Offline Bewegungsplanung: Polyeder

Elmar Langetepe
University of Bonn

Polyeder-Szene in 3D

Polyeder-Szene in 3D

- Startpunkt s,

Polyeder-Szene in 3D

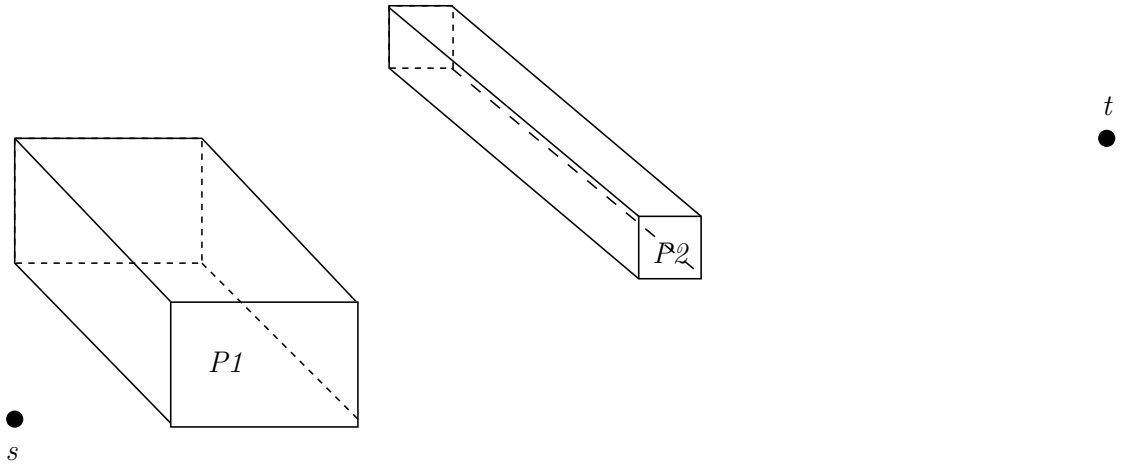
- Startpunkt s, Zielpunkt t

Polyeder-Szene in 3D

- Startpunkt s, Zielpunkt t
- Menge von Polyedern

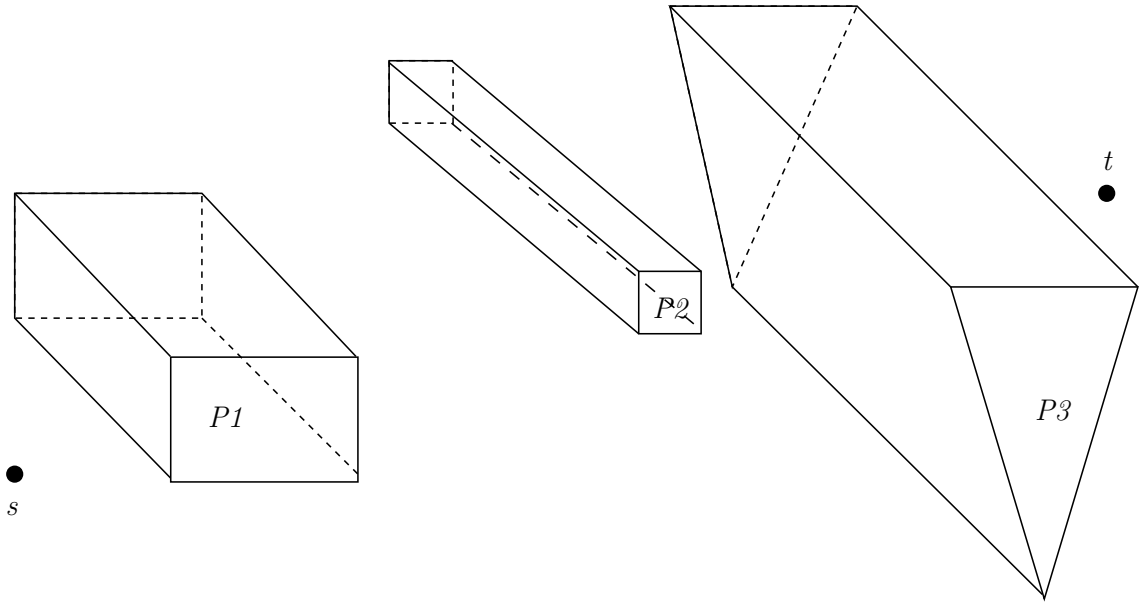
Polyeder-Szene in 3D

- Startpunkt s, Zielpunkt t
- Menge von Polyedern



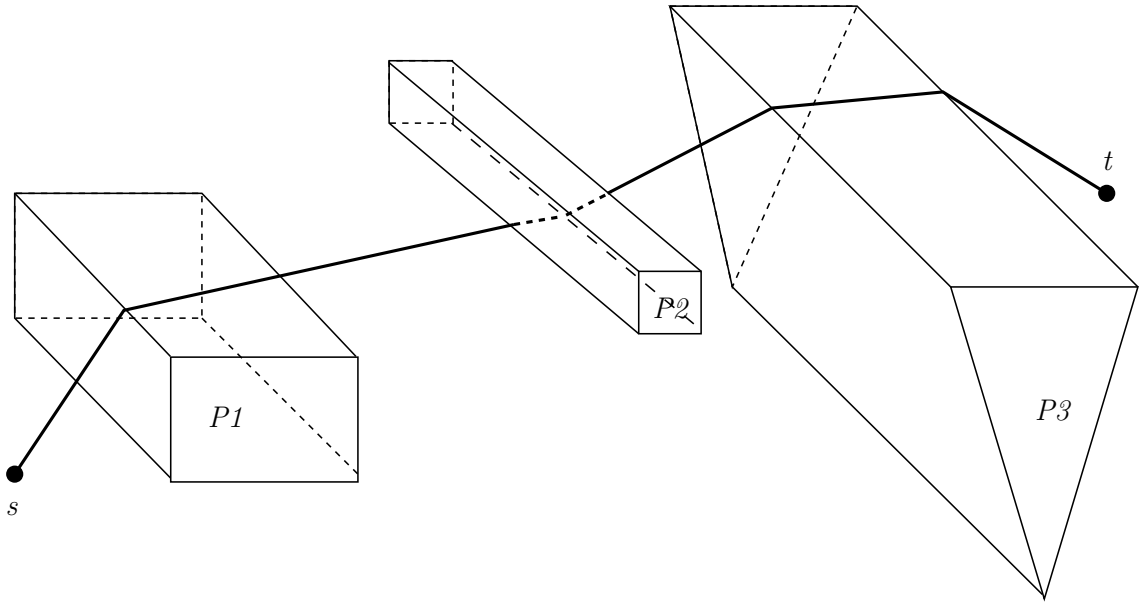
Polyeder-Szene in 3D

- Startpunkt s, Zielpunkt t
- Menge von Polyedern



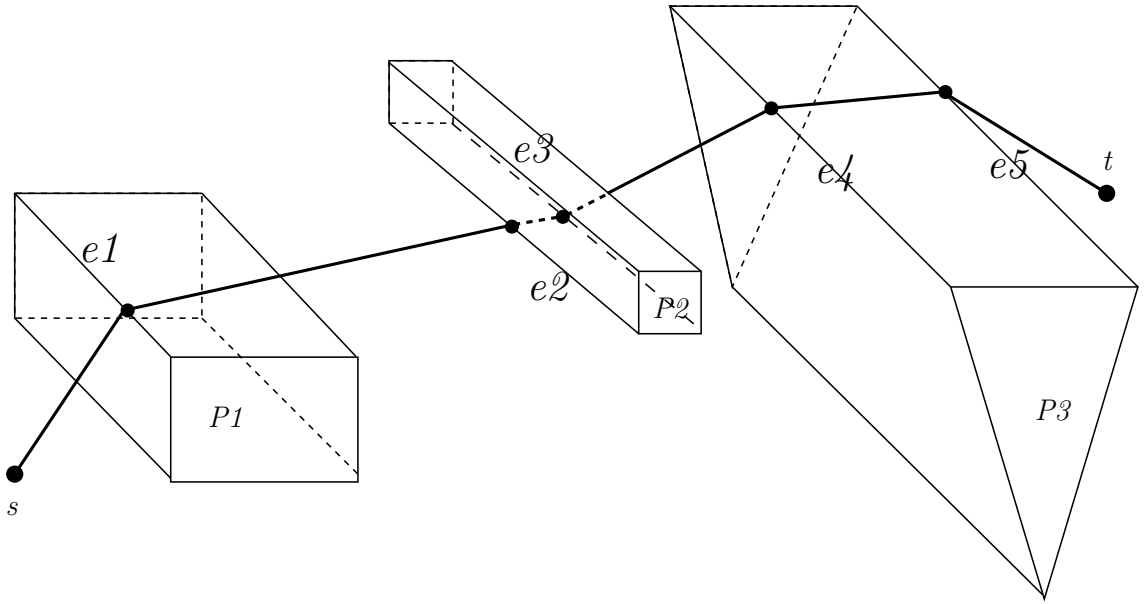
Polyeder-Szene in 3D

- Startpunkt s, Zielpunkt t
- Menge von Polyedern
- Kürzester Weg von s nach t:



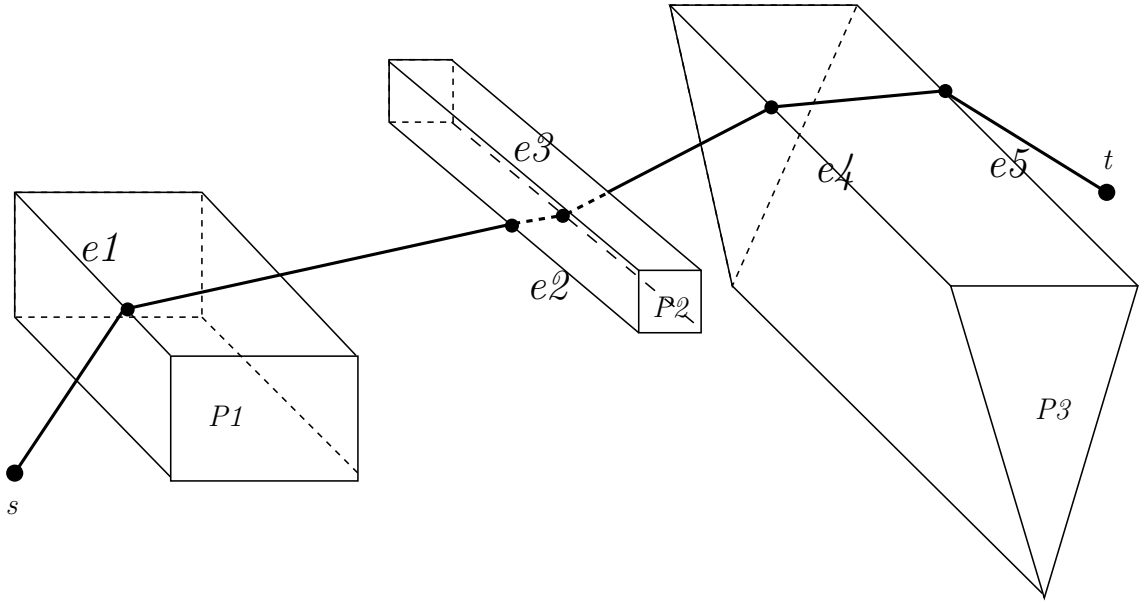
Polyeder-Szene in 3D

- Startpunkt s, Zielpunkt t
- Menge von Polyedern
- Kürzester Weg von s nach t:

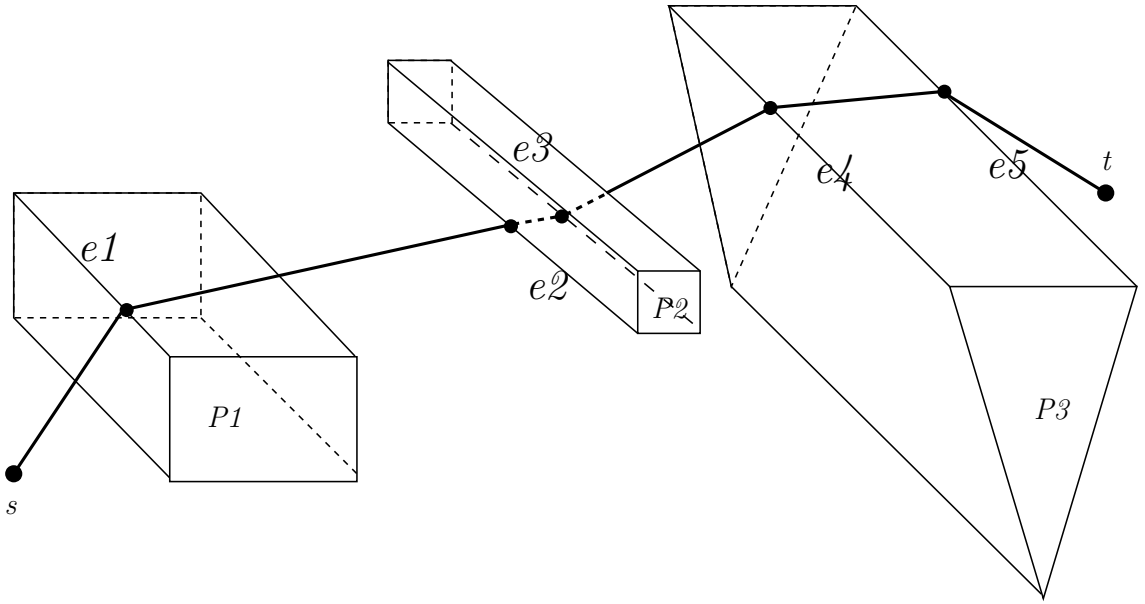


Polyeder-Szene in 3D

- Startpunkt s, Zielpunkt t
- Menge von Polyedern
- Kürzester Weg von s nach t: NP-hard

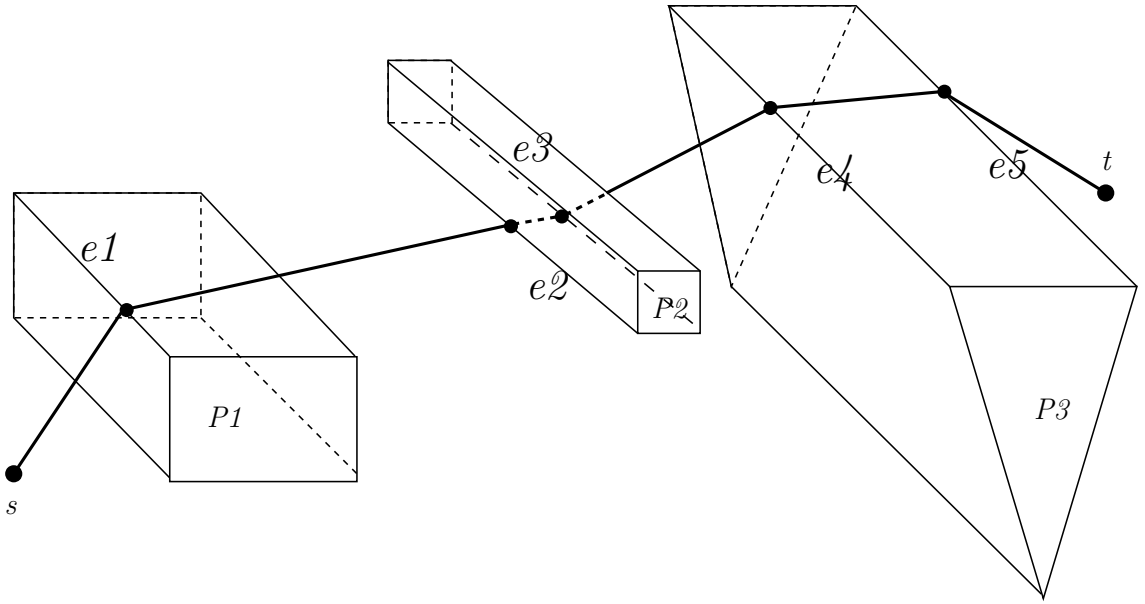


Polyeder-Szene in 3D



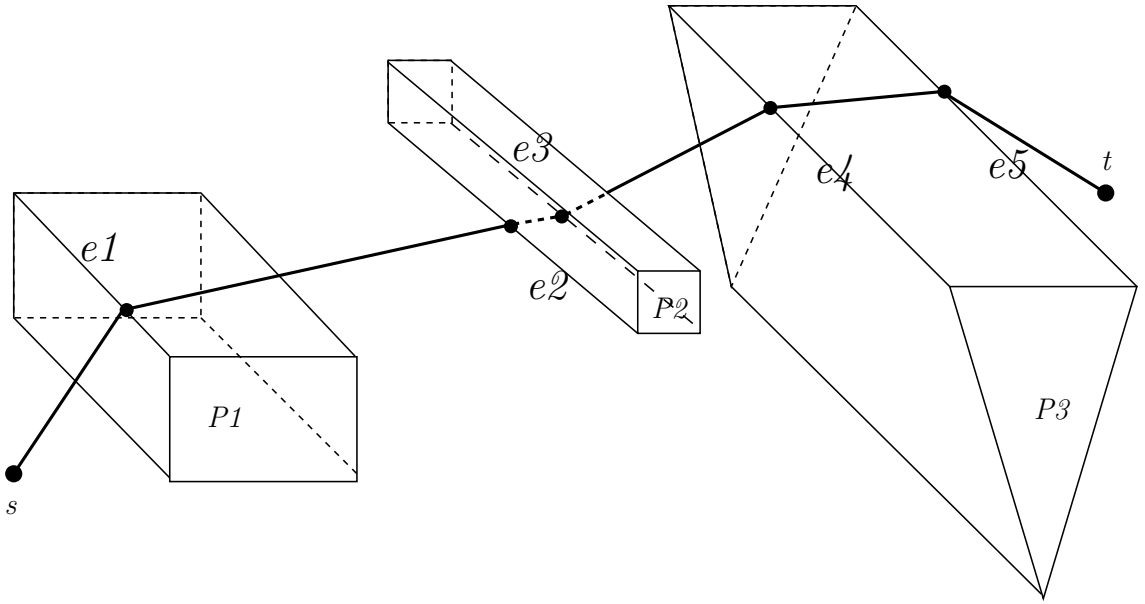
Polyeder-Szene in 3D

- Teilprobleme



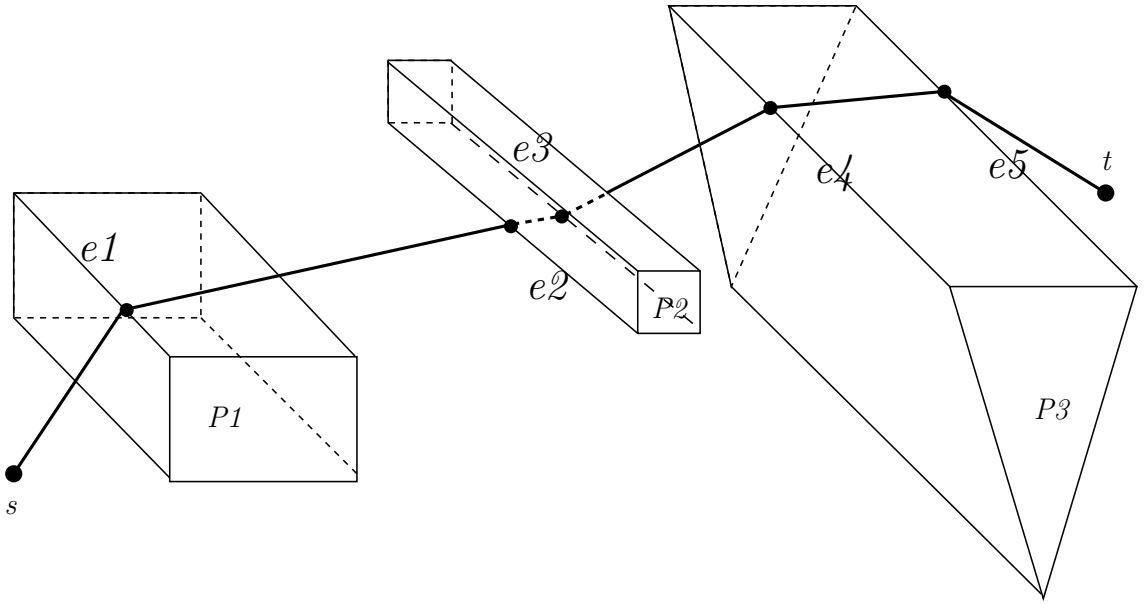
Polyeder-Szene in 3D

- Teilprobleme
- 1) Kantenreihenfolge



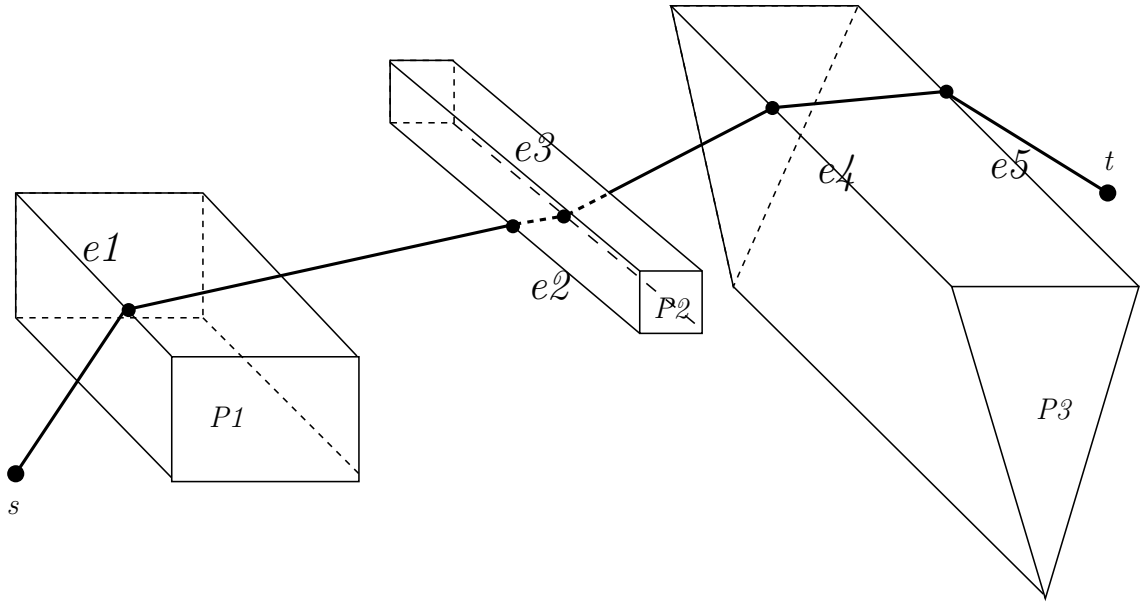
Polyeder-Szene in 3D

- Teilprobleme
- 1) Kantenreihenfolge
- 2) Verschiebung auf der Kante



Polyeder-Szene in 3D

- Teilprobleme
- 1) Kantenreihenfolge
- 2) Verschiebung auf der Kante
- Bereits 1) ist NP hard



Kantenreihenfolge: NP hard

Kantenreihenfolge: NP hard

- Reduktion $S' \subset \Omega'$ auf $S \subset \Omega$

Kantenreihenfolge: NP hard

- Reduktion $S' \subset \Omega'$ auf $S \subset \Omega$
- Funktion $f : \Omega' \rightarrow \Omega$

Kantenreihenfolge: NP hard

- Reduktion $S' \subset \Omega'$ auf $S \subset \Omega$
- Funktion $f : \Omega' \to \Omega$

1. $\forall x' \in \Omega'$: $f(x')$ in polynomieller Zeit ($|x'|$)
2. $\forall x' \in \Omega'$: $f(x') \in S \Leftrightarrow x' \in S'$

Kantenreihenfolge: NP hard

- Reduktion $S' \subset \Omega'$ auf $S \subset \Omega$
- Funktion $f : \Omega' \to \Omega$

  1. $\forall x' \in \Omega'$: $f(x')$ in polynomieller Zeit ($|x'|$)
  2. $\forall x' \in \Omega'$: $f(x') \in S \Leftrightarrow x' \in S'$

- 3-SAT NP vollständig

Kantenreihenfolge: NP hard

- Reduktion $S' \subset \Omega'$ auf $S \subset \Omega$
- Funktion $f : \Omega' \to \Omega$

1. $\forall x' \in \Omega'$: $f(x')$ in polynomieller Zeit ($|x'|$)
2. $\forall x' \in \Omega'$: $f(x') \in S \Leftrightarrow x' \in S'$

- 3-SAT NP vollständig
- 3-SAT reduzieren auf Kantenreihenfolge

3-SAT reduzieren auf Kantenreihenfolge

3-SAT reduzieren auf Kantenreihenfolge

$$\alpha = \bigwedge_{i=1}^{m} (L_{i_1} \vee L_{i_2} \vee L_{i_3}) \text{ mit } L_{i_j} \in \{X_k, \neg X_k\}$$

3-SAT reduzieren auf Kantenreihenfolge

$$\alpha = \bigwedge_{i=1}^{m} (L_{i_1} \vee L_{i_2} \vee L_{i_3}) \text{ mit } L_{i_j} \in \{X_k, \neg X_k\}$$

m Klauseln mit n Variablen:

3-SAT reduzieren auf Kantenreihenfolge

$$\alpha = \bigwedge_{i=1}^{m} (L_{i_1} \vee L_{i_2} \vee L_{i_3}) \text{ mit } L_{i_j} \in \{X_k, \neg X_k\}$$

m Klauseln mit n Variablen: Erfüllbarkeit?

3-SAT reduzieren auf Kantenreihenfolge

$$\alpha = \bigwedge_{i=1}^{m} (L_{i_1} \vee L_{i_2} \vee L_{i_3}) \text{ mit } L_{i_j} \in \{X_k, \neg X_k\}$$

m Klauseln mit n Variablen: Erfüllbarkeit?

Konstruiere Parcours P_α, so dass:

3-SAT reduzieren auf Kantenreihenfolge

$$\alpha = \bigwedge_{i=1}^{m} (L_{i_1} \vee L_{i_2} \vee L_{i_3}) \text{ mit } L_{i_j} \in \{X_k, \neg X_k\}$$

m Klauseln mit n Variablen: Erfüllbarkeit?

Konstruiere Parcours P_α, so dass:

- Kürzester Weg (Kantenfolge) von s nach t erzeugt Belegung w

3-SAT reduzieren auf Kantenreihenfolge

$$\alpha = \bigwedge_{i=1}^{m} (L_{i_1} \vee L_{i_2} \vee L_{i_3}) \text{ mit } L_{i_j} \in \{X_k, \neg X_k\}$$

m Klauseln mit n Variablen: Erfüllbarkeit?

Konstruiere Parcours P_α, so dass:

- Kürzester Weg (Kantenfolge) von s nach t erzeugt Belegung w
- w erfüllt $\alpha \Rightarrow$ fertig!

3-SAT reduzieren auf Kantenreihenfolge

$$\alpha = \bigwedge_{i=1}^{m} (L_{i_1} \vee L_{i_2} \vee L_{i_3}) \text{ mit } L_{i_j} \in \{X_k, \neg X_k\}$$

m Klauseln mit n Variablen: Erfüllbarkeit?

