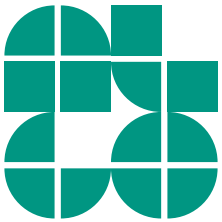
Vorlesung Algorithmische Kartografie

Vereinfachung von Kantenzügen und Unterteilungen

LEHRSTUHL FÜR ALGORITHMIK I · INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

Martin Nöllenburg

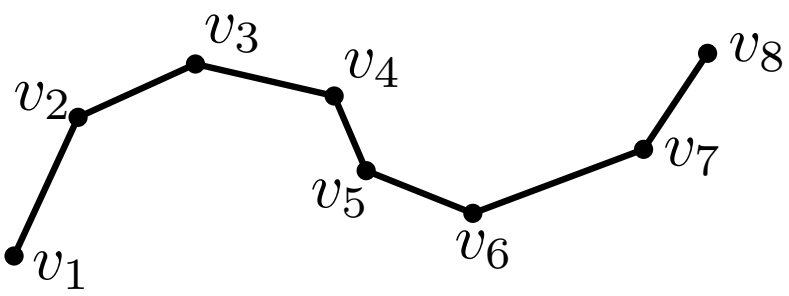
23.04.2013

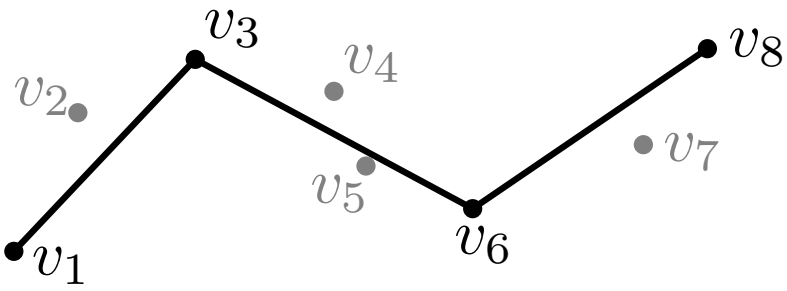


Wiederholung: Linienvereinfachung

geg: Polygonzug $P = (v_1, v_2, \ldots, v_n)$

ges: Polygonzug $Q = (v_1 = v_{i_1}, v_{i_2}, v_{i_3}, \ldots, v_{i_k} = v_n)$ mit $i_1 < i_2 < \cdots < i_k$, so dass Q eine gute Approximation von P und k möglichst klein ist

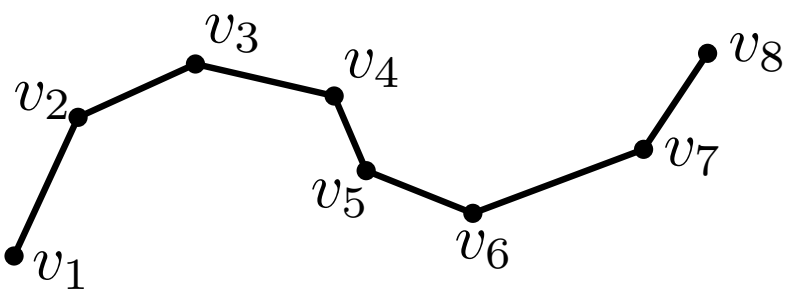


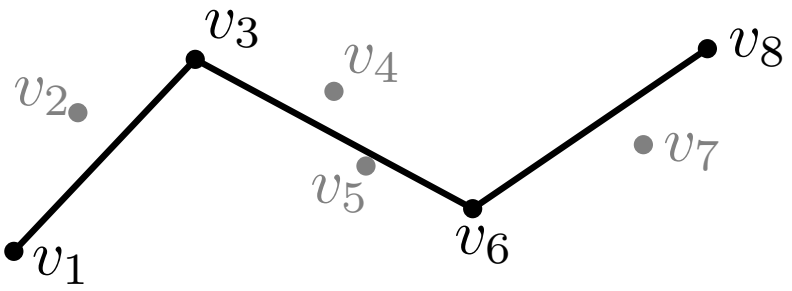


Wiederholung: Linienvereinfachung

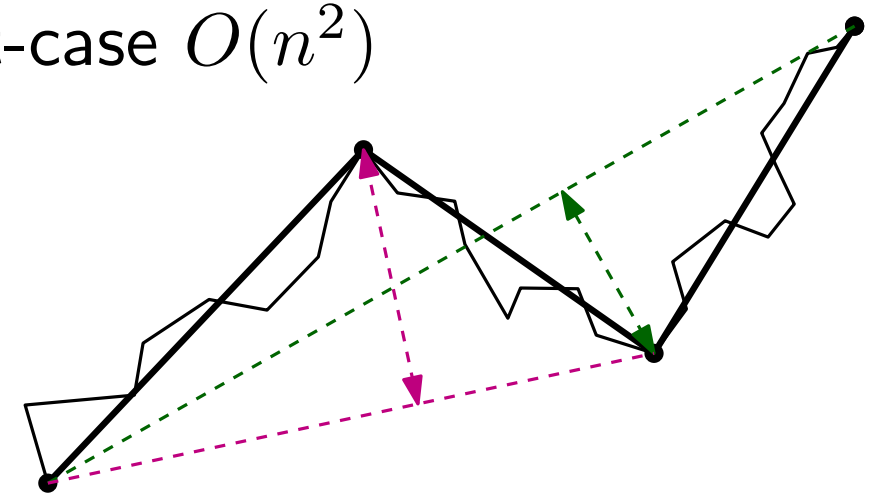
geg: Polygonzug $P = (v_1, v_2, \ldots, v_n)$

ges: Polygonzug $Q = (v_1 = v_{i_1}, v_{i_2}, v_{i_3}, \ldots, v_{i_k} = v_n)$ mit $i_1 < i_2 < \cdots < i_k$, so dass Q eine gute Approximation von P und k möglichst klein ist





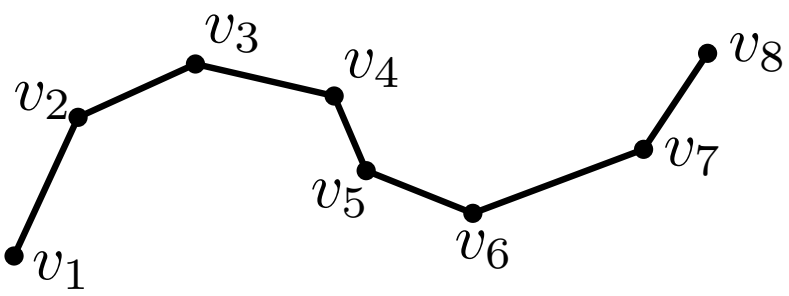
- einfacher rekursiver Douglas-Peucker Algorithmus, Laufzeit worst-case $O(n^2)$

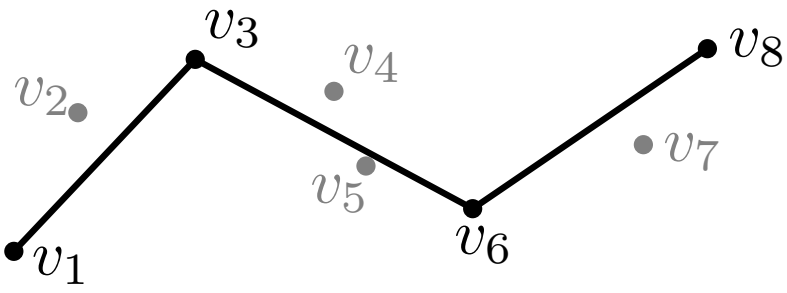


Wiederholung: Linienvereinfachung

geg: Polygonzug $P = (v_1, v_2, \ldots, v_n)$

ges: Polygonzug $Q = (v_1 = v_{i_1}, v_{i_2}, v_{i_3}, \ldots, v_{i_k} = v_n)$ mit $i_1 < i_2 < \cdots < i_k$, so dass Q eine gute Approximation von P und k möglichst klein ist





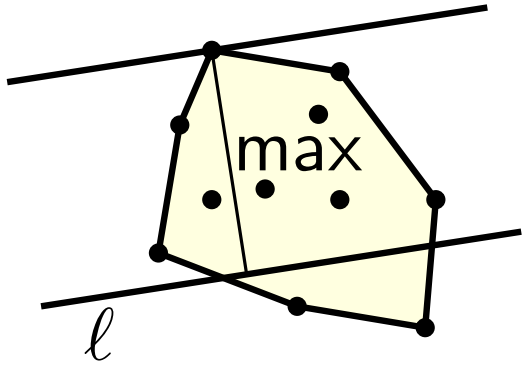
- einfacher rekursiver Douglas-Peucker Algorithmus, Laufzeit worst-case $O(n^2)$
- schnellerer Algorithmus von Hershberger and Snoeyink, der Bestimmung der weitesten Punkte beschleunigt

$\rightarrow$ restlicher Beweis: heute

Beschleunigung des DP-Algorithmus

[Hershberger, Snoeyink '92]

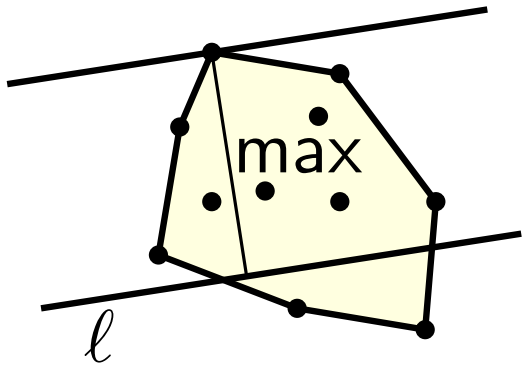
Beobachtung: für Gerade ℓ und Punktmenge P liegt der von ℓ weitest entfernte Punkt auf einer zu ℓ parallelen Tangente an die konvexe Hülle $CH(P)$



Beschleunigung des DP-Algorithmus

[Hershberger, Snoeyink '92]

Beobachtung: für Gerade ℓ und Punktmenge P liegt der von ℓ weitest entfernte Punkt auf einer zu ℓ parallelen Tangente an die konvexe Hülle $CH(P)$

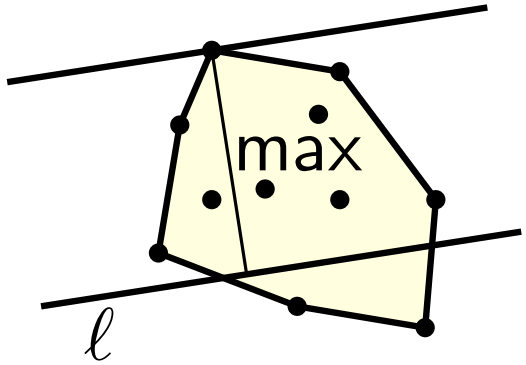


$\rightarrow$ müssen nur solche Punkte als Splitknoten betrachten

Beschleunigung des DP-Algorithmus

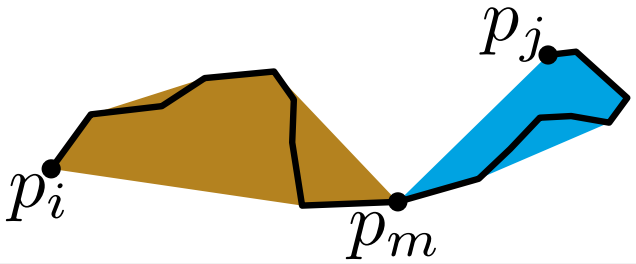
[Hershberger, Snoeyink '92]

Beobachtung: für Gerade ℓ und Punktmenge P liegt der von ℓ weitest entfernte Punkt auf einer zu ℓ parallelen Tangente an die konvexe Hülle $CH(P)$



$\rightarrow$ müssen nur solche Punkte als Splitknoten betrachten

Def: Pfadhülle PH für $(p_i, \ldots, p_j)$ besteht aus einem Zwischenknoten p_m und den konvexen Hüllen $CH(p_i, \ldots, p_m)$ und $CH(p_m, \ldots, p_j)$



Algorithmus von Hershberger und Snoeyink

drei Operationen:

- build(i, j, PH): erstelle Pfadhülle von $(p_i, \ldots, p_j)$ mit Zwischenknoten $p_m = p_{\lfloor (i+j)/2 \rfloor}$
- split(PH, k): trenne Pfad $(p_i, \ldots, p_j)$ an Stelle k und gib Pfadhülle für Teilpfad, der p_m enthält zurück
- farthest(PH, ℓ): finde weitesten von ℓ entfernten Knoten in PH

DPHull(i, j, PH)

$p_f \leftarrow$ farthest$(PH, \overline{p_i p_j})$
if dist$(\overline{p_i p_j}, p_f) \leq \varepsilon$ **then**
  return $\overline{p_i p_j}$
else
  if $f < m$ **then**
    split(PH, f)
    DPHull(f, j, PH)
    build(i, f, PH)
    DPHull(i, f, PH)
  else
    split(PH, f)
    DPHull(i, f, PH)
    build(f, j, PH)
    DPHull(f, j, PH)

Algorithmus von Hershberger und Snoeyink

drei Operationen:

- build(i, j, PH): erstelle Pfadhülle von $(p_i, \ldots, p_j)$ mit Zwischenknoten $p_m = p_{\lfloor (i+j)/2 \rfloor}$
- split(PH, k): trenne Pfad $(p_i, \ldots, p_j)$ an Stelle k und gib Pfadhülle für Teilpfad, der p_m enthält zurück
- farthest(PH, ℓ): finde weitesten von ℓ entfernten Knoten in PH ✓ $O(\log n)$

DPHull(i, j, PH)

$p_f \leftarrow$ farthest$(PH, \overline{p_i p_j})$
if dist$(\overline{p_i p_j}, p_f) \leq \varepsilon$ **then**
  return $\overline{p_i p_j}$
else
  if $f < m$ **then**
    split(PH, f)
    DPHull(f, j, PH)
    build(i, f, PH)
    DPHull(i, f, PH)
  else
    split(PH, f)
    DPHull(i, f, PH)
    build(f, j, PH)
    DPHull(f, j, PH)

build und split Operationen

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

$Q \leftarrow$ double-ended queue (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ */
$H \leftarrow$ history stack /* merkt sich Operationen */
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)

- durch H sind alle Hüllen $CH(p_m, \ldots, p_k)$ rekonstruierbar
- linke Hülle analog für $k = m - 2, \ldots, i$

build und split Operationen

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

$Q \leftarrow$ double-ended queue (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ */
$H \leftarrow$ history stack /* merkt sich Operationen */
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)

P muss einfach sein!

