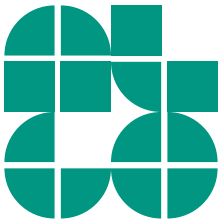
Vorlesung Algorithmische Kartografie

Einführung & Linienvereinfachung

INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

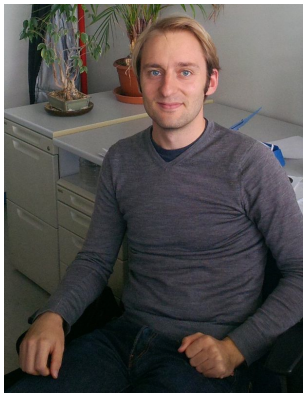
Benjamin Niedermann · **Martin Nöllenburg**

14.04.2015



AlgoKarto-Team

Dozenten

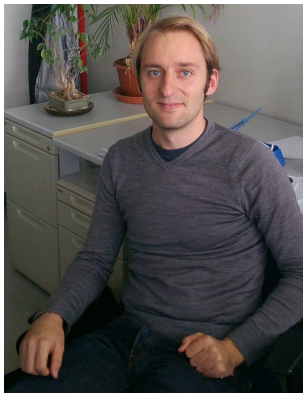


- Benjamin Niedermann
- niedermann@kit.edu
- Raum 309
- Sprechzeiten: individuell per Mail vereinbaren

- Martin Nöllenburg
- noellenburg@kit.edu
- Raum 319
- Sprechzeiten: individuell per Mail vereinbaren

AlgoKarto-Team

Dozenten



- Benjamin Niedermann
- niedermann@kit.edu
- Raum 309
- Sprechzeiten: individuell per Mail vereinbaren

- Martin Nöllenburg
- noellenburg@kit.edu
- Raum 319
- Sprechzeiten: individuell per Mail vereinbaren

Termine

- Di 9:45 – 11:15 Uhr, Raum 301
- Do 9:45 – 11:15 Uhr, Raum 301 (ab 23.04.)
- Ende der Vorlesung Ende Juni/Anfang Juli

Organisatorisches

Webseite

`i11www.iti.kit.edu/teaching/sommer2015/algokarto/index`

- aktuelle Informationen
- Vorlesungsfolien
- Literatur
- Zusatzmaterial

Organisatorisches

Webseite

`i11www.iti.kit.edu/teaching/sommer2015/algokarto/index`

- aktuelle Informationen
- Vorlesungsfolien
- Literatur
- Zusatzmaterial

Algorithmische Kartografie im Master-Studium

Bachelor Informatik

Algorithmen 1+2

Theoretische Grundlagen

Algorithmen für planare Graphen

Algorithm. Methoden für schwere Optimierungsprobleme

Master Informatik

Algorithmische Kartografie

Algorithmische Geometrie

Graphenvisualisierung

⋮

VF Algorithmentechnik, Theoretische Grundlagen, Computergrafik

Algorithmische Kartografie

Lernziele: Am Ende der Vorlesung können Sie

- Begriffe, Strukturen und Problemdefinitionen erklären
- behandelte Algorithmen ausführen, erklären und analysieren
- geeignete Algorithmen und Datenstrukturen auswählen und anpassen
- neue kartografische/geometrische Probleme analysieren und eigene effiziente Lösungen entwerfen

Algorithmische Kartografie

Lernziele: Am Ende der Vorlesung können Sie

- Begriffe, Strukturen und Problemdefinitionen erklären
- behandelte Algorithmen ausführen, erklären und analysieren
- geeignete Algorithmen und Datenstrukturen auswählen und anpassen
- neue kartografische/geometrische Probleme analysieren und eigene effiziente Lösungen entwerfen

Vorkenntnisse: Algorithmik und elementare Geometrie

hilfreich: Algorithmische Geometrie bzw. Algorithmen zur Visualisierung von Graphen

Algorithmische Kartografie

Lernziele: Am Ende der Vorlesung können Sie

- Begriffe, Strukturen und Problemdefinitionen erklären
- behandelte Algorithmen ausführen, erklären und analysieren
- geeignete Algorithmen und Datenstrukturen auswählen und anpassen
- neue kartografische/geometrische Probleme analysieren und eigene effiziente Lösungen entwerfen

Vorkenntnisse: Algorithmik und elementare Geometrie

hilfreich: Algorithmische Geometrie bzw. Algorithmen zur Visualisierung von Graphen

Anforderungen/Zeiteinteilung: 5LP = 150h

aktive Teilnahme an Vorlesung und Übung	ca. 70h
Projektarbeit	ca. 30h
Prüfungsvorbereitung	ca. 50h

Prüfbarkeit

Master Informatik

- Algorithm Engineering und Anwendungen (IN4INAEA, 5LP)
- Design und Analyse von Algorithmen (IN4INDAA, 5LP)
- Algorithmen der Computergrafik (IN4INACG, nur 3LP)
- Algorithmische Kartografie (IN4INAK, 5LP)

Master Informationswirtschaft

- Advanced Algorithms: Engineering and Applications (IW4INAALGOB)
- Advanced Algorithms: Design and Analysis (IW4INAADA)
- Algorithmen der Computergrafik (IW4INACG)

Prüfbarkeit

Master Informatik

- Algorithm Engineering und Anwendungen (IN4INAEA, 5LP)
- Design und Analyse von Algorithmen (IN4INDAA, 5LP)
- Algorithmen der Computergrafik (IN4INACG, nur 3LP)
- Algorithmische Kartografie (IN4INAK, 5LP)

Master Informationswirtschaft

- Advanced Algorithms: Engineering and Applications (IW4INAALGOB)
- Advanced Algorithms: Design and Analysis (IW4INAADA)
- Algorithmen der Computergrafik (IW4INACG)

Prüfungsnote

- 20-minütige mündliche Prüfung (80%)
- Projektarbeit (20%)

Vorlesungsbetrieb

Karlsruhe Institute of Technology

Ablauf

- die Bearbeitung von Übungsaufgaben ist in die regulären Vorlesungstermine integriert
- ca. 20–30 Min Übung pro Vorlesung
- aktive Teilnahme und Diskussionen sind erwünscht
- Programmierprojekt/Referat in kleinen Teams als Teil der Prüfungsleistung (Details und Themen nächste Woche)

Vorlesungsbetrieb

Ablauf

- die Bearbeitung von Übungsaufgaben ist in die regulären Vorlesungstermine integriert
- ca. 20–30 Min Übung pro Vorlesung
- aktive Teilnahme und Diskussionen sind erwünscht
- Programmierprojekt/Referat in kleinen Teams als Teil der Prüfungsleistung (Details und Themen nächste Woche)

Medien

- Nutzung von Folien und Tafel (meist für Beweise)
- eigene Notizen der Tafelanschriebe sinnvoll
- alternativ: Nachlesen in Originalquellen

Vorlesungsbetrieb

Ablauf

- die Bearbeitung von Übungsaufgaben ist in die regulären Vorlesungstermine integriert
- ca. 20–30 Min Übung pro Vorlesung
- aktive Teilnahme und Diskussionen sind erwünscht
- Programmierprojekt/Referat in kleinen Teams als Teil der Prüfungsleistung (Details und Themen nächste Woche)

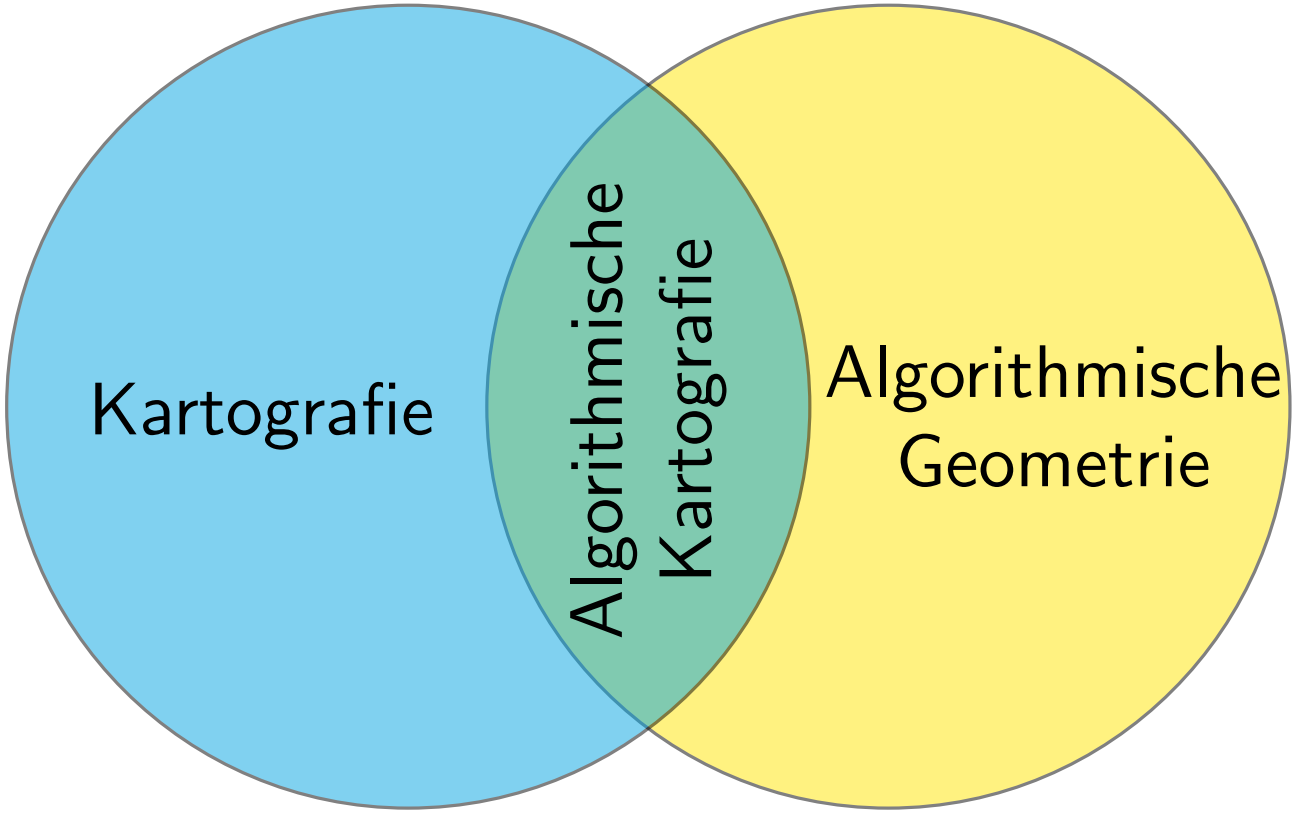
Medien

- Nutzung von Folien und Tafel (meist für Beweise)
- eigene Notizen der Tafelanschriebe sinnvoll
- alternativ: Nachlesen in Originalquellen

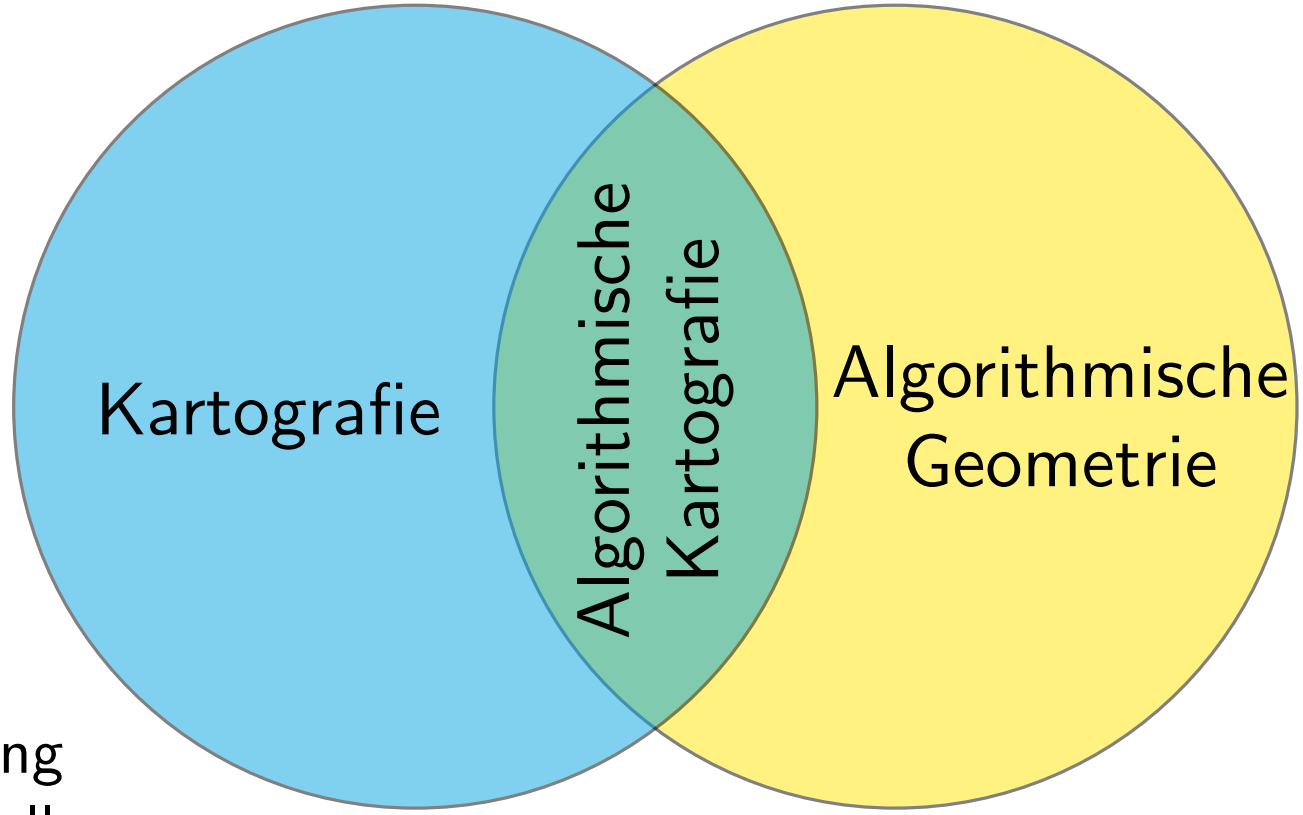
Organisatorische Fragen?

Was ist algorithmische Kartografie?

Was ist algorithmische Kartografie?



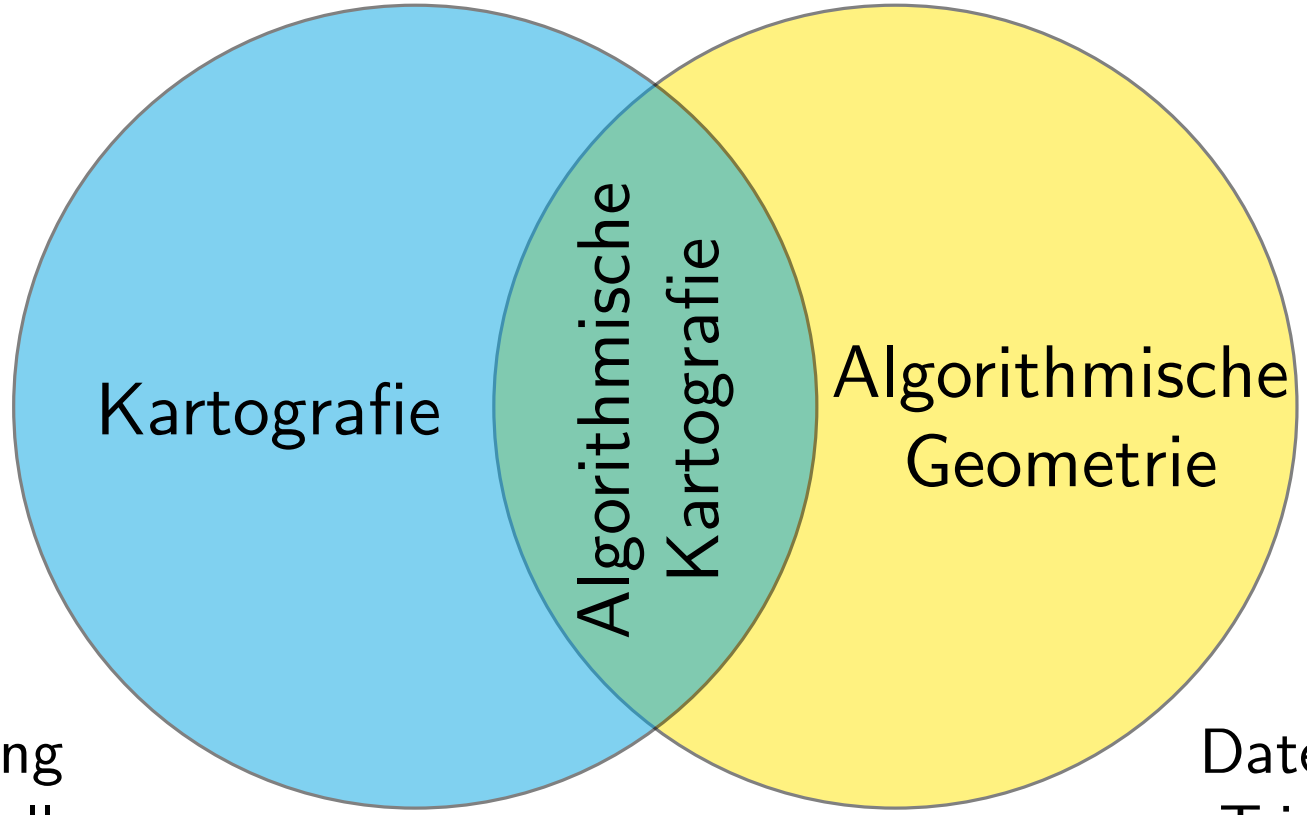
Was ist algorithmische Kartografie?



GIS
Fernerkundung
Geländemodelle
Kartenprojektionen
...

→ Studiengang Geodäsie und Geoinformatik

Was ist algorithmische Kartografie?



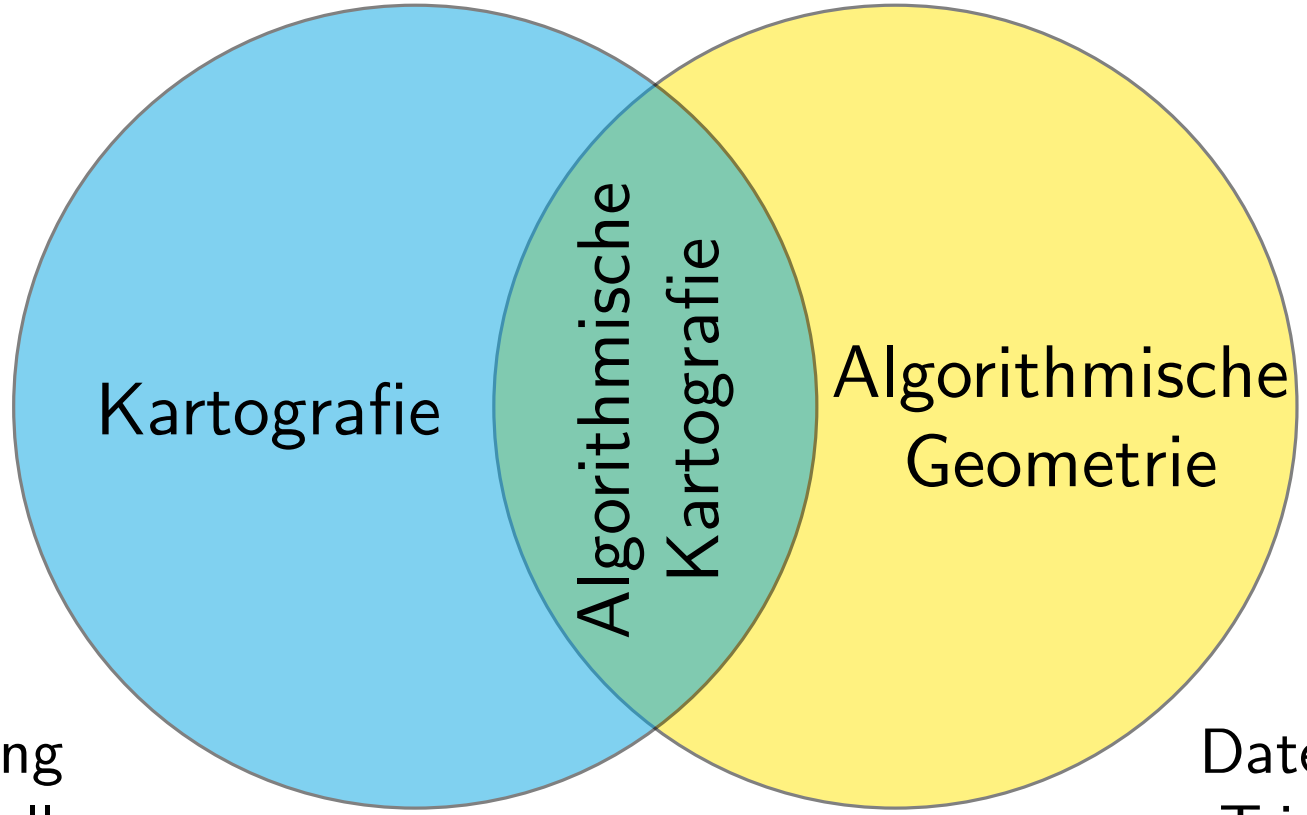
GIS
Fernerkundung
Geländemodelle
Kartenprojektionen
...

→ Studiengang Geodäsie und Geoinformatik

Algorithmen
Datenstrukturen
Triangulationen
Bereichsabfragen
Schnitte, Hüllen ...

→ Vorlesung Algorithmische Geometrie

Was ist algorithmische Kartografie?



GIS
Fernerkundung
Geländemodelle
Kartenprojektionen
...

