

Teil 2: Approximationsalgorithmen

Greedy-Algorithmus

Geg: Punktmenge $P = \{p_1, \dots, p_n\}$, Menge rechteckiger
Label $L = \{l_1, \dots, l_n\}$ mit Höhe 1 und variabler Breite

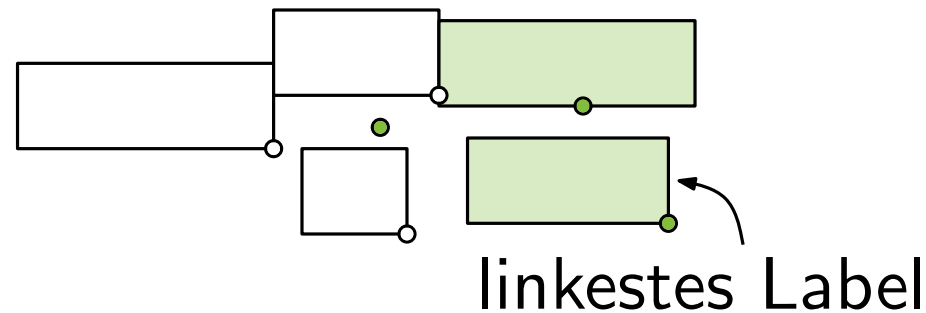
Modell: 4-Slider

Greedy-Algorithmus

Geg: Punktmenge $P = \{p_1, \dots, p_n\}$, Menge rechteckiger Label $L = \{l_1, \dots, l_n\}$ mit Höhe 1 und variabler Breite

Modell: 4-Slider

Def: Für eine Menge P' von Punkten mit bereits platzierten Labeln heißt ein Label l für einen Punkt $p \in P \setminus P'$ **linkestes** Label, falls seine rechte Kante unter allen noch konfliktfrei platzierbaren Labeln am weitesten links liegt.



Greedy-Algorithmus

Geg: Punktmenge $P = \{p_1, \dots, p_n\}$, Menge rechteckiger Label $L = \{l_1, \dots, l_n\}$ mit Höhe 1 und variabler Breite

Modell: 4-Slider

Def: Für eine Menge P' von Punkten mit bereits platzierten Labeln heißt ein Label l für einen Punkt $p \in P \setminus P'$ **linkestes** Label, falls seine rechte Kante unter allen noch konfliktfrei platzierbaren Labeln am weitesten links liegt.

Algorithmus Greedy4S(P,L)

while linkestes Label l existiert **do**
 └ platziere l linkest möglich

Lemma 1: Greedy4S berechnet eine Faktor- Approximation.

Versuchen Sie den Approximationsfaktor zu bestimmen!

Greedy-Algorithmus

Geg: Punktmenge $P = \{p_1, \dots, p_n\}$, Menge rechteckiger Label $L = \{l_1, \dots, l_n\}$ mit Höhe 1 und variabler Breite

Modell: 4-Slider

Def: Für eine Menge P' von Punkten mit bereits platzierten Labeln heißt ein Label l für einen Punkt $p \in P \setminus P'$ **linkestes** Label, falls seine rechte Kante unter allen noch konfliktfrei platzierbaren Labeln am weitesten links liegt.

Algorithmus Greedy4S(P,L)

while linkestes Label l existiert **do**
 | platziere l linkest möglich

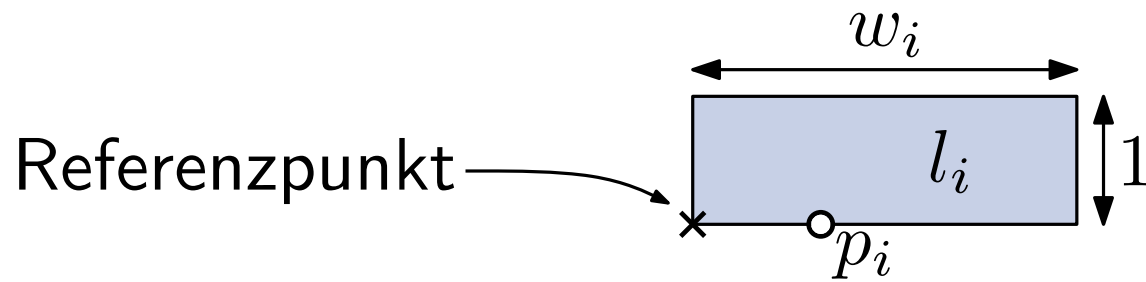
Lemma 1: Greedy4S berechnet eine Faktor-1/2 Approximation.

Laufzeit

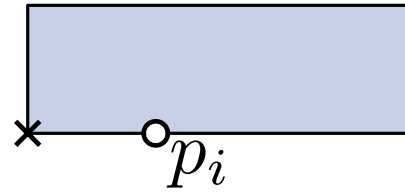
Eine naive Implementierung von Greedy4S benötigt $O(n^3)$ Zeit.

Ziel: Nutze geeignete geometrische Datenstrukturen um die Laufzeit auf $O(n \log n)$ zu senken.