- durch H sind alle Hüllen $CH(p_m, \ldots, p_k)$ rekonstruierbar
- linke Hülle analog für $k = m - 2, \ldots, i$

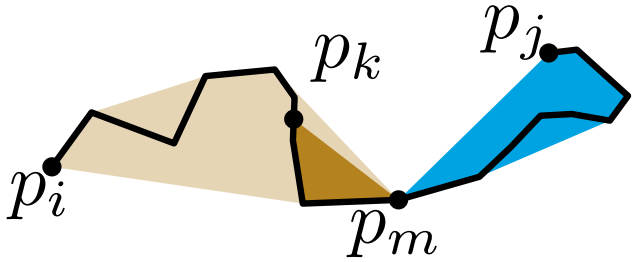
build und split Operationen

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

$Q \leftarrow$ double-ended queue (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ */
$H \leftarrow$ history stack /* merkt sich Operationen */
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)

- durch H sind alle Hüllen $CH(p_m, \ldots, p_k)$ rekonstruierbar
- linke Hülle analog für $k = m - 2, \ldots, i$

split(PH, k)**:** nutze history stack der Teilhülle von PH, die p_k enthält um $CH(p_k, \ldots, p_m)$ bzw. $CH(p_m, \ldots, p_k)$ zu rekonstruieren



Algorithmus von Hershberger und Snoeyink

drei Operationen:

- build(i, j, PH): erstelle Pfadhülle von $(p_i, \ldots, p_j)$ mit Zwischenknoten $p_m = p_{\lfloor (i+j)/2 \rfloor}$ ✓
- split(PH, k): trenne Pfad $(p_i, \ldots, p_j)$ an Stelle k und gib Pfadhülle für Teilpfad, der p_m enthält zurück ✓
- farthest(PH, ℓ): finde weitesten von ℓ entfernten Knoten in PH ✓

$O(\log n)$

Satz: DPHull$(1, n, PH)$ kann mit Laufzeit $O(n \log n)$ implementiert werden.
(Ann: $(p_1, \ldots p_n)$ schnittfrei)

DPHull(i, j, PH)

$p_f \leftarrow$ farthest$(PH, \overline{p_i p_j})$
if dist$(\overline{p_i p_j}, p_f) \leq \varepsilon$ **then**
  return $\overline{p_i p_j}$
else
  if $f < m$ **then**
    split(PH, f)
    DPHull(f, j, PH)
    build(i, f, PH)
    DPHull(i, f, PH)
  else
    split(PH, f)
    DPHull(i, f, PH)
    build(f, j, PH)
    DPHull(f, j, PH)

Alternative Punktauswahl

Entferntesten Punkt wie bei DP zu behalten kann Ausreißer betonen.

Betrachte stattdessen **effektive Fläche** jedes Knotens und entferne iterativ Knoten mit kleinster Fläche.

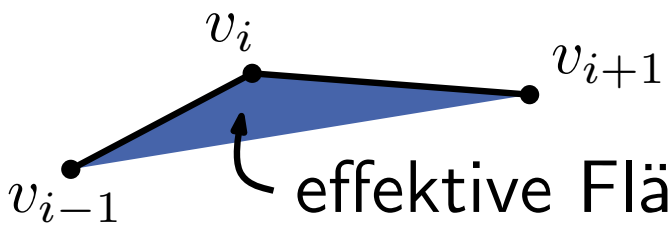
[Visvalingam, Whyatt '93]

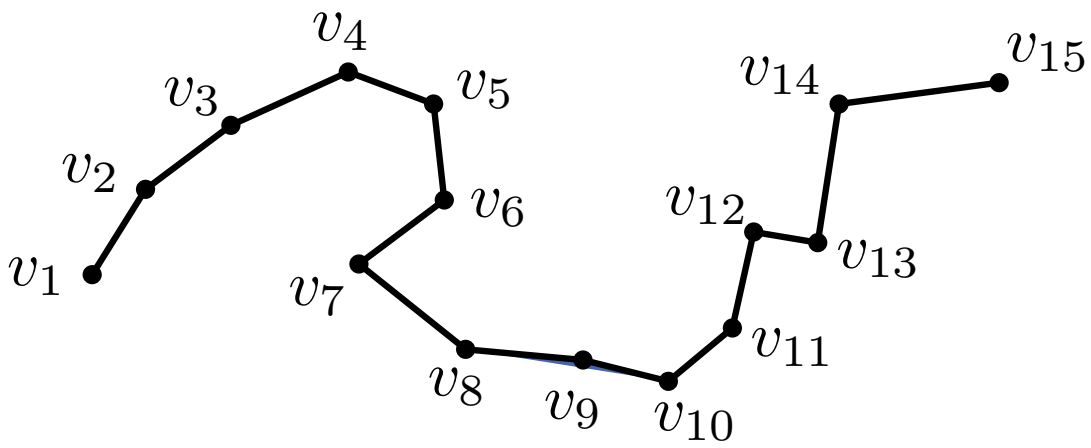
Alternative Punktauswahl

Entferntesten Punkt wie bei DP zu behalten kann Ausreißer betonen.

Betrachte stattdessen **effektive Fläche** jedes Knotens und entferne iterativ Knoten mit kleinster Fläche.

[Visvalingam, Whyatt '93]



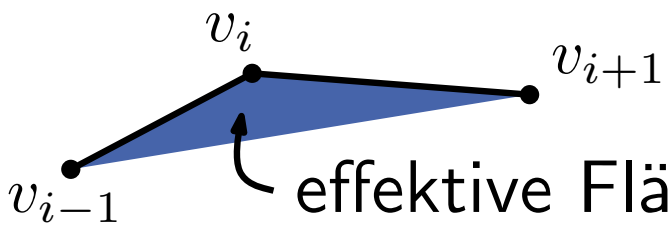


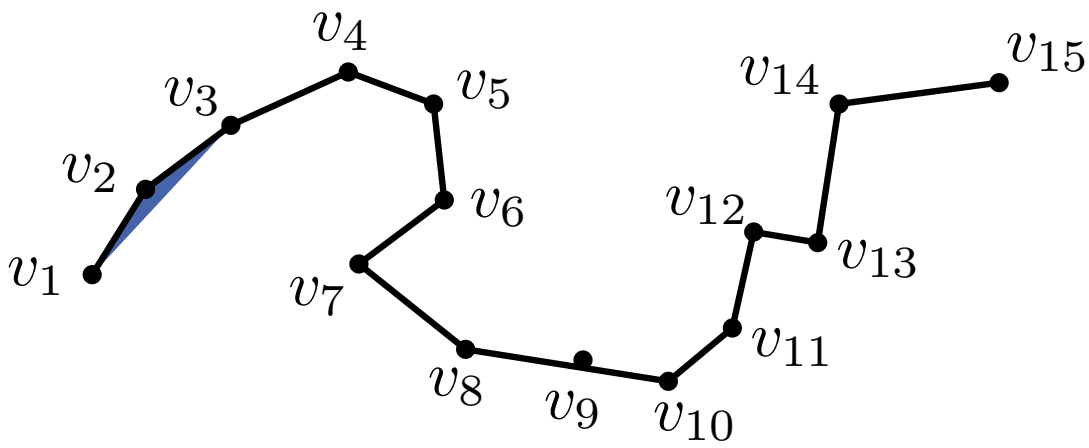
Alternative Punktauswahl

Entferntesten Punkt wie bei DP zu behalten kann Ausreißer betonen.

Betrachte stattdessen **effektive Fläche** jedes Knotens und entferne iterativ Knoten mit kleinster Fläche.

[Visvalingam, Whyatt '93]



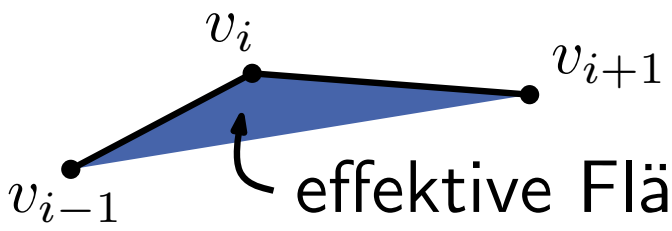


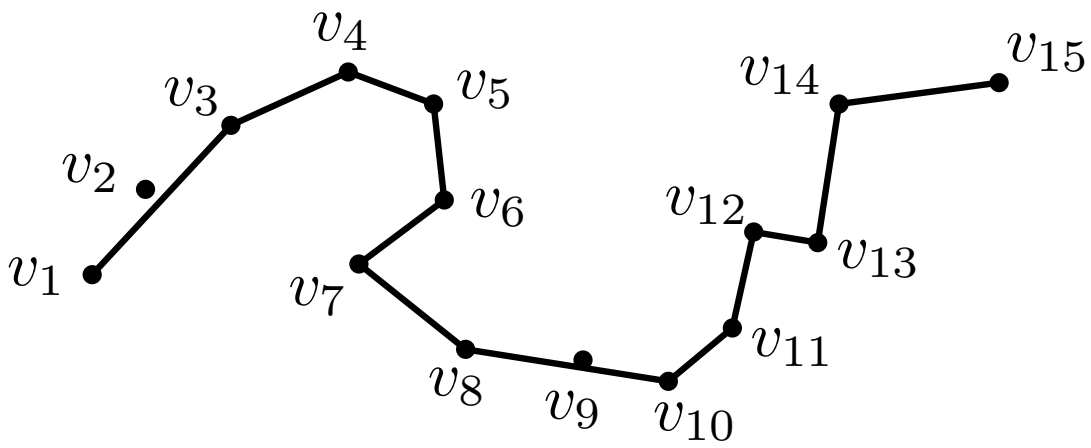
Alternative Punktauswahl

Entferntesten Punkt wie bei DP zu behalten kann Ausreißer betonen.

Betrachte stattdessen **effektive Fläche** jedes Knotens und entferne iterativ Knoten mit kleinster Fläche.

[Visvalingam, Whyatt '93]



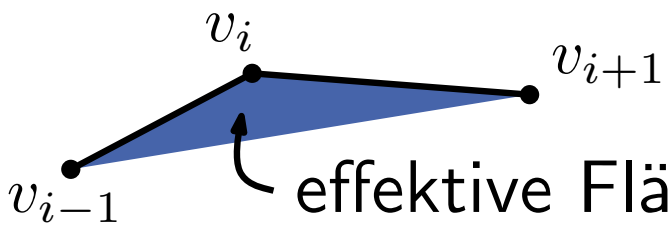


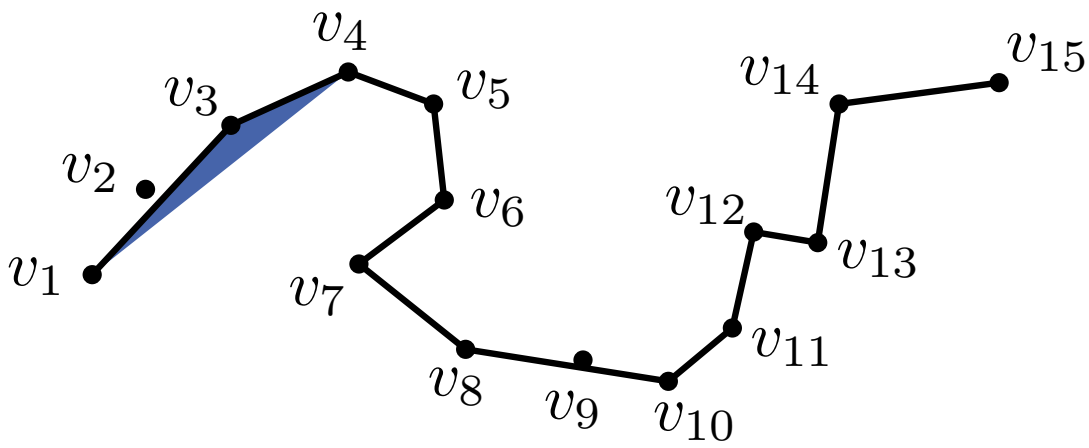
Alternative Punktauswahl

Entferntesten Punkt wie bei DP zu behalten kann Ausreißer betonen.

Betrachte stattdessen **effektive Fläche** jedes Knotens und entferne iterativ Knoten mit kleinster Fläche.

[Visvalingam, Whyatt '93]



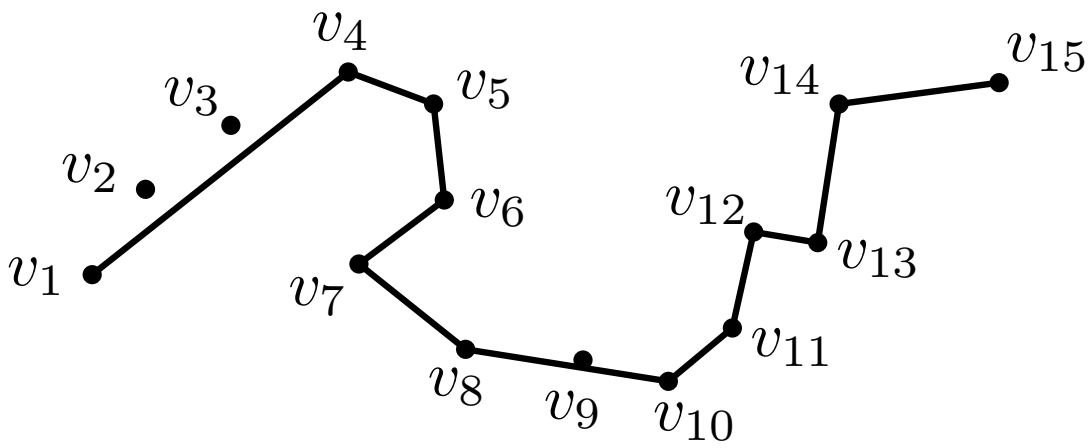


Alternative Punktauswahl

Entferntesten Punkt wie bei DP zu behalten kann Ausreißer betonen.

Betrachte stattdessen **effektive Fläche** jedes Knotens und entferne iterativ Knoten mit kleinster Fläche.

[Visvalingam, Whyatt '93]

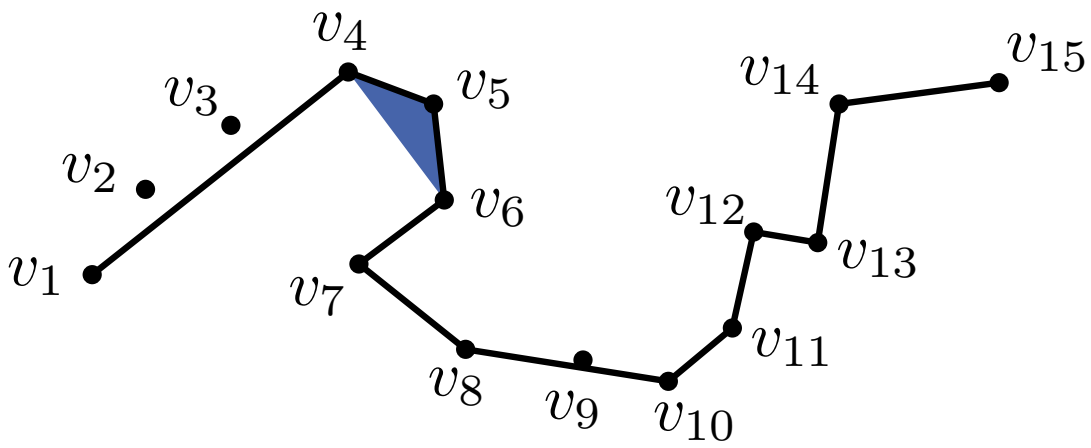


Alternative Punktauswahl

Entferntesten Punkt wie bei DP zu behalten kann Ausreißer betonen.

Betrachte stattdessen **effektive Fläche** jedes Knotens und entferne iterativ Knoten mit kleinster Fläche.

