Algorithmische Graphentheorie

Teil 2

Zurück zum Problem der dichtesten Punkte

Im eindimensionalen könnte man die n Punkte in der Eingabe-Liste sortieren $O(n \cdot \ell ogn)$ und dann mit dem *scan line* Verfahren (auch *sweep* Methode) in $O(n)$ den kleinsten Abstand finden. Insgesamt also $O(n \cdot \ell ogn)$.

Geht es besser?

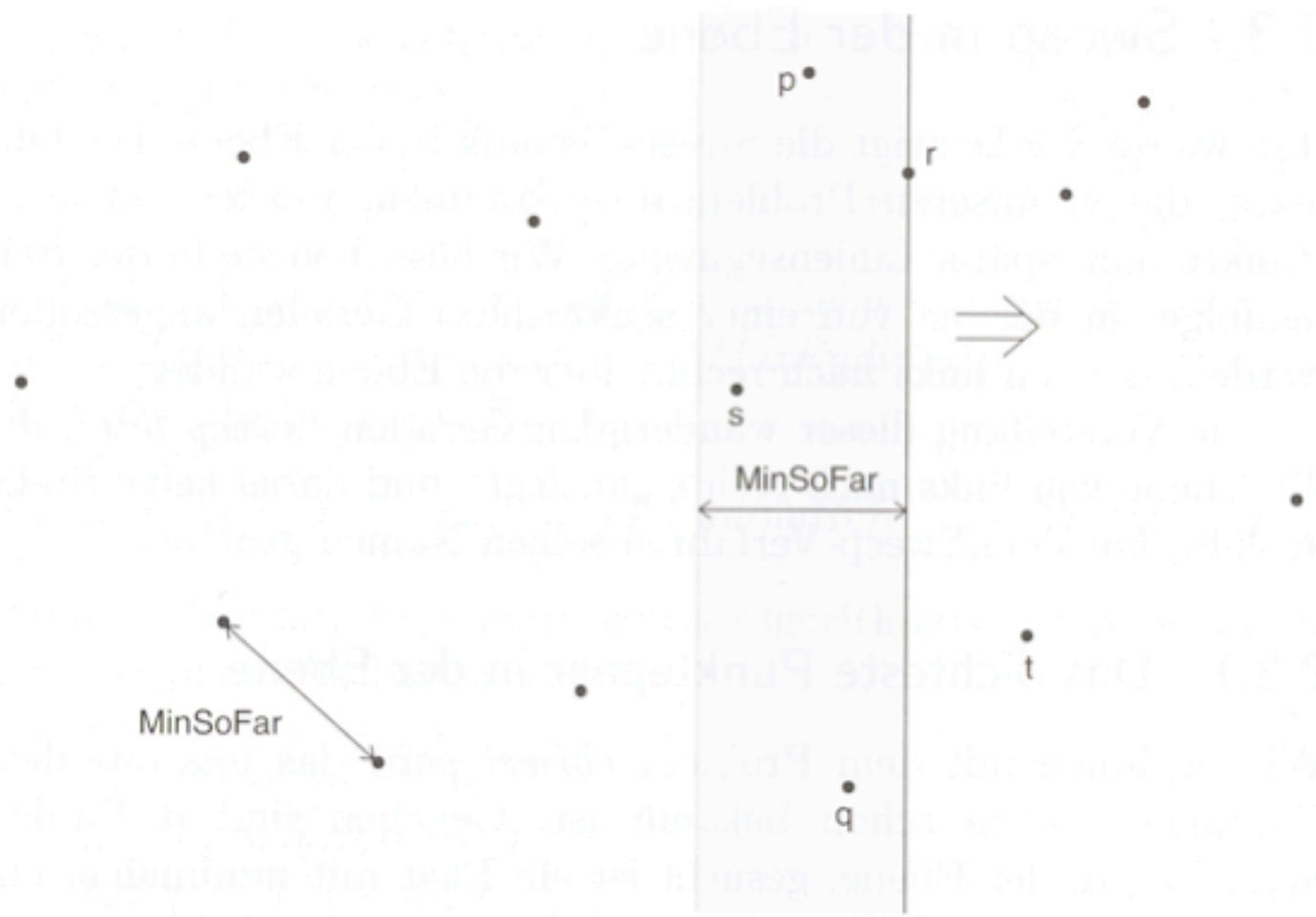
Antwort: Nein!

(Jeder Algorithmus, der den Abstand zweier Punkte aus einer Menge $P \subset \mathbb{N}$ mit $|P| = n$ in $f(n)$ bestimmt, kann dazu verwendet werden, um P zu sortieren, und das geht nicht schneller als in $O(n \cdot \ell ogn)$. (Vergleiche Johnsonbaugh u. Klein).

Sweep Methode

Dichteste Punkte im $\mathbb{N}^2$ können mit diversen Methoden gefunden werden:

Die *sweep* Methode findet man in [Klein] beschrieben. Ein Verfahren mit *devide and conquer* ist in [Johnsonbaugh] enthalten und wir werden später die Voronoi-Diagramme zu Hilfe nehmen. Alle Verfahren arbeiten in $O(n \cdot \ell ogn)$. Zum Problem bei höheren Dimensionen siehe Alt: Computational Discrete Mathematics, Springer (2001).



Zum dichtesten Punkt (*nearest neighbor*)

In der ℓ_1-Metrik oder Manhatten-Metrik ist die Entfernung zwischen (x_1, y_1) und (x_2, y_2) im $\mathbb{N}^2$ leicht zu ermitteln: $|x_1 - x_2| + |y_1 - y_2|$.

In der Euklidischen- oder ℓ_2-Metrik arbeiten einige Methoden bestmöglich in $O(n \cdot \ell ogn)$ Zeit und bei entsprechend aufwändigen Datenstrukturen, ähnlich den *range trees*, mit $O(n)$ Platz.

Bei bekannter Dimension $d > 2$ benötigen die bekannten Verfahren polynomiale Vorbereit-ungszeit (*preprocessing*) sowie ebensolchen Speicherplatz. Die Frage: „Was ist der nächste Punkt zu p?“ kann in $O(\ell ogn)$ beantwortet werden (*query time*), so dass das dichteste Punkte Problem wieder $O(n \cdot \ell ogn)$ wird, allerdings ***ohne*** die Vorbereitungszeit!

Point Location in Hyperplanes

Leider geht die Dimension oft exponentiell in die Konstanten ein! Erst in

[Meiser: Point location in arrangements of hyperplanes,

Inf.□&□Comput. 106, (1993)]

wird Verfahren mit $O(n^{d+\varepsilon})$ *preprocessing* und $O(d^5 \cdot \ell ogn)$ *query time* beschrieben.

Alle diese Verfahren werden von dem *brute force* Vorgehen in den Schatten gestellt. Dieses benötigt keine Vorbereitungszeit, hat nur $O(n \cdot d)$ Platzbedarf, und ebensolche *query time*.

Approximative Verfahren

Approximative Verfahren verwenden ein Ergebnis, dass es erlaubt Projektionen auf kleiner dimensionierte Räume zu verwenden, ohne zu viel Informationen bezgl. des Abstands zu verlieren. Die damit erzielten Ergebnisse sind zu dem von Meiser ähnlich, und relativ aufwändig zu implementieren!

Mehr zum dichtesten Punkt (*nearest neighbor*)

Die *Cube-Method* von Alt/Heinrich-Litan/Hoffman (2001/2002) löst das Problem des dichtesten Punktes exakt und mit kleiner Konstante für die mittlere Laufzeit:

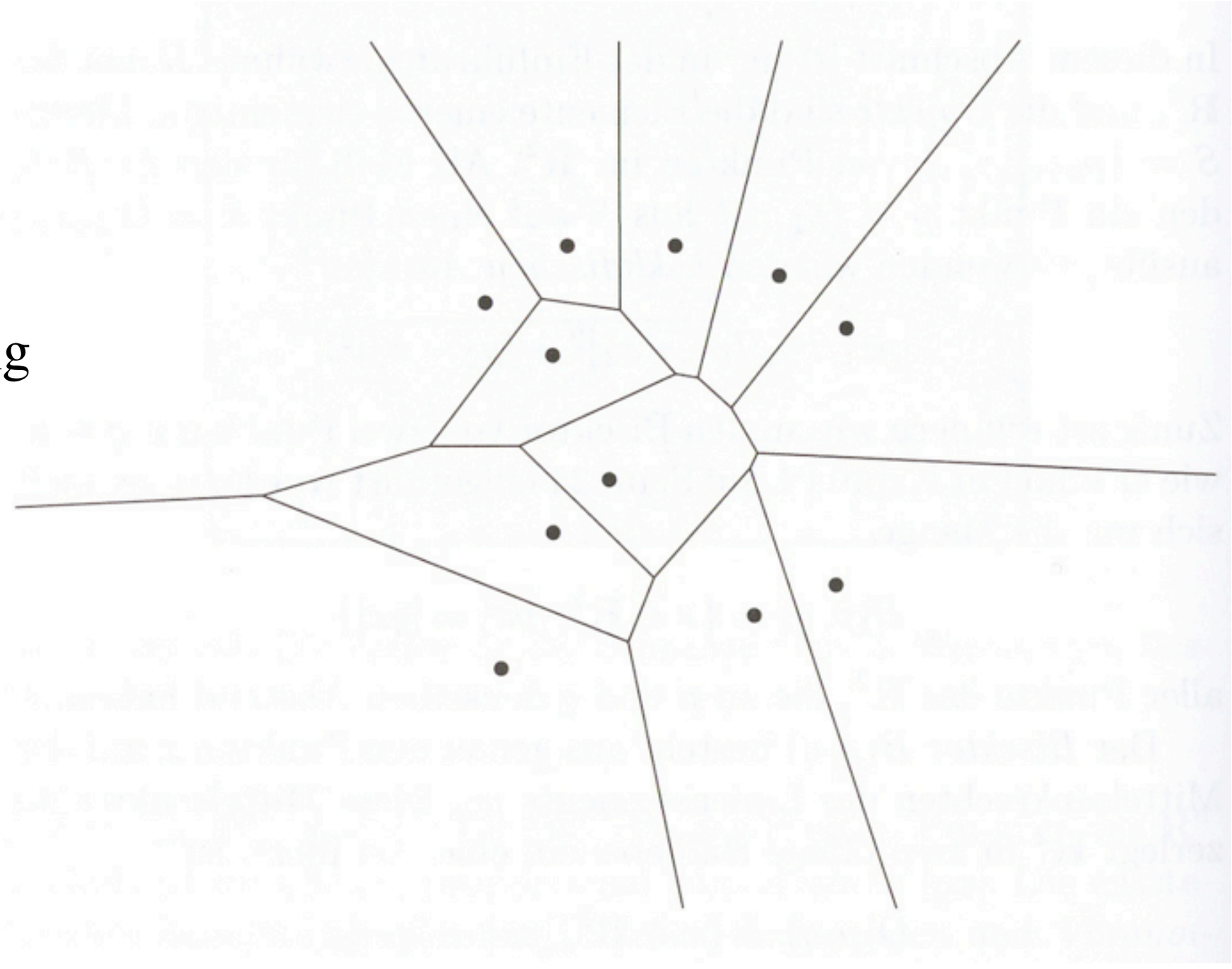
- *preprocessing* ist nicht erforderlich,
- Platzbedarf ist identisch dem zum Speichern der Punktmenge,
- die mittlere Laufzeit ist $\Theta(n \cdot d / \ell ogn)$ für eine Anfrage nach dichtestem Punkt zu einem gegebenen bei zufällig bestimmter Punktmenge P!

Für praktische Werte von d und n arbeitet dieser Algorithmus bis zu 10-mal schneller als die *brute force* Methode!

Siehe dazu: Alt/Heinrich-Litan/Hoffman: Exact ℓ_∞ nearest neighbor search in high dimensions, Proc. 17th ACM Symp. on Computational Geometry, 157-163. (June 2001).

Das sog. Telefonzellen Problem (*post office problem*)

Mit dieser Karte lässt sich feststellen, wo die nächstgelegene Telefonzelle/ Sendemast liegt, sofern Luftlinienentfernung angemessen ist!
Solche Karten heißen “Voronoi-Diagramme”.



Voronoi Diagramm im $\mathbb{R}^2$

Sei $S \subset \mathbb{R}^2$ eine (meist endliche) Menge von Punkten, dann bezeichne $VR(p, S)$ die **Voronoi-Region** bezüglich $p \in S$:

- $VR(p, S)$ ist jene Menge von Punkten, zu denen $p \in S$ der Punkt mit dem kleinsten Abstand ist.

- Die (Halb-)Gerade zwischen zwei Voronoi-Regionen heißt **Voronoi-Kante**, diese ist Teil des **Bisektors** von zwei Punkten $p, q \in S$, alle Punkte mit gleichem Abstandvon p und q.

