Karlsruher Institut für Technologie

Algorithmische Kartografie
Übung am 02.05.2013

INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER

Linievereinfachung als Optimierungsproblem

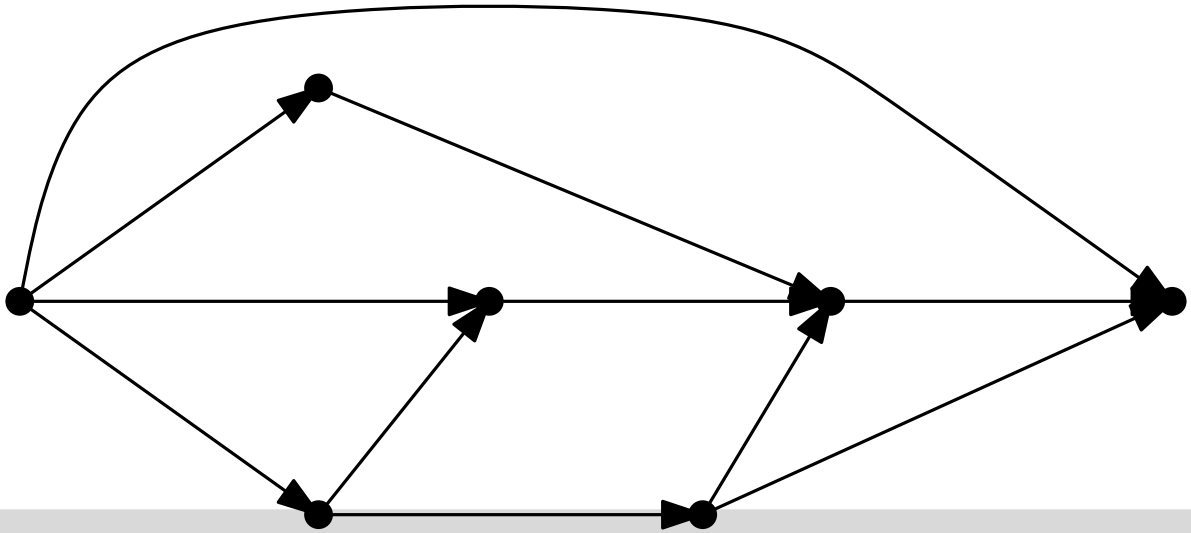
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Vereinfachung als Optimierungsproblem

Douglas-Peucker-Algorithmus liefert nur heurisitsch gute Approximationen an die Eingabe, jedoch nicht notwendig eine optimale Lösung.

Formulierung als Graphenproblem:

- vollständiger, gewichteter DAG G auf $\{v_1, \ldots, v_n\}$ mit Kante (v_i, v_j) für alle $i < j$
- Kosten c_{ij} für das Ersetzen des Teilpfads $(v_i, v_{i+1}, \ldots, v_j)$ durch Kante (v_i, v_j)
- finde kostenminimalen (kürzesten) Weg von v_1 nach v_n in G mit maximal k Kanten



Aufgabe 2:

Gegeben sei ein azyklischer, gerichteter Graph $G = (V, E)$ mit Gewichtsfunktion $w\colon E \to \mathbb{R}$. Ein gerichteter s, t-Pfad $(s = v_0, \ldots, v_\ell = t)$ in G heißt *k-minimal*, falls er nicht mehr als k Kanten enthält und für alle anderen s, t-Pfade $(s = v'_0, \ldots, v'_{\ell'} = t)$ mit höchsten k Kanten gilt:

$$\sum_{j=1}^{\ell} w((v_{j-1}, v_j)) \leq \sum_{j=1}^{\ell'} w((v'_{j-1}, v'_j))$$

1. Geben Sie einen Algorithmus an, der einen k-minimalen Pfad $(s = v_0, \ldots, v_n = t)$ zwischen zwei gegebenen Knoten $s, t \in V$ berechnet.
2. Welche Laufzeit und welchen Speicherplatzbedarf hat der von Ihnen vorgeschlagene Algorithmus?
3. Begründen Sie die Korrektheit Ihres Algorithmus.

Algorithmus von de Berg et al. (1998)

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Algorithmus von de Berg et al. (1998)

Idee:

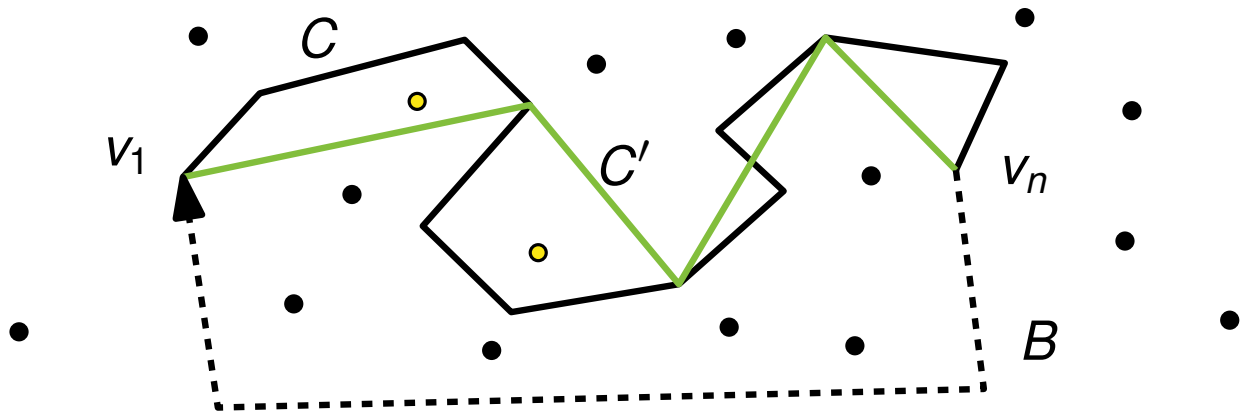
- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** shortcuts bzgl. Fehler ε bilden Graph G_1
- **konsistente** shortcuts bzgl. Punktmenge P bilden Graph G_2
- kürzester Weg in Graph $G = (V, A_1 \cap A_2)$ liefert optimale zulässige und konsistente Vereinfachung

Algorithmus von de Berg et al. (1998)

Idee:

- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** shortcuts bzgl. Fehler ε bilden Graph G_1
- **konsistente** shortcuts bzgl. Punktmenge P bilden Graph G_2
- kürzester Weg in Graph $G = (V, A_1 \cap A_2)$ liefert optimale zulässige und konsistente Vereinfachung

Def: Kantenzug $C = (v_1, \ldots, v_n)$ und Vereinfachung C' heißen **konsistent bzgl.** P, wenn es Kantenzug B von v_n nach v_1 gibt, der C und C' zu einfachen Polygonen abschließt, die die gleiche Teilmenge von P einschließen.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Algorithmus von de Berg et al. (1998)

Idee:

- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** shortcuts bzgl. Fehler ε bilden Graph G_1
- **konsistente** shortcuts bzgl. Punktmenge P bilden Graph G_2
- kürzester Weg in Graph $G = (V, A_1 \cap A_2)$ liefert optimale zulässige und konsistente Vereinfachung

Def: Kantenzug $C = (v_1, \ldots, v_n)$ und Vereinfachung C' heißen **konsistent bzgl.** P, wenn es Kantenzug B von v_n nach v_1 gibt, der C und C' zu einfachen Polygonen abschließt, die die gleiche Teilmenge von P einschließen.

Zunächst: Berechnung von G_2 für x-monotone Kantenzüge

x-monotone Kantenzüge

Bestimmung aller konsistenten shortcuts von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

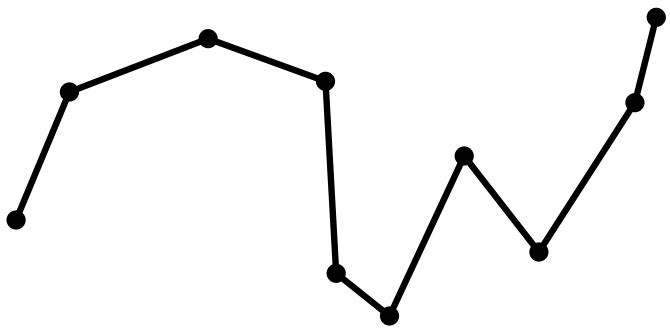
x-monotone Kantenzüge

Bestimmung aller konsistenten shortcuts von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i
2. verteile P auf Regionen von S_i
3. berechne konsistente shortcuts



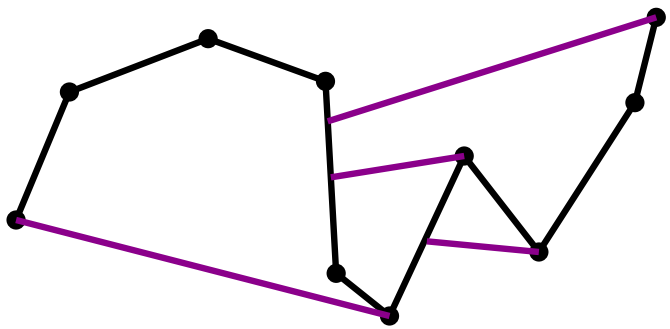
x-monotone Kantenzüge

Bestimmung aller konsistenten shortcuts von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i
2. verteile P auf Regionen von S_i
3. berechne konsistente shortcuts



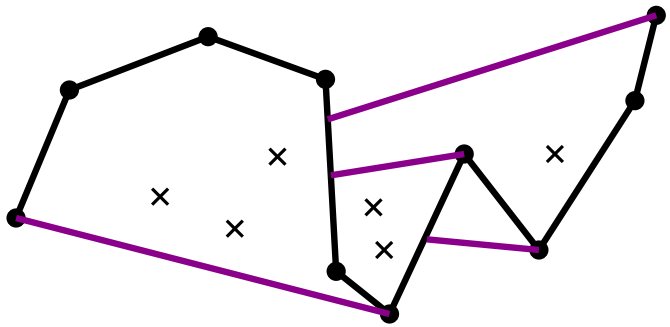
x-monotone Kantenzüge

Bestimmung aller konsistenten shortcuts von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i
2. verteile P auf Regionen von S_i
3. berechne konsistente shortcuts



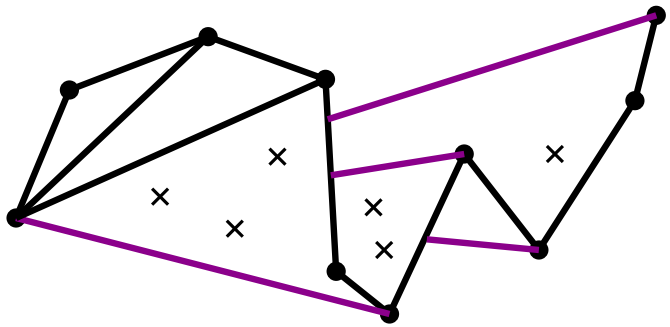
x-monotone Kantenzüge

Bestimmung aller konsistenten shortcuts von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i
2. verteile P auf Regionen von S_i
3. berechne konsistente shortcuts



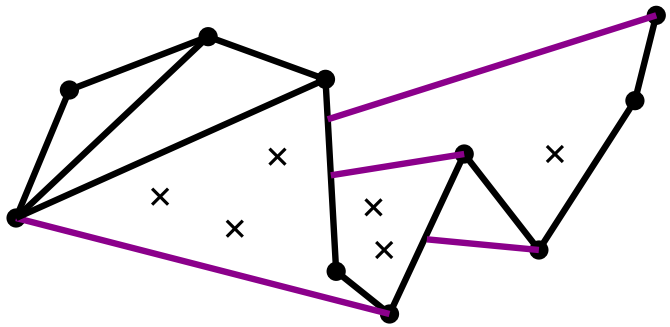
x-monotone Kantenzüge

Bestimmung aller konsistenten shortcuts von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i
2. verteile P auf Regionen von S_i
3. berechne konsistente shortcuts



2) Punktzuweisung: Algorithmus

Punktzuweisung(S_i, i, P)

Input: Unterteilung S_i, Index $1 \leq i \leq n$, Punktmenge P

Output: Punkt p_f für jede Facette f

$P' \leftarrow$ Punkte aus P rechts von v_i

priority queue $Q \leftarrow P' \cup \{v_{i+1}, \ldots, v_n\}$ sortiert nach Winkel bzgl. v_i

$T \leftarrow$ leerer binärer Suchbaum für radialen Sweep

while *Q nicht leer* **do**
    $p \leftarrow Q$.extractMax
    if *p Knoten von* C_{ij} **then**
        update T mit Kanten von p
    else
        $e \leftarrow$ linkester Kante $v_k v_{k+1}$ in T rechts von p auf sweep line
        speichere p an e

gehe über alle Kanten $v_i v_{i+1}, \ldots, v_{n-1} v_n$ und weise Facetten f maximalen/minimalen Punkt p_f zu

Behauptung: Benötigt $O((|P| + |C|) \log |C|)$ Zeit

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

2) Punktzuweisung: Algorithmus

Punktzuweisung(S_i, i, P)

Input: Unterteilung S_i, Index $1 \leq i \leq n$, Punktmenge P

Output: Punkt p_f für jede Facette f

$P' \leftarrow$ Punkte aus P rechts von v_i

priority queue $Q \leftarrow P' \cup \{v_{i+1}, \ldots, v_n\}$ sortiert nach Winkel bzgl. v_i