[Visvalingam, Whyatt '93]

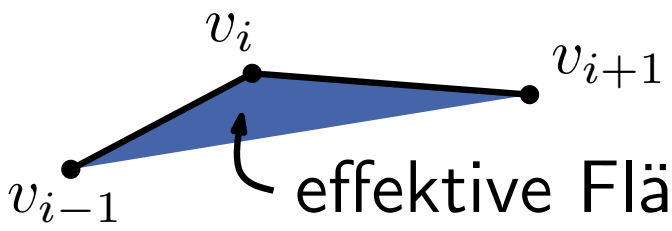


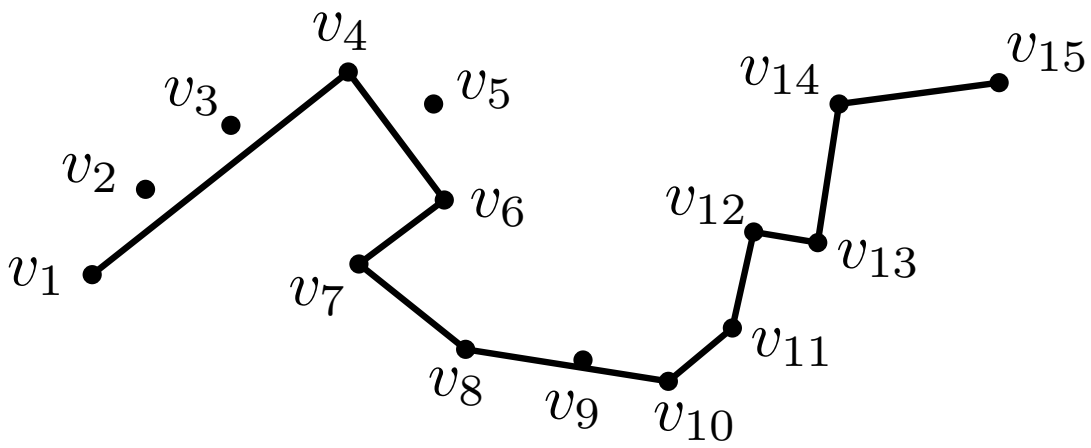
Alternative Punktauswahl

Entferntesten Punkt wie bei DP zu behalten kann Ausreißer betonen.

Betrachte stattdessen **effektive Fläche** jedes Knotens und entferne iterativ Knoten mit kleinster Fläche.

[Visvalingam, Whyatt '93]



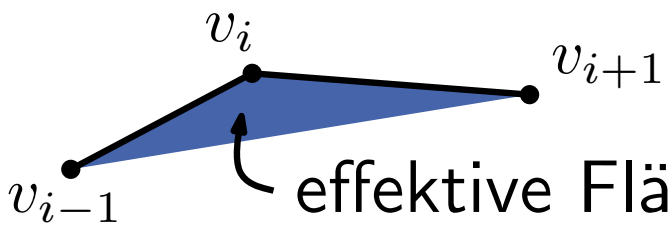


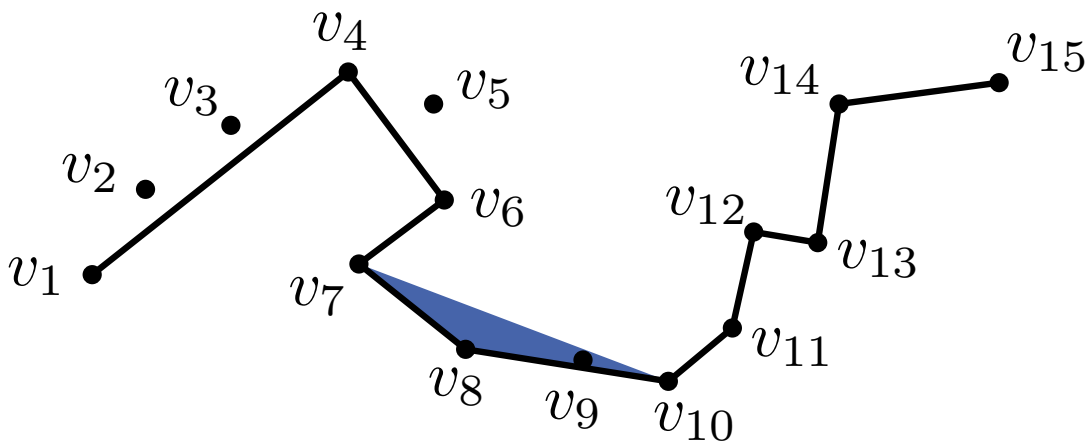
Alternative Punktauswahl

Entferntesten Punkt wie bei DP zu behalten kann Ausreißer betonen.

Betrachte stattdessen **effektive Fläche** jedes Knotens und entferne iterativ Knoten mit kleinster Fläche.

[Visvalingam, Whyatt '93]



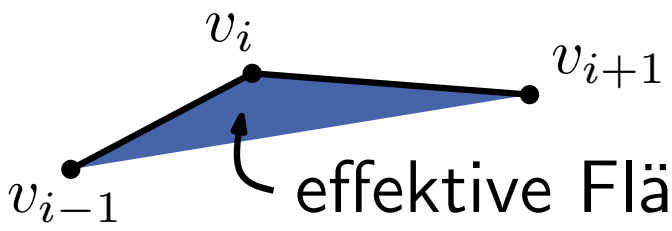


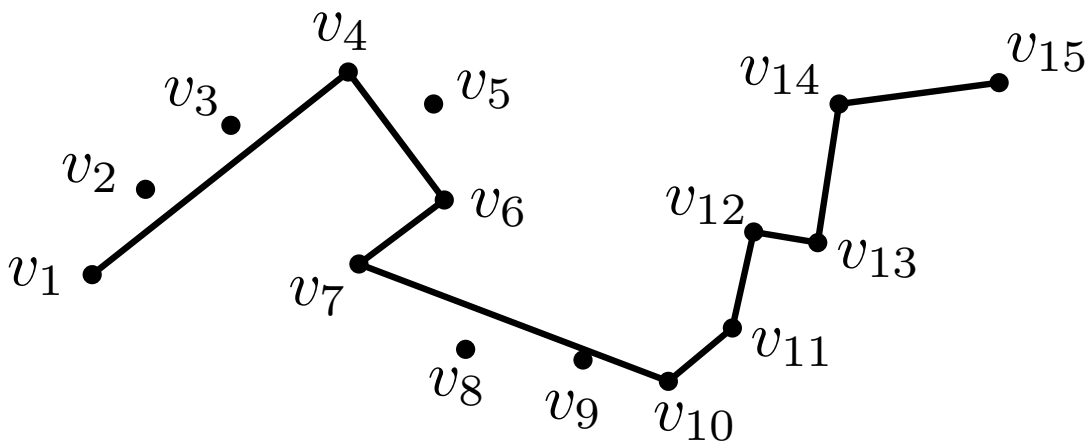
Alternative Punktauswahl

Entferntesten Punkt wie bei DP zu behalten kann Ausreißer betonen.

Betrachte stattdessen **effektive Fläche** jedes Knotens und entferne iterativ Knoten mit kleinster Fläche.

[Visvalingam, Whyatt '93]



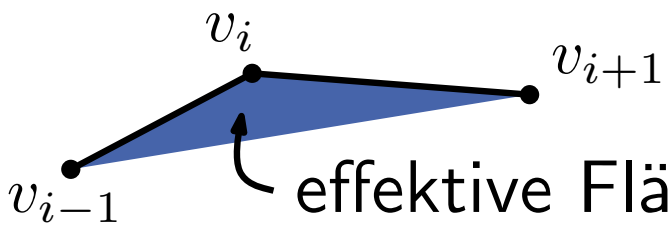


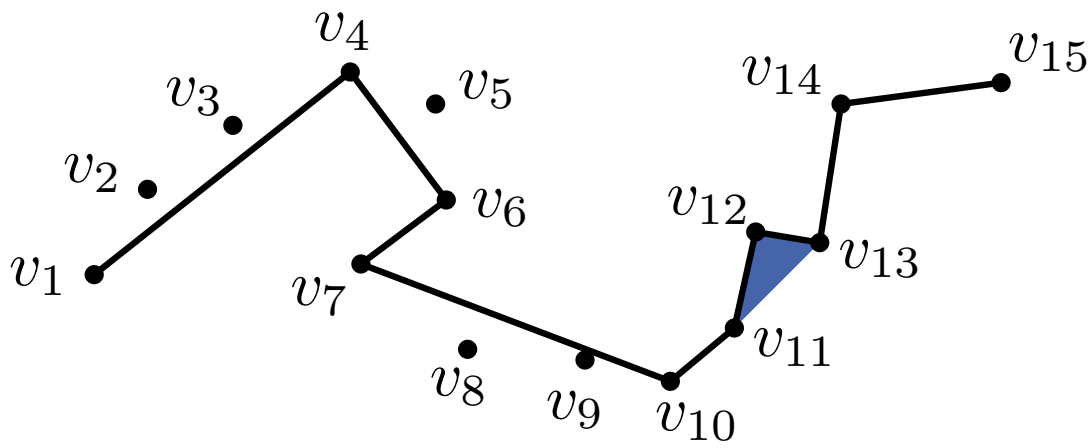
Alternative Punktauswahl

Entferntesten Punkt wie bei DP zu behalten kann Ausreißer betonen.

Betrachte stattdessen **effektive Fläche** jedes Knotens und entferne iterativ Knoten mit kleinster Fläche.

[Visvalingam, Whyatt '93]



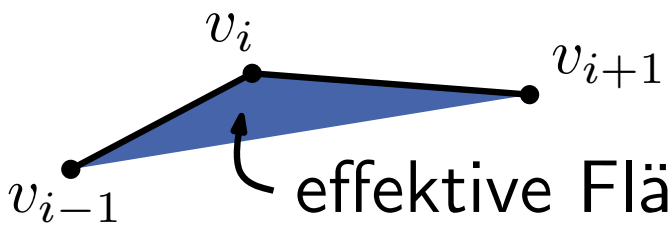


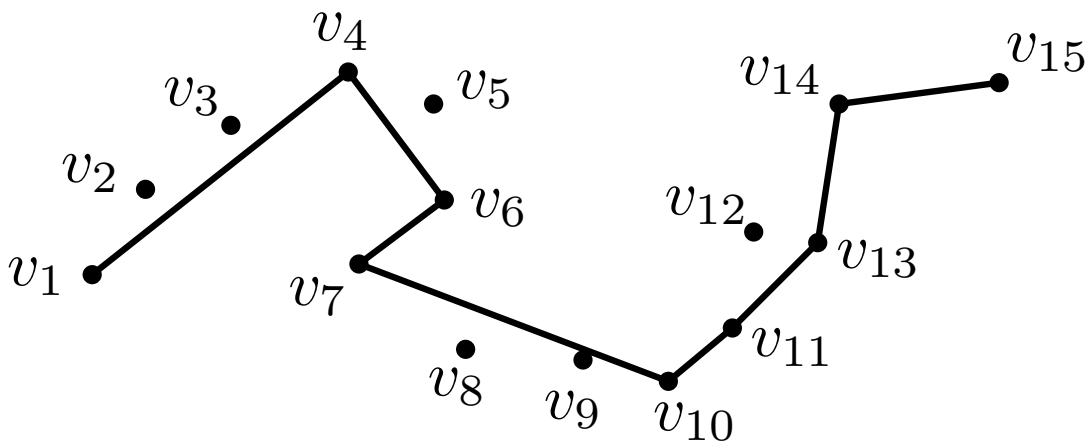
Alternative Punktauswahl

Entferntesten Punkt wie bei DP zu behalten kann Ausreißer betonen.

Betrachte stattdessen **effektive Fläche** jedes Knotens und entferne iterativ Knoten mit kleinster Fläche.

[Visvalingam, Whyatt '93]



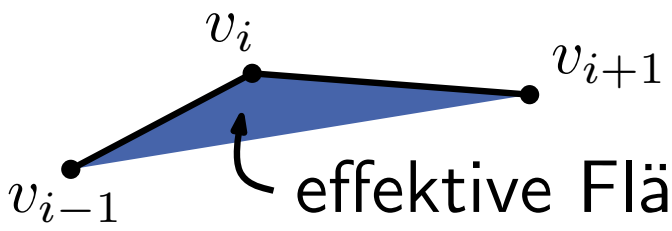


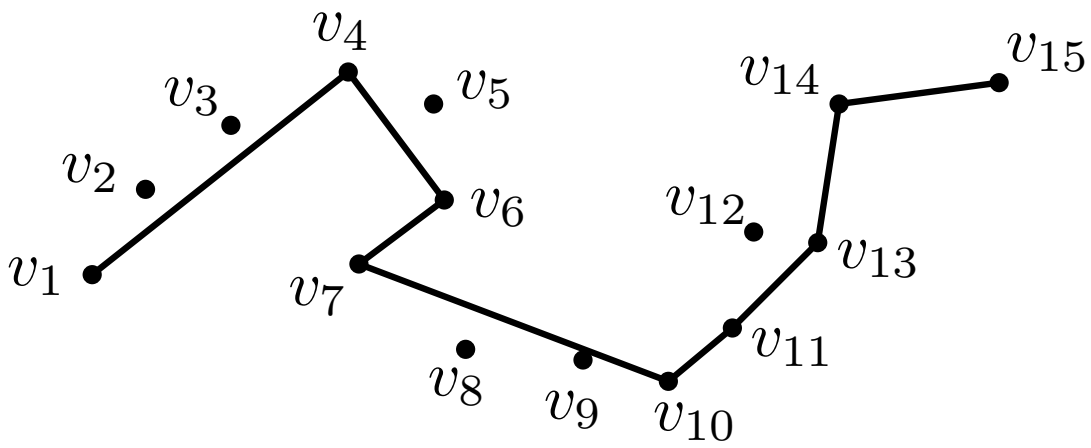
Alternative Punktauswahl

Entferntesten Punkt wie bei DP zu behalten kann Ausreißer betonen.

Betrachte stattdessen **effektive Fläche** jedes Knotens und entferne iterativ Knoten mit kleinster Fläche.

[Visvalingam, Whyatt '93]





Laufzeit?

Vereinfachung als Optimierungsproblem

Karlsruhe Institute of Technology

Bisherige Algorithmen liefern heuristisch gute Approximationen an die Eingabe, jedoch nicht notwendig eine optimale Lösung.

Formulierung als Graphenproblem:

Fall 1: feste Maximallänge, minimiere Fehler

- 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

Vereinfachung als Optimierungsproblem

Bisherige Algorithmen liefern heuristisch gute Approximationen an die Eingabe, jedoch nicht notwendig eine optimale Lösung.

Formulierung als Graphenproblem:

Fall 1: feste Maximallänge, minimiere Fehler

- 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

Solche „constrained shortest path“ Probleme sind i. Allg. NP-schwer, hier jedoch nicht.

$\rightarrow$ mehr dazu in der Übung!

Vereinfachung als Optimierungsproblem

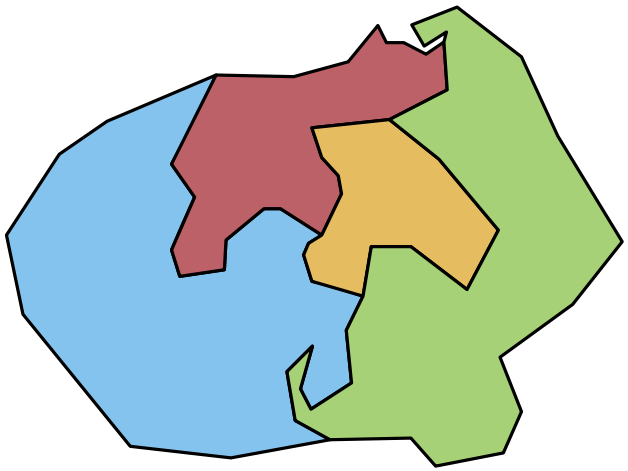
Bisherige Algorithmen liefern heuristisch gute Approximationen an die Eingabe, jedoch nicht notwendig eine optimale Lösung.