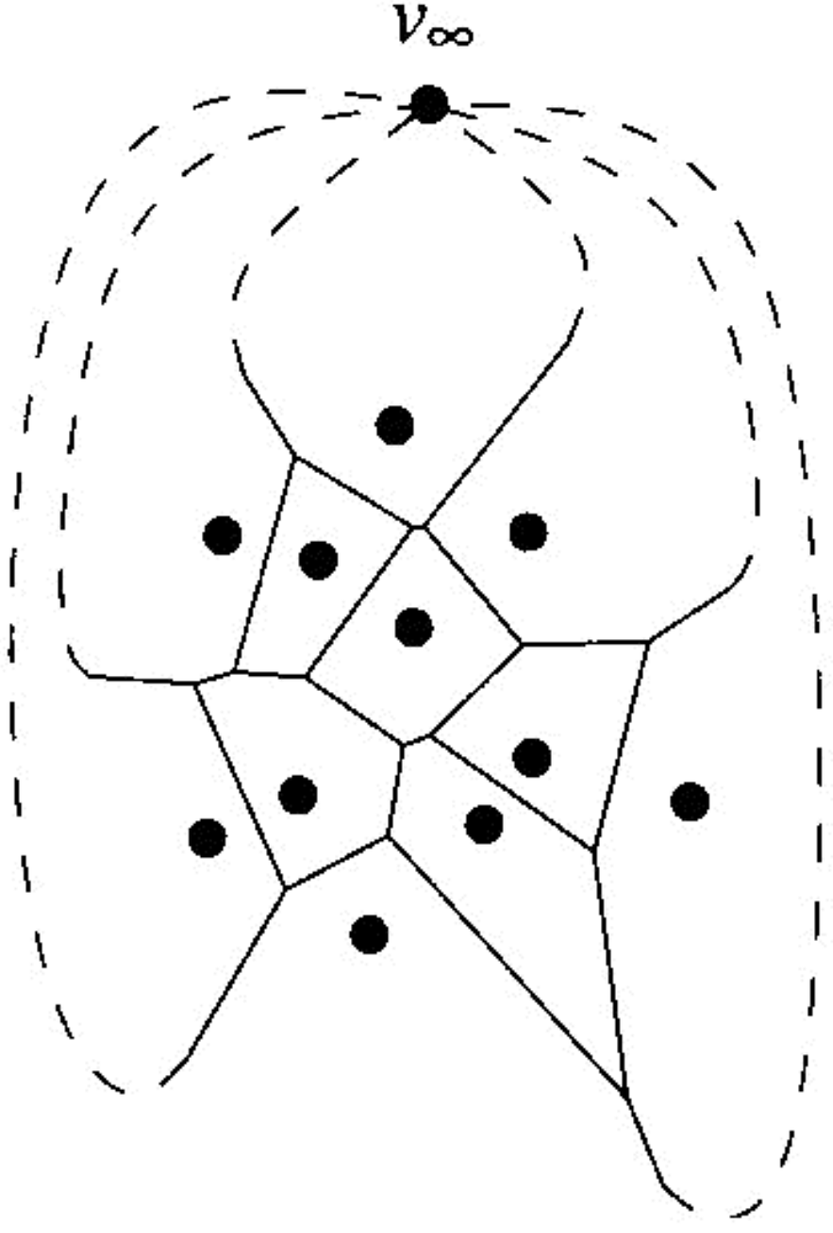
- Der Endknoten einer Voronoi-Kante heißt **Voronoi-Knoten**

Kanten- und Knotenzahl des Voronoi-Diagramms

Aus der Eulerschen Formel folgt:

Die Zahl der **Voronoi-Knoten** eines Voronoi-Diagramms zu n Punkten ist maximal $2n - 5$, die seiner **Voronoi-Kanten** höchstens $3n - 6$.

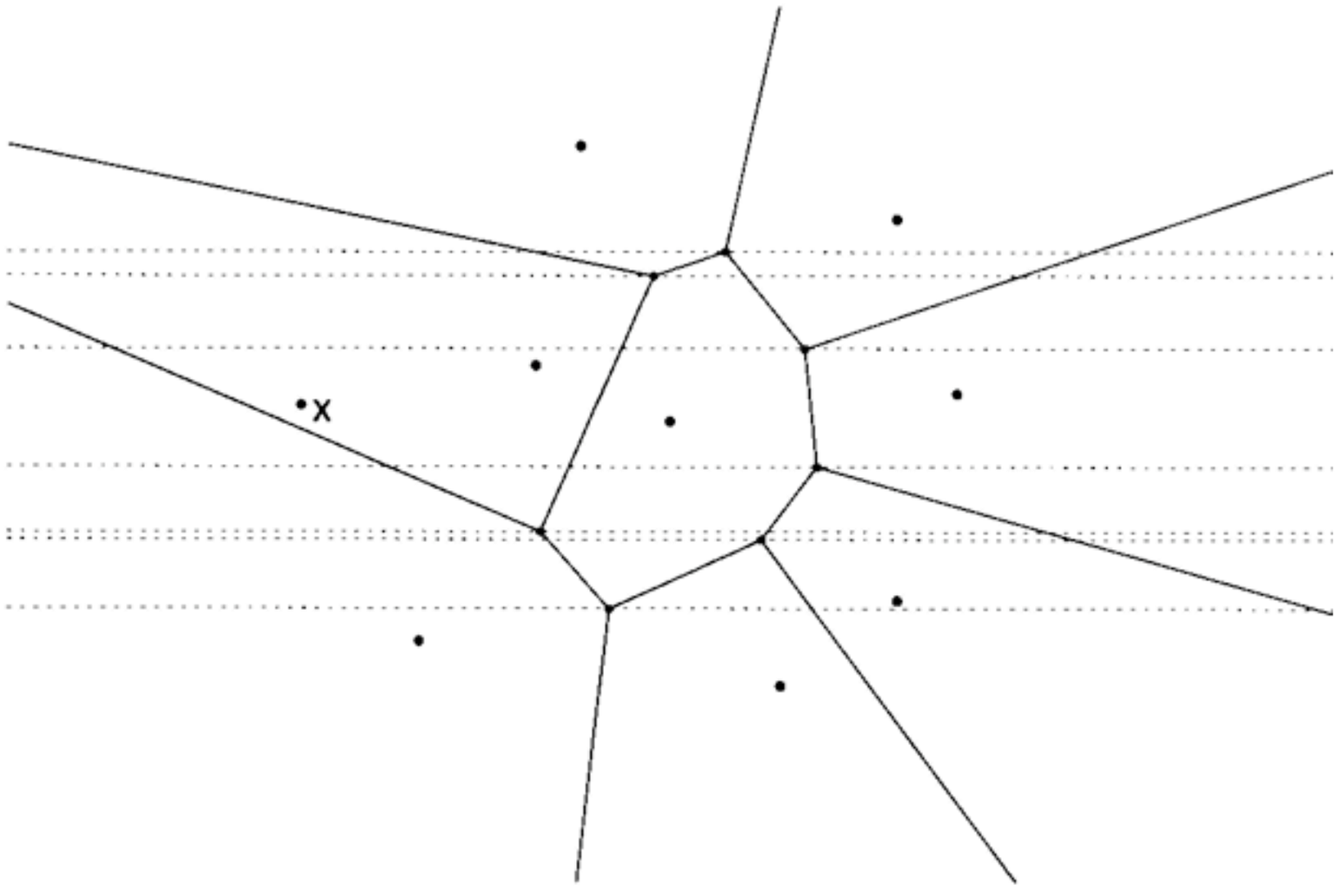
Man betrachtet dazu den zusammenhängenden Graphen, der aus dem Zusammenführen aller offenen Voronoi-Kanten entsteht (vergleiche de Berg, S. 148, und ähnlich Klein, S. 220.)



Point Location im Voronoi-Diagramm

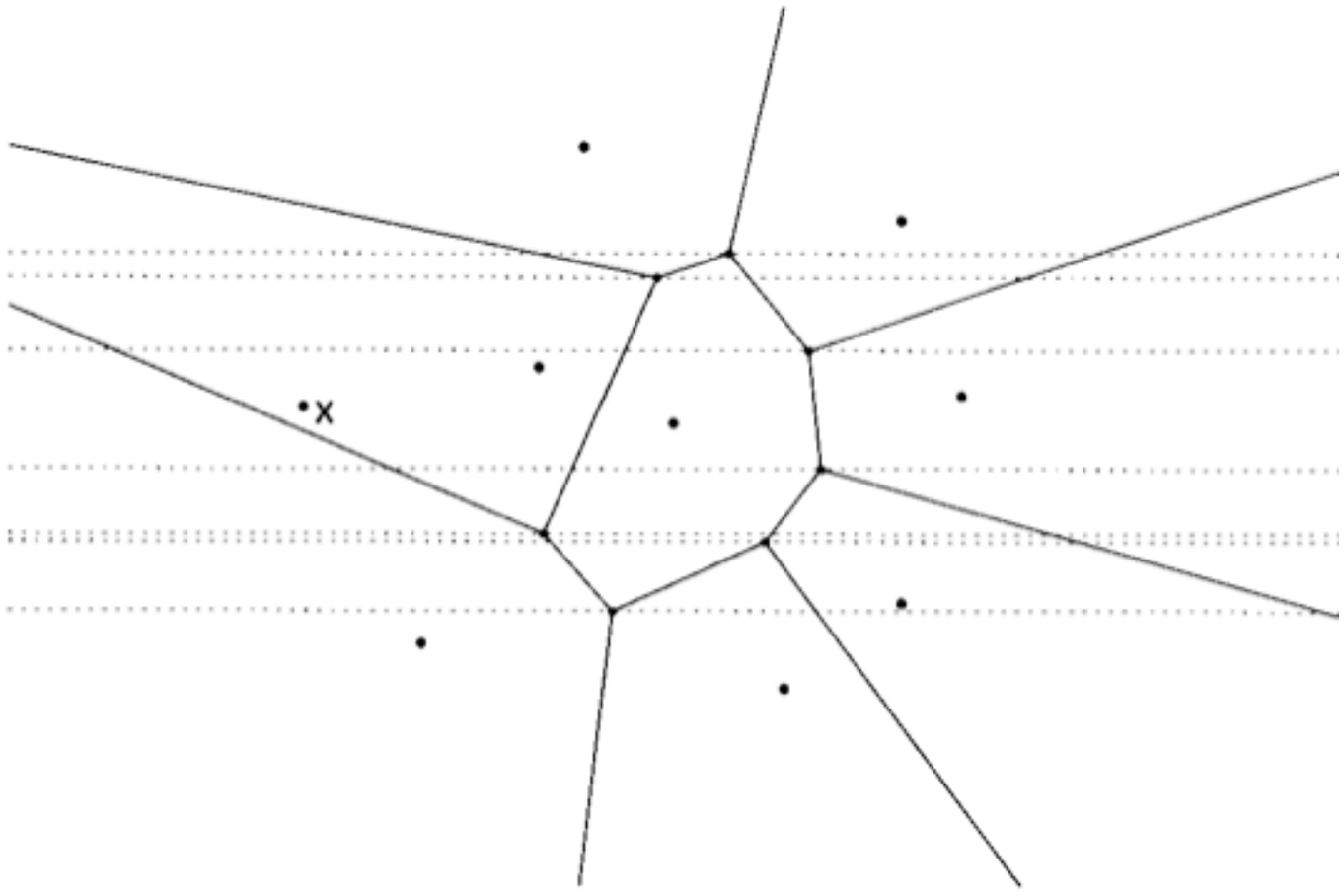
Um festzustellen, in welcher **Voronoi-Region** ein beliebiger Punkt $x = (x_1, x_2)$ liegt, zerlegt man zunächst das Voronoi-Diagramm in waagerechte Streifen indem Geraden durch die Voronoi-Knoten gelegt werden.

Bei festem x_1 wird für den Punkt x in der y-Koordinate durch binäre Suche der Streifen bestimmt, in dem x liegt. Innerhalb des gefundenen Streifens treffen keine zwei Voronoi-Kanten aufeinander, so dass diese den Streifen in Abschnitte teilen.



Point Location im Voronoi-Diagramm

Den Abschnitt, der x enthält findet man wieder logarithmisch heraus. Nach dem vorigen Ergebnis gibt es nur $O(n)$ viele Kanten im Voronoi-Diagramm, also auch nicht mehr Abschnitte bzw. Streifen, so dass diese Suche in $O(\ell og(n))$ Zeit beendet werden kann.



Point Location im Voronoi-Diagramm

Theorem
Zu einer endlichen Menge S von n Punkten in der Ebene kann mit Hilfe des Voronoi-Diagramms eine Struktur aufgebaut werden, mit der sich für jeden beliebigen Anfragepunkt x in $O(\ell og(n))$ Zeit derjenige Punkt aus S finden läßt, der x am nächsten liegt.

Leider ist der Platzbedarf bei diesem Vorgehen quadratisch in n.

Ist das Voronoi-Diagramm bekannt, so gibt es jedoch Verfahren, die mit linearem Zeit- und Platzbedarf eine Struktur aufbauen, in der jeder Punkt in logarithmischer Zeit lokalisiert werden kann!

Siehe: H. Edelsbrunner: Algorithms in Combinatorial Geometry, EATCS Monographs, vol. 10 (1987).

Dichtestes Punktepaar mit Hilfe des Voronoi-Diagramms ermitteln:

Lemma:
Sei $S = P \cup Q$ eine Zerlegung einer endlichen Punktmenge S in nicht leere P und Q. Weiter seien $p_0 \in P$ und $q_0 \in Q$ derart, dass $|p_0 q_0| = \min\{\ |pq| \mid p \in P \wedge q \in Q\}$.
Dann haben die Regionen $VR(p_0, S)$ und $VR(q_0, S)$ eine gemeinsame Kante.

Beweis:
Wäre dies nicht so, dann gäbe es einen Punkt z auf der Linie $p_0 q_0$, der in einer Region $VR(r, S)$ liegt mit $q_0 \neq r \neq p_0$.
Sei $r \in Q$ ($r \in S$ ist symmetrisch). Wegen $z \in VR(r, S)$ ist $|rz| \leq |q_0 z|$.
Folglich also:
$|p_0 r| \leq |p_0 z| + |zr|$ (Dreiecksungl.)
$\leq |p_0 z| + |z q_0| = |p_0 q_0| \leq |p_0 r|$ (Minimalität von $|p_0 q_0|$). Somit gilt $|p_0 q_0| = |p_0 r|$ sowie $|zr| = |z q_0|$.

Fortsetzung des Beweises:
Mit $|p_0q_0|=|p_0r|$ sowie $|zr|=|zq_0|$ ist p_0 auf Bisektor von q_0 und r und das Liniensegment q_0p_0 - und damit auch z - auf der Seite zwischen q_0 und dem Bisektor. Damit ist aber $|zq_0|<|zr|$ im

Widerspruch zu Obigem. Also gibt es weder einen anderen Punkt z noch den Punkt r und alles ist gezeigt.