Konstruiere Parcours P_α, so dass:

- Kürzester Weg (Kantenfolge) von s nach t erzeugt Belegung w
- w erfüllt $\alpha \Rightarrow$ fertig!
- w erfüllt α nicht $\Rightarrow$ kein w erfüllt α

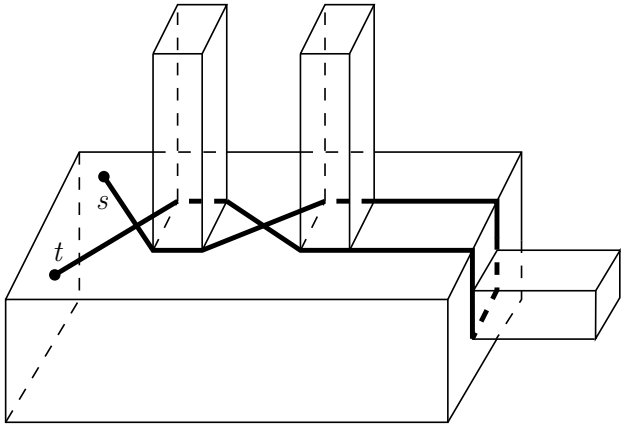
Parcours in $O(p(mn))$ erzeugen

Parcours in $O(p(mn))$ erzeugen

- 2^n Belegungen der n Variablen

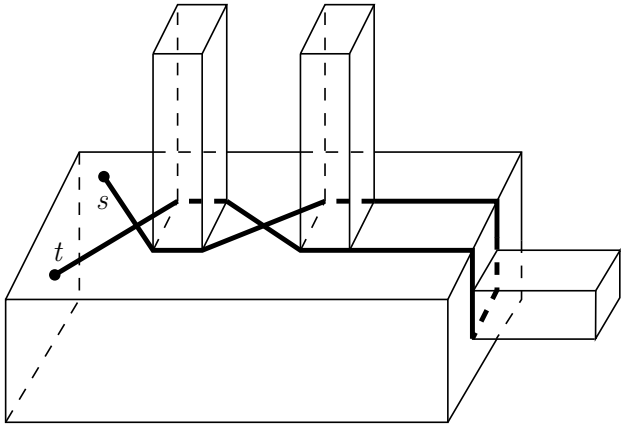
Parcours in $O(p(mn))$ erzeugen

- 2^n Belegungen der n Variablen
- 2^n *geodätisch* Kürzeste Wege



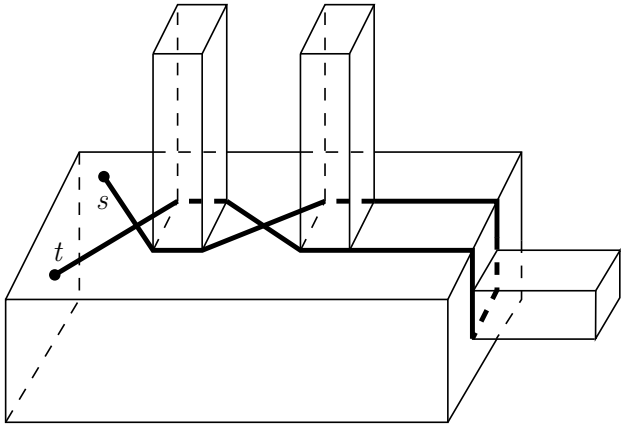
Parcours in $O(p(mn))$ erzeugen

- 2^n Belegungen der n Variablen
- 2^n *geodätisch* Kürzeste Wege
- Eine davon wird die Kürzeste sein



Parcours in $O(p(mn))$ erzeugen

- 2^n Belegungen der n Variablen
- 2^n *geodätisch* Kürzeste Wege
- Eine davon wird die Kürzeste sein
- Ergibt Variablen-Belegung nach Kantenreihenfolge



Parcours erzeugen: Prinzip

Parcours erzeugen: Prinzip

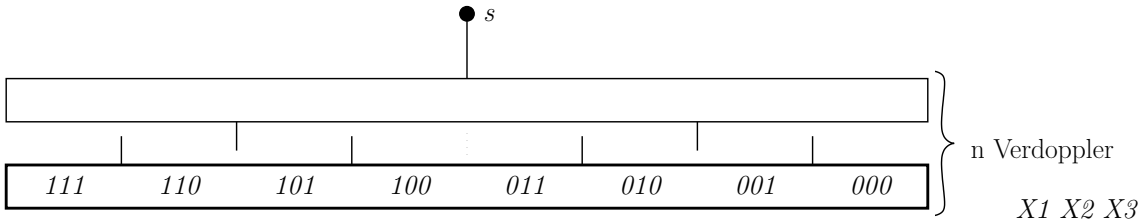
Beispiel: $(X1 \vee \neg X3) \wedge (\neg X2)$

