hülle** einer Menge $M \subseteq \mathbb{R}^n$ ist die kleinste konvexe Menge, die M enthält.

Algorithmische Geometrie
Konvexe Hülle - Lösungsidee

- Divide-and-conquer-Verfahren
  - Divide: Teile Punkteschar in zwei etwa gleichgroße Teile
  - Bestimme konvexe Hülle für jede Teilpunktemenge
  - Conquer: Verschmelze beide konvexen Hüllen zu einer einzigen für die Gesamtpunktemenge

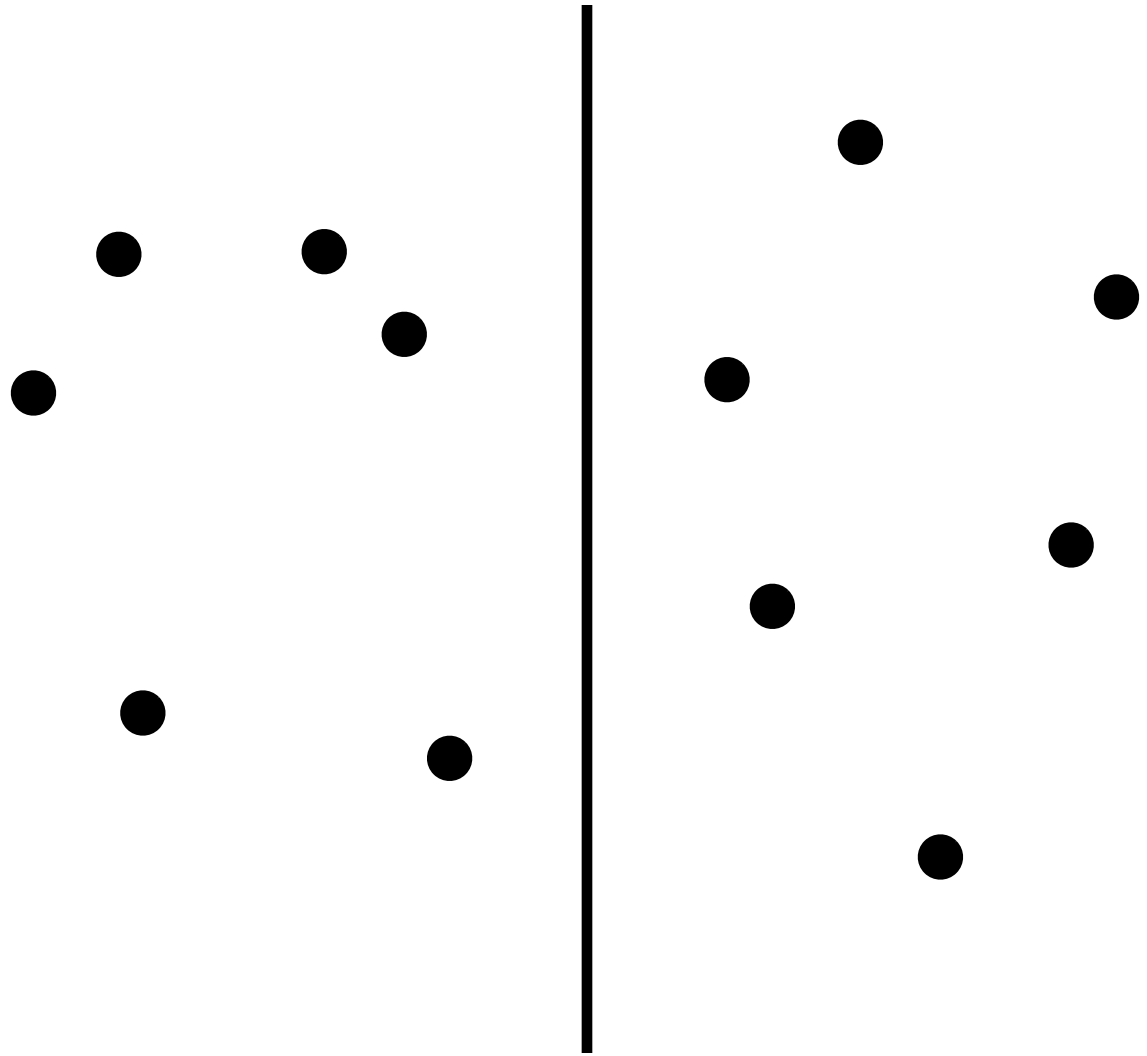
Algorithmische Geometrie

Konvexe Hülle - Divide



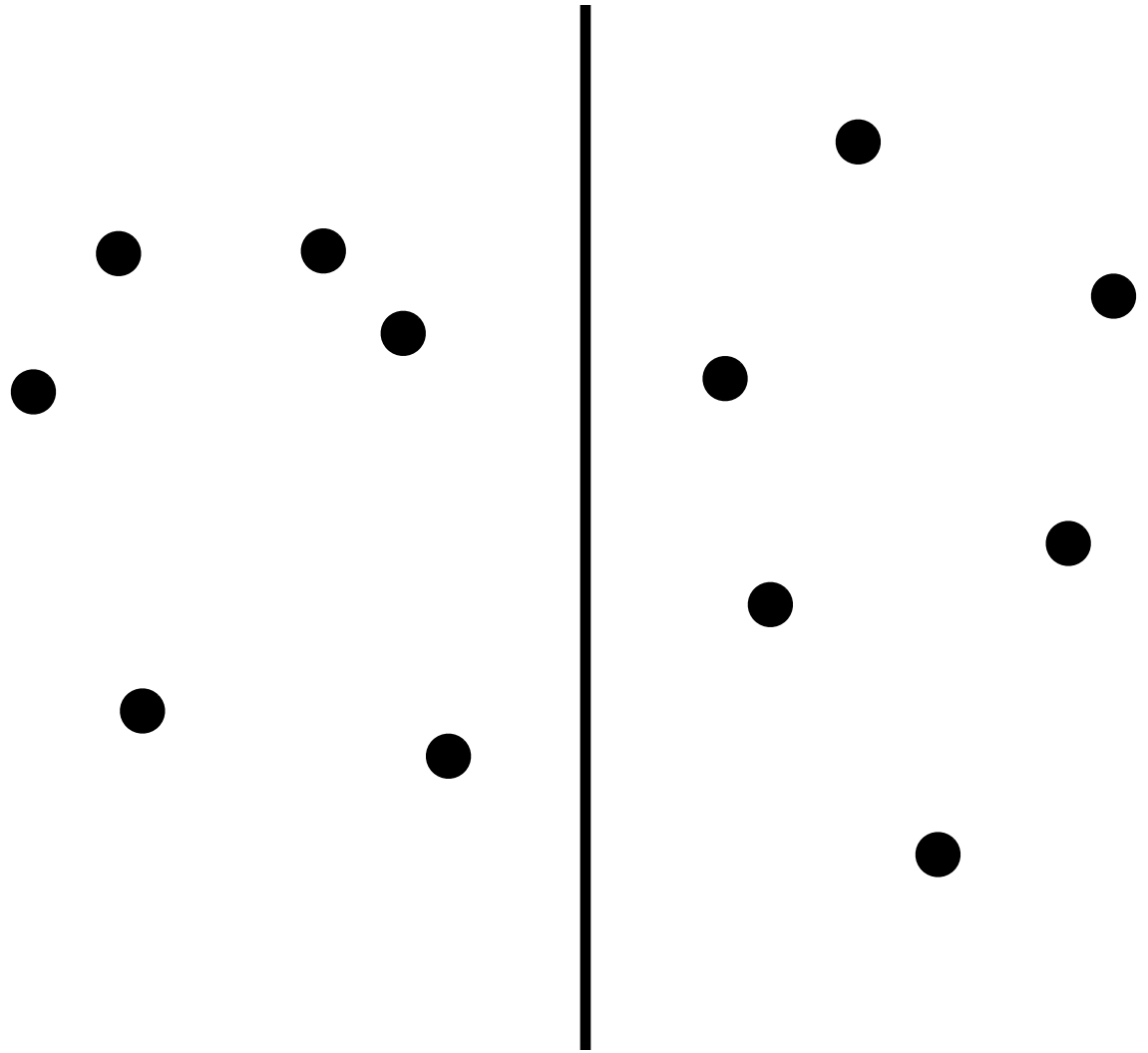
Algorithmische Geometrie

Konvexe Hülle - Divide



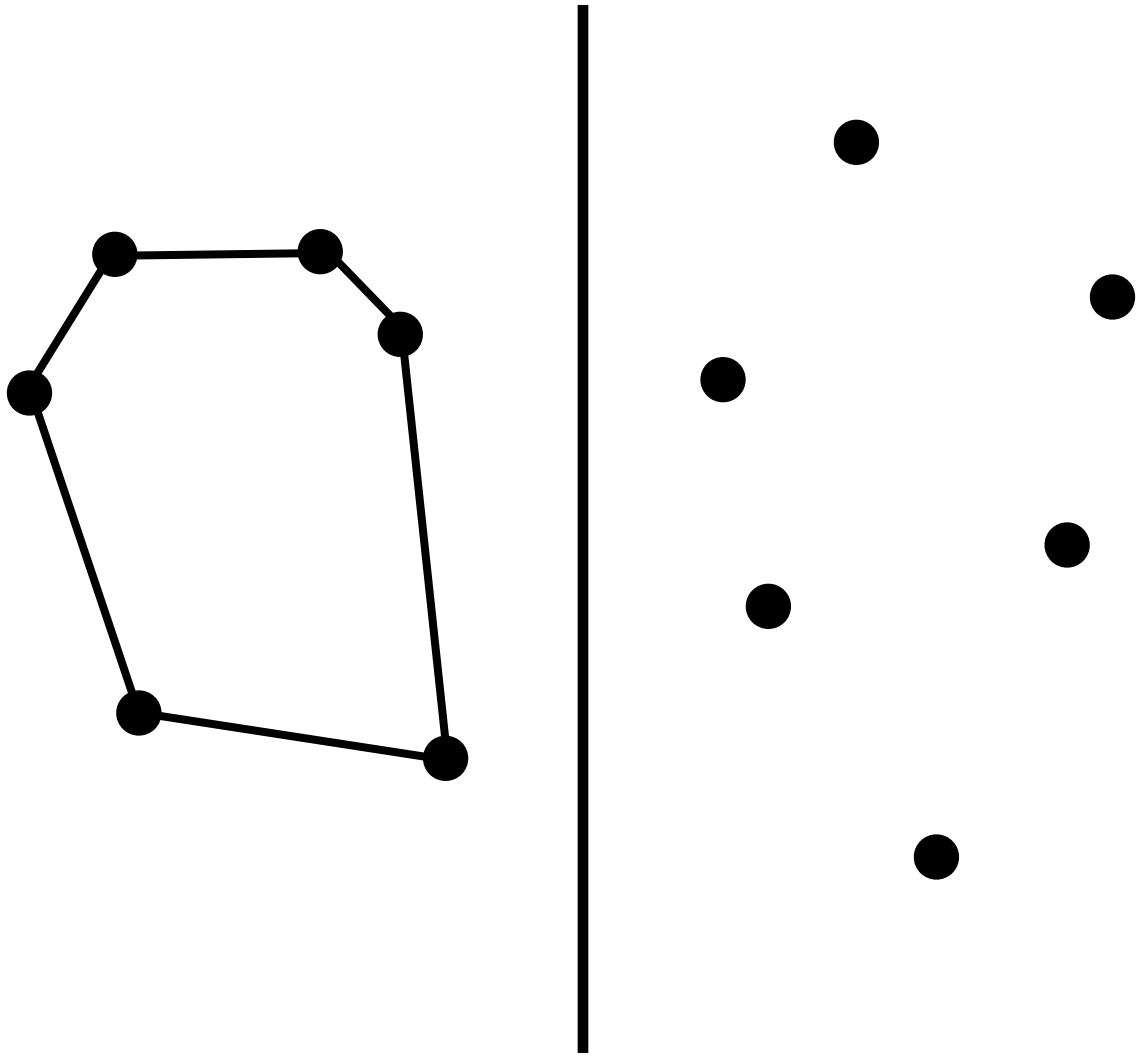
Algorithmische Geometrie

Konvexe Hülle - Teillösungen



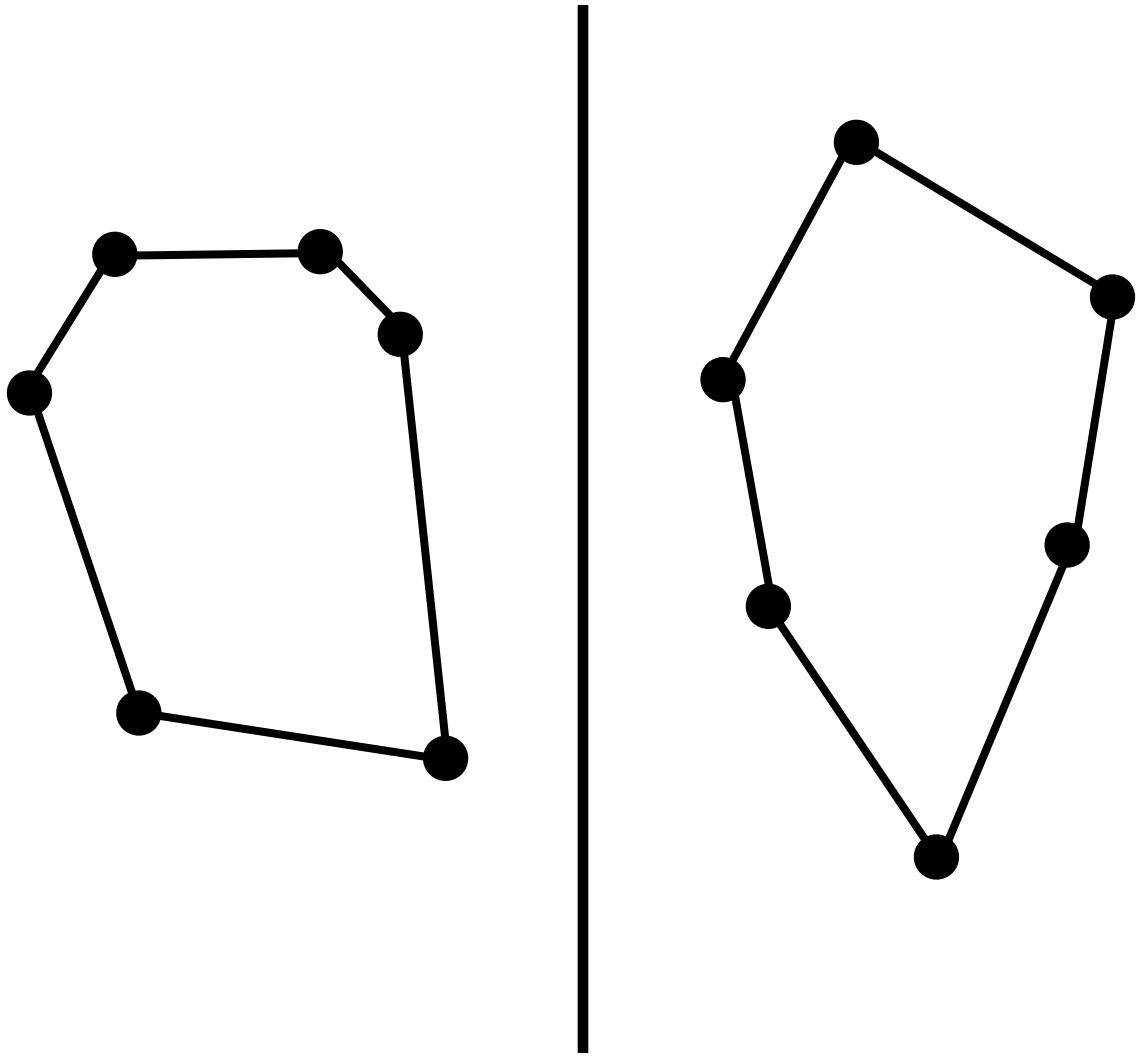
Algorithmische Geometrie

Konvexe Hülle - Teillösungen



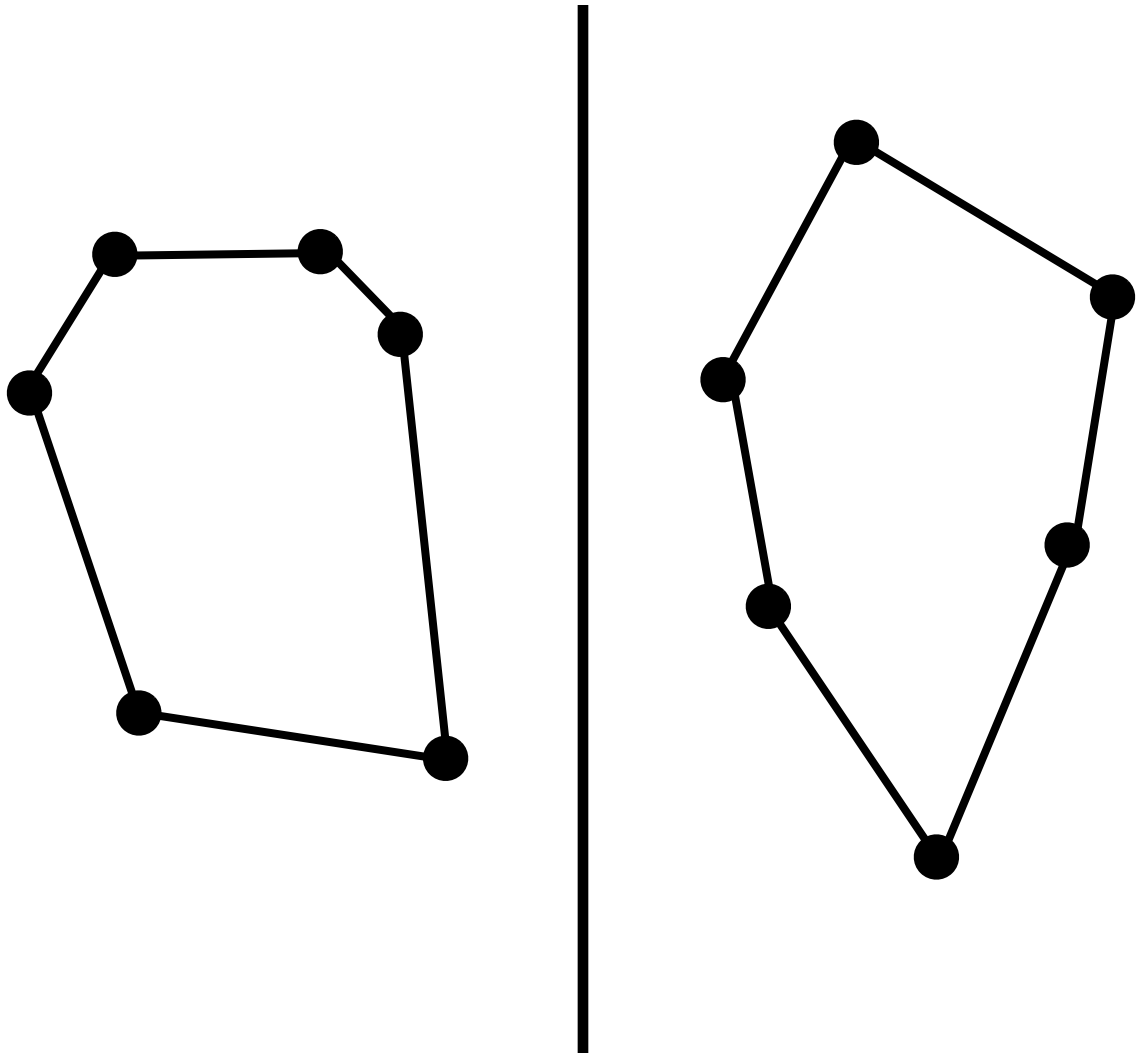
Algorithmische Geometrie

Konvexe Hülle - Teillösungen