Formulierung als Graphenproblem:

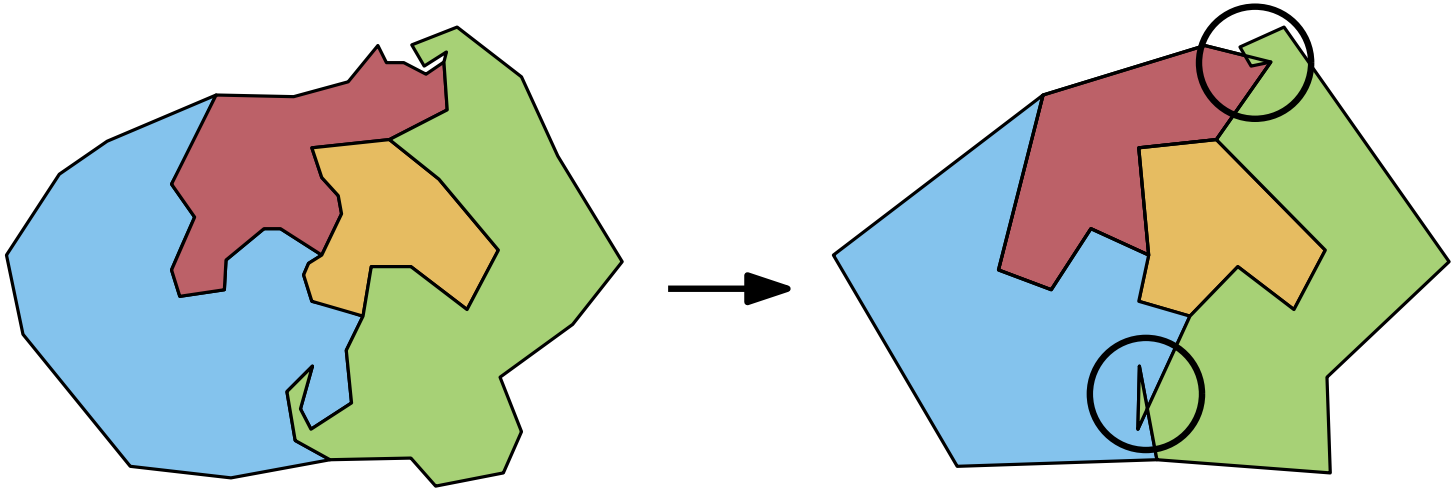
Fall 2: fester maximaler Fehler, minimiere Länge [Imai, Iri '88]

- DAG $G = (V, A)$ mit $V = \{v_1, \ldots, v_n\}$ und Kante $(v_i, v_j) \in A$, $i < j$, gdw. Fehler($\overline{v_i v_j}$) $< \varepsilon$
- finde kürzesten Weg von v_1 nach v_n in G
- topologisches Sortieren in linearer Zeit
- Berechnung von A naiv in $O(n^3)$, optimal in $\Theta(n^2)$ [Chan, Chin '92]

Pfadvereinfachung für Landkarten

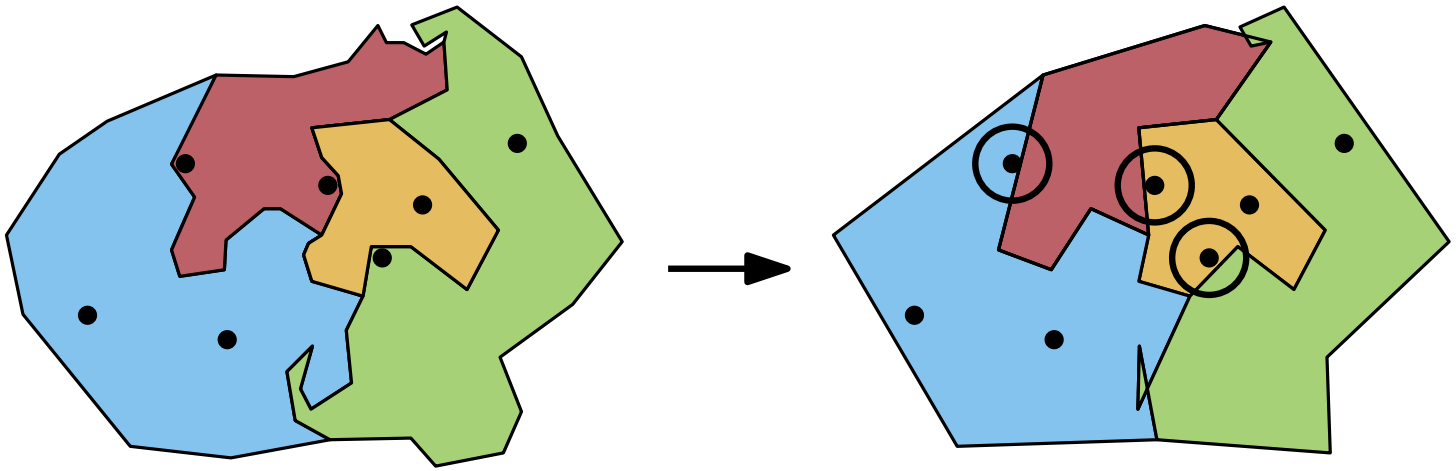


Pfadvereinfachung für Landkarten



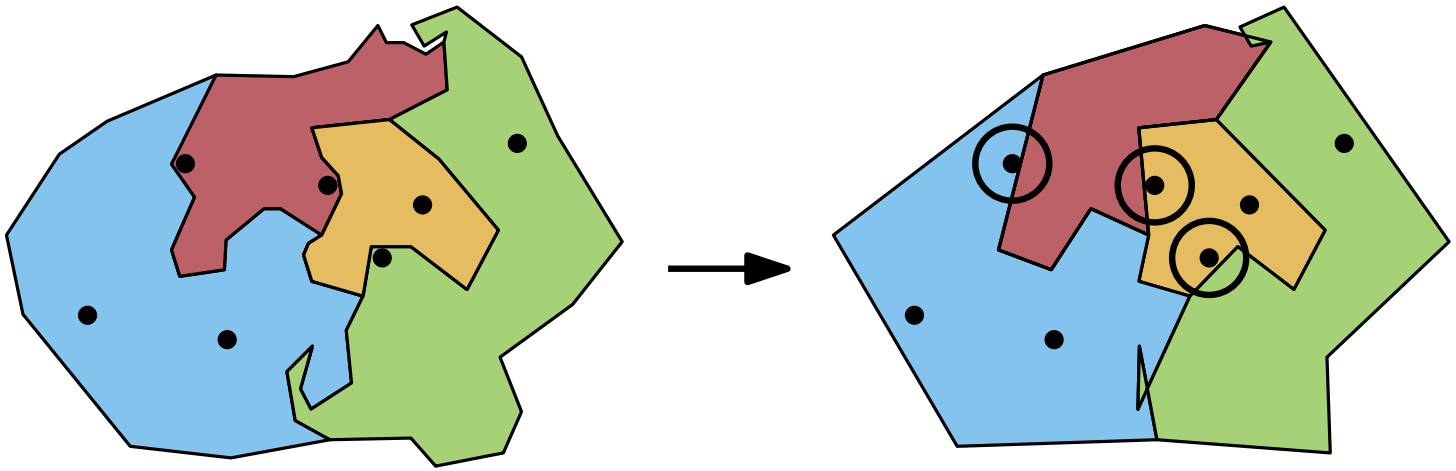
- Selbstschnitte und Schnitte mit anderen Pfaden möglich

Pfadvereinfachung für Landkarten



- Selbstschnitte und Schnitte mit anderen Pfaden möglich
- Punkte (z.B. Städte) können auf falsche Seite gelangen

Pfadvereinfachung für Landkarten



- Selbstschnitte und Schnitte mit anderen Pfaden möglich
- Punkte (z.B. Städte) können auf falsche Seite gelangen

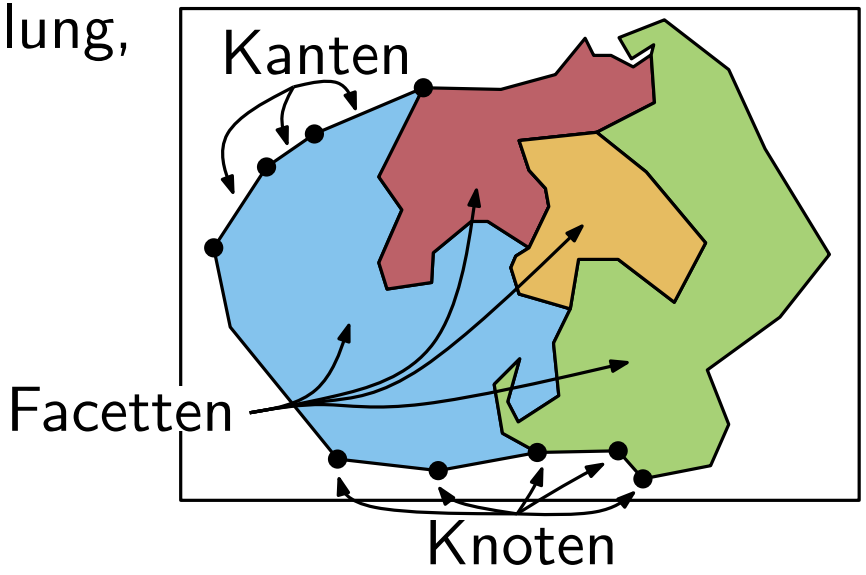
$\rightarrow$ benötigen topologisch korrekten Vereinfachungsalgorithmus

Ansatz 1: nachträgliches Korrigieren

Ansatz 2: Vermeiden durch „schlauere“ Algorithmen

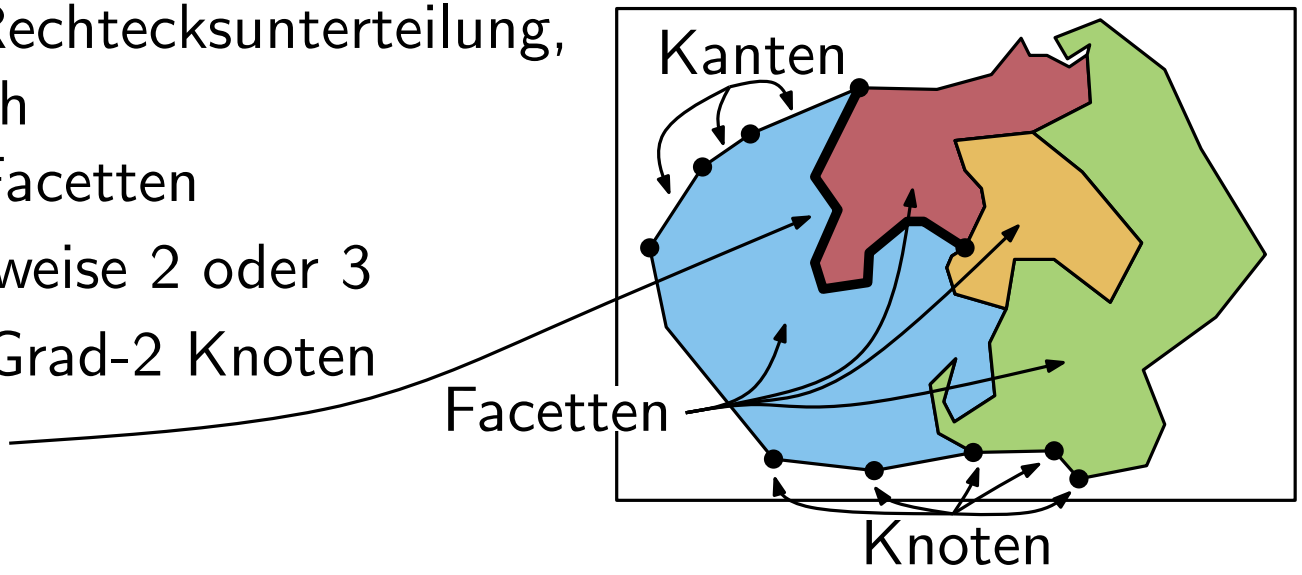
Ein paar Begriffe

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph
  $\rightarrow$ Knoten, Kanten, Facetten



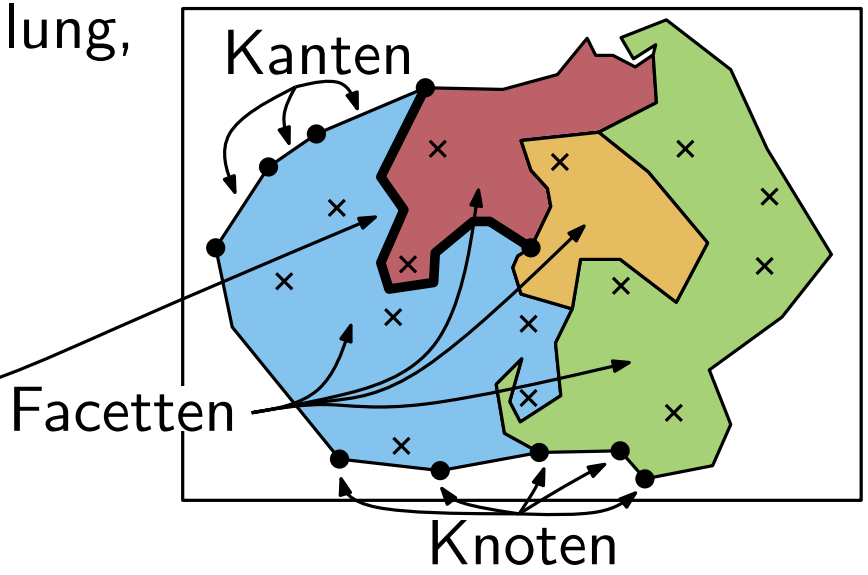
Ein paar Begriffe

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph
  $\rightarrow$ Knoten, Kanten, Facetten
- Knotengrade üblicherweise 2 oder 3
- maximale Pfade von Grad-2 Knoten bilden Kantenzüge



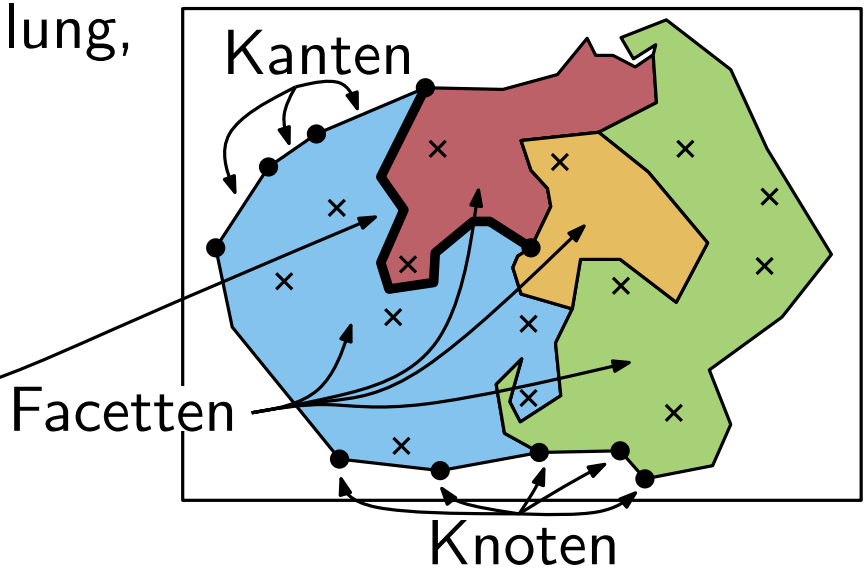
Ein paar Begriffe

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph
  $\rightarrow$ Knoten, Kanten, Facetten
- Knotengrade üblicherweise 2 oder 3
- maximale Pfade von Grad-2 Knoten bilden Kantenzüge
- zusätzlich Menge P von Punkten (x)



Ein paar Begriffe

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph
  $\rightarrow$ Knoten, Kanten, Facetten
- Knotengrade üblicherweise 2 oder 3
- maximale Pfade von Grad-2 Knoten bilden Kantenzüge
- zusätzlich Menge P von Punkten (x)