$T \leftarrow$ leerer binärer Suchbaum für radialen Sweep

while *Q nicht leer* **do**

  $p \leftarrow Q$.extractMax

  if *p Knoten von* C_{ij} **then**

    update T mit Kanten von p

  else

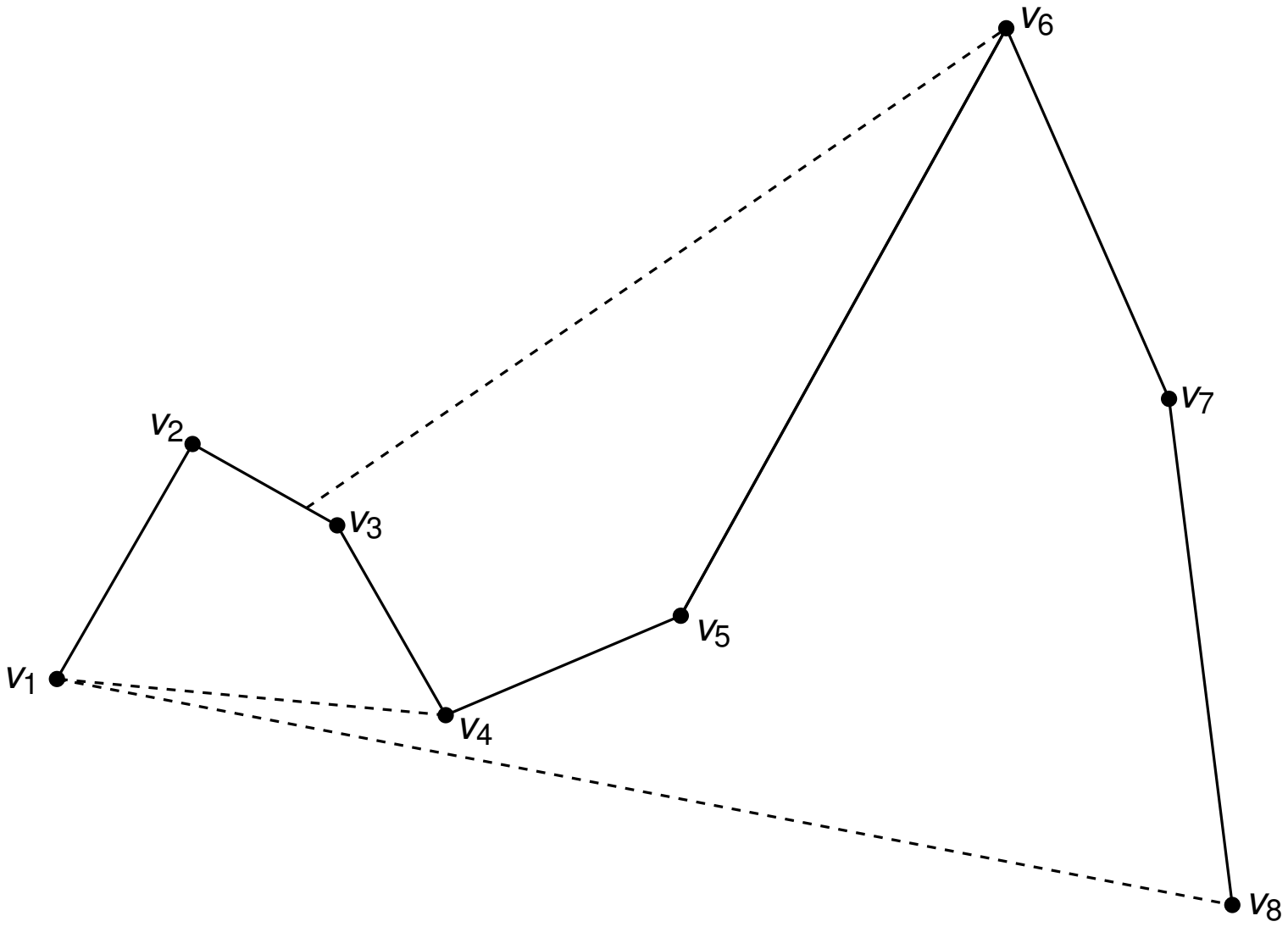
    $e \leftarrow$ linkester Kante $v_k v_{k+1}$ in T rechts von p auf sweep line

    speichere p an e

gehe über alle Kanten $v_i v_{i+1}, \ldots, v_{n-1} v_n$ und weise Facetten f maximalen/minimalen Punkt p_f zu

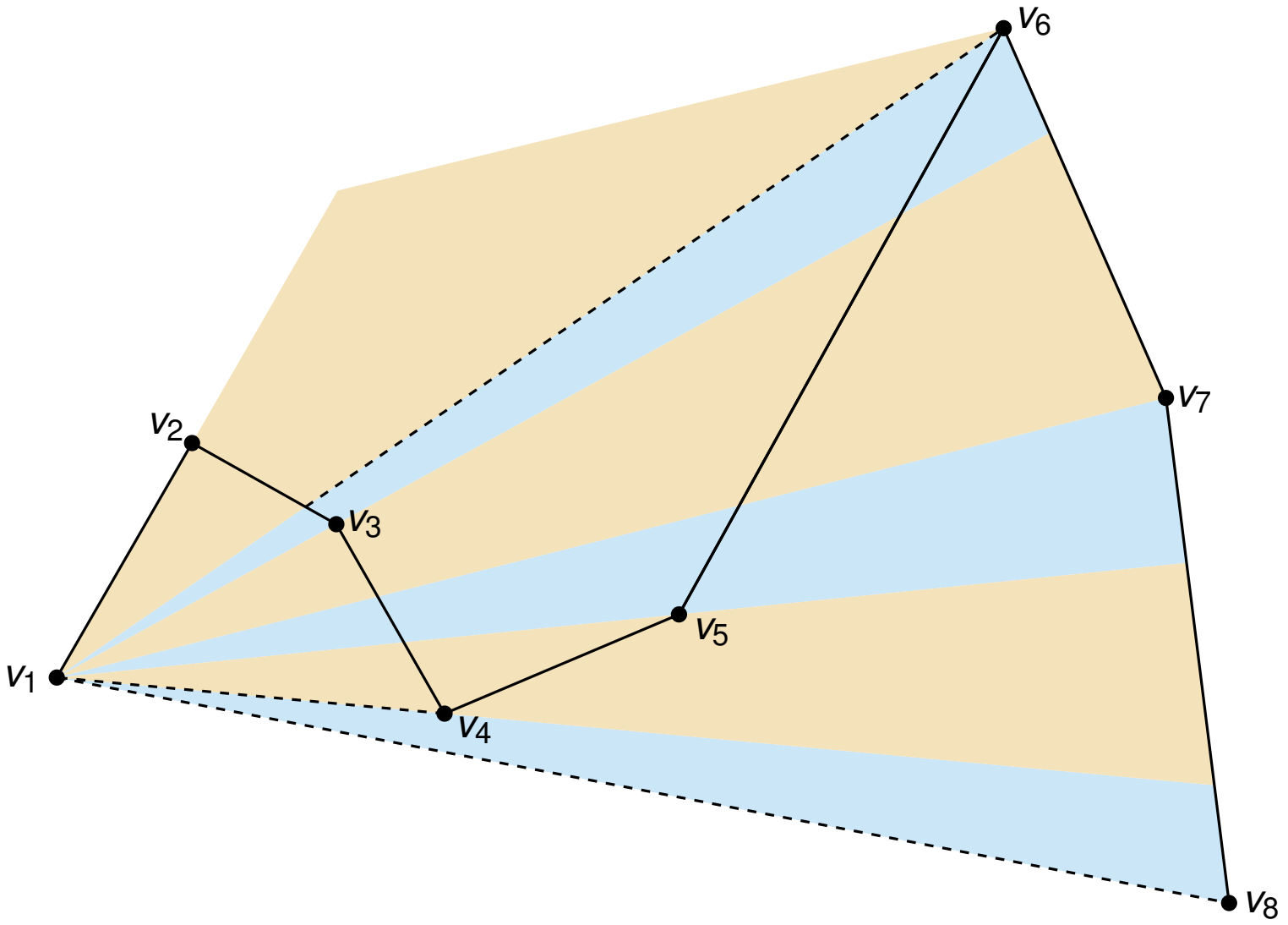
Behauptung: Benötigt $O((|P| + |C|) \log |C|)$ Zeit

Sortierung nach Winkeln



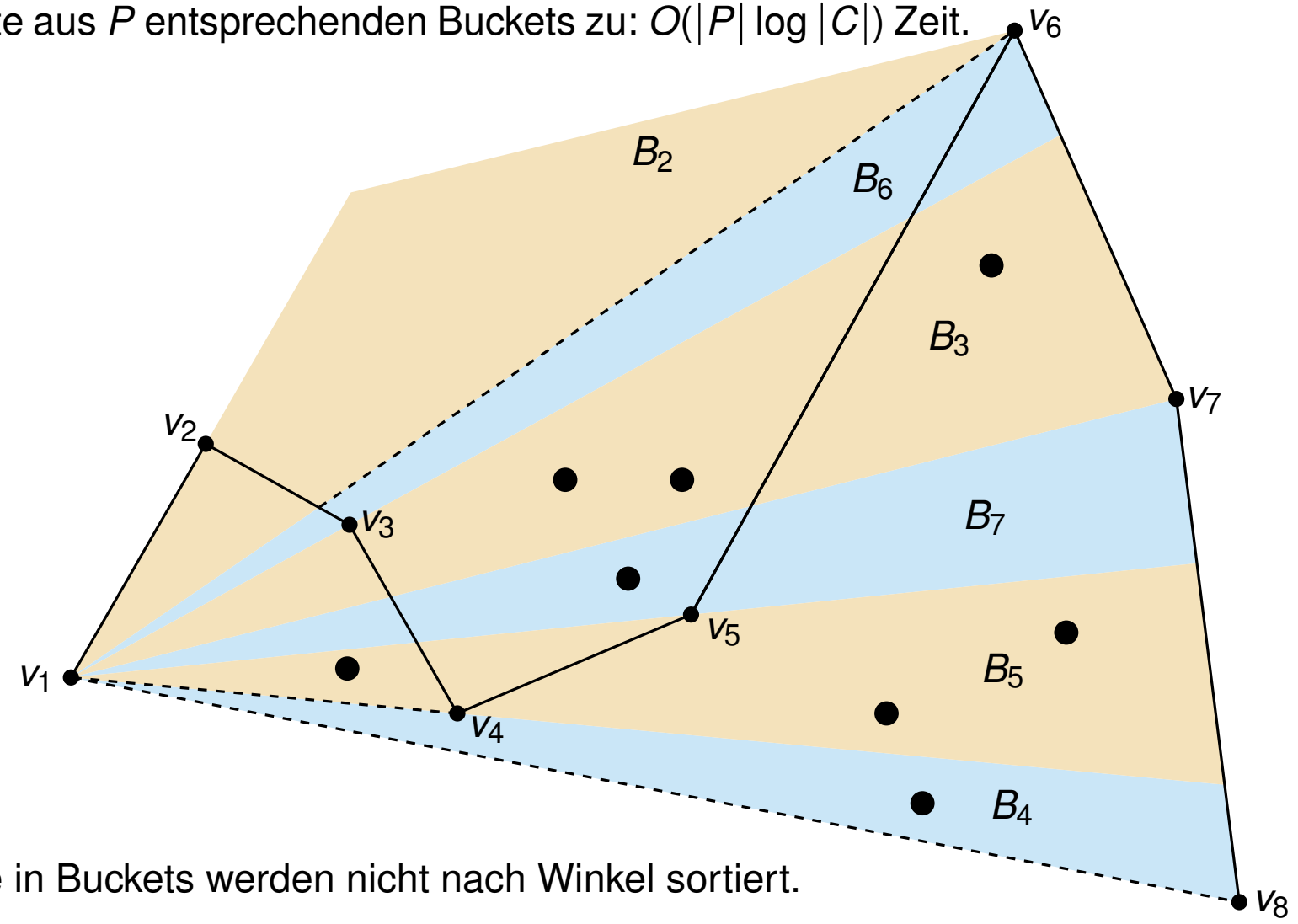
Sortierung nach Winkeln

1. Schritt: Sortiere Knoten von Polygonzug C entsprechend ihrer Winkel.
   Jeder Knoten v_i induziert einen Bucket B_i.



Sortierung nach Winkeln

1. Schritt: Sortiere Knoten von Polygonzug C entsprechend ihrer Winkel.
   Jeder Knoten v_i induziert einen Bucket B_i.
2. Schritt: Weise Punkte aus P entsprechenden Buckets zu: $O(|P| \log |C|)$ Zeit.