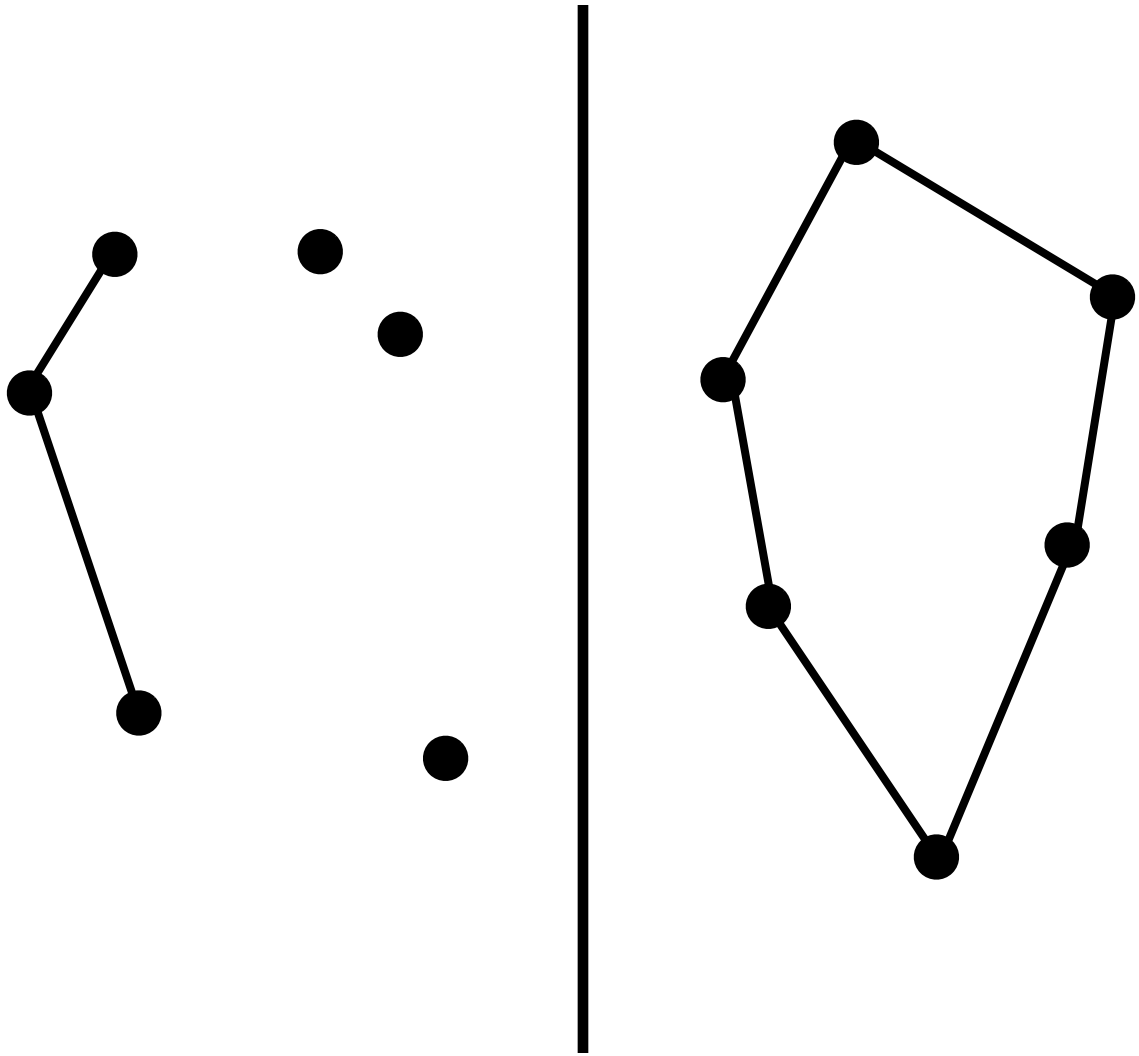
Algorithmische Geometrie

Konvexe Hülle - Conquer



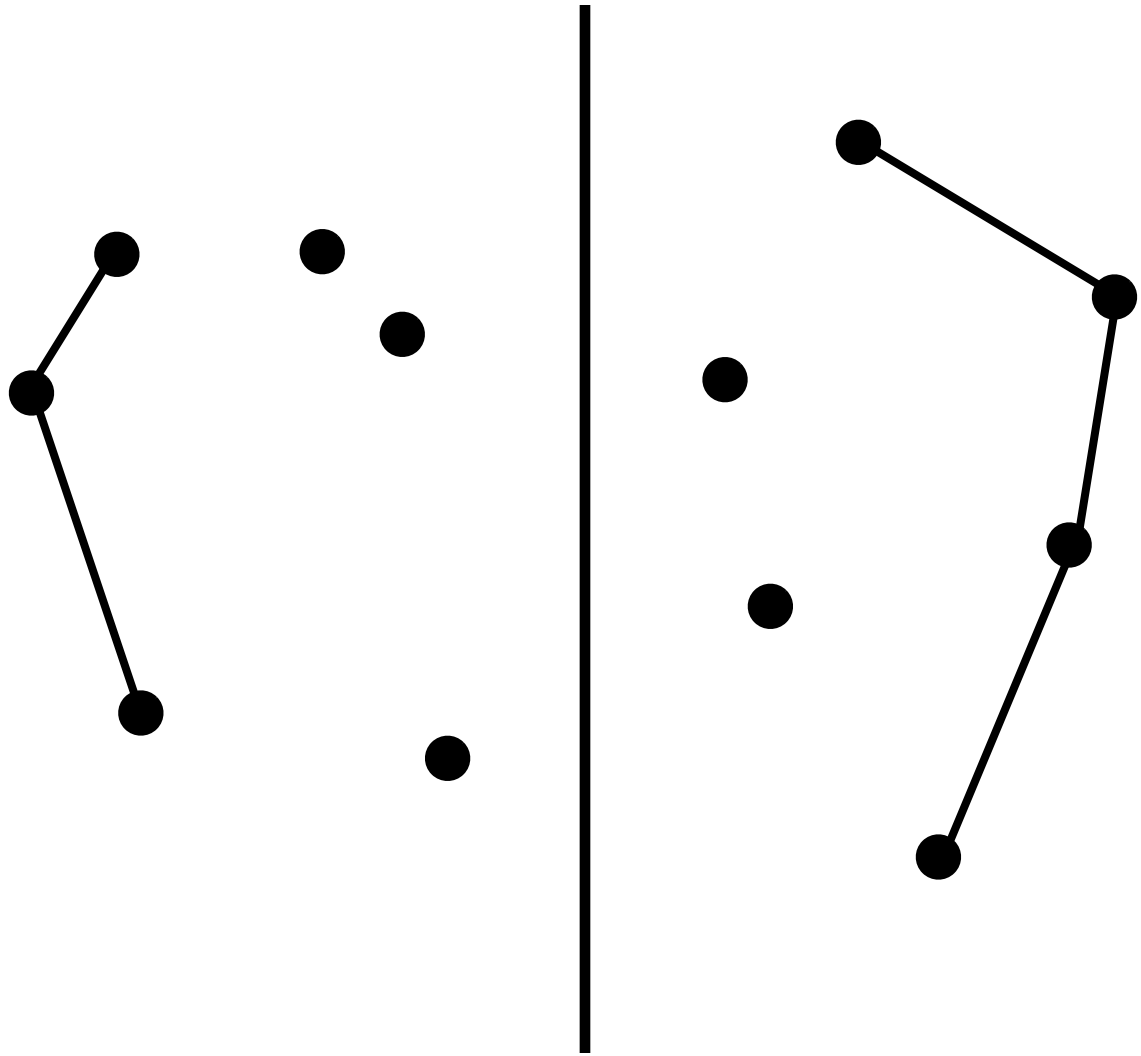
Algorithmische Geometrie

Konvexe Hülle - Conquer



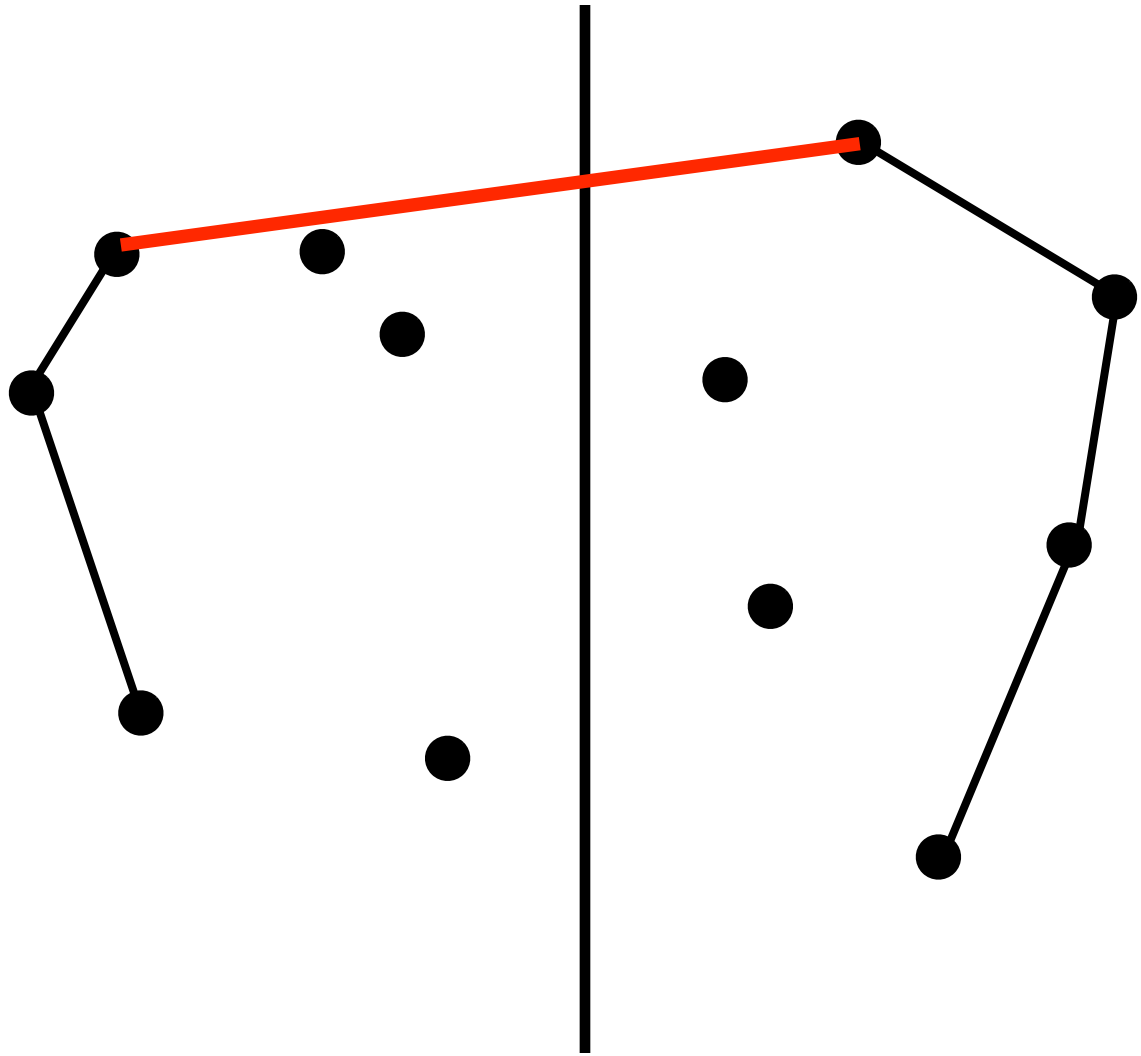
Algorithmische Geometrie

Konvexe Hülle - Conquer



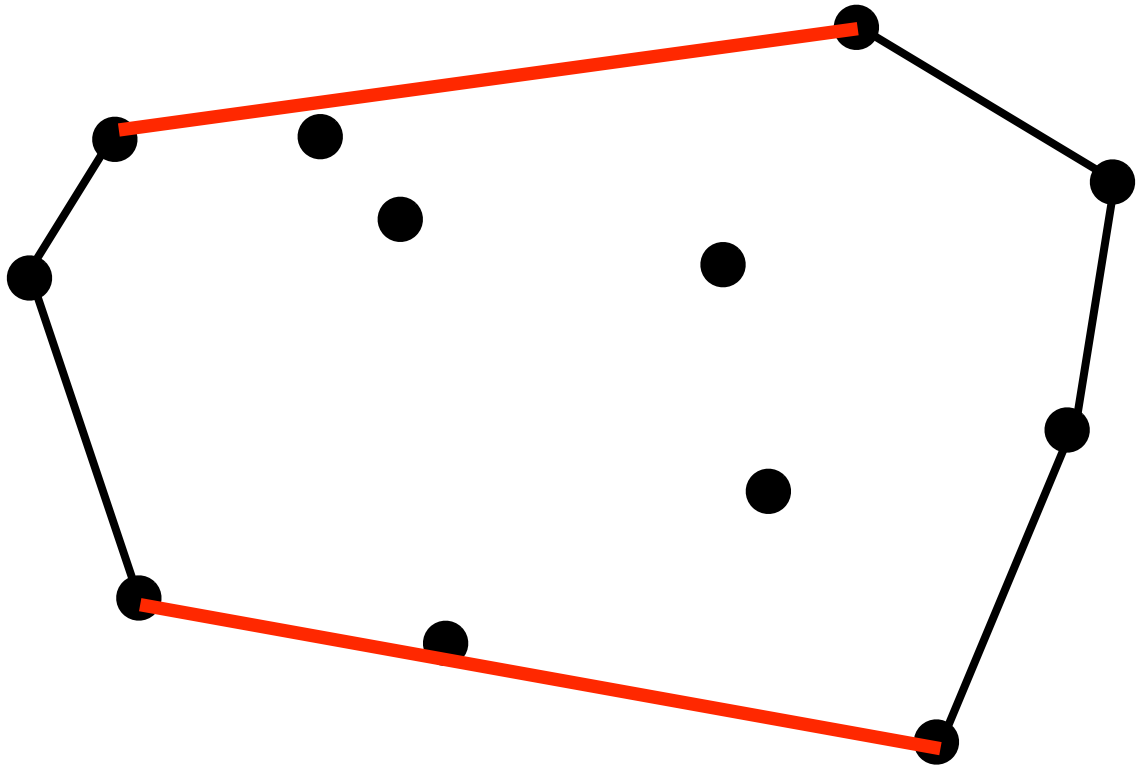
Algorithmische Geometrie

Konvexe Hülle - Conquer



Algorithmische Geometrie

Konvexe Hülle - Conquer



Algorithmische Geometrie
Konvexe Hülle - Algorithmus

var P: "nach x-Koordinate aufsteigend sortierte Folge von Punkten ($p_1,...,p_n$)";
k: integer;
if $|P| \leq 3$ then
"Berechne konvexe Hülle H von P auf direkte Weise"
else
$k := \lfloor |P|/2 \rfloor$;
$P_1 := (p_1,...,p_k)$; $P_2 := (p_{k+1},...,p_n)$;
"Wende Verfahren rekursiv auf P_1 und P_2 an und erhalte zwei konvexe Hüllen H_1 und H_2";
"Verbinde H_1 und H_2 und erhalte die gesuchte konvexe Hülle H von P".