Ziel: ersetze alle Kantenzüge C durch vereinfachte Kantenzüge C' mit

- kein Punkt von C ist weiter als ε von C' entfernt
- C' ist einfach (d.h. schnittfrei)
- C' schneidet keinen weiteren Kantenzug
- alle Punkte in P liegen auf den gleichen Seiten bzgl. C und C', d.h. Punkte bleiben in den richtigen Facetten

$\rightarrow$ solch ein C' heißt **konsistente** Vereinfachung

Algorithmus von de Berg et al. (1998)

Karlsruhe Institute of Technology

Idee:

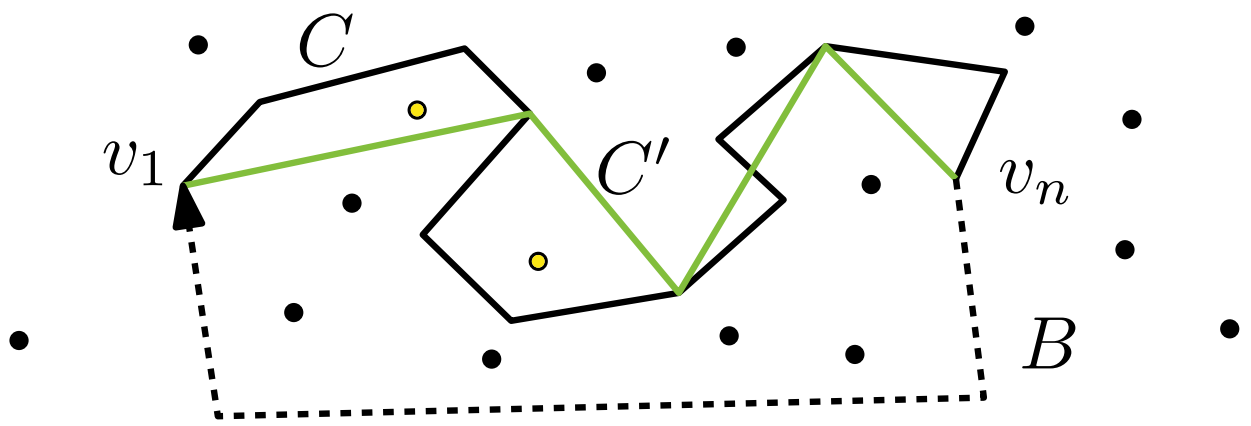
- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** shortcuts bzgl. Fehler ε bilden Graph G_1
- **konsistente** shortcuts bzgl. P bilden Graph G_2
- kürzester Weg in Graph $G = (V, A_1 \cap A_2)$ liefert optimale zulässige und konsistente Vereinfachung

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. 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.



Algorithmus von de Berg et al. (1998)

Karlsruhe Institute of Technology

Idee:

- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** shortcuts bzgl. Fehler ε bilden Graph G_1
- **konsistente** shortcuts bzgl. 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

Lemma 1: C' konsistente Vereinfachung von C bzgl. $P \Leftrightarrow$ kein Punkt von P liegt in Facette von $C \cup C'$

x-monotone Kantenzüge

Lemma 1: C' konsistente Vereinfachung von C bzgl. $P \Leftrightarrow$ kein Punkt von P liegt in Facette von $C \cup C'$

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

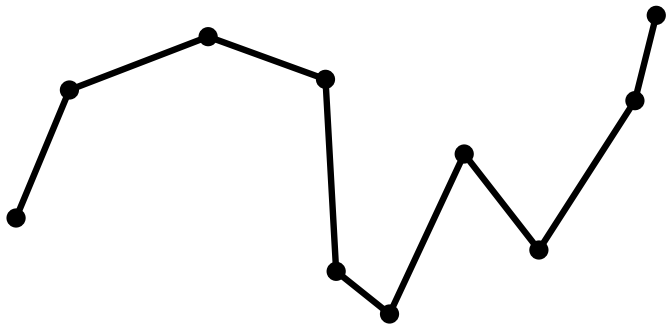
Lemma 1: C' konsistente Vereinfachung von C bzgl. $P \Leftrightarrow$ kein Punkt von P liegt in Facette von $C \cup C'$

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

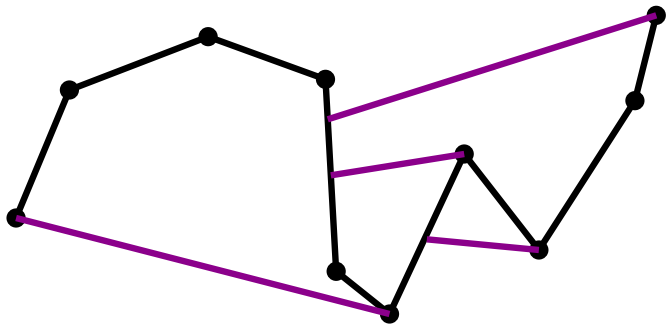
Lemma 1: C' konsistente Vereinfachung von C bzgl. $P \Leftrightarrow$ kein Punkt von P liegt in Facette von $C \cup C'$

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

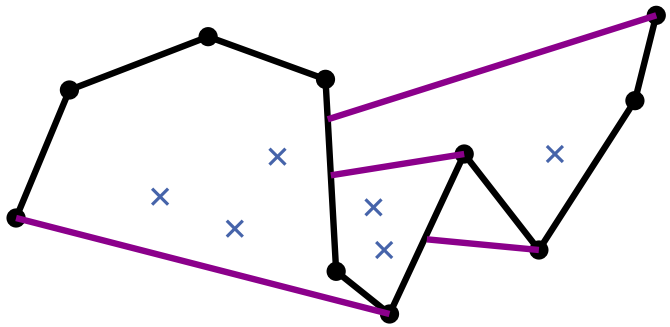
Lemma 1: C' konsistente Vereinfachung von C bzgl. $P \Leftrightarrow$ kein Punkt von P liegt in Facette von $C \cup C'$

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

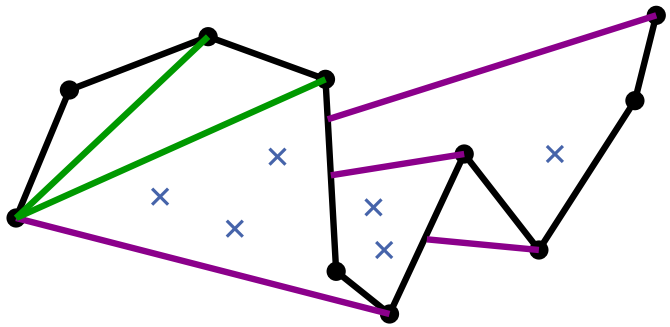
Lemma 1: C' konsistente Vereinfachung von C bzgl. $P \Leftrightarrow$ kein Punkt von P liegt in Facette von $C \cup C'$

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

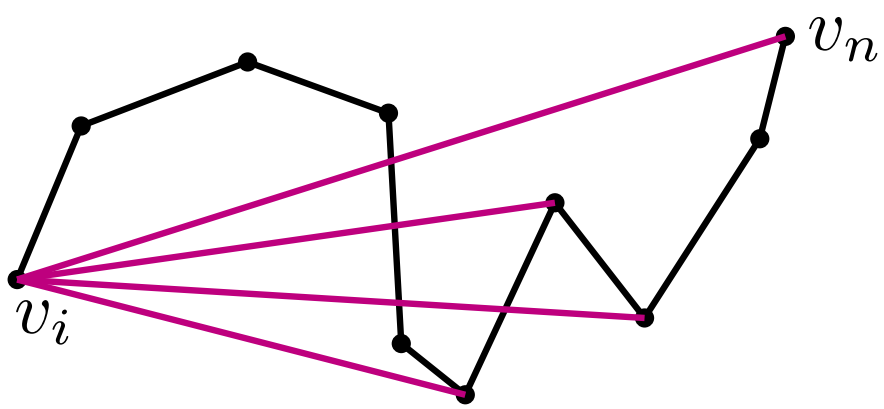
Vorgehen für festes v_i in 3 Schritten

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



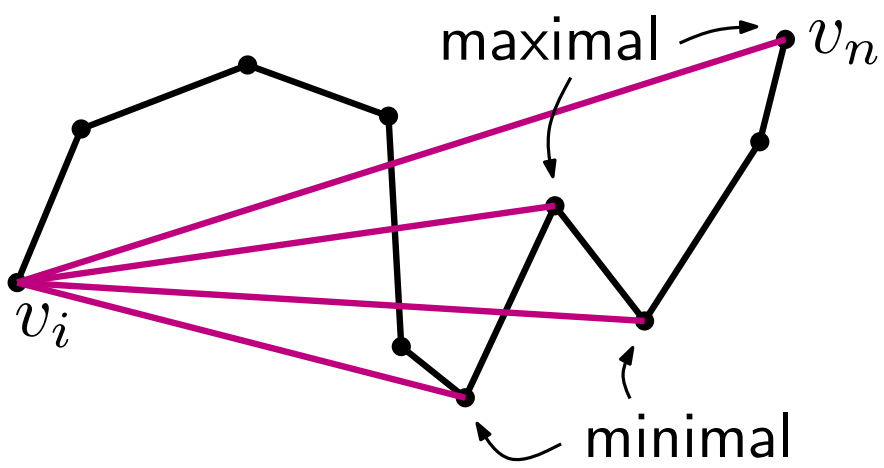
1) Tangentensegmente

- shortcut $v_i v_j$ ist **Tangente** an C_{in}, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen



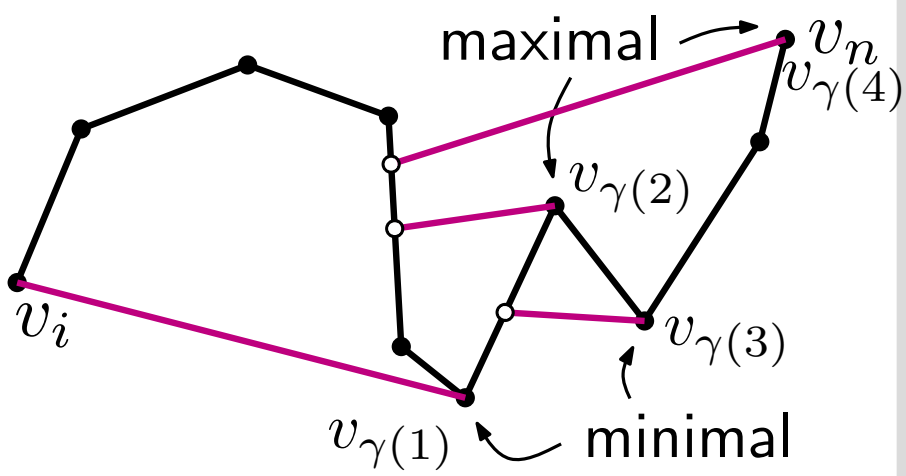
1) Tangentensegmente

- shortcut $v_i v_j$ ist **Tangente** an C_{in}, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen
- Tangente ist **minimal/maximal**, wenn v_{j-1} oberhalb/unterhalb $v_i v_j$



1) Tangentensegmente

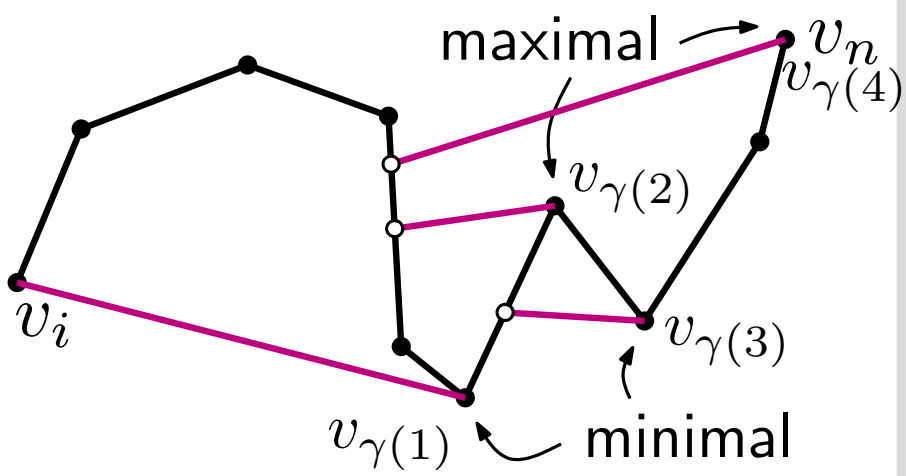
- shortcut $v_i v_j$ ist **Tangente** an C_{in}, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen
- Tangente ist **minimal/maximal**, wenn v_{j-1} oberhalb/unterhalb $v_i v_j$
- **Tangentensegment** $\overline{w_j v_j}$, wobei w_j rechtester Punkt in $C_{in} \cap \overline{v_i v_j}$



- Tangenten $v_i v_{\gamma(1)}, \ldots, v_i v_{\gamma(r)}$ definieren Unterteilung S_i mit r Facetten
- Knoten mit höchstem Index jeder Facette definiert Tangentensegment und Facettenindex

1) Tangentensegmente

- shortcut $v_i v_j$ ist **Tangente** an C_{in}, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen
- Tangente ist **minimal/maximal**, wenn v_{j-1} oberhalb/unterhalb $v_i v_j$
- **Tangentensegment** $\overline{w_j v_j}$, wobei w_j rechtester Punkt in $C_{in} \cap \overline{v_i v_j}$

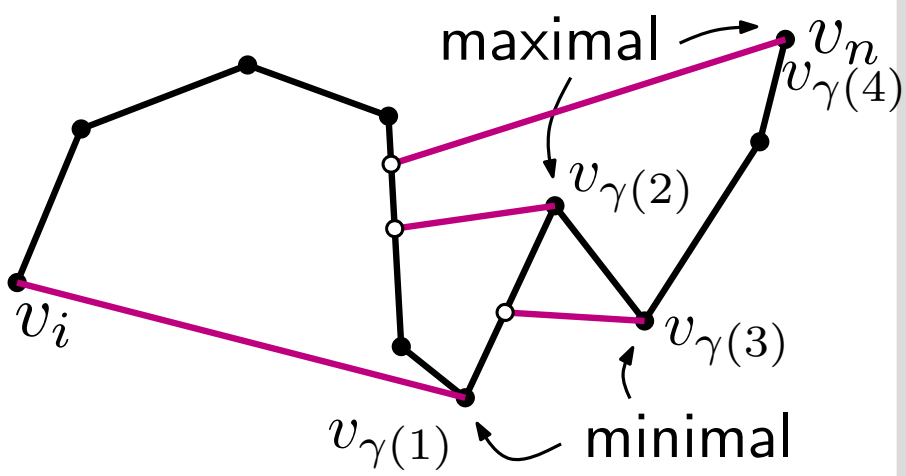


- Tangenten $v_i v_{\gamma(1)}, \ldots, v_i v_{\gamma(r)}$ definieren Unterteilung S_i mit r Facetten
- Knoten mit höchstem Index jeder Facette definiert Tangentensegment und Facettenindex

Lemma 2: Jede begrenzte Facette f von S_i ist θ-monoton bzgl. v_i, d.h. $|r \cap f| \in \{0, 1\}$ für jeden Strahl r von v_i

1) Tangentensegmente

- shortcut $v_i v_j$ ist **Tangente** an C_{in}, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen
- Tangente ist **minimal/maximal**, wenn v_{j-1} oberhalb/unterhalb $v_i v_j$
- **Tangentensegment** $\overline{w_j v_j}$, wobei w_j rechtester Punkt in $C_{in} \cap \overline{v_i v_j}$