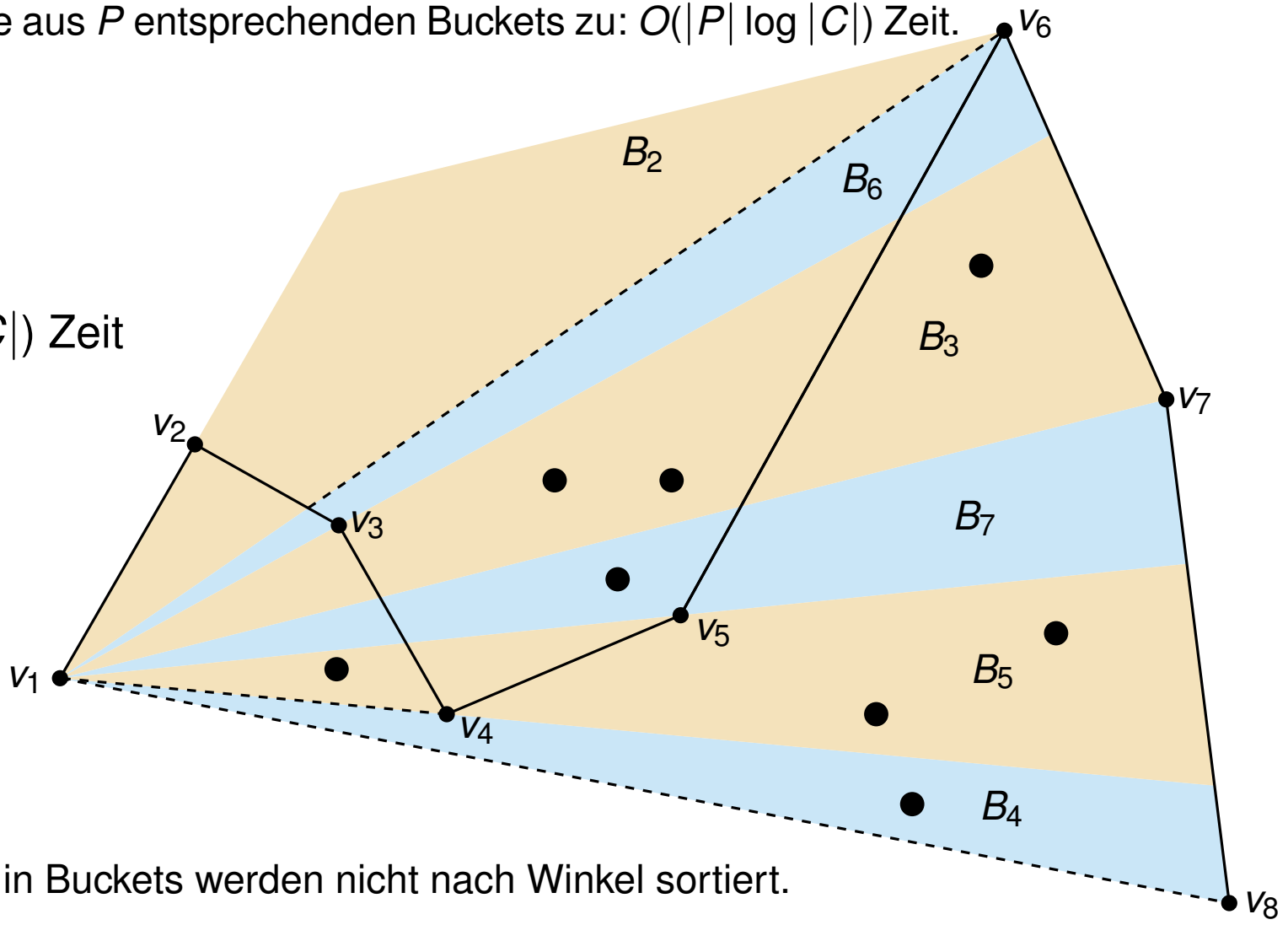
Bemerkung: Punkte in Buckets werden nicht nach Winkel sortiert.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Sortierung nach Winkeln

1. Schritt: Sortiere Knoten von Polygonzug C entsprechend ihrer Winkel.
   Jeder Knoten v_i induziert einen Bucket B_i.
2. Schritt: Weise Punkte aus P entsprechenden Buckets zu: $O(|P| \log |C|)$ Zeit.

$O((|P| + |C|) \log |C|)$ Zeit



Bemerkung: Punkte in Buckets werden nicht nach Winkel sortiert.

2) Punktzuweisung: Algorithmus

Punktzuweisung(S_i, i, P)

Input: Unterteilung S_i, Index $1 \leq i \leq n$, Punktmenge P

Output: Punkt p_f für jede Facette f

$P' \leftarrow$ Punkte aus P rechts von v_i

priority queue $Q \leftarrow P' \cup \{v_{i+1}, \ldots, v_n\}$ sortiert nach Winkel bzgl. v_i

$T \leftarrow$ leerer binärer Suchbaum für radialen Sweep

while *Q nicht leer* **do**
    $p \leftarrow Q$.extractMax
    if *p Knoten von* C_{ij} **then**
        update T mit Kanten von p
    else
        $e \leftarrow$ linkester Kante $v_k v_{k+1}$ in T rechts von p auf sweep line
        speichere p an e

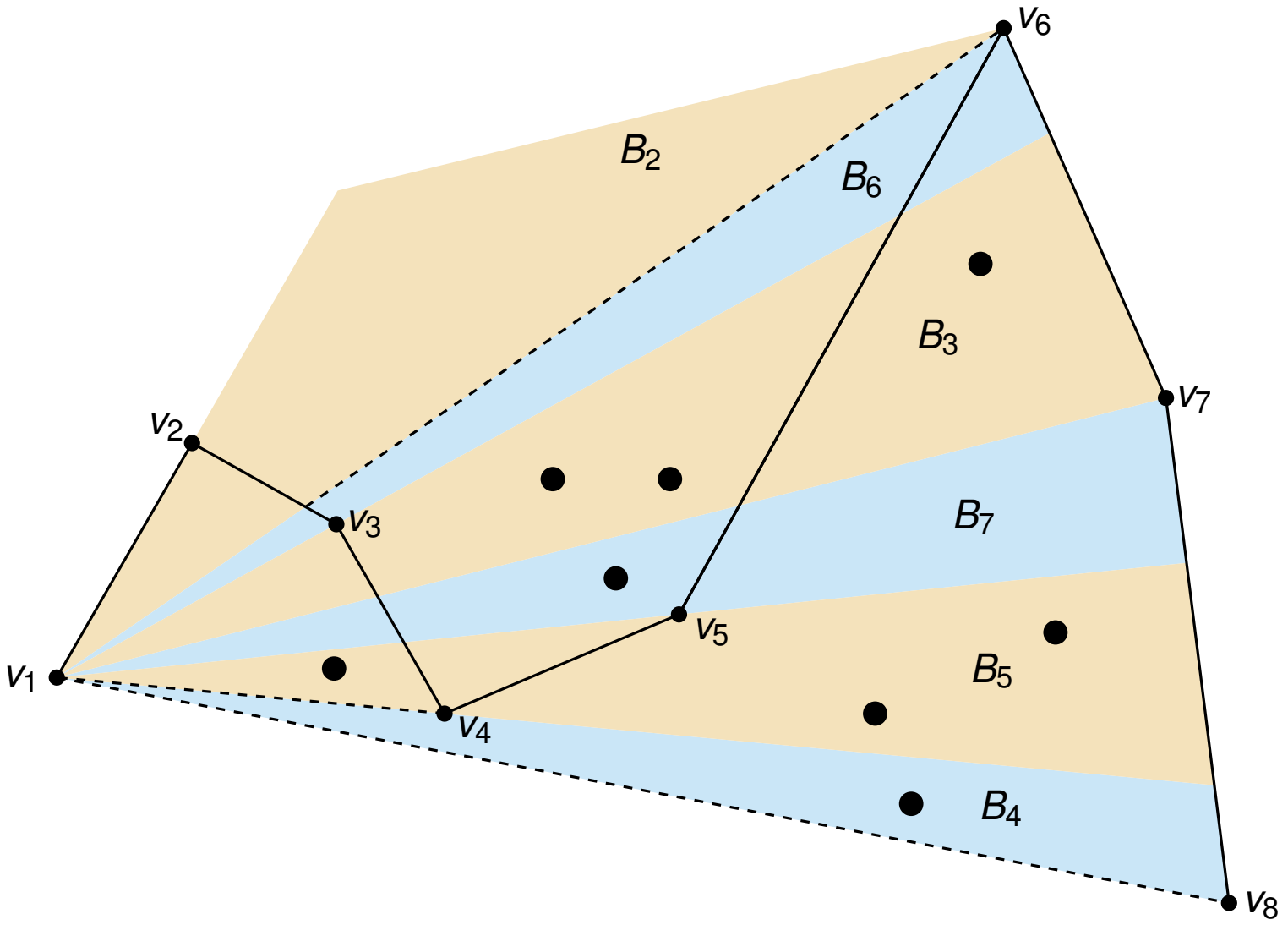
gehe über alle Kanten $v_i v_{i+1}, \ldots, v_{n-1} v_n$ und weise Facetten f maximalen/minimalen Punkt p_f zu

Behauptung: Benötigt $O((|P| + |C|) \log |C|)$ Zeit

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Sweep-Line-Verfahren

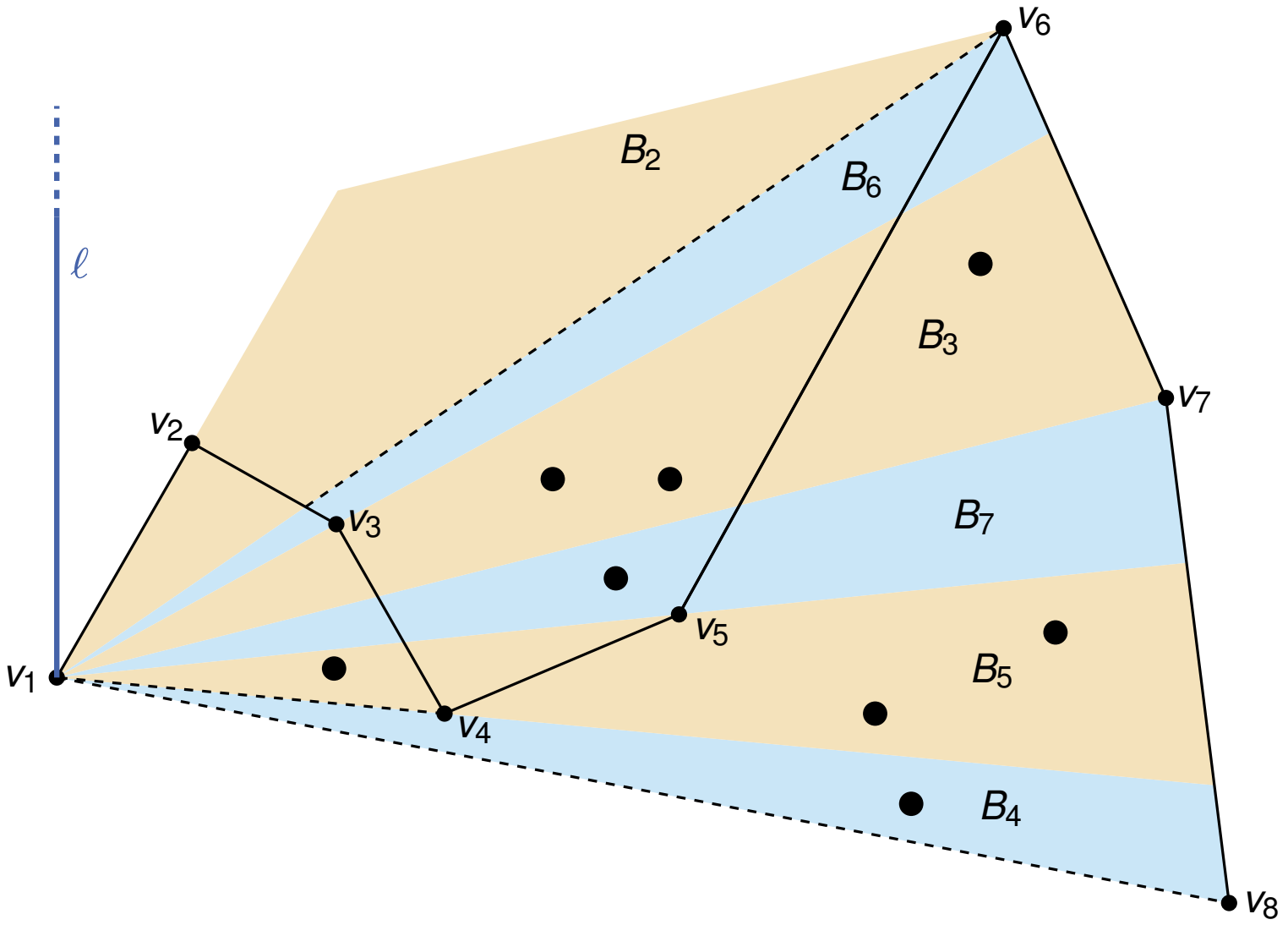
- Rotiere Sweep-Line um v_1 und stoppe bei Punkten aus P und Knoten von C
- Speichere von ℓ geschnittene Kanten in binären Suchbaum T.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Sweep-Line-Verfahren