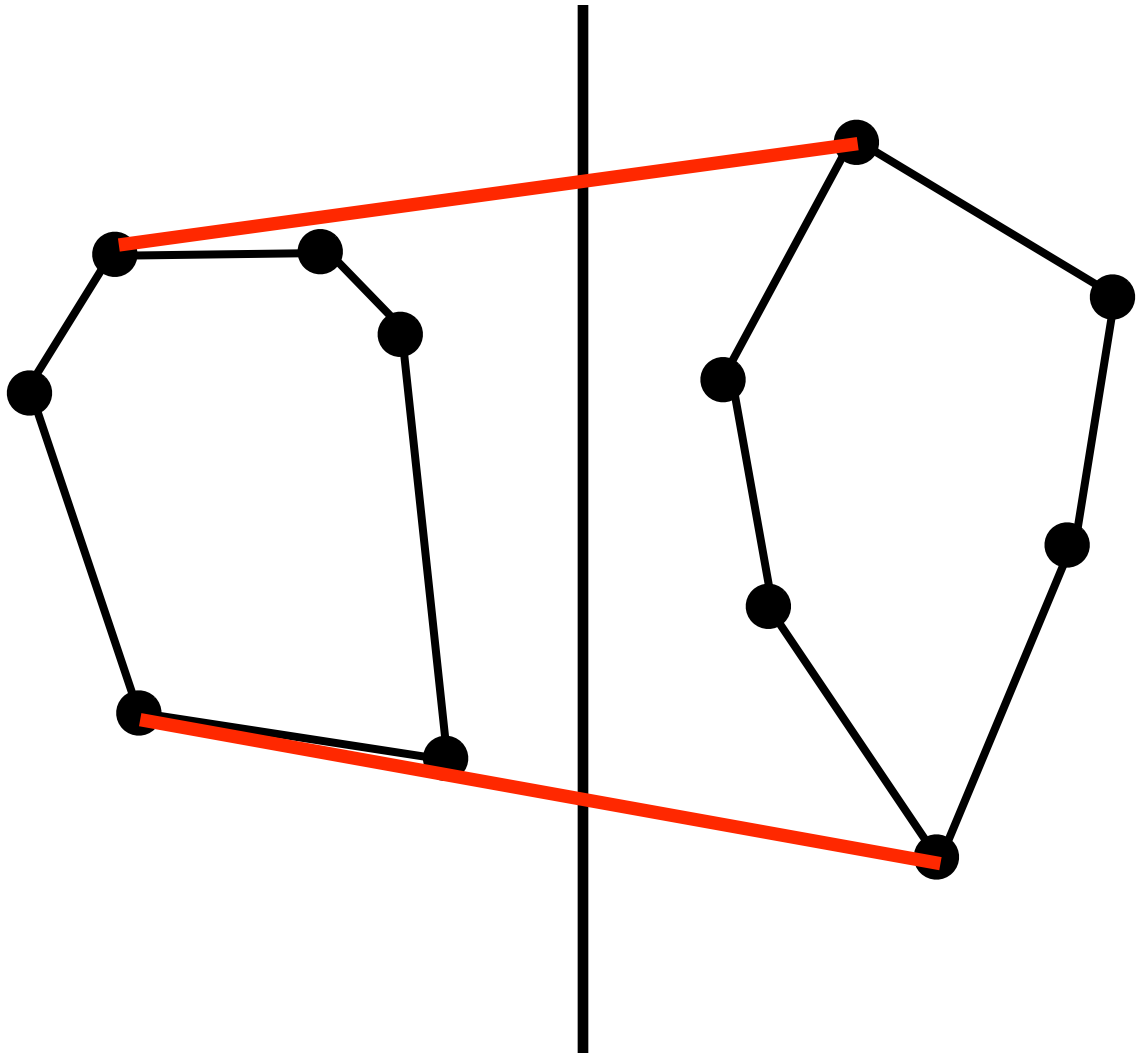
Algorithmische Geometrie
Konvexe Hülle - Algorithmus

var P: "nach x-Koordinate aufsteigend sortierte
Folge von Punkten ($p_1,...,p_n$)";
k: integer;
if $|P| \leq 3$ then
"Berechne konvexe Hülle H von P auf direkte Weise"
else
k:= $\lfloor |P|/2 \rfloor$;
$P_1 := (p_1,...,p_k)$; $P_2 := (p_{k+1},...,p_n)$;
"Wende Verfahren rekursiv auf
und erhalte zwei konvexe Hüll
"Verbinde H_1 und H_2 und erhalte die gesuchte konvexe Hülle H von P".