→ Studiengang Geodäsie und Geoinformatik

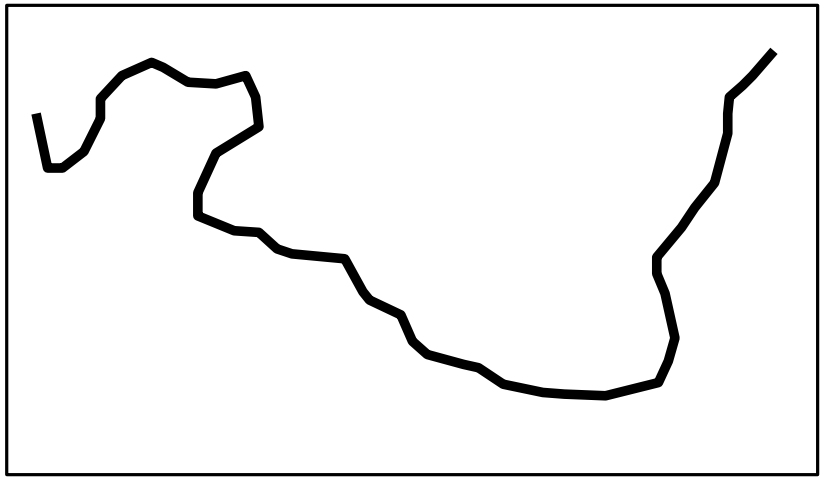
geometrische Algorithmen für kartografische Probleme

Algorithmen
Datenstrukturen
Triangulationen
Bereichsabfragen
Schnitte, Hüllen ...

→ Vorlesung Algorithmische Geometrie

Beispiel 1: Vereinfachung von Kantenzügen

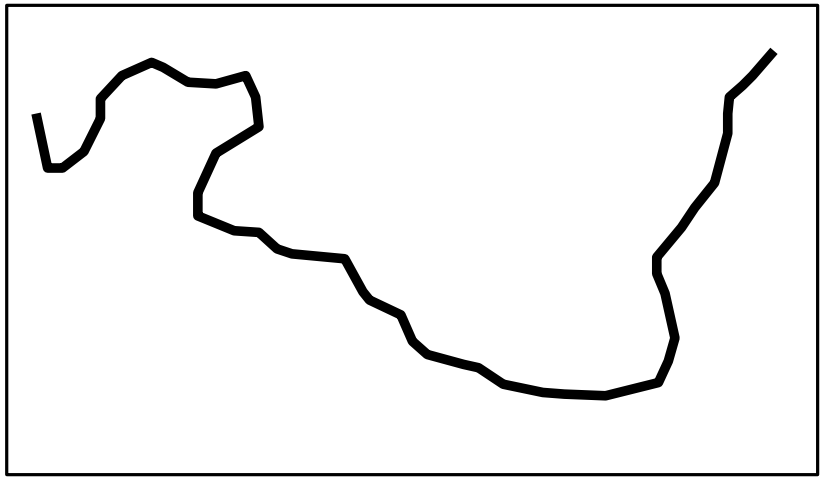
- viele Objekte in Landkarten werden durch Polygone oder Kantenzüge repräsentiert
- Detailgrad abhängig vom Maßstab (Generalisierung)



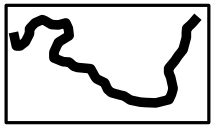
Maßstab $1 : X$

Beispiel 1: Vereinfachung von Kantenzügen

- viele Objekte in Landkarten werden durch Polygone oder Kantenzüge repräsentiert
- Detailgrad abhängig vom Maßstab (Generalisierung)



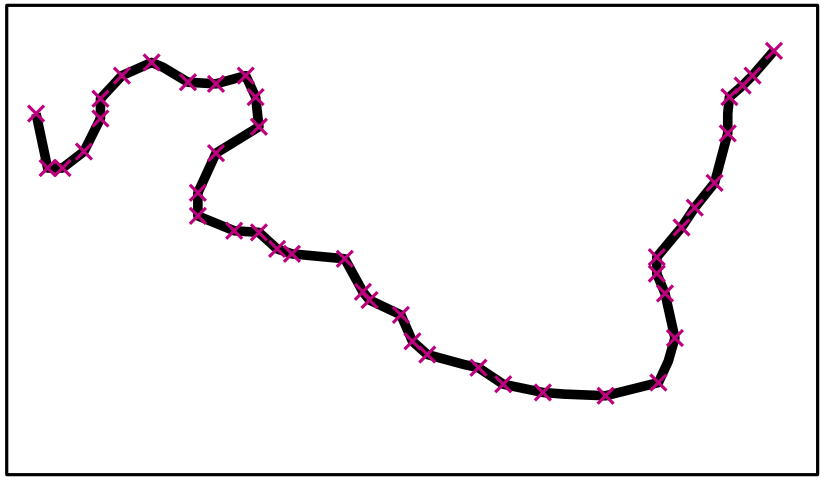
Maßstab $1 : X$



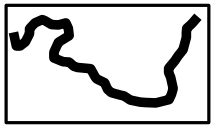
Maßstab $1 : (4X)$

Beispiel 1: Vereinfachung von Kantenzügen

- viele Objekte in Landkarten werden durch Polygone oder Kantenzüge repräsentiert
- Detailgrad abhängig vom Maßstab (Generalisierung)



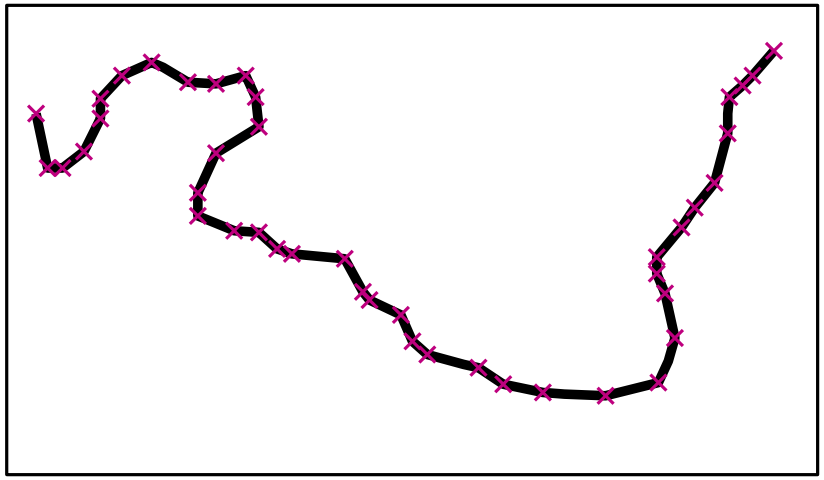
Maßstab $1 : X$



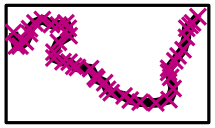
Maßstab $1 : (4X)$

Beispiel 1: Vereinfachung von Kantenzügen

- viele Objekte in Landkarten werden durch Polygone oder Kantenzüge repräsentiert
- Detailgrad abhängig vom Maßstab (Generalisierung)



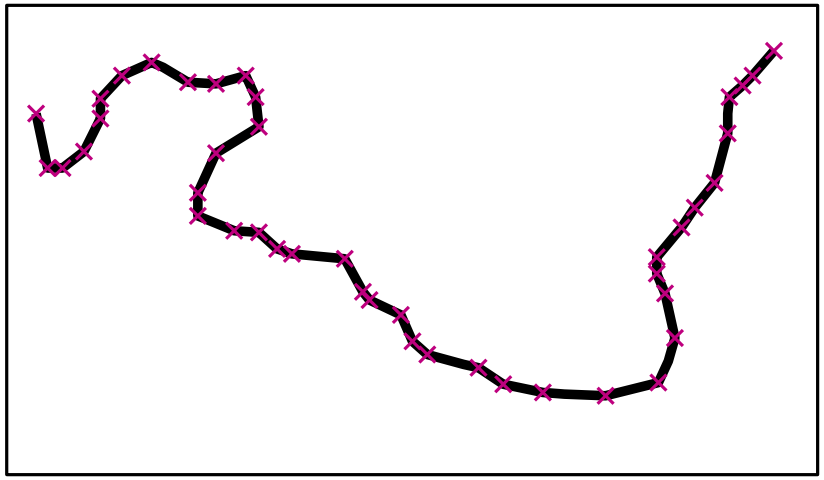
Maßstab $1 : X$



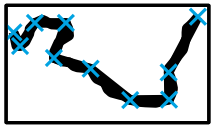
Maßstab $1 : (4X)$

Beispiel 1: Vereinfachung von Kantenzügen

- viele Objekte in Landkarten werden durch Polygone oder Kantenzüge repräsentiert
- Detailgrad abhängig vom Maßstab (Generalisierung)



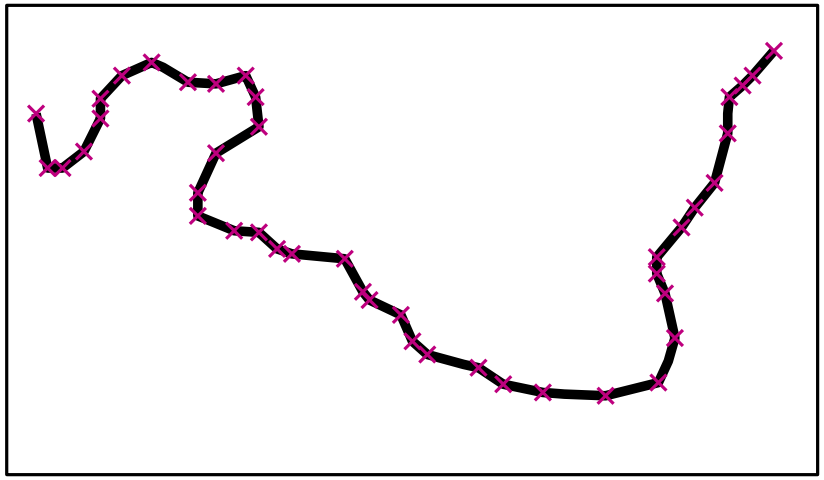
Maßstab $1 : X$



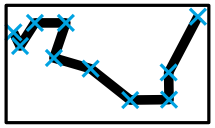
Maßstab $1 : (4X)$

Beispiel 1: Vereinfachung von Kantenzügen

- viele Objekte in Landkarten werden durch Polygone oder Kantenzüge repräsentiert
- Detailgrad abhängig vom Maßstab (Generalisierung)



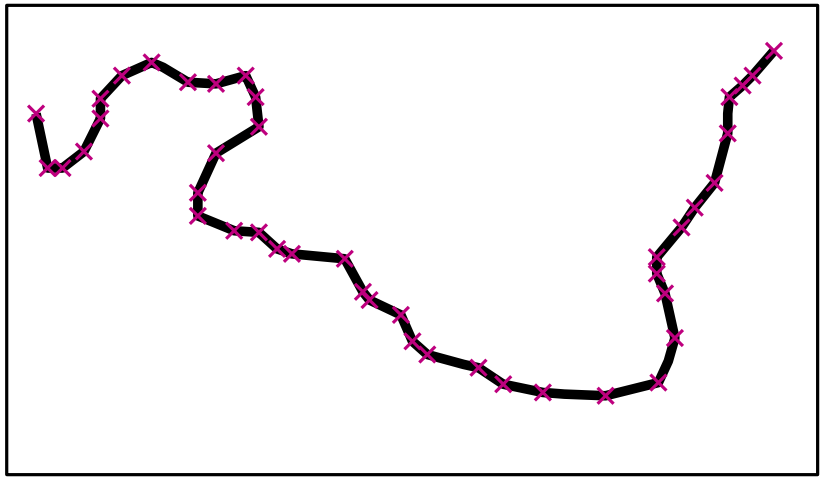
Maßstab $1 : X$



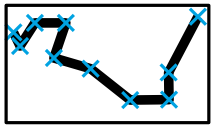
Maßstab $1 : (4X)$

Beispiel 1: Vereinfachung von Kantenzügen

- viele Objekte in Landkarten werden durch Polygone oder Kantenzüge repräsentiert
- Detailgrad abhängig vom Maßstab (Generalisierung)



Maßstab $1 : X$



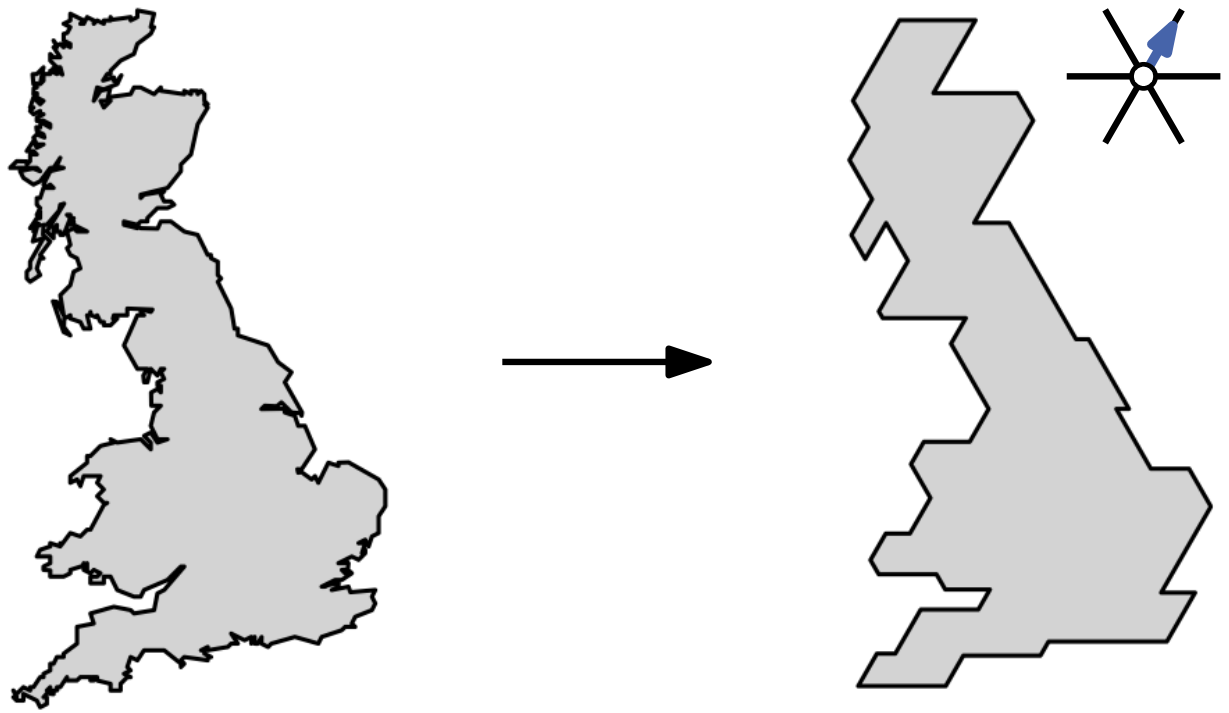
Maßstab $1 : (4X)$

geg: Kantenzug P

ges: Kantenzug Q mit weniger Knoten und kleinem Fehler $|P - Q|$

Beispiel 2: Schematisierung von Polygonen

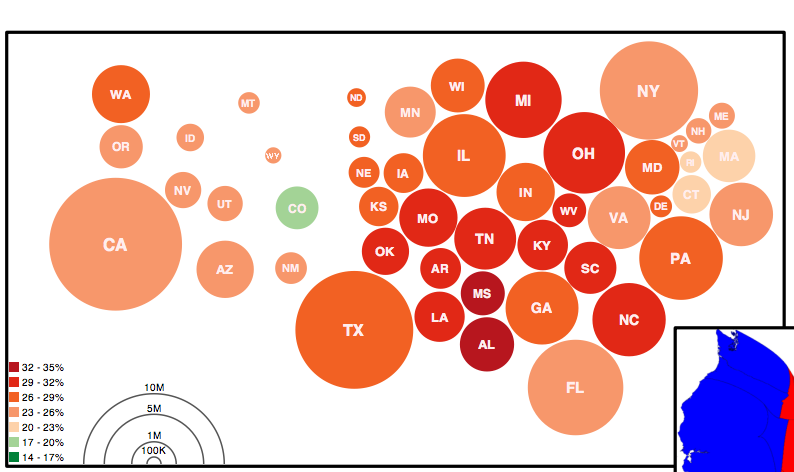
- schematische Karten nutzen oft eingeschränkte Kantenrichtungen
- Polygonflächen müssen geeignet schematisiert werden

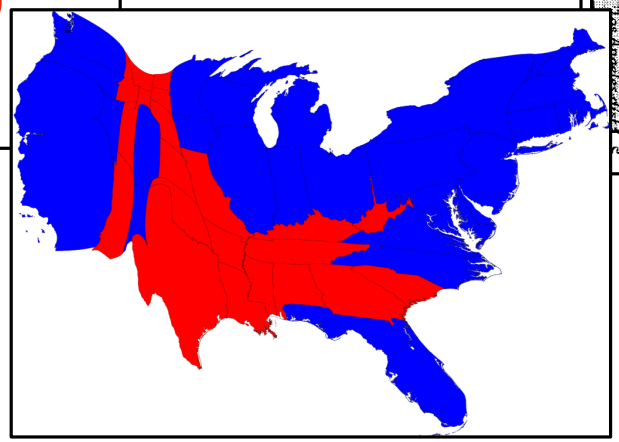


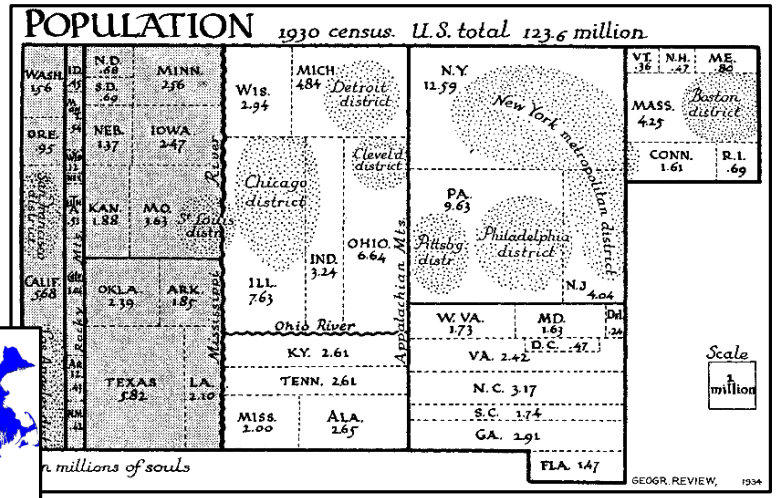
geg: Polygon P mit Fläche A, Richtungsmenge $\mathcal{C}$
ges: $\mathcal{C}$-orientiertes Polygon Q mit gleicher Fläche A und kleinem Fehler $|P - Q|$

Beispiel 3: Flächenkartogramme

- abstrakte statistische thematische Karten nutzen u.a. verzerrte proportionale Flächen
- proportionale Kontaktrepräsentation des Dualgraphs



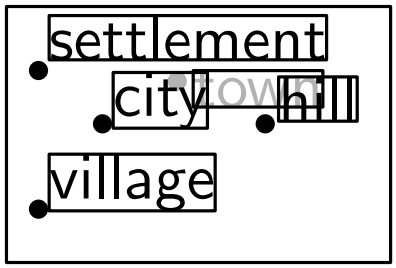




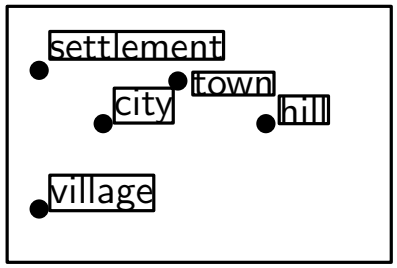
geg: gewichtete politische Karte (Unterteilung der Ebene)
ges: entsprechende verzerrte Karte, deren Flächen proportional zu den Gewichten sind

Beispiel 4: Beschriftung von Landkarten

- Objekte in Karten benötigen meist einen eindeutig zugeordneten Namen (Label)
- verschiedene interne und externe Labelpositionen möglich



MaxNumber



MaxSize

interne Label

externe Label

geg: Punktmenge P mit Labelmenge L

ges: gültige, optimale Beschriftung von P mit L, je nach Beschriftungsmodell

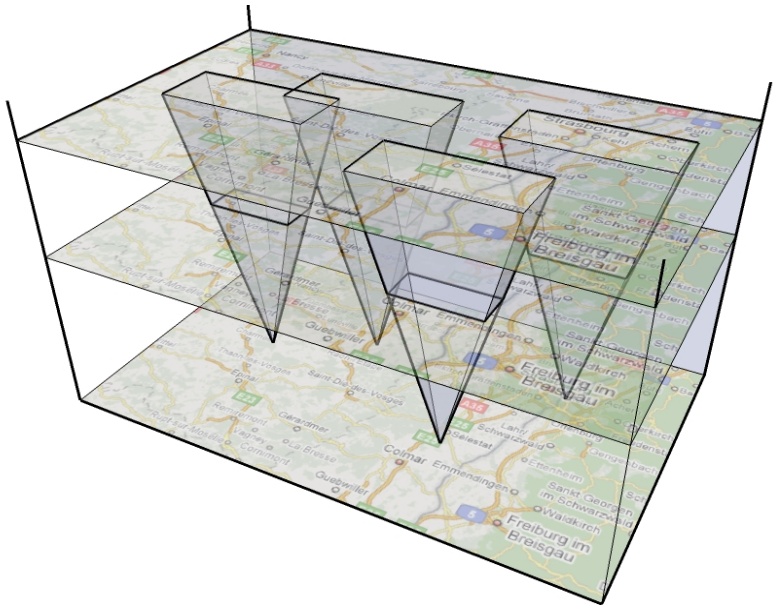
Beispiel 5: Dynamische Karten

- moderne elektronische Karten sind interaktiv & dynamisch
- Formen und Beschriftungen müssen sich kontinuierlich an die Kartenansicht anpassen

z.B. Beschriftung:

geg: Punktmenge P mit Labelmenge L

ges: gültige, optimale und **konsistente** Beschriftung von P mit L unter Zoom, Drehen etc.