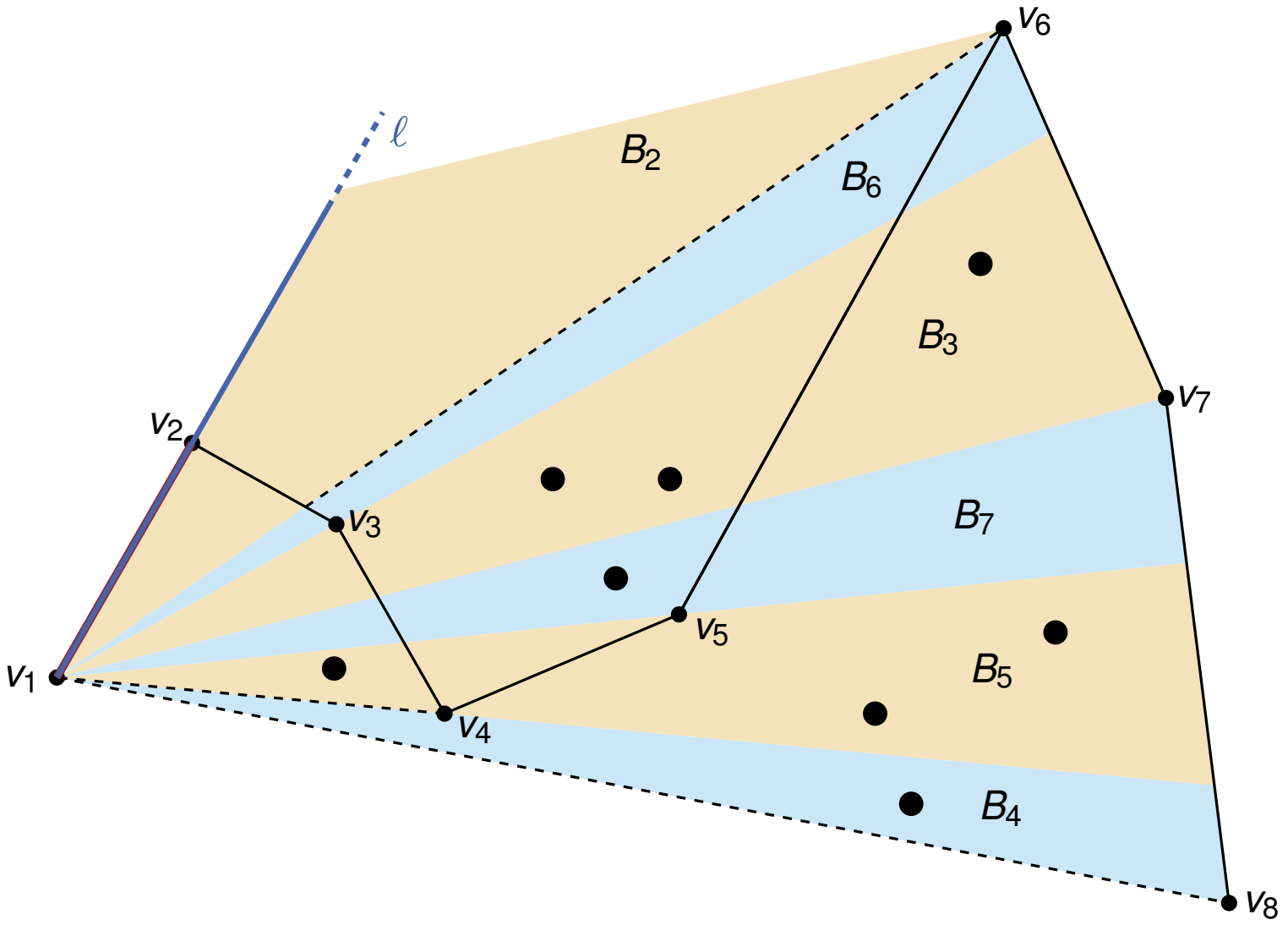
- Rotiere Sweep-Line um v_1 und stoppe bei Punkten aus P und Knoten von C
- Speichere von ℓ geschnittene Kanten in binären Suchbaum T.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

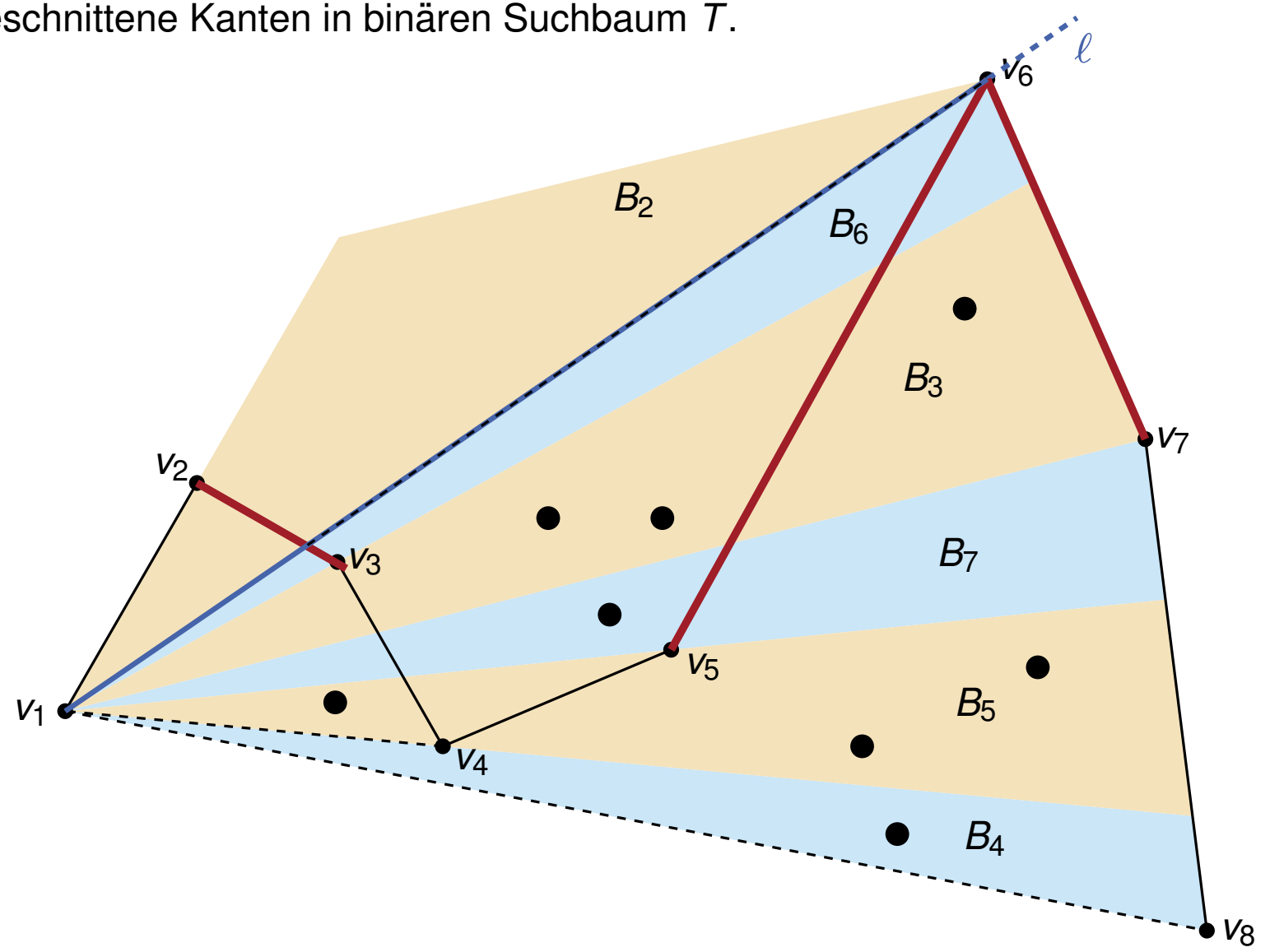
Sweep-Line-Verfahren

- Rotiere Sweep-Line um v_1 und stoppe bei Punkten aus P und Knoten von C
- Speichere von ℓ geschnittene Kanten in binären Suchbaum T.



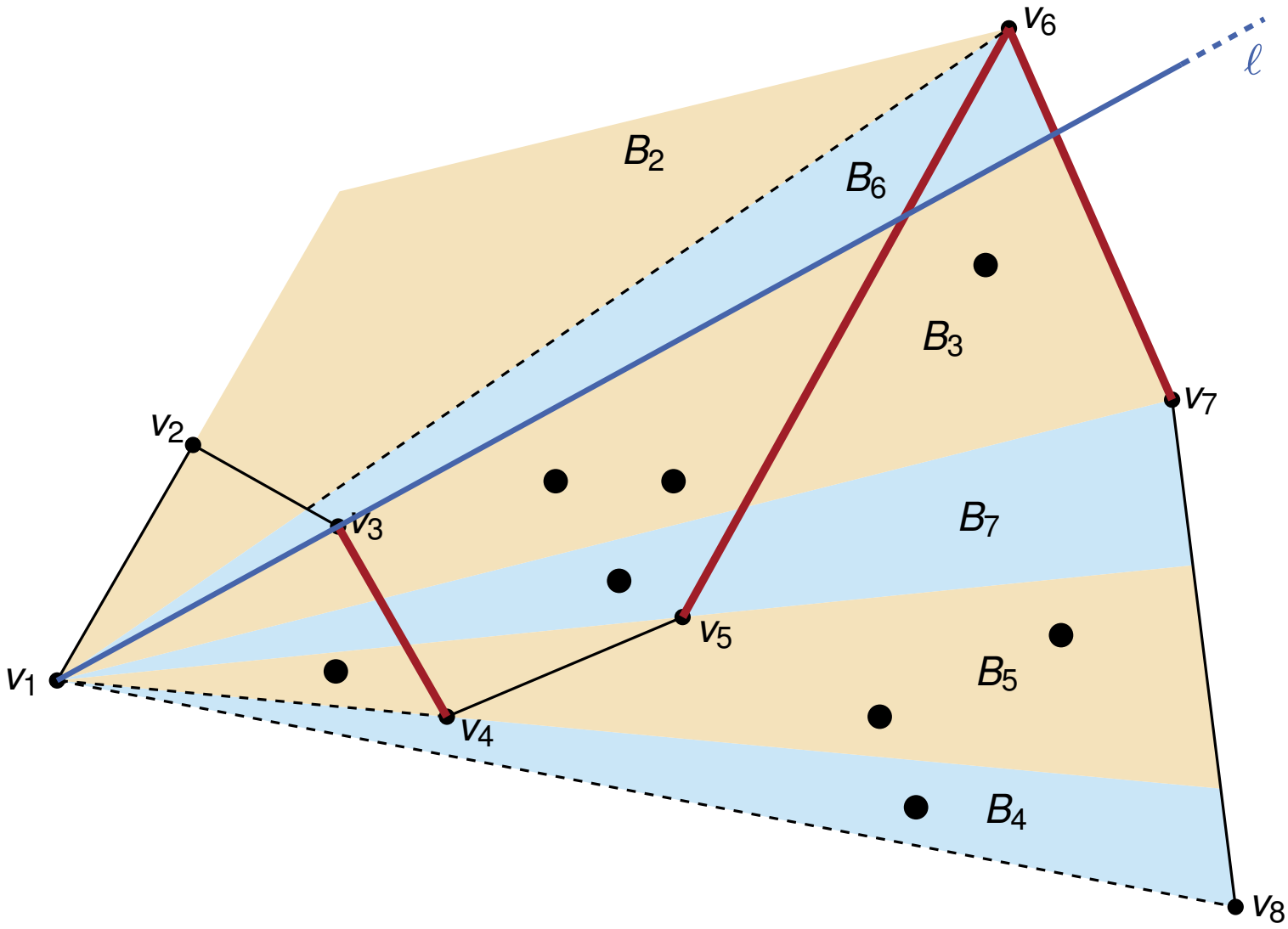
Sweep-Line-Verfahren

- Rotiere Sweep-Line um v_1 und stoppe bei Punkten aus P und Knoten von C
- Speichere von ℓ geschnittene Kanten in binären Suchbaum T.



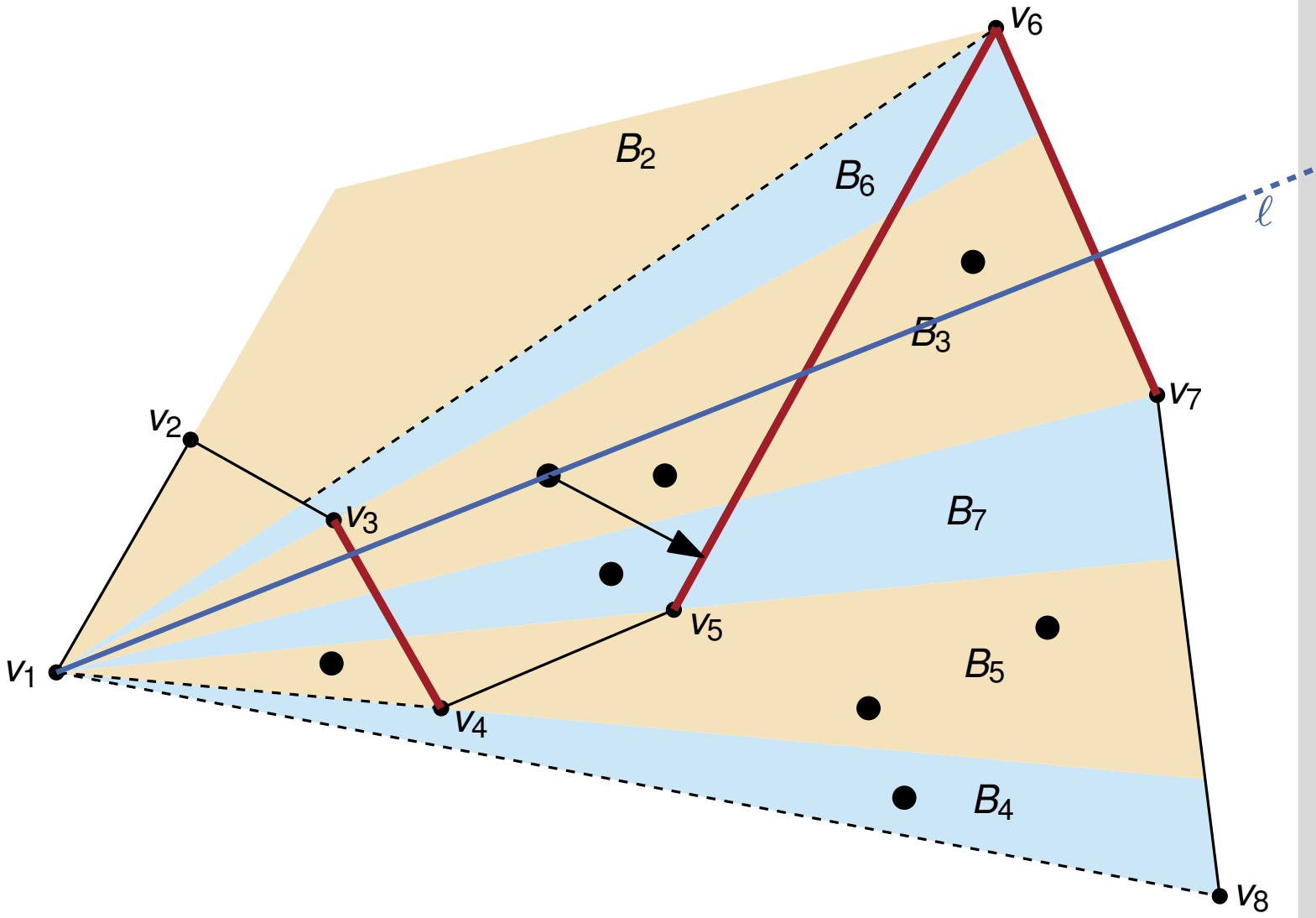
Sweep-Line-Verfahren

- Rotiere Sweep-Line um v_1 und stoppe bei Punkten aus P und Knoten von C
- Speichere von ℓ geschnittene Kanten in binären Suchbaum T.