schwierigster Teil

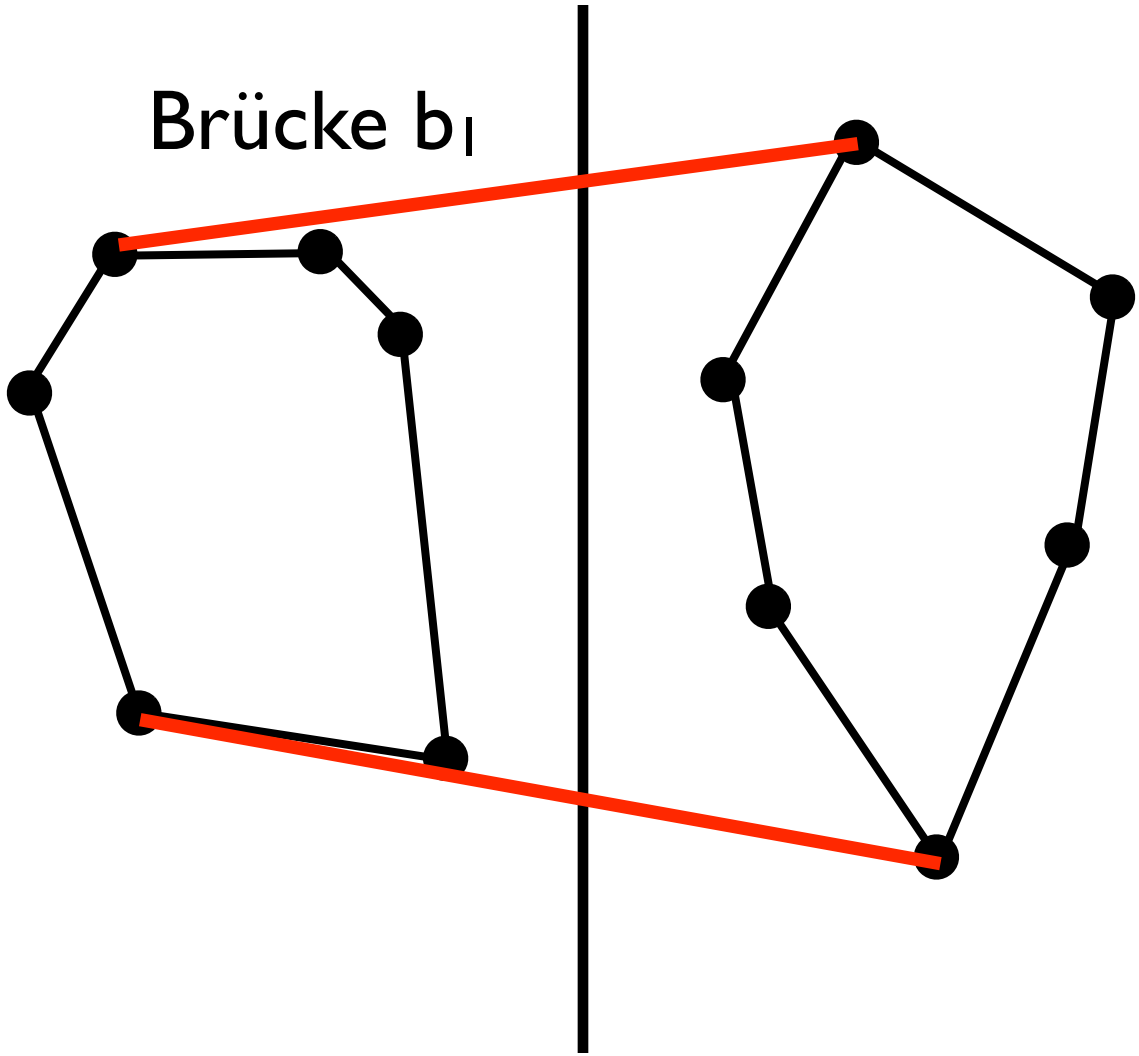
Algorithmische Geometrie

Konvexe Hülle - Conquer



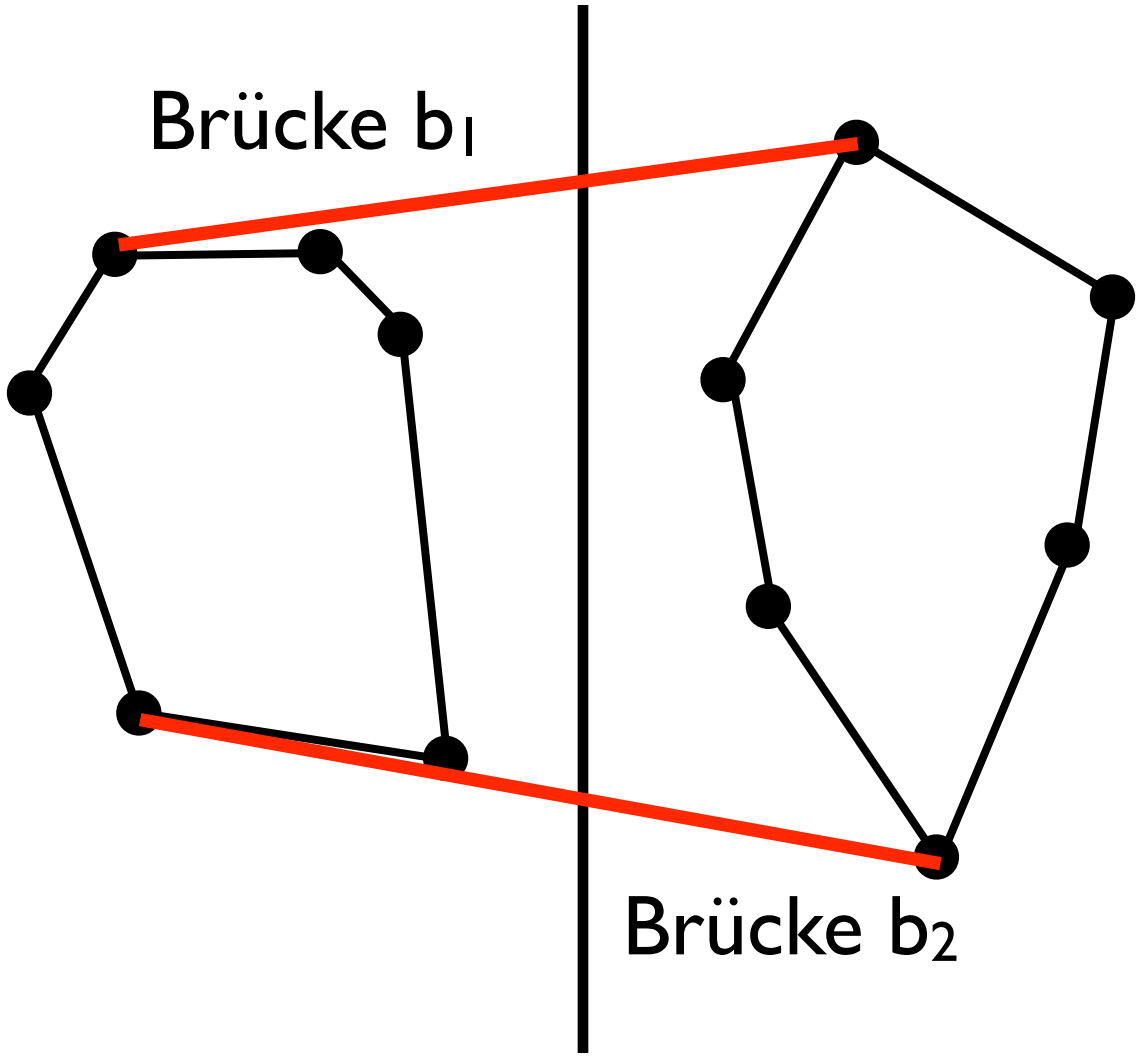
Algorithmische Geometrie

Konvexe Hülle - Conquer



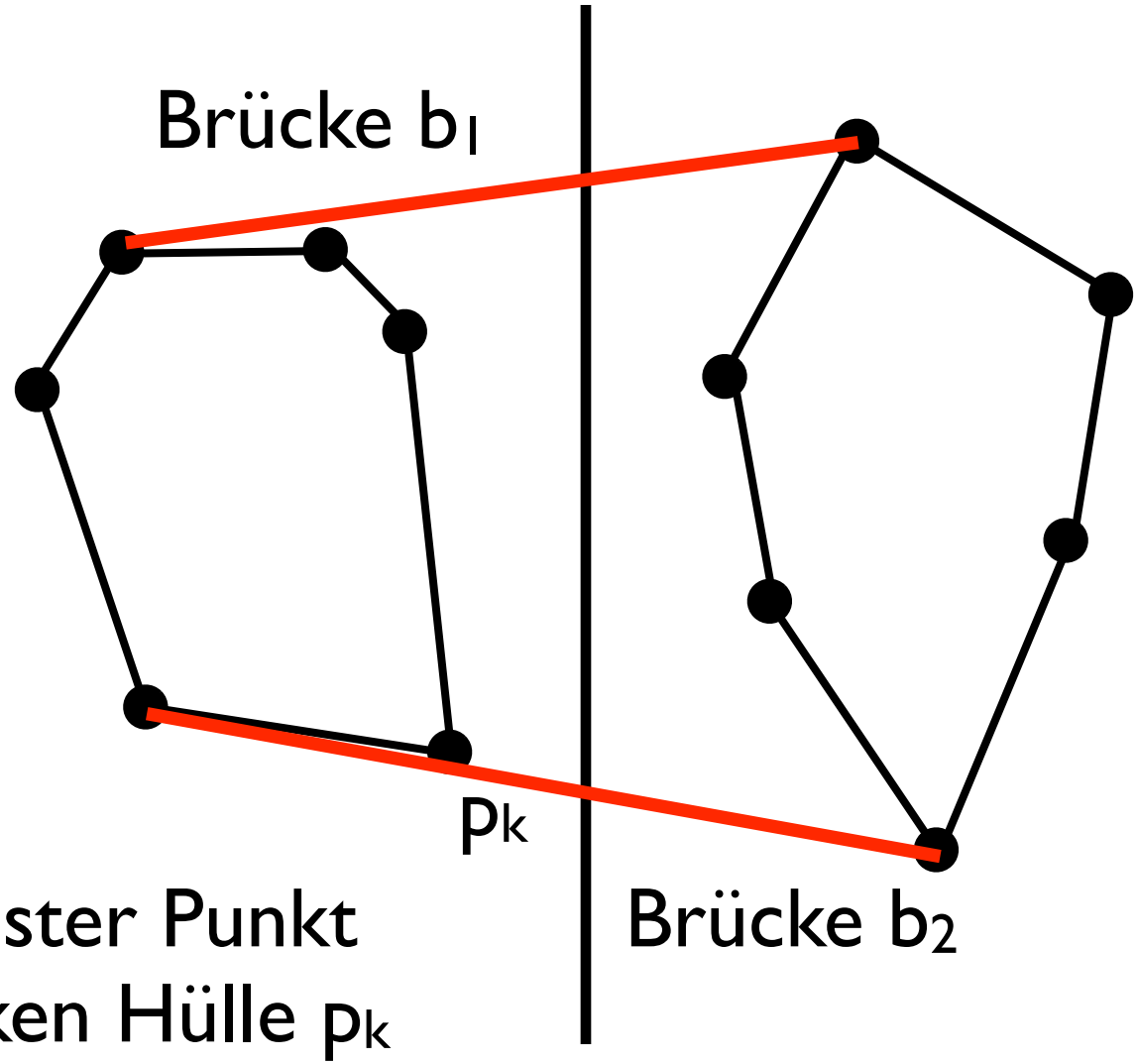
Algorithmische Geometrie

Konvexe Hülle - Conquer



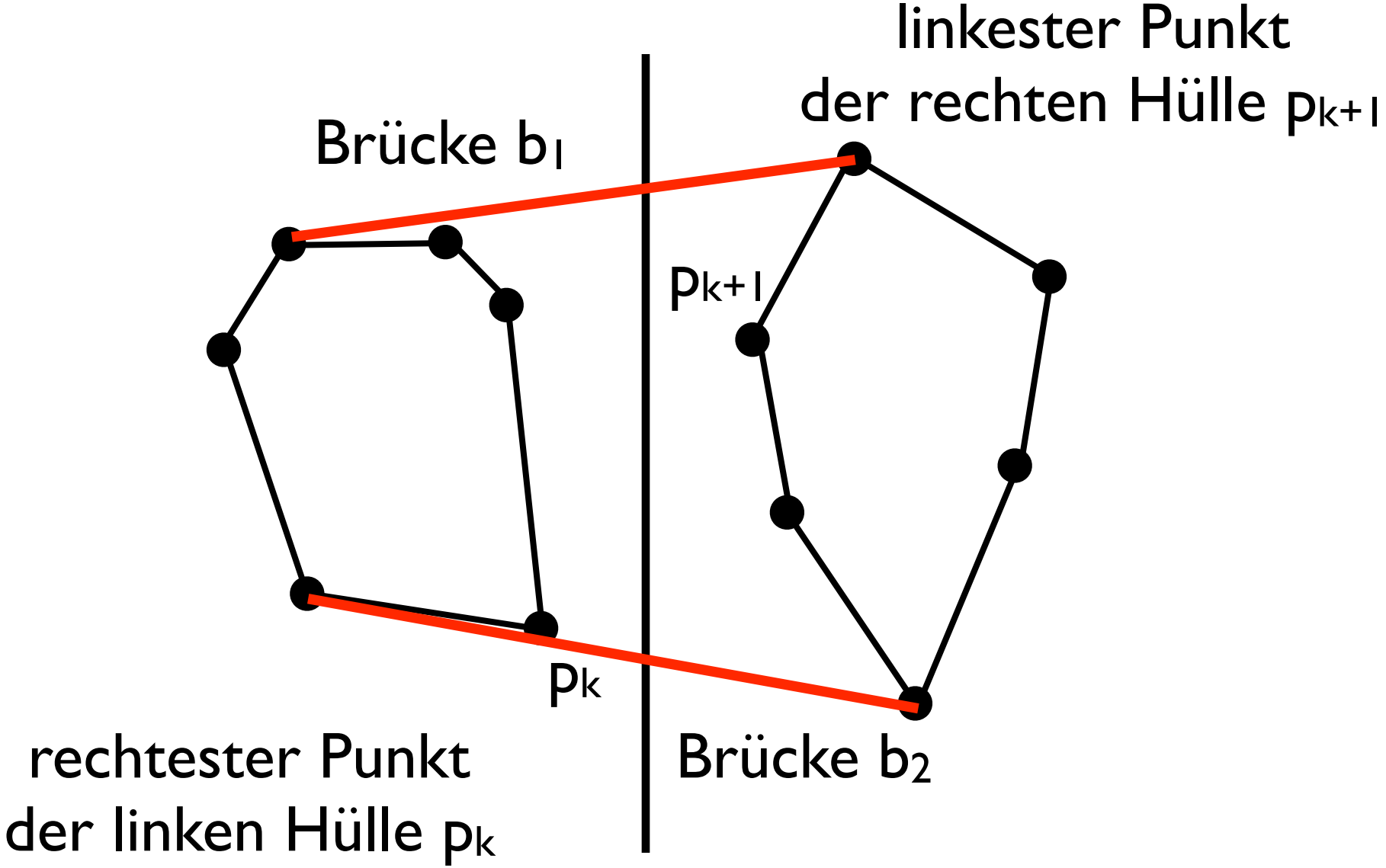
Algorithmische Geometrie

Konvexe Hülle - Conquer



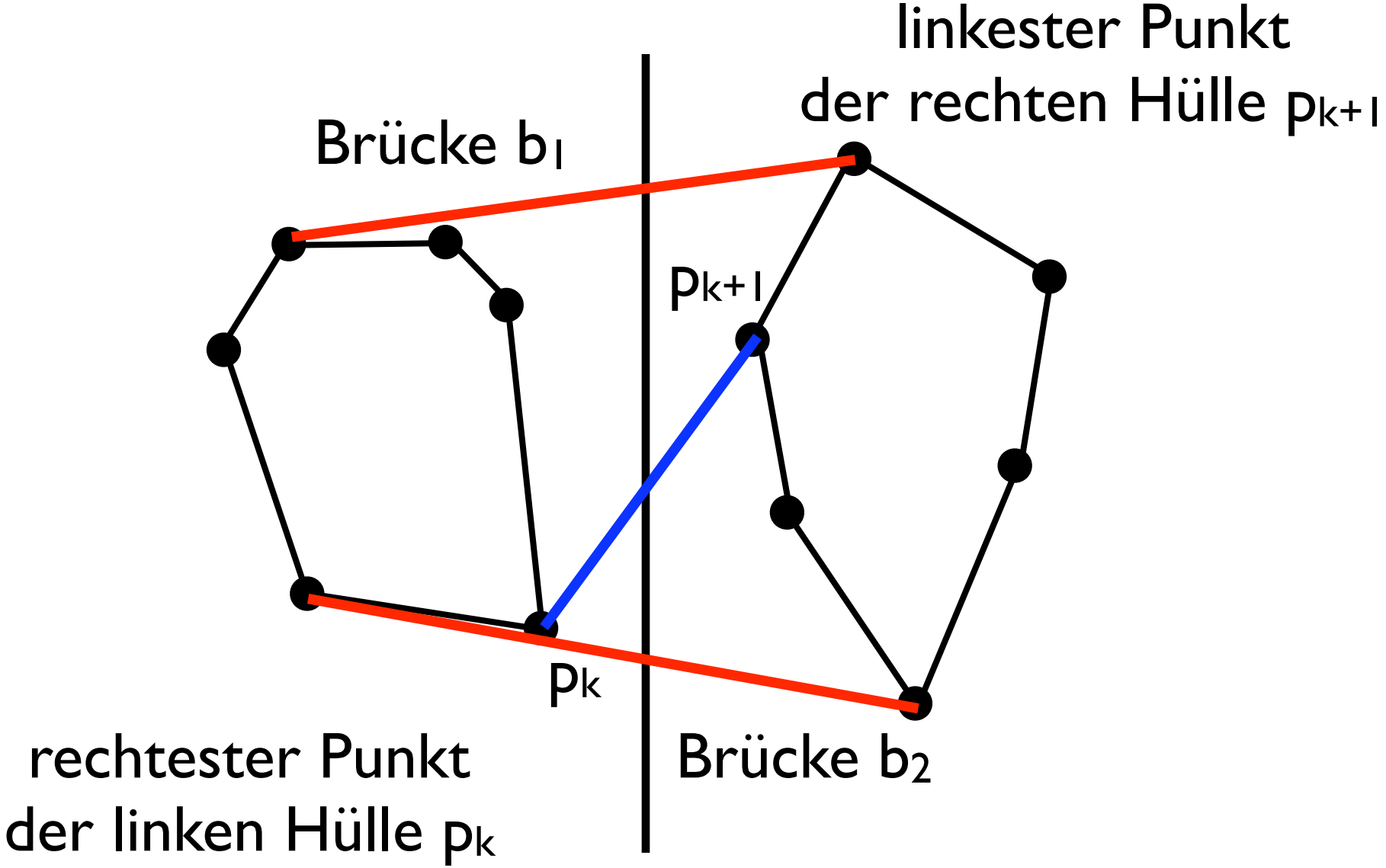
Algorithmische Geometrie

Konvexe Hülle - Conquer



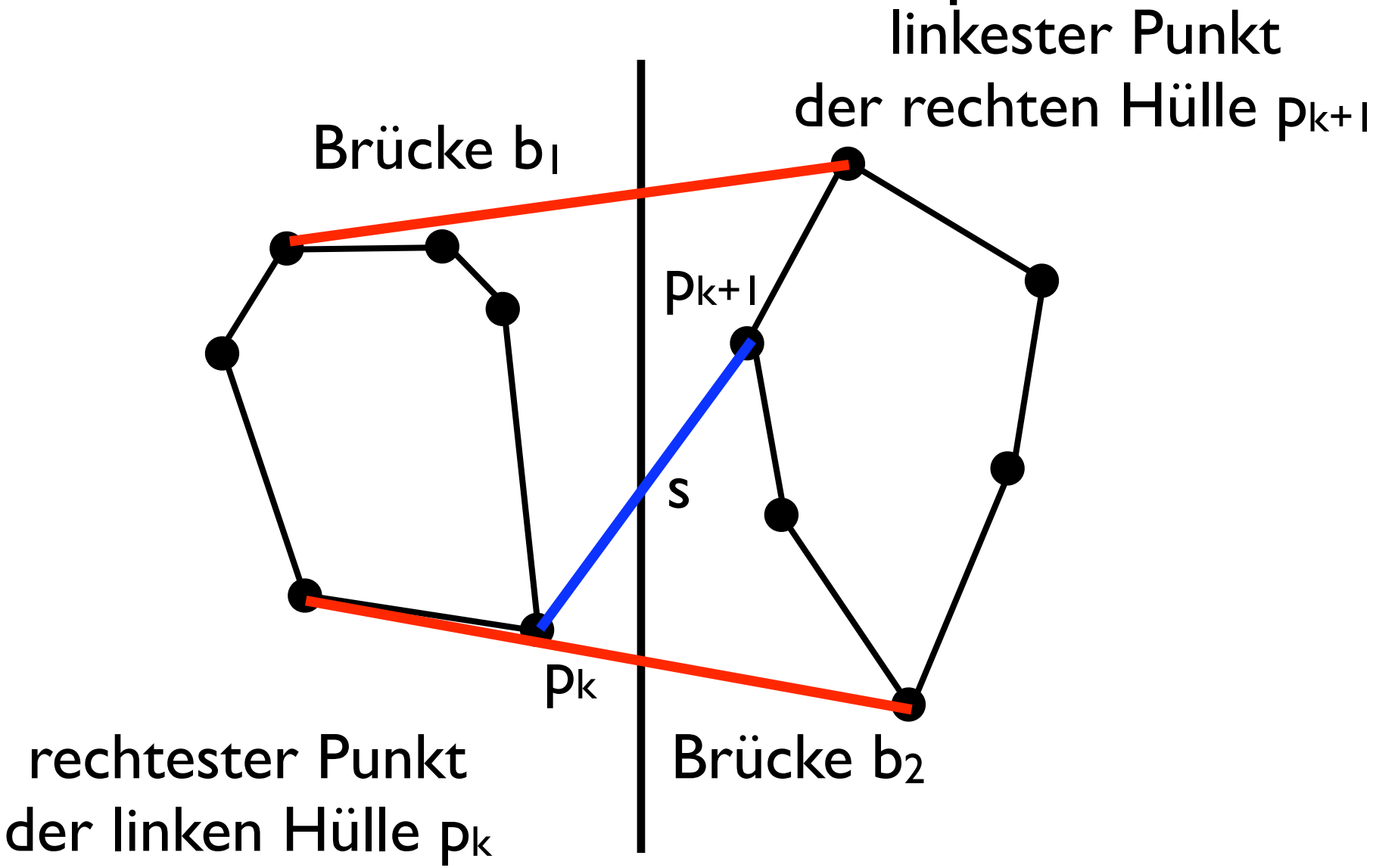
Algorithmische Geometrie

Konvexe Hülle - Conquer



Algorithmische Geometrie

Konvexe Hülle - Conquer



Algorithmische Geometrie

Konvexe Hülle - Bestimmung Brücke b_1

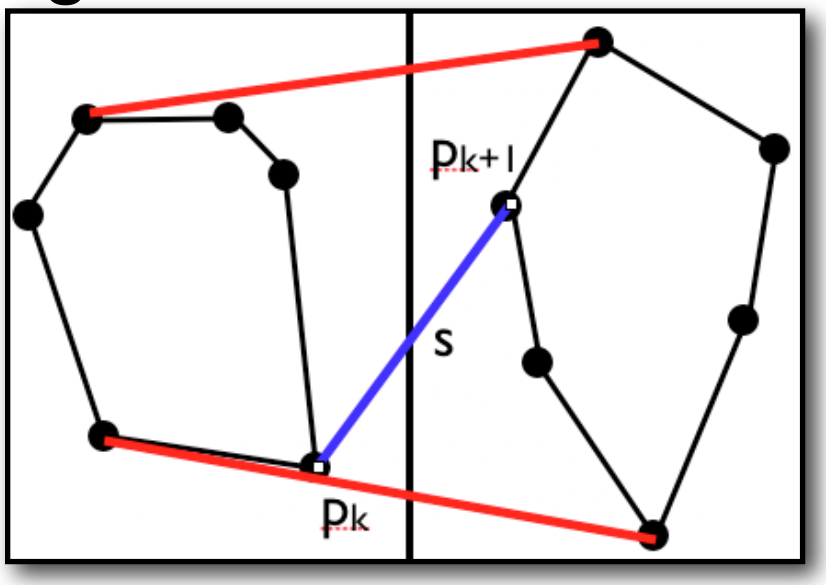
$u:=p_k$; $v:=p_{k+1}$; s:=Strecke uv;

while "der auf u *gegen* den Uhrzeigersinn in der linken Hülle folgende Punkt u' oder der auf v *im* Uhrzeigersinn in der rechten Hülle folgende Punkt v' liegen oberhalb von s" do

if "die while-Bedingung trifft auf u zu"

then u:=u'