Parcours erzeugen: Prinzip

Beispiel: $(X1 \vee \neg X3) \wedge (\neg X2)$

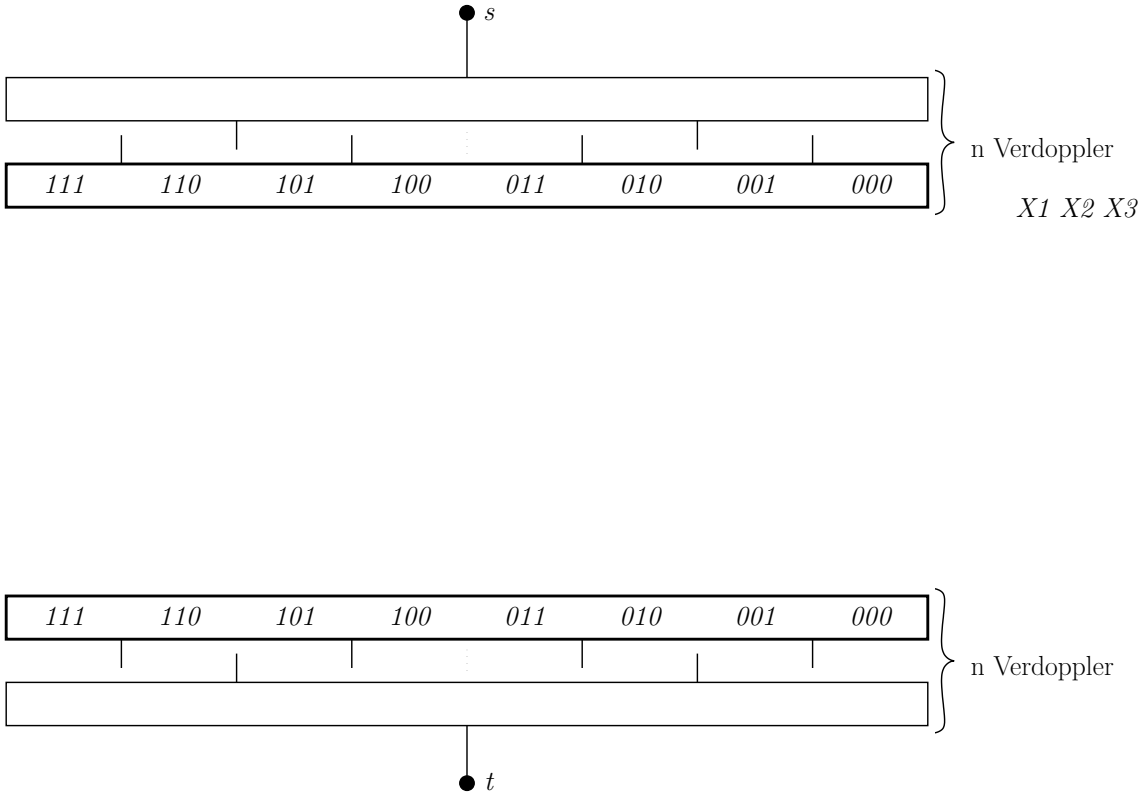
Parcours erzeugen: Prinzip

Beispiel: $(X1 \vee \neg X3) \wedge (\neg X2)$



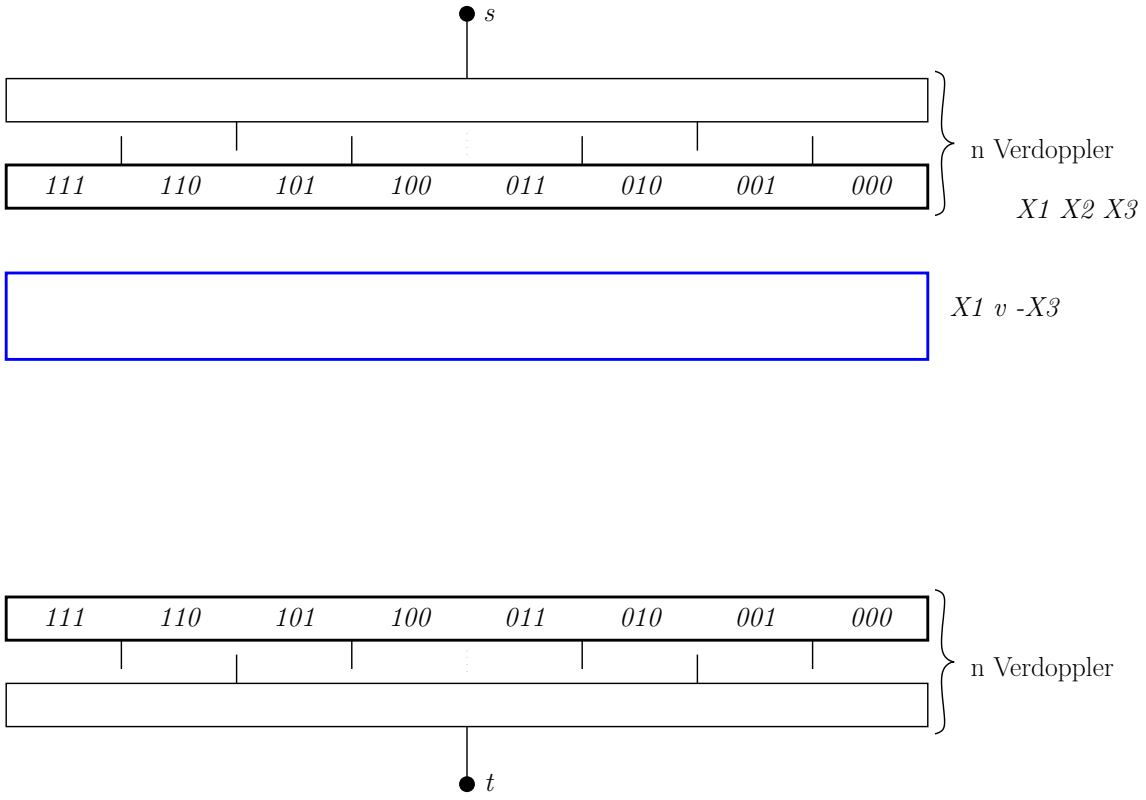
Parcours erzeugen: Prinzip

Beispiel: $(X1 \vee \neg X3) \wedge (\neg X2)$



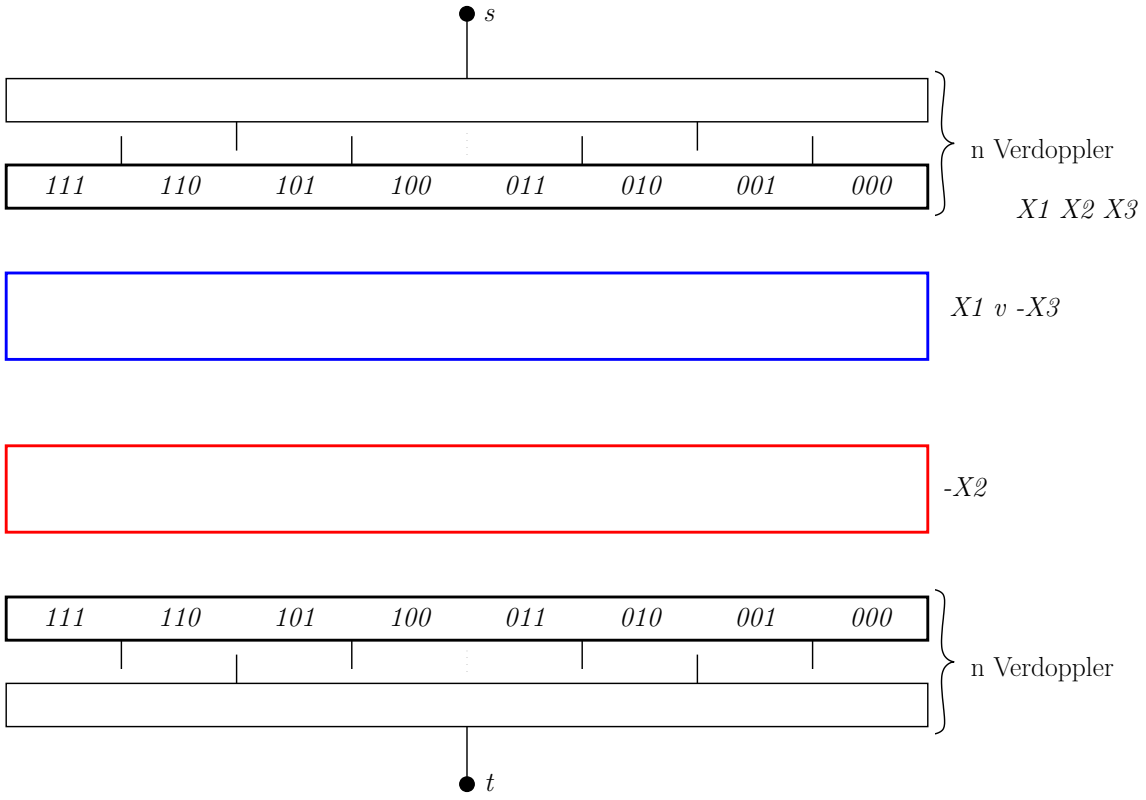
Parcours erzeugen: Prinzip

Beispiel: $(X1 \vee \neg X3) \wedge (\neg X2)$



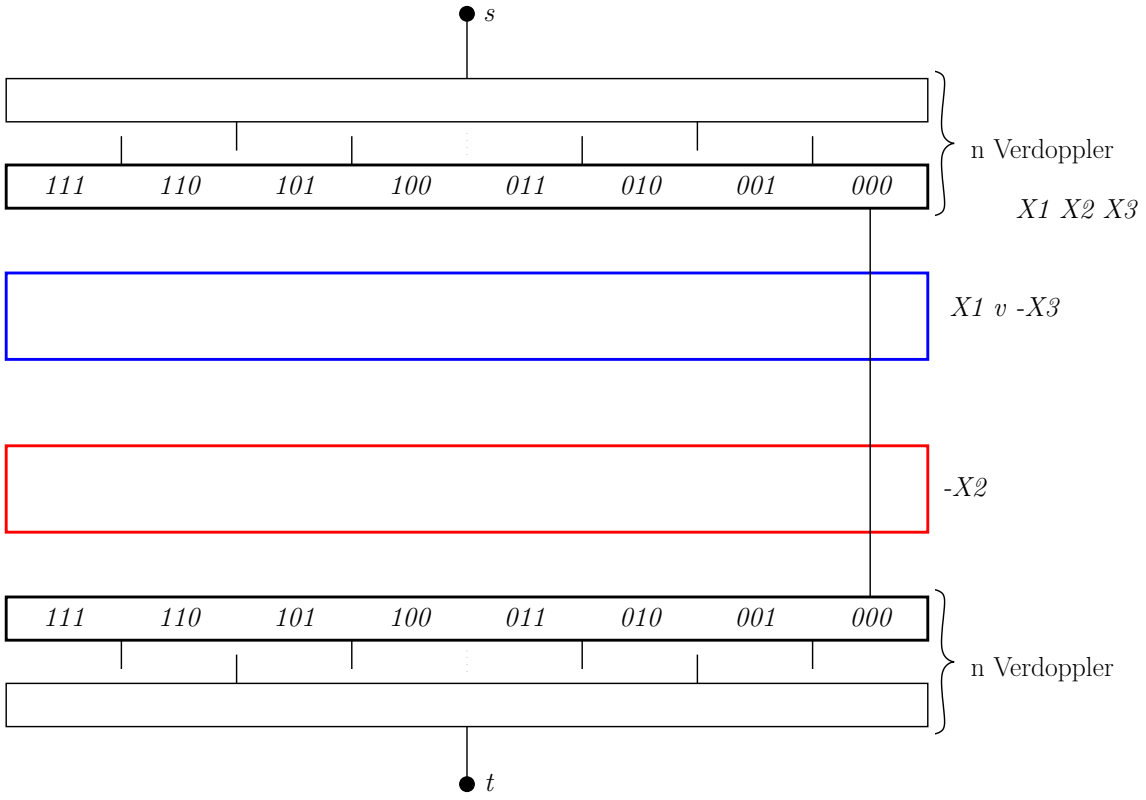
Parcours erzeugen: Prinzip

Beispiel: $(X1 \vee \neg X3) \wedge (\neg X2)$



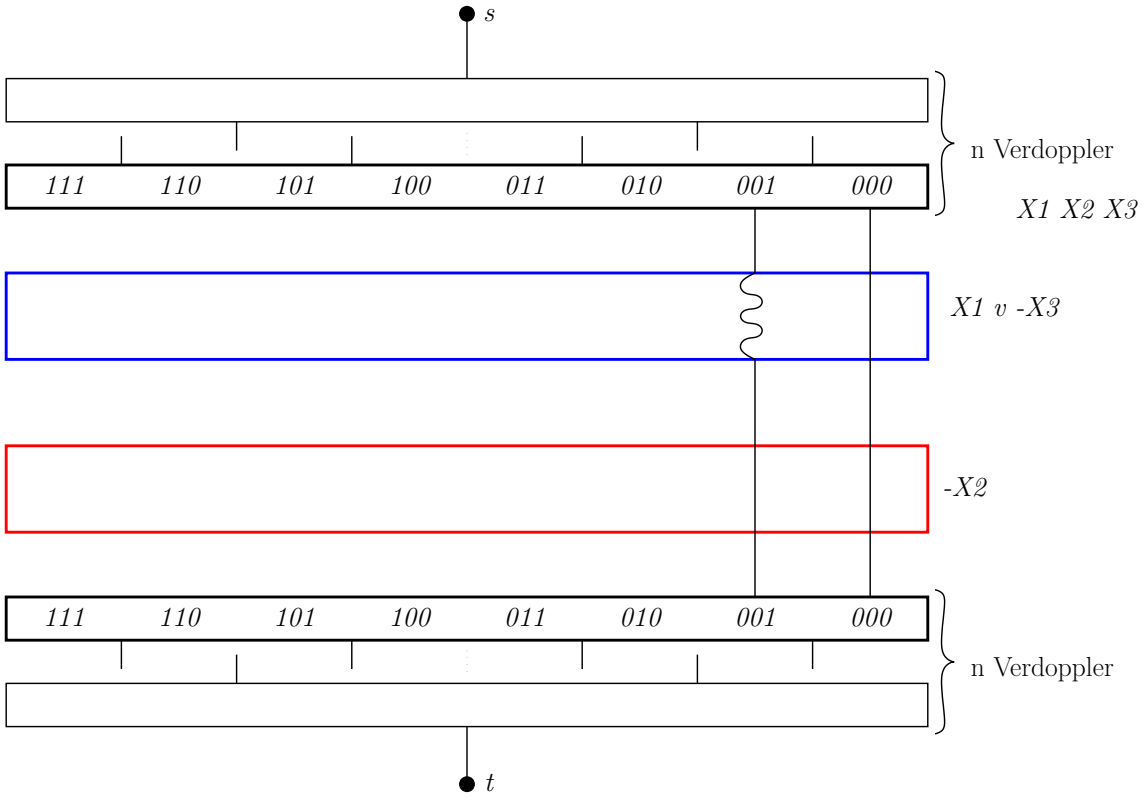
Parcours erzeugen: Prinzip

Beispiel: $(X1 \vee \neg X3) \wedge (\neg X2)$



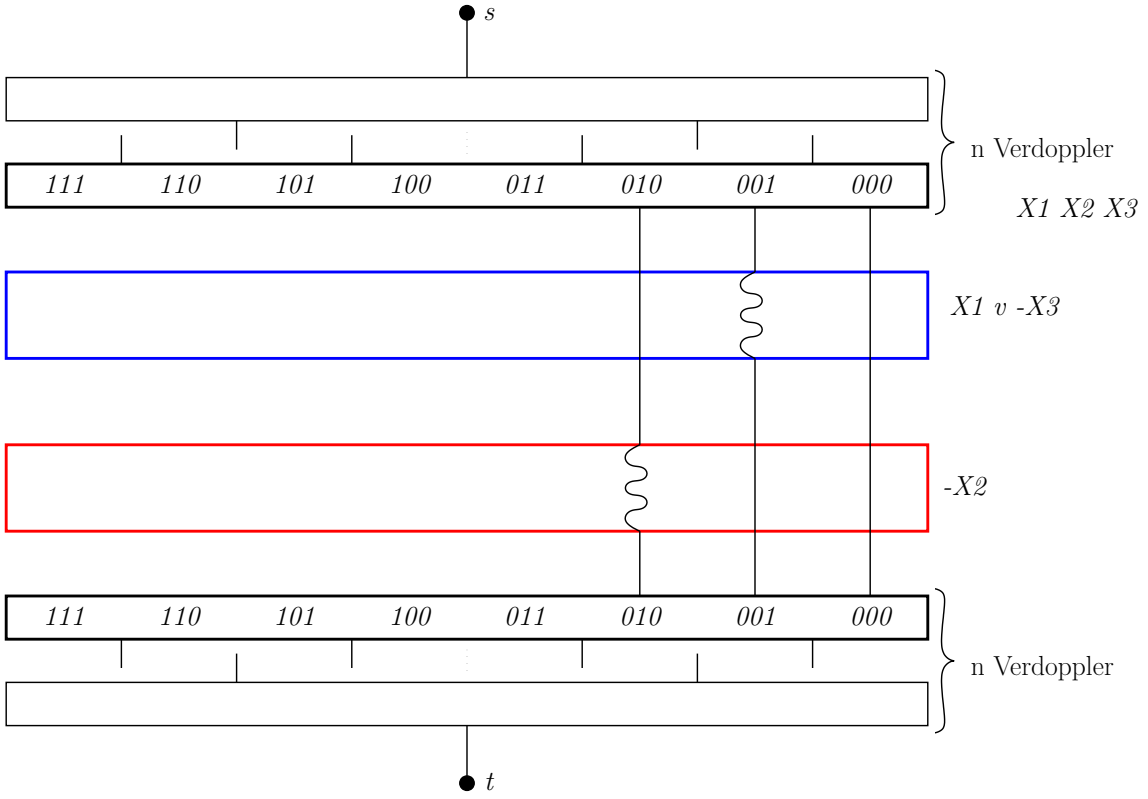
Parcours erzeugen: Prinzip

Beispiel: $(X1 \vee \neg X3) \wedge (\neg X2)$



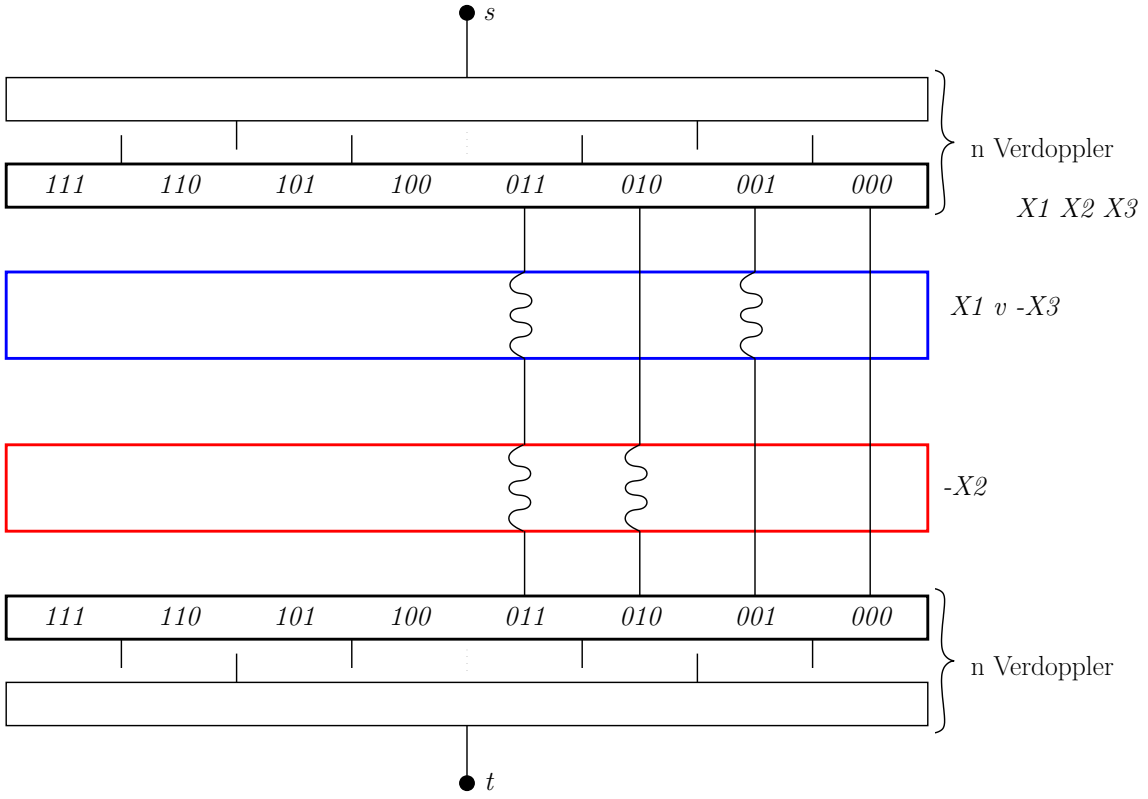
Parcours erzeugen: Prinzip

Beispiel: $(X1 \vee \neg X3) \wedge (\neg X2)$



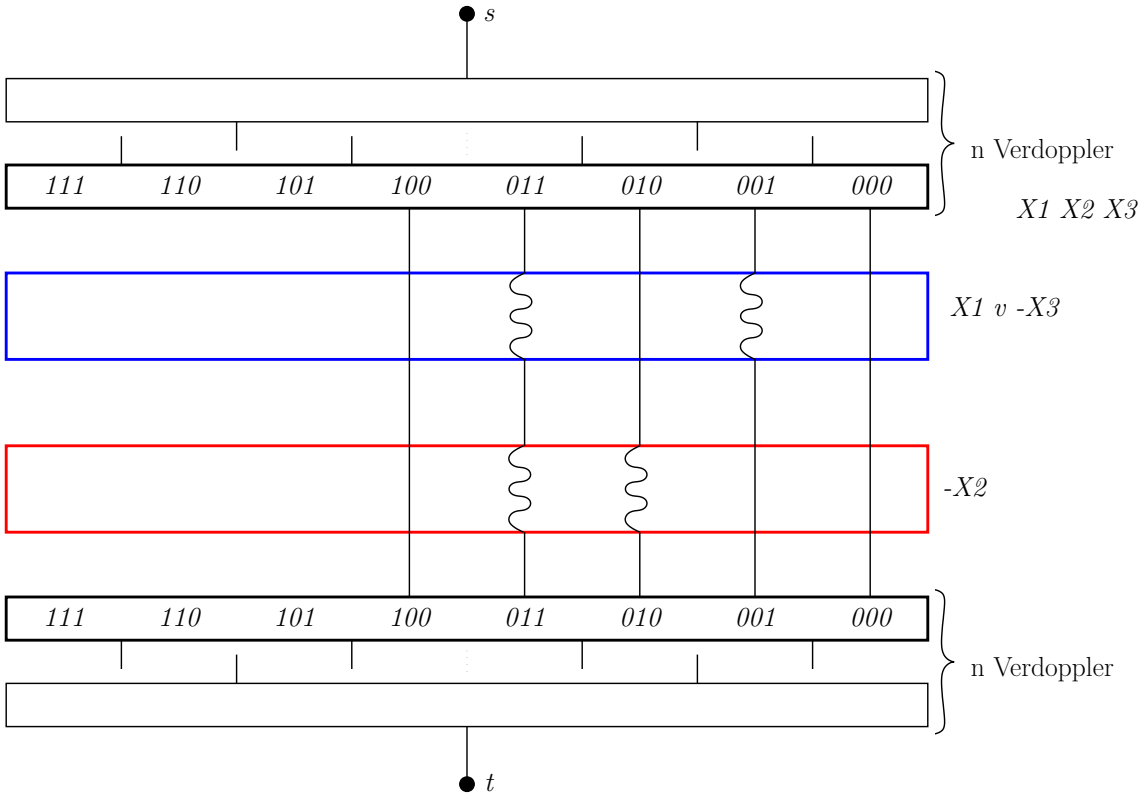
Parcours erzeugen: Prinzip

Beispiel: $(X1 \vee \neg X3) \wedge (\neg X2)$



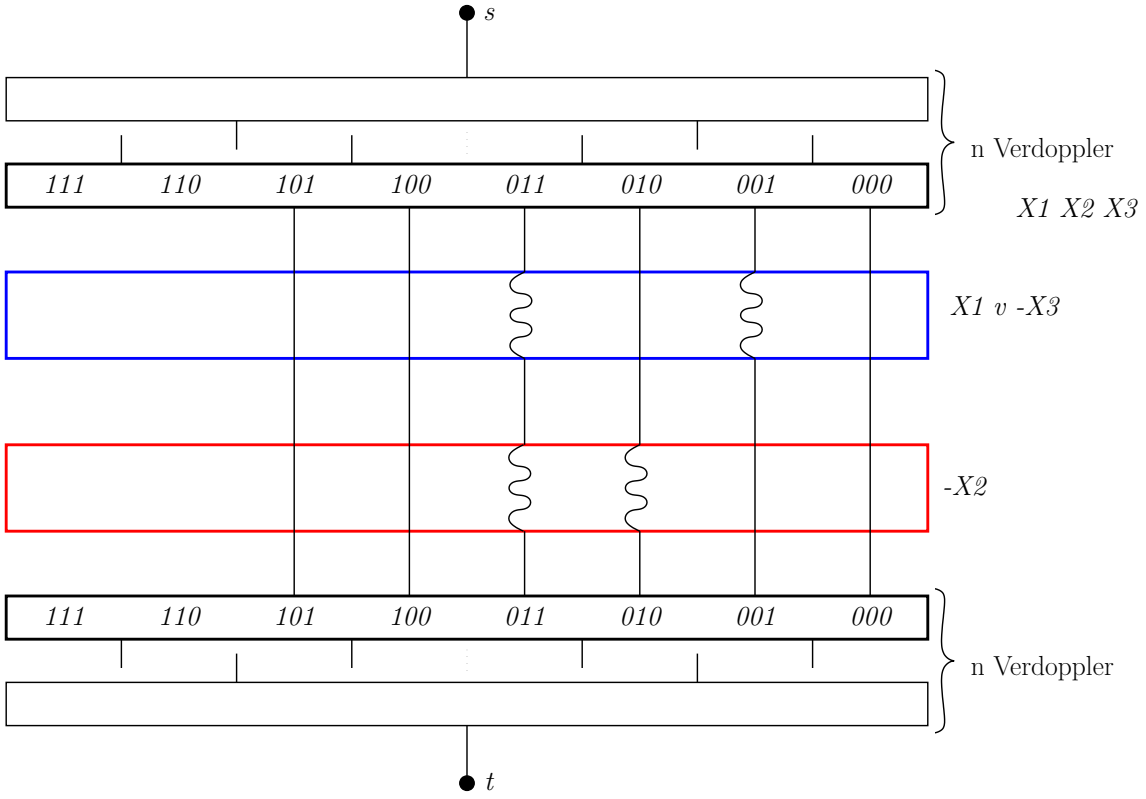
Parcours erzeugen: Prinzip

Beispiel: $(X1 \vee \neg X3) \wedge (\neg X2)$



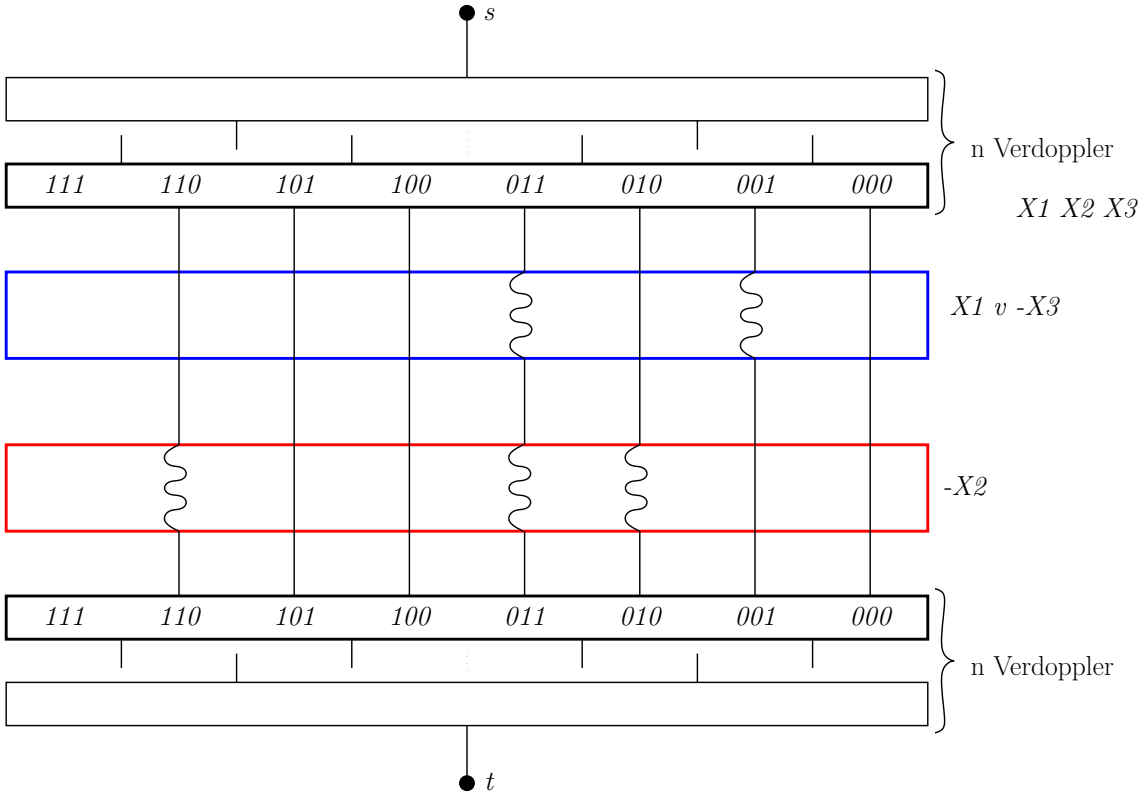
Parcours erzeugen: Prinzip

Beispiel: $(X1 \vee \neg X3) \wedge (\neg X2)$



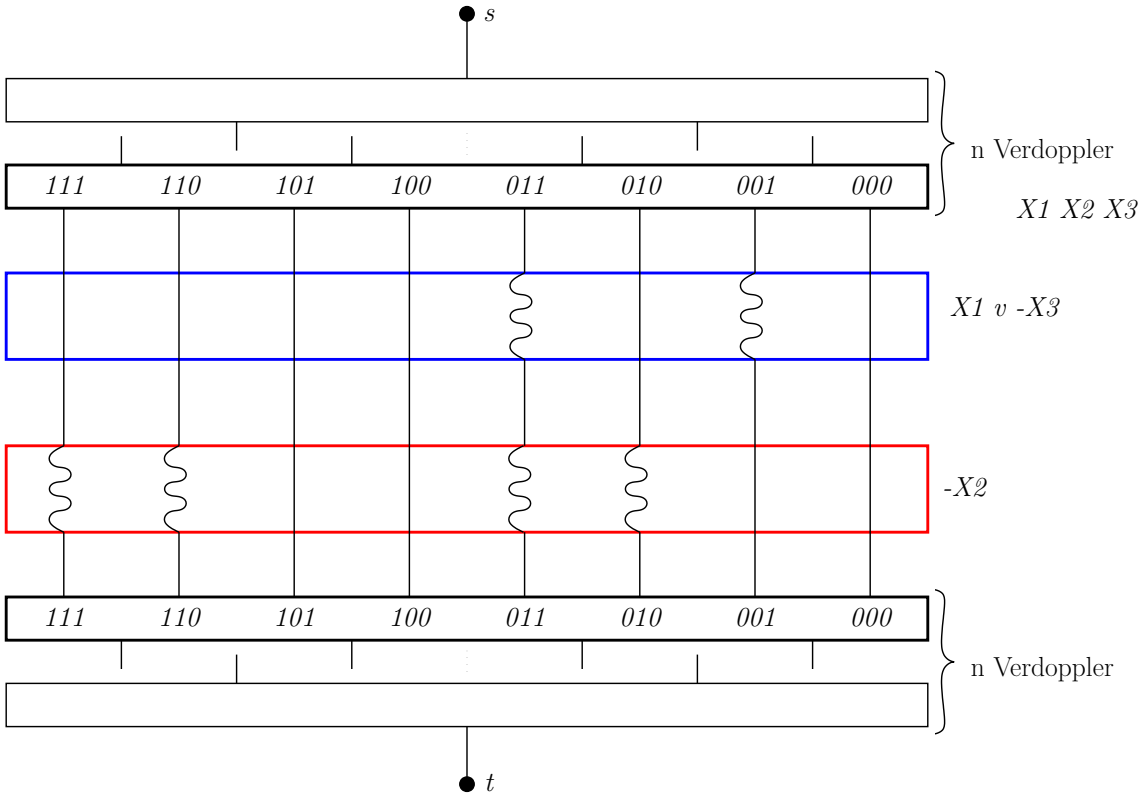
Parcours erzeugen: Prinzip

Beispiel: $(X1 \vee \neg X3) \wedge (\neg X2)$



Parcours erzeugen: Prinzip

Beispiel: $(X1 \vee \neg X3) \wedge (\neg X2)$



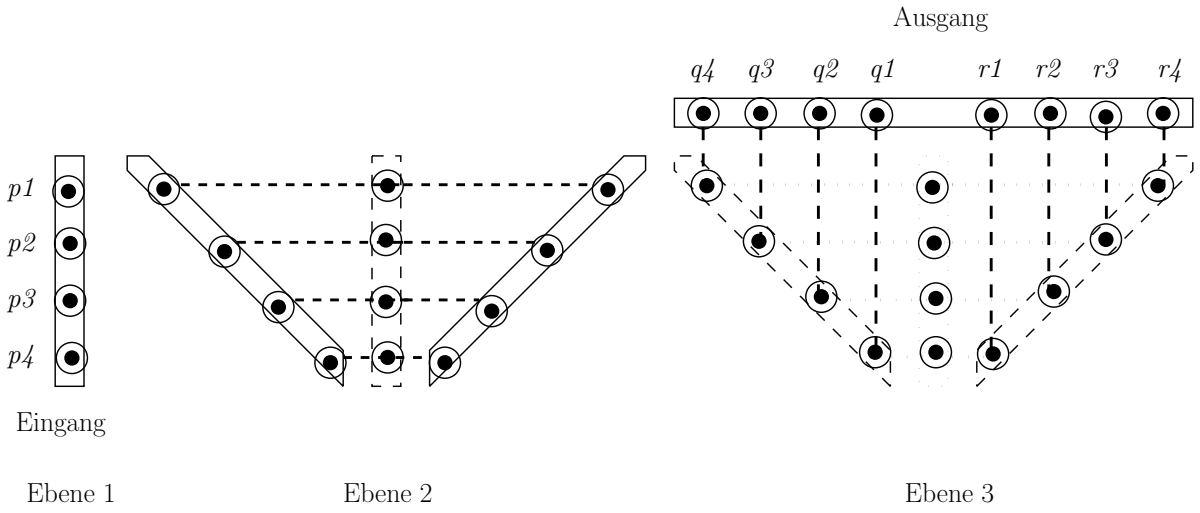
Komponenten: Verdoppler!

Komponenten: Verdoppler!

Dünne Platten mit Schlitzen eng hintereinander!

Komponenten: Verdoppler!

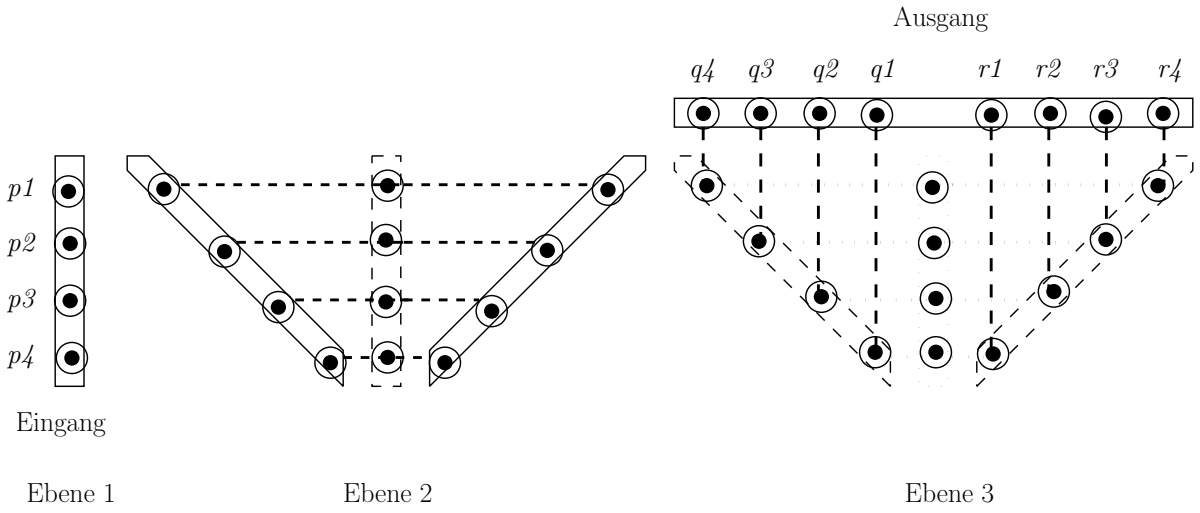
Dünne Platten mit Schlitzen eng hintereinander!



Sukzessive 2^n ungefähr gleichlange Wege erzeugen!

Komponenten: Verdoppler!

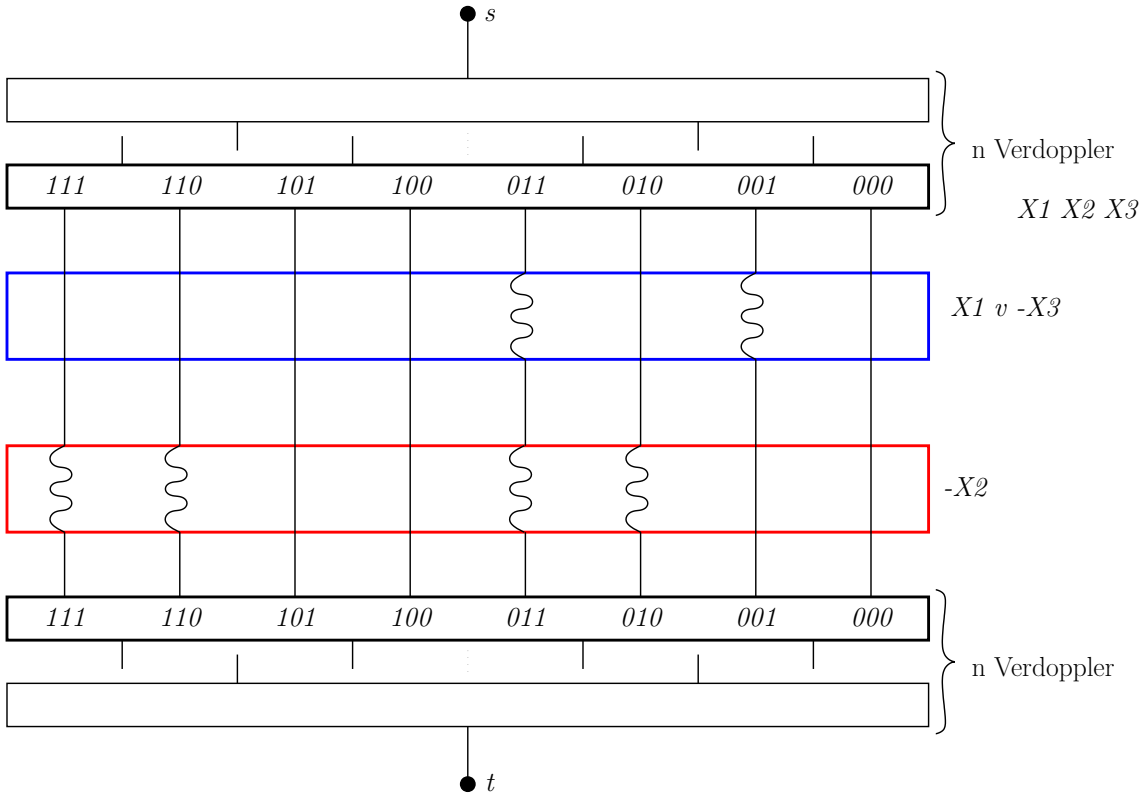
Dünne Platten mit Schlitzen eng hintereinander!



Sukzessive 2^n ungefähr gleichlange Wege erzeugen!

Kantenreihenfolge ist gleich!

Gesamtprinzip: Nacheinander Klauseln!



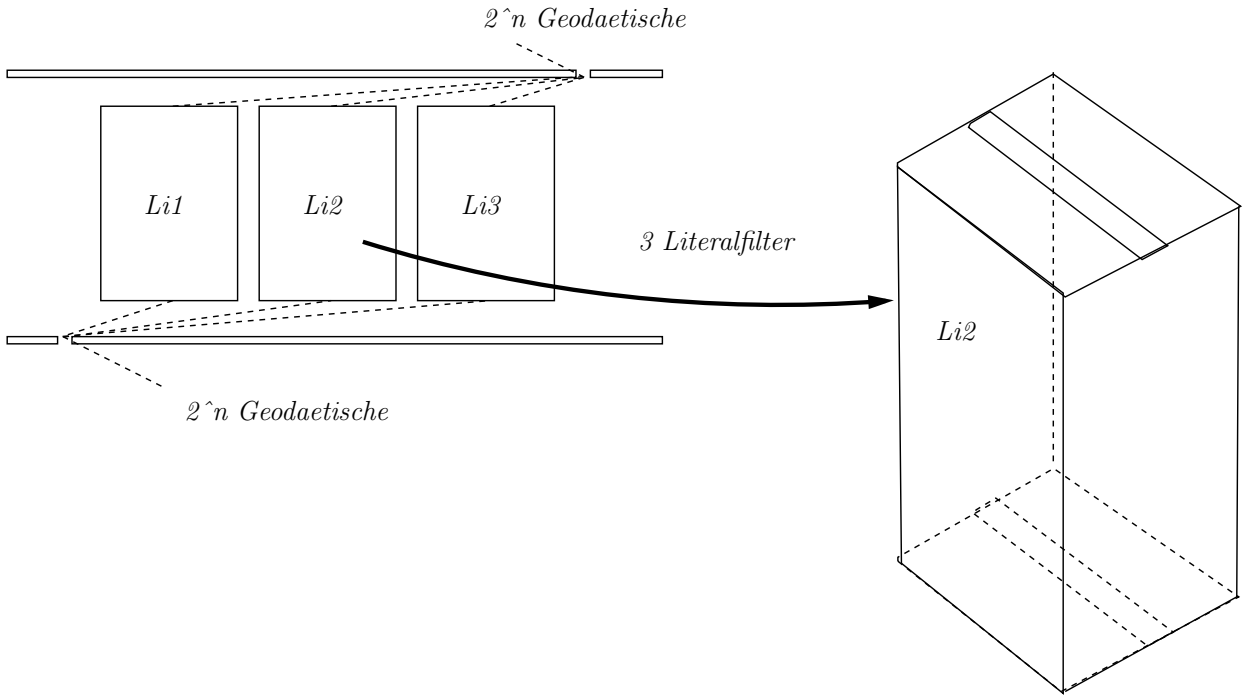
Komponenten: Klauselfilter

Komponenten: Klauselfilter

Sukzessive durch die Klauseln schicken! Auf Literale aufteilen!
Dünne Platten!

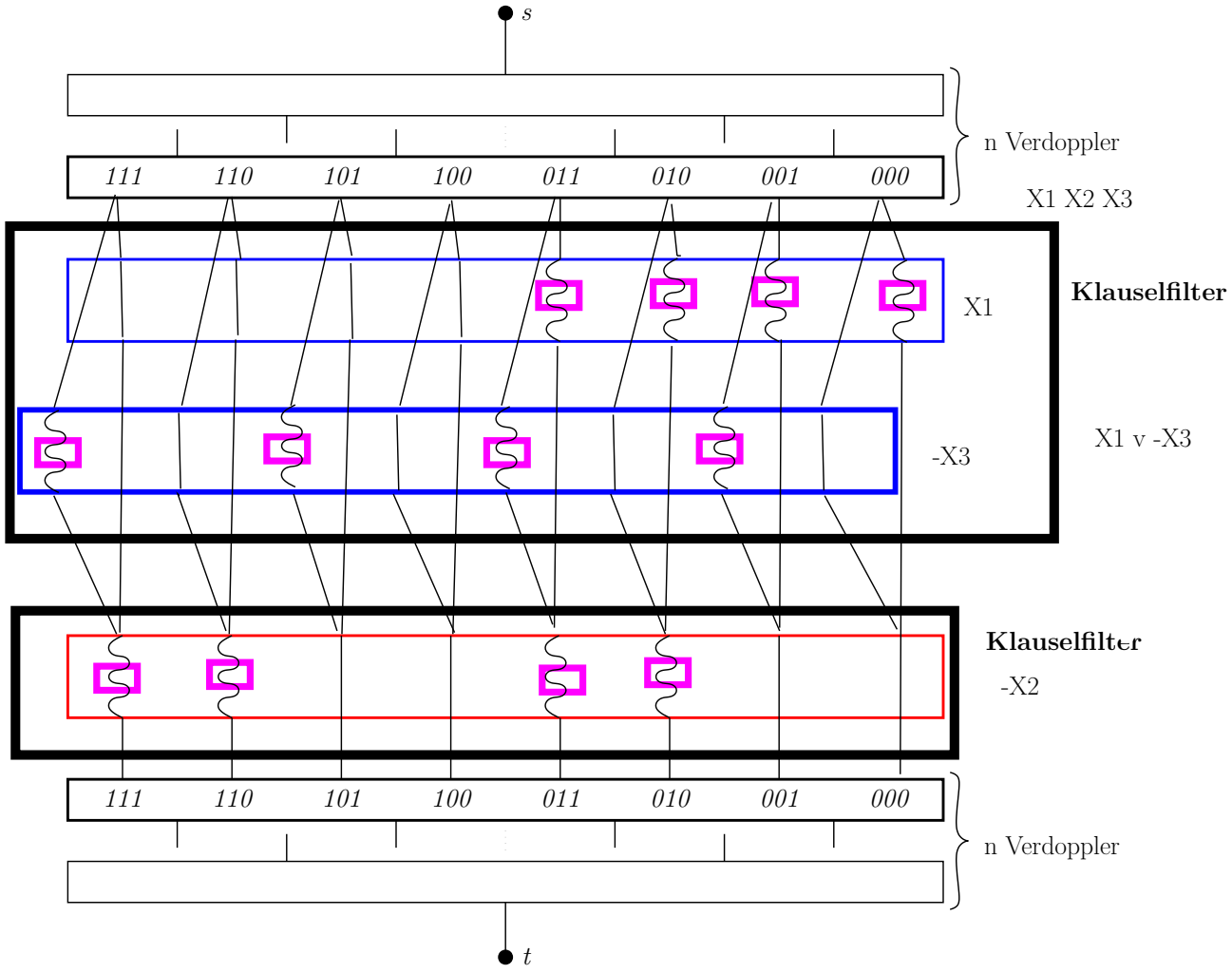
Komponenten: Klauselfilter

Sukzessive durch die Klauseln schicken! Auf Literale aufteilen!
Dünne Platten!



Gleich lang, bis auf das, was in den Literalfiltern passiert!

Gesamtprinzip: Einzelne Literale



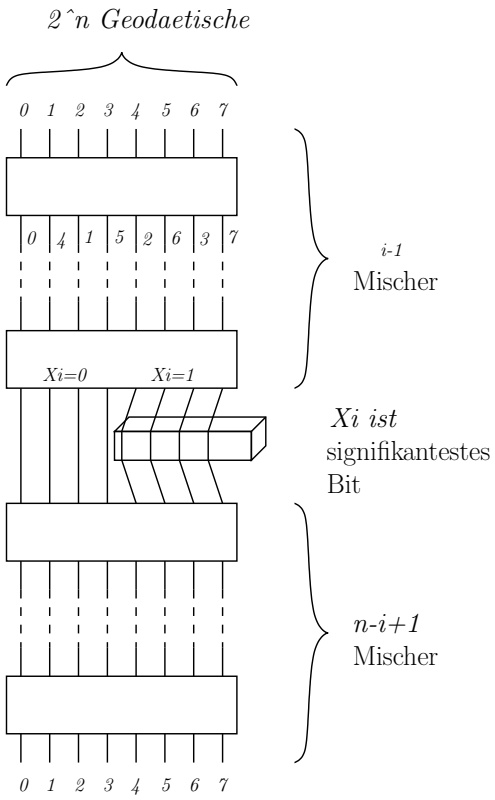
Komponenten: Literalfilter

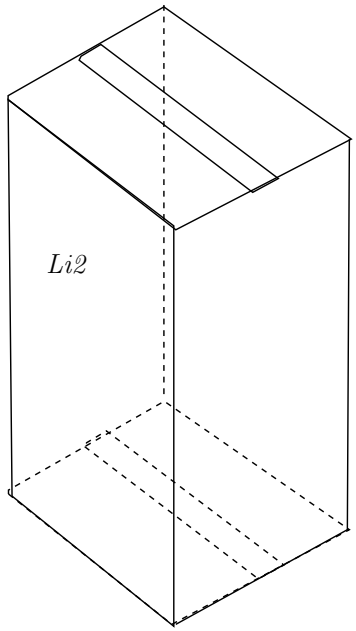
Komponenten: Literalfilter

Wege für signifikantes Bit sammeln!

Komponenten: Literalfilter

Wege für signifikantes Bit sammeln!





Falls X_i dann $X_i = 0$ verlängern! Falls $\neg X_i$ dann $X_i = 1$ verlängern!

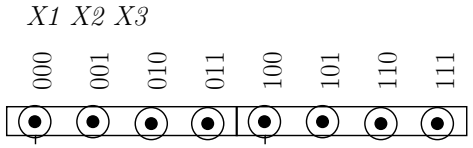
n Mischer pro Literalfilter!!

n Mischer pro Literalfilter!!

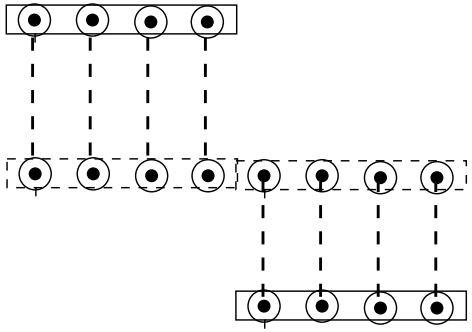
Ein Mischer erzeugt Bitverschiebung der Wege um 1! Alle bleiben gleich lang!!

n Mischer pro Literalfilter!!