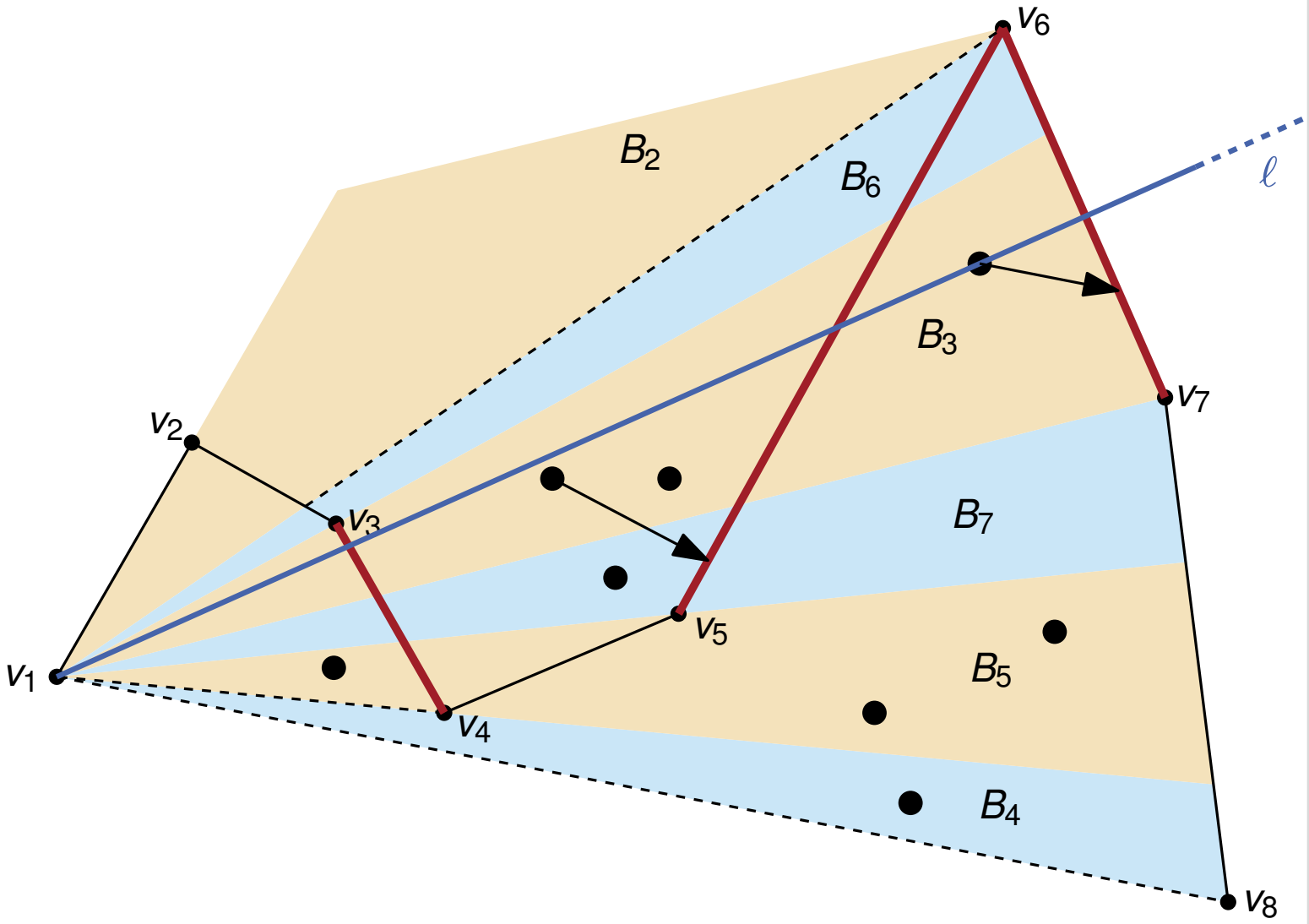
Sweep-Line-Verfahren

- Rotiere Sweep-Line um v_1 und stoppe bei Punkten aus P und Knoten von C
- Speichere von ℓ geschnittene Kanten in binären Suchbaum T.



Sweep-Line-Verfahren

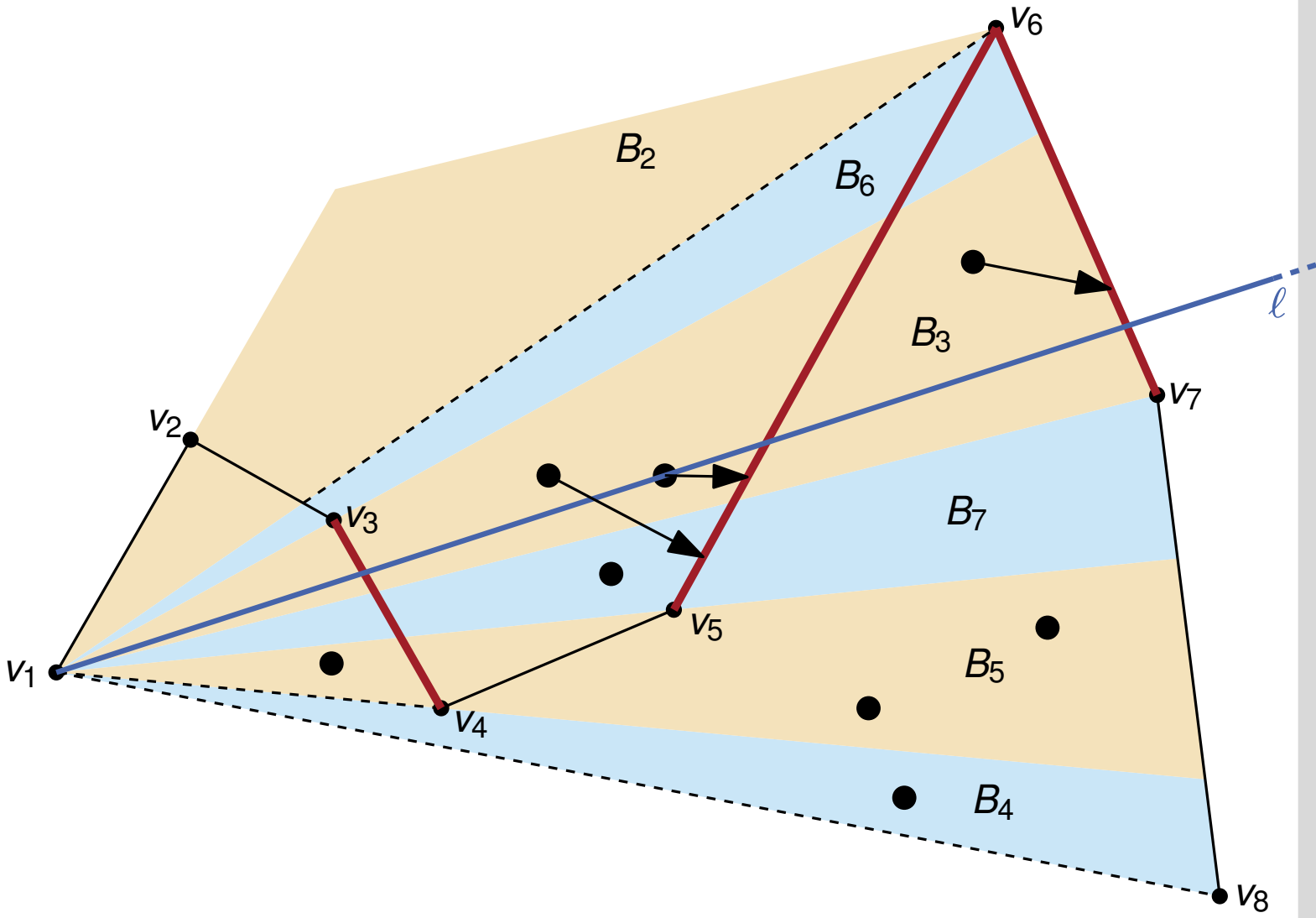
- Rotiere Sweep-Line um v_1 und stoppe bei Punkten aus P und Knoten von C
- Speichere von ℓ geschnittene Kanten in binären Suchbaum T.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

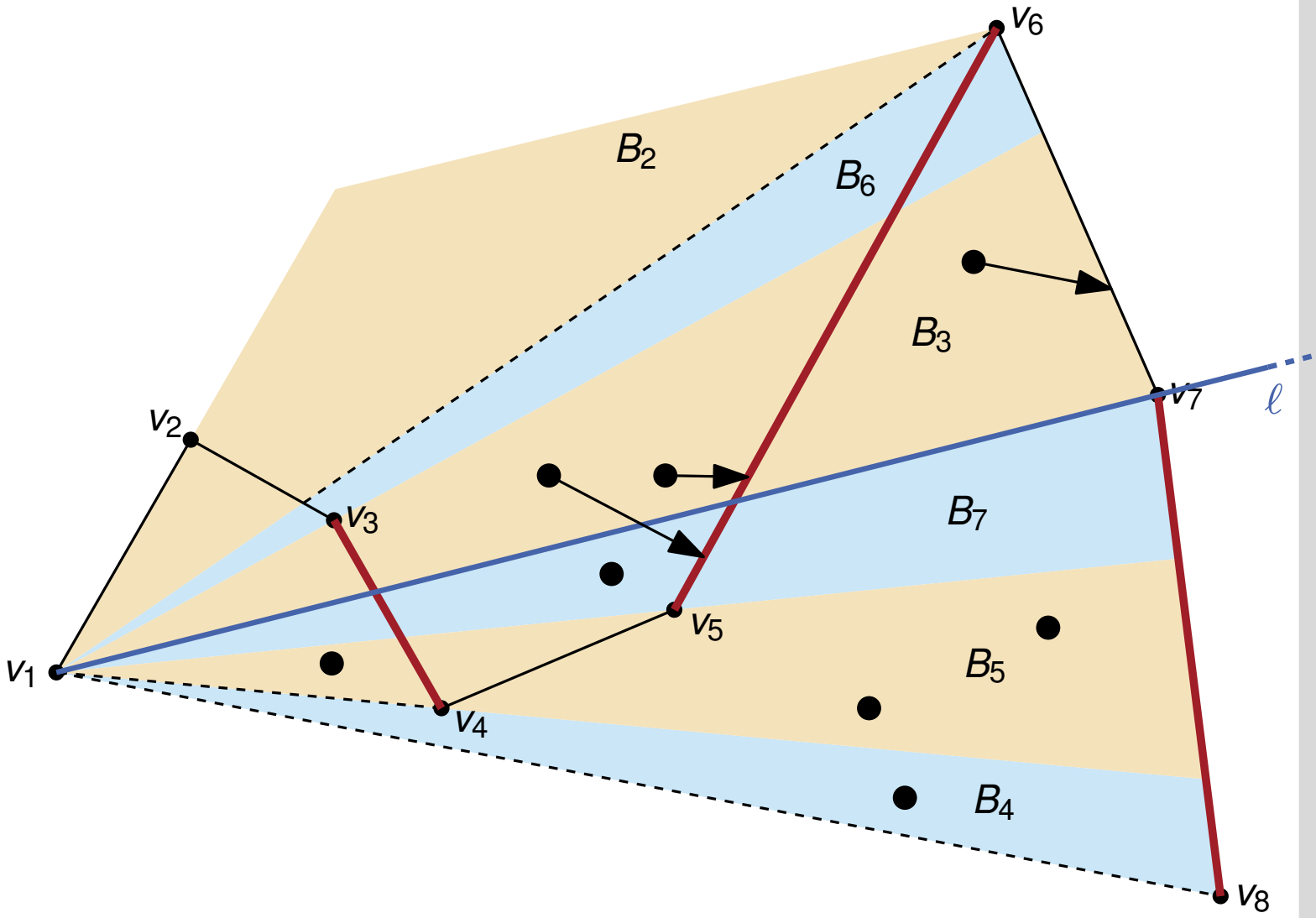
Sweep-Line-Verfahren

- Rotiere Sweep-Line um v_1 und stoppe bei Punkten aus P und Knoten von C
- Speichere von ℓ geschnittene Kanten in binären Suchbaum T.