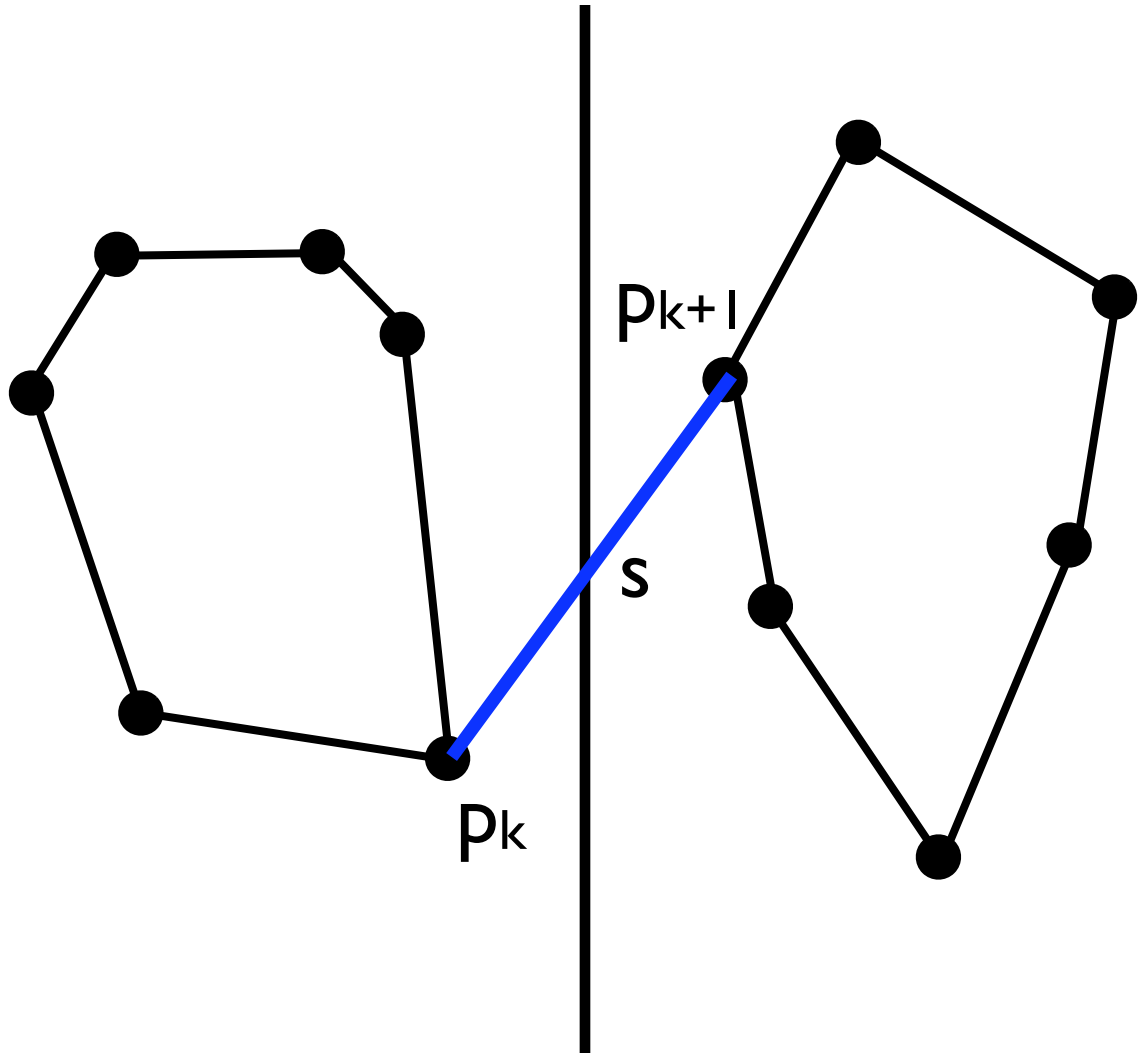
else v:=v'.