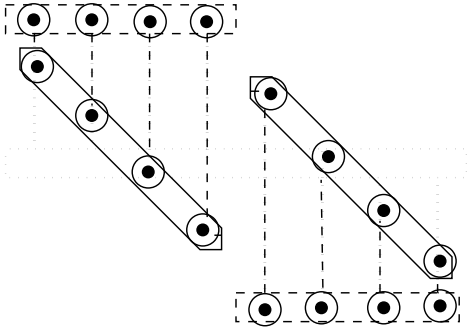
Ein Mischer erzeugt Bitverschiebung der Wege um 1! Alle bleiben gleich lang!!



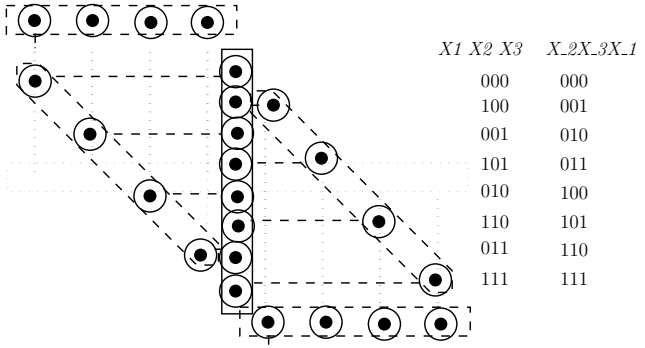
Ebene 1



Ebene 2



Ebene 3



Ebene 4

Kürzeste Wege Alg. für P_α

Kürzeste Wege Alg. für P_α

Ein Weg, der nicht verlängert wird entspricht genau einer Belegung, die die Formel erfüllt!!!

Kürzeste Wege Alg. für P_α

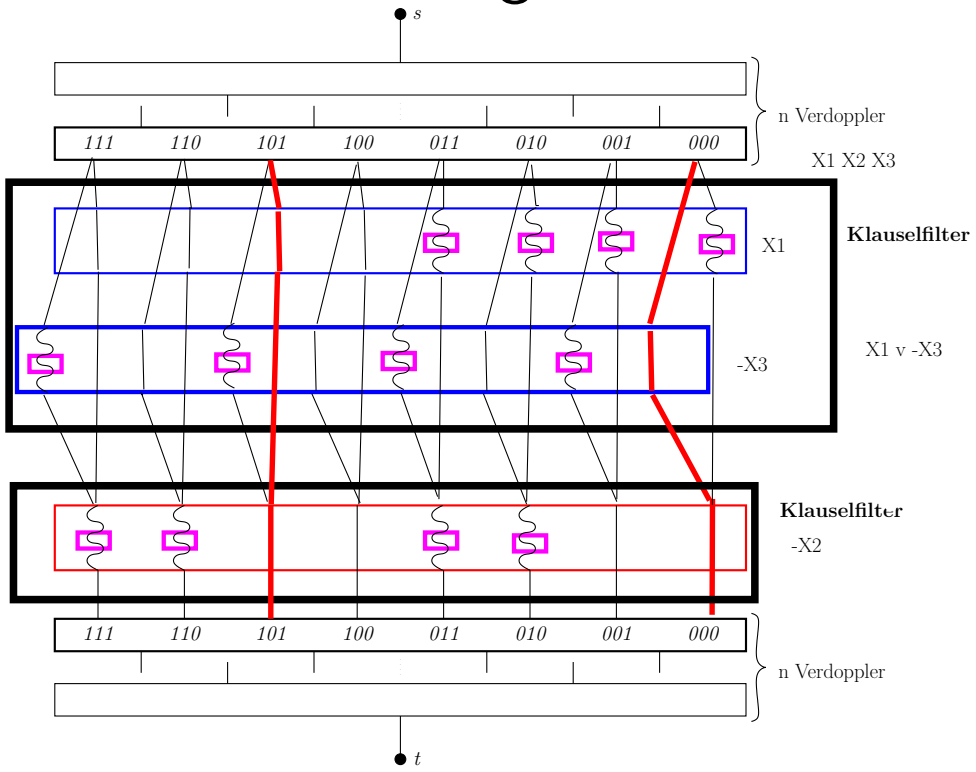
Ein Weg, der nicht verlängert wird entspricht genau einer Belegung, die die Formel erfüllt!!!

Das kann man der Kantenreihenfolge entnehmen!!

Kürzeste Wege Alg. für P_α

Ein Weg, der nicht verlängert wird entspricht genau einer Belegung, die die Formel erfüllt!!!

Das kann man der Kantenreihenfolge entnehmen!!



Konstruktion insgesamt!!

Konstruktion insgesamt!!

- $2n$ Verdoppler: $O(n)$ Kanten

Konstruktion insgesamt!!

- $2n$ Verdoppler: $O(n)$ Kanten
- m Klauselfilter: je Klauselfilter

  – 3 Literalfilter

Konstruktion insgesamt!!

- $2n$ Verdoppler: $O(n)$ Kanten
- m Klauselfilter: je Klauselfilter
  - 3 Literalfilter
  - n Mischer je Filter

Konstruktion insgesamt!!

- $2n$ Verdoppler: $O(n)$ Kanten
- m Klauselfilter: je Klauselfilter
  - 3 Literalfilter
  - n Mischer je Filter
- Insgesamt $O(mn)$ Kanten

Konstruktion insgesamt!!

- $2n$ Verdoppler: $O(n)$ Kanten
- m Klauselfilter: je Klauselfilter
  - 3 Literalfilter
  - n Mischer je Filter
- Insgesamt $O(mn)$ Kanten
- In polynomieller Zeit konstruierbar

Ergebnis!!!

Ergebnis!!!

Theorem 1.38 (Canny/Reif): Bestimmung der optimalen Kantenfolge bei der Berechnung Kürzester Wege in polyedrischer Szene in 3D ist NP hart.

Ergebnis!!!

Theorem 1.38 (Canny/Reif): Bestimmung der optimalen Kantenfolge bei der Berechnung Kürzester Wege in polyedrischer Szene in 3D ist NP hart.

Beweis!!

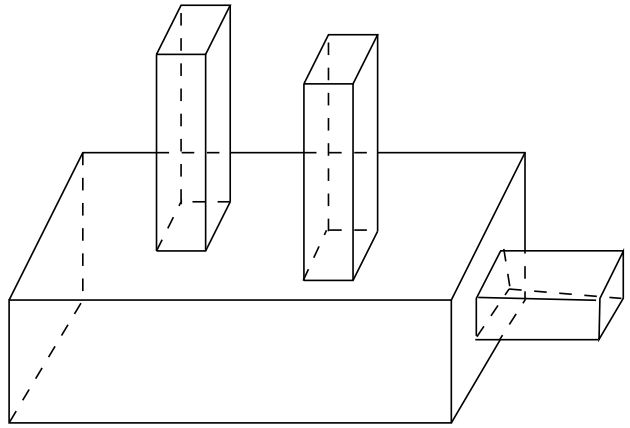
Downgrading: Oberfläche Polyeder!

Downgrading: Oberfläche Polyeder!

- Natürliche Erweiterung der Polygone auf 3D

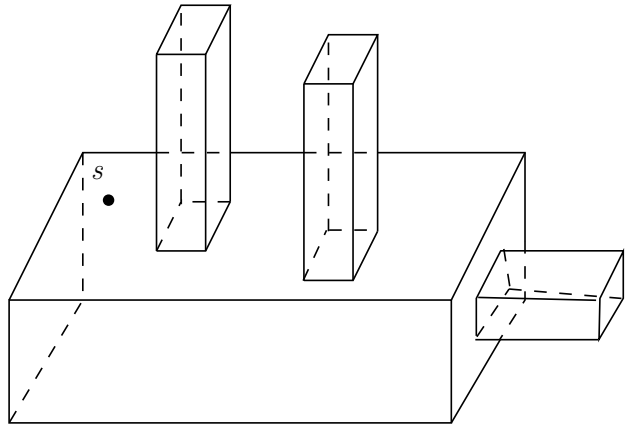
Downgrading: Oberfläche Polyeder!

- Natürliche Erweiterung der Polygone auf 3D



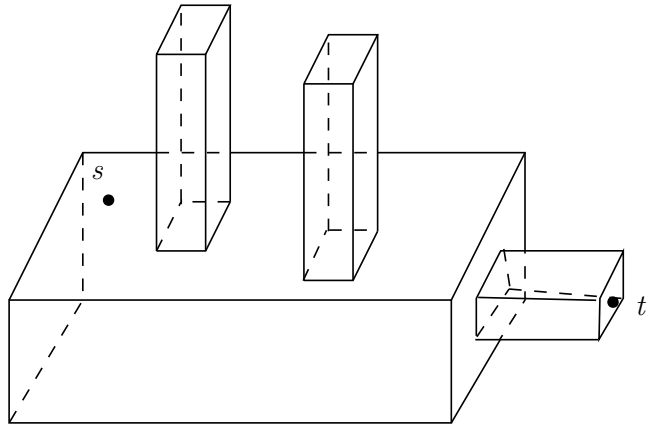
Downgrading: Oberfläche Polyeder!

- Natürliche Erweiterung der Polygone auf 3D



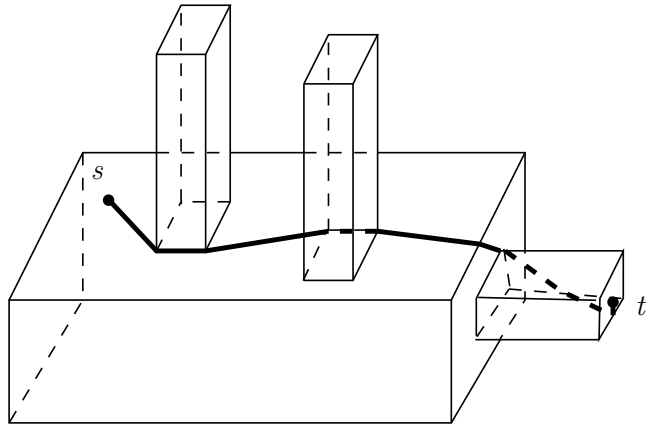
Downgrading: Oberfläche Polyeder!

- Natürliche Erweiterung der Polygone auf 3D



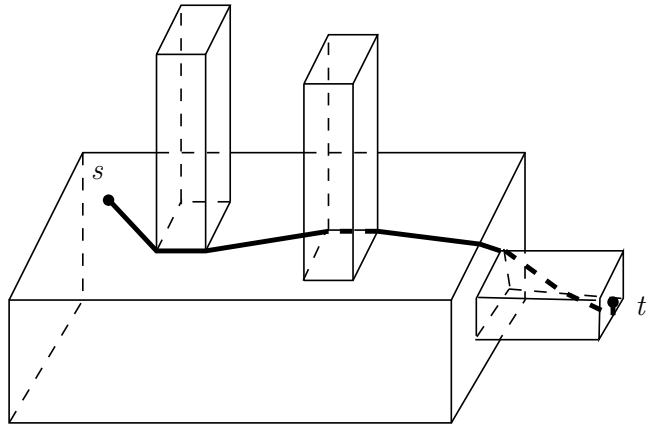
Downgrading: Oberfläche Polyeder!

- Natürliche Erweiterung der Polygone auf 3D



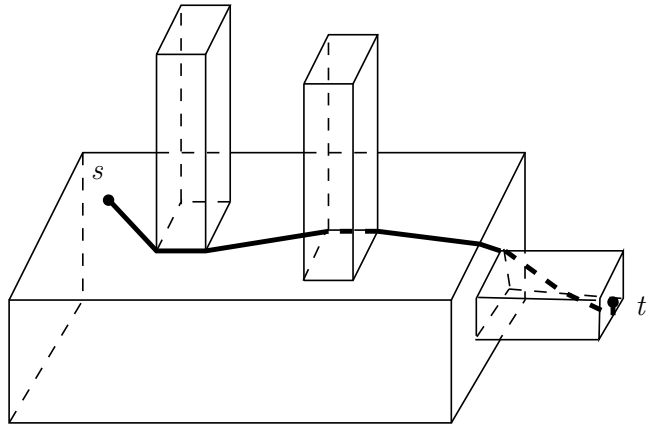
Downgrading: Oberfläche Polyeder!

- Natürliche Erweiterung der Polygone auf 3D
- Rand besteht aus Polygonen



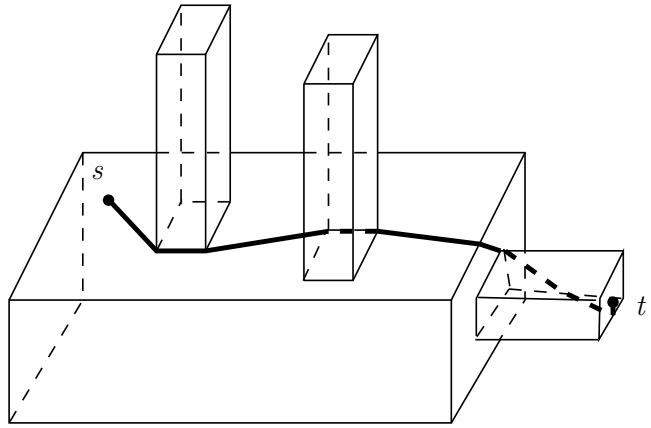
Downgrading: Oberfläche Polyeder!

- Natürliche Erweiterung der Polygone auf 3D
- Rand besteht aus Polygonen
- Keine dünnen Stellen:



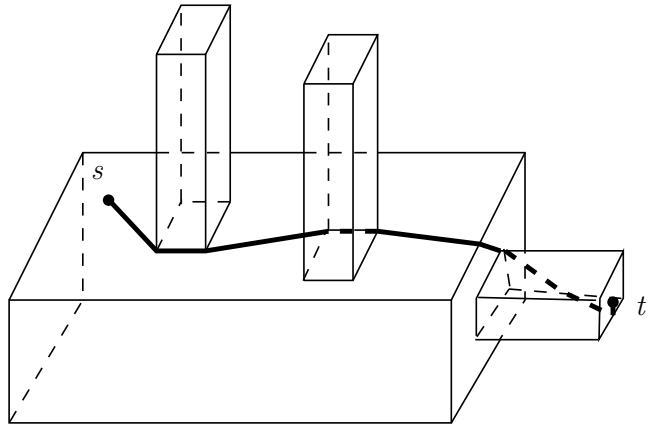
Downgrading: Oberfläche Polyeder!

- Natürliche Erweiterung der Polygone auf 3D
- Rand besteht aus Polygonen
- Keine dünnen Stellen: ϵ-Kugeln



Downgrading: Oberfläche Polyeder!

- Natürliche Erweiterung der Polygone auf 3D
- Rand besteht aus Polygonen
- Keine dünnen Stellen: ϵ-Kugeln
- Datenstruktur QEDS: Triangulation Oberflächen, Navigation!



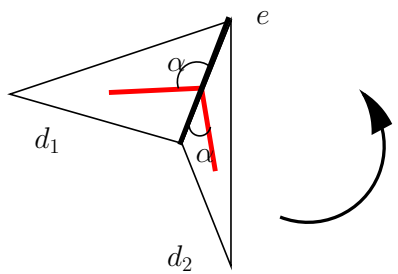
Lokale Eigenschaften (geodätisch)! Lem. 1.39

Lokale Eigenschaften (geodätisch)! Lem. 1.39

- Lokale Eigenschaften (geodätisch) kürzester Wege

Lokale Eigenschaften (geodätisch)! Lem. 1.39

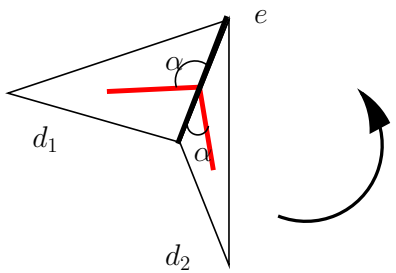
- Lokale Eigenschaften (geodätisch) kürzester Wege
- Alle Kanten: Eingangswinkel=Ausgangswinkel



(i)

Lokale Eigenschaften (geodätisch)! Lem. 1.39

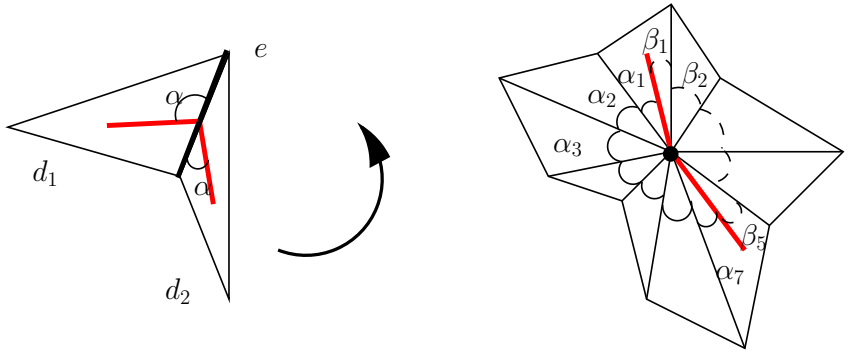
- Lokale Eigenschaften (geodätisch) kürzester Wege
- Alle Kanten: Eingangswinkel=Ausgangswinkel
- Knoten: Nur *nicht-konvexe* Ecken



(i)

Lokale Eigenschaften (geodätisch)! Lem. 1.39

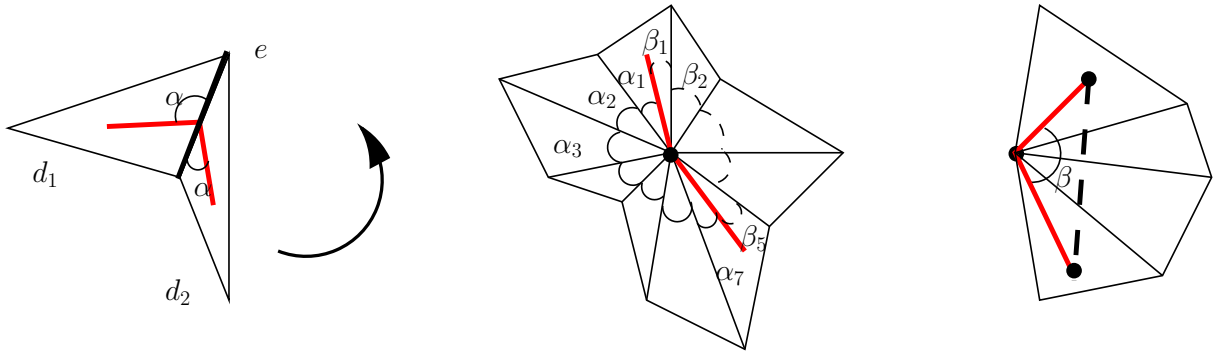
- Lokale Eigenschaften (geodätisch) kürzester Wege
- Alle Kanten: Eingangswinkel=Ausgangswinkel
- Knoten: Nur *nicht-konvexe* Ecken



(i) (ii)

Lokale Eigenschaften (geodätisch)! Lem. 1.39

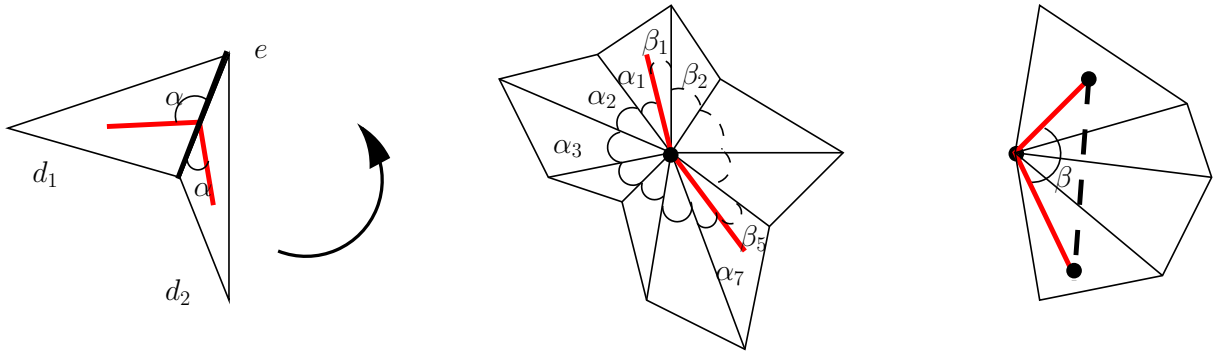
- Lokale Eigenschaften (geodätisch) kürzester Wege
- Alle Kanten: Eingangswinkel=Ausgangswinkel
- Knoten: Nur *nicht-konvexe* Ecken



(i) (ii)

Lokale Eigenschaften (geodätisch)! Lem. 1.39

- Lokale Eigenschaften (geodätisch) kürzester Wege
- Alle Kanten: Eingangswinkel=Ausgangswinkel
- Knoten: Nur *nicht-konvexe* Ecken
- Konvexe Ecke: Ebenenschnitt



(i) (ii)

Globale Eigenschaften (Kürzeste)! Lem. 1.40

Globale Eigenschaften (Kürzeste)! Lem. 1.40

π_i, π_j kürzeste von s nach b_i resp. b_j

Globale Eigenschaften (Kürzeste)! Lem. 1.40

π_i, π_j kürzeste von s nach b_i resp. b_j

(i) π_i hat keine Selbstschnitte

Globale Eigenschaften (Kürzeste)! Lem. 1.40

π_i, π_j kürzeste von s nach b_i resp. b_j

(i) π_i hat keine Selbstschnitte

(ii) π_i schneidet jede Fläche max. einmal

Globale Eigenschaften (Kürzeste)! Lem. 1.40

π_i, π_j kürzeste von s nach b_i resp. b_j

(i) π_i hat keine Selbstschnitte

(ii) π_i schneidet jede Fläche max. einmal

(iii) π_i, π_j kreuzen sich nicht im Innern einer Fläche

Globale Eigenschaften (Kürzeste)! Lem. 1.40

π_i, π_j kürzeste von s nach b_i resp. b_j

(i) π_i hat keine Selbstschnitte

(ii) π_i schneidet jede Fläche max. einmal

(iii) π_i, π_j kreuzen sich nicht im Innern einer Fläche

Beweis!!! Lokal verkürzbar! (Tafel)

Berechnungsansatz: Ideen

Berechnungsansatz: Ideen

- Locus approach: Startpunkt s fest

Berechnungsansatz: Ideen

- Locus approach: Startpunkt s fest
- Nur letzte Schritte des Weges konkret

Berechnungsansatz: Ideen

- Locus approach: Startpunkt s fest
- Nur letzte Schritte des Weges konkret
- Kombinatorisch gleiche zusammenfassen

Berechnungsansatz: Ideen

- Locus approach: Startpunkt s fest
- Nur letzte Schritte des Weges konkret
- Kombinatorisch gleiche zusammenfassen
- Zunächst nur Intervalle auf Kanten betrachten

Berechnungsansatz: Ideen

- Locus approach: Startpunkt s fest
- Nur letzte Schritte des Weges konkret
- Kombinatorisch gleiche zusammenfassen
- Zunächst nur Intervalle auf Kanten betrachten
- Sukzessive erweitern: Continous Dijkstra

Berechnungsansatz: Ideen

- Locus approach: Startpunkt s fest
- Nur letzte Schritte des Weges konkret
- Kombinatorisch gleiche zusammenfassen
- Zunächst nur Intervalle auf Kanten betrachten
- Sukzessive erweitern: Continous Dijkstra
- In die Dreiecke fortpflanzen

Berechnungsansatz: Ideen

- Locus approach: Startpunkt s fest
- Nur letzte Schritte des Weges konkret
- Kombinatorisch gleiche zusammenfassen
- Zunächst nur Intervalle auf Kanten betrachten
- Sukzessive erweitern: Continous Dijkstra
- In die Dreiecke fortpflanzen
- Query Struktur für alle Punkte auf P

Kombinatorik: Gesamtweg analysieren!

Kombinatorik: Gesamtweg analysieren!

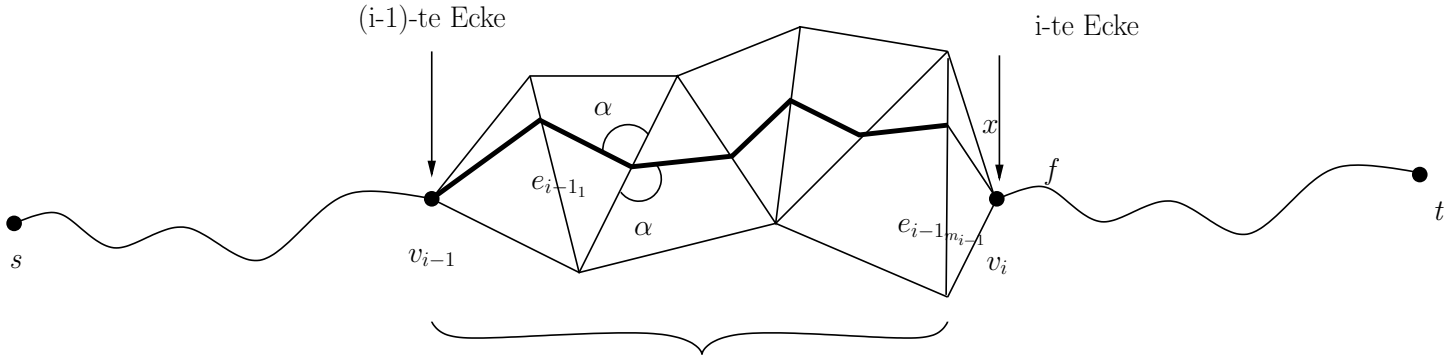
Folge von Kanten und Knoten!

Kombinatorik: Gesamtweg analysieren!

Folge von Kanten und Knoten!

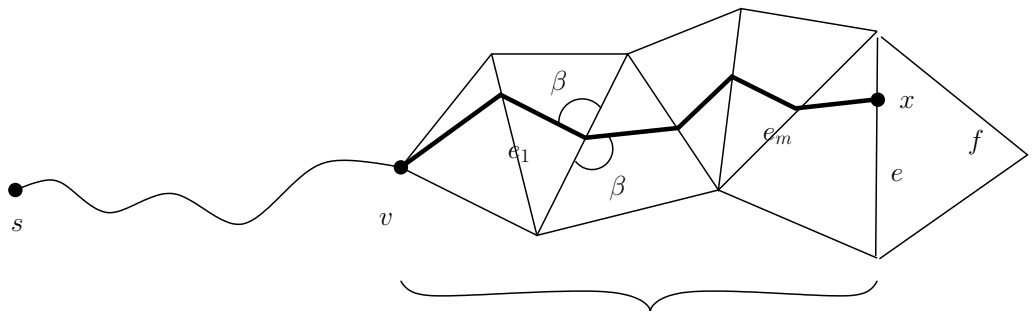
$$\pi = \underbrace{v_0}_{=s}, \underbrace{e_{1,1}, \ldots, e_{1,m_1}}_{\text{Kanten}}, \underbrace{v_1}_{\text{2. Ecke}},$$

$$\underbrace{e_{2,1}, \ldots, e_{2,m_2}}_{\text{Kanten}}, v_2, \ldots, e_{k-1,1}, \ldots, e_{k-1,m_{k-1}}, \underbrace{v_k}_{=t}$$



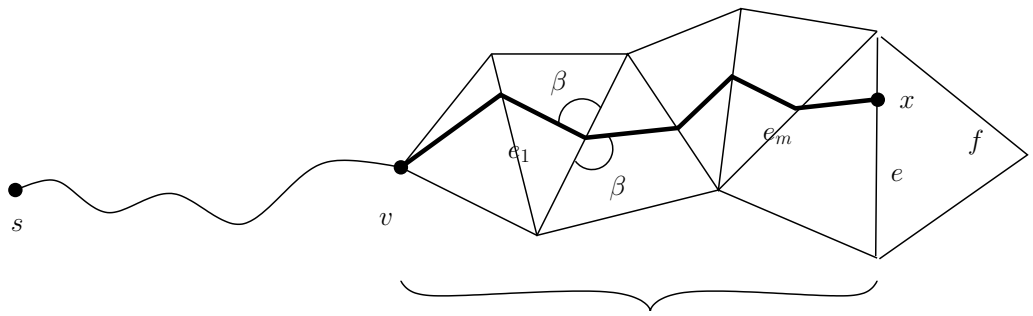
Letzte Schritte: Für beliebige Kanten!