Wendet man das Lemma auf $P:=\{p\}$ und $Q:=S\setminus\{p\}$ an, so folgt: Jeder nächste Voronoi-Knoten liegt in einer benachbarten Region, d.h. in einer, die gemeinsame Kante mit der eigenen Region hat.

Satz:
Ist das Voronoi-Diagramm $V(S)$ vorhanden, so kann für jeden Punkt p der nächste Nachbar in Zeit $O(n)$ gefunden werden.

Dichtestes Punktepaar mit Hilfe des Voronoi-Diagramms:

Beweis:
Man braucht nur die endlich vielen Nachbarregionen eines Voronoi-Knotens aufzusuchen. Davon gibt es aber nur $O(n)$ viele, weil es ja höchstens $3n-6$ Kanten gibt!

Wenn man jede Kante maximal zweimal überschreitet, so kann man dies für alle Voronoi-Knoten machen und dann in der Liste der nächsten Nachbar-Entfernungen die kürzeste in $O(n)$ Zeit heraussuchen:

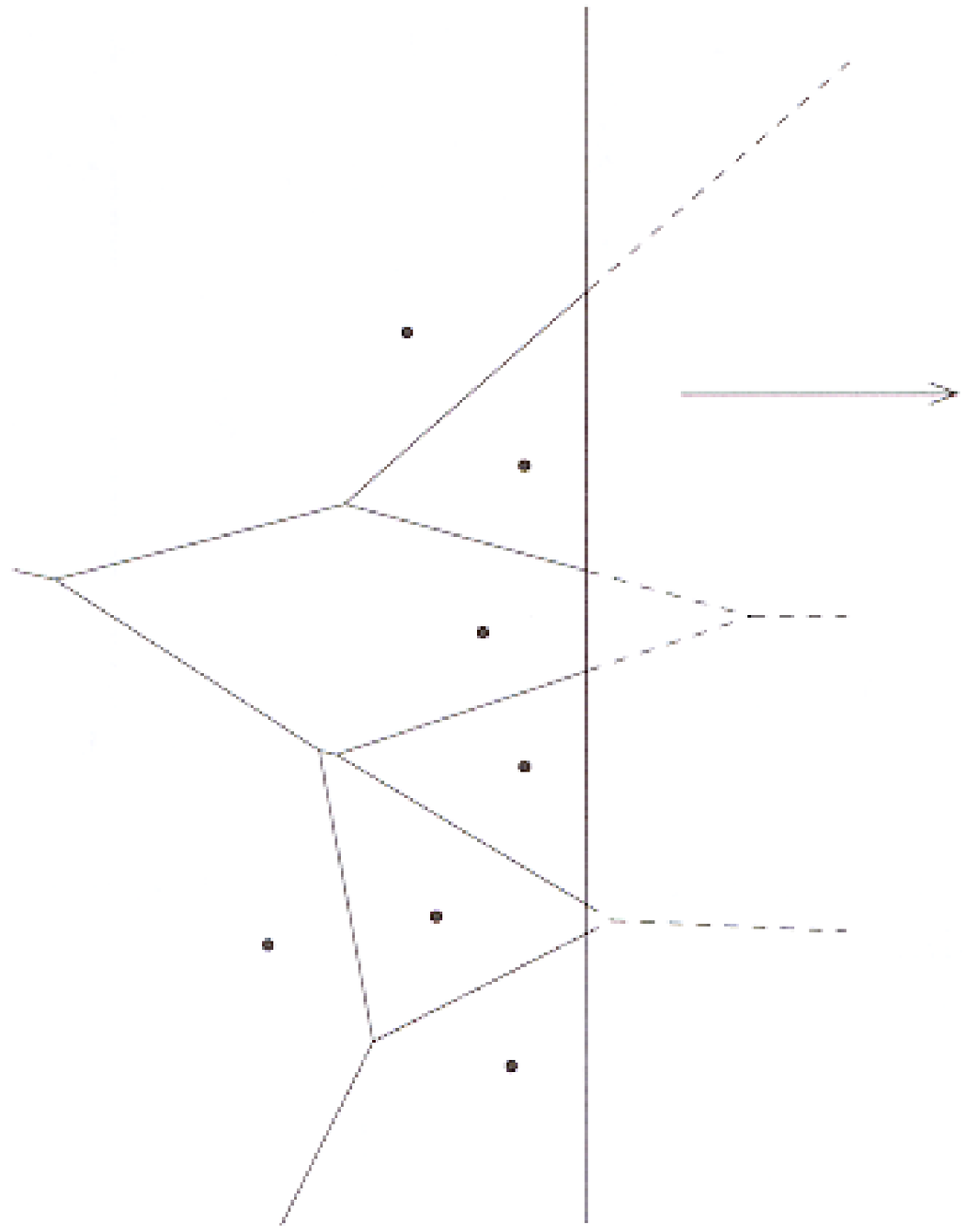
Satz:
Ist das Voronoi-Diagramm $V(S)$ vorhanden, so kann ein dichtestes Punktepaar in Zeit $O(n)$ gefunden werden.

Beide Fragen lassen sich somit insgesamt in Zeit $O(n \cdot \ell og\ n)$ beantworten!

S.J. Fortune: Algorithmus zur Konstruktion des Voronoi Diagramms

Eine *plane sweep* Technik führt in der Ebene zum $O(n\ \ell og\ n)$ Algorithmus [siehe de Berg und Klein]:

Die *sweep line* bewegt sich von links nach rechts und definiert durch Schnittpunkte die *update*-Ereignisse der bisher gefundenen Voronoistruktur.

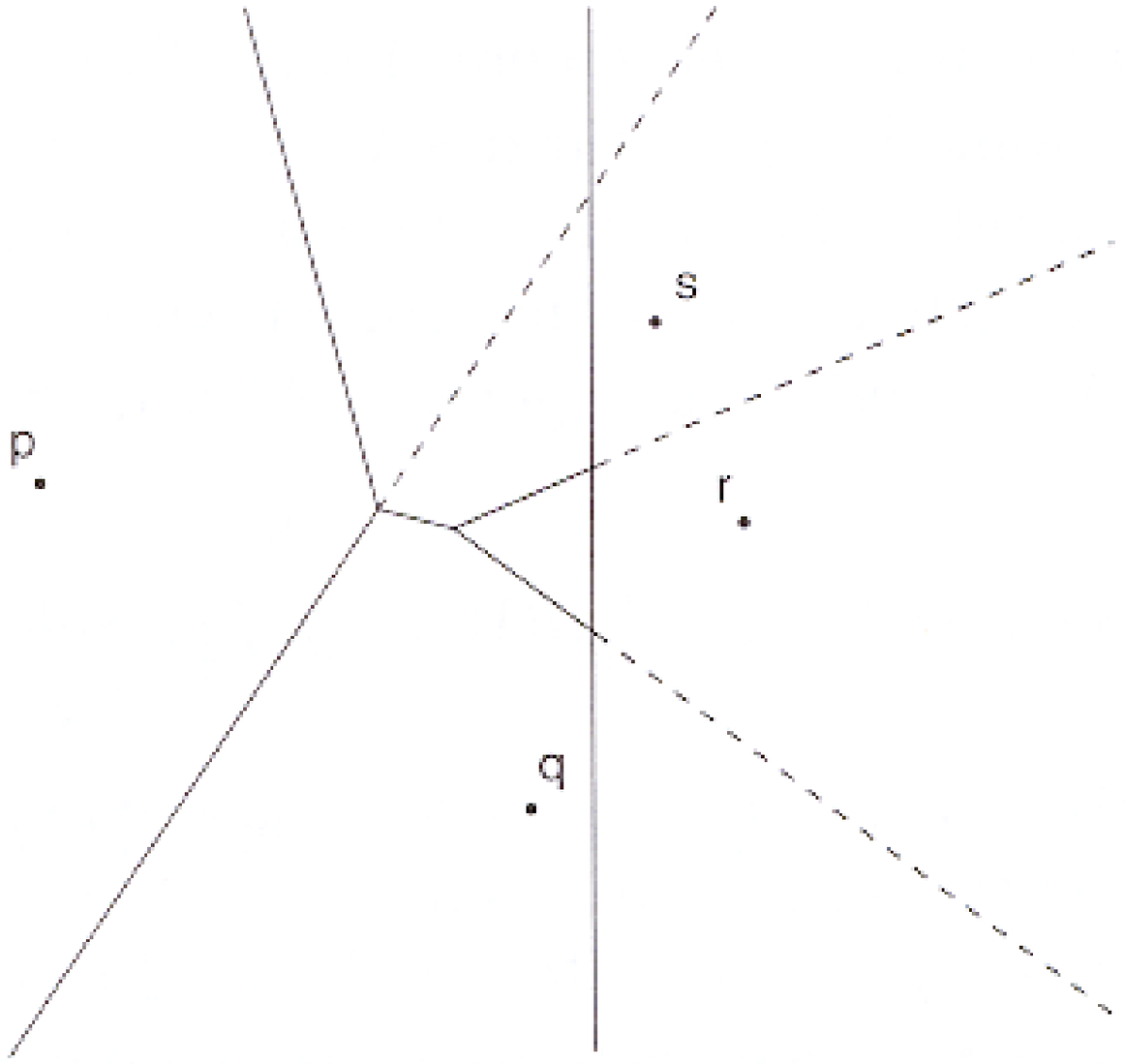


Was aber sind die *update*-Ereignisse?
Schnittpunkte der *sweep line* mit einer Voronoi-Kante?

Wegen dieses Problems dauerte es bis 1987 um Voronoi Diagramme mit dieser Technik zu konstruieren. Das erste Verfahren von Shamos & Hoey (1975) nutzte *divide and conquer* [siehe Klein oder Ottmann & Widmayer]

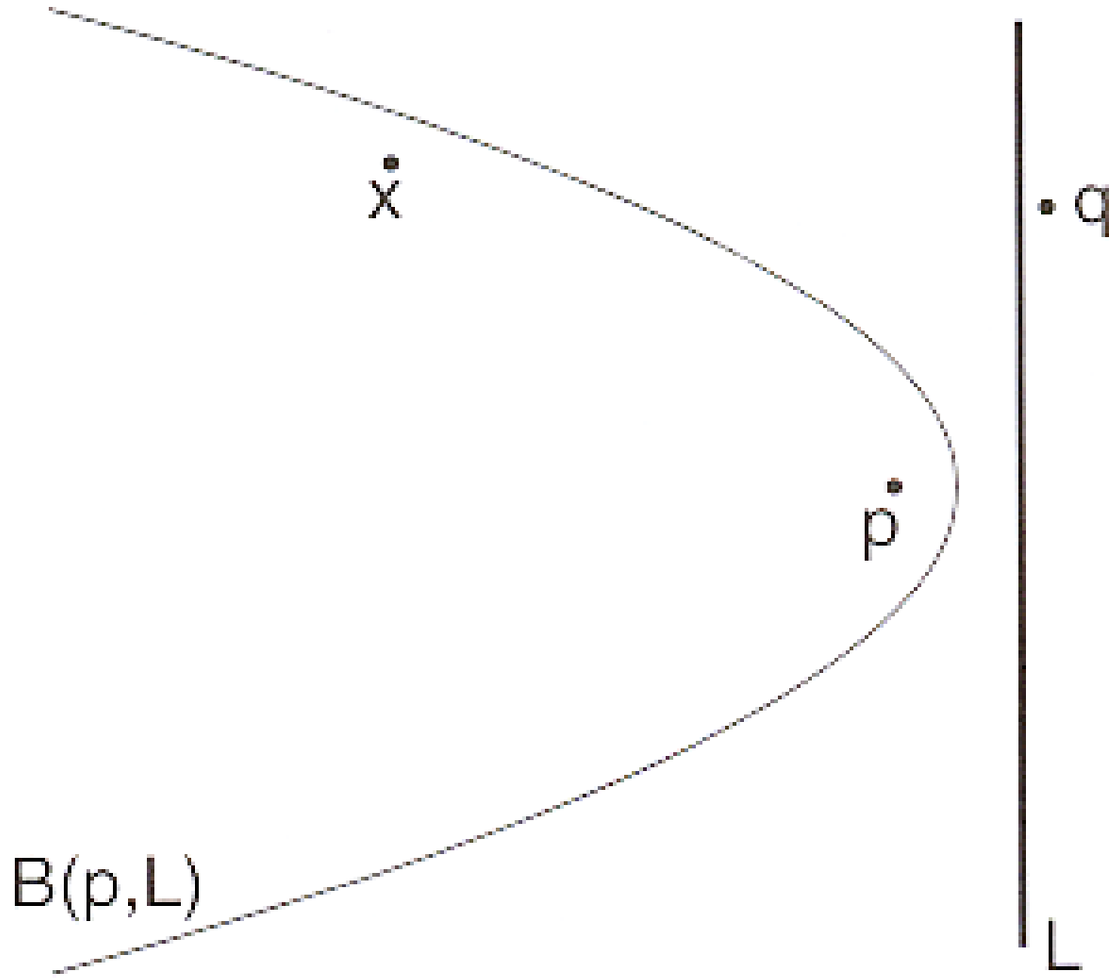
Was war das Problem?

Zu dem abgebildeten Zeitpunkt kennt man die Punkte s und t noch nicht, und damit auch nicht die Abschnitte der Voronoi-Kanten und deren Schnittpunkt links von der *sweep line*!



eine “Welle” der Wellenfront

Für die Punkte rechts von der *sweep line* L ist der Bereich links von der Bisektions-Parabel $B(p,L)$ nicht mehr beeinflussbar und braucht nie mehr geändert zu werden!



Wieso ist die Bisektionslinie zwischen einem Punkt und einer Geraden eine Parabel?