Analog für b_2

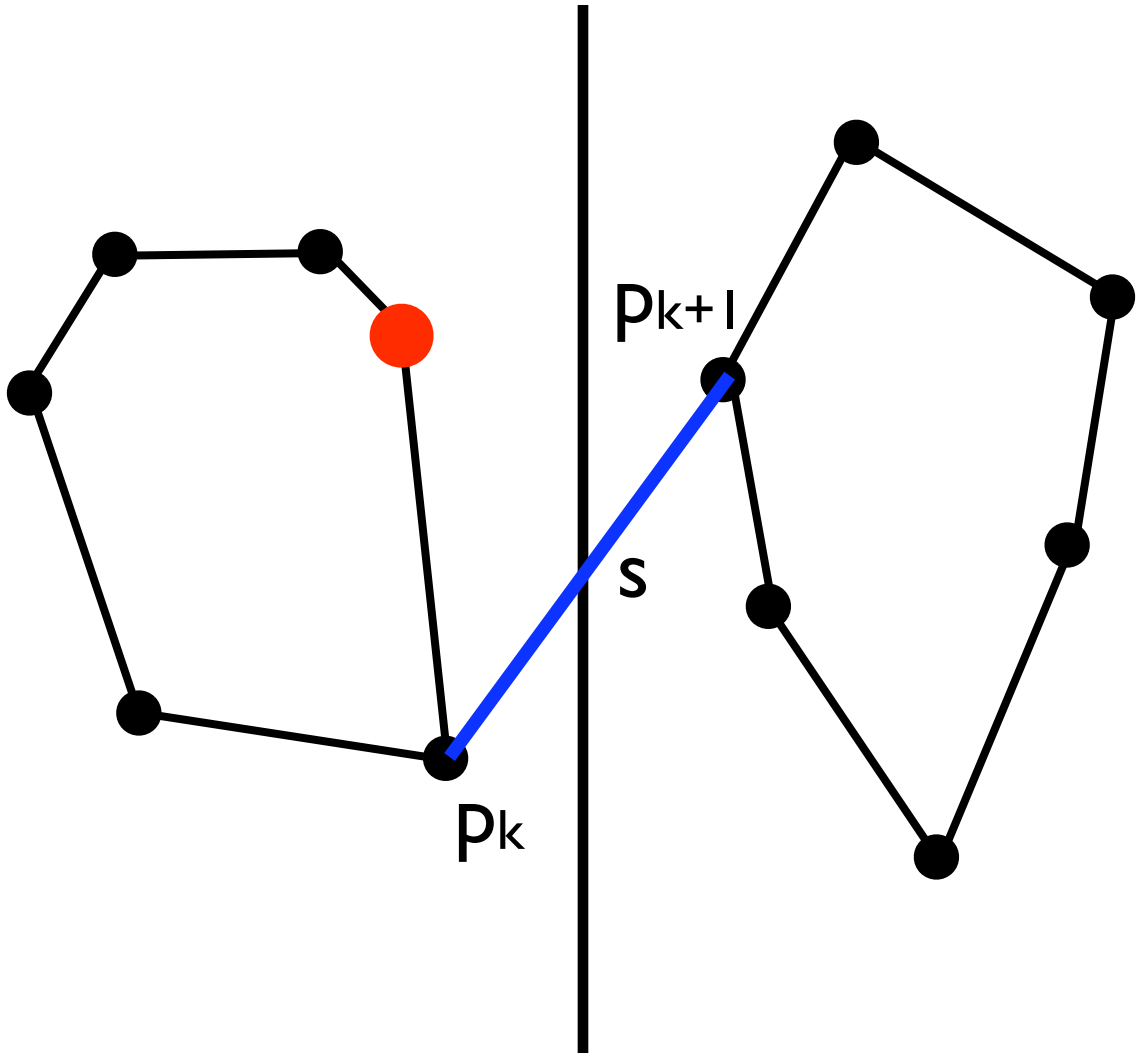
Algorithmische Geometrie

Konvexe Hülle - Conquer



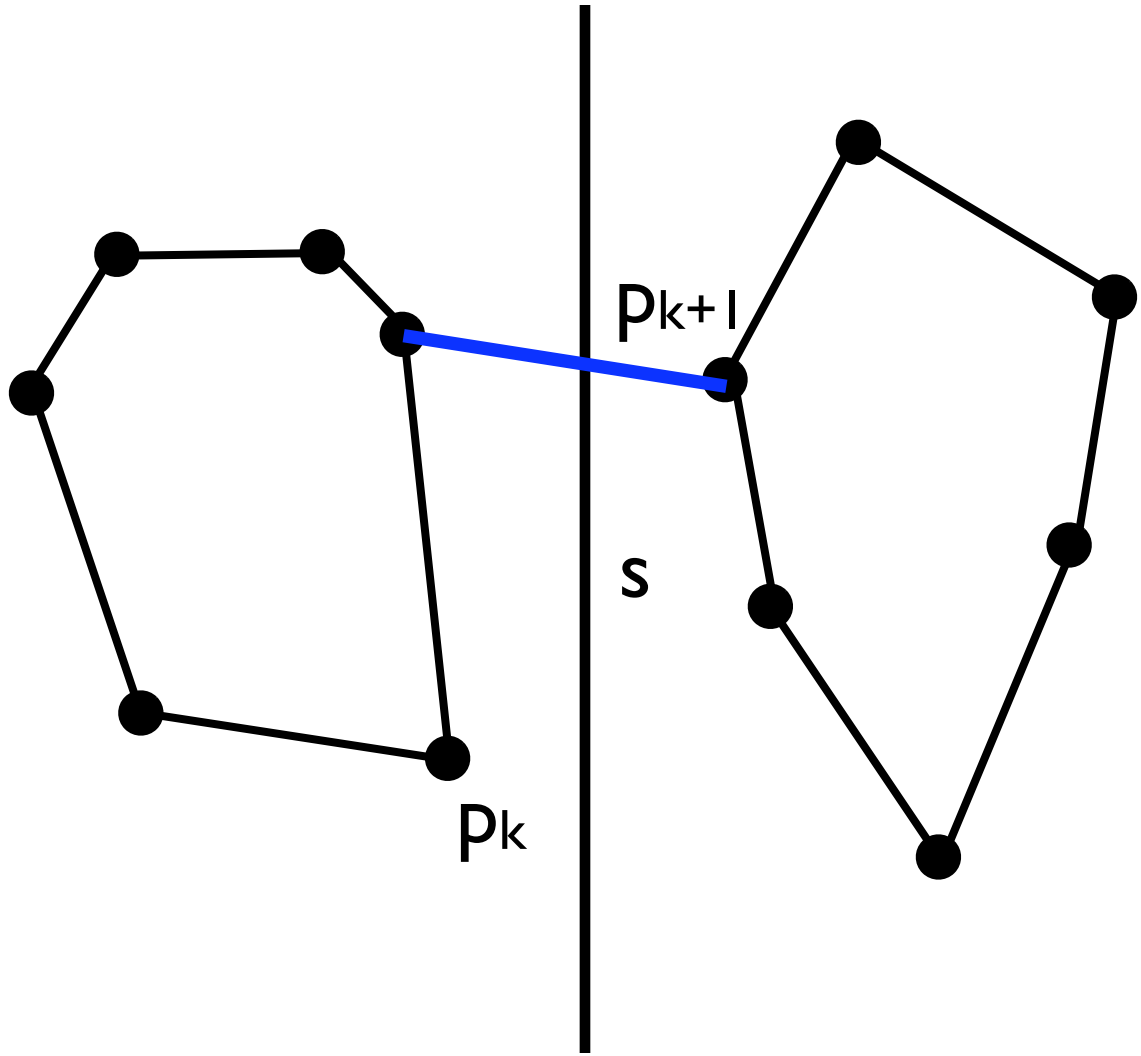
Algorithmische Geometrie

Konvexe Hülle - Conquer



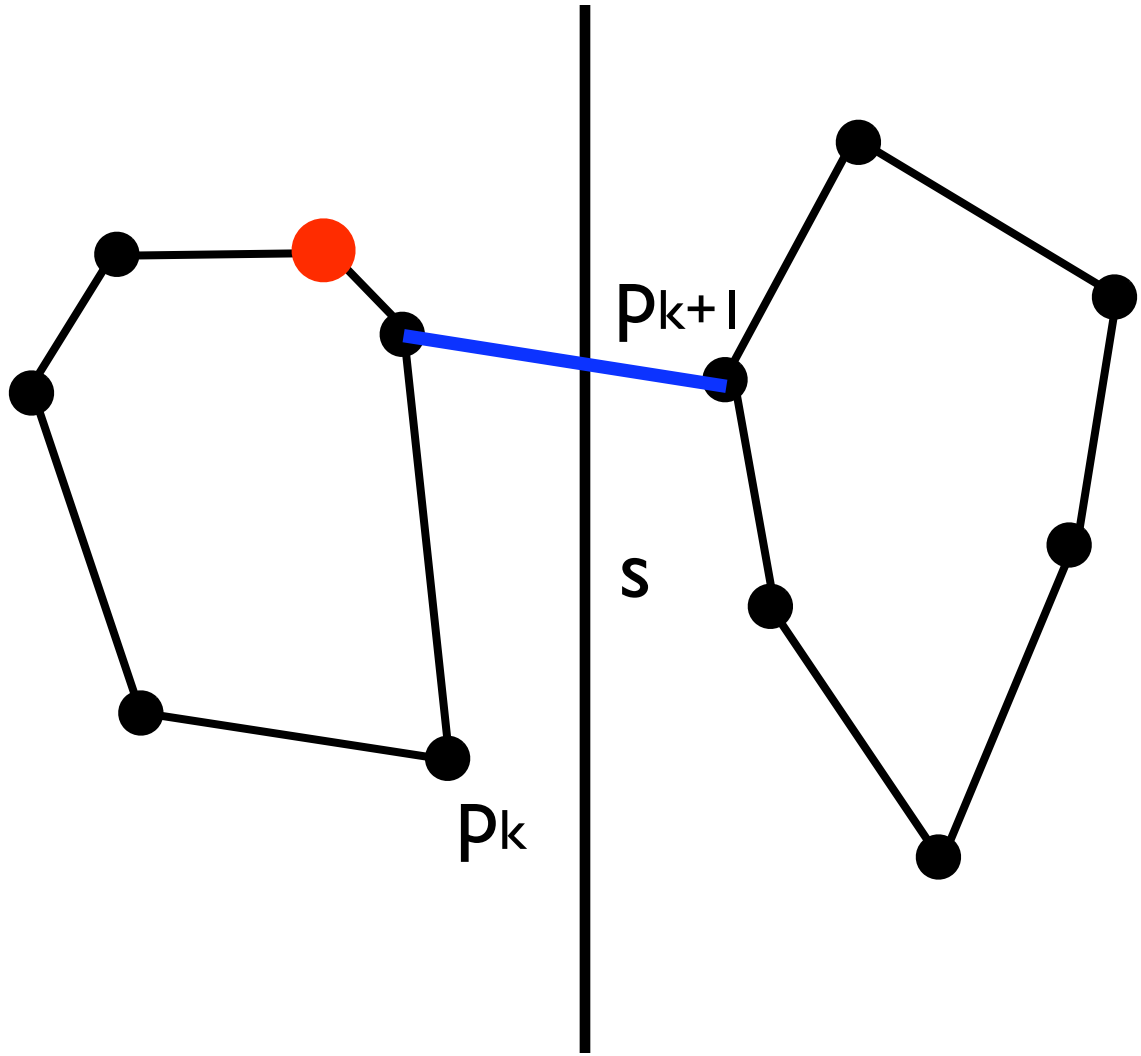
Algorithmische Geometrie

Konvexe Hülle - Conquer



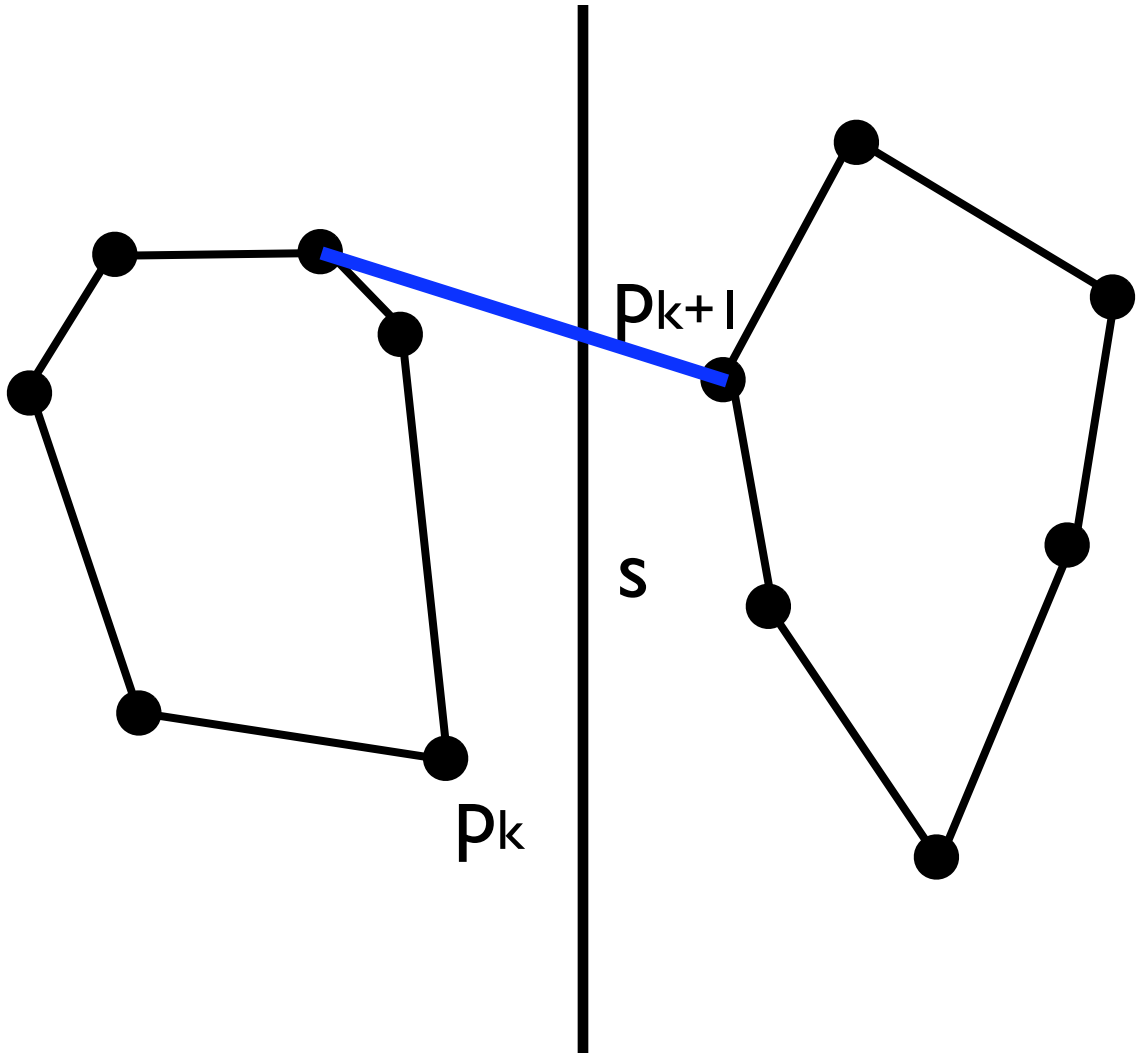
Algorithmische Geometrie

Konvexe Hülle - Conquer



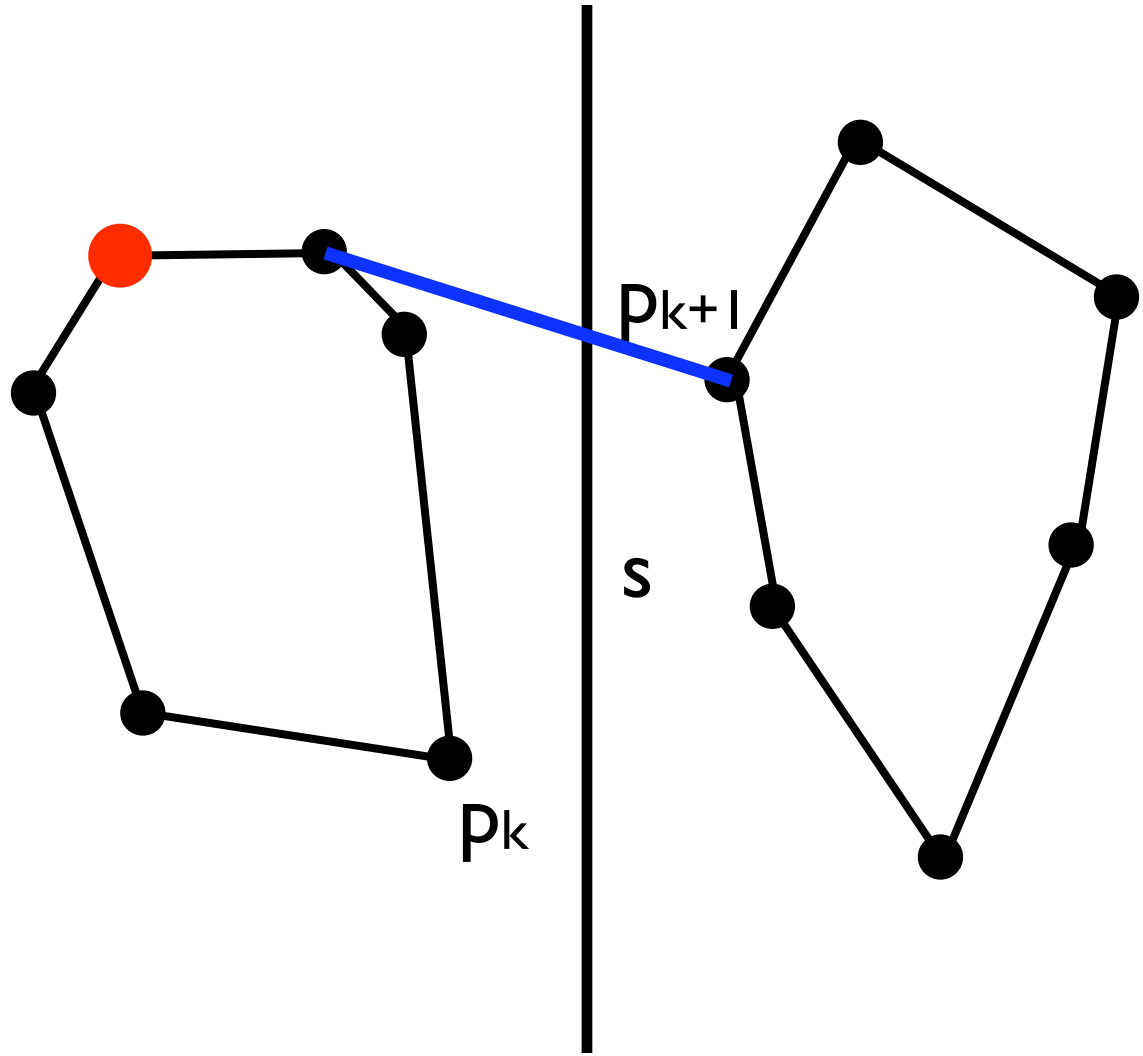
Algorithmische Geometrie

Konvexe Hülle - Conquer



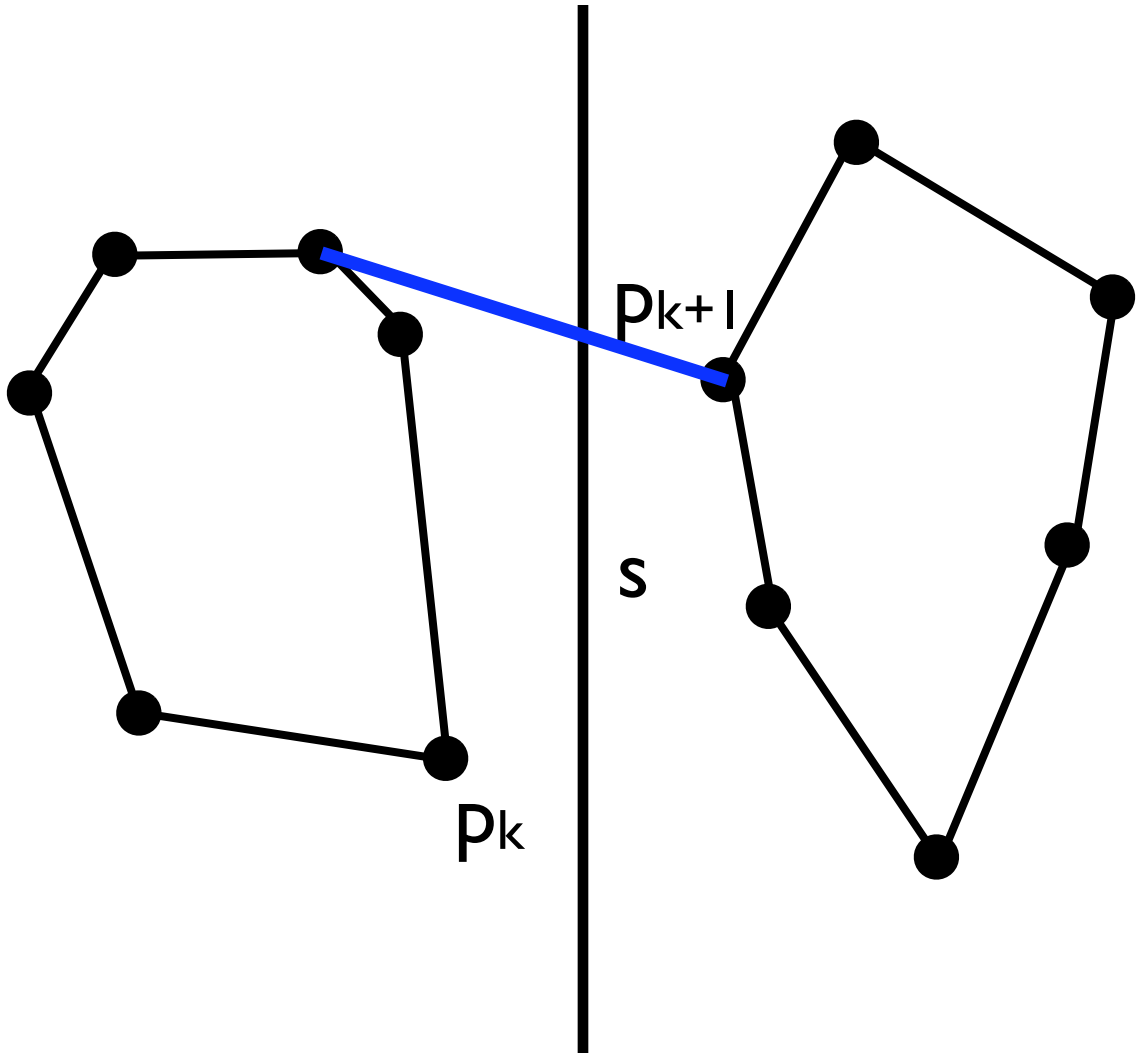
Algorithmische Geometrie

Konvexe Hülle - Conquer



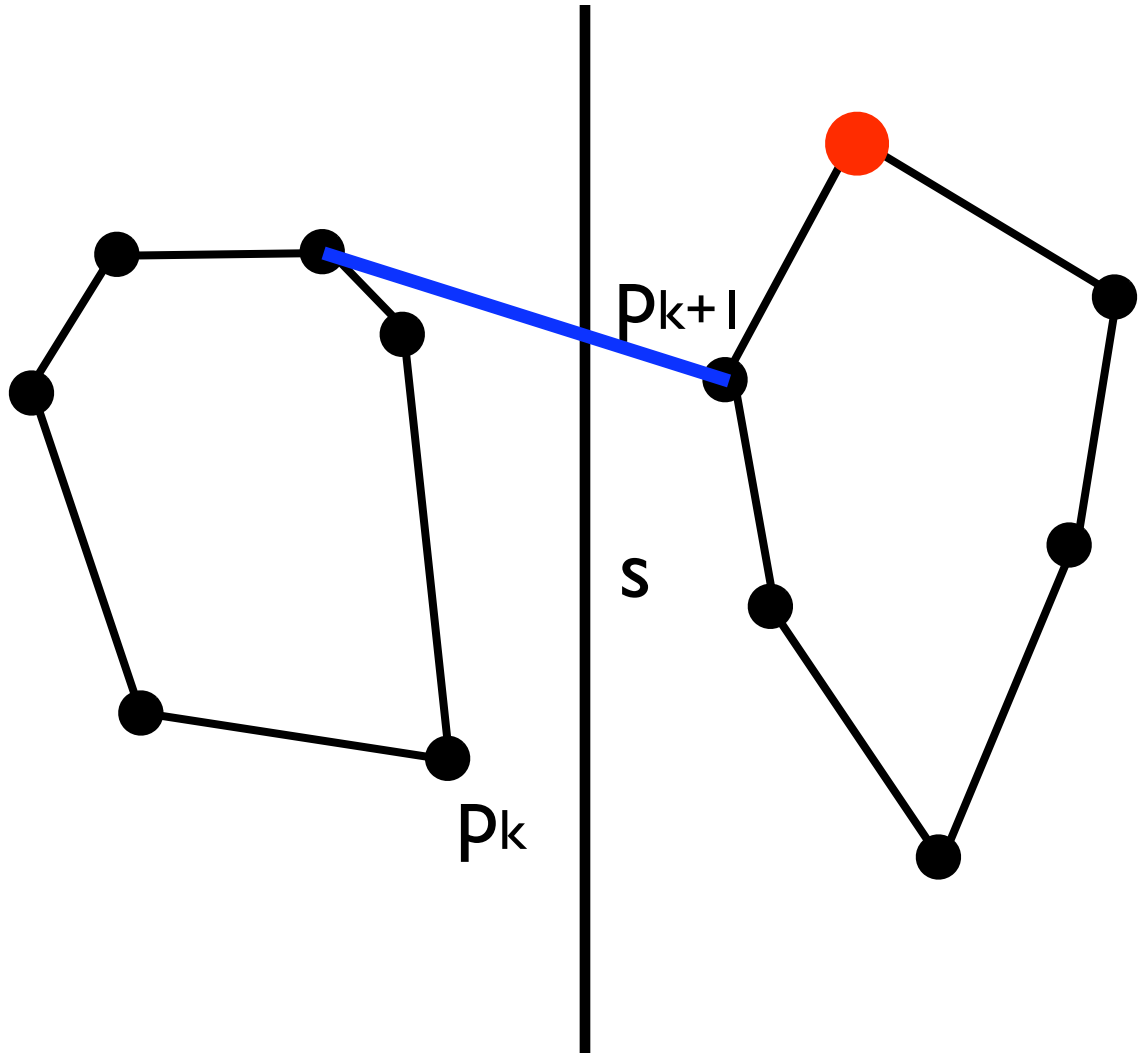
Algorithmische Geometrie

Konvexe Hülle - Conquer



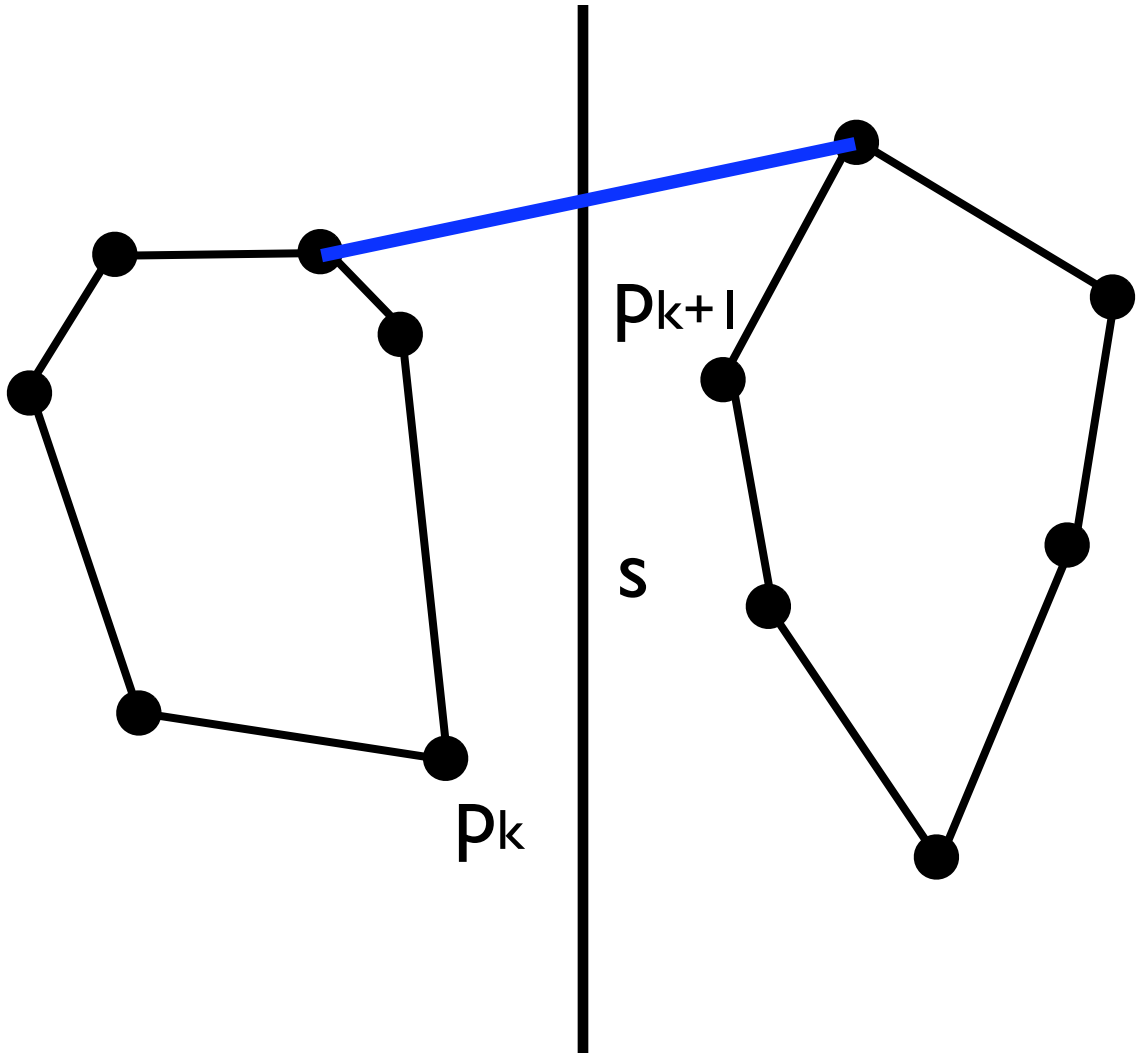
Algorithmische Geometrie

Konvexe Hülle - Conquer



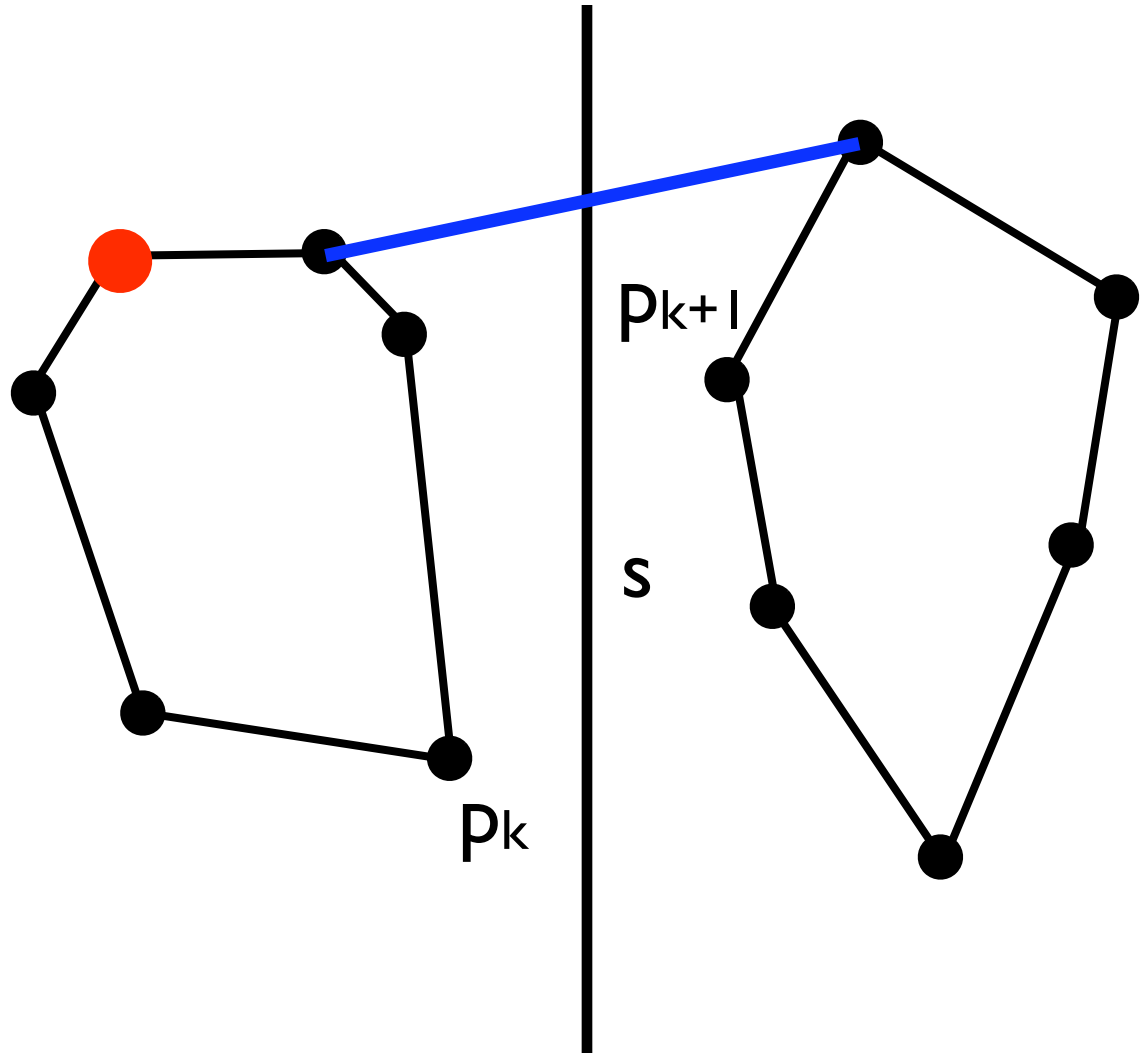
Algorithmische Geometrie

Konvexe Hülle - Conquer



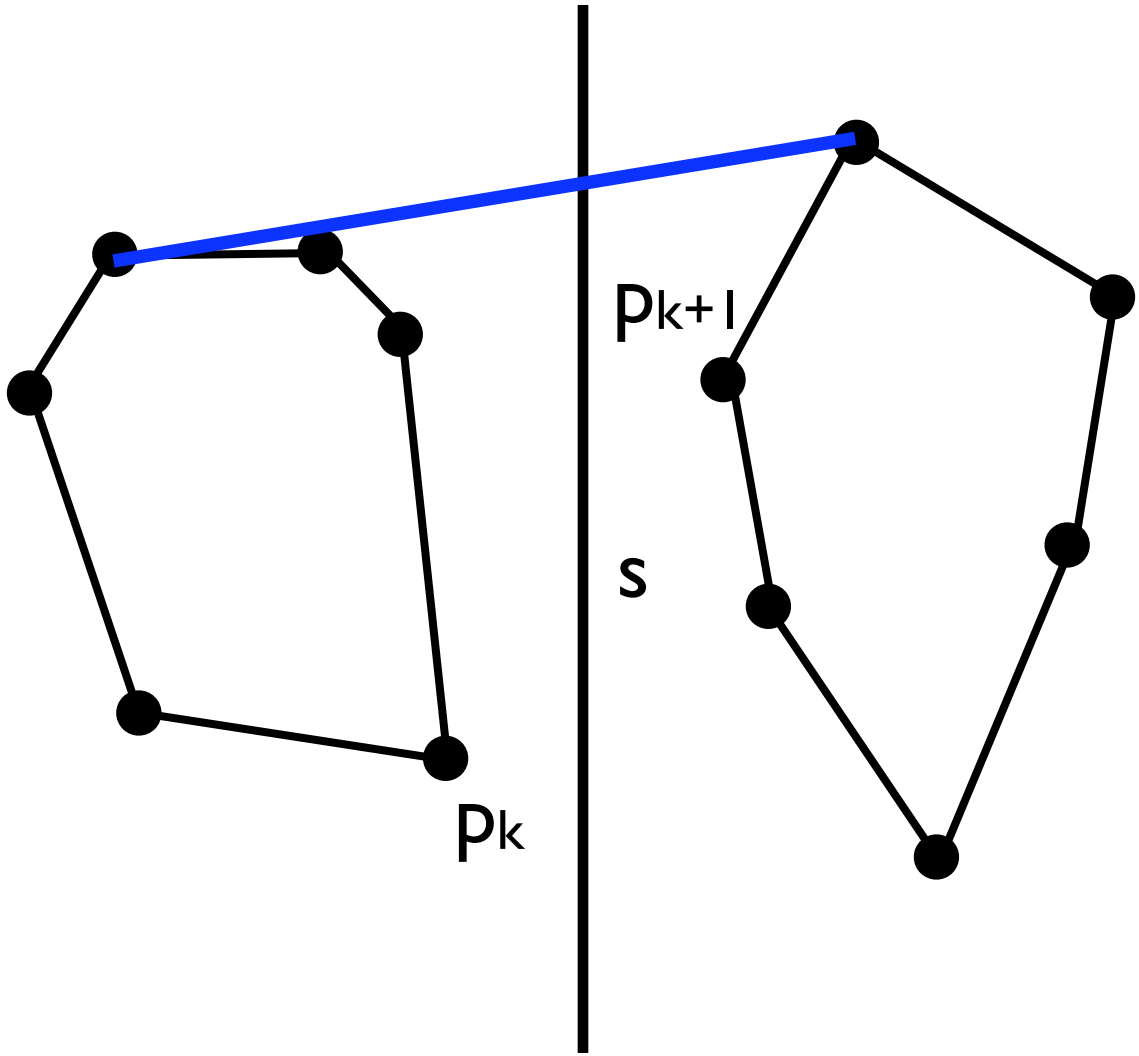
Algorithmische Geometrie

Konvexe Hülle - Conquer



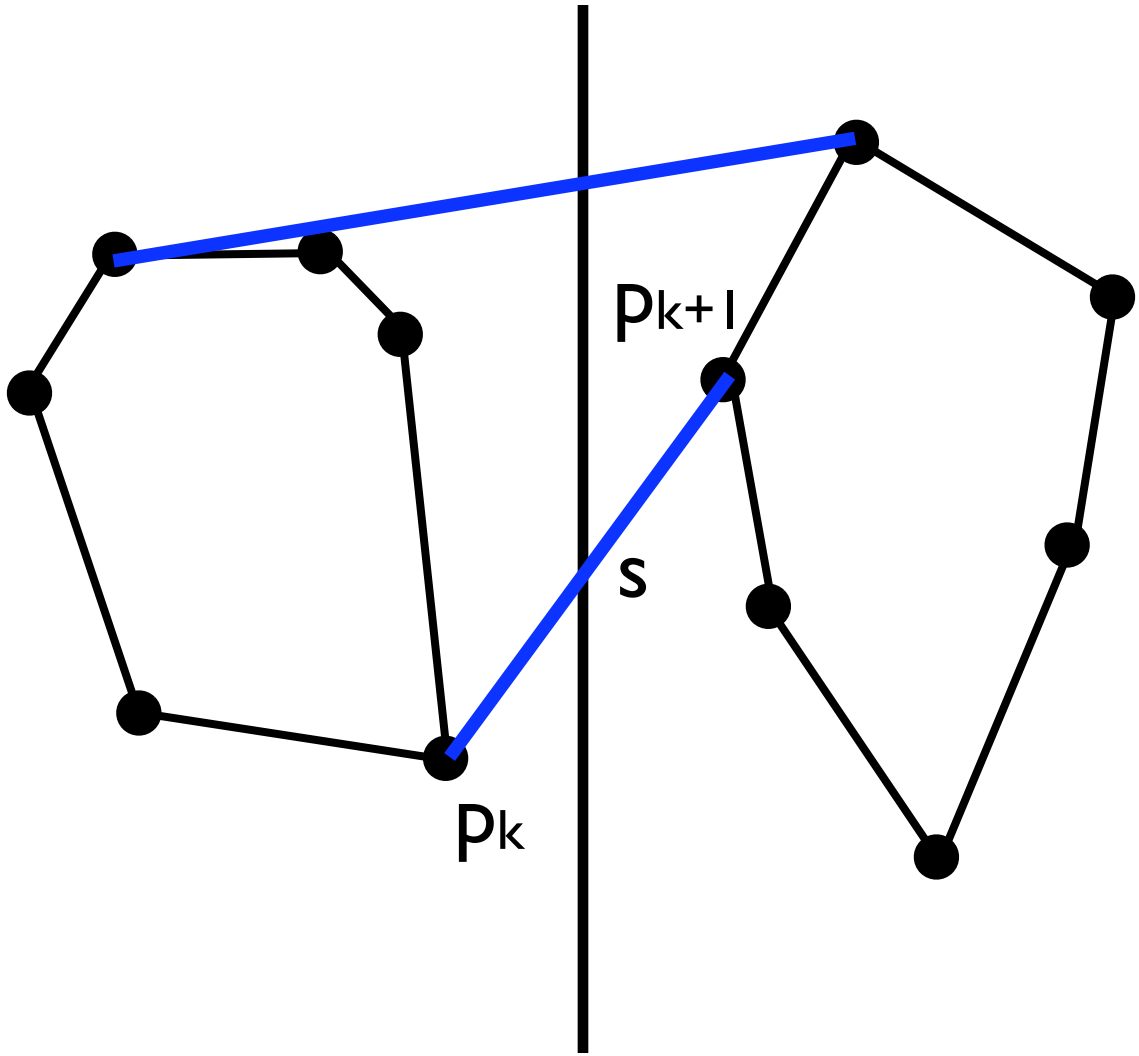
Algorithmische Geometrie

Konvexe Hülle - Conquer