Vorlesungsthemen

Ziel: Überblick über grundlegende und aktuelle algorithmische Forschungsthemen in der Kartografie

Ebene	Block	Termine	Themen
Karteninhalt	Linienzüge	3	Einführung Linienvereinfachung Konsistente Vereinfachung von Kantenzügen
	Flächen	5-6	Flächenaggregation Vereinfachung von Gebäudeumrissen Flächenkartogramme
Beschriftung	Statisch	2	Punktbeschriftung
			Straßenbeschriftung
	Dynamisch	6	Zoomen
			Rotieren
			Trajektorienbasierte Beschriftung
Geovisualisierung	Annotation	3	Beschriftung von bewegten Punkten
			Randbeschriftung
	Diagramme	1	Proportional Symbol Maps

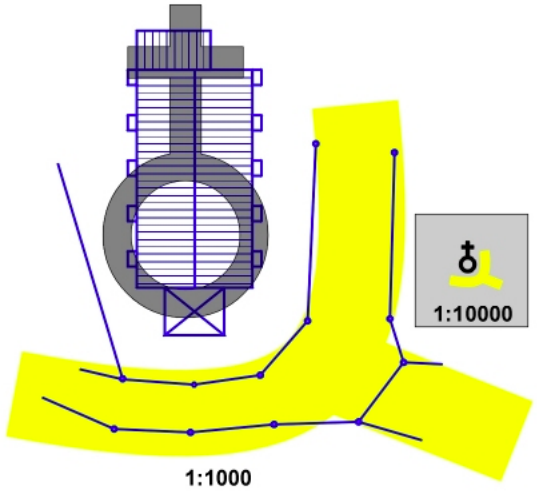
Algorithmen zur Linienvereinfachung

Generalisierung

Generalisierung in der Kartografie bezeichnet Verfahren zur Vereinfachung des Karteninhalts für die Verbesserung der Lesbarkeit in kleineren Maßstäben.

Beispiele:

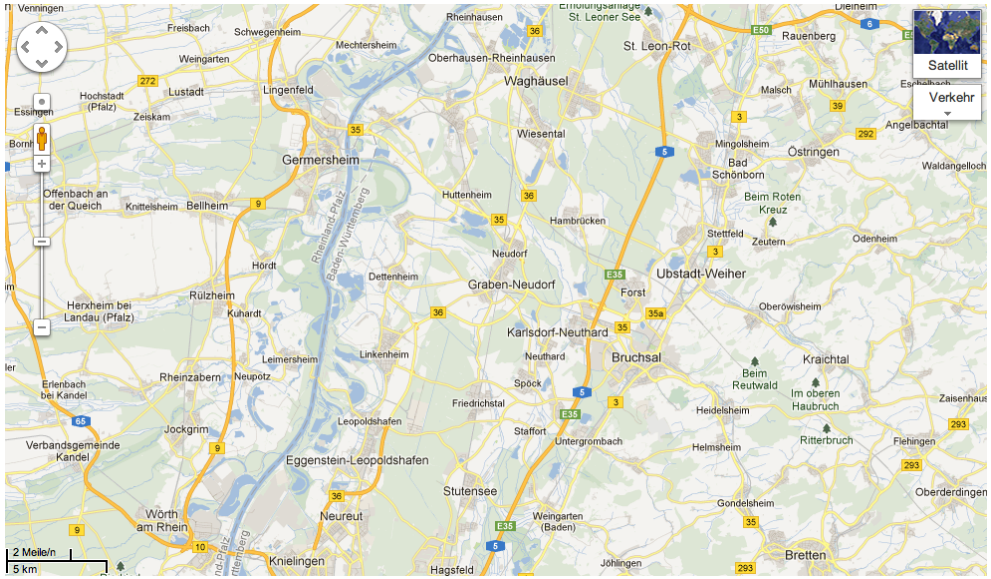
- Auswahl/Filtern von Objekten
- Vereinfachung
- Symbolisierung
- Aggregation
- Glättung
- Vergrößerung
- Verdrängung
- ...



Quelle: Wikipedia

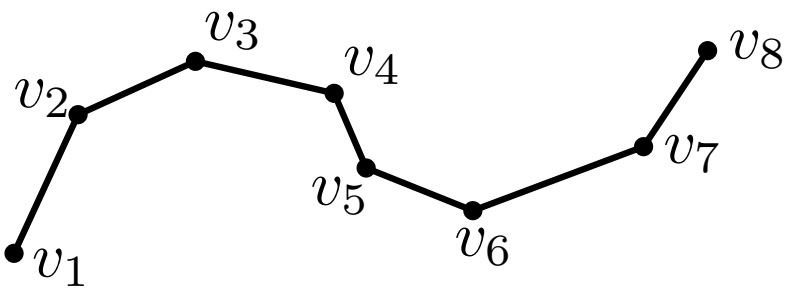
Linienvereinfachung

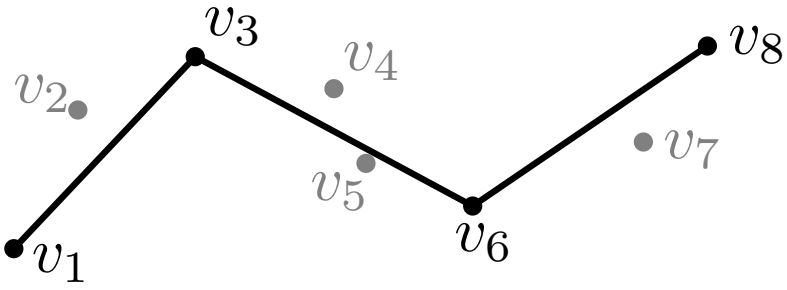
- Karten bestehen zum Großteil aus Linien-/ Polygonzügen
- Linienvereinfachung reduziert Komplexität: visuell und Speicherplatz



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

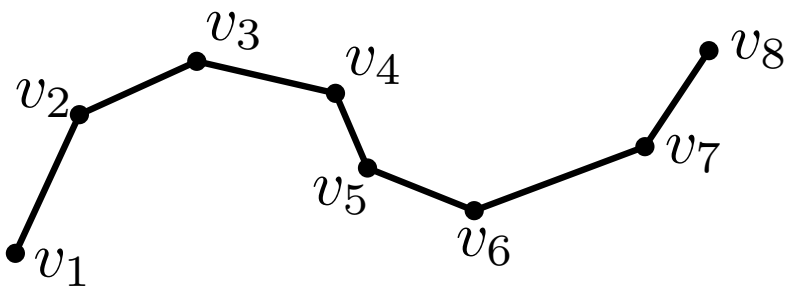
ges: Polygonzug $Q = (v_1 = v_{i_1}, v_{i_2}, v_{i_3}, \ldots, v_{i_k} = v_n)$ mit $i_1 < i_2 < \cdots < i_k$, so dass Q eine gute Approximation von P und k möglichst klein ist

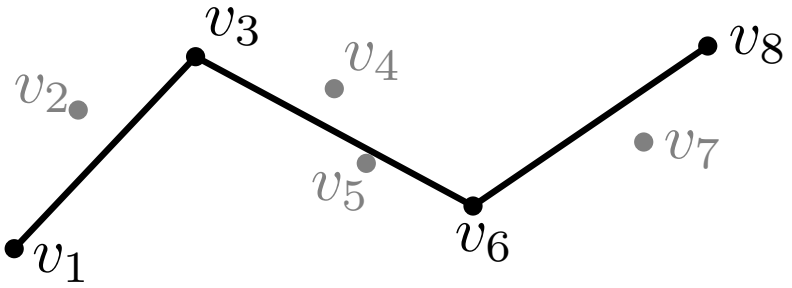




Qualitätskriterien

Überlegen Sie sich mögliche Kriterien für die Approximationsqualität!

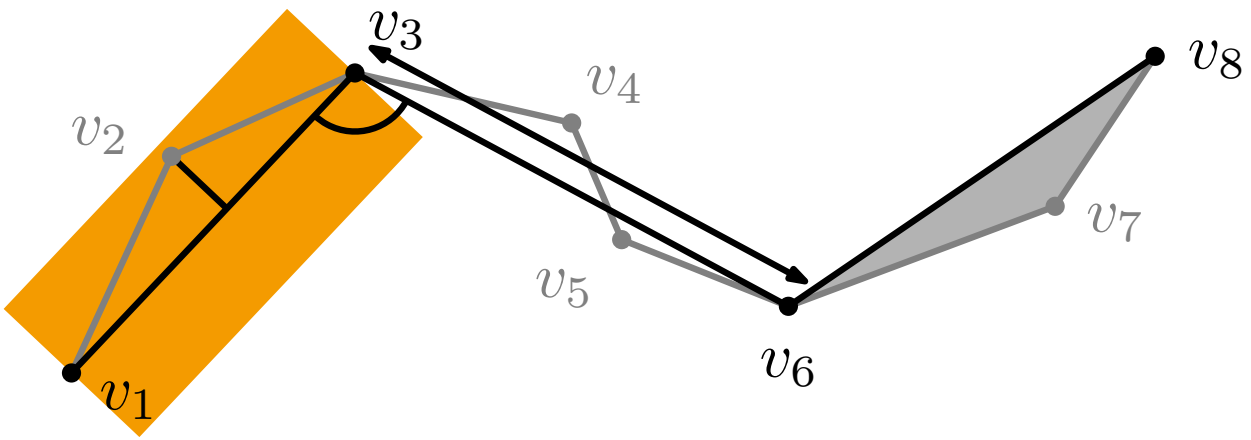




Qualitätskriterien

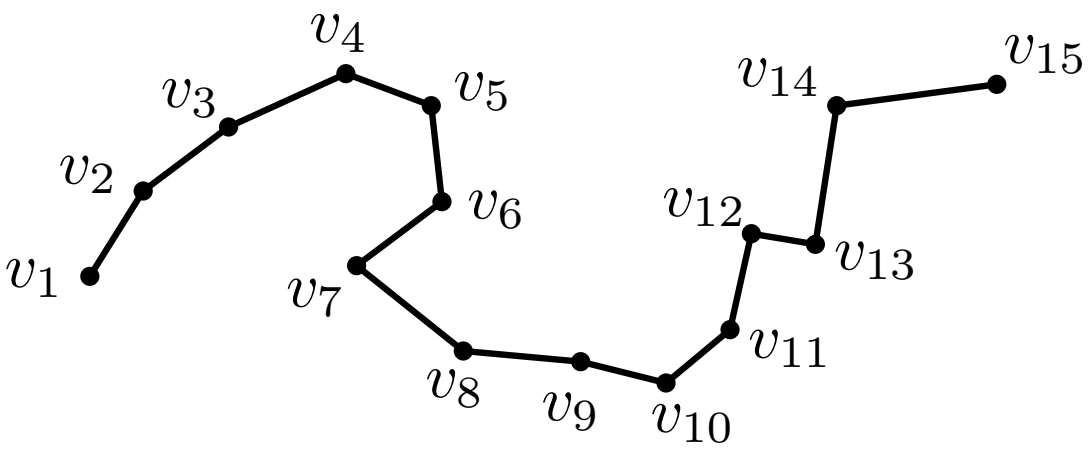
Überlegen Sie sich mögliche Kriterien für die Approximationsqualität!

- ähnliche Länge
- ähnliche Winkel
- geringer Abstand
- ε-Korridor
- kleine Flächenänderung
- wenige Segmente



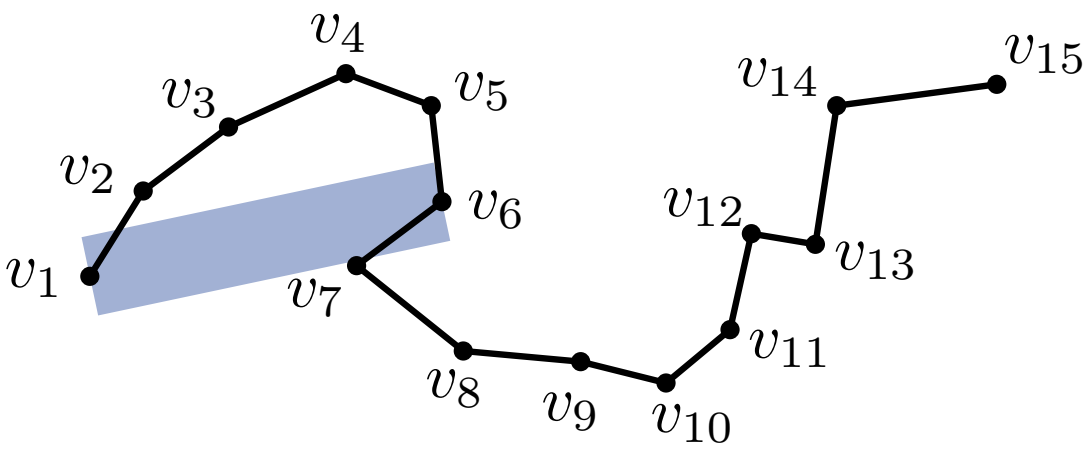
Erste Algorithmen

lokale Suche mit look-ahead s und ε-Korridor



Erste Algorithmen

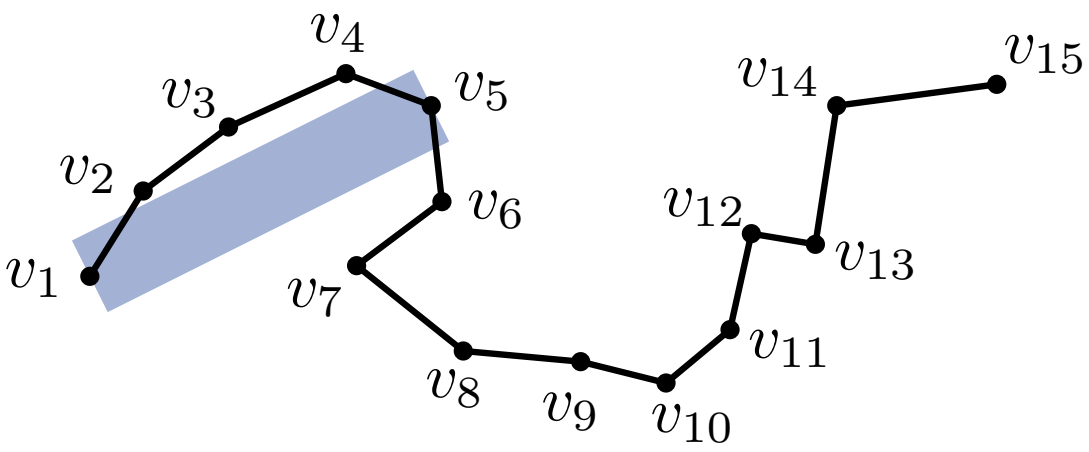
lokale Suche mit look-ahead s und ε-Korridor



$s = 5$

Erste Algorithmen

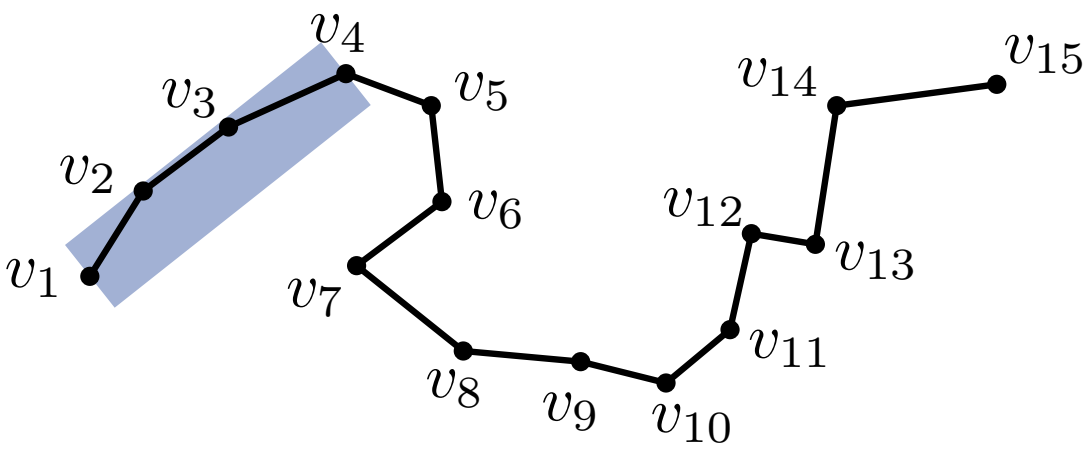
lokale Suche mit look-ahead s und ε-Korridor



$s = 5$

Erste Algorithmen

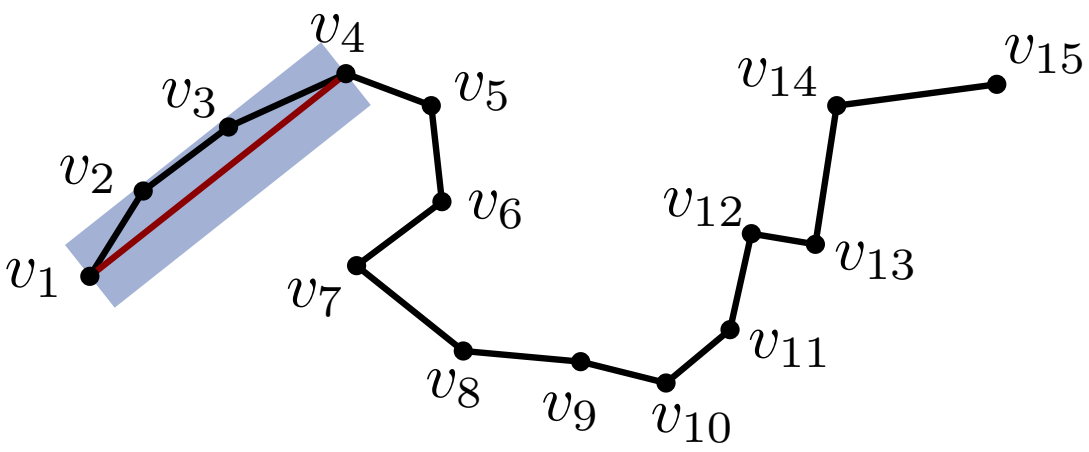
lokale Suche mit look-ahead s und ε-Korridor



$s = 5$

Erste Algorithmen

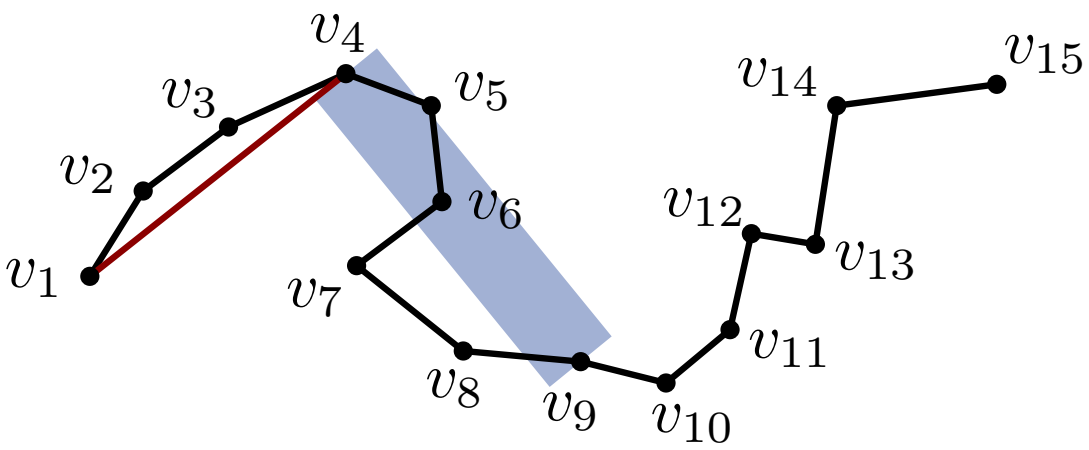
lokale Suche mit look-ahead s und ε-Korridor



$s = 5$

Erste Algorithmen

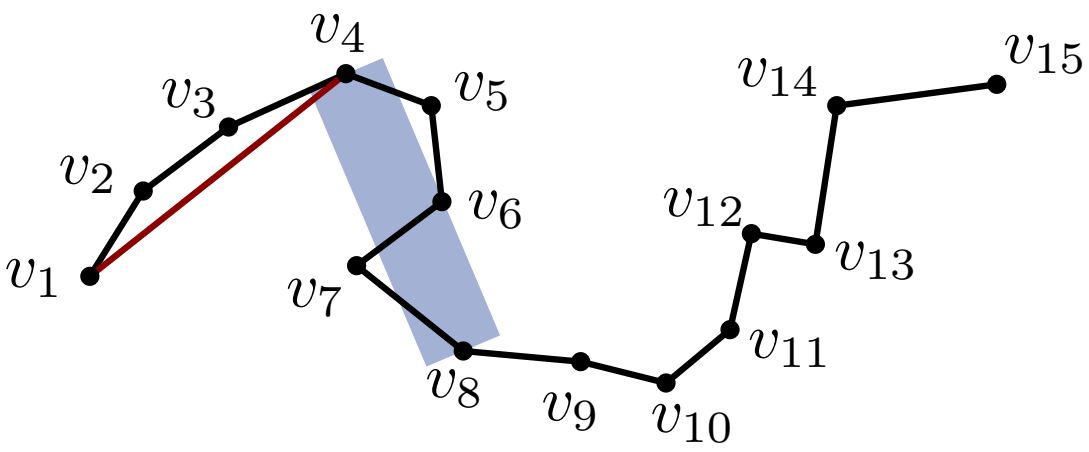
lokale Suche mit look-ahead s und ε-Korridor