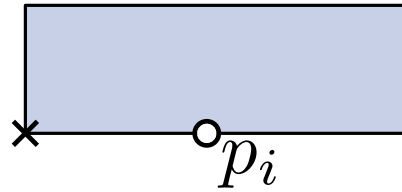
Notation



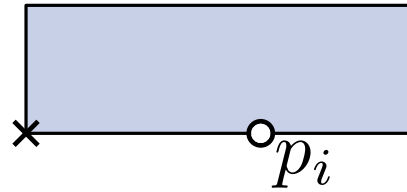
Notation



Notation



Notation



Notation



Notation



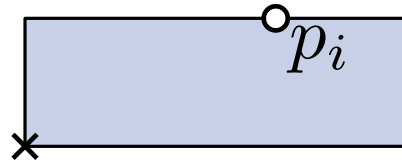
Notation



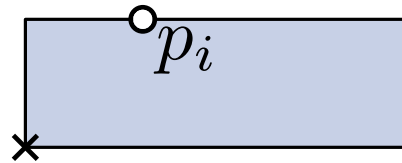
Notation



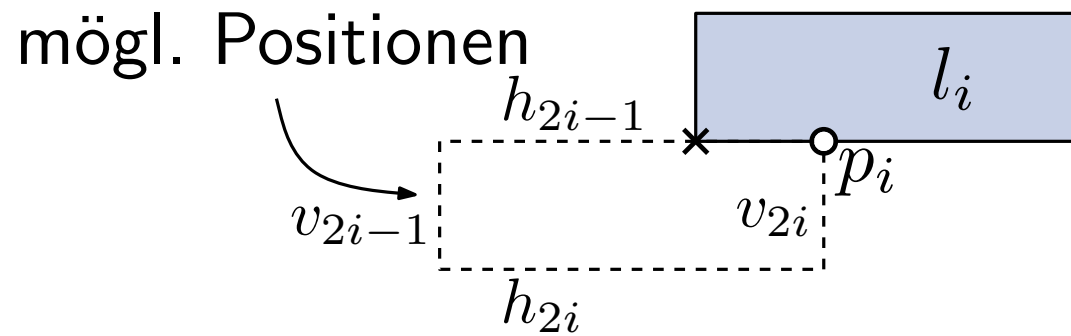
Notation

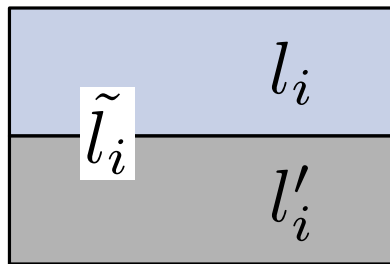
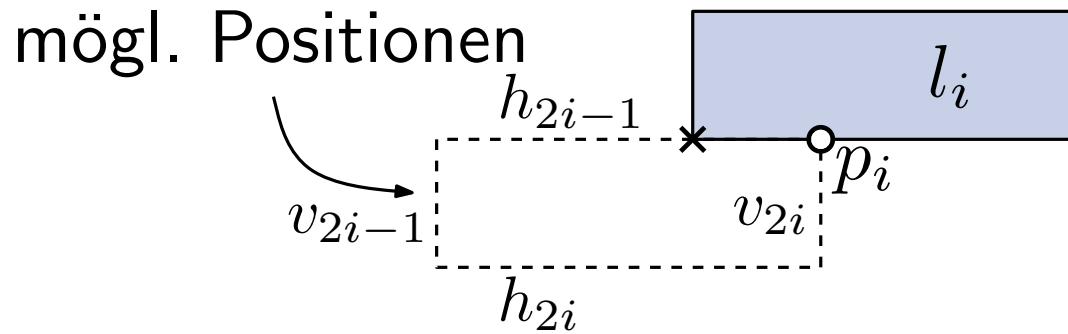


Notation



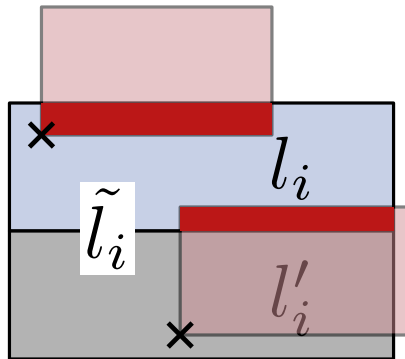
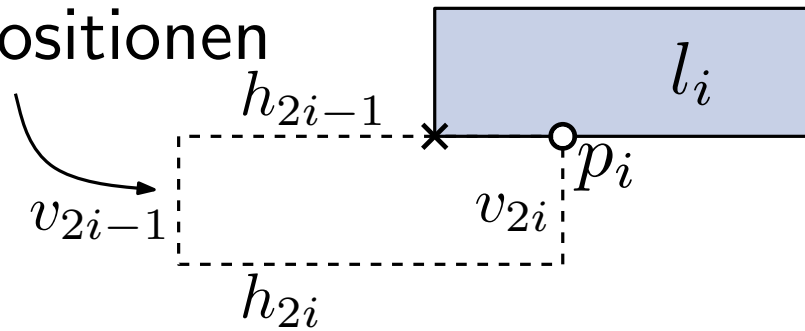
Notation



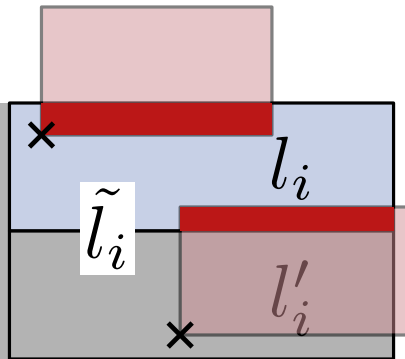
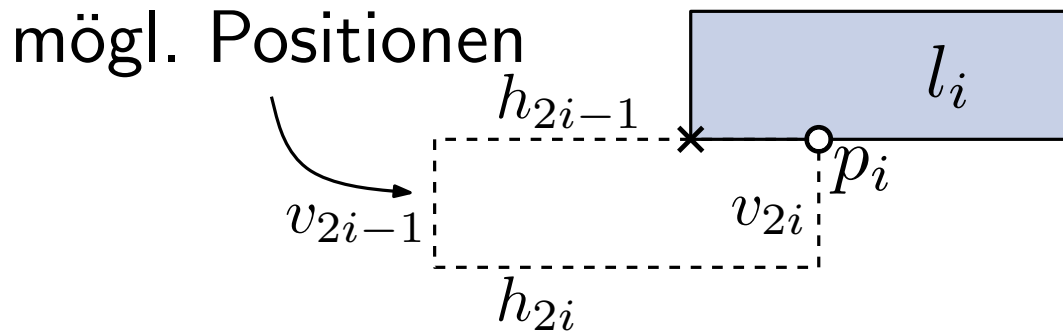


Label l_i und **erweitertes Label** $\tilde{l}_i$

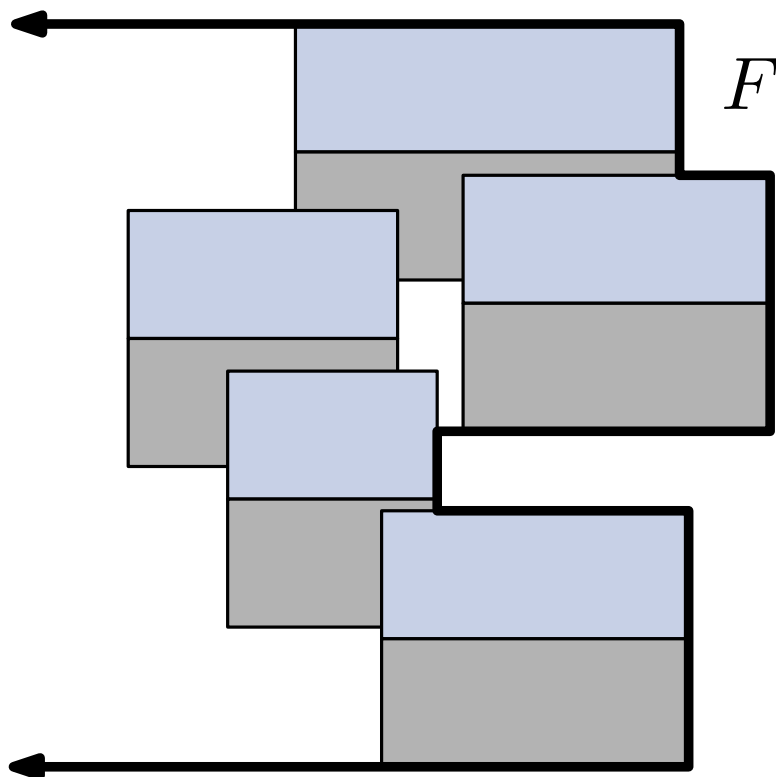
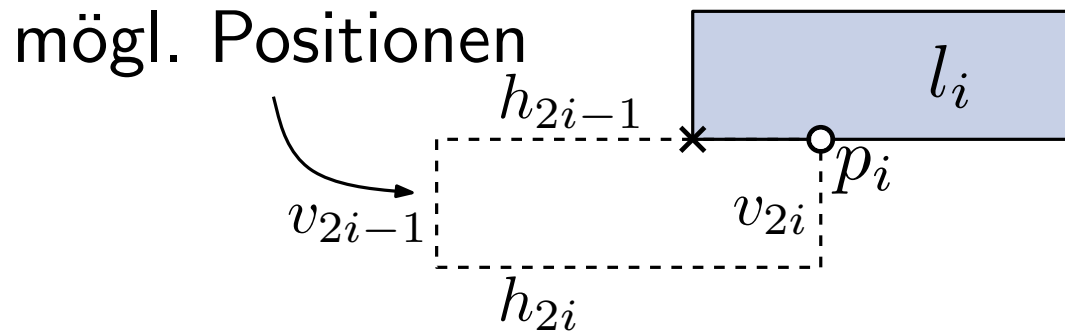
mögl. Positionen