$B(p,L)$

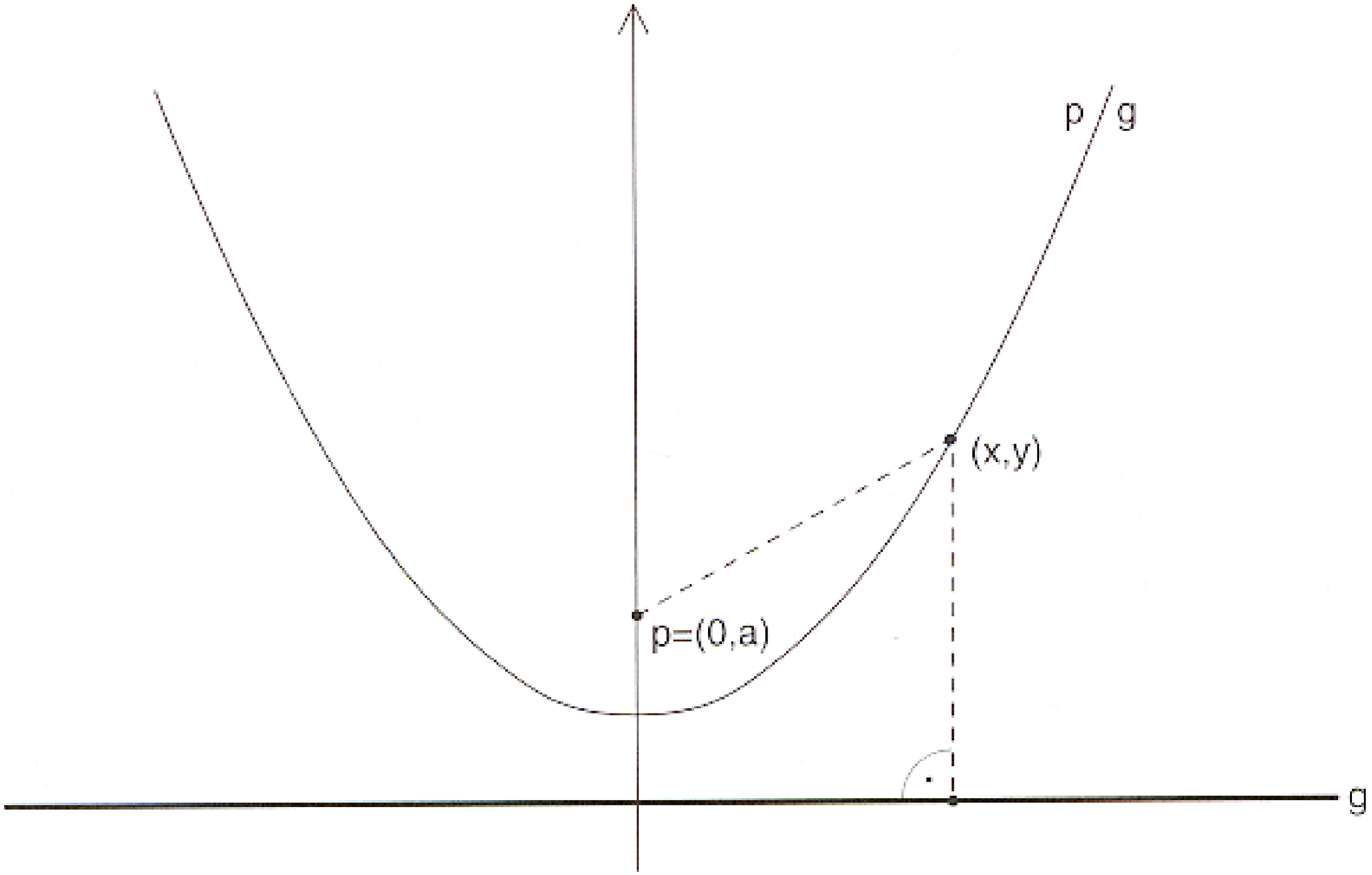


Abbildung 5.42: Der Bisektor $B(p,g)$ von $p=(0,a)$ und $g=\{Y=0\}$ ist die Parabel $\{Y=\frac{X^2}{2a}+\frac{a}{2}\}$.

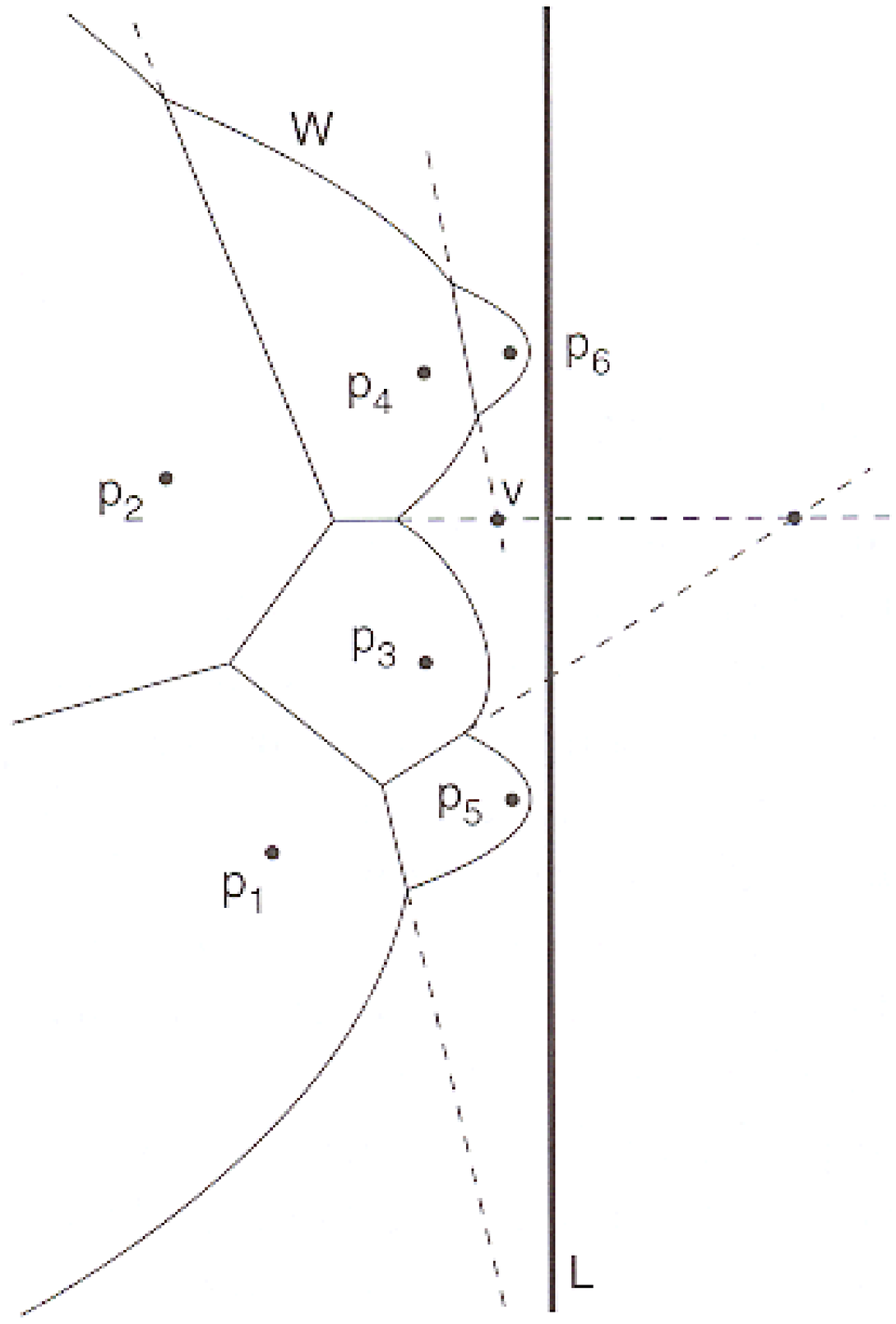
Die Wellenfront

Die Bisektions-Parabeln zwischen *sweep line* und Voronoi-Punkt (nicht Voronoi-Knoten!) heißen “Wellen”,

die Parabelstücke der Wellenfront heißen “Wellenstücke”.

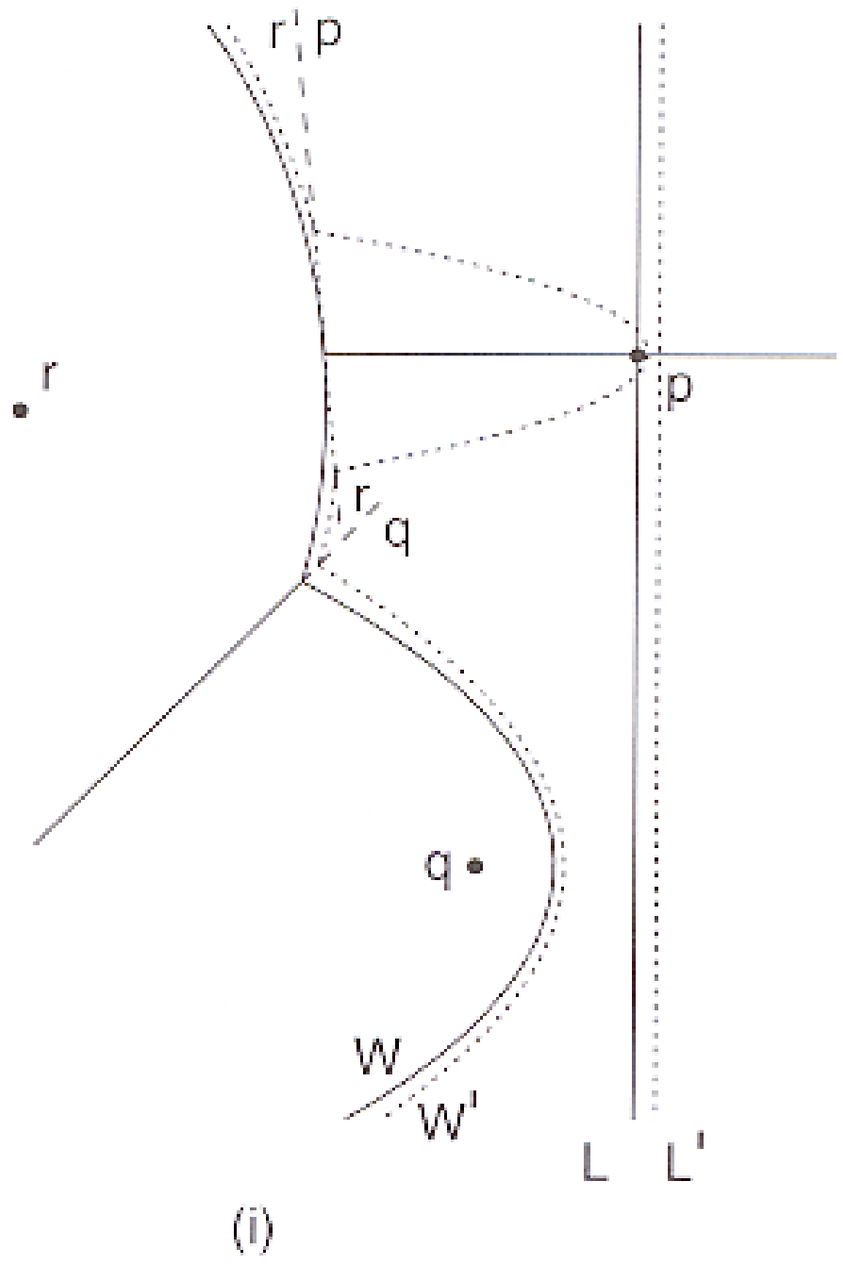
Alles links von der Wellenfront bleibt unverändert.
Gespeichert wird also nur das, was sich garantiert nicht mehr ändert!

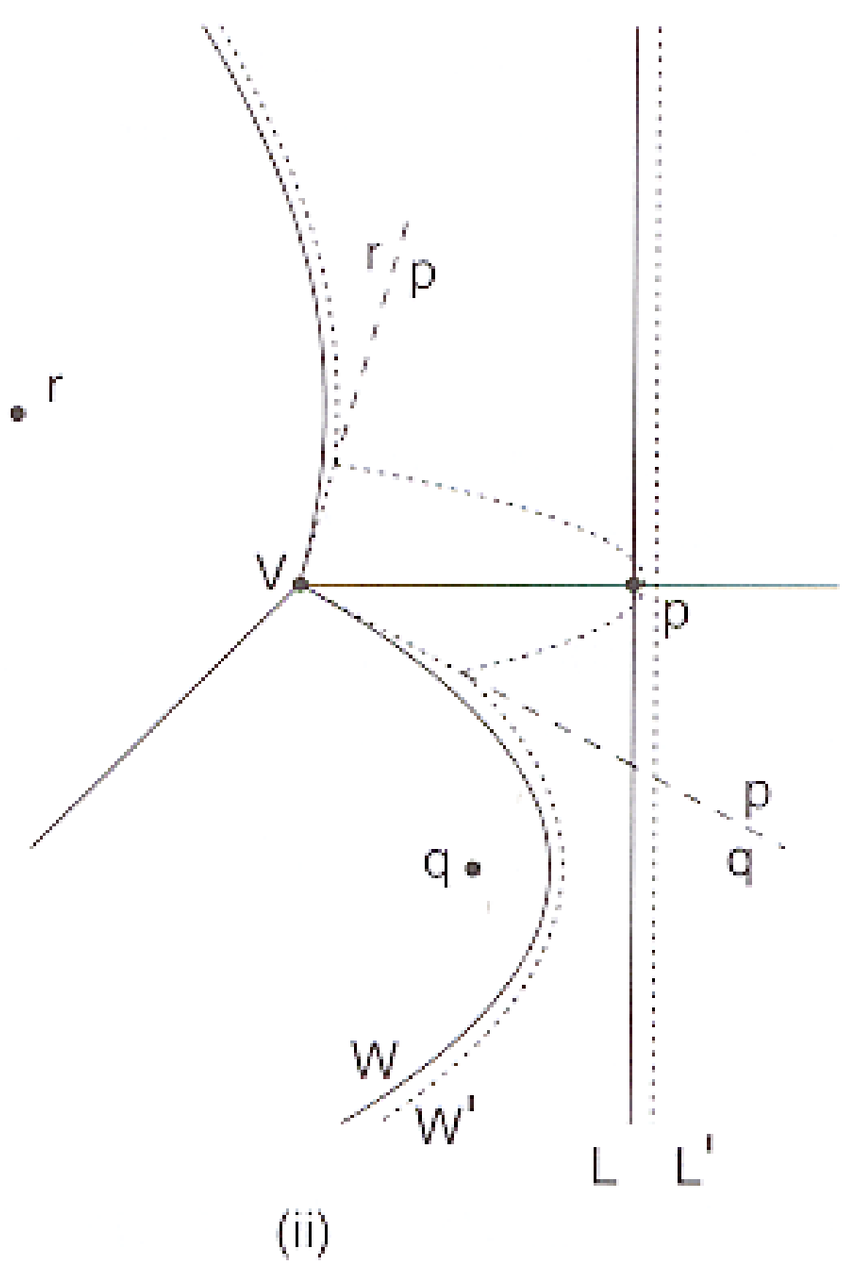
Die gestrichelten Teile der Voronoi-Kanten könnten sich noch verändern, in dem neue Verzweigungsknoten und Kanten hinzukommen.