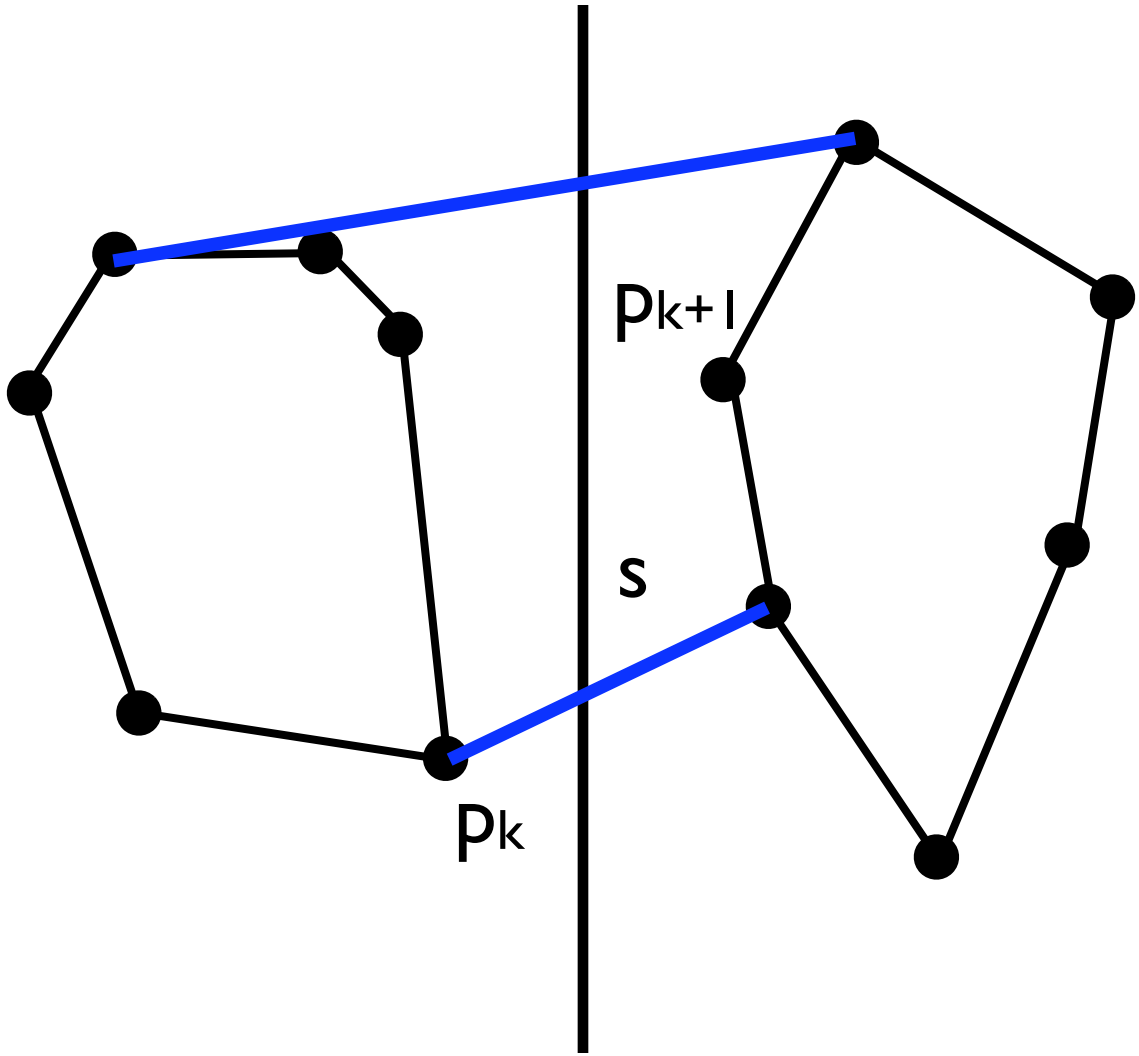
Algorithmische Geometrie

Konvexe Hülle - Conquer



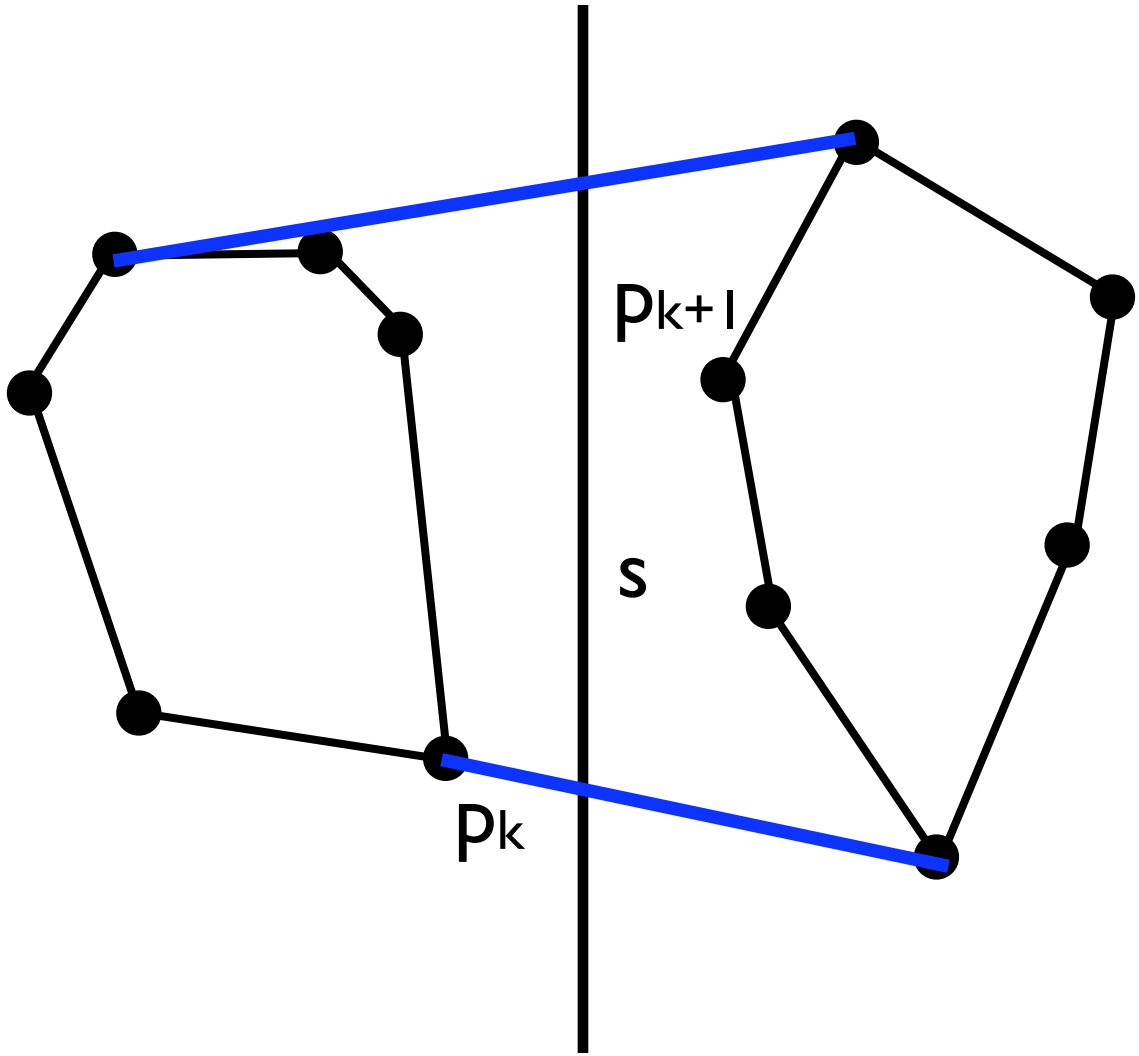
Algorithmische Geometrie

Konvexe Hülle - Conquer



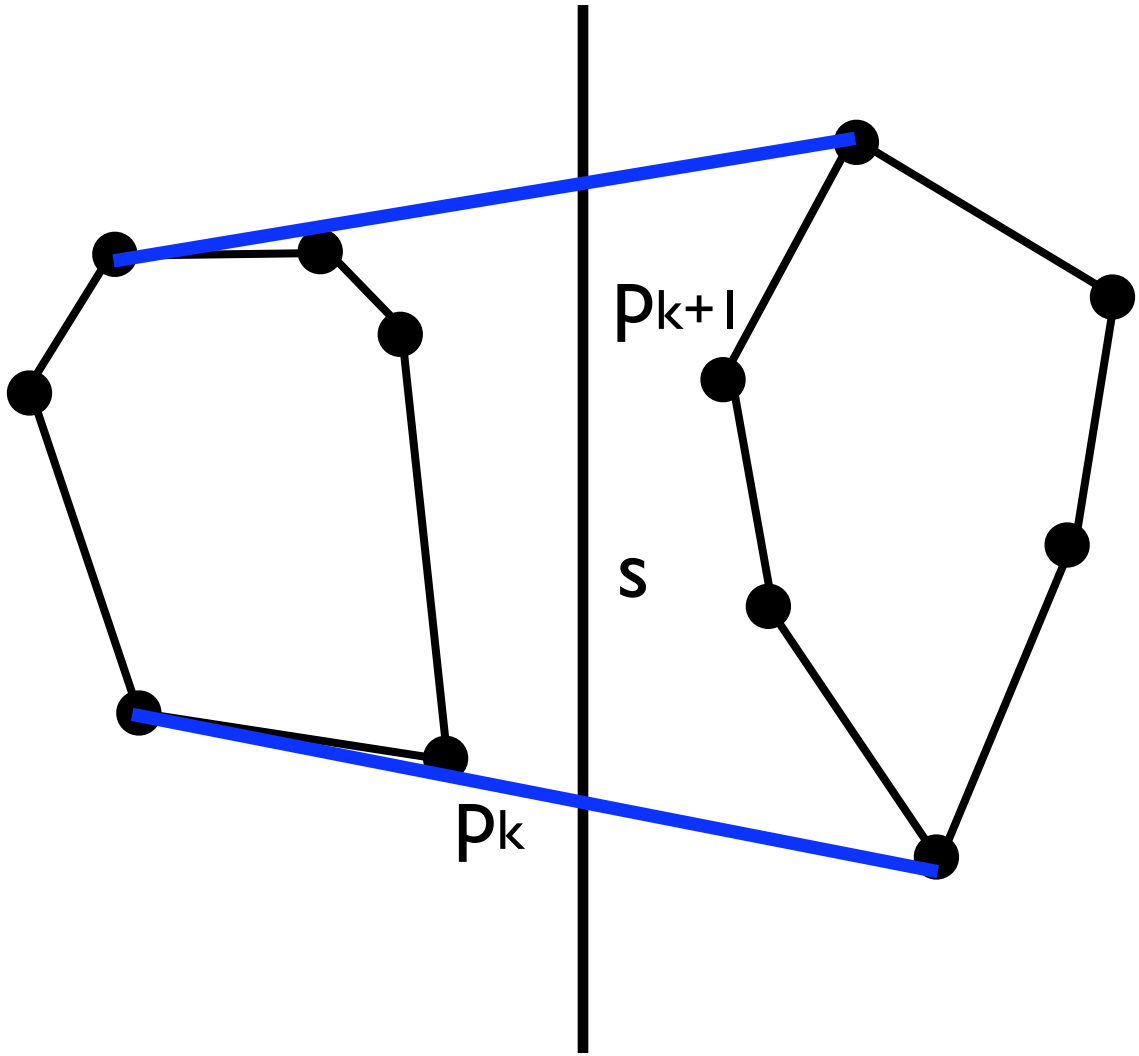
Algorithmische Geometrie

Konvexe Hülle - Conquer



Algorithmische Geometrie

Konvexe Hülle - Conquer



Algorithmische Geometrie
Konvexe Hülle - Laufzeit

- Man zeigt für die Laufzeit T leicht:

$$T(n)=2T(n/2)+O(n)$$

$$T(3)=1$$

- Lösung: $T(n)=O(n \log n)$

Algorithmische Geometrie
Konvexe Hülle - untere Schranke

- Geht es noch effizienter als O(n log n)? NEIN!
- Zeige:
  - Ein Algorithmus zur Bestimmung der konvexen Hülle ist zugleich ein Algorithmus zum Sortieren.
  - Könnte man die konvexe Hülle für n Punkte also schneller bestimmen als O(n log n), so könnte man schneller sortieren als O(n log n). Widerspruch!!
- Man reduziert das Sortierproblem auf das Problem der konvexen Hülle (**Reduktion**).

Algorithmische Geometrie
Konvexe Hülle - untere Schranke

Satz