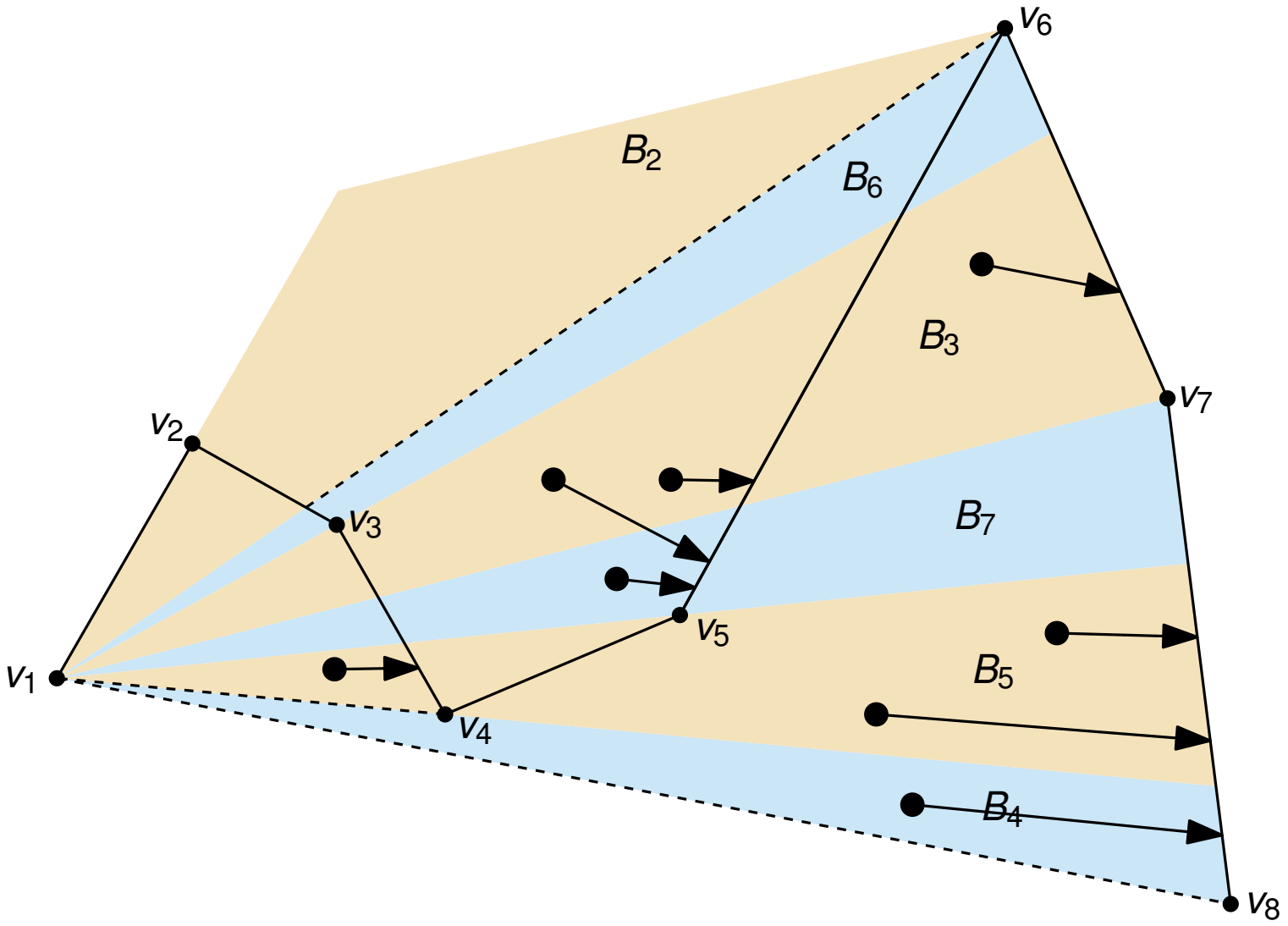
Sweep-Line-Verfahren

- Rotiere Sweep-Line um v_1 und stoppe bei Punkten aus P und Knoten von C
- Speichere von ℓ geschnittene Kanten in binären Suchbaum T.



Sweep-Line-Verfahren

- Rotiere Sweep-Line um v_1 und stoppe bei Punkten aus P und Knoten von C
- Speichere von ℓ geschnittene Kanten in binären Suchbaum T.

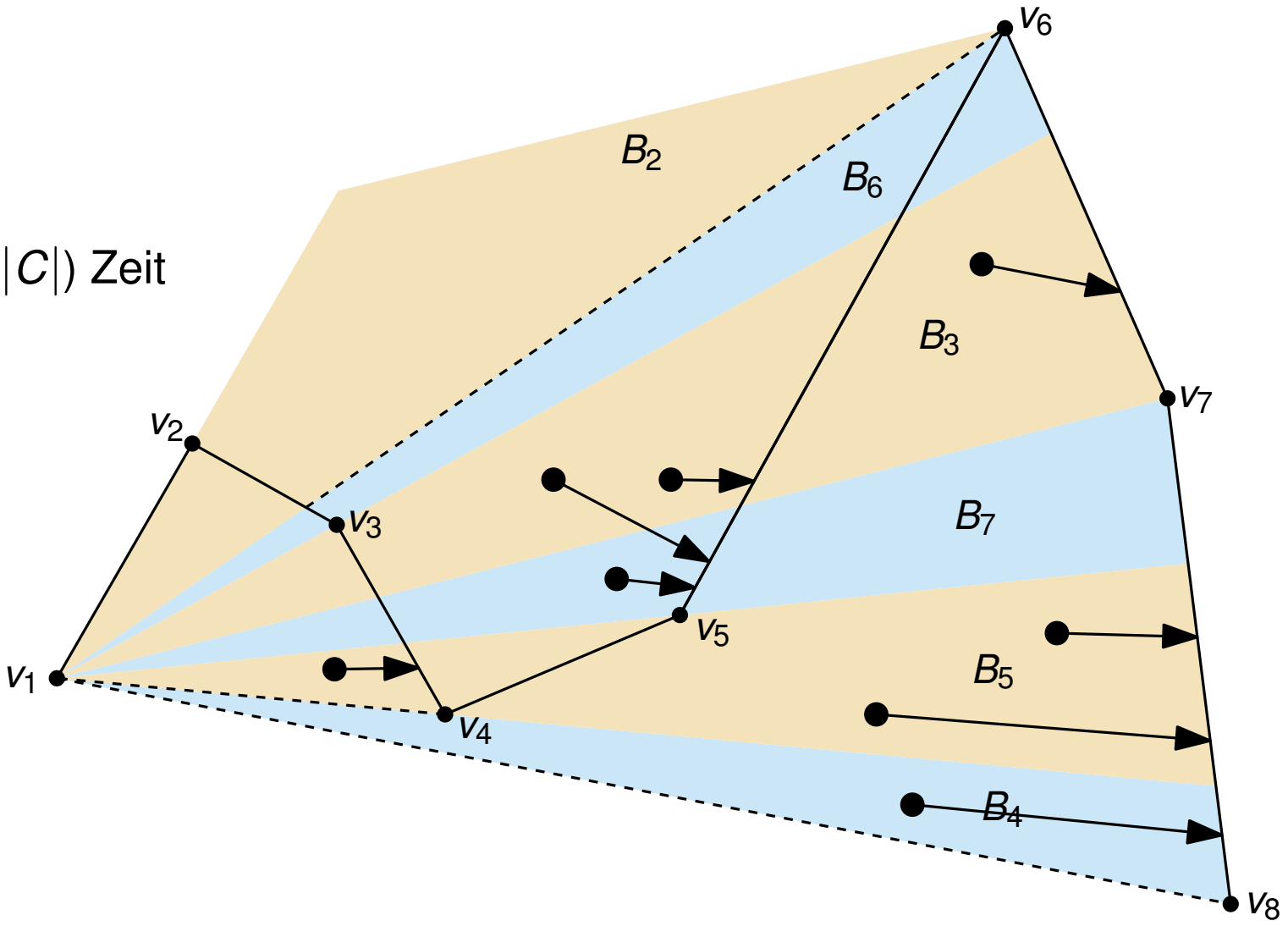


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Sweep-Line-Verfahren

- Rotiere Sweep-Line um v_1 und stoppe bei Punkten aus P und Knoten von C
- Speichere von ℓ geschnittene Kanten in binären Suchbaum T.

$O((|P| + |C|) \log |C|)$ Zeit



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

2) Punktzuweisung: Algorithmus

Punktzuweisung(S_i, i, P)

Input: Unterteilung S_i, Index $1 \leq i \leq n$, Punktmenge P

Output: Punkt p_f für jede Facette f

$P' \leftarrow$ Punkte aus P rechts von v_i

priority queue $Q \leftarrow P' \cup \{v_{i+1}, \ldots, v_n\}$ sortiert nach Winkel bzgl. v_i

$T \leftarrow$ leerer binärer Suchbaum für radialen Sweep

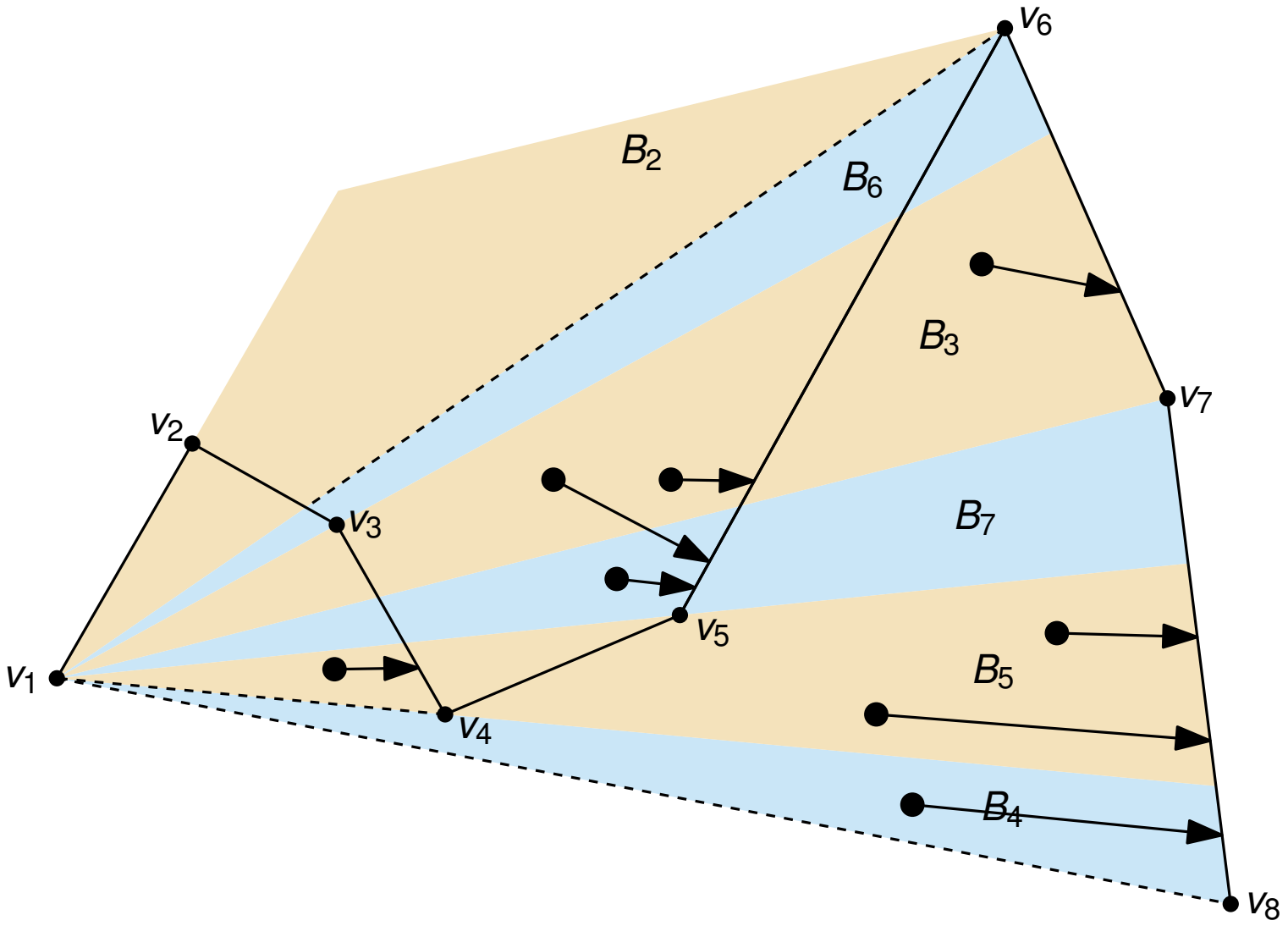
while *Q nicht leer* **do**
    $p \leftarrow Q$.extractMax
    if *p Knoten von* C_{ij} **then**
        update T mit Kanten von p
    else
        $e \leftarrow$ linkester Kante $v_k v_{k+1}$ in T rechts von p auf sweep line
        speichere p an e

gehe über alle Kanten $v_i v_{i+1}, \ldots, v_{n-1} v_n$ und weise Facetten f maximalen/minimalen Punkt p_f zu