Das Wandern der Wellenfront

Die Welenfront wandert mit halber Geschwindigkeit hinter der *sweep line* her, wobei die Schnittpunkte der Wellen, also die Kontaktpunkte der Wellen, entlang der Bisektionslinien (*spikes*, im Bild gestrichelt) weiter "rutschen". Neue Wellen entstehen mit auftauchenden Voronoi-Punkten.

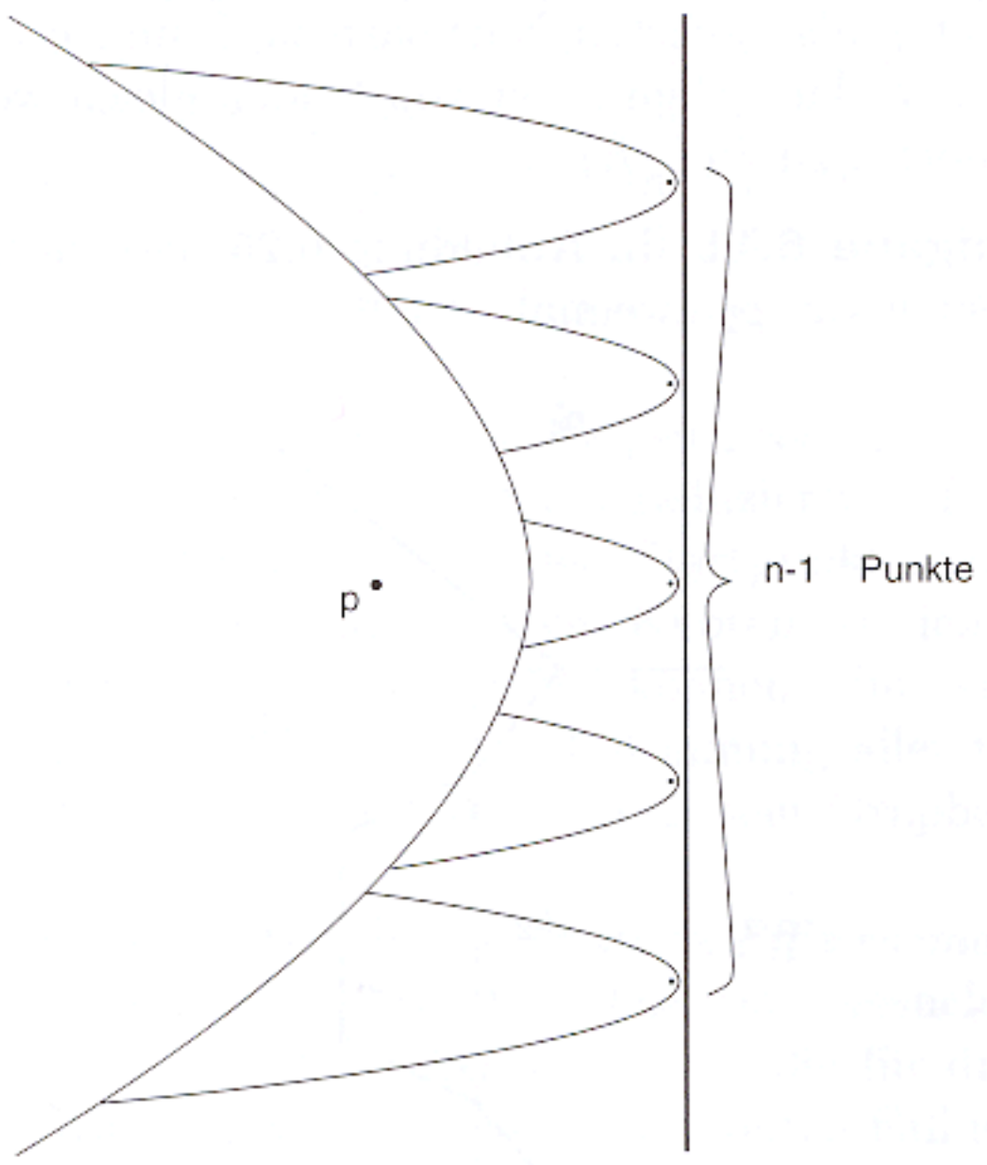




Monotonie und Zusammenhang der Wellenfront

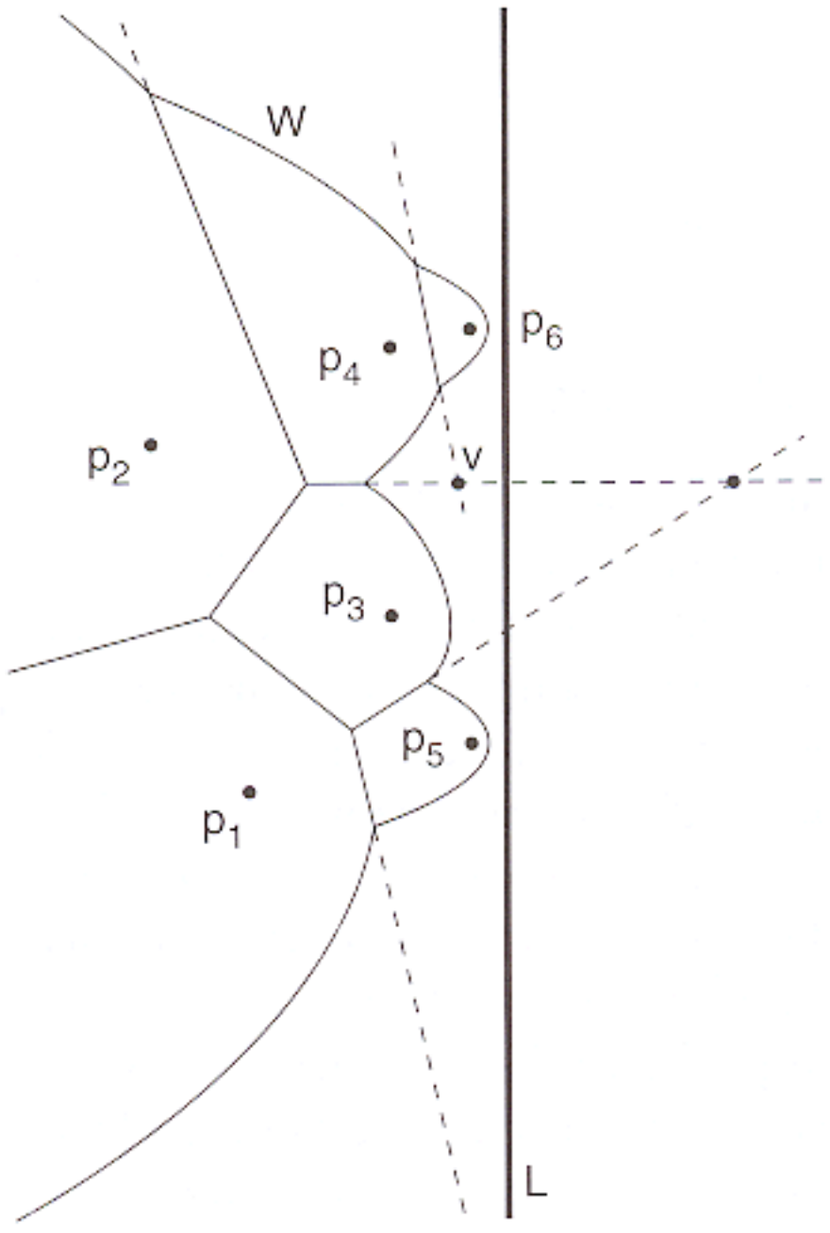
Da sich die Wellen (nach links geöffnete Bisektionsparabeln) an den Verbindungspunkten der Wellenstücke schneiden ist die Wellenfront von oben nach unten zusammenhängend und monoton.

Eine Welle kann an mit mehreren Stücken an der Wellenfront beteiligt sein!
Deren Annzahl ist stets in $O(n)$.



Ereignisse, die strukturelle Änderung der Wellenfront auslösen:

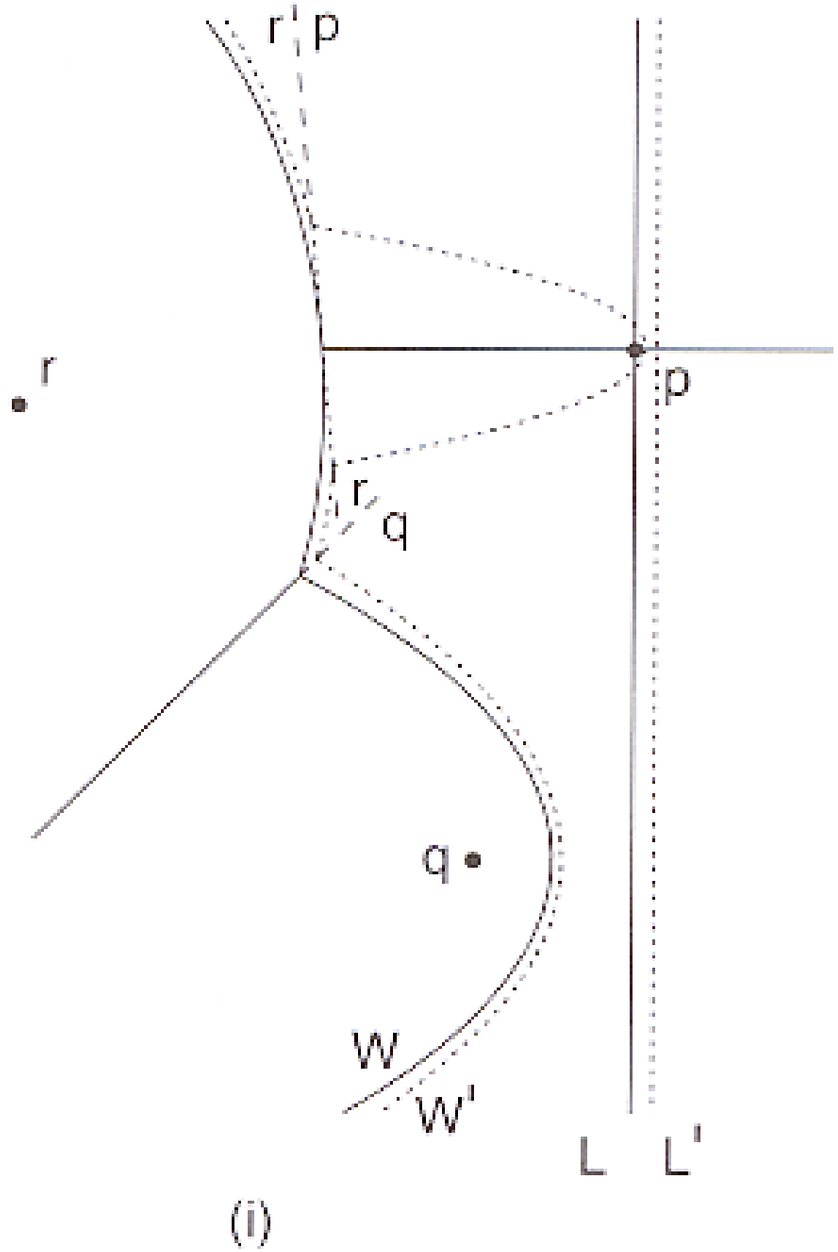
1. Ein Wellenstück verschwindet:



Je weiter die *sweep line* vorrückt, desto kleiner wird das Wellenstück zwischen den Bisektoren $B(p_3,p_4)$ und $B(p_4,p_6)$, bis seine benachbarten Wellenstücke im Punkt v zusammen kleben!