$s = 5$

Erste Algorithmen

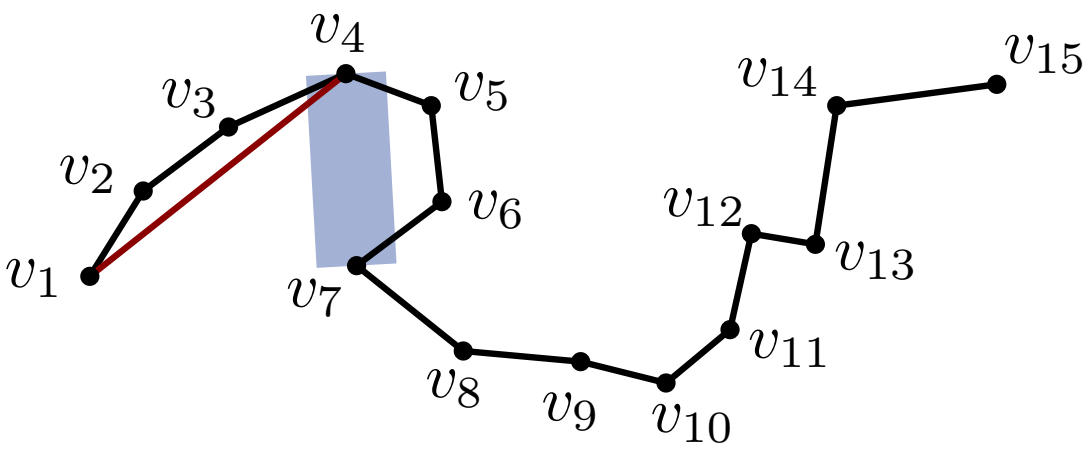
lokale Suche mit look-ahead s und ε-Korridor



$s = 5$

Erste Algorithmen

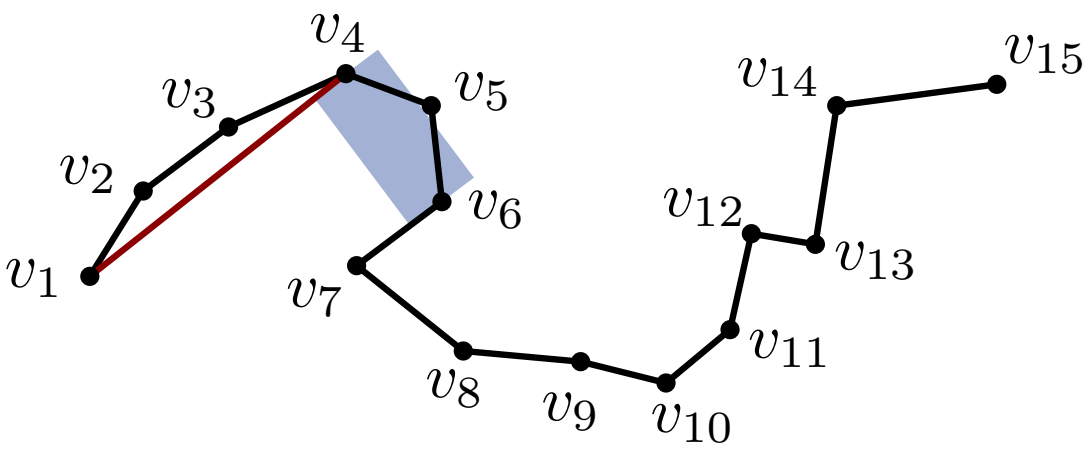
lokale Suche mit look-ahead s und ε-Korridor



$s = 5$

Erste Algorithmen

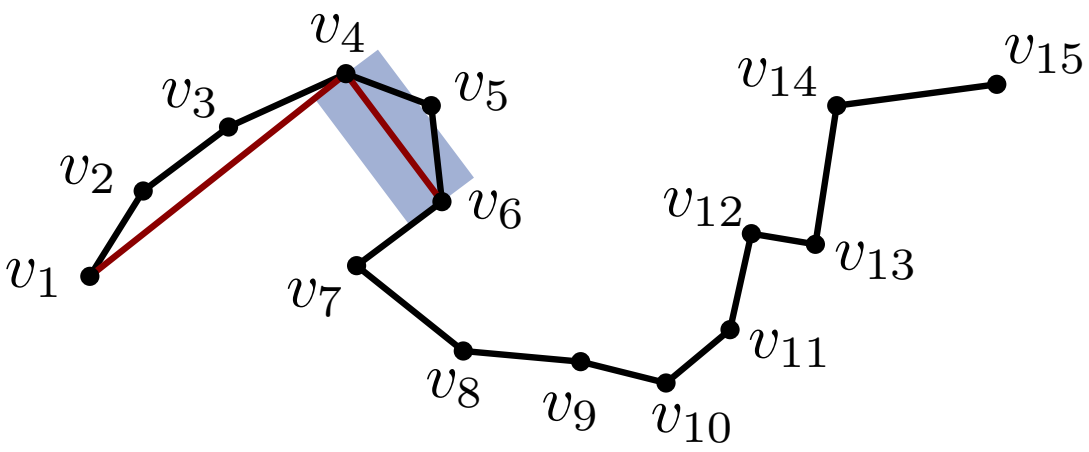
lokale Suche mit look-ahead s und ε-Korridor



$s = 5$

Erste Algorithmen

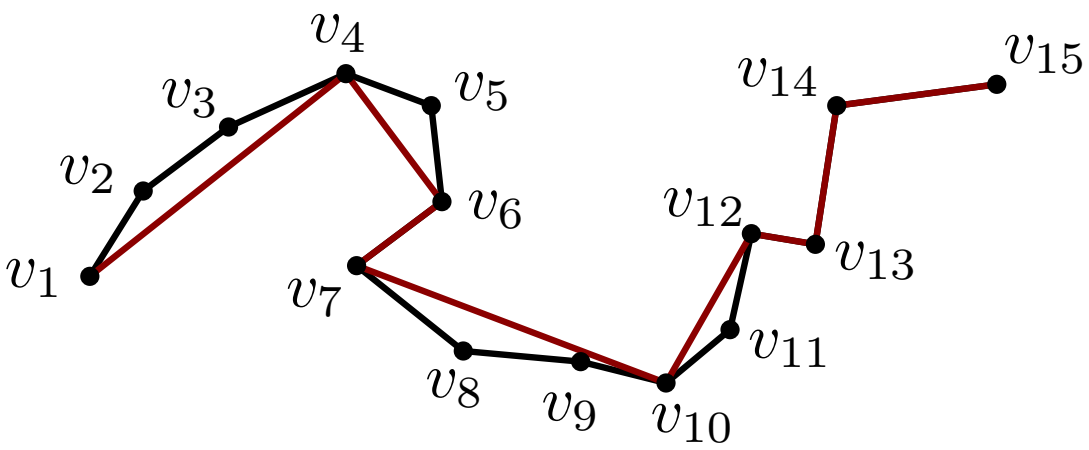
lokale Suche mit look-ahead s und ε-Korridor



$s = 5$

Erste Algorithmen

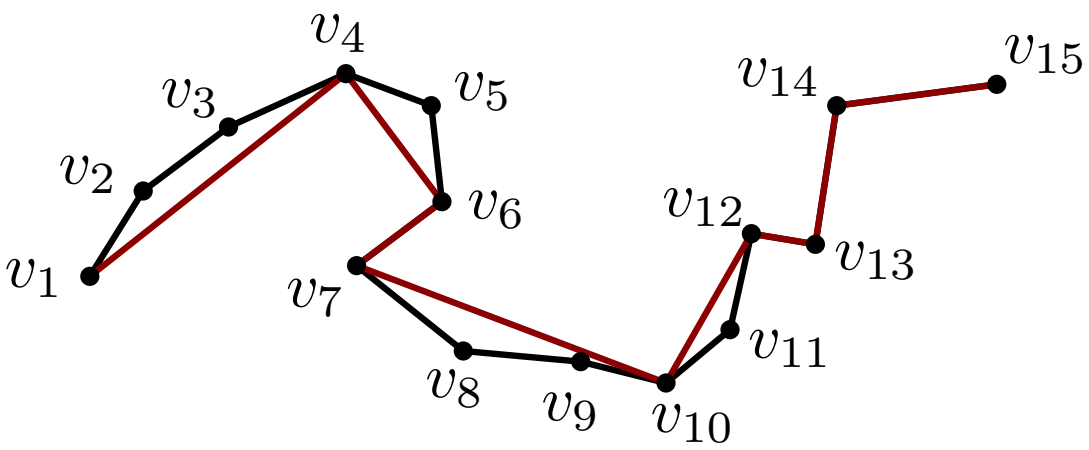
lokale Suche mit look-ahead s und ε-Korridor