Ein Algorithmus zur Bestimmung der konvexen Hülle von n Punkten in der Ebene benötigt mindestens $O(n \log n)$ Vergleiche.

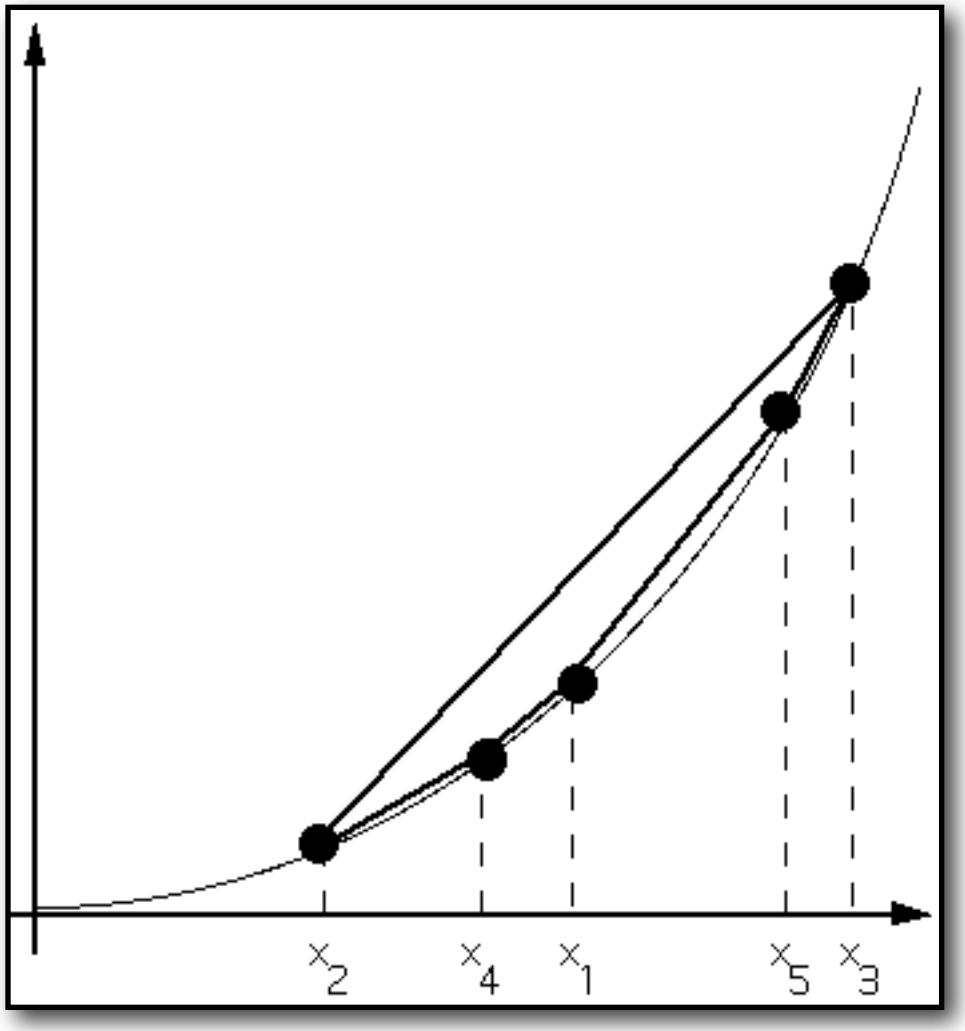
Algorithmische Geometrie
Konvexe Hülle - untere Schranke

Beweisskizze

Wähle beliebige zu sortierende Zahlen $x_1,...,x_n \in IR$.

Betrachte die Punkte $(x_1,x_1^2),...,(x_n,x_n^2)$ und bestimme ihre konvexe Hülle.

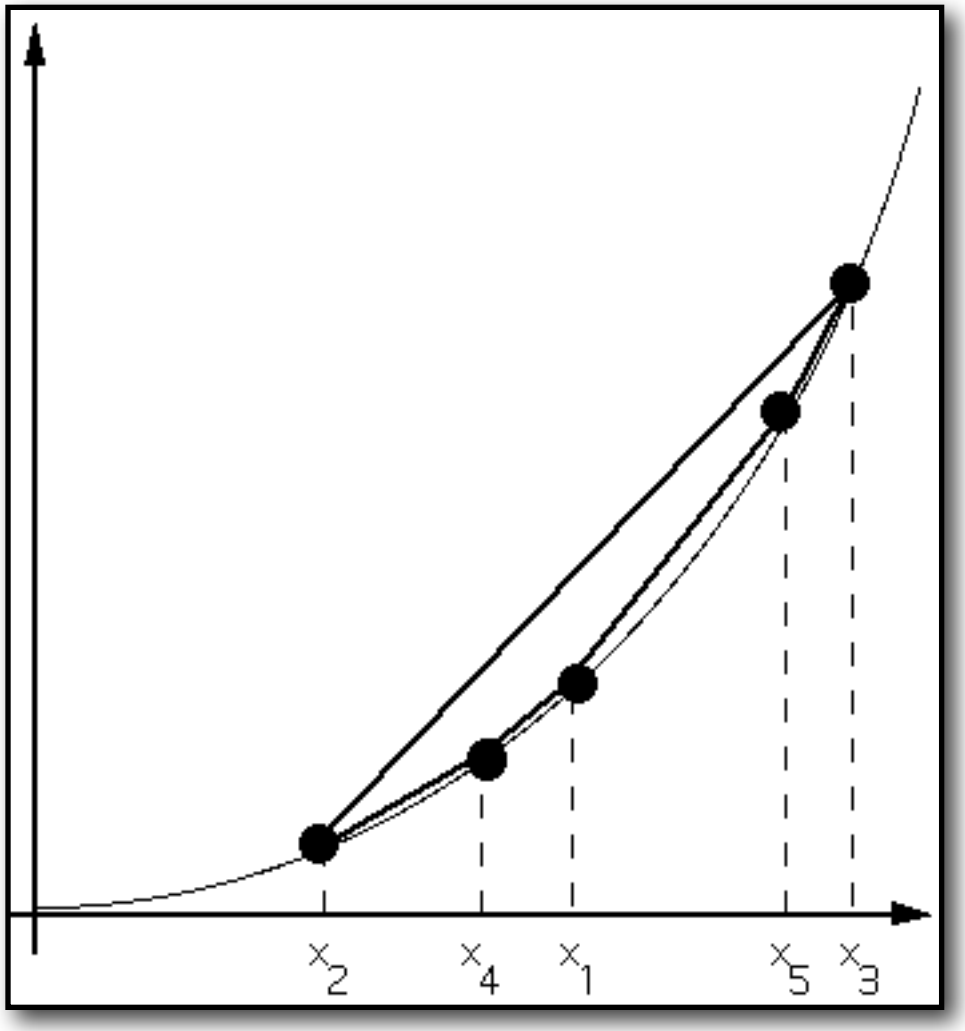
Offenbar gehören alle Punkte (x_i,x_i^2) zur konvexen Hülle.



Algorithmische Geometrie
Konvexe Hülle - untere Schranke

Beweisskizze (Forts.)

Die Ausgabe des Algorithmus aller Punkte der konvexen Hülle im Uhrzeigersinn ist zugleich eine Sortierung der Zahlen $x_1,...,x_n$, wenn man die zweite Koordinate streicht.



Algorithmische Geometrie
Konvexe Hülle - untere Schranke

Beweisskizze (Forts.)

Ein Algorithmus zur Bestimmung der konvexen Hülle von n Punkten ist also zugleich ein Algorithmus zum Sortieren von n reellen Zahlen.

Folglich ist die untere Schranke für Sortieren zugleich untere Schranke für das Problem der konvexen Hülle.