Ereignisse, die strukturelle Änderung der Wellenfront auslösen:

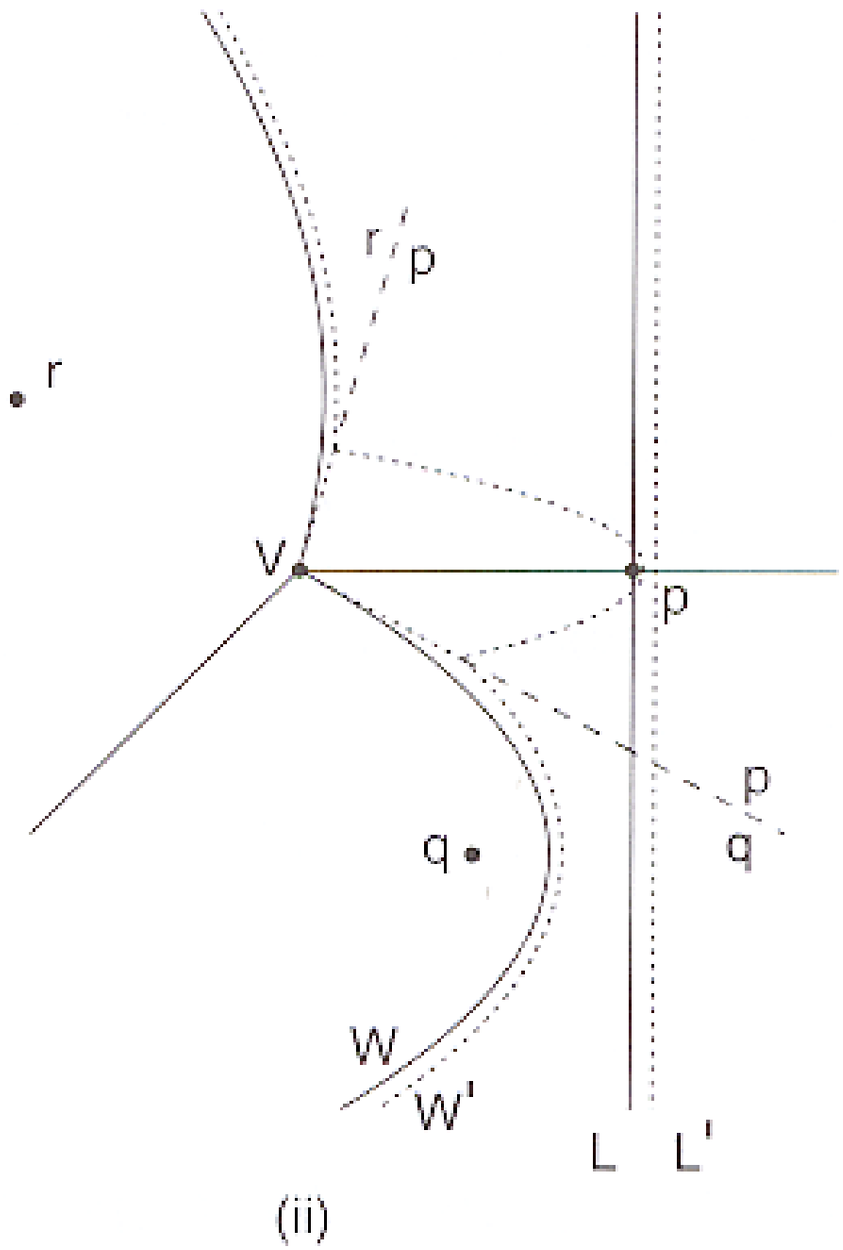
2a. Ein neues Wellenstück erscheint:



Wenn die *sweep line* beim Vorrücken, einen neuen Punkt p trifft, so ist zunächst die Senkrechte zu L durch p der Bisektor. Beim weiteren Vorrücken der *sweep line* öffnet sich der linke Teil zu einer Parabel als neuen Wellenabschnitt. Es entsteht eine neue Voronoi-Kante zwischen den Schnittpunkten der Parabel mit dem alten Wellenstück zum Punkt r. (Warum ist dies Teil des Bisektors $B(r,p)$?)
Mit den zwei neuen *spikes* daran.

Ereignisse, die strukturelle Änderung der Wellenfront auslösen:

2b. Ein neues Wellenstück erscheint:



Die *sweep line* kann beim Vorrücken, auf einen neuen Punkt p stoßen, dessen Senkrechte zu L einen Verbindungs-Knoten v zweier Wellenstücke trifft. Dieser wird zu einem Voronoi-Knoten, und die Parabel endet an den *spikes*, die zu $B(r,p)$ und $B(p,q)$ gehören.

Spike-Ereignis und Punkt-Ereignis

Spike-Ereignis:

Ein Wellenstück trifft auf den Schnittpunkt seiner beiden *spikes* und verschwindet.

Punkt-Ereignis:

Die *sweep line* trifft auf neuen Voronoi-Punkt und
ein neues Wellenstück erscheint in der Wellenfront.

Andere Ereignisse kann es nicht geben!

Die Wellenstücke werden als **record** gespeichert:

type tWellenStück = **record**
Pkt: tPunkt;
$VorPkt,\ NachPkt$: tPunkt;

Pkt bezeichnet den Punkt, aus dessen Welle das Wellenstück stammt. $VorPkt$ (bzw. $NachPkt$) bezeichnet den Punkt des unteren (bzw. oberen) benachbarten Wellenstücks. Da zwei Wellen p und q nie in der Reihenfolge

$$-p-q-p-q-$$

in der Wellefront auftauchen, ist diese Beschreibung eindeutig.

Sweep-Status-Struktur

Die **Sweep-Status-Struktur** (SSS) wird als balancierter Binärbaum (AVL-Baum) aufgebaut und enthält die nach den y-Koordinaten geordneten Punkte links der *sweep line* in einem festen Streifen. Einfügen und Entfernen ist dann in $O(\ell og\ n)$ Zeit möglich und Bereichsanfragen in $O(k + \ell og\ n)$.

Daneben wird eine **Ereignis-Struktur** (ES) als **record** verwendet:

```
type tEreignis = record
        Zeit: real;
        case Typ:(SpikeEreig, PunktEreig) of
            SpikeEreig:
                AltWStück: tWellenStück;
                SchnittPkt: tPunkt;
            PunktEreig:
                NeuPkt: tPunkt;
```

Welche Informationen werden gespeichert?

Die Zeit wird durch die x-Koordinate der *sweep line* "gemessen".

Die SSS muss weiter enthalten:

$UWStück(SSS,\ y,\ x)$:

bestimmt das Wellenstück unterhalb von $\{Y=y\}$ zum Zeitpunkt x.

und

$OWStück(SSS,\ y,\ x)$:

bestimmt das Wellenstück oberhalb von $\{Y=y\}$ zum Zeitpunkt x.

sowie auch:

$FügeEin(SSS, WStück,\ x)$:

fügt Wellenstück *WStück* in die Ordnung zum Zeitpunkt x ein.

und

$Entferne(SSS, WStück,\ x)$:

entfernt Wellenstück *WStück* zum Zeitpunkt x.

Der *sweep line* Algorithmus

Initialisierung:

Bilde SSS und ES; sortiere die n Punkte nach x-Koordinate; erzeuge Punktereignisse und füge dies in ES ein.

Sweep:

while: $ES \neq \emptyset$ **do**

$Ereignis := NächstesEreignis(ES)$

with $Ereignis$ **do**

case Typ **of**

SpikeEreig: $Entferne(SSS,\ AltWStück,\ x);$

$U{:=}UWStück(SSS,\ SchnittPkt.Y,\ Zeit);$

$O{:=}OWStück(SSS,\ SchnittPkt.Y,\ Zeit);$

$TesteSchnittErzeugeEreignis(U.VorPkt,\ U.Pkt,\ O.Pkt,\ Zeit);$

$TesteSchnittErzeugeEreignis(U.Pkt,\ O.Pkt,\ O.NachPkt,\ Zeit);$

PunktEreig: (siehe nächste Folie)

weiter im *sweep line* Algorithmus

Sweep:

while: $ES \neq \emptyset$ **do**

$Ereignis := NächstesEreignis(ES)$

with $Ereignis$ **do**

case Typ **of**

SpikeEreig: (wie vorige Folie)

PunktEreig:

$U{:=}UWStück(SSS,\ NeuPkt.Y,\ Zeit);$

$O{:=}OWStück(SSS,\ NeuPkt.Y,\ Zeit);$

with $NeuWStück$ **do**

$PKT := NeuPkt\ ;$

$VorPKT := U.Pkt\ ;$

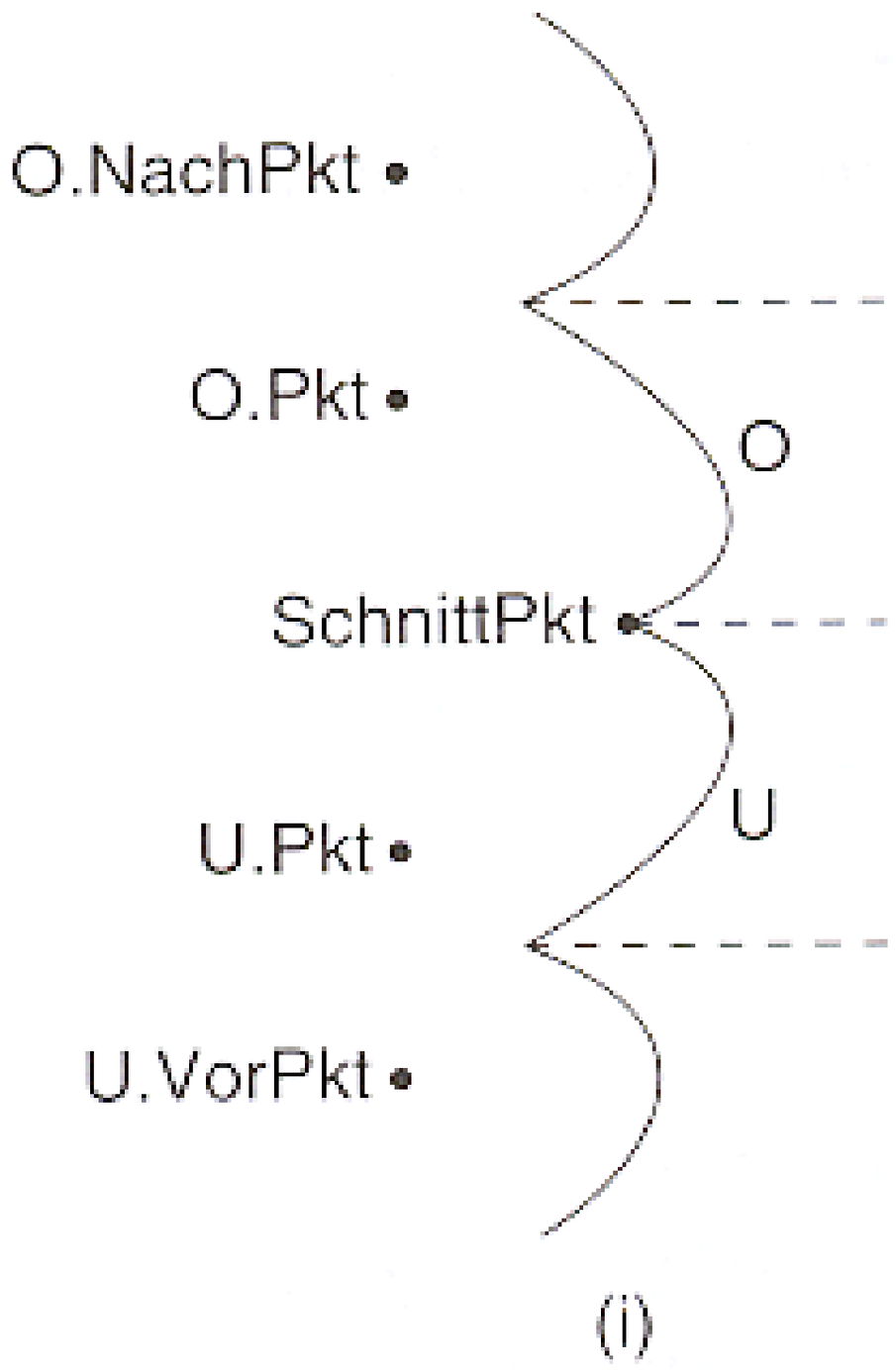
$NachPKT := O.Pkt\ ;$

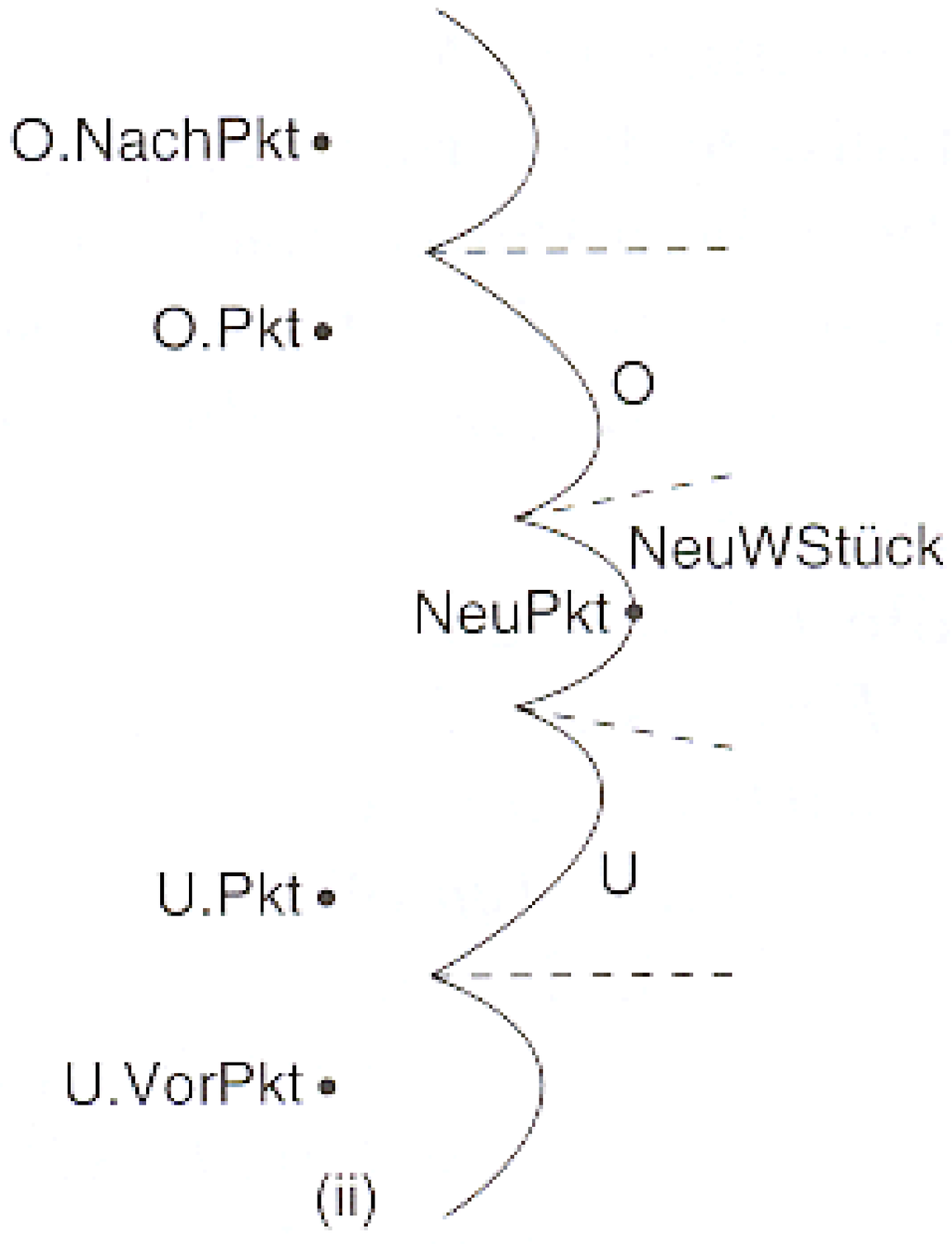
$FügeEin(SSS,\ NeuWStück,\ Zeit);$

$TesteSchnittErzeugeEreignis(U.VorPkt,\ U.Pkt,\ NeuPkt,\ Zeit);$

$TesteSchnittErzeugeEreignis(NeuPkt,O.Pkt,\ O.NachPkt,\ Zeit)$

Bearbeitung von *Spike*- und Punkt-Ereignissen





Wo ist das Voronoi-Diagramm?