- Kante e, letzter Knoten v, Kantenfolge E



Letzte Schritte: Für beliebige Kanten!

- Kante e, letzter Knoten v, Kantenfolge E
- Kombinatorisch gleiche zuammenfassen,

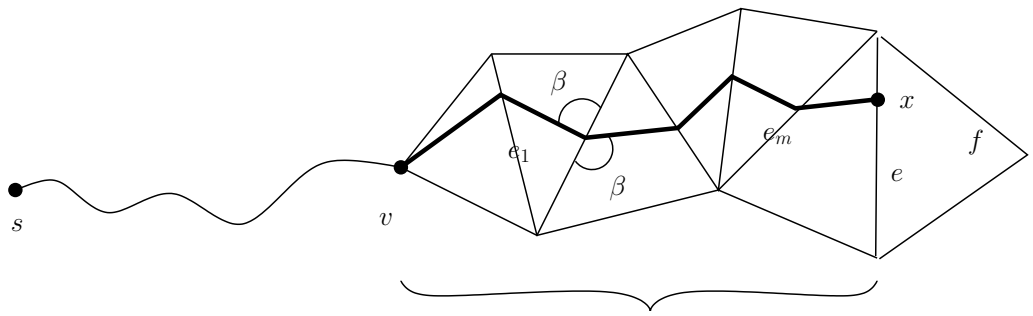


Kantenfolge E

(Weg verlaeuft nicht ueber Ecken von P)

Letzte Schritte: Für beliebige Kanten!

- Kante e, letzter Knoten v, Kantenfolge E
- Kombinatorisch gleiche zuammenfassen, Dreiecke nicht beachten

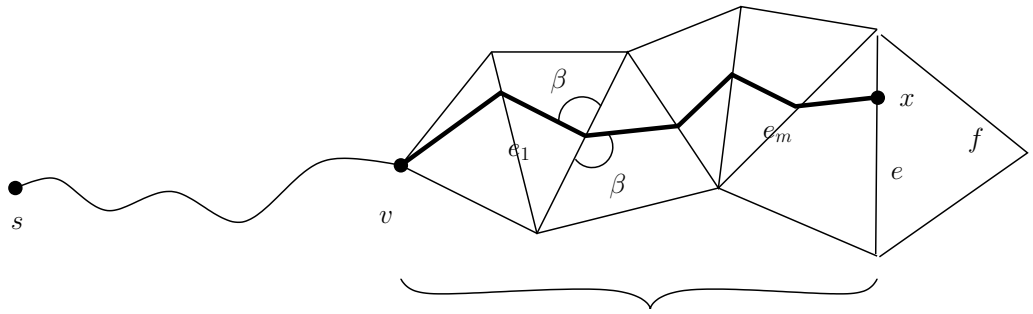


Kantenfolge E

(Weg verlaeuft nicht ueber Ecken von P)

Letzte Schritte: Für beliebige Kanten!

- Kante e, letzter Knoten v, Kantenfolge E
- Kombinatorisch gleiche zuammenfassen, Dreiecke nicht beachten
- Optimalitätsintervall: Def. 1.42



Kantenfolge E

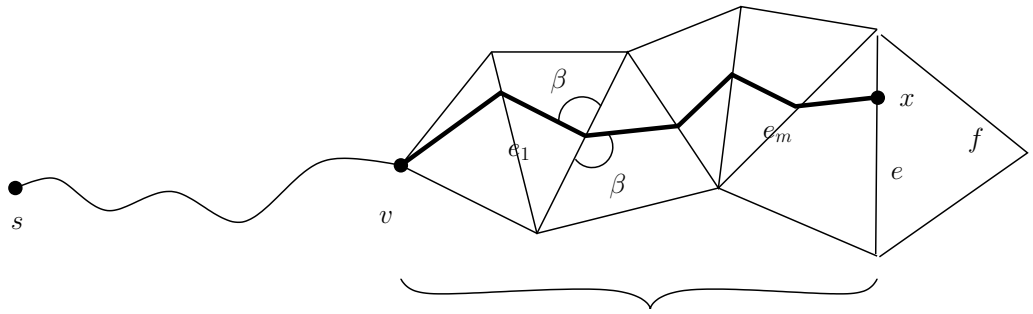
(Weg verlaeuft nicht ueber Ecken von P)

Letzte Schritte: Für beliebige Kanten!

- Kante e, letzter Knoten v, Kantenfolge E
- Kombinatorisch gleiche zuammenfassen, Dreiecke nicht beachten
- Optimalitätsintervall: Def. 1.42

$I(v, \mathsf{E}) := \{x \in e |$ $\exists$ Kürzeste δ von s nach x mit

- $\delta \cap \mathsf{INT}(f) = \emptyset$
- δ endet mit $v, e_1, \ldots, e_m, x\}$.



Kantenfolge E

(Weg verlaeuft nicht ueber Ecken von P)

Lem. 1.43 Eigenschaften: $I(v, \epsilon)$

(i) Jede solche Menge $I(v, \mathrm{E})$ ist Intervall auf e (evtl. leer).
(ii) Zwei verschiedene Intervalle können sich nicht überlappen.
(iii) e wird von Intervallen $I(v, \mathrm{E})$ ganz überdeckt.

Beweis!!! Gegeben: Kante e, letzter Knoten v, Kantenfolge E

$I(v, \mathbf{E})$ ist Intervall auf e

$I(v, \mathsf{E})$ ist Intervall auf e

- $I(v, \mathsf{E})$ leer $\Rightarrow$ fertig!

$I(v, \mathsf{E})$ ist Intervall auf e

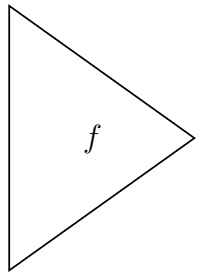
- $I(v, \mathsf{E})$ leer $\Rightarrow$ fertig!
- Annahme: Es gibt zwei Punkte x_1 und x_2 in $I(v, \mathsf{E})$

$I(v, \mathsf{E})$ ist Intervall auf e

- $I(v, \mathsf{E})$ leer $\Rightarrow$ fertig!
- Annahme: Es gibt zwei Punkte x_1 und x_2 in $I(v, \mathsf{E})$
- Zu zeigen: Alle Punkte dazwischen gehören zu $I(v, \mathsf{E})$

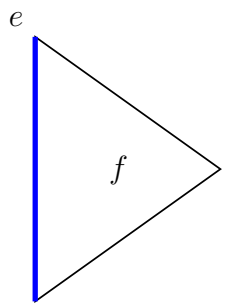
$I(v, \mathsf{E})$ ist Intervall auf e

- $I(v, \mathsf{E})$ leer $\Rightarrow$ fertig!
- Annahme: Es gibt zwei Punkte x_1 und x_2 in $I(v, \mathsf{E})$
- Zu zeigen: Alle Punkte dazwischen gehören zu $I(v, \mathsf{E})$



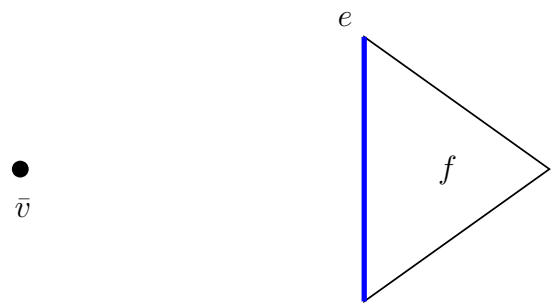
$I(v, \mathsf{E})$ ist Intervall auf e

- $I(v, \mathsf{E})$ leer $\Rightarrow$ fertig!
- Annahme: Es gibt zwei Punkte x_1 und x_2 in $I(v, \mathsf{E})$
- Zu zeigen: Alle Punkte dazwischen gehören zu $I(v, \mathsf{E})$



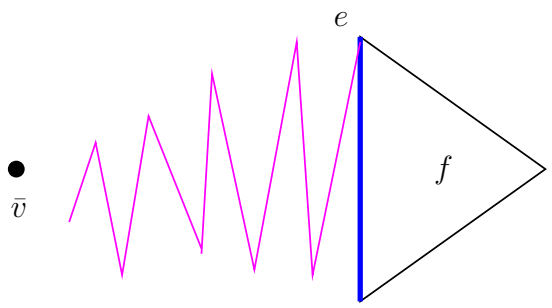
$I(v, \mathsf{E})$ ist Intervall auf e

- $I(v, \mathsf{E})$ leer $\Rightarrow$ fertig!
- Annahme: Es gibt zwei Punkte x_1 und x_2 in $I(v, \mathsf{E})$
- Zu zeigen: Alle Punkte dazwischen gehören zu $I(v, \mathsf{E})$



$I(v, \mathsf{E})$ ist Intervall auf e

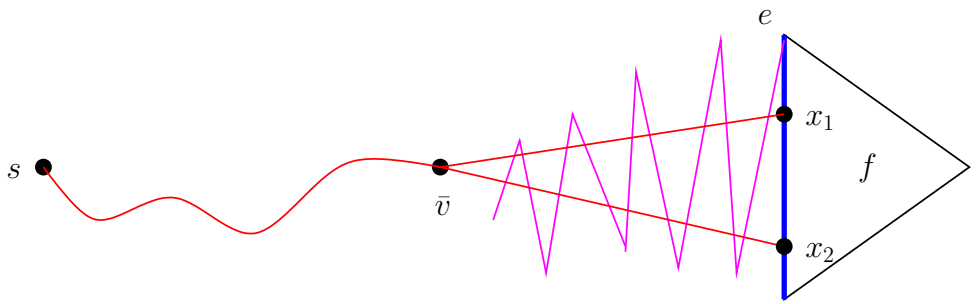
- $I(v, \mathsf{E})$ leer $\Rightarrow$ fertig!
- Annahme: Es gibt zwei Punkte x_1 und x_2 in $I(v, \mathsf{E})$
- Zu zeigen: Alle Punkte dazwischen gehören zu $I(v, \mathsf{E})$



E (aufklappen in Ebene zu f)

$I(v, \mathsf{E})$ ist Intervall auf e

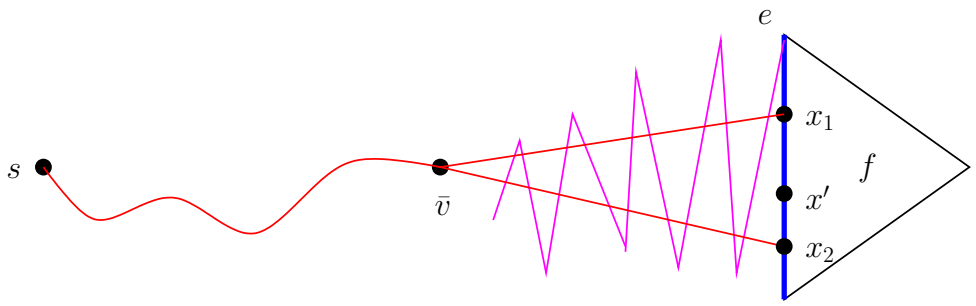
- $I(v, \mathsf{E})$ leer $\Rightarrow$ fertig!
- Annahme: Es gibt zwei Punkte x_1 und x_2 in $I(v, \mathsf{E})$
- Zu zeigen: Alle Punkte dazwischen gehören zu $I(v, \mathsf{E})$



E (aufklappen in Ebene zu f)

$I(v, \mathsf{E})$ ist Intervall auf e

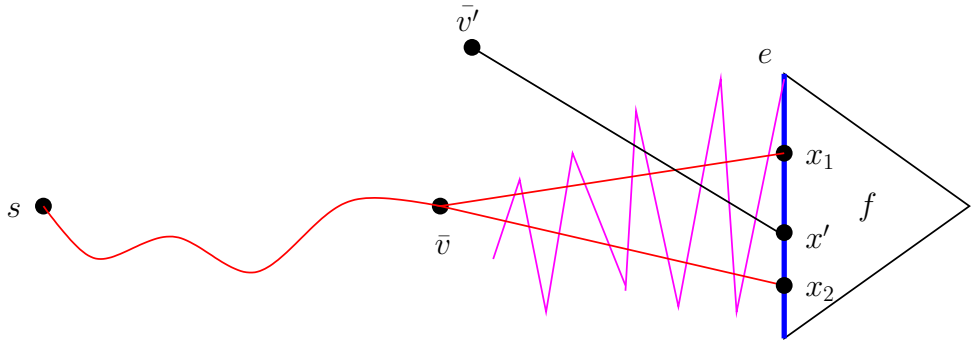
- $I(v, \mathsf{E})$ leer $\Rightarrow$ fertig!
- Annahme: Es gibt zwei Punkte x_1 und x_2 in $I(v, \mathsf{E})$
- Zu zeigen: Alle Punkte dazwischen gehören zu $I(v, \mathsf{E})$



E (aufklappen in Ebene zu f)

$I(v, \mathsf{E})$ ist Intervall auf e

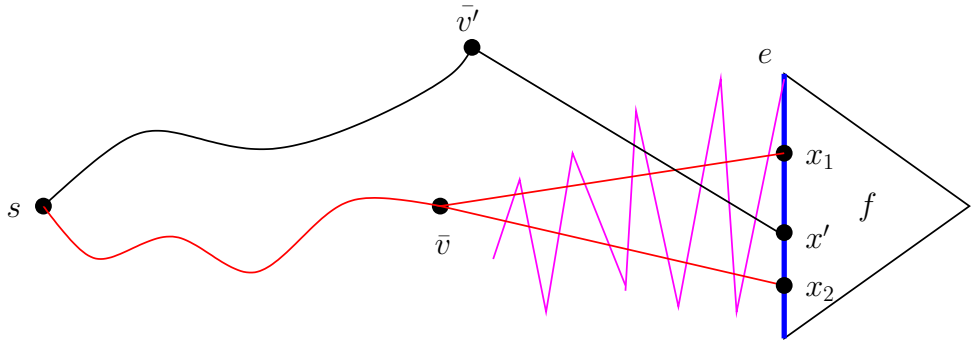
- $I(v, \mathsf{E})$ leer $\Rightarrow$ fertig!
- Annahme: Es gibt zwei Punkte x_1 und x_2 in $I(v, \mathsf{E})$
- Zu zeigen: Alle Punkte dazwischen gehören zu $I(v, \mathsf{E})$



E (aufklappen in Ebene zu f)

$I(v, \mathsf{E})$ ist Intervall auf e

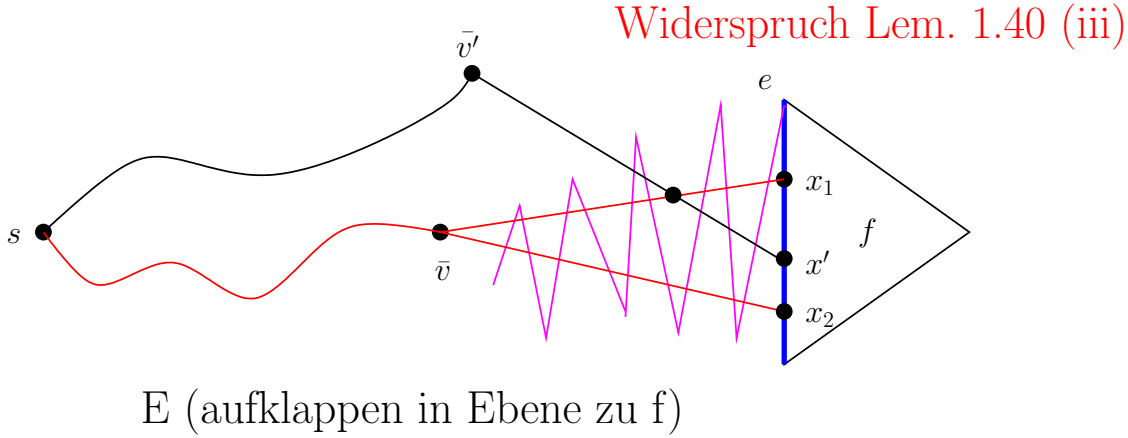
- $I(v, \mathsf{E})$ leer $\Rightarrow$ fertig!
- Annahme: Es gibt zwei Punkte x_1 und x_2 in $I(v, \mathsf{E})$
- Zu zeigen: Alle Punkte dazwischen gehören zu $I(v, \mathsf{E})$



E (aufklappen in Ebene zu f)

$I(v, \mathsf{E})$ ist Intervall auf e

- $I(v, \mathsf{E})$ leer $\Rightarrow$ fertig!
- Annahme: Es gibt zwei Punkte x_1 und x_2 in $I(v, \mathsf{E})$
- Zu zeigen: Alle Punkte dazwischen gehören zu $I(v, \mathsf{E})$



Intervalle $I(v, \mathbf{E})$ überlappen sich nicht

Intervalle $I(v, \mathbf{E})$ überlappen sich nicht

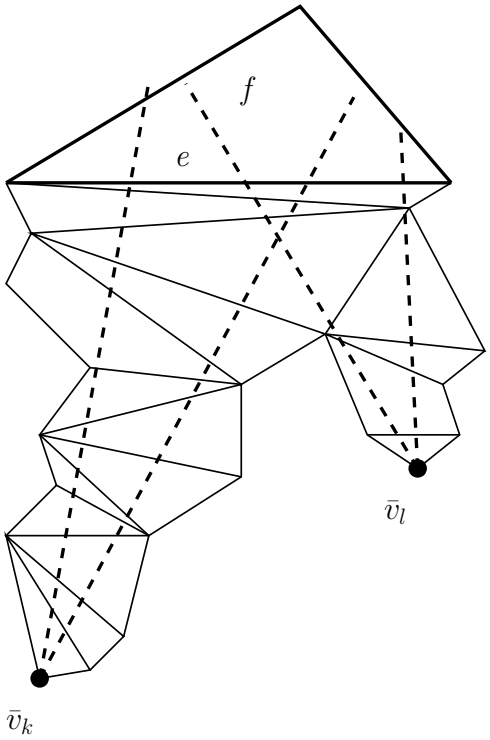
- Gleiche Argumentation geht auch!!

Intervalle $I(v, \mathbf{E})$ überlappen sich nicht

- Gleiche Argumentation geht auch!!
- Schönere Argumentation!!

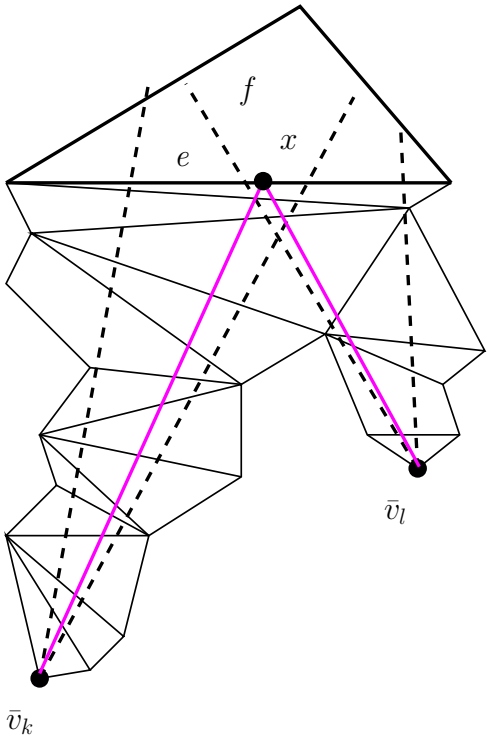
Intervalle $I(v, \mathbf{E})$ überlappen sich nicht

- Gleiche Argumentation geht auch!!
- Schönere Argumentation!!



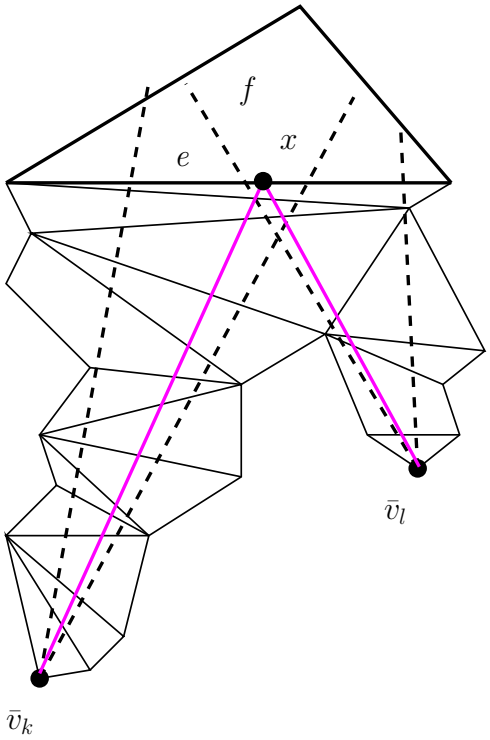
Intervalle $I(v, \mathbf{E})$ überlappen sich nicht

- Gleiche Argumentation geht auch!!
- Schönere Argumentation!!



Intervalle $I(v, \mathbf{E})$ überlappen sich nicht

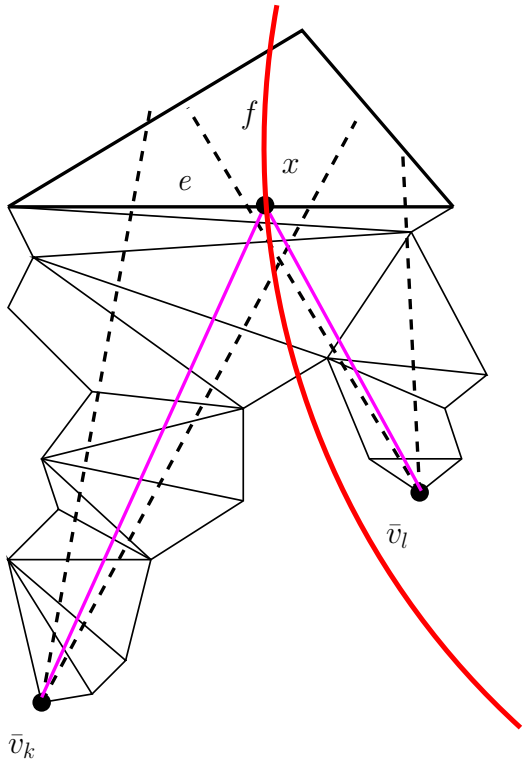
- Gleiche Argumentation geht auch!!
- Schönere Argumentation!!



$$|\bar{v}_k - x| + d(v_k, s) = |\bar{v}_l - x| + d(v_l, s)$$

Intervalle $I(v, \mathbf{E})$ überlappen sich nicht

- Gleiche Argumentation geht auch!!
- Schönere Argumentation!!

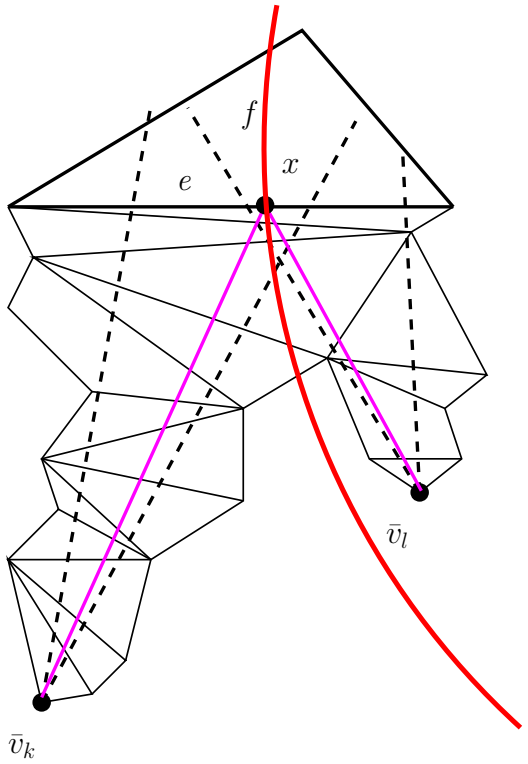


$$|\bar{v}_k - x| + d(v_k, s) = |\bar{v}_l - x| + d(v_l, s)$$

Hyperbel

Intervalle $I(v, \mathbf{E})$ überlappen sich nicht

- Gleiche Argumentation geht auch!!
- Schönere Argumentation!!



$$|\bar{v}_k - x| + d(v_k, s) = |\bar{v}_l - x| + d(v_l, s)$$

Hyperbel

Genau ein x!!!

e wird von Intervallen $I(v, \mathbf{E})$ überdeckt

e wird von Intervallen $I(v, \mathbf{E})$ überdeckt

Klar!

e wird von Intervallen $I(v, \mathbf{E})$ überdeckt

Klar! Jeder Punkt $x \in e$ wird von einem Kürzesten Weg besucht!

e wird von Intervallen $I(v, \mathbf{E})$ überdeckt

Klar! Jeder Punkt $x \in e$ wird von einem Kürzesten Weg besucht!

Insgesamt:

e wird von Intervallen $I(v, \mathbf{E})$ überdeckt

Klar! Jeder Punkt $x \in e$ wird von einem Kürzesten Weg besucht!

Insgesamt: Lem. 1.43:

e wird von Intervallen $I(v, \mathsf{E})$ überdeckt

Klar! Jeder Punkt $x \in e$ wird von einem Kürzesten Weg besucht!

Insgesamt: Lem. 1.43:

(i) Jede solche Menge $I(v, \mathsf{E})$ ist Intervall auf e (evtl. leer).
(ii) Zwei verschiedene Intervalle können sich nicht überlappen.
(iii) e wird von Intervallen $I(v, \mathsf{E})$ ganz überdeckt.

Alle $I(v, \mathbf{E})$ berechnen!

Alle $I(v, \mathbf{E})$ berechnen!

- Wie viele?

Alle $I(v, \mathbf{E})$ berechnen!

- Wie viele?
- Was machen wir mit dem Inneren der Dreiecke?
- Lem. 1.44: Kante e, $O(n)$ Intervalle $I(v, \epsilon)$

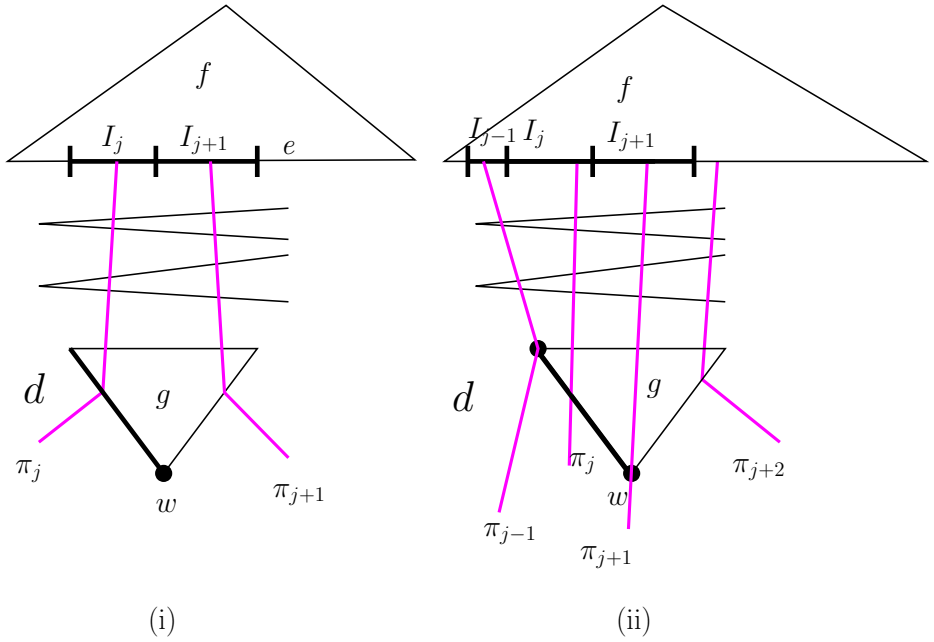
Alle $I(v, \mathbf{E})$ berechnen!