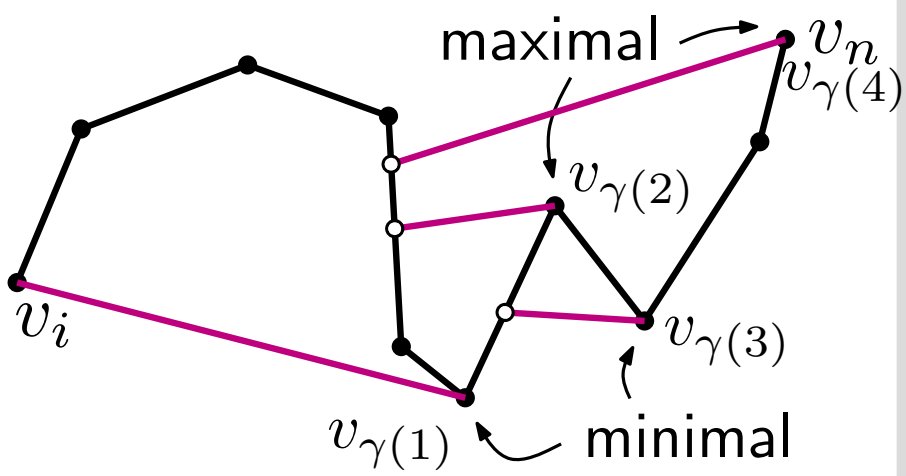
- Tangenten $v_i v_{\gamma(1)}, \ldots, v_i v_{\gamma(r)}$ definieren Unterteilung S_i mit r Facetten
- Knoten mit höchstem Index jeder Facette definiert Tangentensegment und Facettenindex

Lemma 2: Jede begrenzte Facette f von S_i ist θ-monoton bzgl. v_i, d.h. $|r \cap f| \in \{0, 1\}$ für jeden Strahl r von v_i

Lemma 3: Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen

1) Tangentensegmente

- shortcut v_iv_j ist **Tangente** an C_{in}, wenn v_{j-1} und v_{j+1} auf gleicher Seite von v_iv_j liegen
- Tangente ist **minimal/maximal**, wenn v_{j-1} oberhalb/unterhalb v_iv_j
- **Tangentensegment** $\overline{w_jv_j}$, wobei w_j rechtester Punkt in $C_{in} \cap \overline{v_iv_j}$



- Tangenten $v_iv_{\gamma(1)}, \ldots, v_iv_{\gamma(r)}$ definieren Unterteilung S_i mit r Facetten
- Knoten mit höchstem Index jeder Facette definiert Tangentensegment und Facettenindex

Lemma 2: Jede begrenzte Facette f von S_i ist θ-monoton bzgl. v_i, d.h. $|r \cap f| \in \{0, 1\}$ für jeden Strahl r von v_i

Lemma 3: Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen

Lemma 4: Jeder Strahl von v_i schneidet die Facetten von S_i in aufsteigender Reihenfolge

1) Tangentensegmente: Algorithmus

Karlsruhe Institute of Technology

Tangentensegmente(C, i)

Input: Kantenzug $C = (v_1, \ldots, v_n)$, Index $1 \leq i \leq n$
Output: Tangentensegmente bzgl. v_i
for $j = i + 1, \ldots, n$ **do**
  if $v_i v_j$ maximale Tangente **then**
    $k \leftarrow j - 1$
    while $v_i v_j \cap \overline{v_{k-1} v_k} = \emptyset$ **do**
      $k \leftarrow k - 1$
    $w_j \leftarrow v_i v_j \cap \overline{v_{k-1} v_k}$
  else if $v_i v_j$ minimale Tangente **then**
    ... /* analoges Vorgehen */

1) Tangentensegmente: Algorithmus

Karlsruhe Institute of Technology

Tangentensegmente(C, i)

Input: Kantenzug $C = (v_1, \ldots, v_n)$, Index $1 \leq i \leq n$
Output: Tangentensegmente bzgl. v_i
for $j = i + 1, \ldots, n$ **do**
  if $v_i v_j$ maximale Tangente **then**
    $k \leftarrow j - 1$
    while $v_i v_j \cap \overline{v_{k-1} v_k} = \emptyset$ **do**
      $k \leftarrow k - 1$
    $w_j \leftarrow v_i v_j \cap \overline{v_{k-1} v_k}$
  else if $v_i v_j$ minimale Tangente **then**
    ... /* analoges Vorgehen */

Laufzeit?

1) Tangentensegmente: Algorithmus

Tangentensegmente(C, i)

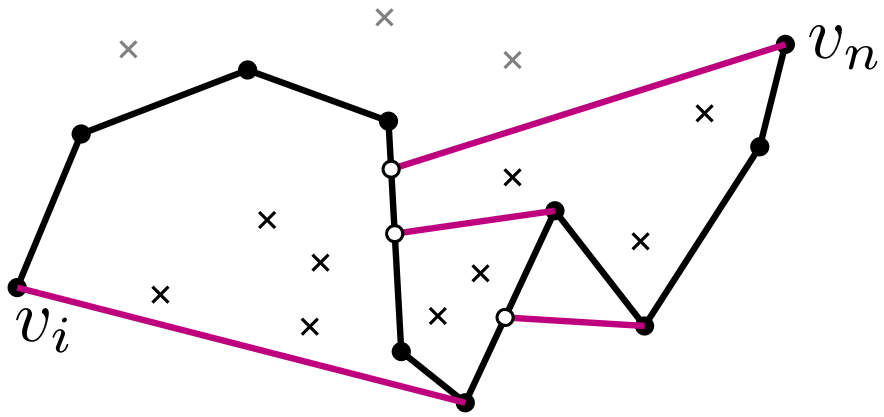
Input: Kantenzug $C = (v_1, \ldots, v_n)$, Index $1 \leq i \leq n$
Output: Tangentensegmente bzgl. v_i
for $j = i + 1, \ldots, n$ **do**
  if $v_i v_j$ maximale Tangente **then**
    $k \leftarrow j - 1$
    while $v_i v_j \cap \overline{v_{k-1} v_k} = \emptyset$ **do**
      $k \leftarrow k - 1$
      if $v_i v_k$ maximale Tangente **then**
        $k \leftarrow h$, wobei $v_{h-1} v_h$ Punkt w_k enthält
    $w_j \leftarrow v_i v_j \cap \overline{v_{k-1} v_k}$
  else if $v_i v_j$ minimale Tangente **then**
    ... `/* analoges Vorgehen */`

Laufzeit?

$\rightarrow$ Tangentensegmente(C, i) benötigt $O(n)$ Zeit

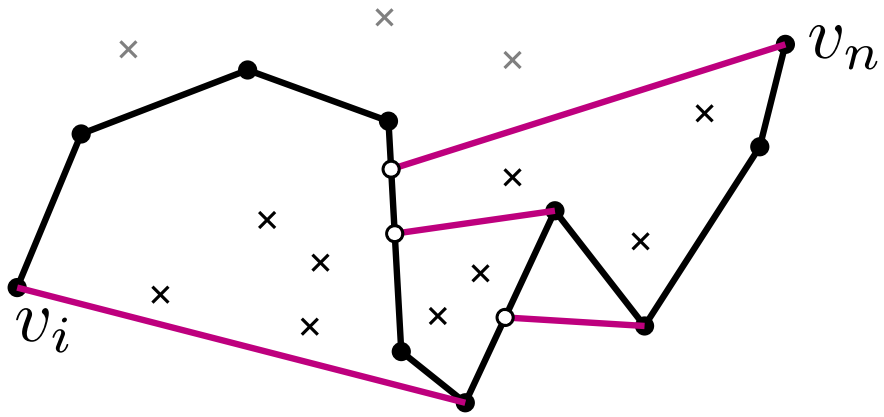
2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu



2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu

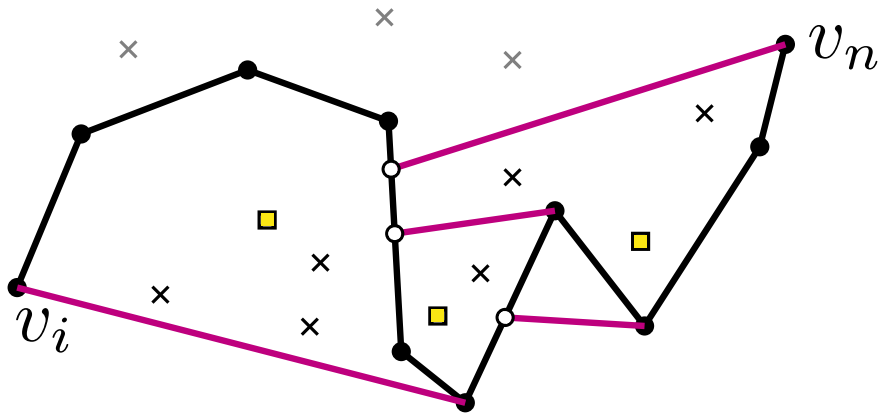


Sind alle verbleibenden Punkte nötig?

2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu
- für minimale Facette f, behalte Punkt p_f mit maximaler Steigung von $v_i p_f$
- für maximale Facette f, behalte Punkt p_f mit minimaler Steigung von $v_i p_f$

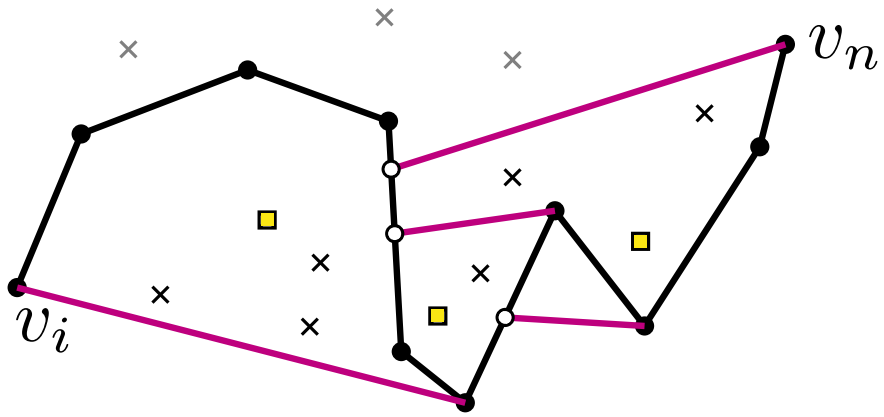
$\rightarrow$ verbleibende Punktmenge $P' \subseteq P$



2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu
- für minimale Facette f, behalte Punkt p_f mit maximaler Steigung von $v_i p_f$
- für maximale Facette f, behalte Punkt p_f mit minimaler Steigung von $v_i p_f$

$\rightarrow$ verbleibende Punktmenge $P' \subseteq P$



Lemma 5: Ein shortcut $v_i v_j$ ist konsistent für C_{ij} und P gdw. er konsistent für P' ist.

2) Punktzuweisung: Algorithmus

Karlsruhe Institute of Technology

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

2) Punktzuweisung: Algorithmus

Karlsruhe Institute of Technology

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

Laufzeit $O((n+m)\log(n+m))$

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

Laufzeit $O((n+m)\log(n+m))$ s. Übung!

3) Konsistenzberechnung

Input: Unterteilung S_i mit Facetten $f_1, \ldots, f_r$, Punkt $p_r \in P \cup \{\perp\}$
für jede Facette f_r, Index i

Output: Menge A aller konsistenten shortcuts von v_i

$Q \leftarrow$ double-ended queue für shortcuts $v_i v_j$ sortiert nach Steigung mit Pointern zw. $v_i v_j$ und v_j

$j \leftarrow i + 1$

for $h = 1, \ldots, r$ **do**

- **if** $p_h \neq \perp$ **then**
  - **if** f_h maximale Facette **then**
    - entferne shortcuts vom Anfang von Q
    - solange Steigung größer als $v_i p_h$
  - **else**
    - entferne shortcuts vom Ende von Q
    - solange Steigung kleiner als $v_i p_h$
- **while** $v_j \neq v_{\gamma(h)}$ **do**
  - **if** $v_i v_j \in Q$ **then** A.add($v_i v_j$); Q.remove($v_i v_j$)
  - $j \leftarrow j + 1$

3) Konsistenzberechnung

Input: Unterteilung S_i mit Facetten $f_1, \ldots, f_r$, Punkt $p_r \in P \cup \{\perp\}$
für jede Facette f_r, Index i

Output: Menge A aller konsistenten shortcuts von v_i

$Q \leftarrow$ double-ended queue für shortcuts $v_i v_j$ sortiert nach Steigung mit Pointern zw. $v_i v_j$ und v_j

$j \leftarrow i + 1$

for $h = 1, \ldots, r$ **do**

  if $p_h \neq \perp$ **then**

    if f_h maximale Facette **then**

      entferne shortcuts vom Anfang von Q
      solange Steigung größer als $v_i p_h$

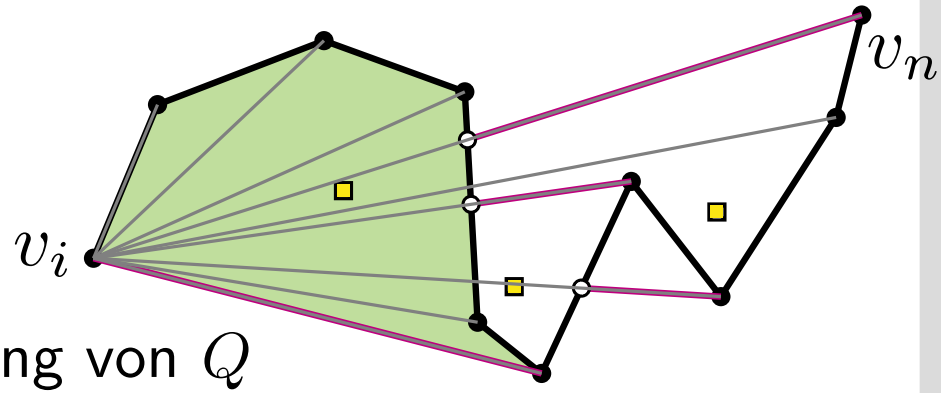
    else

      entferne shortcuts vom Ende von Q
      solange Steigung kleiner als $v_i p_h$

  while $v_j \neq v_{\gamma(h)}$ **do**

    if $v_i v_j \in Q$ **then** A.add($v_i v_j$); Q.remove($v_i v_j$)

    $j \leftarrow j + 1$



3) Konsistenzberechnung

Input: Unterteilung S_i mit Facetten $f_1, \ldots, f_r$, Punkt $p_r \in P \cup \{\perp\}$
für jede Facette f_r, Index i

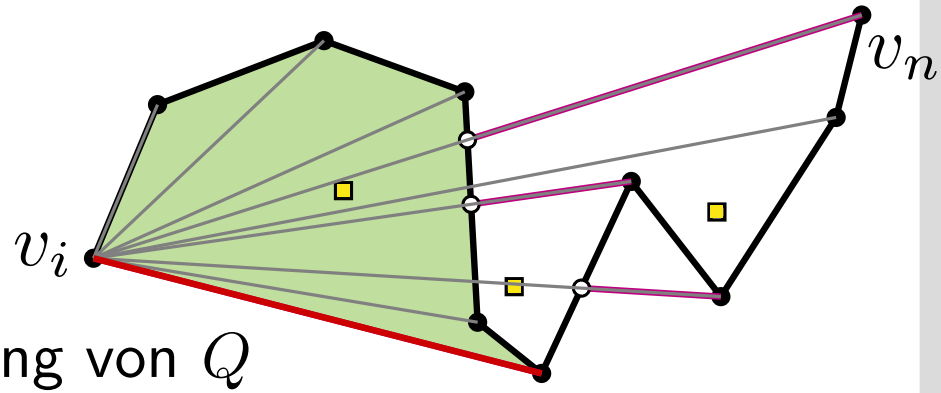
Output: Menge A aller konsistenten shortcuts von v_i