Die Punkte, an denen sich die Wellenstücke berühren beschreiben die Voronoi-Kanten! Der jeweils fertige Teil kann so leicht erweitert werden. Wenn zum Schluss kein Ereignis mehr zu bearbeiten ist, bilden die noch vorhandenen *spikes* die unbeschränkten Voronoi-Kanten.

Da die Strukturen SSS und ES so implementiert werden können, dass nur $O(\ell og\ n)$ Zugrifszeit benötigt wird, und nur $O(n)$ viele Ereignisse vorkommen, ergibt die Analyse tatsächlich das optimale Ergebnis mit

$$O(n\ \ell og\ n)$$

Zeitbedarf im schlechtesten Fall und

$$O(n)$$

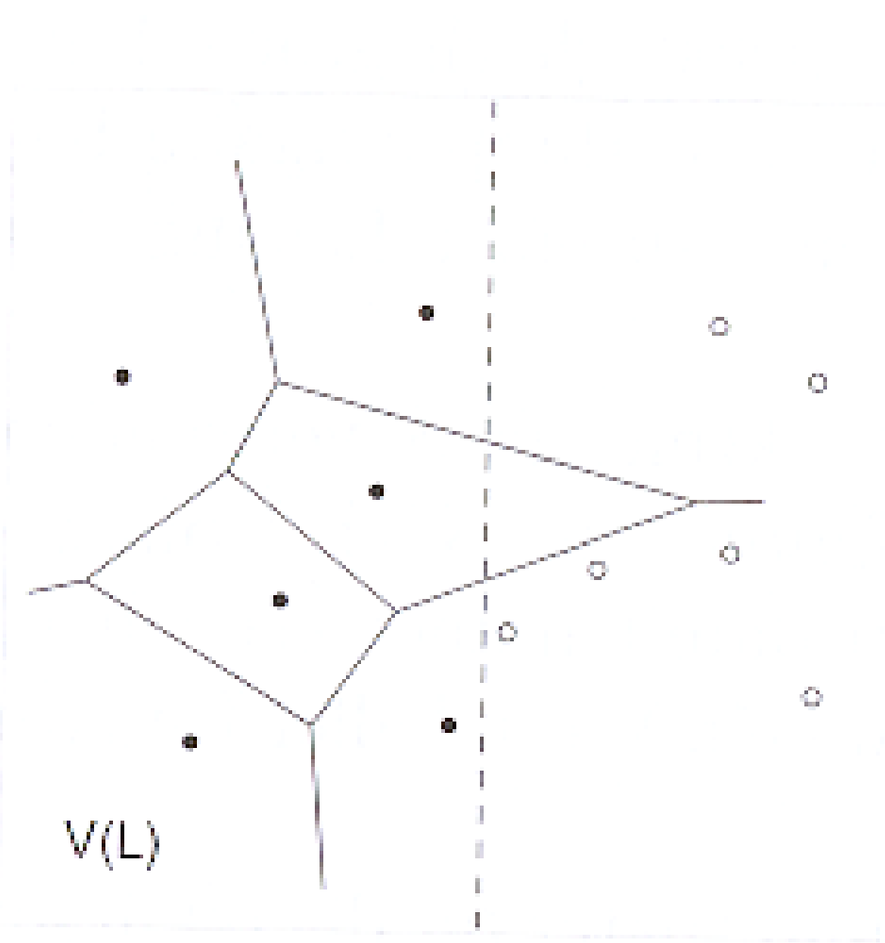
Speicherbedarf!

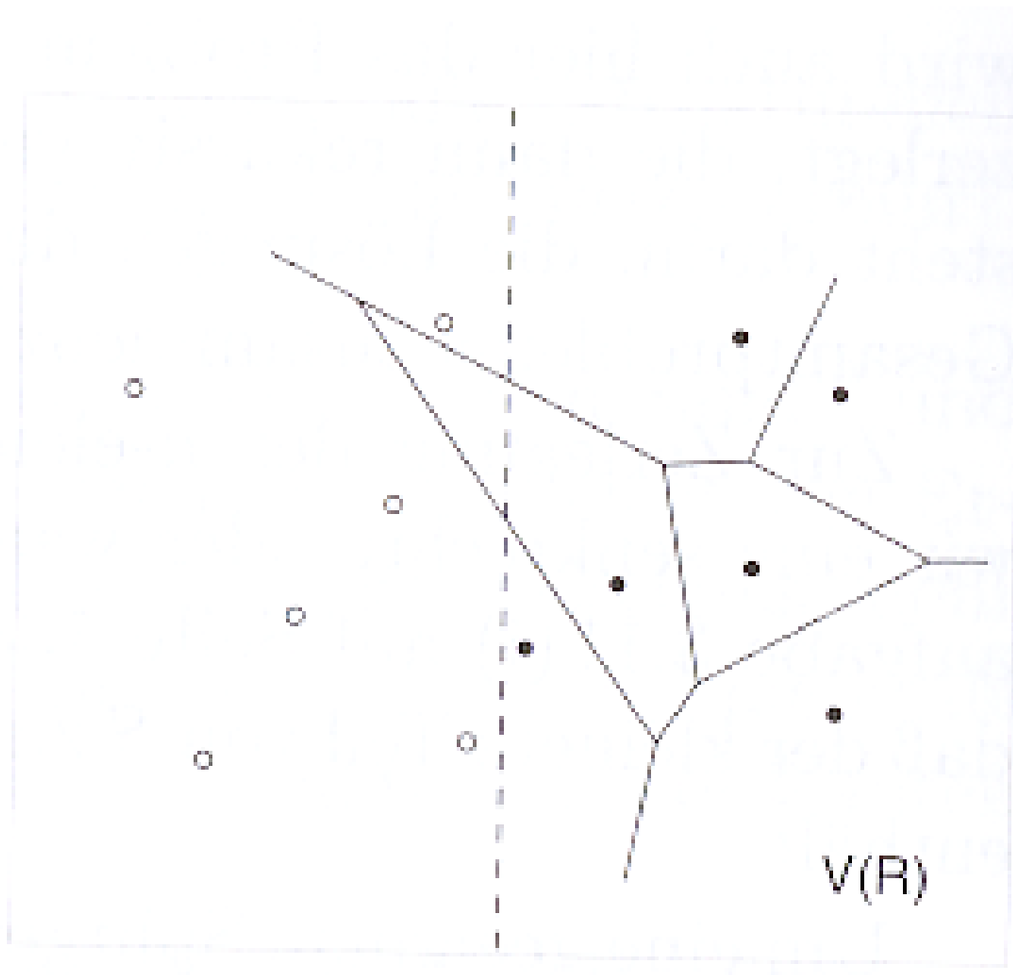
Basis des divide and conquer

1. Zerlegen der Punktmenge S in zwei etwa gleich große Teile S_1 und S_2.

2. Bestimmen der Voronoi-Diagramme $V(S_1)$ und $V(S_2)$ rekursiv mit dem gleichen Verfahren.

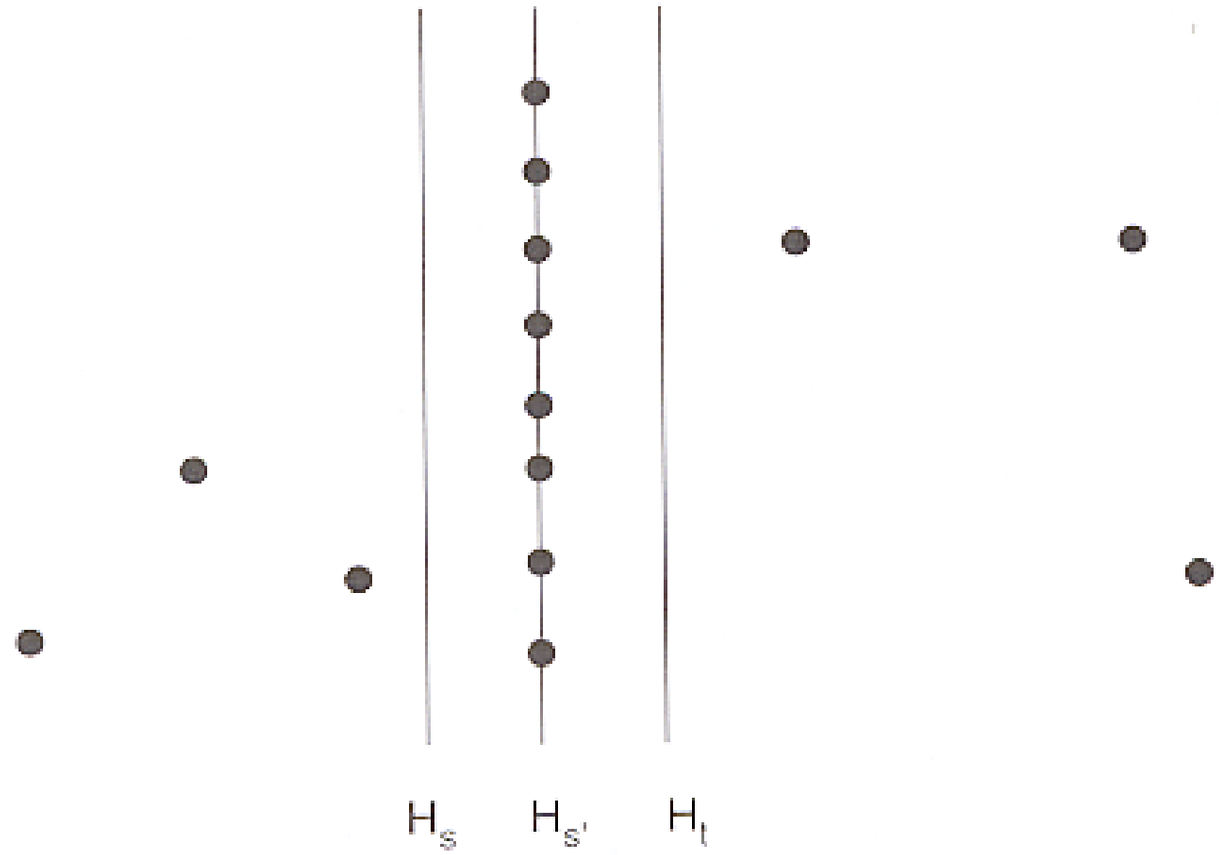
3. Zusammenfügen der fertigen Diagramme.





1. Problem: Aufteilung der Menge S

Wenn man den Median der x-Koordinaten aller Punkte bestimmt, so ist das ein Verfahren, was in $O(n\ \ell og\ n)$ funktioniert, aber die Teilung der Punktmenge ist vielleicht nicht erreicht:



In [Klein] wird in der Lösung zur Übungsaufgabe 3.11 beschrieben, das eine Menge von Punkten des $D \subset \mathbb{R}^k$ *immer* durch eine Hyperebene so in 2 Teile zerlegt werden kann, dass kein Punkt auf der Hyperebene liegt und jeder Teil mindestens $(n-1)/2k$ Punkte enthält. Für $k=2$ und $n>5$ ist $(n-1)/4 > n/5$ aller Punkte im kleinsten Teil. Für eine Aufteilungstechnik in $O(n\ \ell og\ n)$ Zeit sei auf die Literatur verwiesen. Die zuerst versuchte Rekursion $T(n)=2\cdot T(4/5\cdot n)+b\cdot n$, führt mit der allgemeinen Lösung leider NICHT zum schnellen Erfolg:

Berechnung der Rekursion

Die Rekursionsgleichung, in der beide Teile rekursiv mit dem gleichen Verfahren zu bearbeiten wären, möge Versuchsweise

$$T(n) = 2 \cdot T(4/5 \cdot n) + b \cdot n,$$

sein. In ihr ist b das Maximum von der Bearbeitungszeit $T(1)$ und der Zeit für das Zerteilen und Zusammensetzen. Aber der Faktor 2 ist hier falsch, weil der andere Teil ja nur $1/5 \cdot n$ groß ist! Bei Zweiteilung würde die Rekursion $2 \cdot T(4/5 \cdot n) + b \cdot n$ benutzt werden, die auch benutzt werden kann, falls die Punkte in der Ebene durch die früher beschriebenen Techniken linear sortierbar werden: statt (x, y) wähle $(x|y,\ y|x)$.