$s = 5$

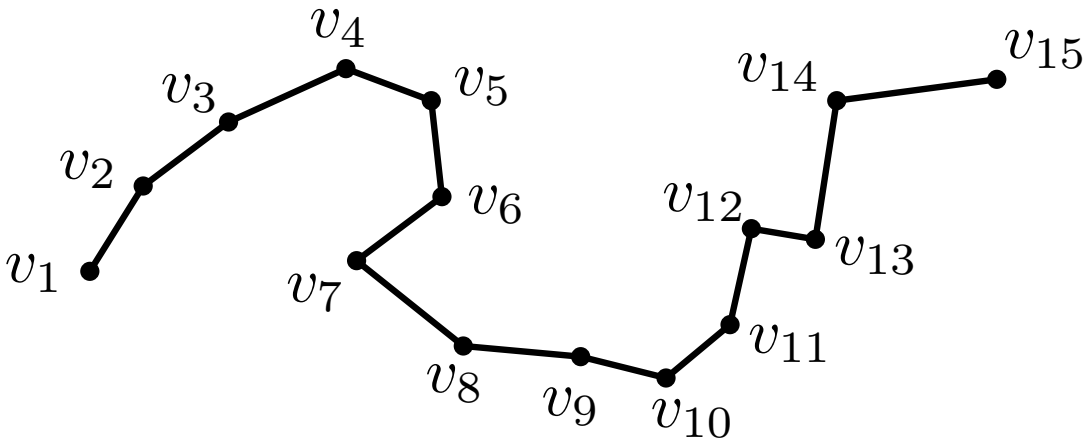
Erste Algorithmen

lokale Suche mit look-ahead s und ε-Korridor



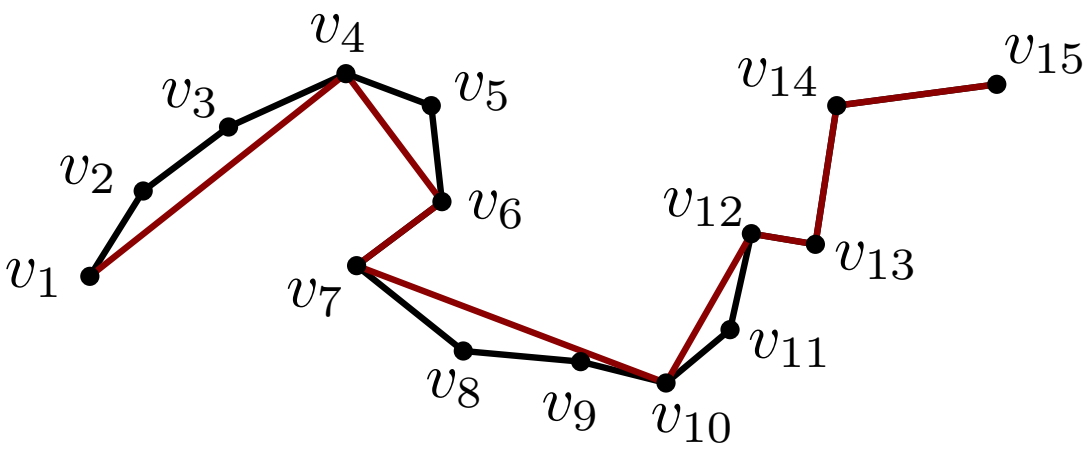
$s = 5$

unbeschränkte greedy Suche mit ε-Korridor



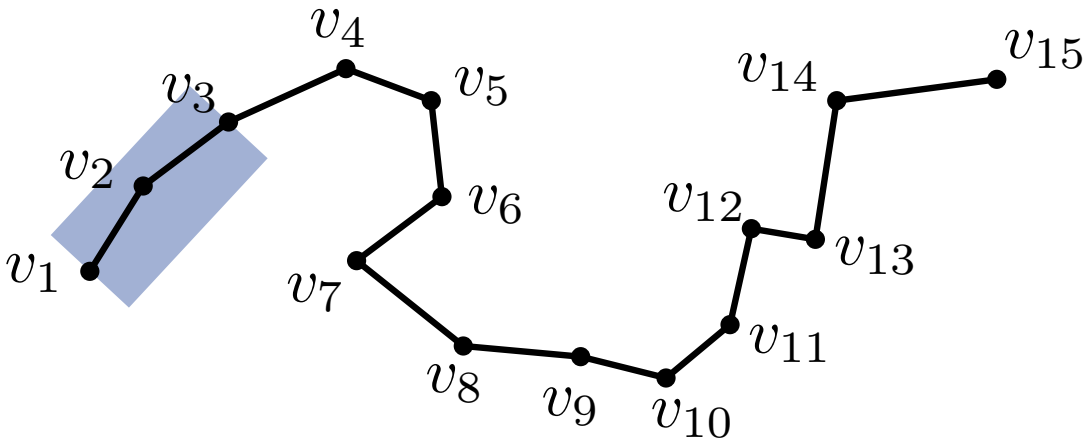
Erste Algorithmen

lokale Suche mit look-ahead s und ε-Korridor



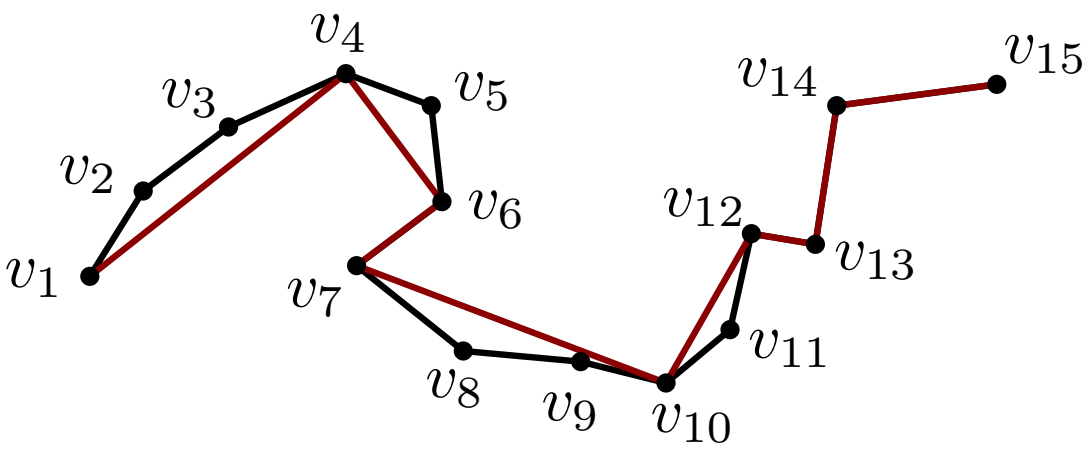
$s = 5$

unbeschränkte greedy Suche mit ε-Korridor



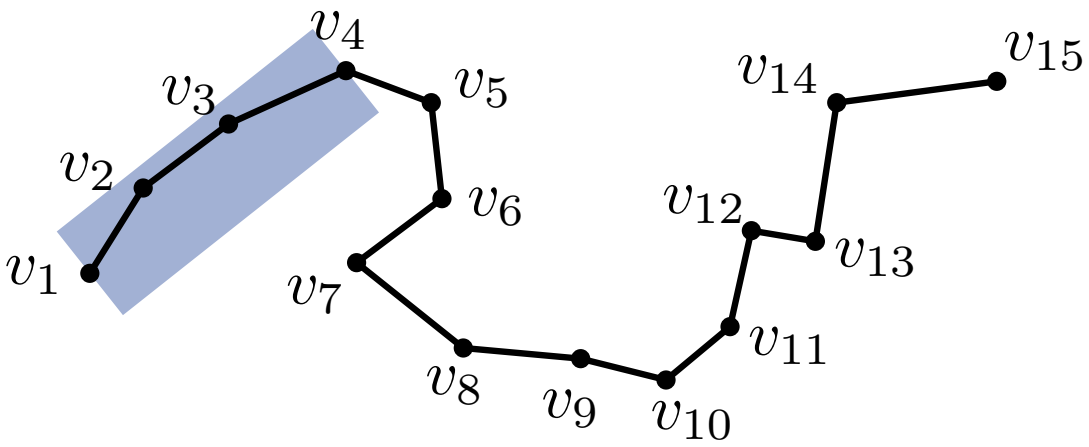
Erste Algorithmen

lokale Suche mit look-ahead s und ε-Korridor



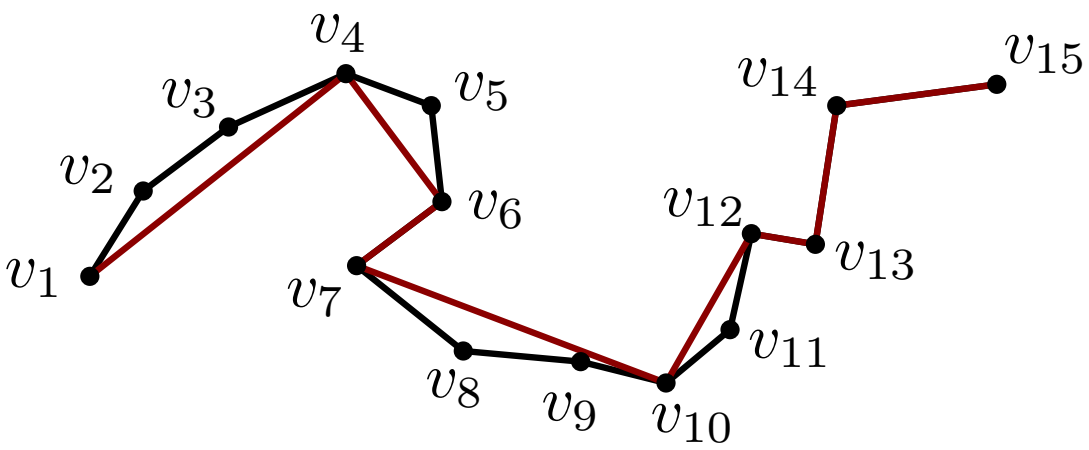
$s = 5$

unbeschränkte greedy Suche mit ε-Korridor



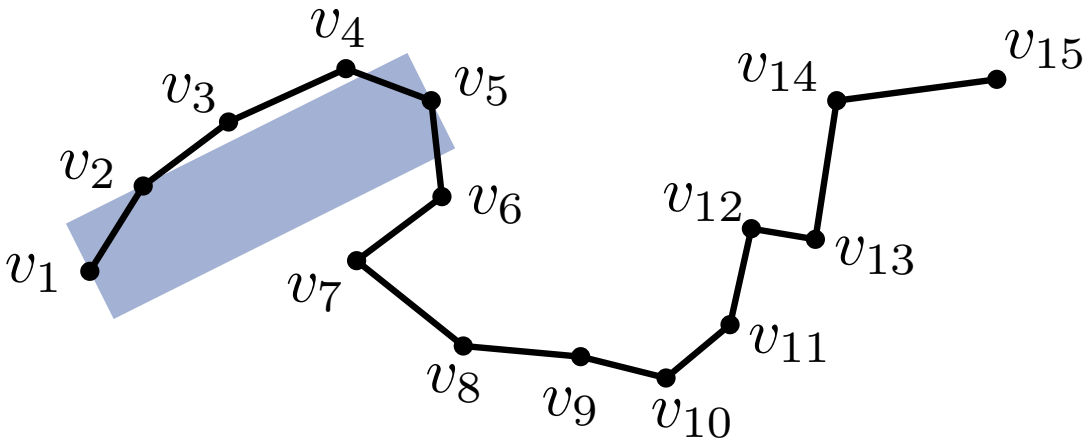
Erste Algorithmen

lokale Suche mit look-ahead s und ε-Korridor



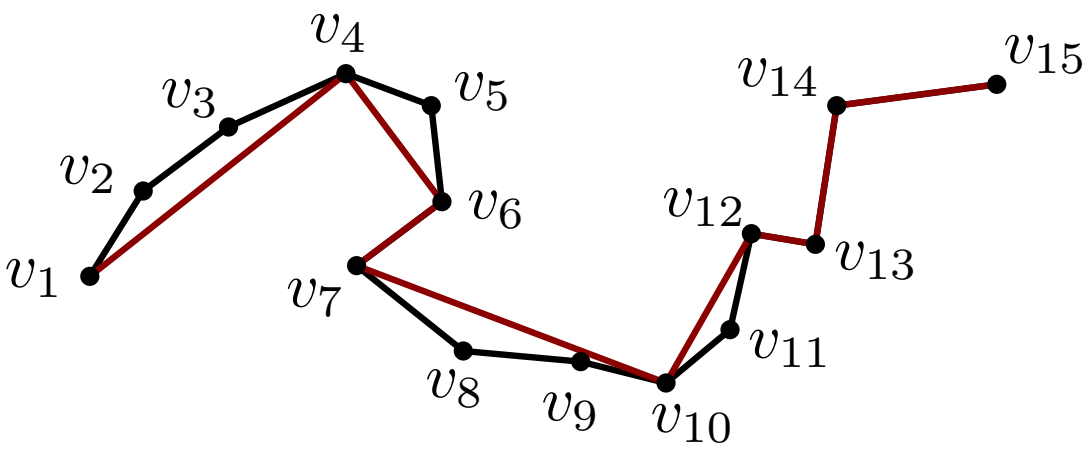
$s = 5$

unbeschränkte greedy Suche mit ε-Korridor



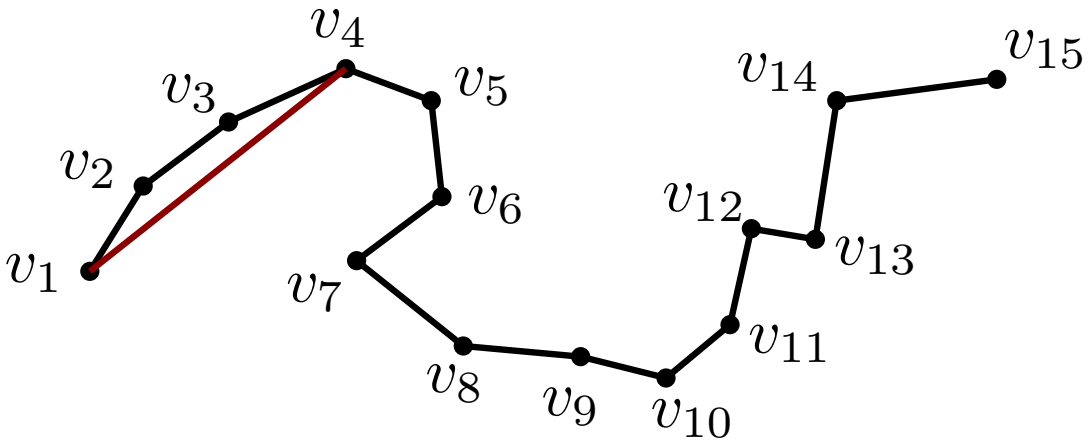
Erste Algorithmen

lokale Suche mit look-ahead s und ε-Korridor



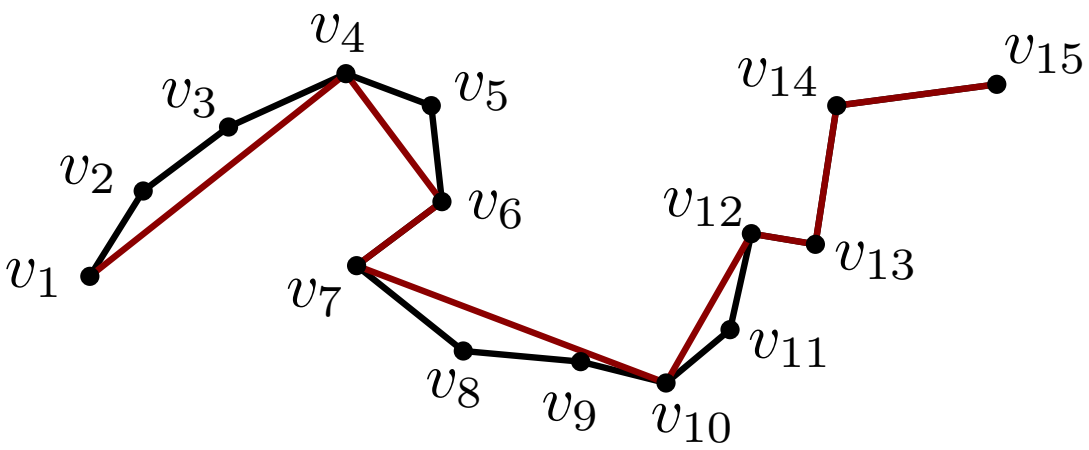
$s = 5$

unbeschränkte greedy Suche mit ε-Korridor



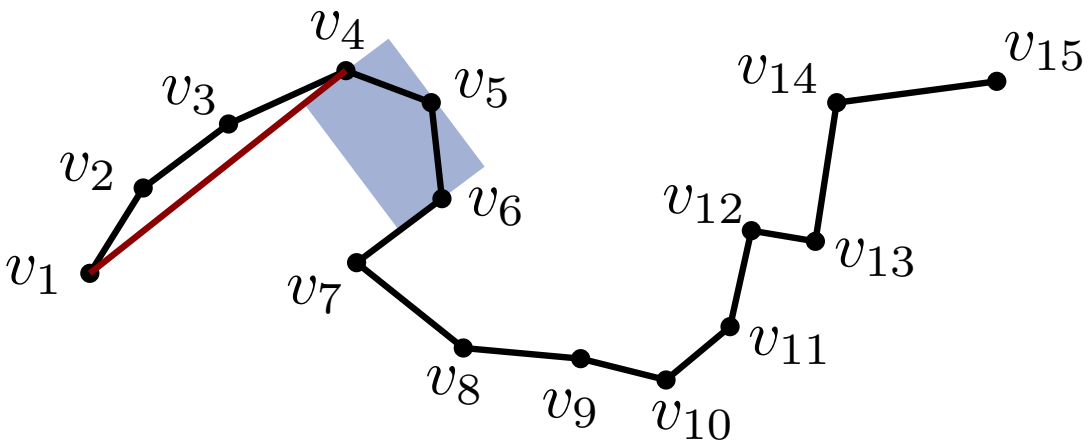
Erste Algorithmen

lokale Suche mit look-ahead s und ε-Korridor



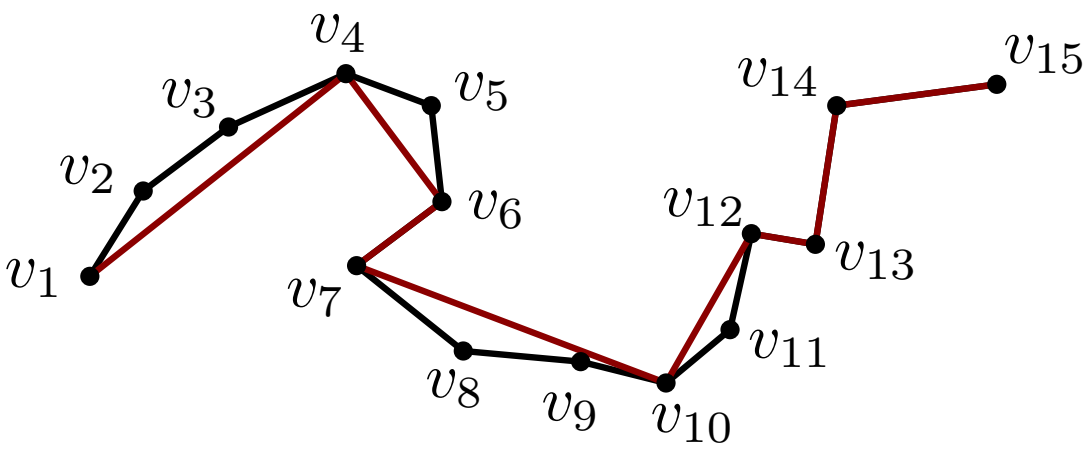
$s = 5$

unbeschränkte greedy Suche mit ε-Korridor



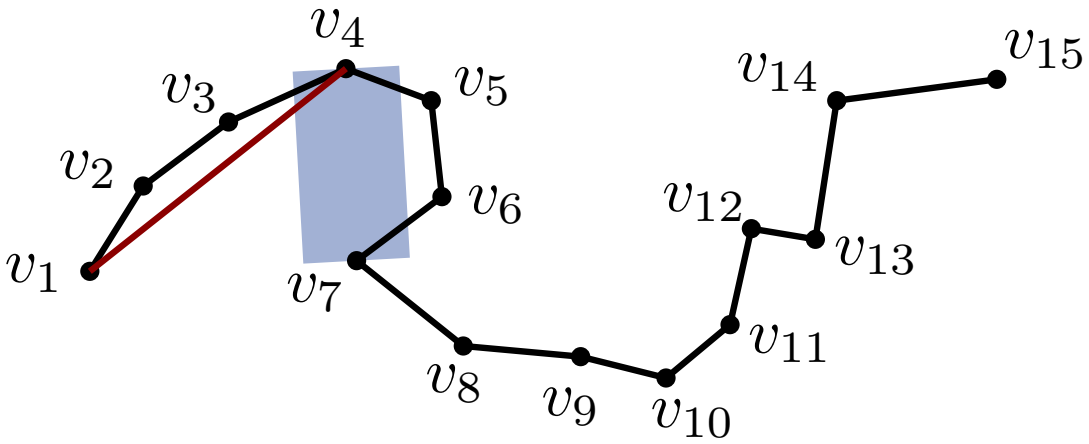
Erste Algorithmen

lokale Suche mit look-ahead s und ε-Korridor



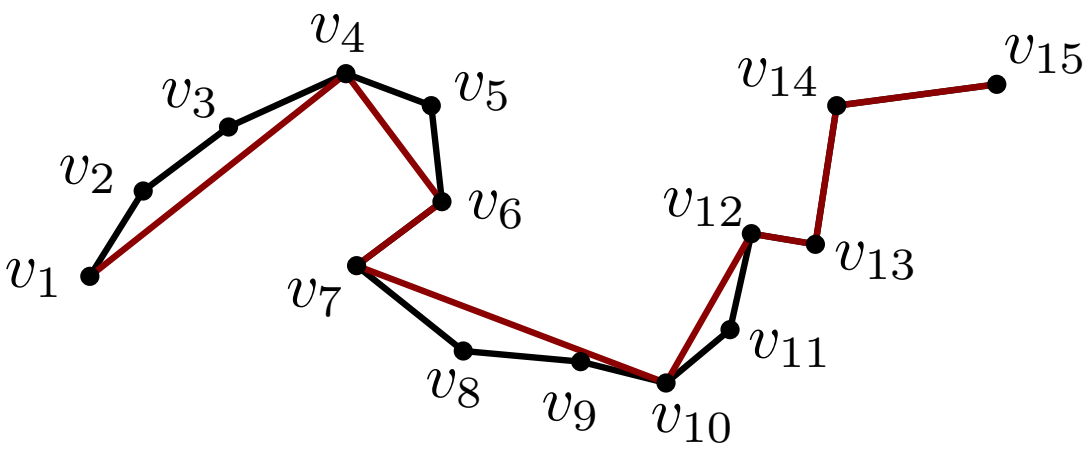
$s = 5$

unbeschränkte greedy Suche mit ε-Korridor



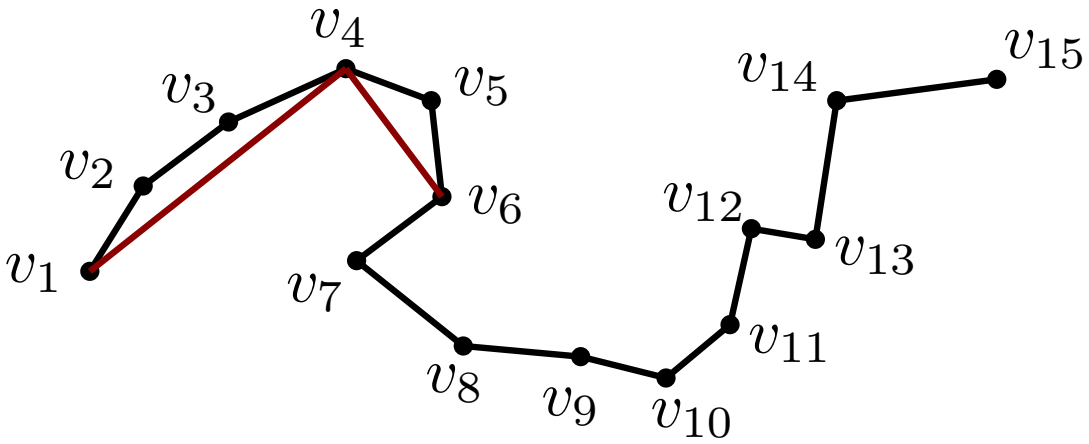
Erste Algorithmen

lokale Suche mit look-ahead s und ε-Korridor



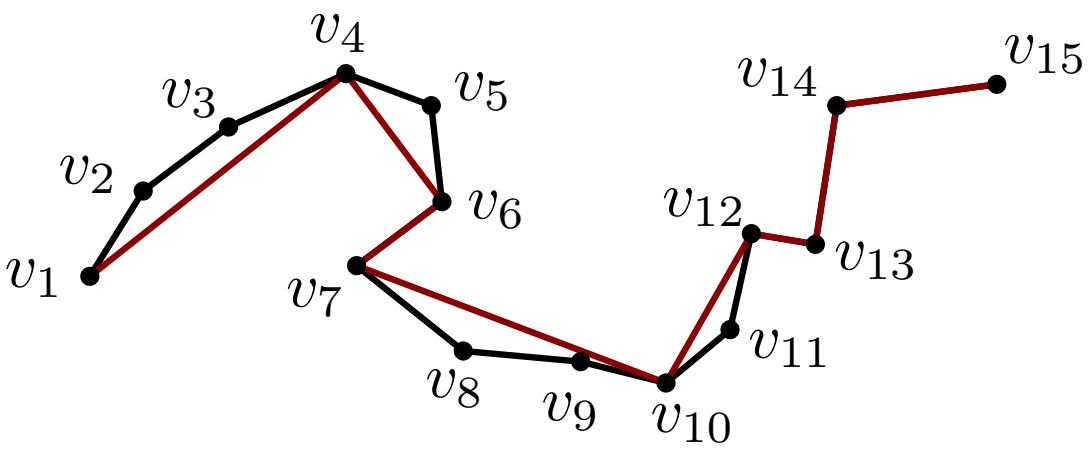
$s = 5$

unbeschränkte greedy Suche mit ε-Korridor



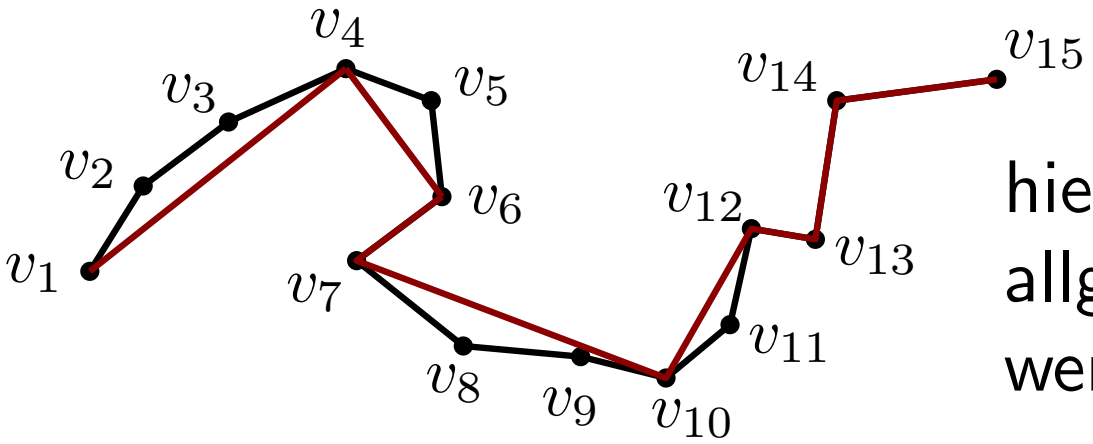
Erste Algorithmen

lokale Suche mit look-ahead s und ε-Korridor



$s = 5$

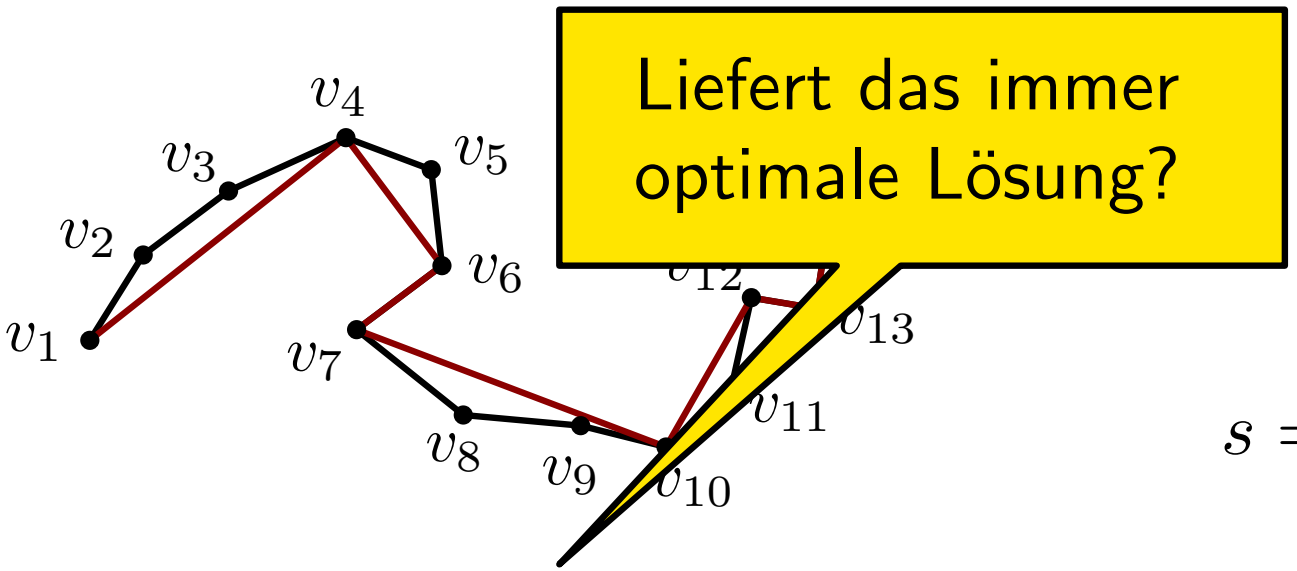
unbeschränkte greedy Suche mit ε-Korridor