Label l_i und **erweitertes Label** $\tilde{l}_i$
kein Referenzpunkt darf in $\tilde{l}_i$ liegen



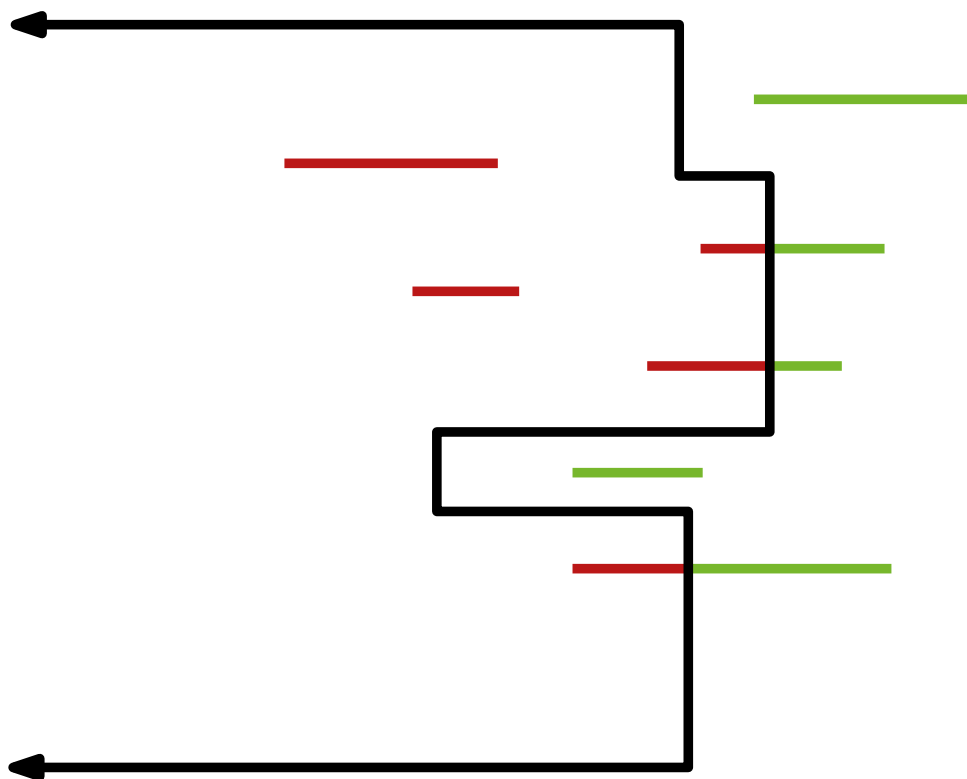
Label l_i und **erweitertes Label** $\tilde{l}_i$
 kein Referenzpunkt darf in $\tilde{l}_i$ liegen
 da immer linkstes Label platziert
 wird, kein Referenzpunkt links von $\tilde{l}_i$



Grenze F als *right envelope* der platzierten Label

Grenze und linkstes Label

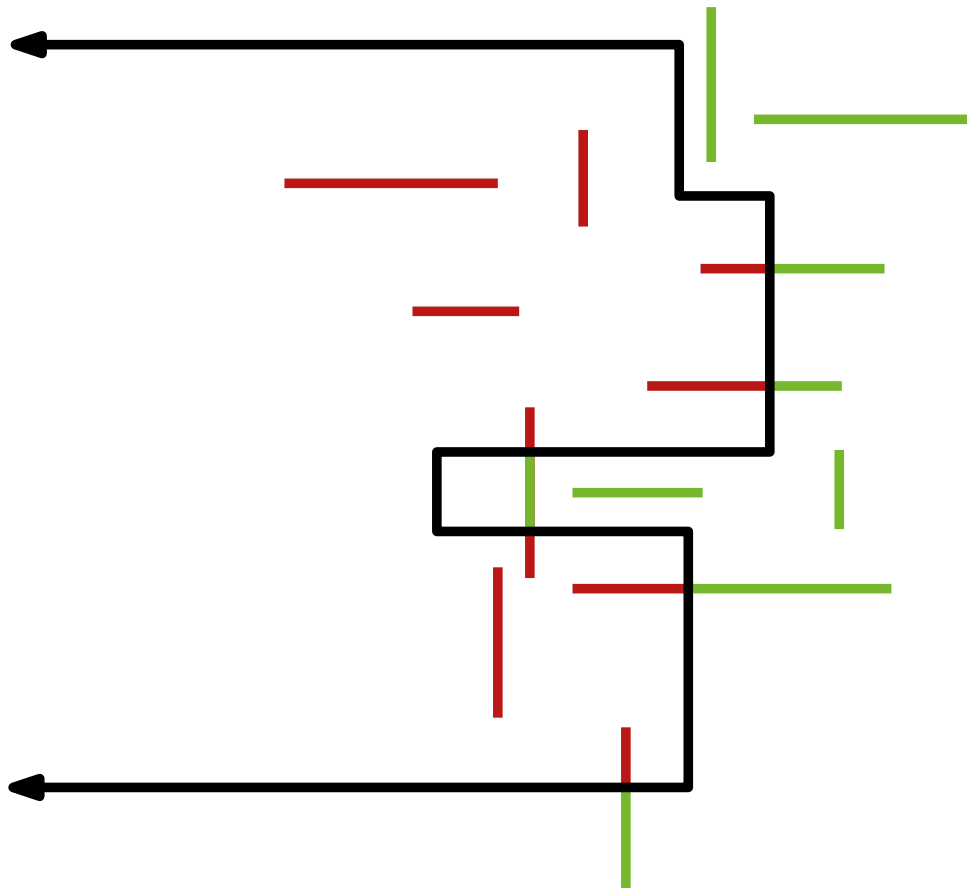
Zur Bestimmung des nächsten linken Labels genügt es F und die Strecken $h_{2i-1}, h_{2i}, v_{2i-1}, v_{2i}$ für alle Punkte p_i rechts von F zu betrachten.