Behauptung: Benötigt $O((|P| + |C|) \log |C|)$ Zeit

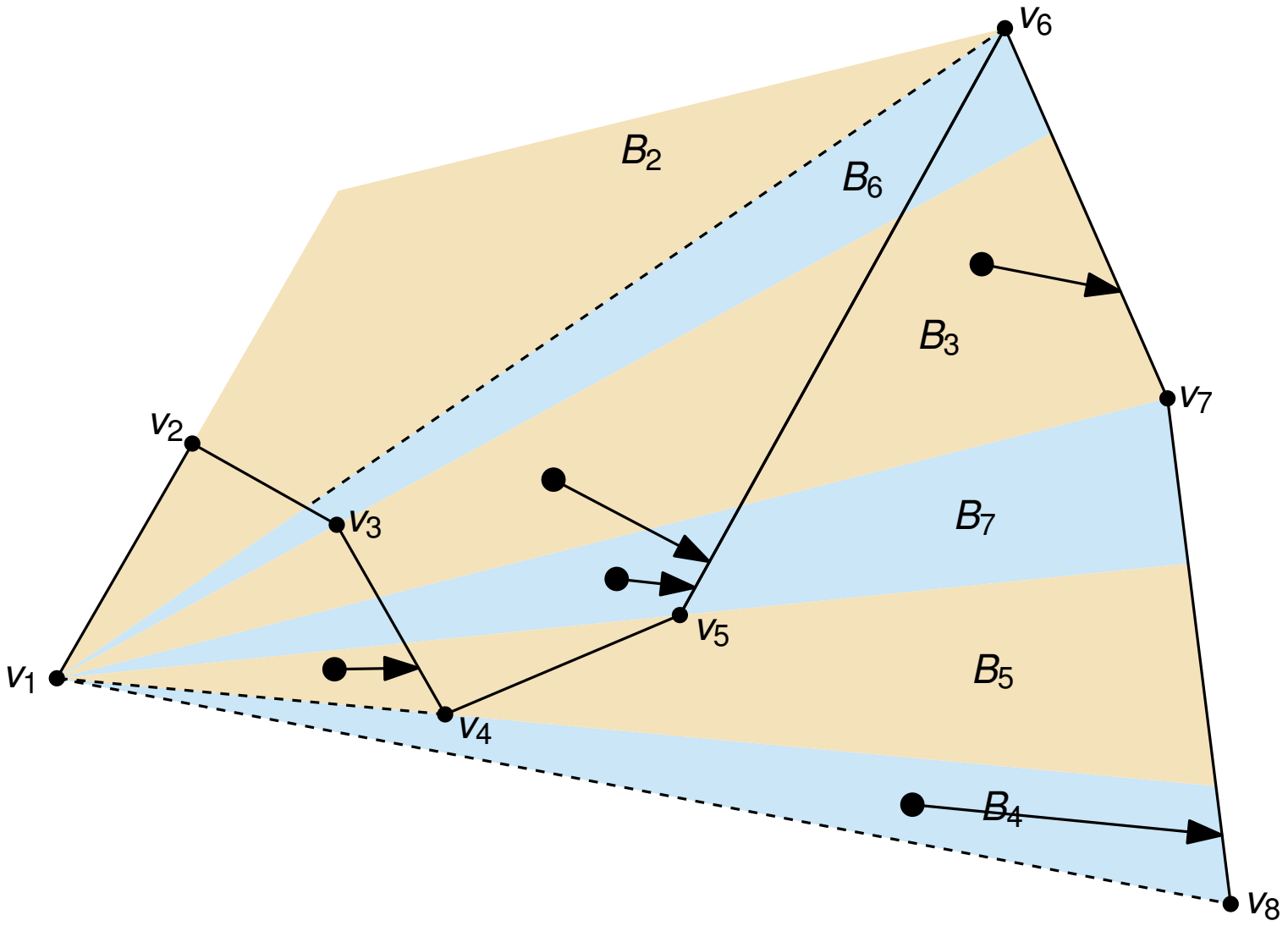
Auswahl Punkte

Bestimme für jede Facette Punkte mit maximalen/minimalen Winkel.



Auswahl Punkte

Bestimme für jede Facette Punkte mit maximalen/minimalen Winkel.

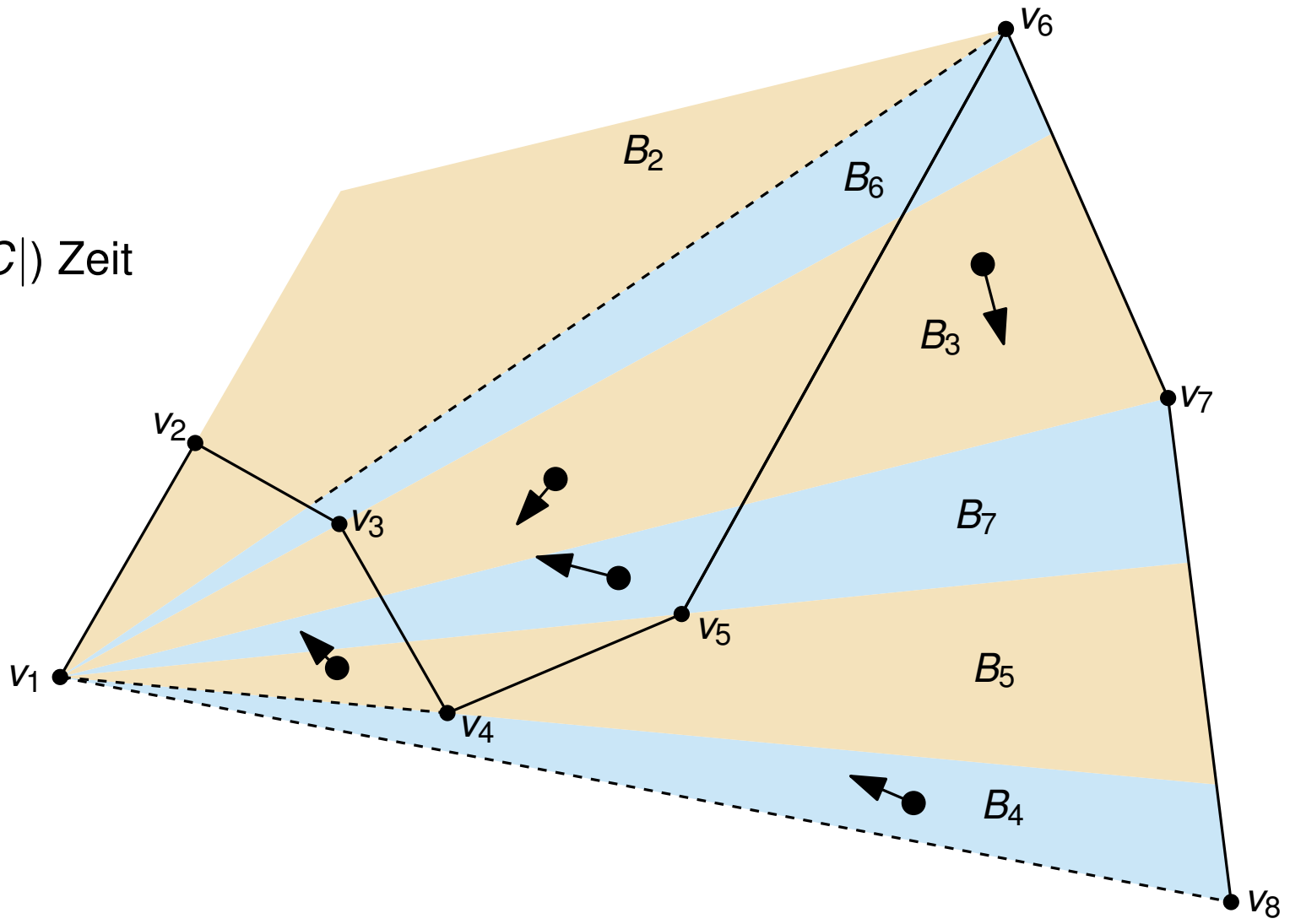


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Auswahl Punkte

Bestimme für jede Facette Punkte mit maximalen/minimalen Winkel.

$O((|P| + |C|) \log |C|)$ Zeit



2) Punktzuweisung: Algorithmus

Punktzuweisung(S_i, i, P)

Input: Unterteilung S_i, Index $1 \leq i \leq n$, Punktmenge P

Output: Punkt p_f für jede Facette f

$P' \leftarrow$ Punkte aus P rechts von v_i

priority queue $Q \leftarrow P' \cup \{v_{i+1}, \ldots, v_n\}$ sortiert nach Winkel bzgl. v_i

$T \leftarrow$ leerer binärer Suchbaum für radialen Sweep

while *Q nicht leer* **do**
    $p \leftarrow Q$.extractMax
    if *p Knoten von* C_{ij} **then**
        update T mit Kanten von p
    else
        $e \leftarrow$ linkester Kante $v_k v_{k+1}$ in T rechts von p auf sweep line
        speichere p an e

gehe über alle Kanten $v_i v_{i+1}, \ldots, v_{n-1} v_n$ und weise Facetten f maximalen/minimalen Punkt p_f zu

Behauptung: Benötigt $O((|P| + |C|) \log |C|)$ Zeit 

Aufgabe 4.2

Überlegen Sie sich basierend auf geometrischen Beobachtungen ein Beschleunigungsverfahren, mit dessen Hilfe die Punktmenge P eingeschränkt werden kann.

Aufgabe 4.3

Überlegen Sie sich ein Beschleunigungsverfahren, mit dessen Hilfe die Konsistenzberechnung von shortcuts beschleunigt werden kann.

Algorithmus von Hershberger und Snoeyink

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Aufgabe 3.1

1. Geben Sie einen nicht einfachen Polygonzug an, für den der Algorithmus von Hershberger und Snoeyink nicht funkioniert.
2. An welcher Stelle müsste der Algorithmus abgeändert werden, damit er auch auf nicht einfachen Polygonzügen funktioniert?

```
DPHull(i, j, PH)
    p_f ← farthest(PH, p_i p_j)
    if dist(p_i p_j, p_f) ≤ ε then
        return p_i p_j
    else
        if f < m then
            split(PH, f)
            DPHull(f, j, PH)
            build(i, f, PH)
            DPHull(i, f, PH)
        else
            ...
```


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Aufgabe 3.1

1. Geben Sie einen nicht einfachen Polygonzug an, für den der Algorithmus von Hershberger und Snoeyink nicht funkioniert.
2. An welcher Stelle müsste der Algorithmus abgeändert werden, damit er auch auf nicht einfachen Polygonzügen funktioniert?

```
DPHull(i, j, PH)
    p_f ← farthest(PH, p_i p_j)
    if dist(p_i p_j, p_f) ≤ ε then
        return p_i p_j
    else
        if f < m then
            split(PH, f)
            DPHull(f, j, PH)
            build(i, f, PH)
            DPHull(i, f, PH)
        else
            ...
```


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Aufgabe 3.1

build(i, j, PH)**:** berechne iterativ rechte (linke) konvexe Hülle

$Q \leftarrow$ double-ended queue (p_{m+1}, p_m, p_{m+1}) ; /* $CH(p_m, \ldots, p_k)$ */

for $k = m + 2, \ldots, j$ **do**

while p_k *nicht rechts von top*(Q) **do** pop-top(Q)

while p_k *nicht rechts von bottom*(Q) **do** pop-bottom(Q)

if *popped* **then** push-top(Q, p_k); push-bottom(Q, p_k)

Für folgenden Polygonzug P funkioniert die Berechnung der konvexen Hülle nicht:

Aufgabe 3.1

build(i, j, PH)**:** berechne iterativ rechte (linke) konvexe Hülle

$Q \leftarrow$ double-ended queue (p_{m+1}, p_m, p_{m+1}) ; /* $CH(p_m, \ldots, p_k)$ */

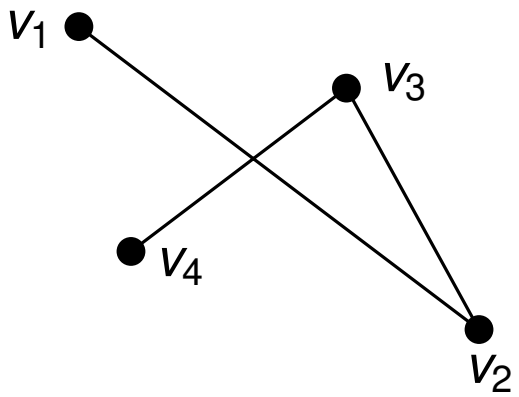
for $k = m + 2, \ldots, j$ **do**

while p_k *nicht rechts von top*(Q) **do** pop-top(Q)

while p_k *nicht rechts von bottom*(Q) **do** pop-bottom(Q)

if *popped* **then** push-top(Q, p_k); push-bottom(Q, p_k)

Für folgenden Polygonzug P funkioniert die Berechnung der konvexen Hülle nicht:



Aufgabe 3.1

build(i, j, PH)**:** berechne iterativ rechte (linke) konvexe Hülle

$Q \leftarrow$ double-ended queue (p_{m+1}, p_m, p_{m+1}) ; /* $CH(p_m, \ldots, p_k)$ */

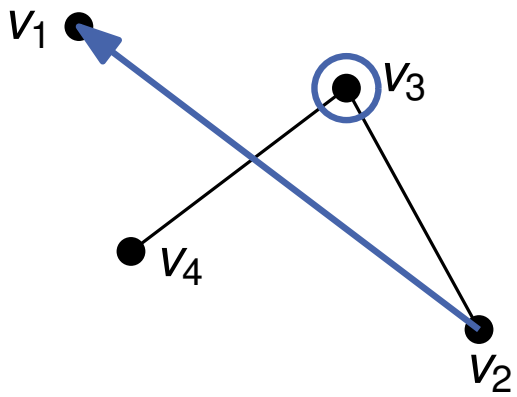
for $k = m + 2, \ldots, j$ **do**

while p_k *nicht rechts von top*(Q) **do** pop-top(Q)