hier identisch,
allg. jedoch
weniger Segmente

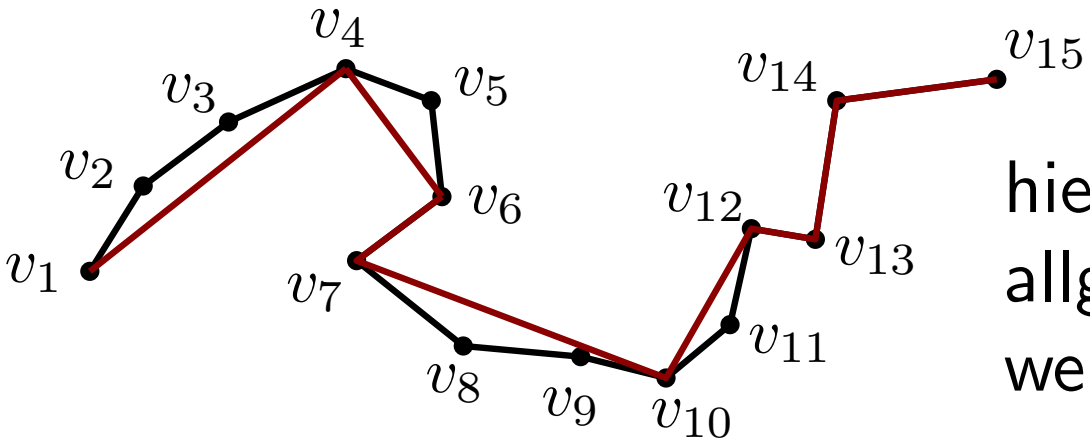
Erste Algorithmen

lokale Suche mit look-ahead s und ε-Korridor



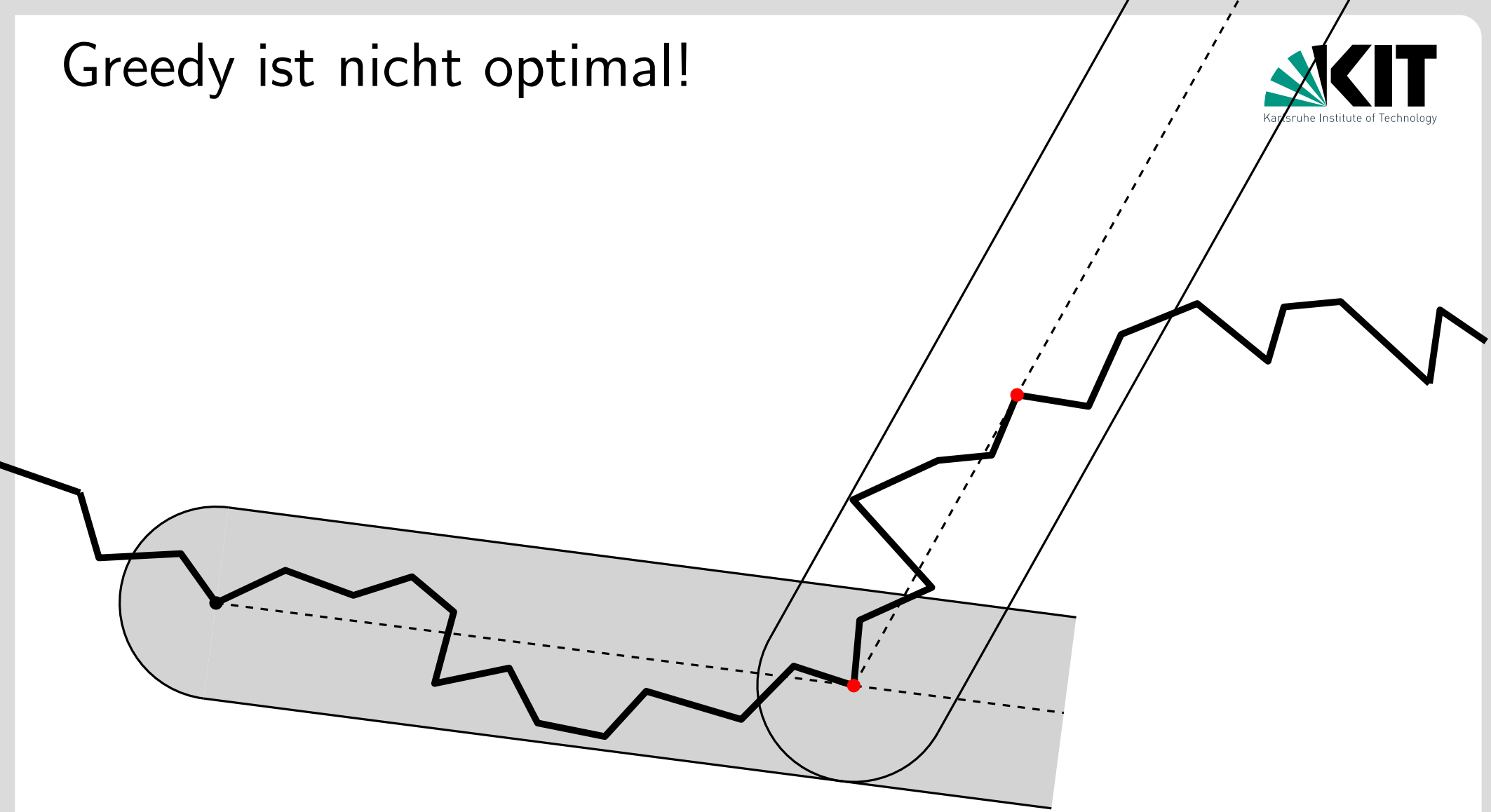
$s = 5$

unbeschränkte greedy Suche mit ε-Korridor

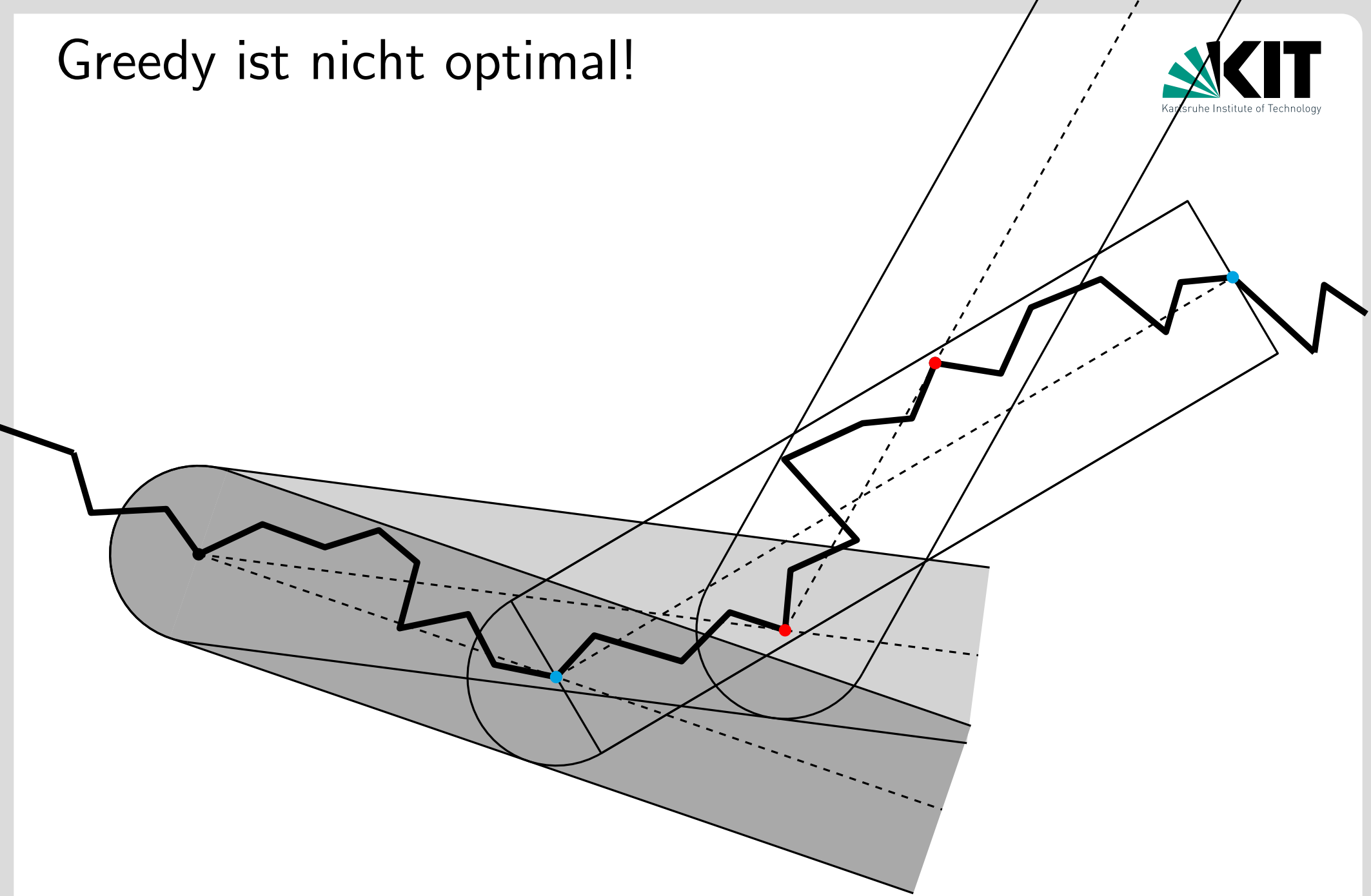


hier identisch, allg. jedoch weniger Segmente

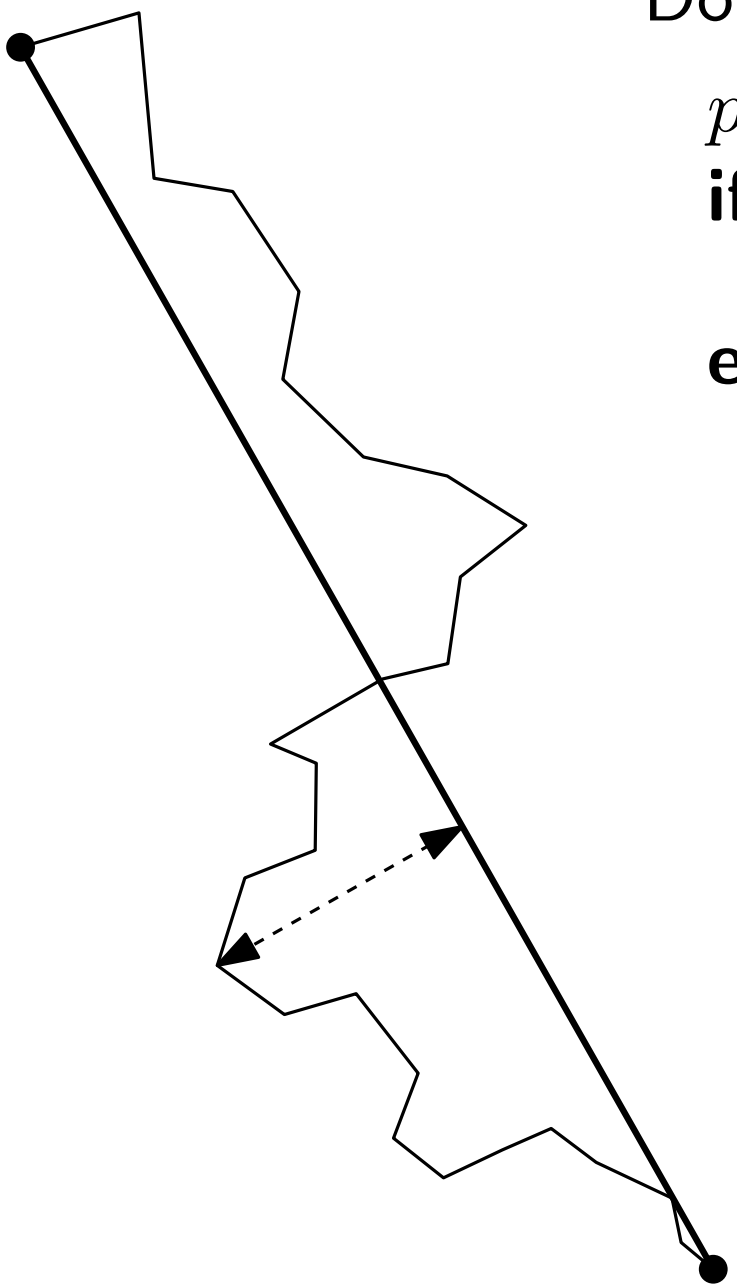
Greedy ist nicht optimal!



Greedy ist nicht optimal!



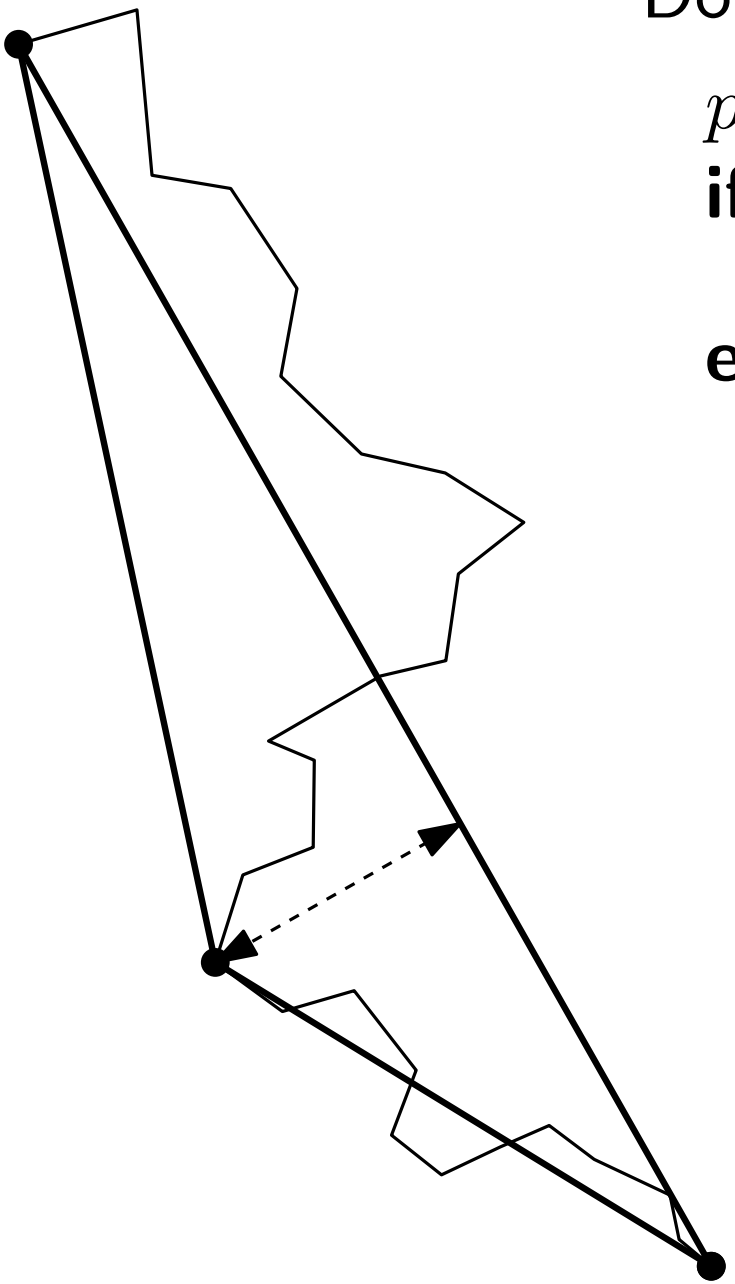
Douglas-Peucker Algorithmus [1973]



DouglasPeucker(P, i, j)

$p_f \leftarrow$ weitester Knoten von $\overline{p_i p_j}$
if dist$(\overline{p_i p_j}, p_f) < \varepsilon$ **then**
  return $\overline{p_i p_j}$
else
  DouglasPeucker(P, i, f)
  DouglasPeucker(P, f, j)

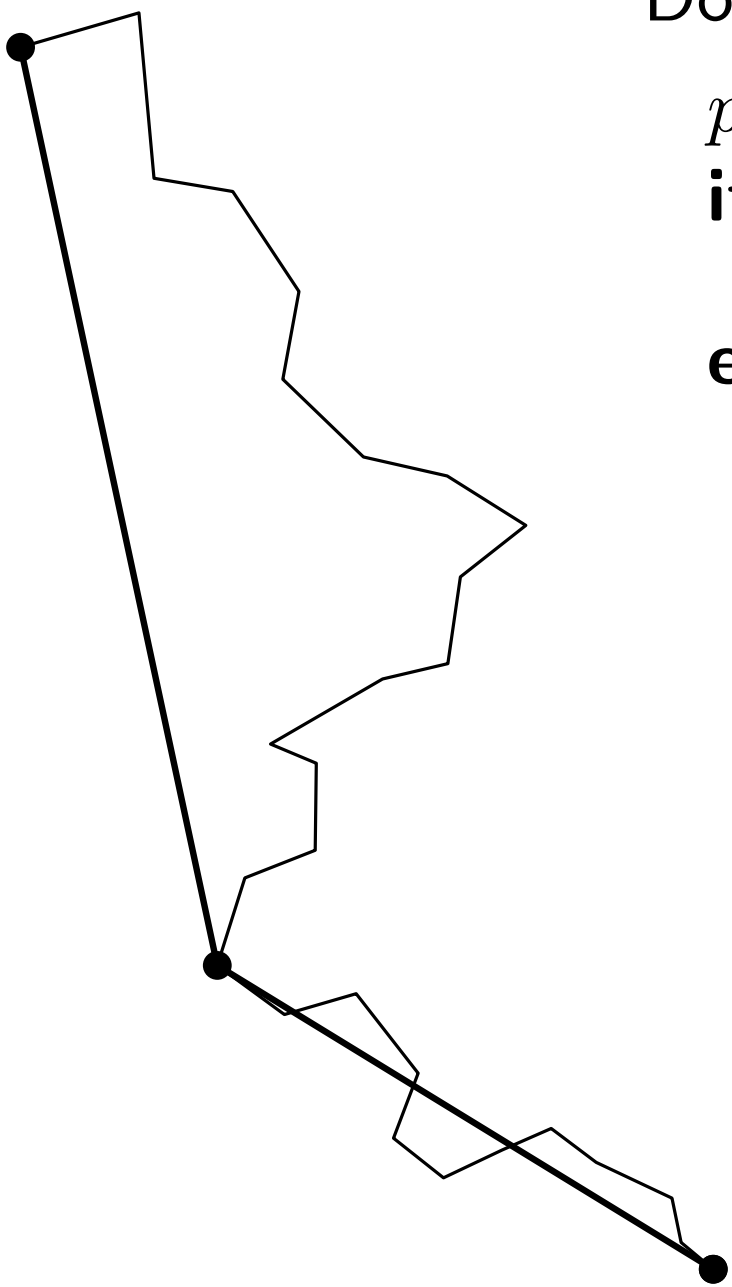
Douglas-Peucker Algorithmus [1973]



DouglasPeucker(P, i, j)

$p_f \leftarrow$ weitester Knoten von $\overline{p_i p_j}$
if dist$(\overline{p_i p_j}, p_f) < \varepsilon$ **then**
  return $\overline{p_i p_j}$
else
  DouglasPeucker(P, i, f)
  DouglasPeucker(P, f, j)

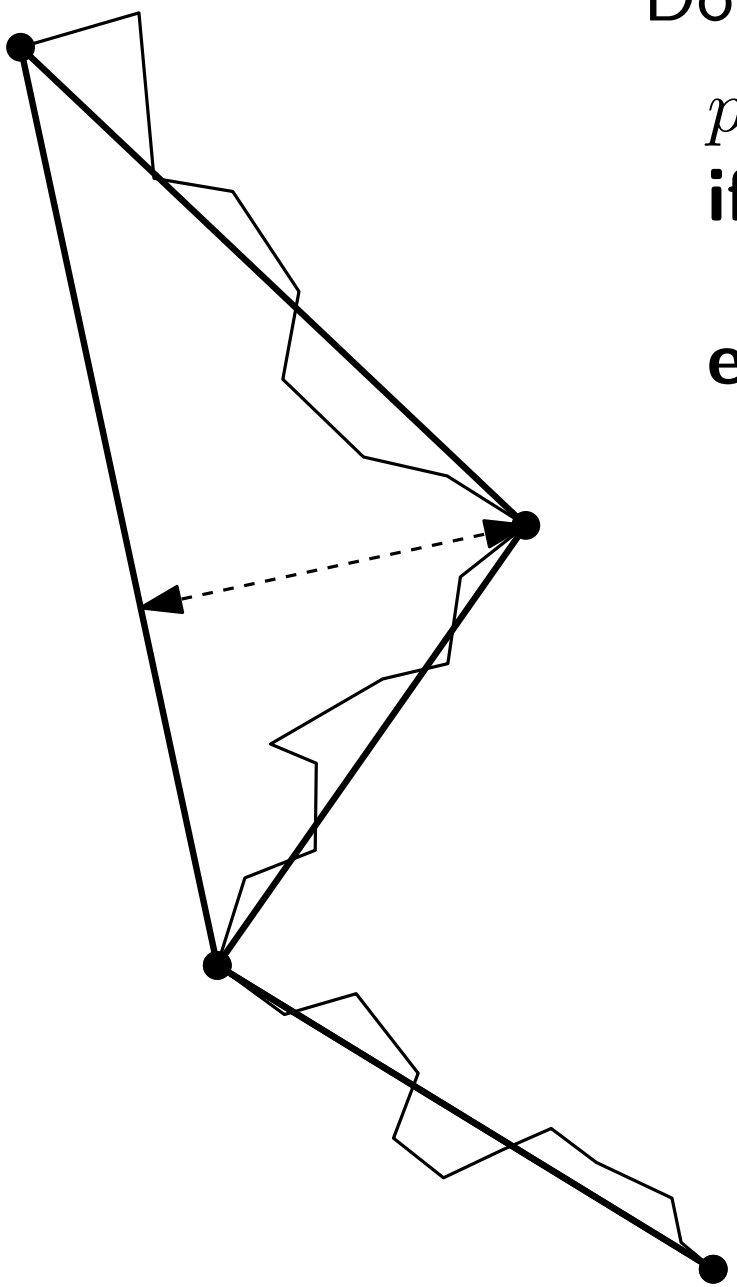
Douglas-Peucker Algorithmus [1973]



DouglasPeucker(P, i, j)

$p_f \leftarrow$ weitester Knoten von $\overline{p_i p_j}$
if dist$(\overline{p_i p_j}, p_f) < \varepsilon$ **then**
  return $\overline{p_i p_j}$
else
  DouglasPeucker(P, i, f)
  DouglasPeucker(P, f, j)

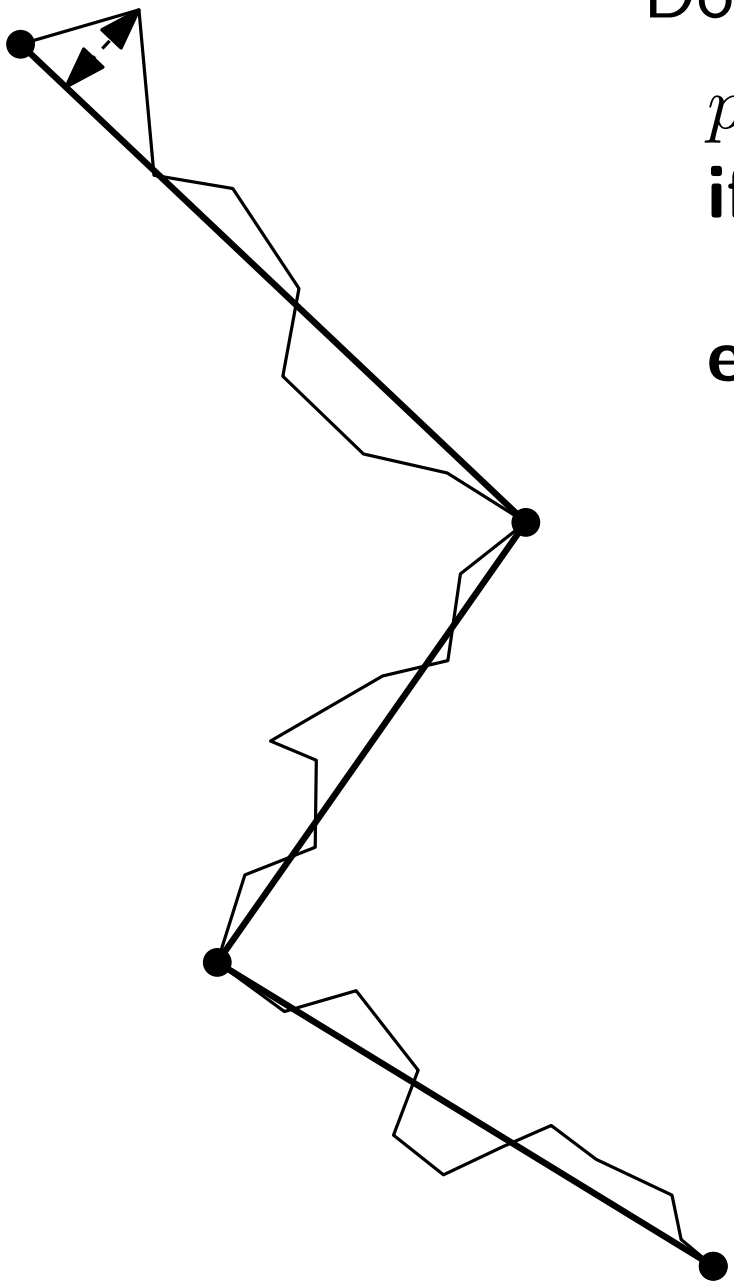
Douglas-Peucker Algorithmus [1973]