- Wie viele?
- Was machen wir mit dem Inneren der Dreiecke?
- Lem. 1.44: Kante e, $O(n)$ Intervalle $I(v, \epsilon)$
- Zählargument: Klassisch!!!

Lem. 1.44: $O(n)$ Intervalle $I(v, \mathbf{E})$

Lem. 1.44: $O(n)$ Intervalle $I(v, \mathbf{E})$

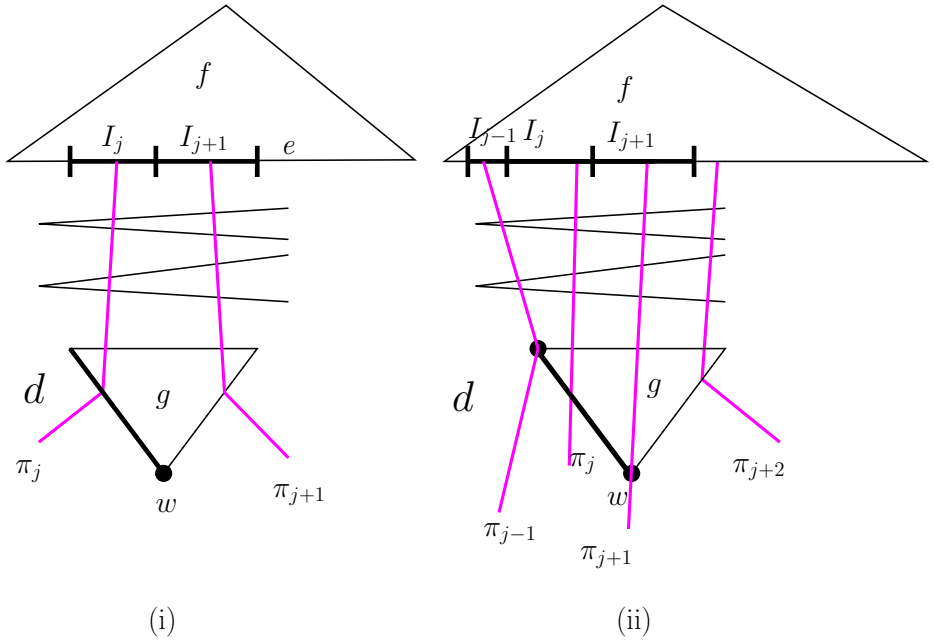
Benachbarte Intervalle trennen sich!!



- Kante d kann max zweimal als Trenner vorkommen!!

Lem. 1.44: $O(n)$ Intervalle $I(v, \mathbf{E})$

Benachbarte Intervalle trennen sich!!



- Kante d kann max zweimal als Trenner vorkommen!!
- Wegen Schnitteigenschaft!! Ausklappen ohne Überlappungen!

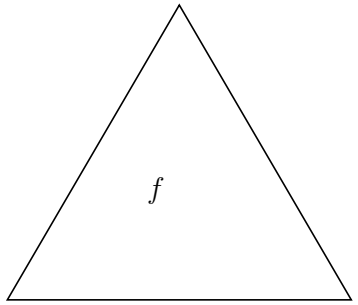
Das Innere der Dreiecke füllen

Das Innere der Dreiecke füllen

Annahme: Intervalle $I(v, \mathsf{E})$ berechnet!

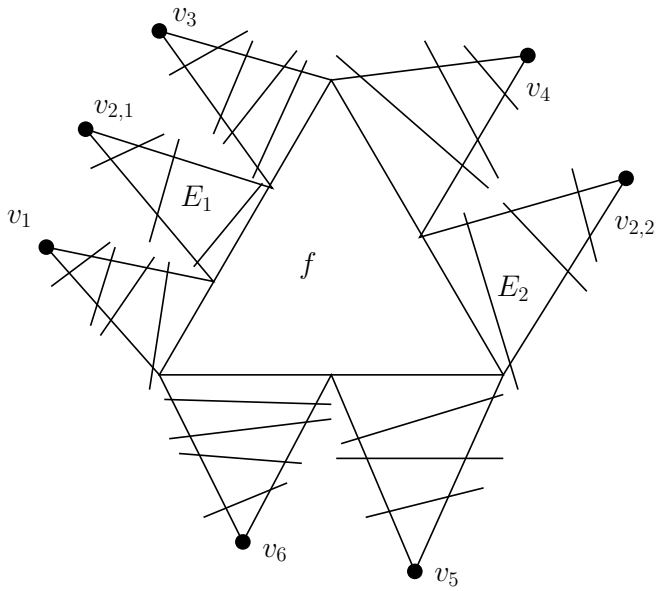
Das Innere der Dreiecke füllen

Annahme: Intervalle $I(v, \mathsf{E})$ berechnet! Situation für jedes Dreieck!



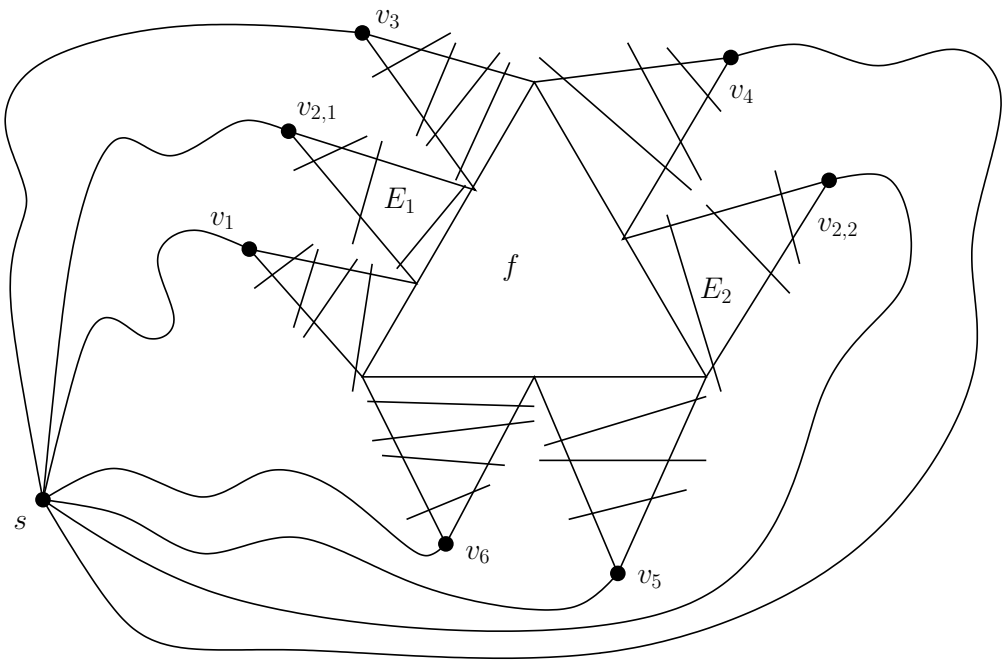
Das Innere der Dreiecke füllen

Annahme: Intervalle $I(v, \mathrm{E})$ berechnet! Situation für jedes Dreieck! Mehrere Kantenfolgen!!



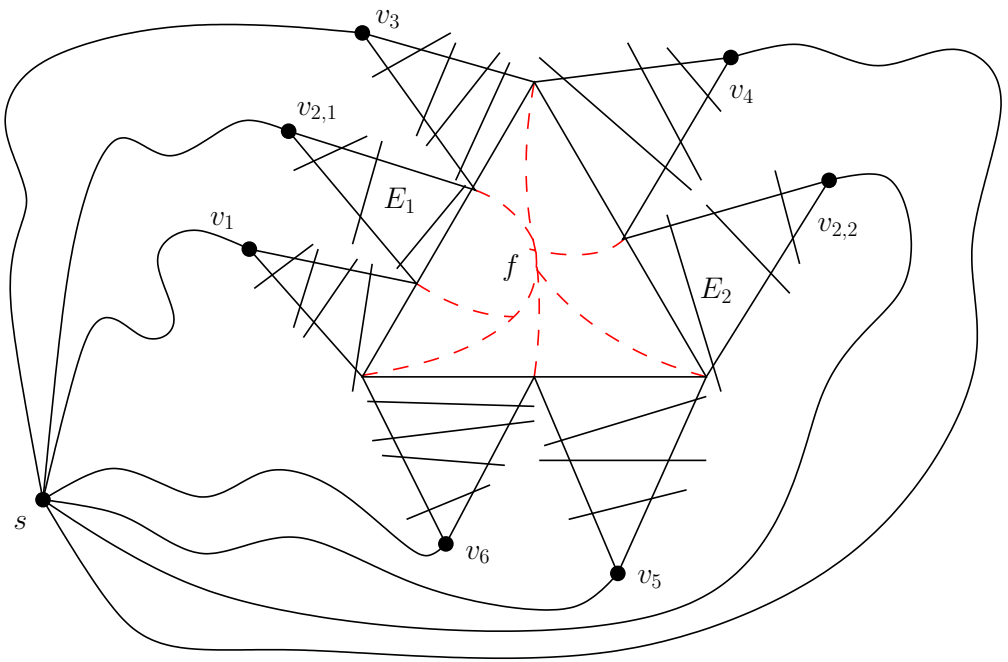
Das Innere der Dreiecke füllen

Annahme: Intervalle $I(v, \mathrm{E})$ berechnet! Situation für jedes Dreieck! Mehrere Kantenfolgen!!



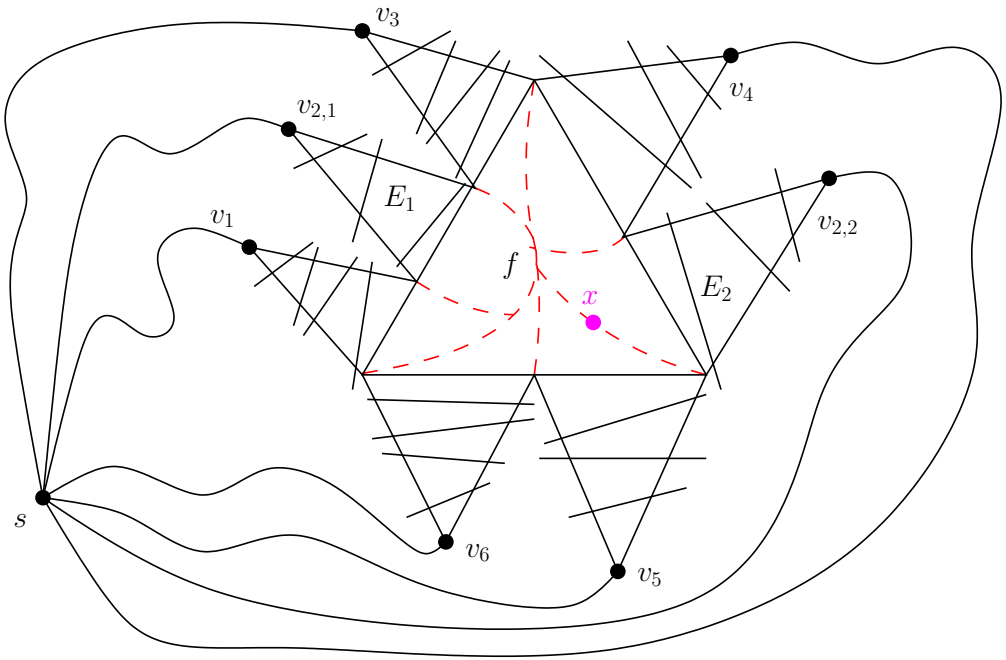
Das Innere der Dreiecke füllen

Annahme: Intervalle $I(v, \mathrm{E})$ berechnet! Situation für jedes Dreieck! Mehrere Kantenfolgen!!



Das Innere der Dreiecke füllen

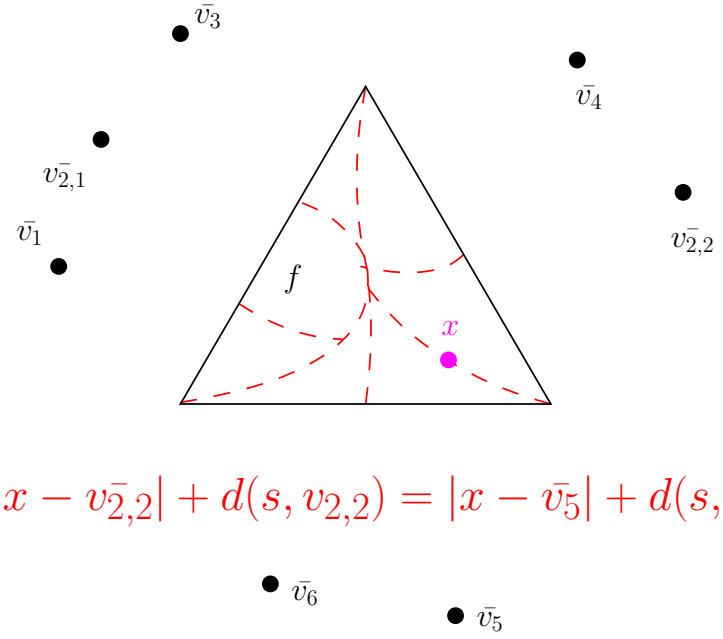
Annahme: Intervalle $I(v, \mathrm{E})$ berechnet! Situation für jedes Dreieck! Mehrere Kantenfolgen!!



$$|x - \bar{v_{2,2}}| + d(s, v_{2,2}) = |x - \bar{v_5}| + d(s, v_5)$$

Aufteilen der Dreiecke!

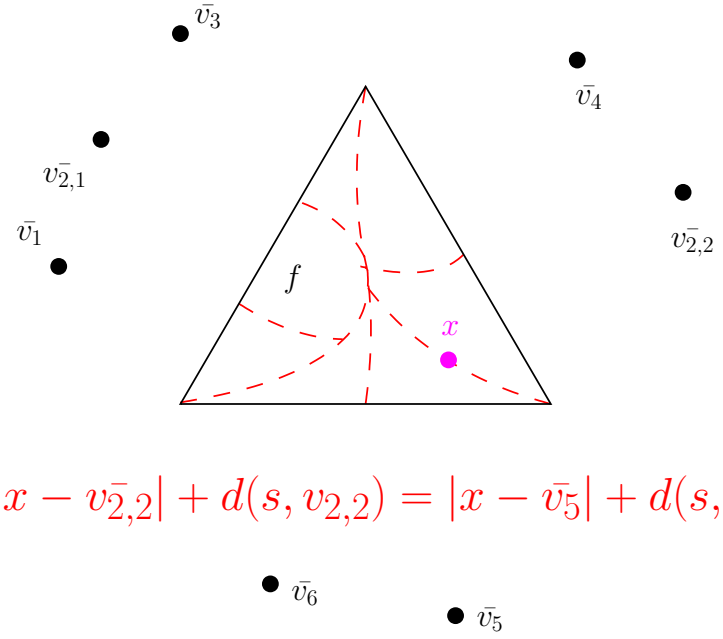
Aufteilen der Dreiecke!



$$|x - v_{\bar{2},2}| + d(s, v_{2,2}) = |x - \bar{v_5}| + d(s, v_5)$$

- Gewichte: $d(s, v_i)$,

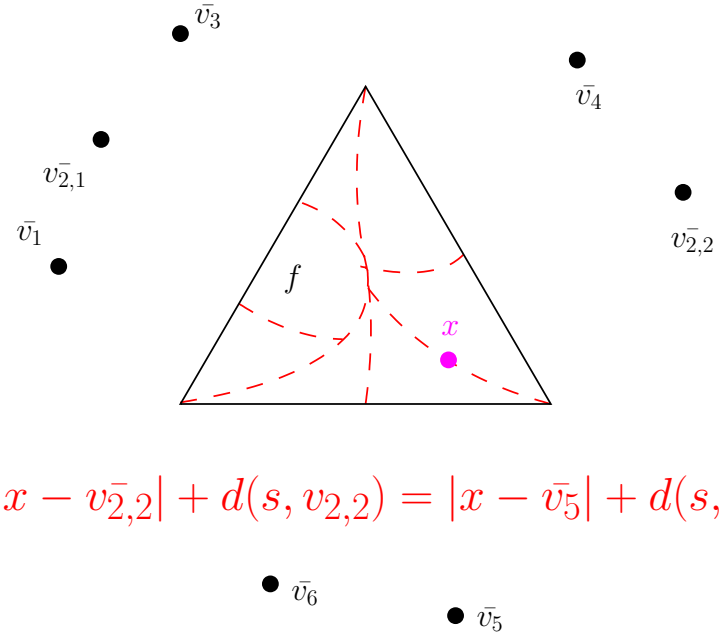
Aufteilen der Dreiecke!



$$|x - v_{\bar{2},2}| + d(s, v_{2,2}) = |x - \bar{v_5}| + d(s, v_5)$$

- Gewichte: $d(s, v_i)$, Regionen bezüglich Orte $\bar{v_i}$

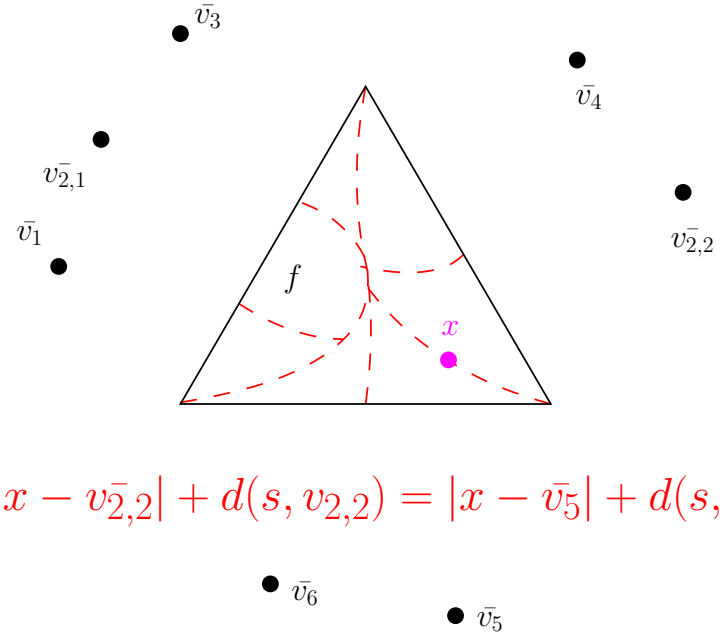
Aufteilen der Dreiecke!



$$|x - \bar{v_{2,2}}| + d(s, v_{2,2}) = |x - \bar{v_5}| + d(s, v_5)$$

- Gewichte: $d(s, v_i)$, Regionen bezüglich Orte $\bar{v_i}$
- Voronoi Diagramm mit additiven Gewichten $d(s, v_i)$
  `voroAdd.html`

Aufteilen der Dreiecke!



$$|x - \bar{v_{2,2}}| + d(s, v_{2,2}) = |x - \bar{v_5}| + d(s, v_5)$$

- Gewichte: $d(s, v_i)$, Regionen bezüglich Orte $\bar{v_i}$
- Voronoi Diagramm mit additiven Gewichten $d(s, v_i)$
  `voroAdd.html`
- Lokalisationsmöglichkeit! (Separators/Seidel)

Alg. 1.13

Alg. 1.13

- Input: Triang. Polyder P, n Ecken, s

Alg. 1.13

- Input: Triang. Polyder P, n Ecken, s
- Output: Kürzester Weg für beliebigen Anfragepunkt $t \in f$

Alg. 1.13

- Input: Triang. Polyder P, n Ecken, s
- Output: Kürzester Weg für beliebigen Anfragepunkt $t \in f$
- Preprocessing:

Alg. 1.13

- Input: Triang. Polyder P, n Ecken, s
- Output: Kürzester Weg für beliebigen Anfragepunkt $t \in f$
- Preprocessing:
  - Berechne Intervalle $I(v, \mathsf{E})$,

Alg. 1.13

- Input: Triang. Polyder P, n Ecken, s
- Output: Kürzester Weg für beliebigen Anfragepunkt $t \in f$
- Preprocessing:
  - Berechne Intervalle $I(v, \mathsf{E})$, $O(n^2 \log n)$
  - Für alle f **add. gew. VD**,

Alg. 1.13

- Input: Triang. Polyder P, n Ecken, s
- Output: Kürzester Weg für beliebigen Anfragepunkt $t \in f$
- Preprocessing:
  - Berechne Intervalle $I(v, \mathsf{E})$, $O(n^2 \log n)$
  - Für alle f **add. gew. VD**, $O(n^2 \log n)$

Alg. 1.13

- Input: Triang. Polyder P, n Ecken, s
- Output: Kürzester Weg für beliebigen Anfragepunkt $t \in f$
- Preprocessing:
  - Berechne Intervalle $I(v, \mathrm{E})$, $O(n^2 \log n)$
  - Für alle f **add. gew. VD**, $O(n^2 \log n)$
  - **Lokalisationstechnik!(Separators/Seidel)**,

Alg. 1.13

- Input: Triang. Polyder P, n Ecken, s
- Output: Kürzester Weg für beliebigen Anfragepunkt $t \in f$
- Preprocessing:
  - Berechne Intervalle $I(v, \mathrm{E})$, $O(n^2 \log n)$
  - Für alle f **add. gew. VD**, $O(n^2 \log n)$
  - **Lokalisationstechnik!(Separators/Seidel)**, $O(n^2)$

Alg. 1.13

- Input: Triang. Polyder P, n Ecken, s
- Output: Kürzester Weg für beliebigen Anfragepunkt $t \in f$
- Preprocessing:
  - Berechne Intervalle $I(v, \mathsf{E})$, $O(n^2 \log n)$
  - Für alle f **add. gew. VD**, $O(n^2 \log n)$
  - **Lokalisationstechnik!(Separators/Seidel)**, $O(n^2)$
- Query:

Alg. 1.13

- Input: Triang. Polyder P, n Ecken, s
- Output: Kürzester Weg für beliebigen Anfragepunkt $t \in f$
- Preprocessing:
  - Berechne Intervalle $I(v, \mathrm{E})$, $O(n^2 \log n)$
  - Für alle f **add. gew. VD**, $O(n^2 \log n)$
  - **Lokalisationstechnik!(Separators/Seidel)**, $O(n^2)$
- Query:
  - **Voronoi Region von t in $VD(f)$**

Alg. 1.13

- Input: Triang. Polyder P, n Ecken, s
- Output: Kürzester Weg für beliebigen Anfragepunkt $t \in f$
- Preprocessing:
  - Berechne Intervalle $I(v, \mathrm{E})$, $O(n^2 \log n)$
  - Für alle f **add. gew. VD**, $O(n^2 \log n)$
  - **Lokalisationstechnik!(Separators/Seidel)**, $O(n^2)$
- Query:
  - **Voronoi Region von** t **in** $VD(f)$ $O(\log n)$
  - Kürzesten Weg aus $I(v, \mathrm{E})$: Über E und in v abgesp. Kürz. Weg zu s, k Segmente,

Alg. 1.13

- Input: Triang. Polyder P, n Ecken, s
- Output: Kürzester Weg für beliebigen Anfragepunkt $t \in f$
- Preprocessing:
  - Berechne Intervalle $I(v, \mathrm{E})$, $O(n^2 \log n)$
  - Für alle f **add. gew. VD**, $O(n^2 \log n)$
  - **Lokalisationstechnik!(Separators/Seidel)**, $O(n^2)$
- Query:
  - **Voronoi Region von t in $VD(f)$** $O(\log n)$
  - Kürzesten Weg aus $I(v, \mathrm{E})$: Über E und in v abgesp. Kürz. Weg zu s, k Segmente, O(k)

Alg. 1.13

- Input: Triang. Polyder P, n Ecken, s
- Output: Kürzester Weg für beliebigen Anfragepunkt $t \in f$
- Preprocessing:
  - Berechne Intervalle $I(v, \mathrm{E})$, $O(n^2 \log n)$
  - Für alle f **add. gew. VD**, $O(n^2 \log n)$
  - **Lokalisationstechnik!(Separators/Seidel)**, $O(n^2)$
- Query:
  - **Voronoi Region von t in $VD(f)$** $O(\log n)$
  - Kürzesten Weg aus $I(v, \mathrm{E})$: Über E und in v abgesp. Kürz. Weg zu s, k Segmente, O(k)
  - Nur Länge,

Alg. 1.13

- Input: Triang. Polyder P, n Ecken, s
- Output: Kürzester Weg für beliebigen Anfragepunkt $t \in f$
- Preprocessing:
  - Berechne Intervalle $I(v, \mathsf{E})$, $O(n^2 \log n)$
  - Für alle f **add. gew. VD**, $O(n^2 \log n)$
  - **Lokalisationstechnik!(Separators/Seidel)**, $O(n^2)$
- Query:
  - **Voronoi Region von t in $VD(f)$** $O(\log n)$
  - Kürzesten Weg aus $I(v, \mathsf{E})$: Über E und in v abgesp. Kürz. Weg zu s, k Segmente, O(k)
  - Nur Länge, $O(1)$

Ergebnis: Theorem 1.45

Ergebnis: Theorem 1.45

Sei s auf P fest, gegeben b auf P. Die Entfernung bzw. die Kürzeste von s nach b auf P läßt sich nach Vorbereitungszeit $O(n^2 \log n)$ mit Platz $O(n^2)$ in Zeit $O(\log n)$ bzw. $O(\log n + k)$ berechnen.

Ergebnis: Theorem 1.45

Sei s auf P fest, gegeben b auf P. Die Entfernung bzw. die Kürzeste von s nach b auf P läßt sich nach Vorbereitungszeit $O(n^2 \log n)$ mit Platz $O(n^2)$ in Zeit $O(\log n)$ bzw. $O(\log n + k)$ berechnen.

(Mount,Mitchell,Papdimitriou, 1986)

Berechnung aller $I(v, \mathbf{E})$

Berechnung aller $I(v, \mathbf{E})$

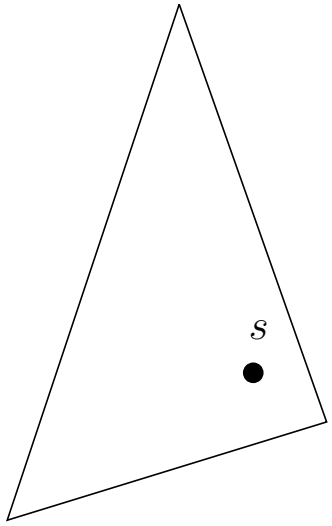
Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen!

Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen!
Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s.