- Menge H der horizontalen Strecken
- Menge V der vertikalen Strecken
- $H_{\text{right}} \subseteq H$: rechts von F
- $H_{\text{int}} \subseteq H$: schneiden F

Grenze und linkstes Label

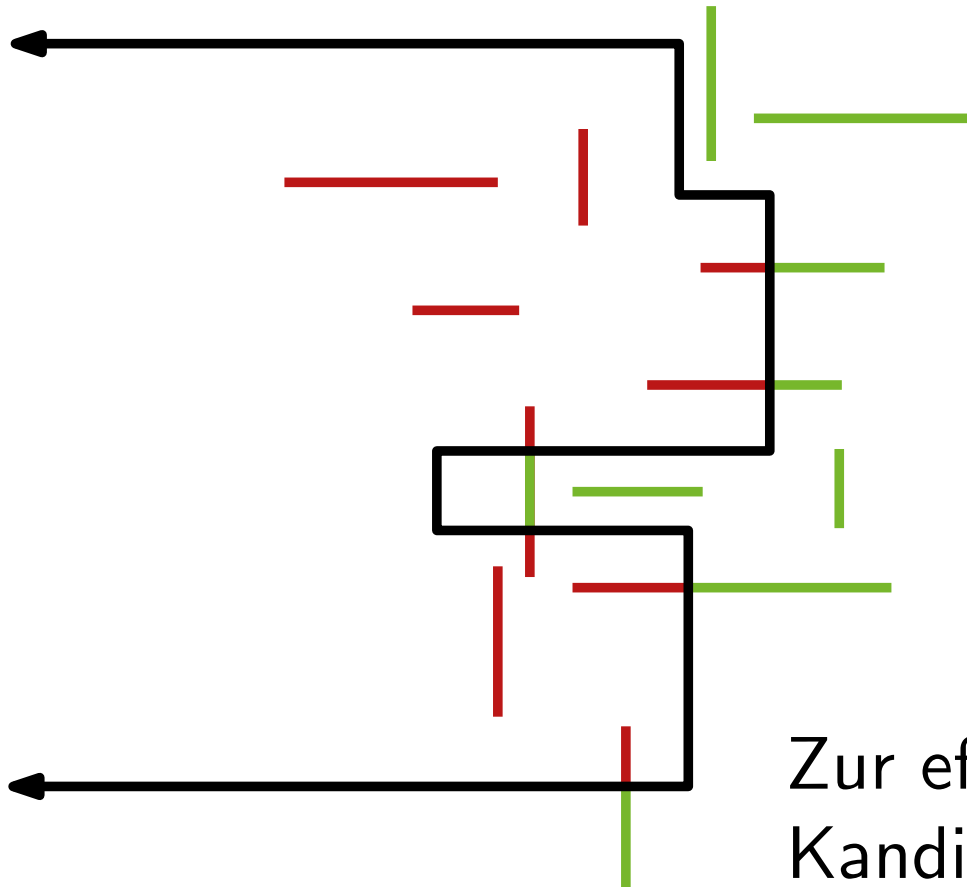
Zur Bestimmung des nächsten linken Labels genügt es F und die Strecken $h_{2i-1}, h_{2i}, v_{2i-1}, v_{2i}$ für alle Punkte p_i rechts von F zu betrachten.



- Menge H der horizontalen Strecken
- Menge V der vertikalen Strecken
- $H_{\text{right}} \subseteq H$: rechts von F
- $H_{\text{int}} \subseteq H$: schneiden F
- $V_{\text{int,right}} \subseteq V$: enthalten Punkt rechts von F
- linkeste Punkte im grünen Bereich der Strecken sind zulässige Kandidaten

Grenze und linkstes Label

Zur Bestimmung des nächsten linken Labels genügt es F und die Strecken $h_{2i-1}, h_{2i}, v_{2i-1}, v_{2i}$ für alle Punkte p_i rechts von F zu betrachten.



- Menge H der horizontalen Strecken
- Menge V der vertikalen Strecken
- $H_{\text{right}} \subseteq H$: rechts von F
- $H_{\text{int}} \subseteq H$: schneiden F
- $V_{\text{int, right}} \subseteq V$: enthalten Punkt rechts von F
- linkeste Punkte im grünen Bereich der Strecken sind zulässige Kandidaten

Zur effizienten Verwaltung der linkensten Kandidaten benötigen wir mehrere geometrische Datenstrukturen.

1) Red-Black Trees

- selbstbalancierender binärer Suchbaum der Größe $O(n)$
- Suchen, Einfügen, Löschen, Auftrennen, Konkatenieren in $O(\log n)$ Zeit