DouglasPeucker(P, i, j)

$p_f \leftarrow$ weitester Knoten von $\overline{p_i p_j}$
if dist$(\overline{p_i p_j}, p_f) < \varepsilon$ **then**
  return $\overline{p_i p_j}$
else
  DouglasPeucker(P, i, f)
  DouglasPeucker(P, f, j)

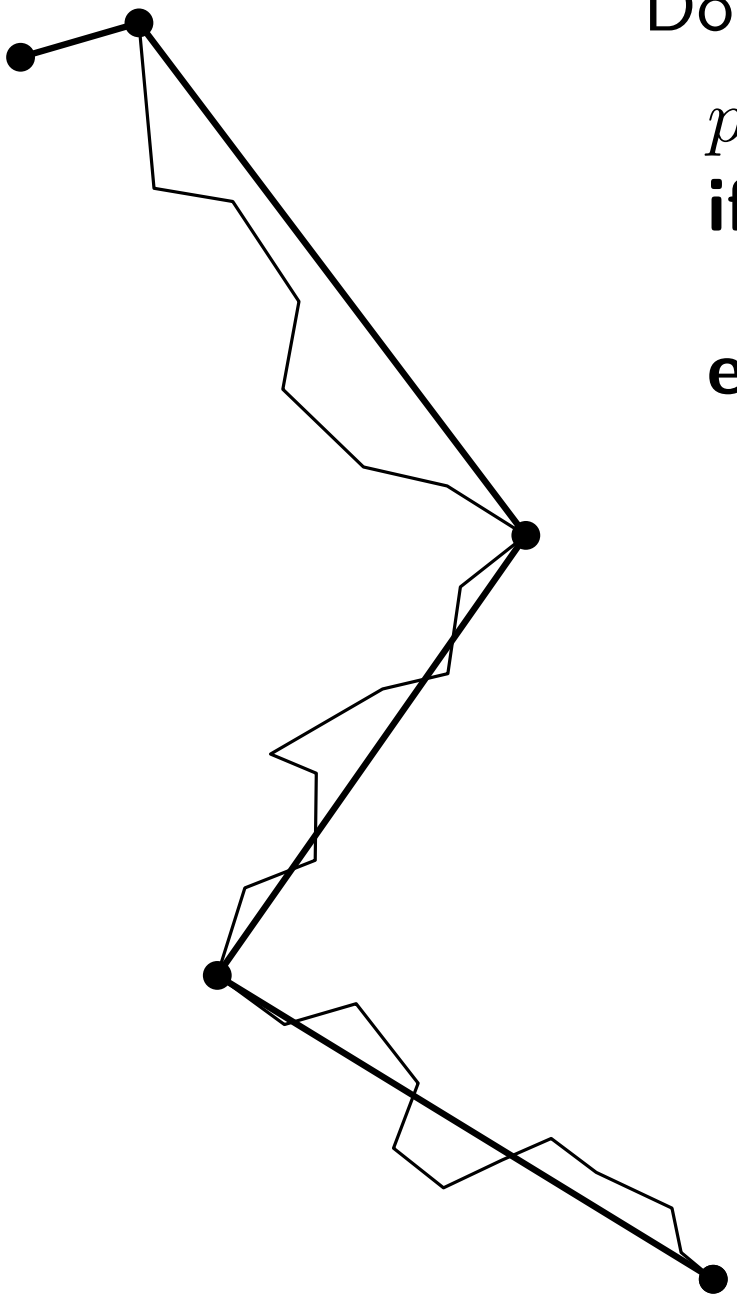
Douglas-Peucker Algorithmus [1973]



DouglasPeucker(P, i, j)

$p_f \leftarrow$ weitester Knoten von $\overline{p_i p_j}$
if dist$(\overline{p_i p_j}, p_f) < \varepsilon$ **then**
  return $\overline{p_i p_j}$
else
  DouglasPeucker(P, i, f)
  DouglasPeucker(P, f, j)

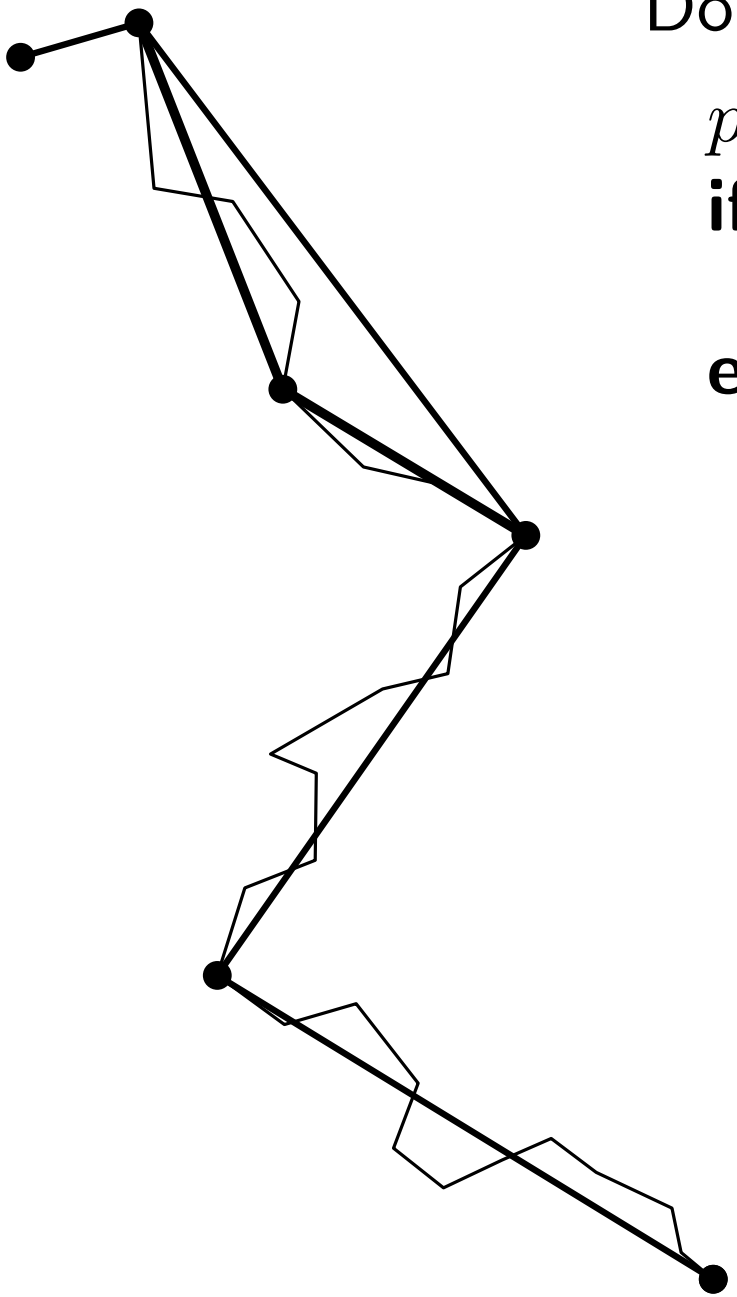
Douglas-Peucker Algorithmus [1973]



DouglasPeucker(P, i, j)

$p_f \leftarrow$ weitester Knoten von $\overline{p_i p_j}$
if dist$(\overline{p_i p_j}, p_f) < \varepsilon$ **then**
  return $\overline{p_i p_j}$
else
  DouglasPeucker(P, i, f)
  DouglasPeucker(P, f, j)

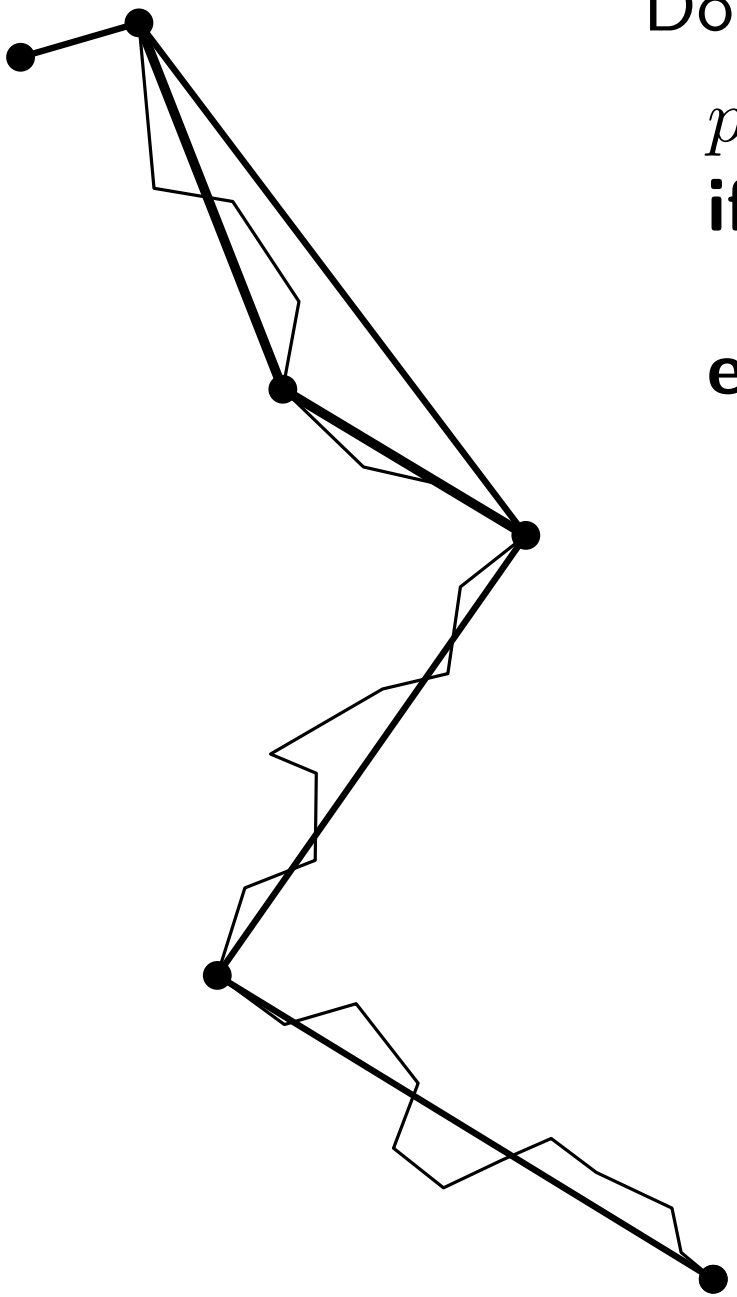
Douglas-Peucker Algorithmus [1973]



DouglasPeucker(P, i, j)

$p_f \leftarrow$ weitester Knoten von $\overline{p_i p_j}$
if dist$(\overline{p_i p_j}, p_f) < \varepsilon$ **then**
  return $\overline{p_i p_j}$
else
  DouglasPeucker(P, i, f)
  DouglasPeucker(P, f, j)

Douglas-Peucker Algorithmus [1973]

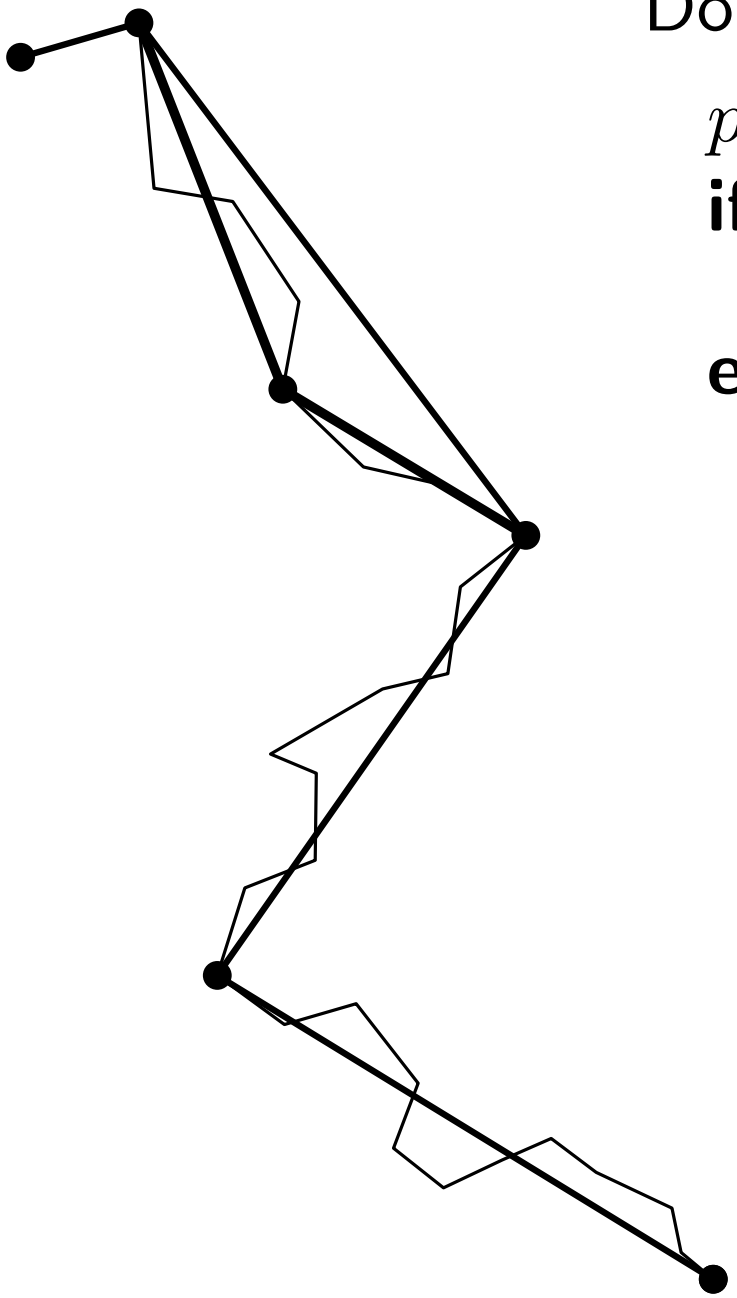


DouglasPeucker(P, i, j)

$p_f \leftarrow$ weitester Knoten von $\overline{p_i p_j}$
if dist$(\overline{p_i p_j}, p_f) < \varepsilon$ **then**
  return $\overline{p_i p_j}$
else
  DouglasPeucker(P, i, f)
  DouglasPeucker(P, f, j)

Laufzeit?

Douglas-Peucker Algorithmus [1973]



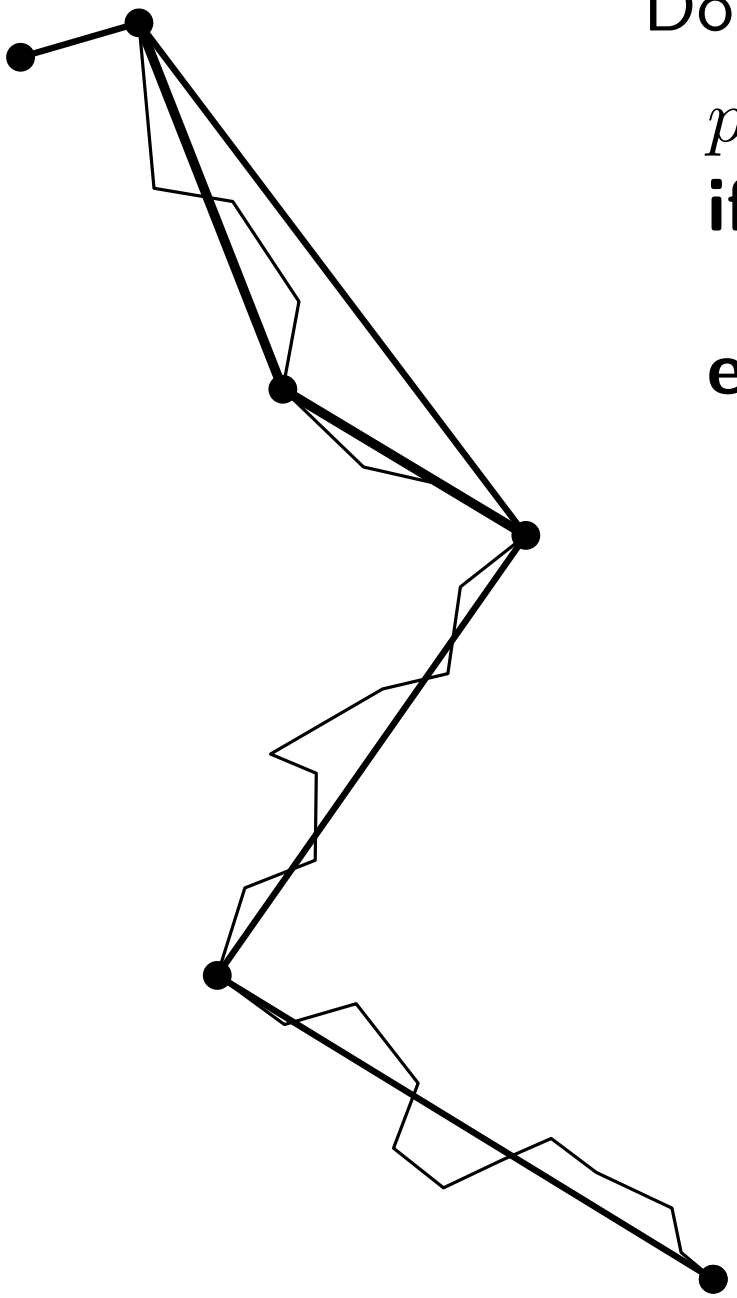
DouglasPeucker(P, i, j)

$p_f \leftarrow$ weitester Knoten von $\overline{p_i p_j}$
if dist$(\overline{p_i p_j}, p_f) < \varepsilon$ **then**
  return $\overline{p_i p_j}$
else
  DouglasPeucker(P, i, f)
  DouglasPeucker(P, f, j)

Laufzeit?

- balancierter Fall: $O(n \log n)$
- worst case: $O(n^2)$

Douglas-Peucker Algorithmus [1973]



DouglasPeucker(P, i, j)

$p_f \leftarrow$ weitester Knoten von $\overline{p_i p_j}$
if dist$(\overline{p_i p_j}, p_f) < \varepsilon$ **then**
  return $\overline{p_i p_j}$
else
  DouglasPeucker(P, i, f)
  DouglasPeucker(P, f, j)

Laufzeit?

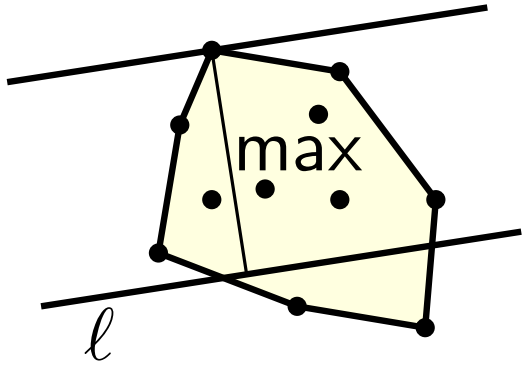
- balancierter Fall: $O(n \log n)$
- worst case: $O(n^2)$

Kann es passieren, dass Selbstschnitte entstehen?