Um den lästigen Bruch $4/5$ zu ersparen setzen wir $c := 5/4$ und betrachten einmal die allgemeine Rekursionsgleichung:

$$T(n) = \begin{cases} b & \text{falls} \quad n = 1 \\ aT(\frac{n}{c}) + bn & \text{falls} \quad n > 1 \end{cases}$$

Analyse der Rekursion: 1. “Abwickeln”

Durch durch wiederholtes Einsetzen – “Abwickeln” der Rekursion – erhalten wir zunächst:

$$
\begin{aligned}
T(n) &= aT\left(\frac{n}{c}\right) + bn = a\left(aT\left(\frac{n}{c^2}\right) + b\frac{n}{c}\right) + bn \\
&= a^2T\left(\frac{n}{c^2}\right) + bn\frac{a}{c} + bn \\
&= a^2\left(aT\left(\frac{n}{c^3}\right) + b\frac{n}{c^2}\right) + bn\frac{a}{c} + bn \\
&= a^3T\left(\frac{n}{c^3}\right) + bn\left(\frac{a}{c}\right)^2 + bn\frac{a}{c} + bn \\
&= a^kT\left(\frac{n}{c^k}\right) + bn \cdot \sum_{i=0}^{k-1}\left(\frac{a}{c}\right)^i
\end{aligned}
$$

Wenn $n = c^k$ ist, so bricht das Verfahren bei $T(n) = a^kb + bn\sum_{i=0}^{k-1}(\frac{a}{c})^i$ ab, denn es ist $T(1) = b$.

Mit $n = c^k$ folgt $k = \log_c(n)$ und somit

$$\begin{aligned} T(n) &= a^k b + bn \sum_{i=0}^{k-1} \left(\frac{a}{c}\right)^i \\ &= a^k b + bn \sum_{i=0}^{k} \left(\frac{a}{c}\right)^i - bn\left(\frac{a}{c}\right)^k \\ &= a^k b\left(1 - \frac{n}{c^k}\right) + bn \sum_{i=0}^{k} \left(\frac{a}{c}\right)^i \\ &= bn \sum_{i=0}^{k} \left(\frac{a}{c}\right)^i \end{aligned}$$

Das Ergebnis der allgemeinen Rekursionsgleichung:

Je nach dem Verhältnis von a zu c ergeben sich unterschiedliche Lösungen:

Fall 1, $a < c$**:** Die Reihe $\sum_{i=0}^{\infty} \left(\frac{a}{c}\right)^i$ konvergiert, so dass $T(n)$ proportional zu n ist.

Fall 2, $a = c$**:** $T(n) = bn \log_c(n)$ ist „proportional“ zu $n \log(n)$.

Fall 3, $a > c$**:** $T(n)$ ist „proportional“ zu $n^{\log_c(a)}$.

Dann folgt

$$
\begin{aligned}
T(n) &= bn \sum_{i=0}^{\log_c n} \left(\frac{a}{c}\right)^i \\
&= bn \frac{\left(\frac{a}{c}\right)^{\log_c n+1} - 1}{\frac{a}{c} - 1} \\
&\approx bn \left(\frac{a}{c}\right)^{\log_c n} \\
&= bn \frac{a^{\log_c n}}{n} \\
&= ba^{\log_c n} = bn^{\log_c a}
\end{aligned}
$$

weil $a^{\log_c n} = n^{\log_c a}$ stets gilt.

Anwendung im *divide and conquer* Verfahren hier möglich?

Mit der Rekursion $T(n) = 2 \cdot T(n/2) + b \cdot n$ ergibt sich nach der allgemeinen Lösung der Rekursion also der Zeitbedarf:

$$O(n \ \ell og \ n).$$

Leider benötigt man hier also eine Codierung der Punkte in der Ebene. so dass eine Zweiteilung (genauer: Medianbildung) möglich ist.

Die in [Klein] beschriebene Methode teilte die Punktemenge aber in $4/5$ und $1/5$.

Anwendung im *divide and conquer* Verfahren hier möglich?

Es ergibt sich bei der falschen Rekursion: $T(n)=2\cdot T(4/5\cdot n)+b\cdot n,$ mit

$c=5/4$ dann: $\log_c(2) = \dfrac{\log_2(2)}{\log_2(c)} = \dfrac{1}{\log_2(c)}$

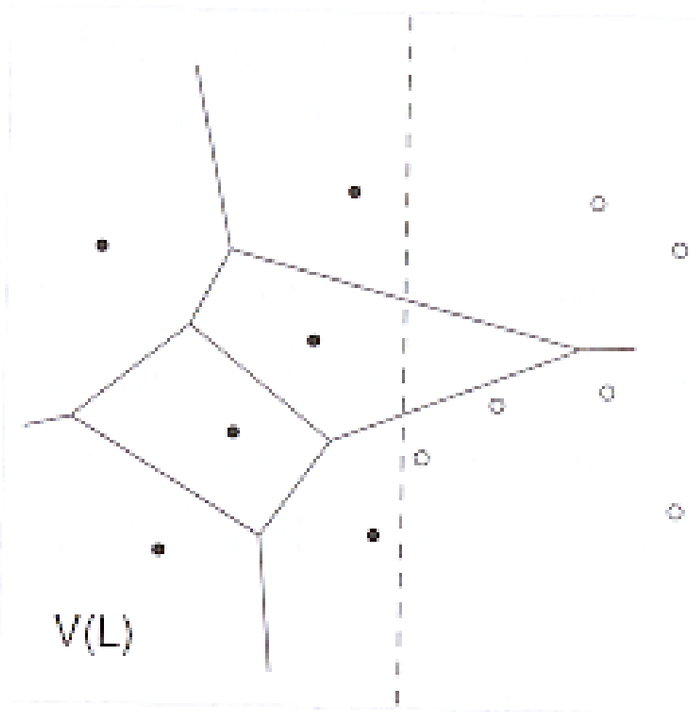
$$\log_2(\frac{5}{4}) = \log_2(5) - \log_2(4) = \log_2(5) - 2 < 0,322$$

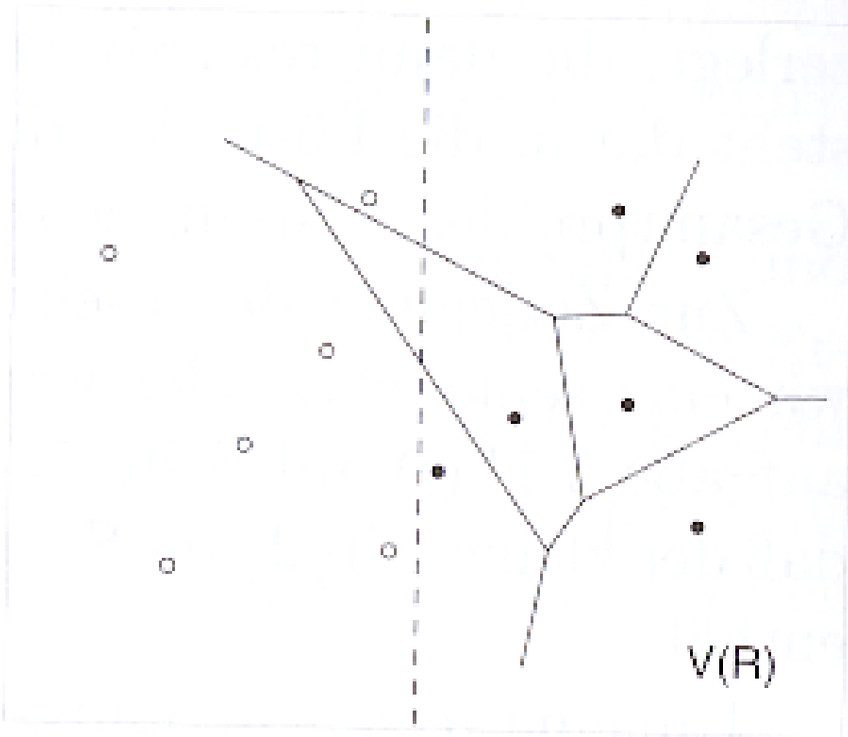
Damit ist dann aber $\log_c(2) > 1/0{,}322 > 3$ und leider

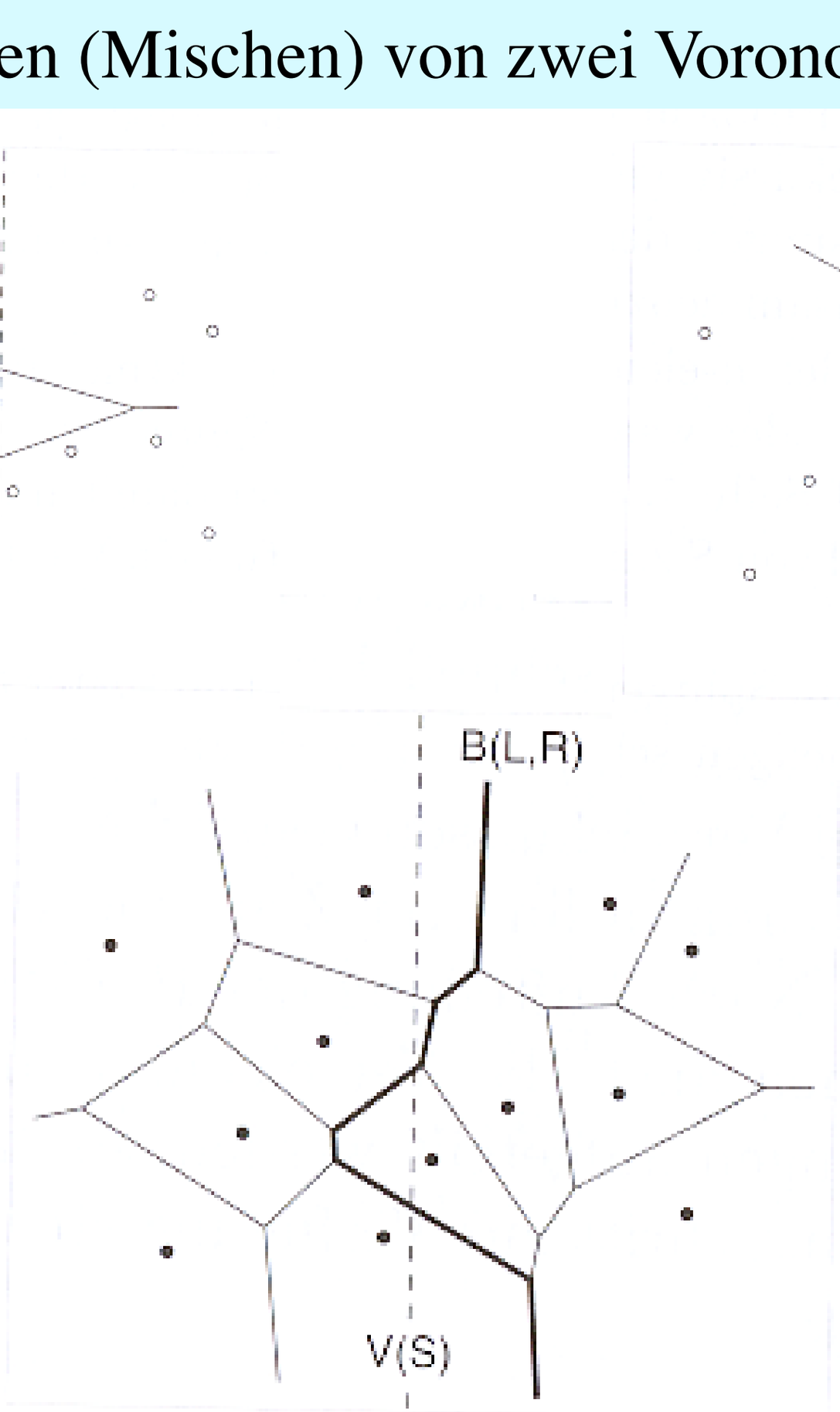
$$T(n) = b \cdot n^{\log_c(2)} \notin O(n \cdot \log n)$$

Wir retten das auf "direkte Weise": Der Baum aller Rekursionsaufrufe hat im schlimmsten Fall, d.h. ungünstigste Aufteilung mit stets $4/5$ und $1/5$ aller Knoten, maximale Tiefe von $O(\ell og\ n)$, weil für $j \geq \log_{\frac{5}{4}}(n)$ gerade $(\frac{5}{4})^j \cdot n \leq 1$ gilt. In jeder Schicht finden Teilungen und Zusammensetzungen in insgesamt $O(n)$ Zeitaufwand statt. (Hier ist n die Gesamtzahl aller Punkte!) Damit ist der Gesamtaufwand tatsächlich $O(n\ \ell og\ n)$ Zeit bei $O(n)$ Platzbedarf, sofern das Zusammenfügen auch in $O(n)$ möglich ist, was noch zu zeigen wäre:

Zusammenfügen (Mischen) von zwei Voronoi-Diagrammen







Aufwand fürs Mischen

Im zusammengefügten $V(S)$ gibt es Kanten,
die nur zwischen Punkten der Region V(L) oder
die nur zwischen Punkten der Region V(R) verlaufen.
Diese Kanten sind auch alle im Zusammengefügten V(S) enthalten!
(wenn auch dort eventuell kürzer).

Das Problem sind jene Kanten zwischen
einem Punkt in V(L) und einem in V(R).

Dieser Bisektor $B(L,R)$ zwischen den beiden Gebieten ist eine monotone Kette von Geradenstücken von oben nach unten, die wir konstruiern müssen, um :

1. Die Diagramme (VL) und $V(R)$ entlang $B(L,R)$ aufzuschneiden,
2. Beide Teile links und rechts von $B(L,R)$ zusammenzusetzen.

Aufwand fürs Mischen

Etwas kompliziert [vergl. Klein] kann gezeigt werden, dass der Kantenzug $B(L,R)$ in $O(n)$ Zeit gefunden werden kann, indem man ein Anfangsstück sucht (nicht sehr übersichtliches Vorgehen) und sich dann von dort aus über die vorhandenen Bisektorenstücke in $V(L)$ und $V(R)$ weiterhangelt.

Tatsächlich ist insgesamt der Aufwand

$$O(n \log n)$$

zur Konstruktion des Voronoi-Diagramms

auch mit diesem Verfahren zu erreichen!