$Q \leftarrow$ double-ended queue für shortcuts $v_i v_j$ sortiert nach Steigung mit Pointern zw. $v_i v_j$ und v_j

$j \leftarrow i + 1$

for $h = 1, \ldots, r$ **do**

- **if** $p_h \neq \perp$ **then**
  - **if** f_h maximale Facette **then**
    - entferne shortcuts vom Anfang von Q
    - solange Steigung größer als $v_i p_h$
  - **else**
    - entferne shortcuts vom Ende von Q
    - solange Steigung kleiner als $v_i p_h$
  - **while** $v_j \neq v_{\gamma(h)}$ **do**
    - **if** $v_i v_j \in Q$ **then** A.add($v_i v_j$); Q.remove($v_i v_j$)
    - $j \leftarrow j + 1$



3) Konsistenzberechnung

Input: Unterteilung S_i mit Facetten $f_1, \ldots, f_r$, Punkt $p_r \in P \cup \{\perp\}$
für jede Facette f_r, Index i

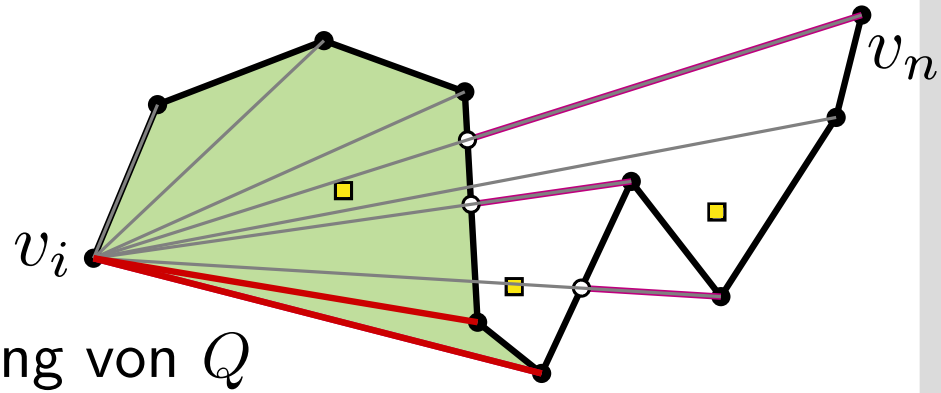
Output: Menge A aller konsistenten shortcuts von v_i

$Q \leftarrow$ double-ended queue für shortcuts $v_i v_j$ sortiert nach Steigung mit Pointern zw. $v_i v_j$ und v_j

$j \leftarrow i + 1$

for $h = 1, \ldots, r$ **do**

- **if** $p_h \neq \perp$ **then**
  - **if** f_h maximale Facette **then**
    - entferne shortcuts vom Anfang von Q
    - solange Steigung größer als $v_i p_h$
  - **else**
    - entferne shortcuts vom Ende von Q
    - solange Steigung kleiner als $v_i p_h$
  - **while** $v_j \neq v_{\gamma(h)}$ **do**
    - **if** $v_i v_j \in Q$ **then** A.add($v_i v_j$); Q.remove($v_i v_j$)
    - $j \leftarrow j + 1$



3) Konsistenzberechnung

Input: Unterteilung S_i mit Facetten $f_1, \ldots, f_r$, Punkt $p_r \in P \cup \{\perp\}$
für jede Facette f_r, Index i

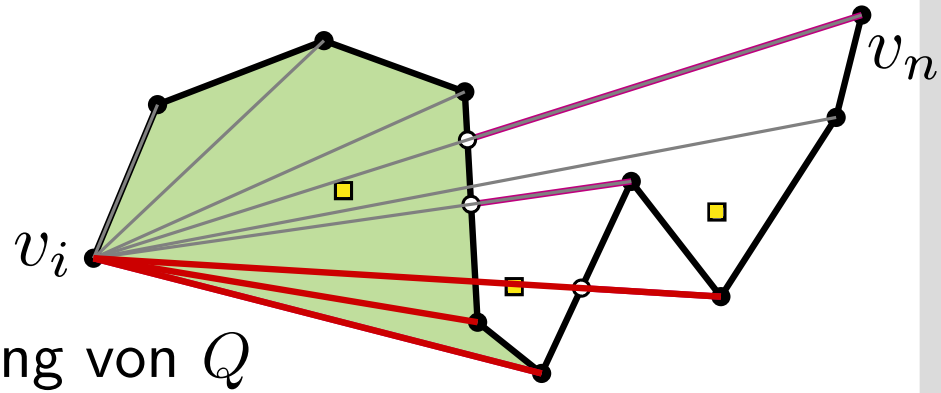
Output: Menge A aller konsistenten shortcuts von v_i

$Q \leftarrow$ double-ended queue für shortcuts $v_i v_j$ sortiert nach Steigung mit Pointern zw. $v_i v_j$ und v_j

$j \leftarrow i + 1$

for $h = 1, \ldots, r$ **do**

- **if** $p_h \neq \perp$ **then**
  - **if** f_h maximale Facette **then**
    - entferne shortcuts vom Anfang von Q
    - solange Steigung größer als $v_i p_h$
  - **else**
    - entferne shortcuts vom Ende von Q
    - solange Steigung kleiner als $v_i p_h$
  - **while** $v_j \neq v_{\gamma(h)}$ **do**
    - **if** $v_i v_j \in Q$ **then** A.add($v_i v_j$); Q.remove($v_i v_j$)
    - $j \leftarrow j + 1$



3) Konsistenzberechnung

Input: Unterteilung S_i mit Facetten $f_1, \ldots, f_r$, Punkt $p_r \in P \cup \{\perp\}$
für jede Facette f_r, Index i

Output: Menge A aller konsistenten shortcuts von v_i

$Q \leftarrow$ double-ended queue für shortcuts $v_i v_j$ sortiert nach Steigung mit Pointern zw. $v_i v_j$ und v_j

$j \leftarrow i + 1$

for $h = 1, \ldots, r$ **do**

  if $p_h \neq \perp$ **then**

    if f_h maximale Facette **then**

      entferne shortcuts vom Anfang von Q

      solange Steigung größer als $v_i p_h$

    else

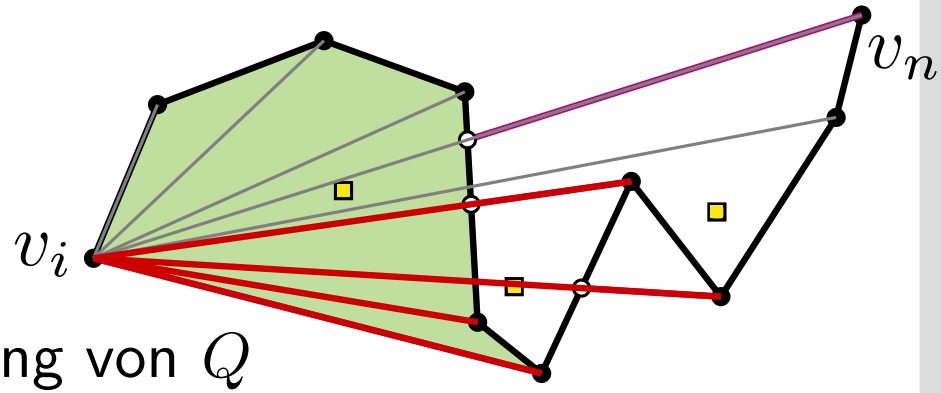
      entferne shortcuts vom Ende von Q

      solange Steigung kleiner als $v_i p_h$

  while $v_j \neq v_{\gamma(h)}$ **do**

    if $v_i v_j \in Q$ **then** A.add($v_i v_j$); Q.remove($v_i v_j$)

    $j \leftarrow j + 1$



3) Konsistenzberechnung

Input: Unterteilung S_i mit Facetten $f_1, \ldots, f_r$, Punkt $p_r \in P \cup \{\perp\}$
für jede Facette f_r, Index i

Output: Menge A aller konsistenten shortcuts von v_i

$Q \leftarrow$ double-ended queue für shortcuts $v_i v_j$ sortiert nach Steigung mit Pointern zw. $v_i v_j$ und v_j

$j \leftarrow i + 1$

for $h = 1, \ldots, r$ **do**

  if $p_h \neq \perp$ **then**

    if f_h maximale Facette **then**

      entferne shortcuts vom Anfang von Q
      solange Steigung größer als $v_i p_h$

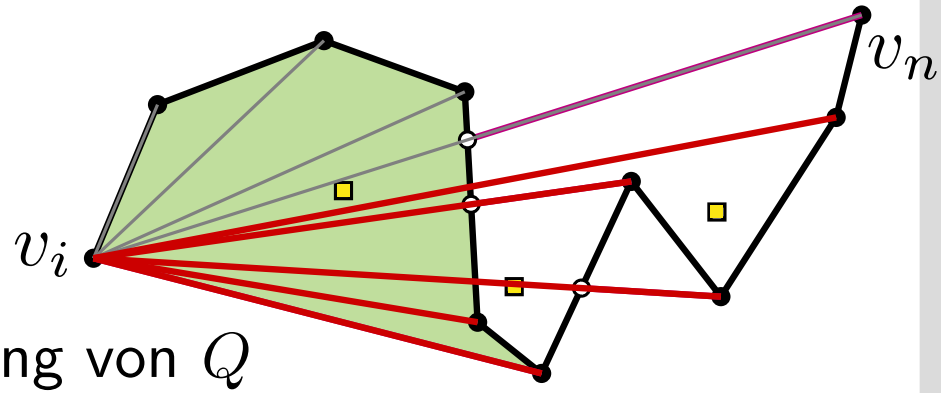
    else

      entferne shortcuts vom Ende von Q
      solange Steigung kleiner als $v_i p_h$

  while $v_j \neq v_{\gamma(h)}$ **do**

    if $v_i v_j \in Q$ **then** A.add($v_i v_j$); Q.remove($v_i v_j$)

    $j \leftarrow j + 1$



3) Konsistenzberechnung

Input: Unterteilung S_i mit Facetten $f_1, \ldots, f_r$, Punkt $p_r \in P \cup \{\perp\}$
für jede Facette f_r, Index i

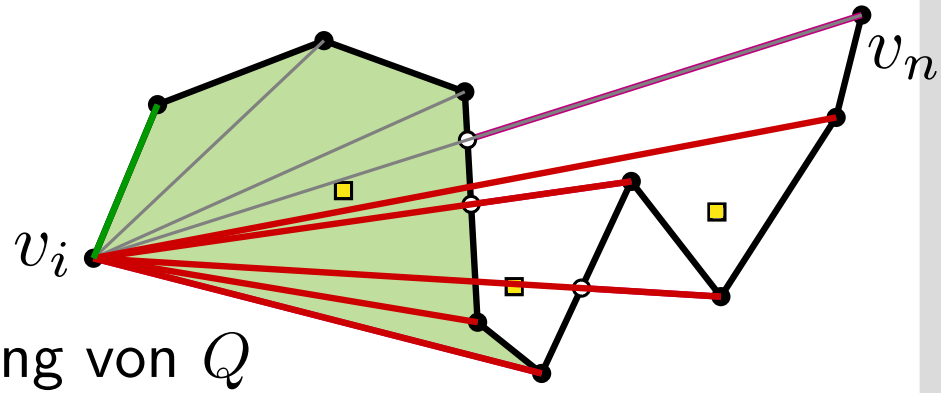
Output: Menge A aller konsistenten shortcuts von v_i

$Q \leftarrow$ double-ended queue für shortcuts $v_i v_j$ sortiert nach Steigung mit Pointern zw. $v_i v_j$ und v_j

$j \leftarrow i + 1$

for $h = 1, \ldots, r$ **do**

- **if** $p_h \neq \perp$ **then**
  - **if** f_h maximale Facette **then**
    - entferne shortcuts vom Anfang von Q
    - solange Steigung größer als $v_i p_h$
  - **else**
    - entferne shortcuts vom Ende von Q
    - solange Steigung kleiner als $v_i p_h$
  - **while** $v_j \neq v_{\gamma(h)}$ **do**
    - **if** $v_i v_j \in Q$ **then** A.add($v_i v_j$); Q.remove($v_i v_j$)
    - $j \leftarrow j + 1$



3) Konsistenzberechnung

Input: Unterteilung S_i mit Facetten $f_1, \ldots, f_r$, Punkt $p_r \in P \cup \{\perp\}$
für jede Facette f_r, Index i

Output: Menge A aller konsistenten shortcuts von v_i

$Q \leftarrow$ double-ended queue für shortcuts $v_i v_j$ sortiert nach Steigung mit Pointern zw. $v_i v_j$ und v_j

$j \leftarrow i + 1$

for $h = 1, \ldots, r$ **do**

  if $p_h \neq \perp$ **then**

    if f_h maximale Facette **then**

      entferne shortcuts vom Anfang von Q
      solange Steigung größer als $v_i p_h$

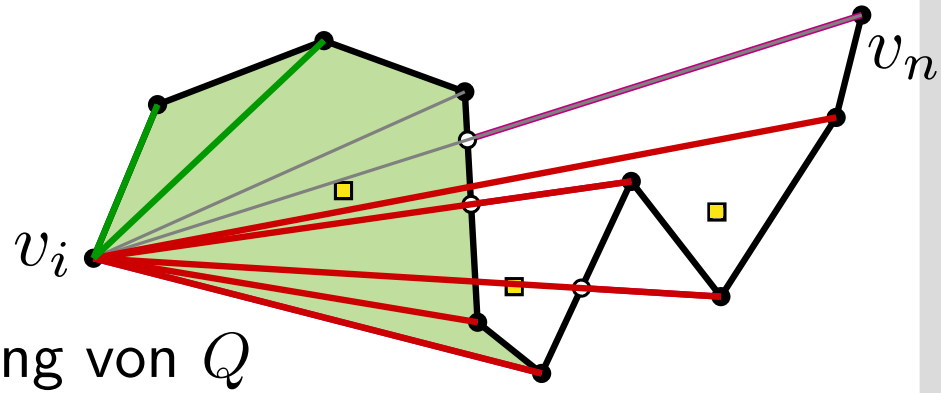
    else

      entferne shortcuts vom Ende von Q
      solange Steigung kleiner als $v_i p_h$

  while $v_j \neq v_{\gamma(h)}$ **do**

    if $v_i v_j \in Q$ **then** A.add($v_i v_j$); Q.remove($v_i v_j$)

    $j \leftarrow j + 1$



3) Konsistenzberechnung

Input: Unterteilung S_i mit Facetten $f_1, \ldots, f_r$, Punkt $p_r \in P \cup \{\perp\}$
für jede Facette f_r, Index i

Output: Menge A aller konsistenten shortcuts von v_i

$Q \leftarrow$ double-ended queue für shortcuts $v_i v_j$ sortiert nach Steigung mit Pointern zw. $v_i v_j$ und v_j