Beschleunigung des DP-Algorithmus

[Hershberger, Snoeyink '92]

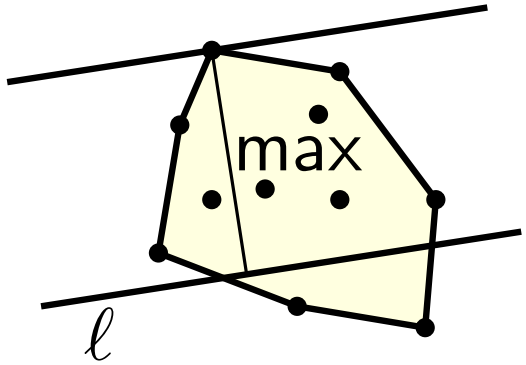
Beobachtung: für Gerade ℓ und Punktmenge P liegt der von ℓ weitest entfernte Punkt auf einer zu ℓ parallelen Tangente an die konvexe Hülle $CH(P)$



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 $CH(P)$

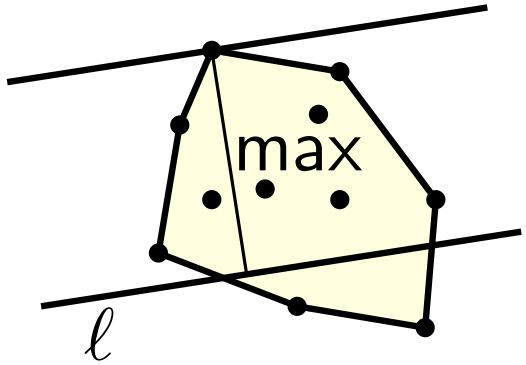


$\rightarrow$ müssen nur solche Punkte als Splitknoten betrachten

Beschleunigung des DP-Algorithmus

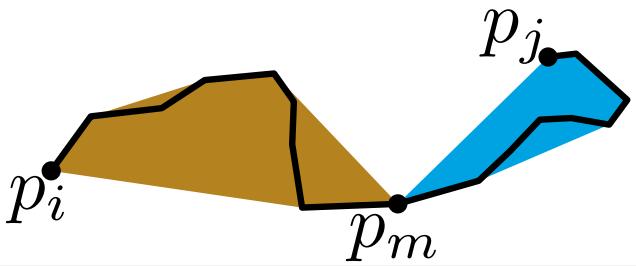
[Hershberger, Snoeyink '92]

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:

- **farthest**(PH, ℓ): finde weitesten von ℓ entfernten Knoten in PH
- **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

Algorithmus von Hershberger und Snoeyink

drei Operationen:

- **farthest**(PH, ℓ): finde weitesten von ℓ entfernten Knoten in PH
- **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

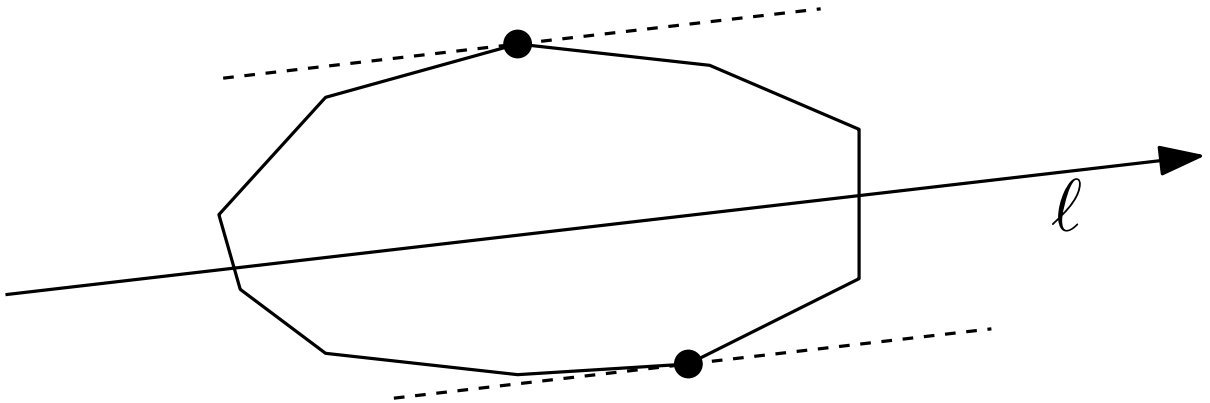
DPHull(i, j, PH)

$p_f \leftarrow$ 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)

farthest Operation

farthest(PH, ℓ)**:** finde entferntesten Punkt von PH zu ℓ

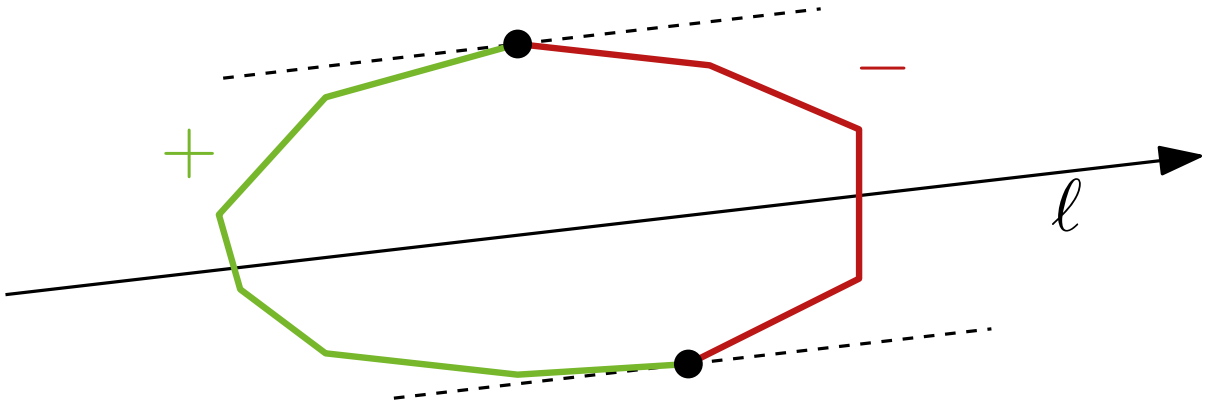
- finde entferntesten Punkt eines konvexen, im UZS gespeicherten Polygons (für jeden Teil von PH)



farthest Operation

farthest(PH, ℓ)**:** finde entferntesten Punkt von PH zu ℓ

- finde entferntesten Punkt eines konvexen, im UZS gespeicherten Polygons (für jeden Teil von PH)

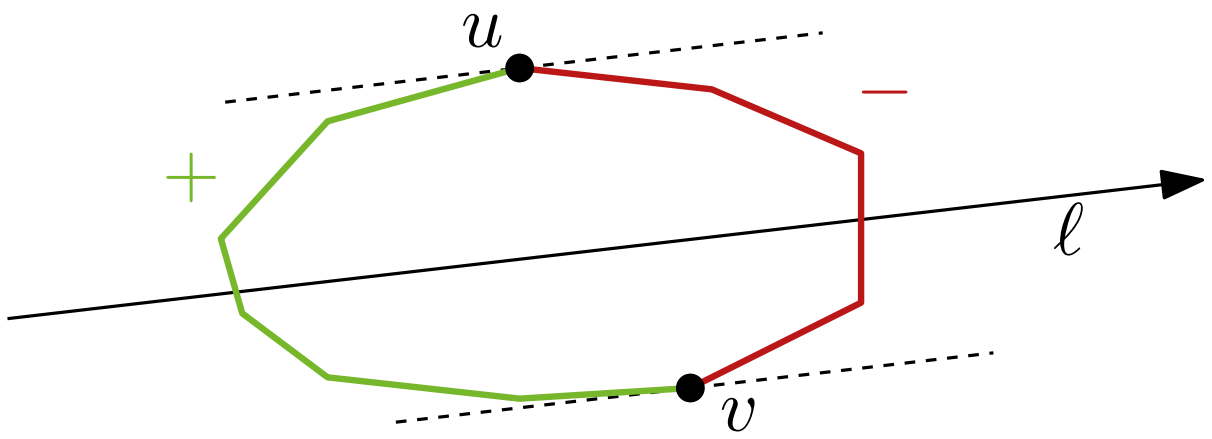


- Kanten e mit $\angle \ell, e \in [0, \pi)$ sind **positiv**
- Kanten e mit $\angle \ell, e \in [\pi, 2\pi)$ sind **negativ**

farthest Operation

farthest(PH, ℓ)**:** finde entferntesten Punkt von PH zu ℓ

- finde entferntesten Punkt eines konvexen, im UZS gespeicherten Polygons (für jeden Teil von PH)



- Kanten e mit $\angle \ell, e \in [0, \pi)$ sind **positiv**
- Kanten e mit $\angle \ell, e \in [\pi, 2\pi)$ sind **negativ**

1. finde positive Kante e und negative Kante e'
2. finde Knoten u und v, die positiv und negativ trennen
3. bestimme $\arg\max\{d(u, \ell), d(v, \ell)\}$

build und split Operationen

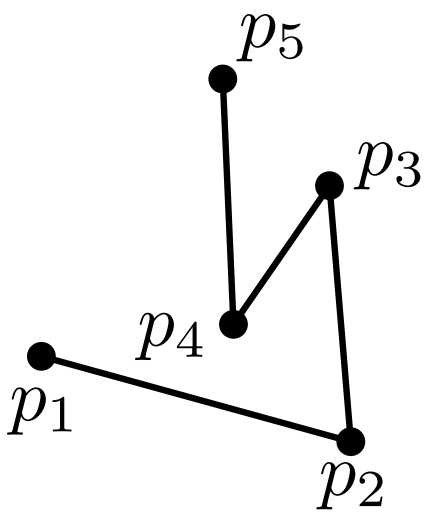
build(i, j, PH)**:** berechne iterativ rechte (linke) konvexe Hülle

$Q \leftarrow$ deque (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ im UZS */
$H \leftarrow$ history stack /* merkt sich Operationen von Q */
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)

build und split Operationen

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

$Q \leftarrow$ deque (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ im UZS */
$H \leftarrow$ history stack /* merkt sich Operationen von Q */
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)

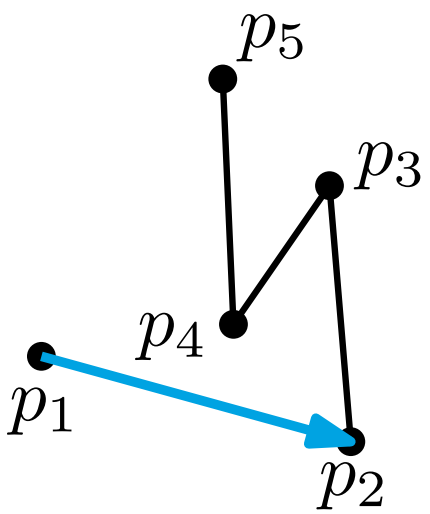


Q: | p_2 | p_1 | p_2 |

build und split Operationen

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

$Q \leftarrow$ deque (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ im UZS */
$H \leftarrow$ history stack /* merkt sich Operationen von Q */
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)

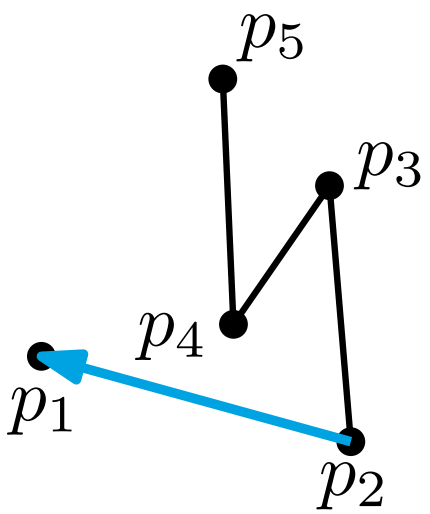


Q: p_2 p_1 p_2

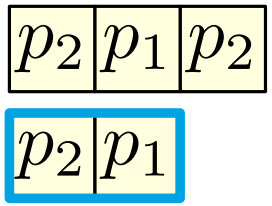
build und split Operationen

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

$Q \leftarrow$ deque (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ im UZS */
$H \leftarrow$ history stack /* merkt sich Operationen von Q */
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)



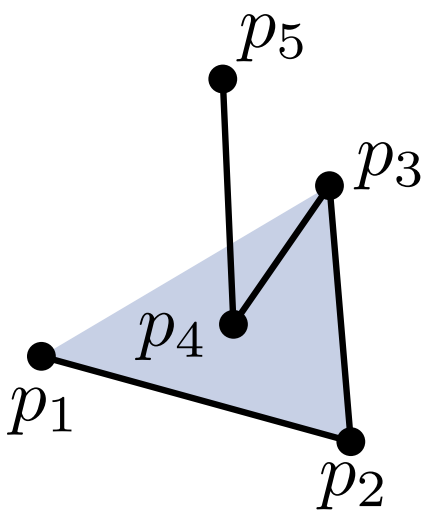
Q:



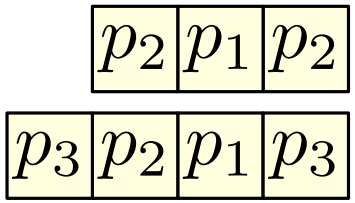
build und split Operationen

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

$Q \leftarrow$ deque (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ im UZS */
$H \leftarrow$ history stack /* merkt sich Operationen von Q */
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)



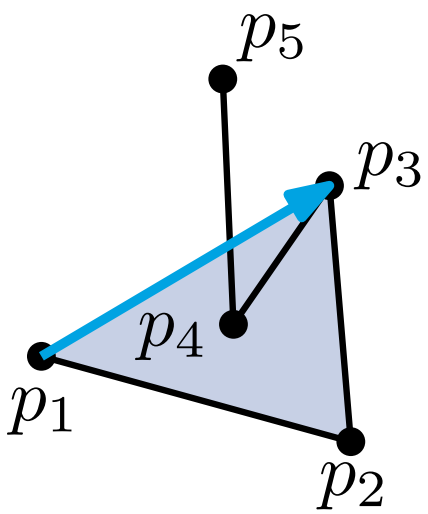
Q:



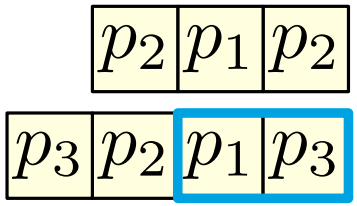
build und split Operationen

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

$Q \leftarrow$ deque (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ im UZS */
$H \leftarrow$ history stack /* merkt sich Operationen von Q */
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)



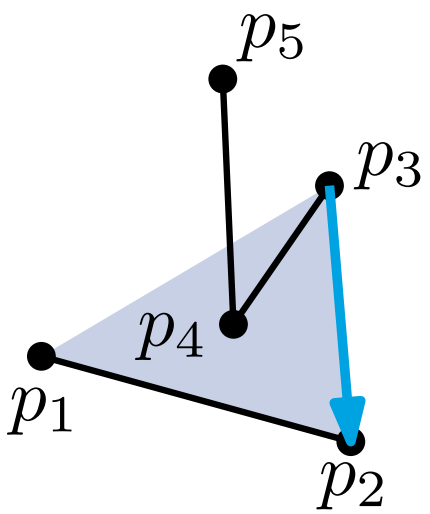
Q:



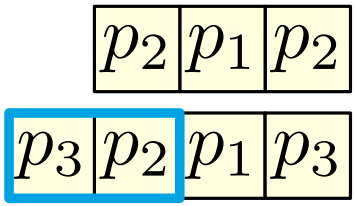
build und split Operationen

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

$Q \leftarrow$ deque (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ im UZS */
$H \leftarrow$ history stack /* merkt sich Operationen von Q */
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)



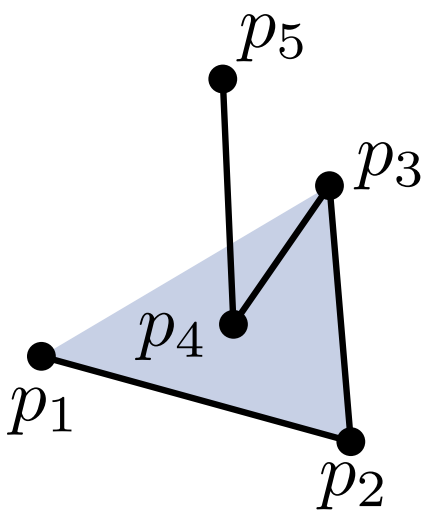
Q:



build und split Operationen

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

$Q \leftarrow$ deque (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ im UZS */
$H \leftarrow$ history stack /* merkt sich Operationen von Q */
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)



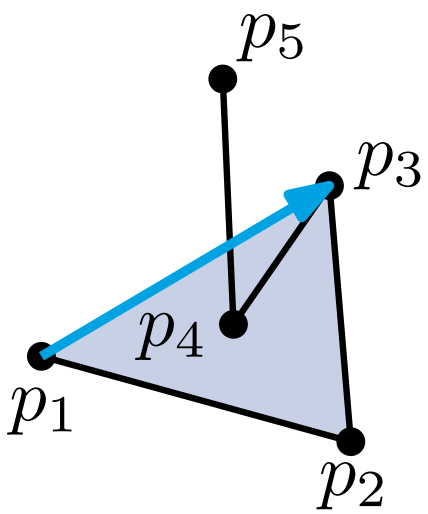
Q:

		p_2	p_1	p_2
p_3	p_2	p_1	p_3
p_3	p_2	p_1	p_3

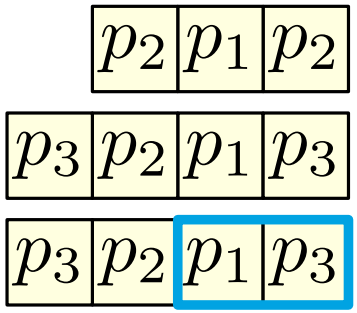
build und split Operationen

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

$Q \leftarrow$ deque (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ im UZS */
$H \leftarrow$ history stack /* merkt sich Operationen von Q */
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)



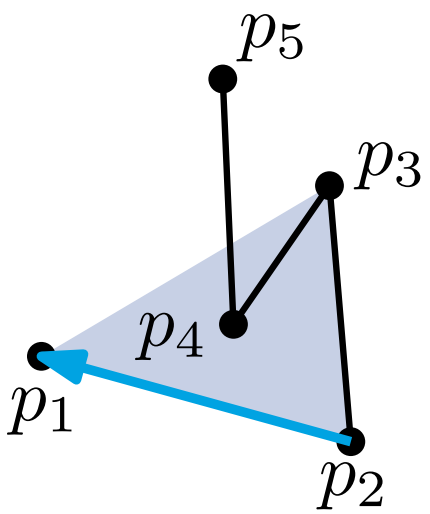
Q:



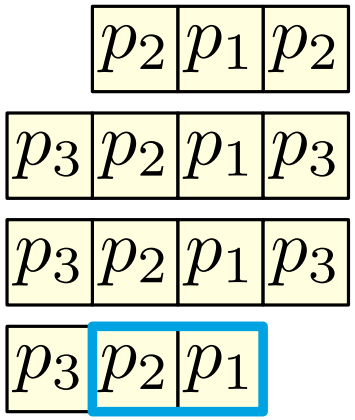
build und split Operationen

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

$Q \leftarrow$ deque (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ im UZS */
$H \leftarrow$ history stack /* merkt sich Operationen von Q */
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)



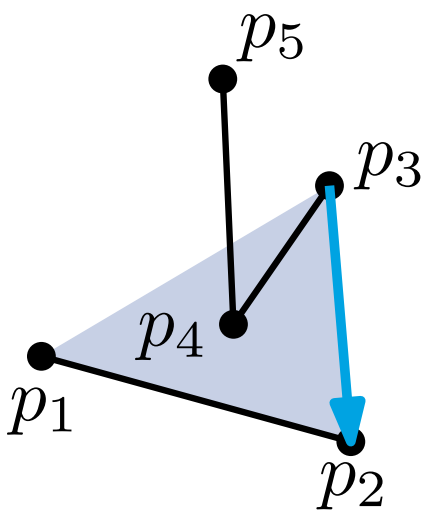
Q:



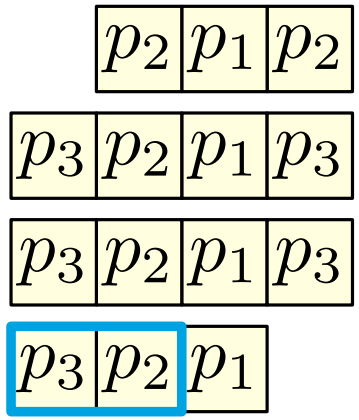
build und split Operationen

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

$Q \leftarrow$ deque (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ im UZS */
$H \leftarrow$ history stack /* merkt sich Operationen von Q */
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)



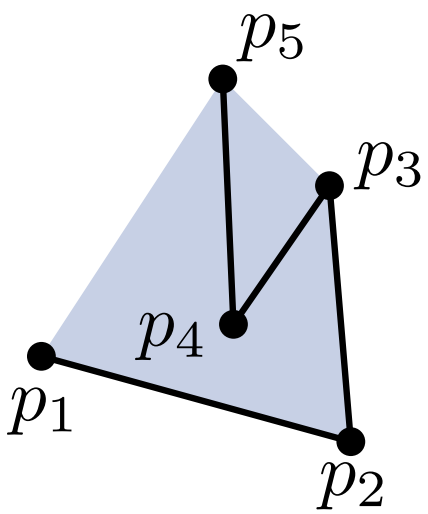
Q:



build und split Operationen

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

$Q \leftarrow$ deque (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ im UZS */
$H \leftarrow$ history stack /* merkt sich Operationen von Q */
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)



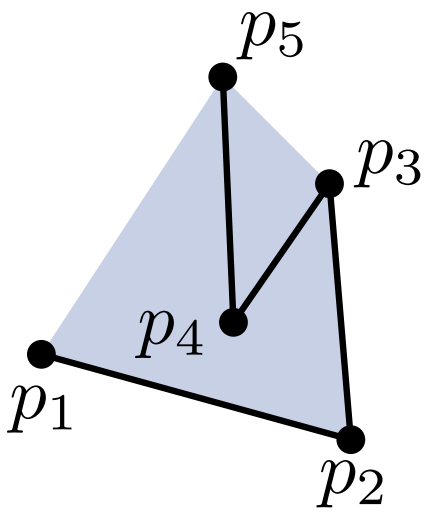
Q:

		p_2	p_1	p_2
	p_3	p_2	p_1	p_3
	p_3	p_2	p_1	p_3
p_5	p_3	p_2	p_1	p_5

build und split Operationen

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

$Q \leftarrow$ deque (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ im UZS */
$H \leftarrow$ history stack /* merkt sich Operationen von Q */
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)



Q:

		p_2	p_1	p_2
	p_3	p_2	p_1	p_3
	p_3	p_2	p_1	p_3
p_5	p_3	p_2	p_1	p_5

P muss einfach sein!

Warum?

$\rightarrow$ Melkman's Algorithmus für konvexe Hüllen

build und split Operationen

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

$Q \leftarrow$ deque (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ im UZS */
$H \leftarrow$ history stack /* merkt sich Operationen von Q */
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 $k = m - 2, \ldots, i$

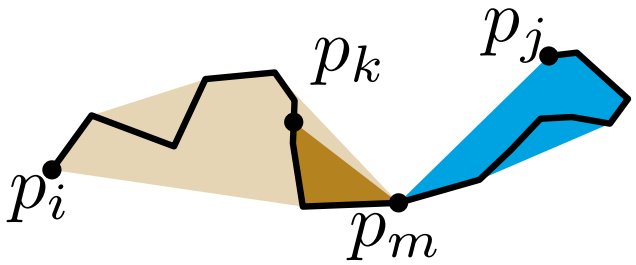
build und split Operationen

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

$Q \leftarrow$ deque (p_{m+1}, p_m, p_{m+1}) /* $CH(p_m, \ldots, p_k)$ im UZS */
$H \leftarrow$ history stack /* merkt sich Operationen von Q */
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



Laufzeit

drei Operationen:

- **farthest**(PH, ℓ): finde weitesten von ℓ entfernten Knoten in PH
- **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

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

$\rightarrow$ Beweis s. Tafel

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)