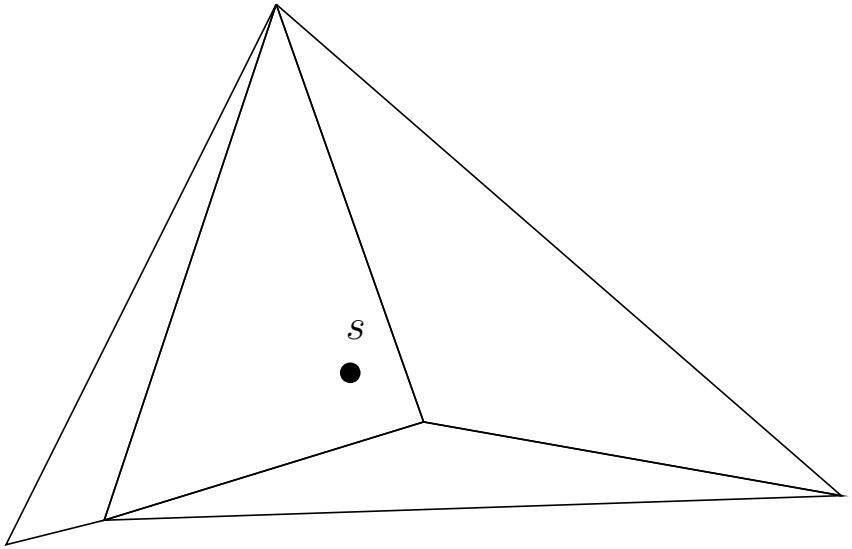
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen!
Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken!



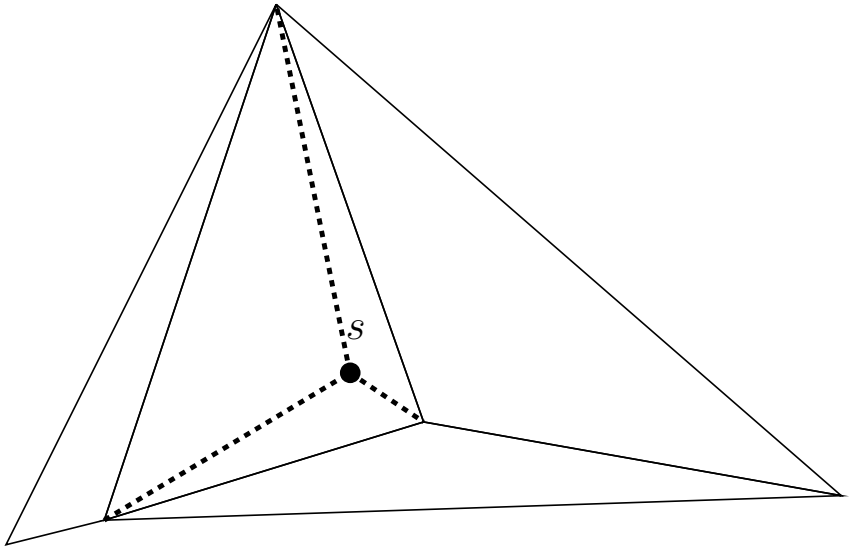
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



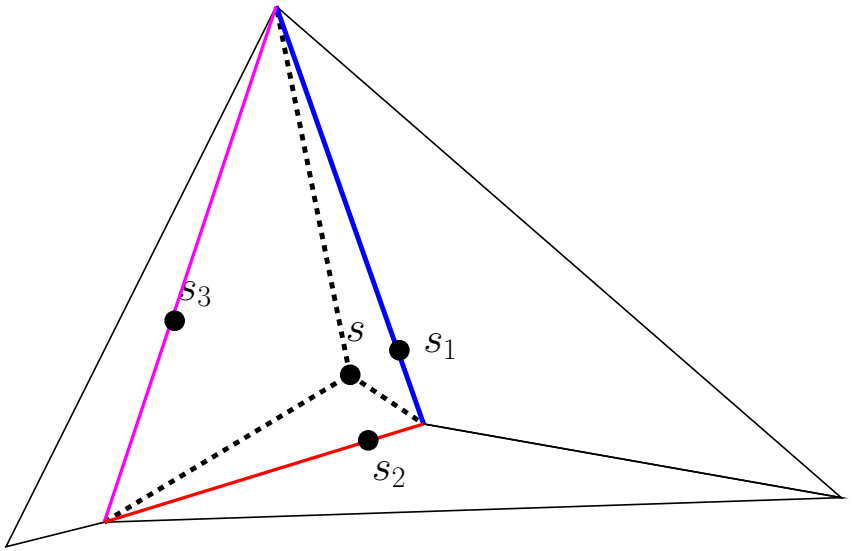
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



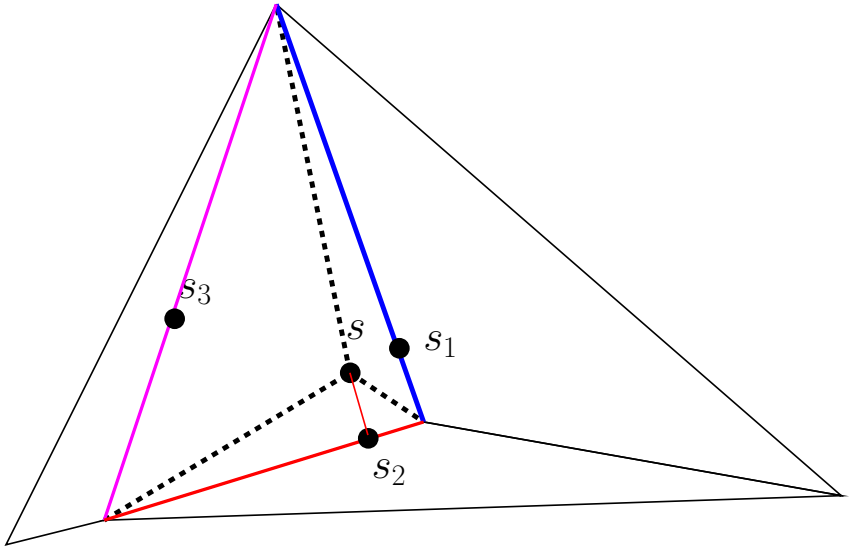
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



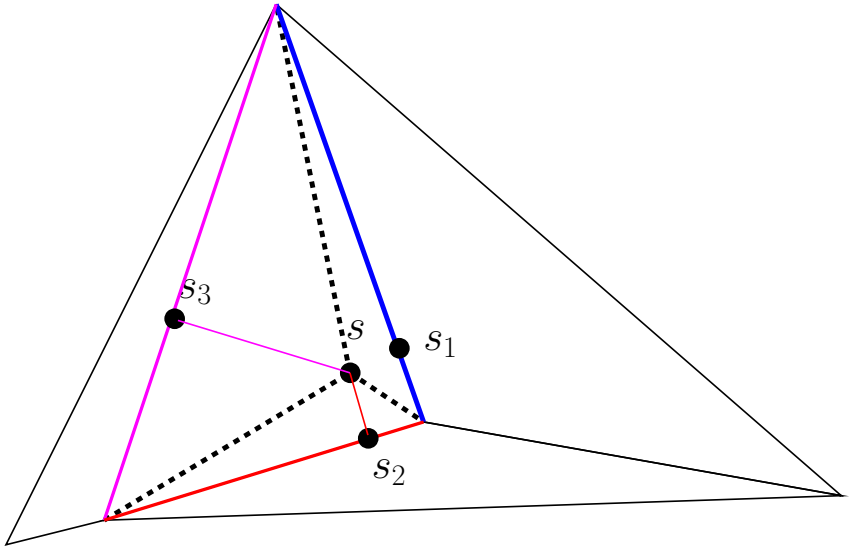
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



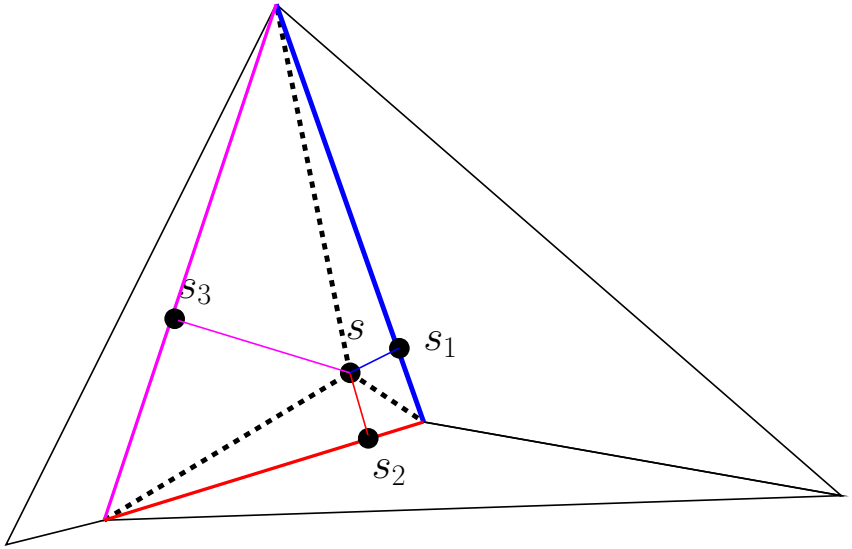
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



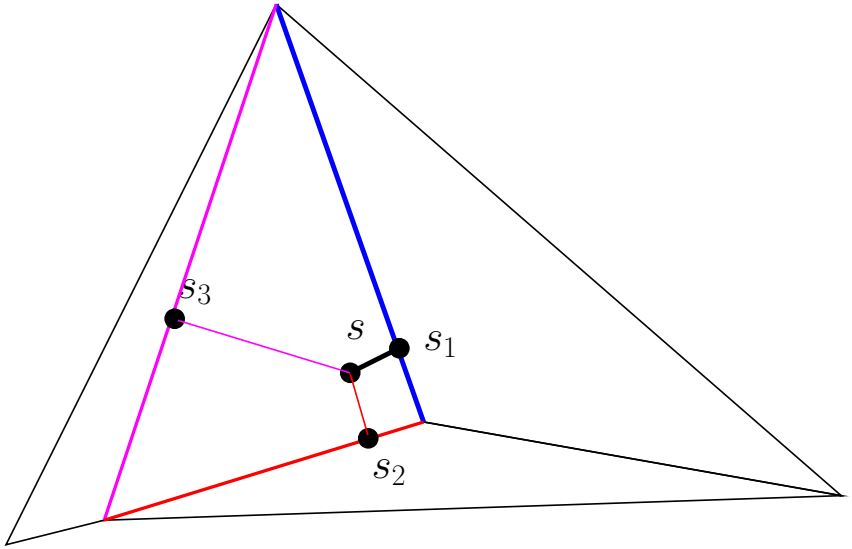
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



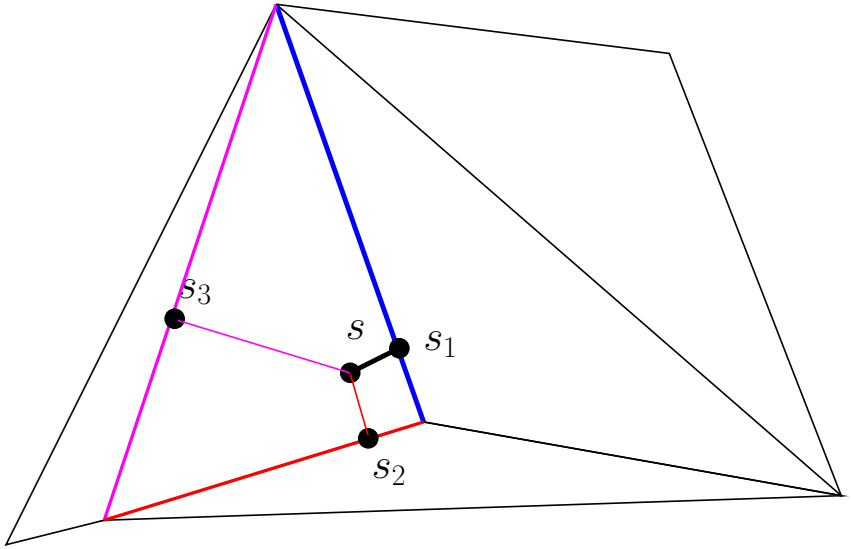
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



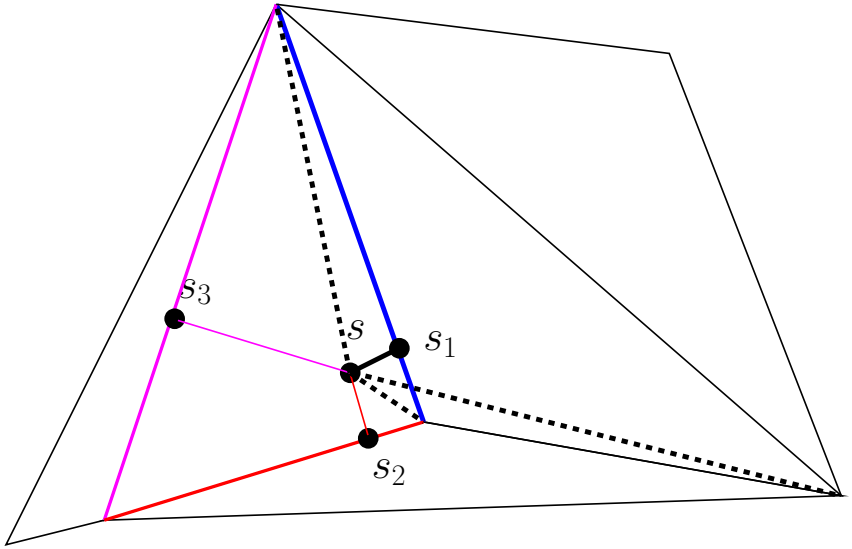
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



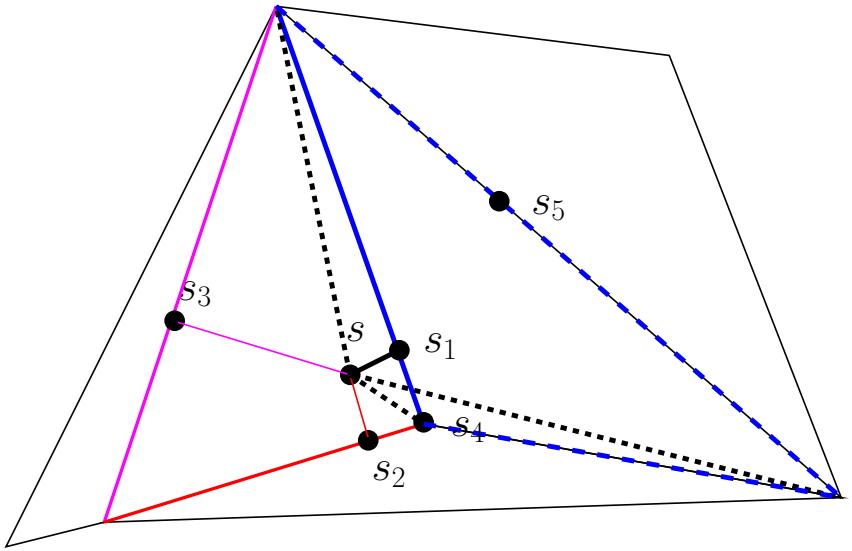
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



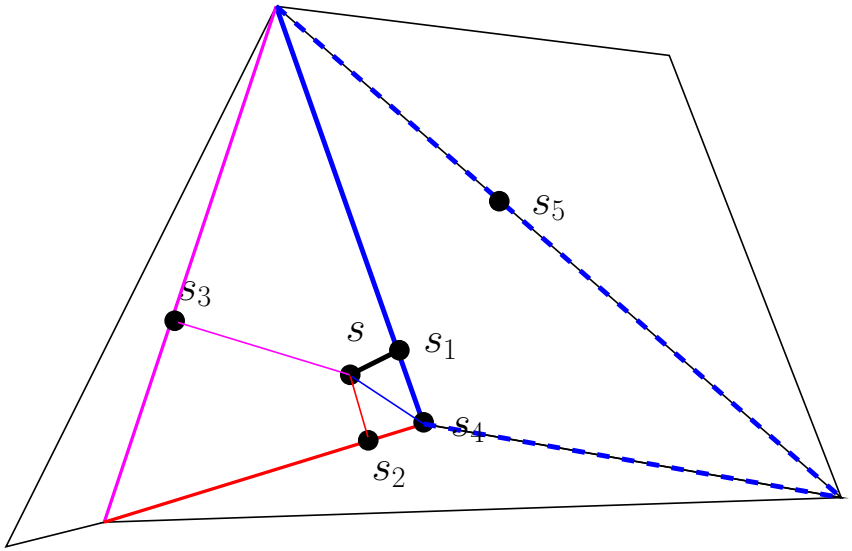
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



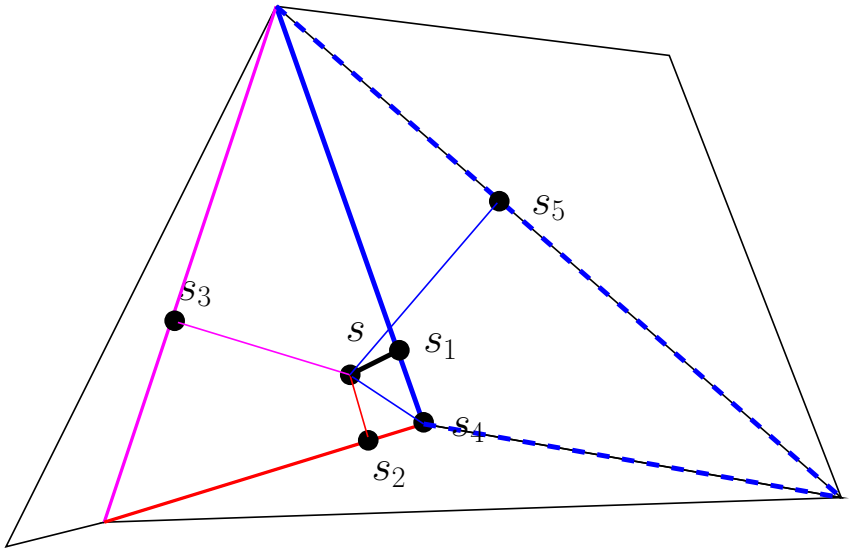
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



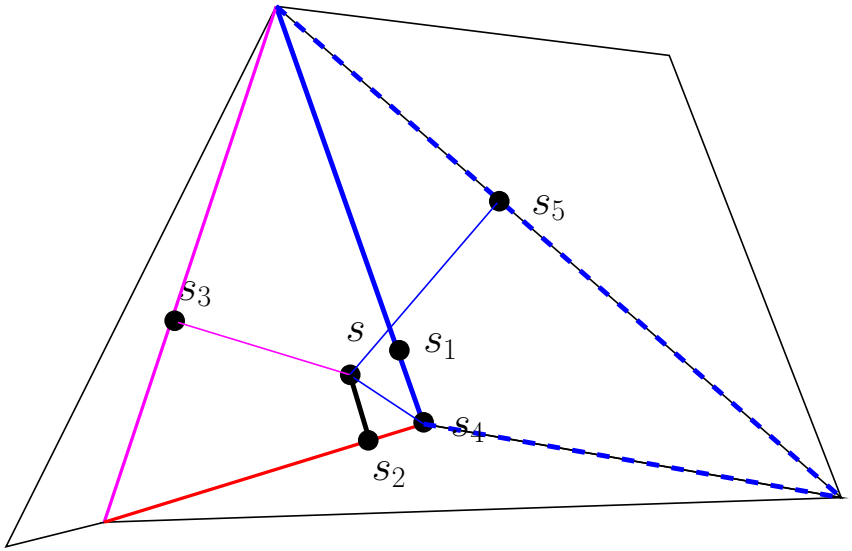
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



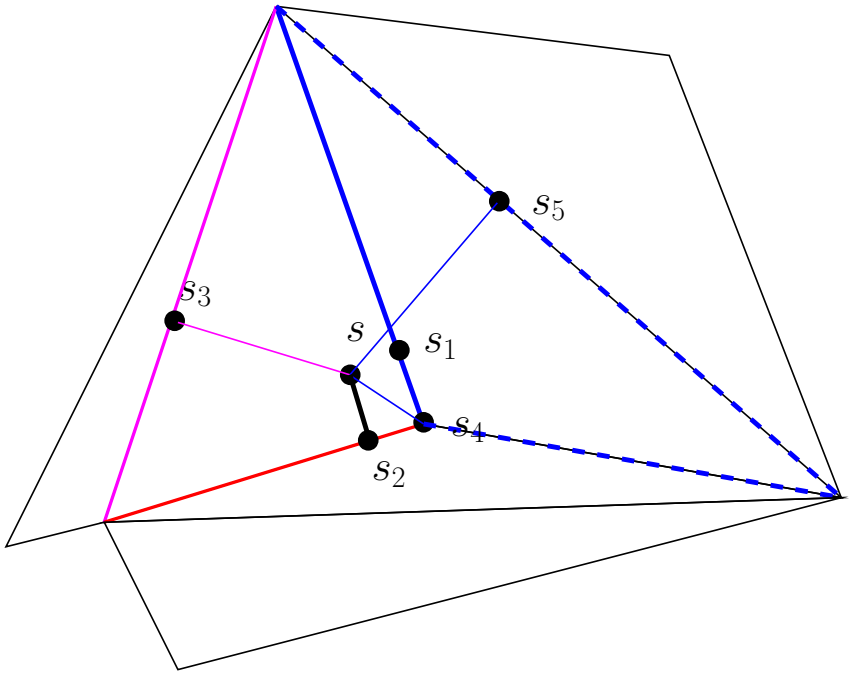
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



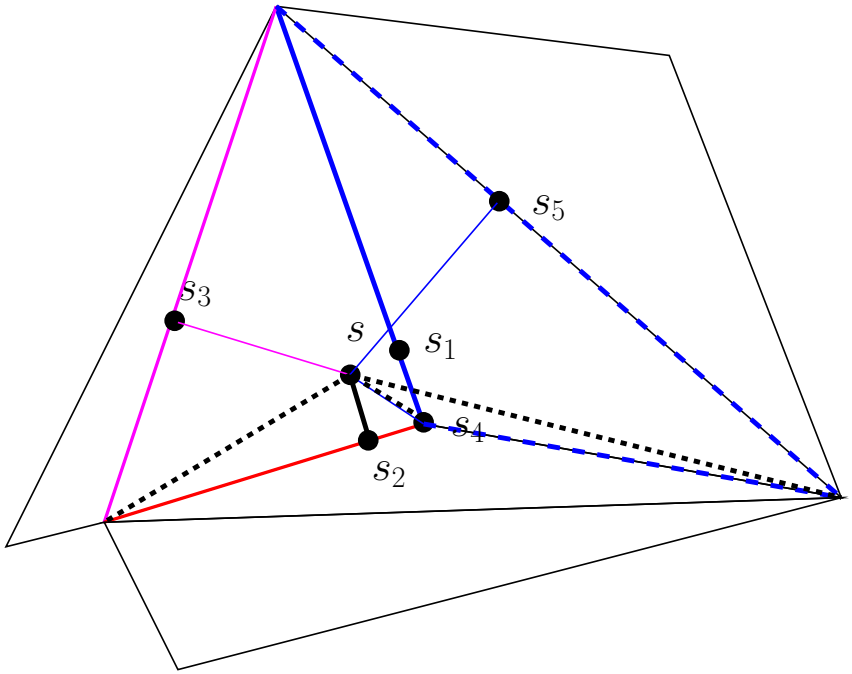
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



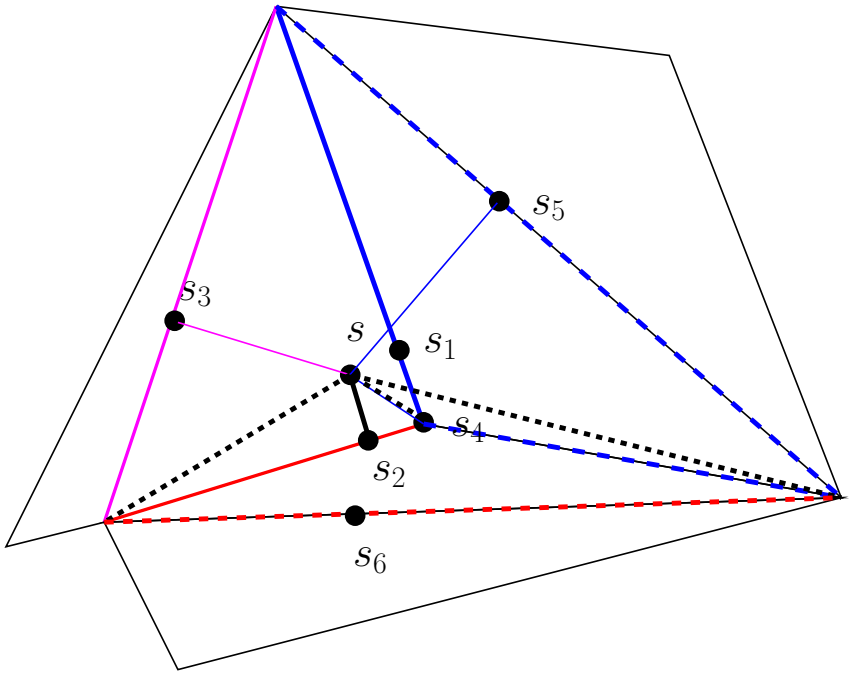
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



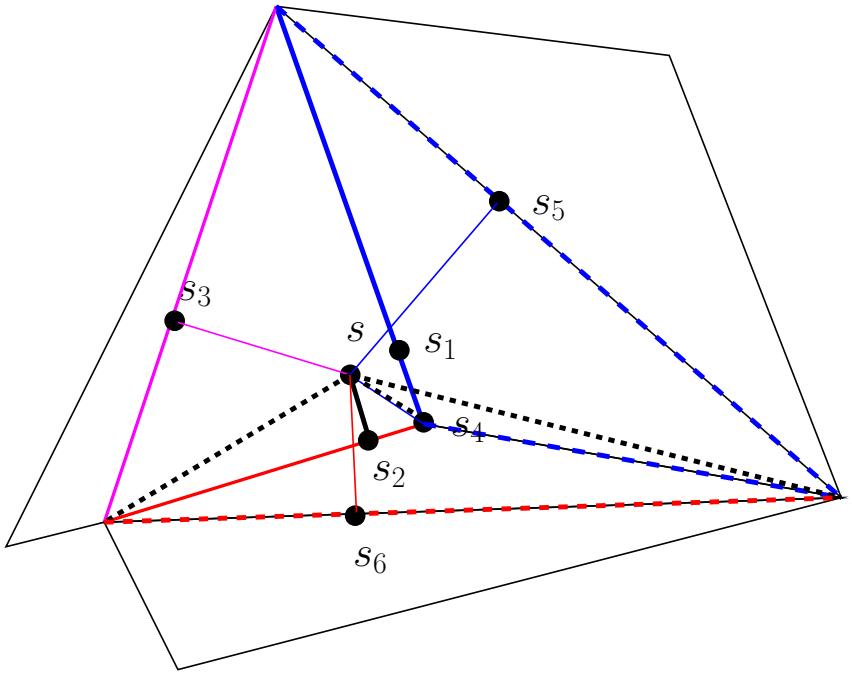
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