$j \leftarrow i + 1$

for $h = 1, \ldots, r$ **do**

  if $p_h \neq \perp$ **then**

    if f_h maximale Facette **then**

      entferne shortcuts vom Anfang von Q
      solange Steigung größer als $v_i p_h$

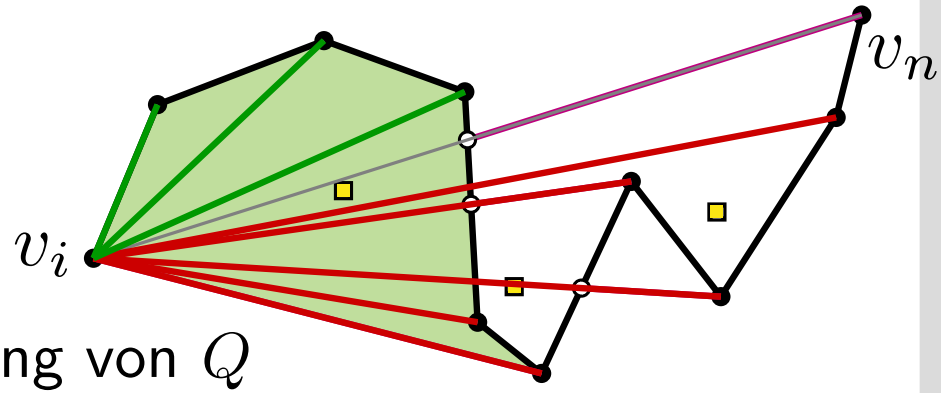
    else

      entferne shortcuts vom Ende von Q
      solange Steigung kleiner als $v_i p_h$

  while $v_j \neq v_{\gamma(h)}$ **do**

    if $v_i v_j \in Q$ **then** A.add($v_i v_j$); Q.remove($v_i v_j$)

    $j \leftarrow j + 1$



3) Konsistenzberechnung

Input: Unterteilung S_i mit Facetten $f_1, \ldots, f_r$, Punkt $p_r \in P \cup \{\perp\}$
für jede Facette f_r, Index i

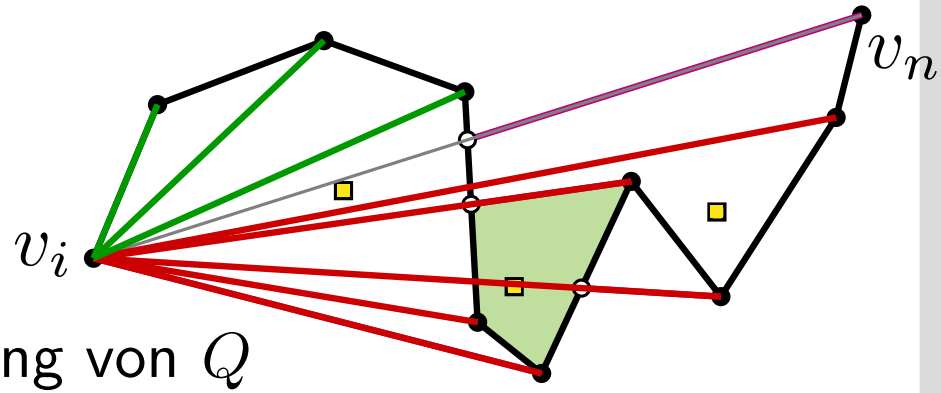
Output: Menge A aller konsistenten shortcuts von v_i

$Q \leftarrow$ double-ended queue für shortcuts $v_i v_j$ sortiert nach Steigung mit Pointern zw. $v_i v_j$ und v_j

$j \leftarrow i + 1$

for $h = 1, \ldots, r$ **do**

- **if** $p_h \neq \perp$ **then**
  - **if** f_h maximale Facette **then**
    - entferne shortcuts vom Anfang von Q
    - solange Steigung größer als $v_i p_h$
  - **else**
    - entferne shortcuts vom Ende von Q
    - solange Steigung kleiner als $v_i p_h$
- **while** $v_j \neq v_{\gamma(h)}$ **do**
  - **if** $v_i v_j \in Q$ **then** A.add($v_i v_j$); Q.remove($v_i v_j$)
  - $j \leftarrow j + 1$



3) Konsistenzberechnung

Input: Unterteilung S_i mit Facetten $f_1, \ldots, f_r$, Punkt $p_r \in P \cup \{\perp\}$
für jede Facette f_r, Index i

Output: Menge A aller konsistenten shortcuts von v_i

$Q \leftarrow$ double-ended queue für shortcuts $v_i v_j$ sortiert nach Steigung mit Pointern zw. $v_i v_j$ und v_j

$j \leftarrow i + 1$

for $h = 1, \ldots, r$ **do**

  if $p_h \neq \perp$ **then**

    if f_h maximale Facette **then**

      entferne shortcuts vom Anfang von Q
      solange Steigung größer als $v_i p_h$

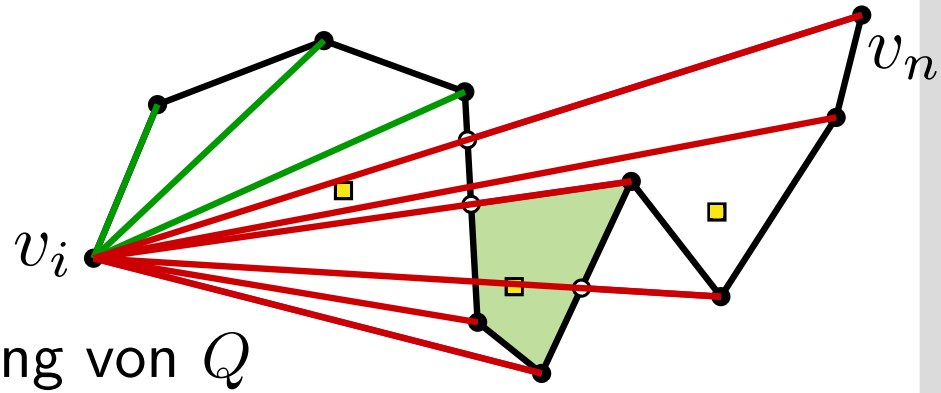
    else

      entferne shortcuts vom Ende von Q
      solange Steigung kleiner als $v_i p_h$

  while $v_j \neq v_{\gamma(h)}$ **do**

    if $v_i v_j \in Q$ **then** A.add($v_i v_j$); Q.remove($v_i v_j$)

    $j \leftarrow j + 1$



3) Konsistenzberechnung

Input: Unterteilung S_i mit Facetten $f_1, \ldots, f_r$, Punkt $p_r \in P \cup \{\perp\}$

für jede Facette f_h, Index i

Output: Menge A aller konsistenten shortcuts von v_i

$Q \leftarrow$ double-ended queue für shortcuts $v_i v_j$ sortiert nach Steigung mit Pointern zw. $v_i v_j$ und v_j

$j \leftarrow i + 1$

for $h = 1, \ldots, r$ **do**

  if $p_h \neq \perp$ **then**

    if f_h maximale Facette **then**

      entferne shortcuts vom Anfang von Q

      solange Steigung größer als $v_i p_h$

    else

      entferne shortcuts vom Ende von Q

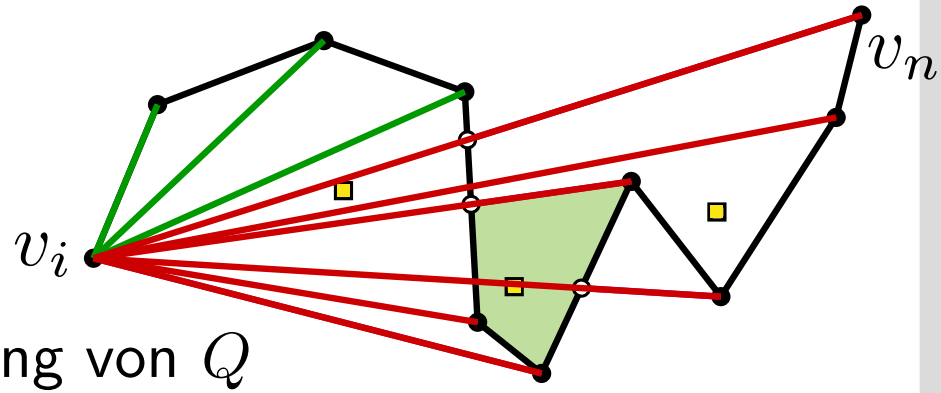
      solange Steigung kleiner als $v_i p_h$

  while $v_j \neq v_{\gamma(h)}$ **do**

    if $v_i v_j \in Q$ **then** A.add($v_i v_j$); Q.remove($v_i v_j$)

    $j \leftarrow j + 1$

Lemma 6: Die Berechnung der shortcuts in A ist korrekt und benötigt $O(n \log n)$ Zeit.



Zusammenfassung

Lemma 7: Für x-monotonen Kantenzug C mit n Knoten und Menge P von m Punkten können alle konsistenten shortcuts für einen Knoten v von C in Zeit $O((n+m)\log n)$ berechnet werden.

Zusammenfassung

Lemma 7: Für x-monotonen Kantenzug C mit n Knoten und Menge P von m Punkten können alle konsistenten shortcuts für einen Knoten v von C in Zeit $O((n+m)\log n)$ berechnet werden.

Satz 1: Für x-monotonen Kantenzug C mit n Knoten, Menge P von m Punkten und Fehlerschranke ε kann die kleinste konsistente Vereinfachung C' mit Fehler $\leq \varepsilon$ in Zeit $O(n(n+m)\log n)$ berechnet werden.

Erweiterung für beliebige Kantenzüge

Idee: finde für jeden Knoten v_i einen Endindex $n(i)$, so dass der x-monotone Algorithmus auf $C_{in(i)}$ anwendbar ist.

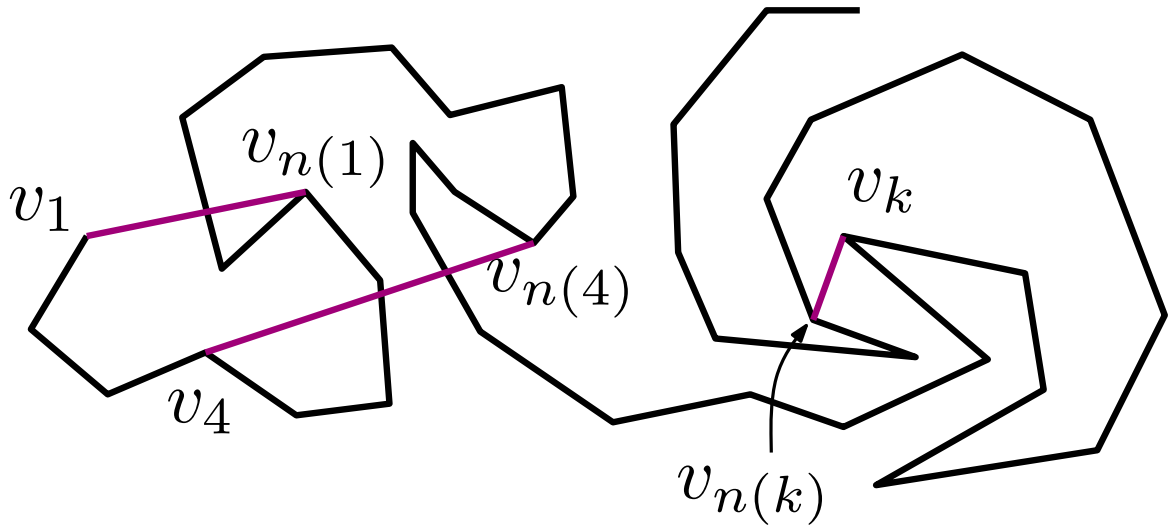
Erweiterung für beliebige Kantenzüge

Idee: finde für jeden Knoten v_i einen Endindex $n(i)$, so dass der x-monotone Algorithmus auf $C_{in(i)}$ anwendbar ist.

kritische Eigenschaften für den Algorithmus:

- θ-Monotonität der Facetten
- Strahlen schneiden Facetten in aufsteigender Reihenfolge

Bestimme maximales $n(i)$ so, dass keine Rückwärts-Tangenten und keine Kreise vorkommen.



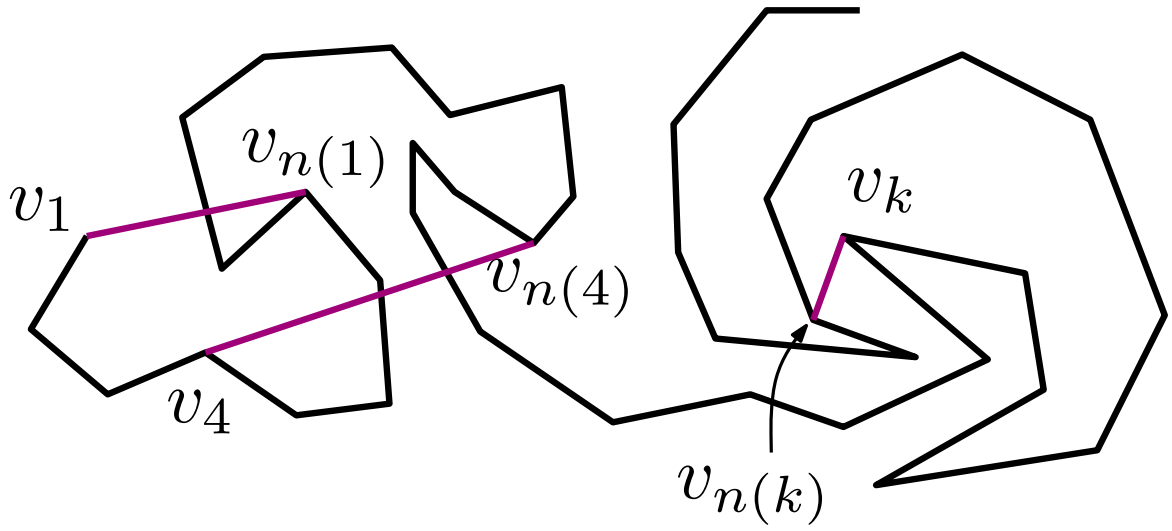
Erweiterung für beliebige Kantenzüge

Idee: finde für jeden Knoten v_i einen Endindex $n(i)$, so dass der x-monotone Algorithmus auf $C_{in(i)}$ anwendbar ist.

kritische Eigenschaften für den Algorithmus:

- θ-Monotonität der Facetten
- Strahlen schneiden Facetten in aufsteigender Reihenfolge

Bestimme maximales $n(i)$ so, dass keine Rückwärts-Tangenten und keine Kreise vorkommen.



Für $n(i) < n$ kann Minimalität der Vereinfachung nicht mehr garantiert werden!

Zusammenfassung

Satz 2: Für einen einfachen Kantenzug C mit n Knoten, Menge P von m Punkten und Fehlerschranke ε kann eine einfache, konsistente Vereinfachung C' mit Fehler $\leq \varepsilon$ in Zeit $O(n(n+m)\log n)$ berechnet werden.

Zusammenfassung

Satz 2: Für einen einfachen Kantenzug C mit n Knoten, Menge P von m Punkten und Fehlerschranke ε kann eine einfache, konsistente Vereinfachung C' mit Fehler $\leq \varepsilon$ in Zeit $O(n(n+m)\log n)$ berechnet werden.

Wendet man das Verfahren auf alle maximalen Pfade einer Unterteilung an, so erhält man eine konsistente Vereinfachung der Unterteilung mit Fehler $\leq \varepsilon$.

Schnitte mit dem eigenen oder anderen Kantenzügen können durch Hinzufügen deren Punkte zu P verhindert werden.

Zusammenfassung

Satz 2: Für einen einfachen Kantenzug C mit n Knoten, Menge P von m Punkten und Fehlerschranke ε kann eine einfache, konsistente Vereinfachung C' mit Fehler $\leq \varepsilon$ in Zeit $O(n(n+m)\log n)$ berechnet werden.

Wendet man das Verfahren auf alle maximalen Pfade einer Unterteilung an, so erhält man eine konsistente Vereinfachung der Unterteilung mit Fehler $\leq \varepsilon$.

Schnitte mit dem eigenen oder anderen Kantenzügen können durch Hinzufügen deren Punkte zu P verhindert werden.

Bem: Lässt man die Beschränkung fallen, dass die Knoten der Vereinfachung eine Teilmenge der Eingabeknoten sein müssen, ist das Minimierungsproblem NP-schwer.

[Guibas et al. '93]