1) Red-Black Trees

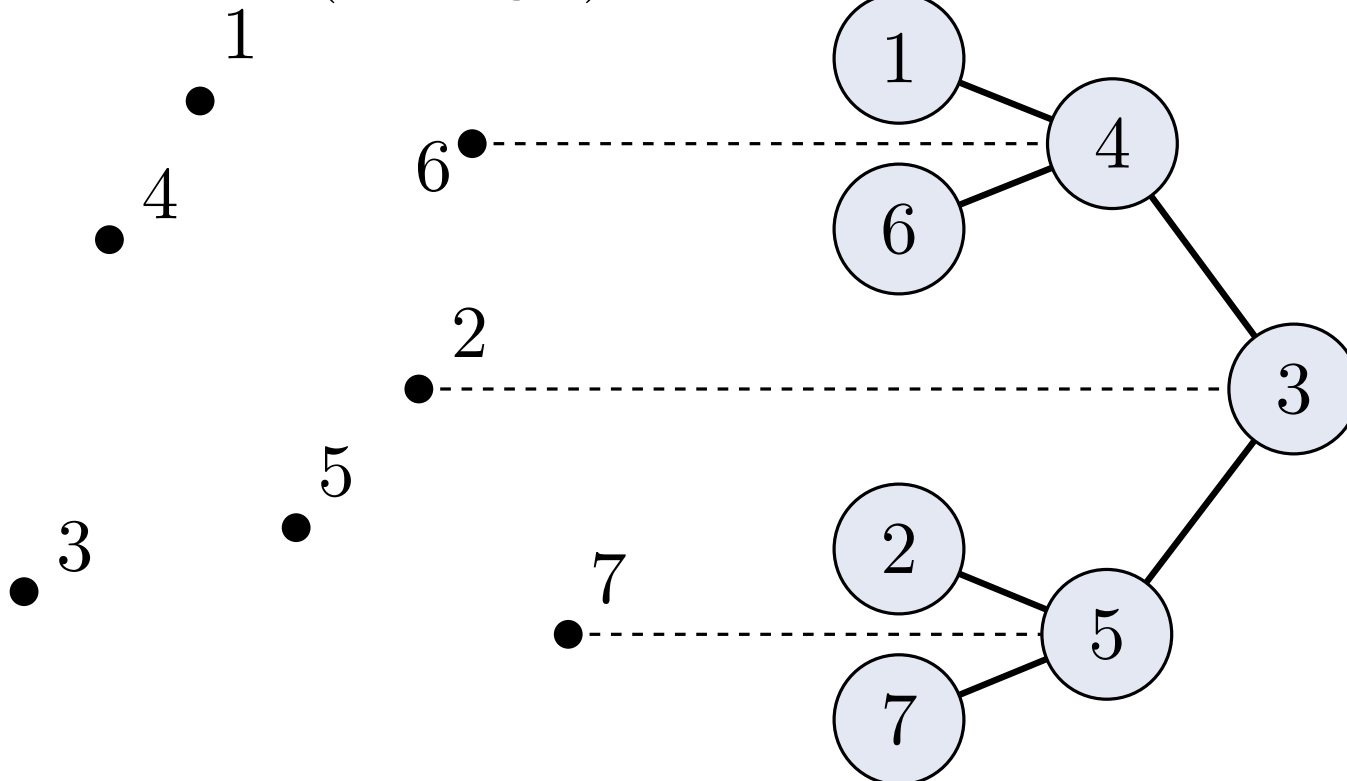
- selbstbalancierender binärer Suchbaum der Größe $O(n)$
- Suchen, Einfügen, Löschen, Auftrennen, Konkatenieren in $O(\log n)$ Zeit

2) Heaps

- baumbasierte Datenstruktur mit heap-Eigenschaft
- $\text{key}(\text{parent}(A)) \leq \text{key}(A)$ für alle A
- Größe $O(n)$
- find-min in $O(1)$
- delete-min, Einfügen, Löschen in $O(\log n)$

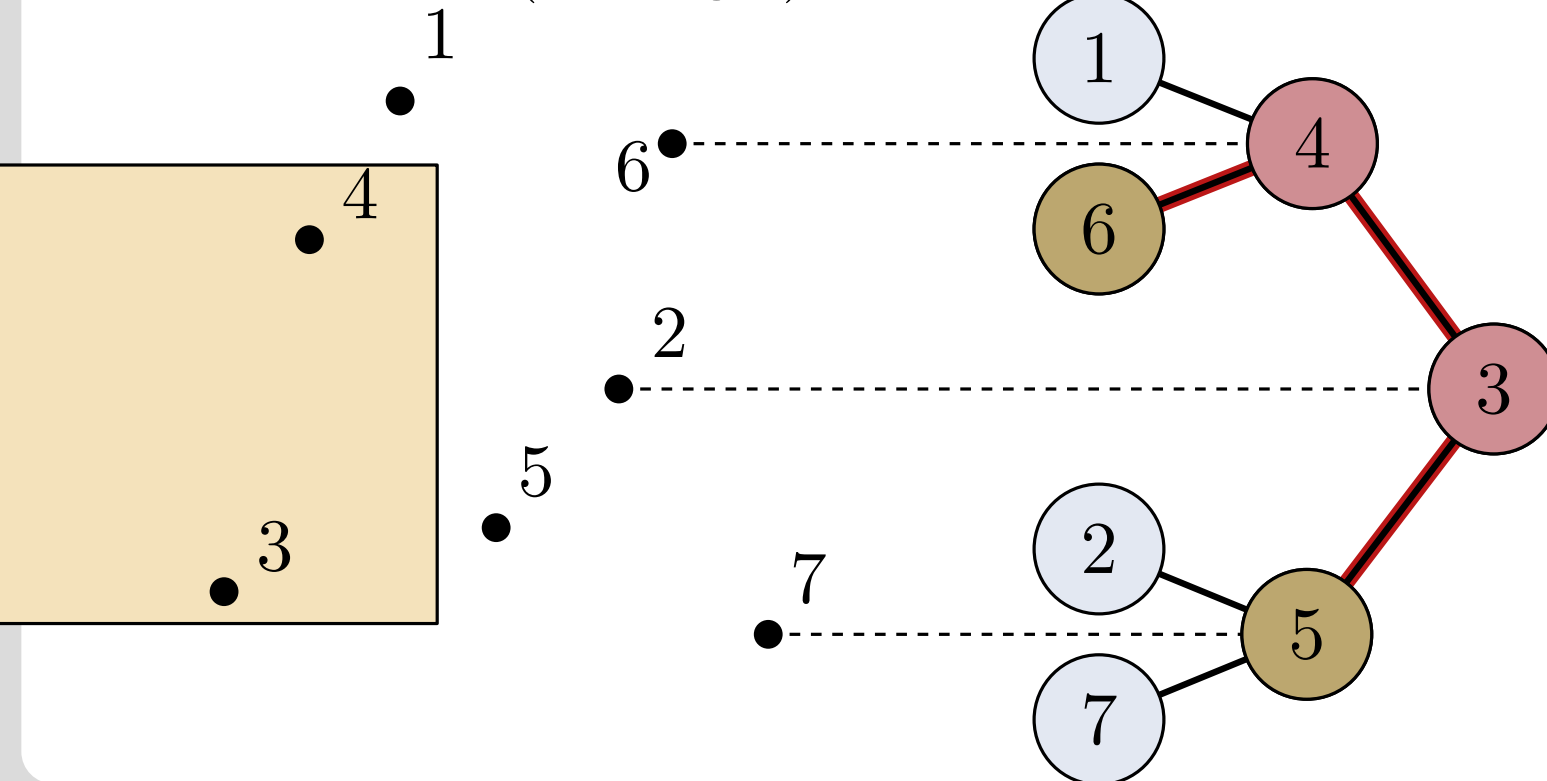
3) Priority Search Trees

- Datenstruktur für Bereichsabfragen der Form $[-\infty, x] \times [y, y']$
- Heap-Eigenschaft für x
- Suchbaum für y
- Größe $O(n)$
- Einfügen, Löschen in $O(\log n)$
- Suchen in $O(k + \log n)$



3) Priority Search Trees

- Datenstruktur für Bereichsabfragen der Form $[-\infty, x] \times [y, y']$
- Heap-Eigenschaft für x
- Suchbaum für y
- Größe $O(n)$
- Einfügen, Löschen in $O(\log n)$
- Suchen in $O(k + \log n)$



1) Menge H_{right}

- speichere für jede Strecke in H_{right} deren rechten Endpunkt (= linkest mögliche Position der rechten Labelseite)
- Datenstruktur: min-Heap $\mathcal{H}_{\text{right}}$

1) Menge H_{right}

- speichere für jede Strecke in H_{right} deren rechten Endpunkt (= linkest mögliche Position der rechten Labelseite)
- Datenstruktur: min-Heap $\mathcal{H}_{\text{right}}$

2) Menge H_{int}

- Red-Black Tree $\mathcal{T}_i$ für jedes vertikale Segment f_i von F
- speichere Strecken aus H_{int} , die f_i schneiden sortiert nach y -Koordinaten in den Blättern von $\mathcal{T}_i$
- speichere zusätzlich Labelbreite an jedem Blatt
- an inneren Knoten minimale Labelbreite im Teilbaum
- min-Heap $\mathcal{H}_{\text{int}}$ für linkestes Label in jedem $\mathcal{T}_i$

3) Menge $V_{\text{int},\text{right}}$

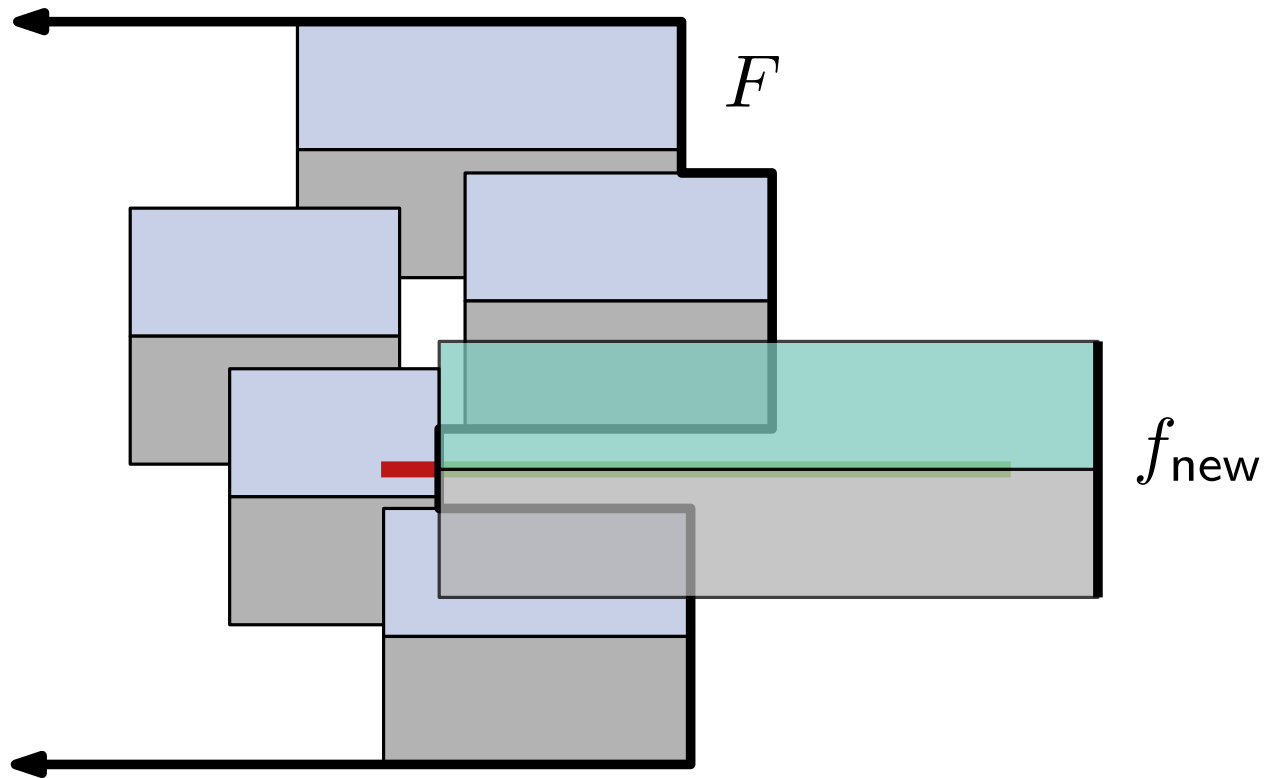
- speichere Obermenge V' mit $V_{\text{int},\text{right}} \subseteq V'$
- min-Heap $\mathcal{H}_V$ der Menge V' geordnet nach x -Koordinaten
- falls Minimum in $\mathcal{H}_V$ nicht in $V_{\text{int},\text{right}}$ liegt, verwwerfe es und wähle nächstes Minimum

3) Menge $V_{\text{int},\text{right}}$

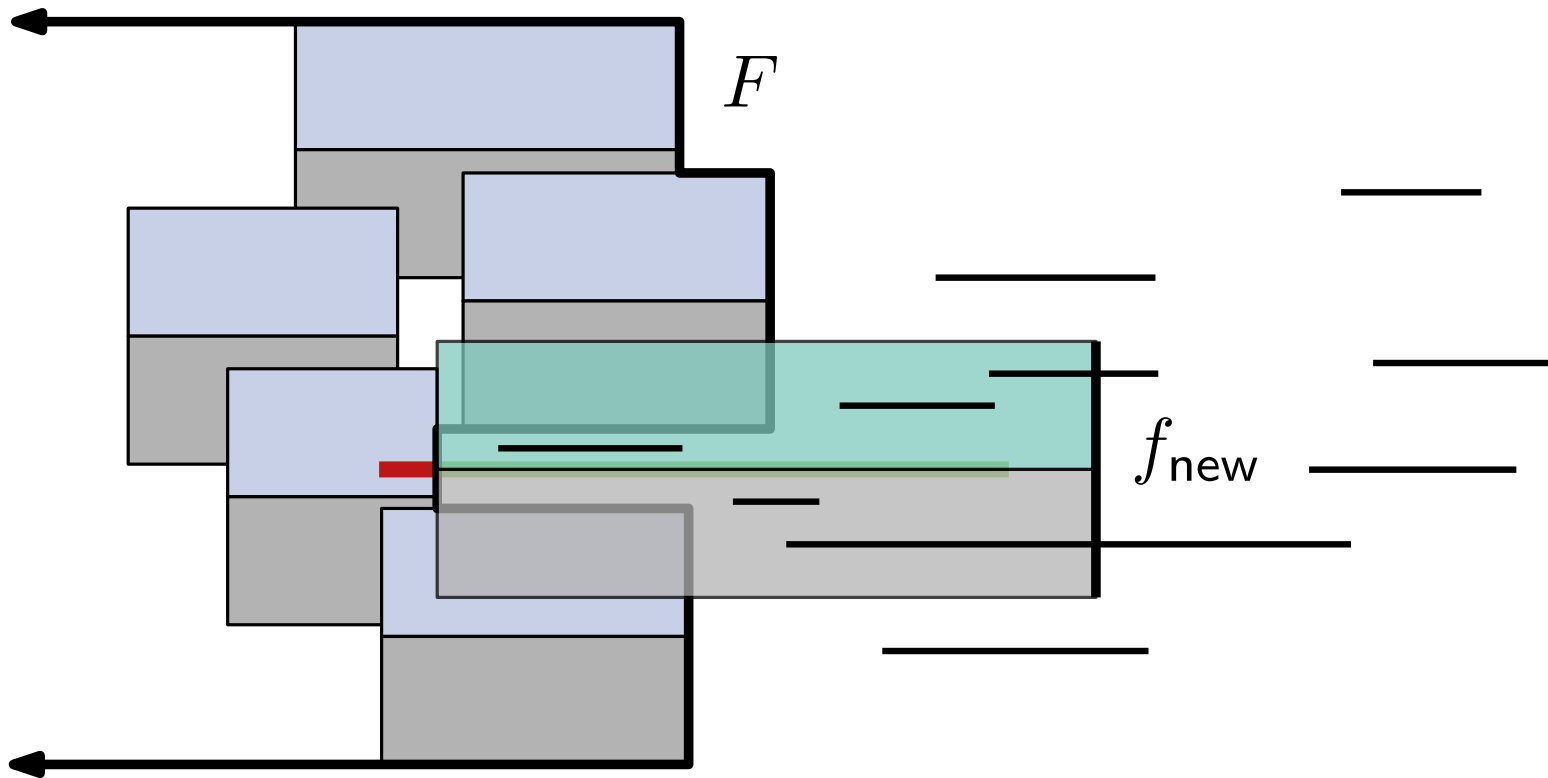
- speichere Obermenge V' mit $V_{\text{int},\text{right}} \subseteq V'$
- min-Heap $\mathcal{H}_V$ der Menge V' geordnet nach x -Koordinaten
- falls Minimum in $\mathcal{H}_V$ nicht in $V_{\text{int},\text{right}}$ liegt, verwwerfe es und wähle nächstes Minimum

Das linkeste Label für den Greedy-Algorithmus ist eines der drei Minima von $\mathcal{H}_{\text{right}}$, $\mathcal{H}_{\text{int}}$ und $\mathcal{H}_V$

Updates

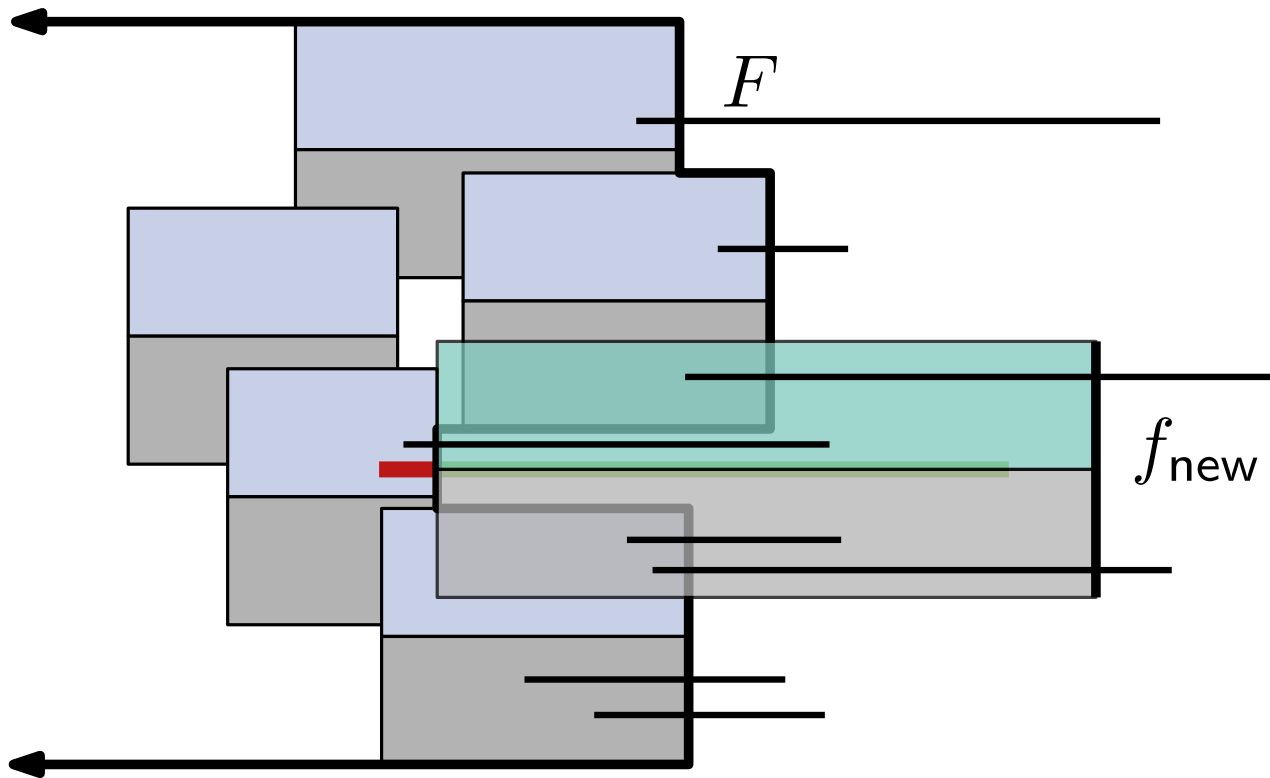


Updates



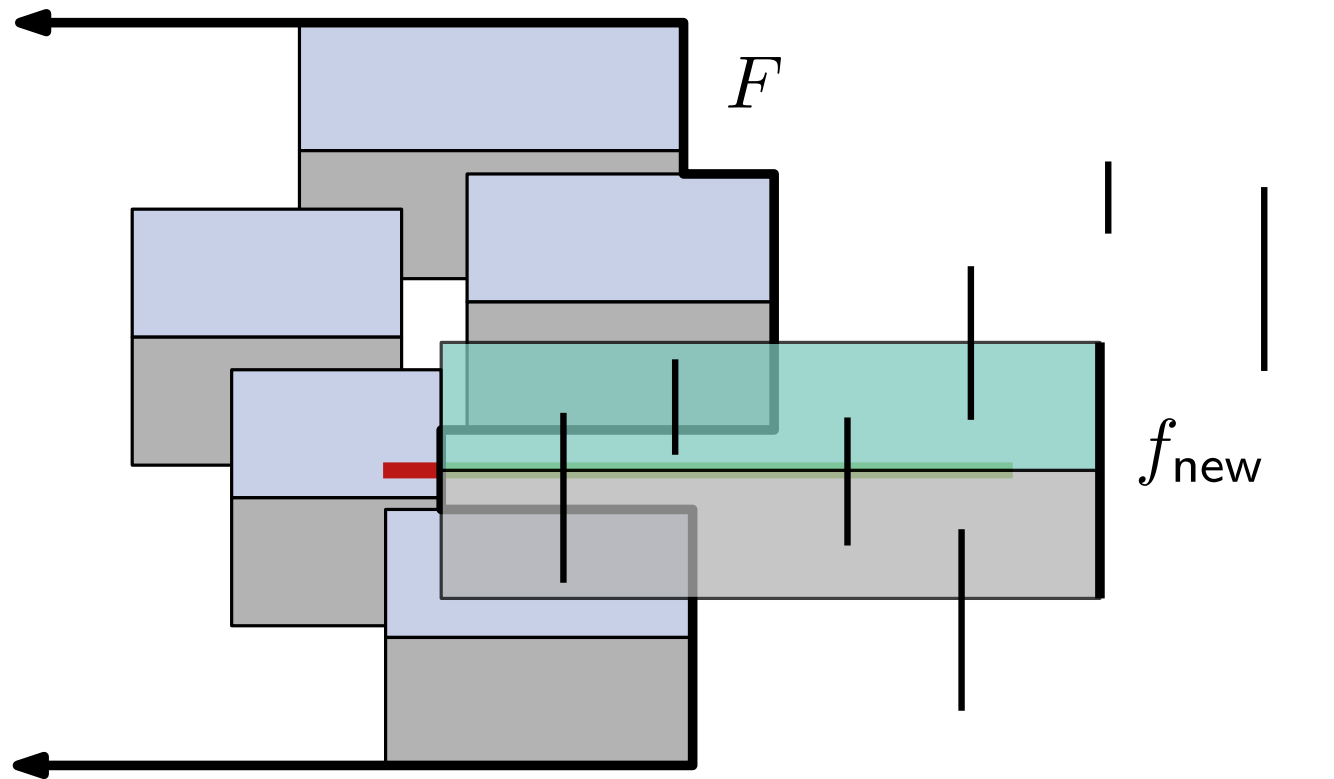
1) Updates von H_{right}

Updates



- 1) Updates von H_{right}
- 2) Updates von H_{int}

Updates



- 1) Updates von H_{right}
- 2) Updates von H_{int}
- 3) Updates von $V_{\text{int, right}}$ bzw. von V'

while $\mathcal{H}_{\text{right}}$, $\mathcal{H}_{\text{int}}$ oder $\mathcal{H}_V$ nicht leer **do**

$v \leftarrow \text{find-min}(\mathcal{H}_V)$

while v links von F **do**

 delete-min($\mathcal{H}_V$); $v \leftarrow \text{find-min}(\mathcal{H}_V)$

$l_i \leftarrow$ linkstes Label aus $\mathcal{H}_{\text{right}}$, $\mathcal{H}_{\text{int}}$ und $\mathcal{H}_V$

 füge l_i zur Beschriftung hinzu

$f_{\text{new}} \leftarrow$ rechte Kante von $\tilde{l}_i$

 update F und $\mathcal{T}_V$ mit f_{new}

 suche mit der Region links von f_{new} in $\mathcal{P}_{\text{left}}$ und $\mathcal{P}_{\text{right}}$ und update

$\mathcal{H}_{\text{right}}$, $\mathcal{P}_{\text{left}}$, $\mathcal{H}_{\text{int}}$, $\mathcal{T}_i$'s und $\mathcal{P}_{\text{right}}$ wie beschrieben

 entferne alle Verweise auf p_i aus den Datenstrukturen

while $\mathcal{H}_{\text{right}}$, $\mathcal{H}_{\text{int}}$ oder $\mathcal{H}_V$ nicht leer **do**

$v \leftarrow \text{find-min}(\mathcal{H}_V)$

while v links von F **do**

$\lfloor$ delete-min($\mathcal{H}_V$); $v \leftarrow \text{find-min}(\mathcal{H}_V)$

$l_i \leftarrow$ linkstes Label aus $\mathcal{H}_{\text{right}}$, $\mathcal{H}_{\text{int}}$ und $\mathcal{H}_V$

 füge l_i zur Beschriftung hinzu

$f_{\text{new}} \leftarrow$ rechte Kante von $\tilde{l}_i$

 update F und $\mathcal{T}_V$ mit f_{new}

 suche mit der Region links von f_{new} in $\mathcal{P}_{\text{left}}$ und $\mathcal{P}_{\text{right}}$ und update

$\mathcal{H}_{\text{right}}$, $\mathcal{P}_{\text{left}}$, $\mathcal{H}_{\text{int}}$, $\mathcal{T}_i$'s und $\mathcal{P}_{\text{right}}$ wie beschrieben

 entferne alle Verweise auf p_i aus den Datenstrukturen

Satz 2: Der Algorithmus Greedy4S(P, L) lässt sich mit $O(n \log n)$ Zeitbedarf und $O(n)$ Platzbedarf implementieren. Die Lösung enthält mindestens halb so viele Label wie die optimale Lösung.

Van Kreveld, Strijk und Wolff geben auch ein polynomielles Approximationsschema (PTAS) für das Beschriften im Slider-Modell an.

Für Approximationsgüte $(1 - \varepsilon)$ ist die Laufzeit $O(n^{4/\varepsilon^2})$.

Van Kreveld, Strijk und Wolff geben auch ein polynomielles Approximationsschema (PTAS) für das Beschriften im Slider-Modell an.

Für Approximationsgüte $(1 - \varepsilon)$ ist die Laufzeit $O(n^{4/\varepsilon^2})$.

Ähnliche Ergebnisse (NP-vollständig, Approximation, PTAS) gelten auch für fixed-position-Modelle, z.B. (Agarwal et al. '98)

Van Kreveld, Strijk und Wolff geben auch ein polynomielles Approximationsschema (PTAS) für das Beschriften im Slider-Modell an.

Für Approximationsgüte $(1 - \varepsilon)$ ist die Laufzeit $O(n^{4/\varepsilon^2})$.

Ähnliche Ergebnisse (NP-vollständig, Approximation, PTAS) gelten auch für fixed-position-Modelle, z.B. (Agarwal et al. '98)

Slider-Modelle erlauben in realen Instanzen bis zu 15% mehr platzierte Labels als ein 4-Positionen-Modell.