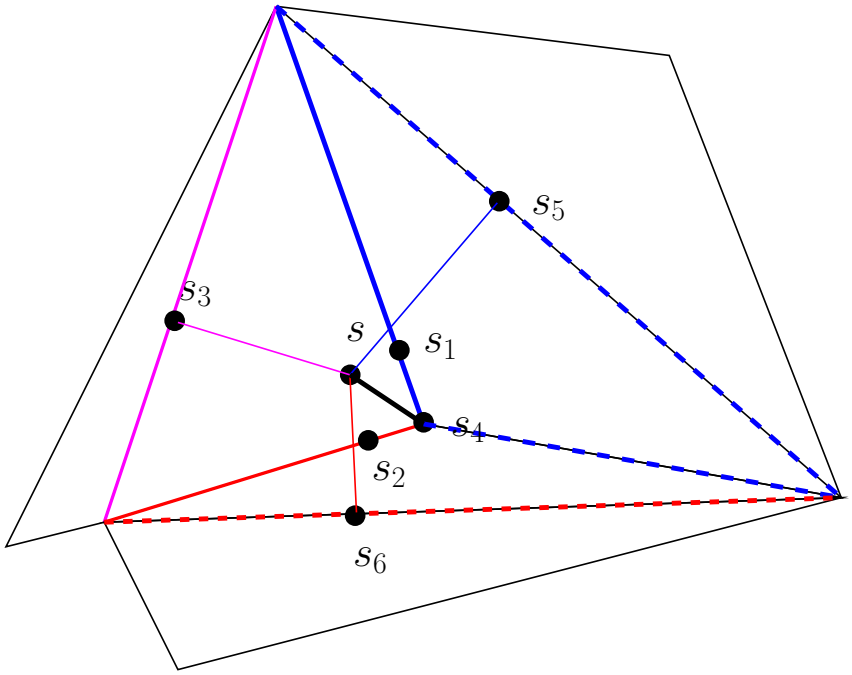
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



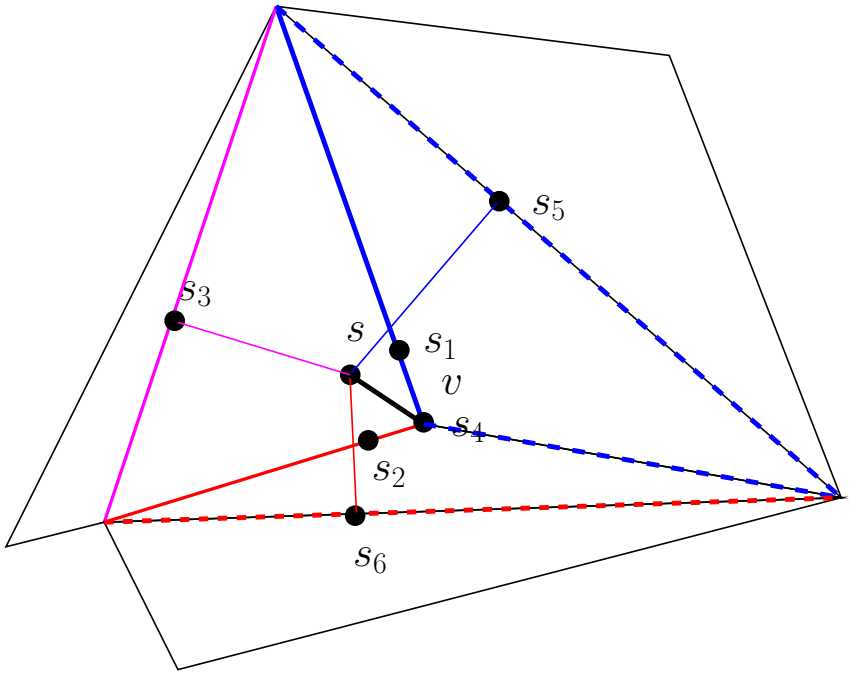
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



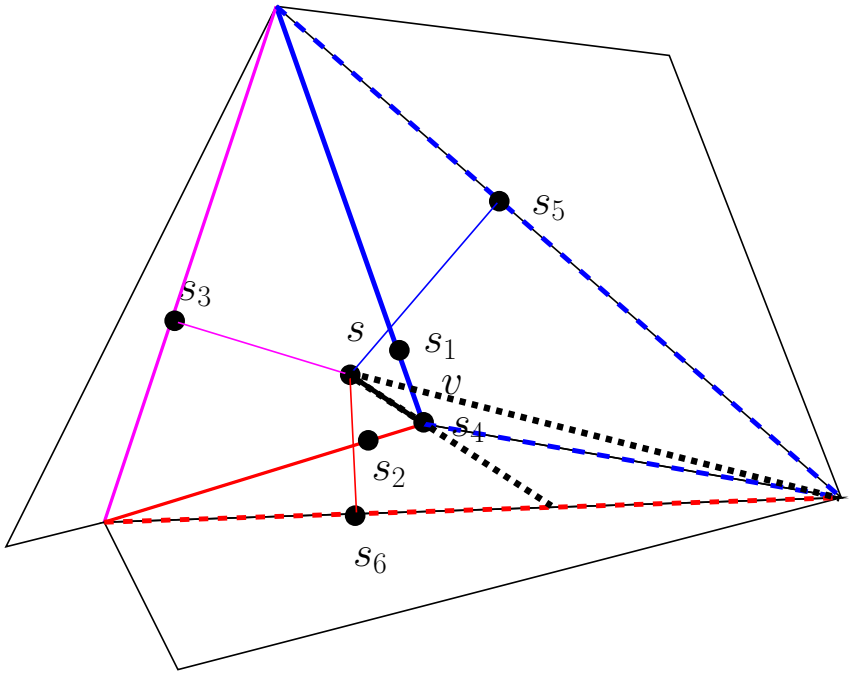
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



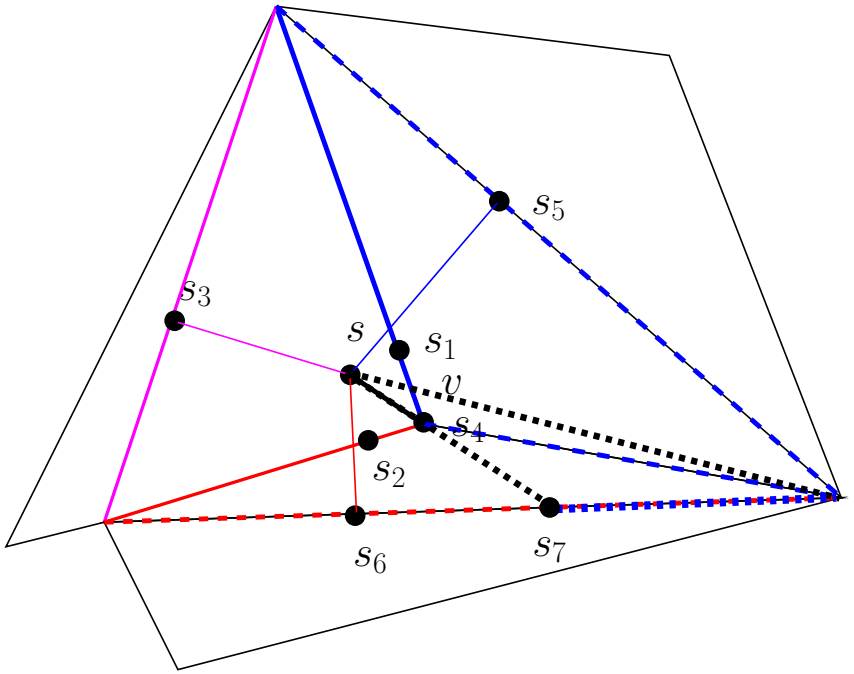
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



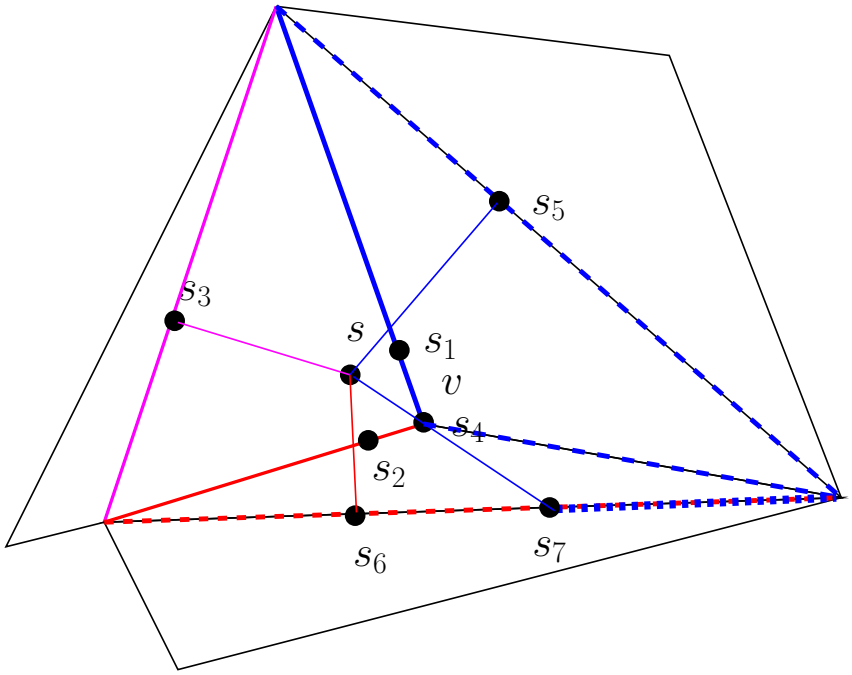
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



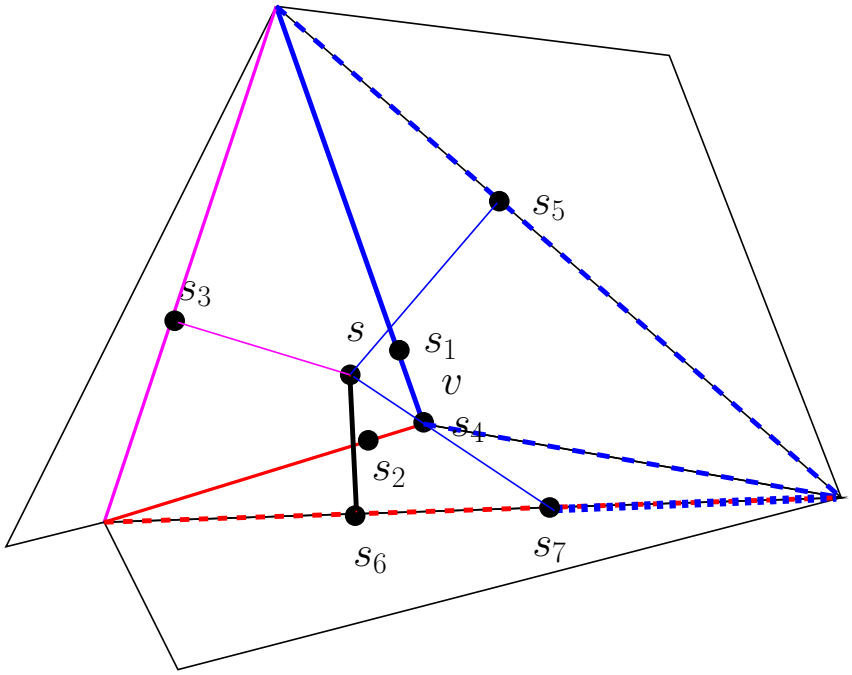
Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



Berechnung aller $I(v, \mathbf{E})$

Continous Dijkstra: Sukzessive (Teil)Intervalle festlegen! Garantiepunkt s_i auf dem Intervall, nächster Punkt zu s. Kürzeste Wege zu Knoten merken! Listen von Intervallen aufbauen!



Alg. 1.12 Continuous Dijkstra

Alg. 1.12 Continuous Dijkstra

- **DS:**

Alg. 1.12 Continuous Dijkstra

- **DS:**
  - Priority Queue W: Intervalle nach s_i-Abstände.
  - Intervalllisten auf den Kanten in balancierten Baum vorhalten.

Alg. 1.12 Continuous Dijkstra

- **DS:**
  - Priority Queue W: Intervalle nach s_i-Abstände.
  - Intervalllisten auf den Kanten in balancierten Baum vorhalten.

- **Iterationsschritt:**

Alg. 1.12 Continuous Dijkstra

- **DS:**
  - Priority Queue W: Intervalle nach s_i-Abstände.
  - Intervalllisten auf den Kanten in balancierten Baum vorhalten.

- **Iterationsschritt:**
  - Wähle Intervall $I(v, \mathrm{E})$ mit kürzestem Abstand.

Alg. 1.12 Continuous Dijkstra

- **DS:**
  - Priority Queue W: Intervalle nach s_i-Abstände.
  - Intervalllisten auf den Kanten in balancierten Baum vorhalten.

- **Iterationsschritt:**
  - Wähle Intervall $I(v, \mathsf{E})$ mit kürzestem Abstand.
    $d(s, v) + |\bar{v} - s_i|$ ist am geringsten!

Alg. 1.12 Continuous Dijkstra

- **DS:**
  - Priority Queue W: Intervalle nach s_i-Abstände.
  - Intervalllisten auf den Kanten in balancierten Baum vorhalten.

- **Iterationsschritt:**
  - Wähle Intervall $I(v, \mathrm{E})$ mit kürzestem Abstand.
    $d(s, v) + |\bar{v} - s_i|$ ist am geringsten! Wird stets Intervall bleiben!
    (Eventuell später gekürzt).

Alg. 1.12 Continuous Dijkstra

- **DS:**
  - Priority Queue W: Intervalle nach s_i-Abstände.
  - Intervalllisten auf den Kanten in balancierten Baum vorhalten.

- **Iterationsschritt:**
  - Wähle Intervall $I(v, \mathrm{E})$ mit kürzestem Abstand. $d(s, v) + |\bar{v} - s_i|$ ist am geringsten! Wird stets Intervall bleiben! (Eventuell später gekürzt).
  - Führe dieses Intervall auf die Kanten des Dreiecks fort, auf dem s_i lag und das jenseits der Folge E liegt.

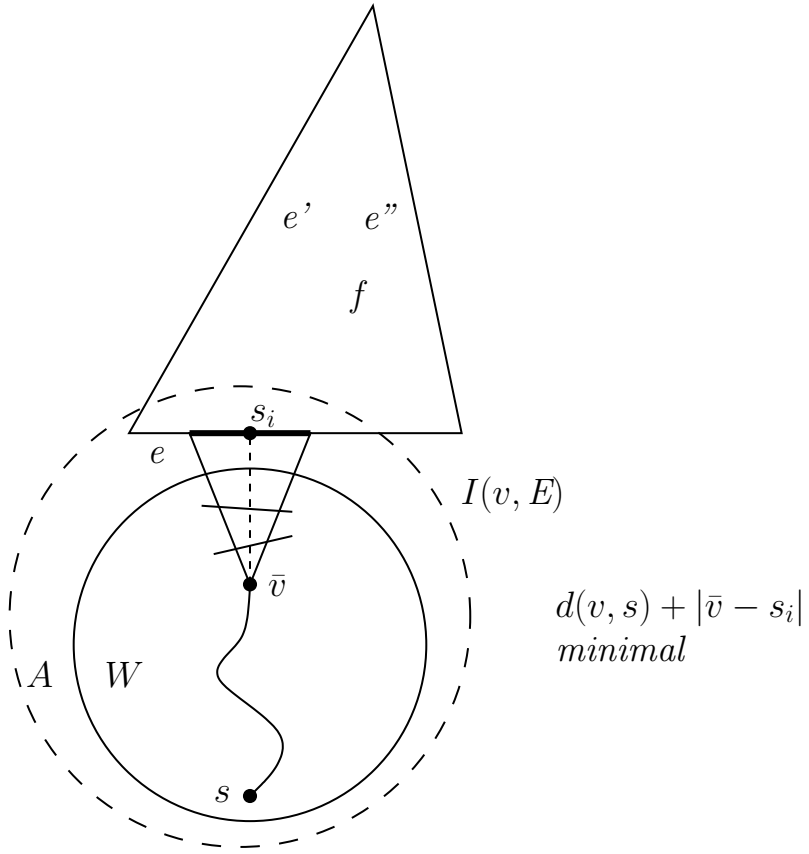
Alg. 1.12 Continuous Dijkstra

- **DS:**
  - Priority Queue W: Intervalle nach s_i-Abstände.
  - Intervalllisten auf den Kanten in balancierten Baum vorhalten.

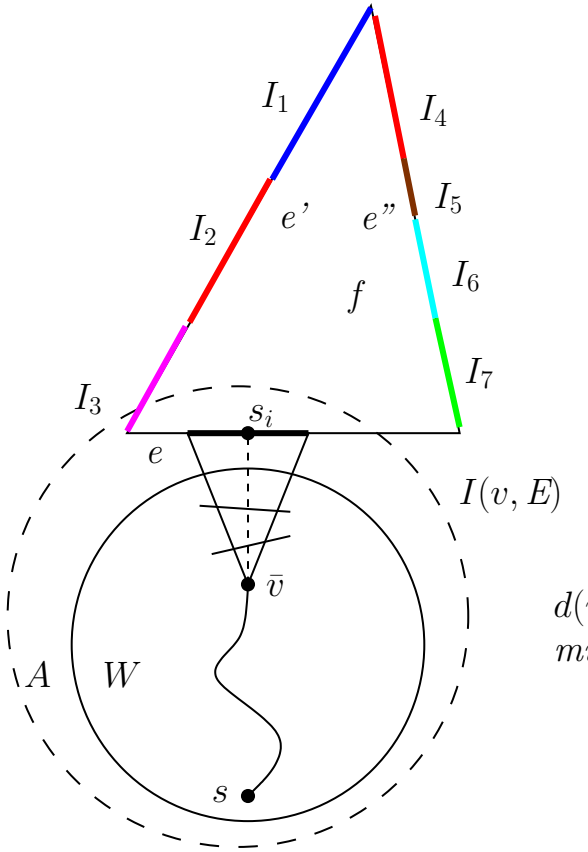
- **Iterationsschritt:**
  - Wähle Intervall $I(v, \mathrm{E})$ mit kürzestem Abstand. $d(s, v) + |\bar{v} - s_i|$ ist am geringsten! Wird stets Intervall bleiben! (Eventuell später gekürzt).
  - Führe dieses Intervall auf die Kanten des Dreiecks fort, auf dem s_i lag und das jenseits der Folge E liegt.
  - Sortiere zwei neue Intervalle in die Liste der Intervalle der beiden Kanten ein. Bestimme jeweilige s_j und füge in W ein.

Schritt des Einsortierens!

Schritt des Einsortierens!

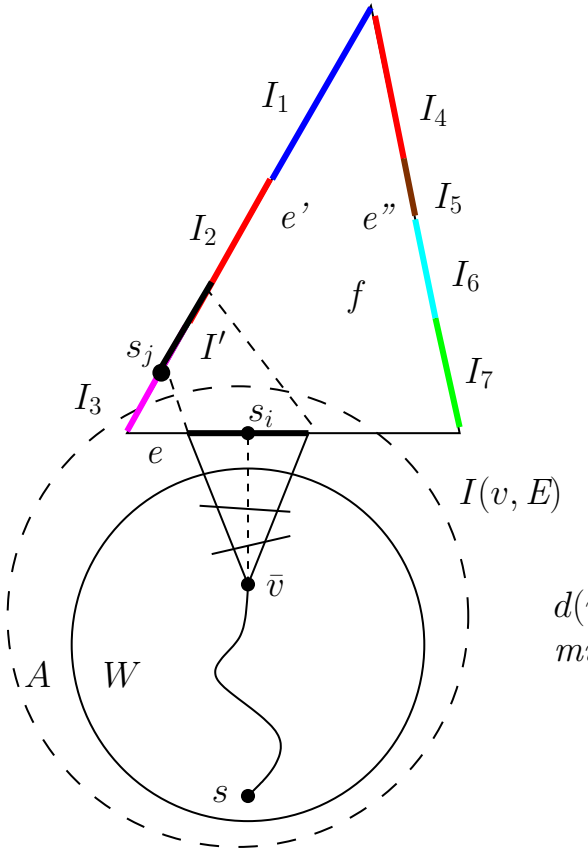


Schritt des Einsortierens!

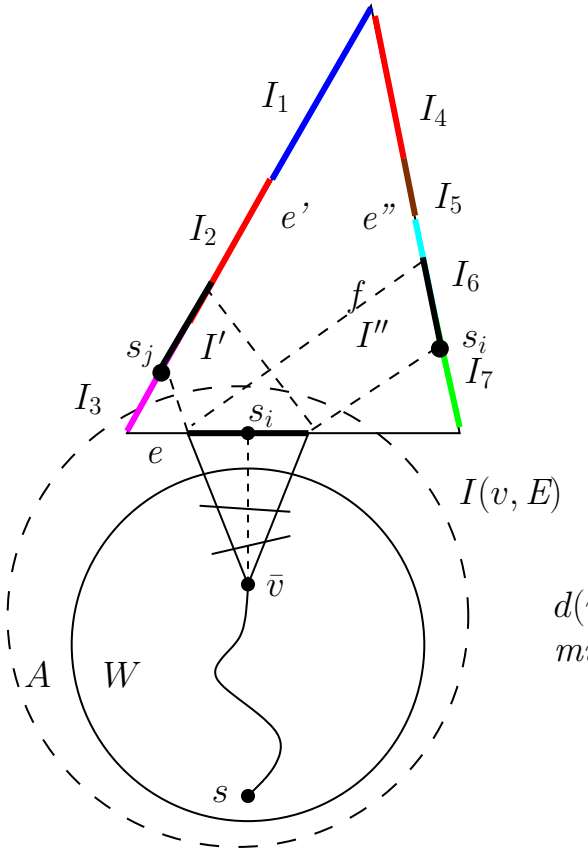


$d(v, s) + |\bar{v} - s_i|$
minimal

Schritt des Einsortierens!

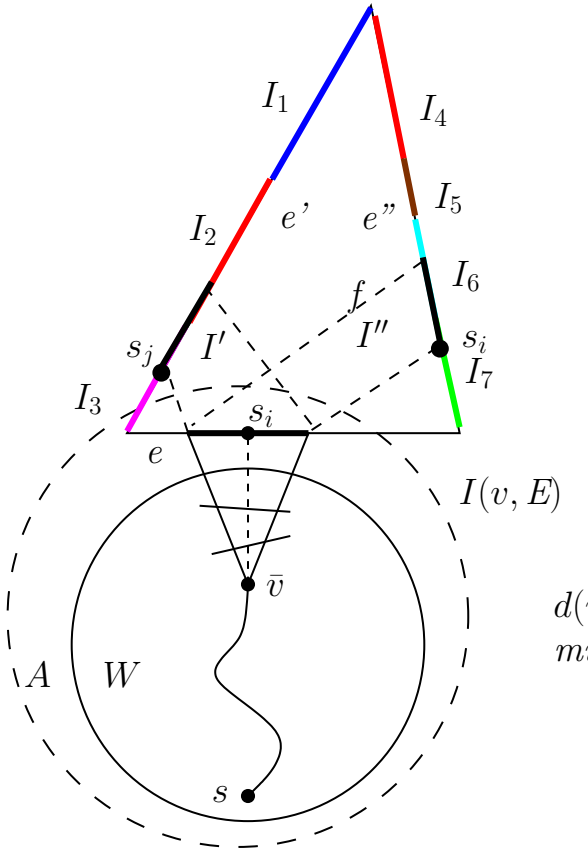


Schritt des Einsortierens!



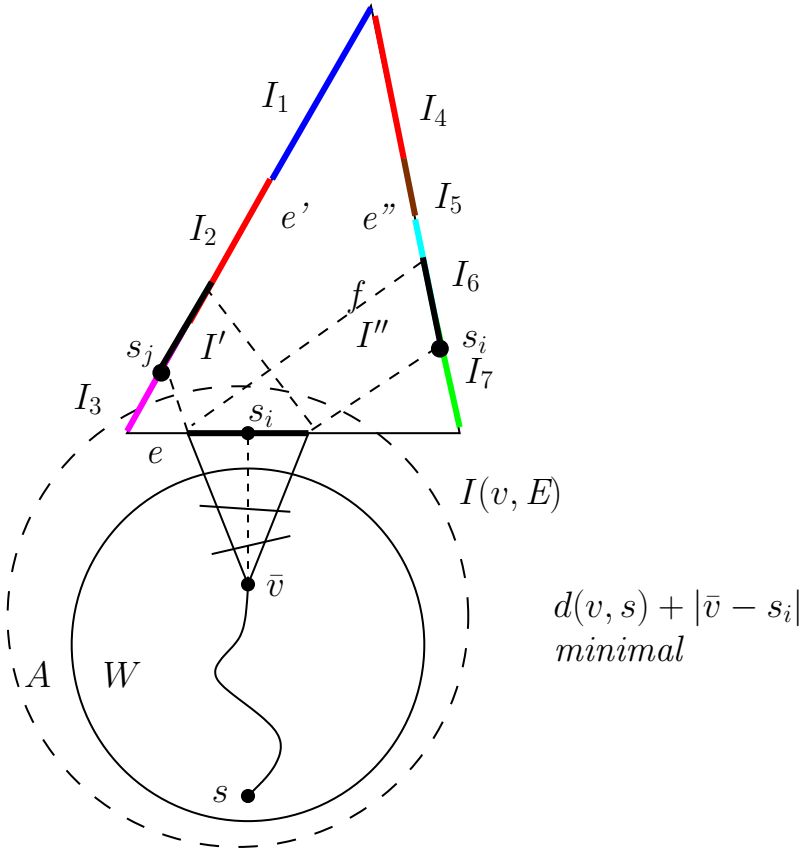
$d(v, s) + |\bar{v} - s_i|$
minimal

Schritt des Einsortierens!



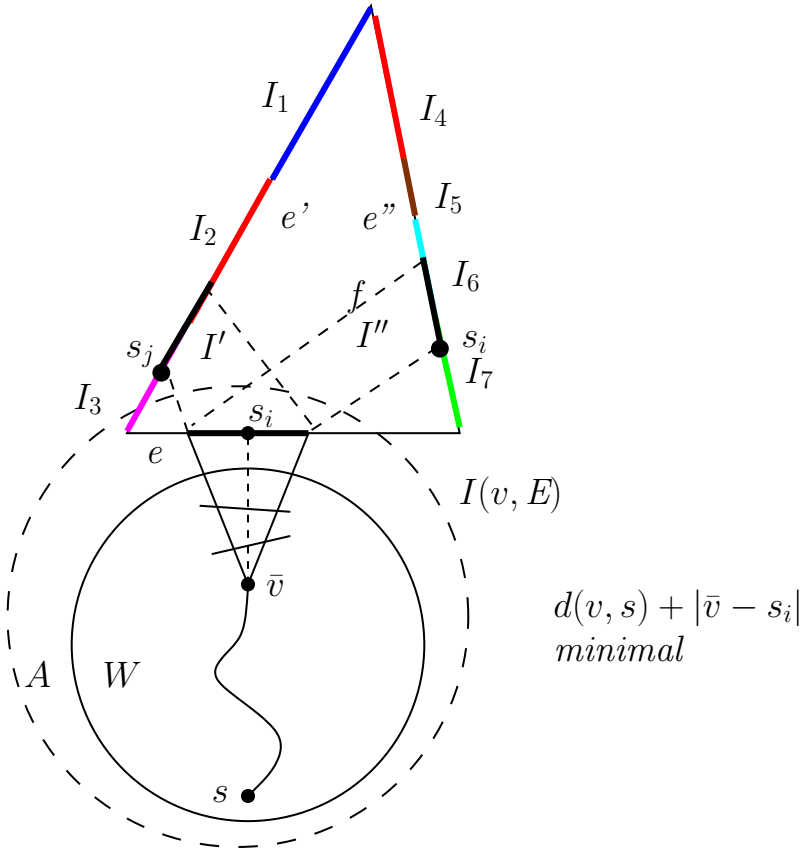
$d(v, s) + |\bar{v} - s_i|$
minimal

Schritt des Einsortierens!



Einsortieren in Baum der Intervalle!

Schritt des Einsortierens!



Einsortieren in Baum der Intervalle! $O(\log n)$!