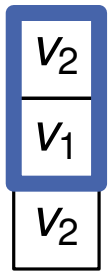
while p_k *nicht rechts von bottom*(Q) **do** pop-bottom(Q)

if *popped* **then** push-top(Q, p_k); push-bottom(Q, p_k)

Für folgenden Polygonzug P funkioniert die Berechnung der konvexen Hülle nicht:



Stack:



Aufgabe 3.1

build(i, j, PH)**:** berechne iterativ rechte (linke) konvexe Hülle

$Q \leftarrow$ double-ended queue (p_{m+1}, p_m, p_{m+1}) ; /* $CH(p_m, \ldots, p_k)$ */

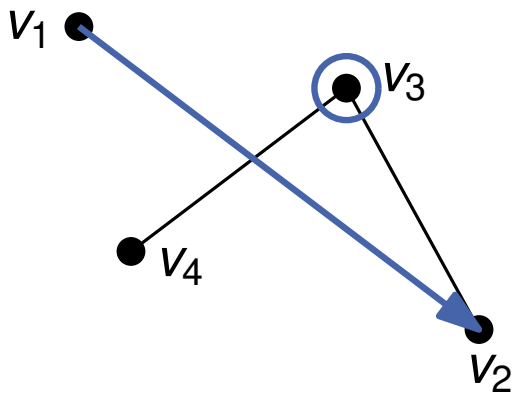
for $k = m + 2, \ldots, j$ **do**

while p_k *nicht rechts von top*(Q) **do** pop-top(Q)

while p_k *nicht rechts von bottom*(Q) **do** pop-bottom(Q)

if *popped* **then** push-top(Q, p_k); push-bottom(Q, p_k)

Für folgenden Polygonzug P funkioniert die Berechnung der konvexen Hülle nicht:



Stack:

Aufgabe 3.1

build(i, j, PH)**:** berechne iterativ rechte (linke) konvexe Hülle

$Q \leftarrow$ double-ended queue (p_{m+1}, p_m, p_{m+1}) ; /* $CH(p_m, \ldots, p_k)$ */

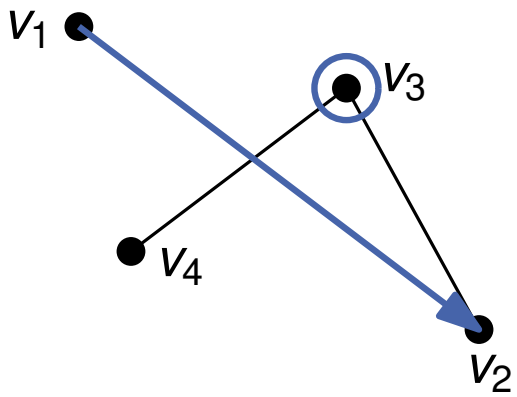
for $k = m + 2, \ldots, j$ **do**

while p_k *nicht rechts von top*(Q) **do** pop-top(Q)

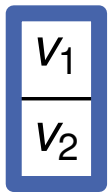
while p_k *nicht rechts von bottom*(Q) **do** pop-bottom(Q)

if *popped* **then** push-top(Q, p_k); push-bottom(Q, p_k)

Für folgenden Polygonzug P funkioniert die Berechnung der konvexen Hülle nicht:



Stack:



Aufgabe 3.1

build(i, j, PH)**:** berechne iterativ rechte (linke) konvexe Hülle

$Q \leftarrow$ double-ended queue (p_{m+1}, p_m, p_{m+1}) ; /* $CH(p_m, \ldots, p_k)$ */

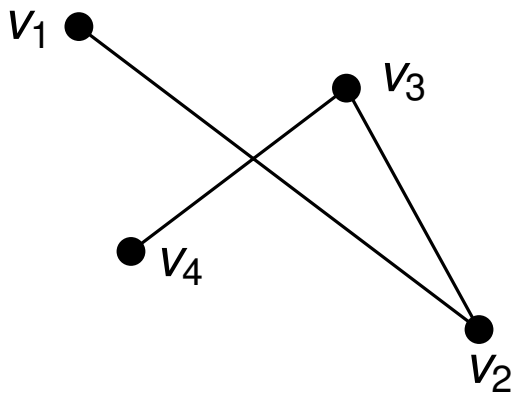
for $k = m + 2, \ldots, j$ **do**

while p_k *nicht rechts von top*(Q) **do** pop-top(Q)

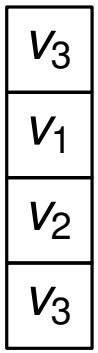
while p_k *nicht rechts von bottom*(Q) **do** pop-bottom(Q)

if *popped* **then** push-top(Q, p_k); push-bottom(Q, p_k)

Für folgenden Polygonzug P funkioniert die Berechnung der konvexen Hülle nicht:



Stack:



Aufgabe 3.1

build(i, j, PH)**:** berechne iterativ rechte (linke) konvexe Hülle

$Q \leftarrow$ double-ended queue (p_{m+1}, p_m, p_{m+1}) ; /* $CH(p_m, \ldots, p_k)$ */

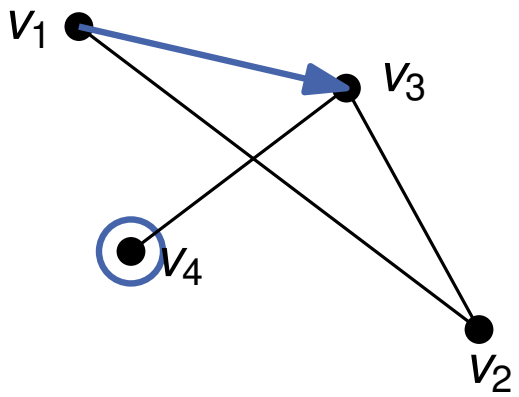
for $k = m + 2, \ldots, j$ **do**

while p_k *nicht rechts von top*(Q) **do** pop-top(Q)

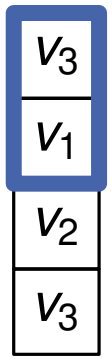
while p_k *nicht rechts von bottom*(Q) **do** pop-bottom(Q)

if *popped* **then** push-top(Q, p_k); push-bottom(Q, p_k)

Für folgenden Polygonzug P funkioniert die Berechnung der konvexen Hülle nicht:



Stack:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Aufgabe 3.1

build(i, j, PH)**:** berechne iterativ rechte (linke) konvexe Hülle

$Q \leftarrow$ double-ended queue (p_{m+1}, p_m, p_{m+1}) ; /* $CH(p_m, \ldots, p_k)$ */

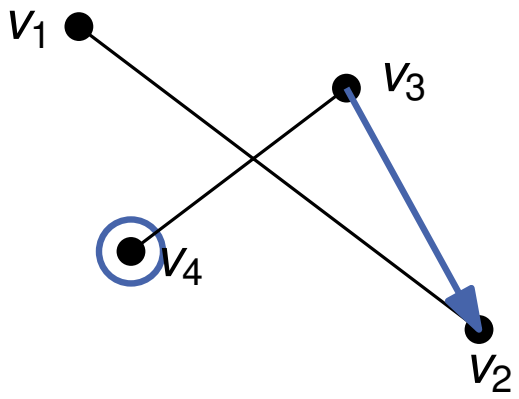
for $k = m + 2, \ldots, j$ **do**

while p_k *nicht rechts von top*(Q) **do** pop-top(Q)

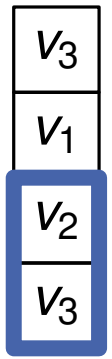
while p_k *nicht rechts von bottom*(Q) **do** pop-bottom(Q)

if *popped* **then** push-top(Q, p_k); push-bottom(Q, p_k)

Für folgenden Polygonzug P funkioniert die Berechnung der konvexen Hülle nicht:



Stack:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Aufgabe 3.1

build(i, j, PH)**:** berechne iterativ rechte (linke) konvexe Hülle

$Q \leftarrow$ double-ended queue (p_{m+1}, p_m, p_{m+1}) ; /* $CH(p_m, \ldots, p_k)$ */

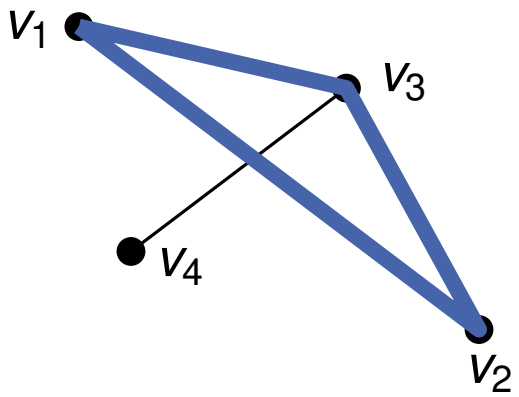
for $k = m + 2, \ldots, j$ **do**

while p_k *nicht rechts von top*(Q) **do** pop-top(Q)

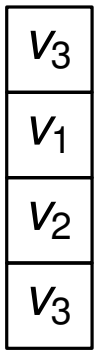
while p_k *nicht rechts von bottom*(Q) **do** pop-bottom(Q)

if *popped* **then** push-top(Q, p_k); push-bottom(Q, p_k)

Für folgenden Polygonzug P funkioniert die Berechnung der konvexen Hülle nicht:



Stack:



Aufgabe 3.1

build(i, j, PH)**:** berechne iterativ rechte (linke) konvexe Hülle

$Q \leftarrow$ double-ended queue (p_{m+1}, p_m, p_{m+1}) ; /* $CH(p_m, \ldots, p_k)$ */

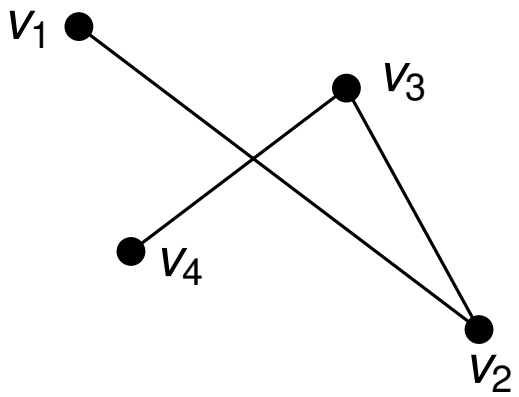
for $k = m + 2, \ldots, j$ **do**

while p_k *nicht rechts von top*(Q) **do** pop-top(Q)

while p_k *nicht rechts von bottom*(Q) **do** pop-bottom(Q)

if *popped* **then** push-top(Q, p_k); push-bottom(Q, p_k)

Für folgenden Polygonzug P funkioniert die Berechnung der konvexen Hülle nicht:



Lösung Aufgabe 3.1: Konstruiere mithilfe von P einen geeigneten Polygonzug.

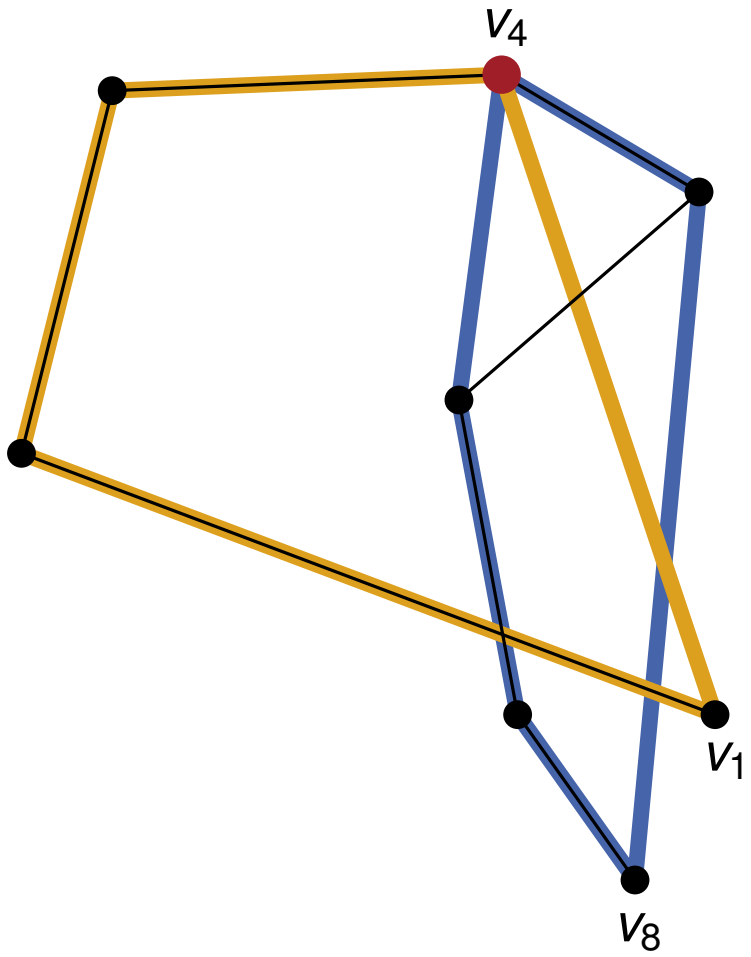
Aufgabe 3.3

Weist der Algorithmus für jeden nicht einfachen Polygonzug ein Fehlverhalten auf?

Aufgabe 3.3

Weist der Algorithmus für jeden nicht einfachen Polygonzug ein Fehlverhalten auf?

Polygonzug wird an geeigneter Stelle aufgetrennt:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner