Kap. 3: Hierarchische Zeichenverfahren

3.3 Zählen von Kreuzungen ff

Prof. Dr. Petra Mutzel

Lehrstuhl für

Algorithm Engineering LS11

Universität Dortmund

7. VO WS07/08 5. November 2007

Literatur für diese VO

- W. Barth, M. Jünger und P. Mutzel: Simple and Efficient 2-Layer Cross Counting, Journal of Graph Algorithms and Applications (JGAA), 2003

- H. Nagamochi und N. Yamada: Counting Edge Crossings in a 2-Layered Drawing. Information Processing Letters 91, 221-225, 2004

Überblick: Zählen von Kreuzungen

- Einführung

- Algorithmen
  - Sweepline Algorithmus
  - BJM-Algorithmus und Variationen

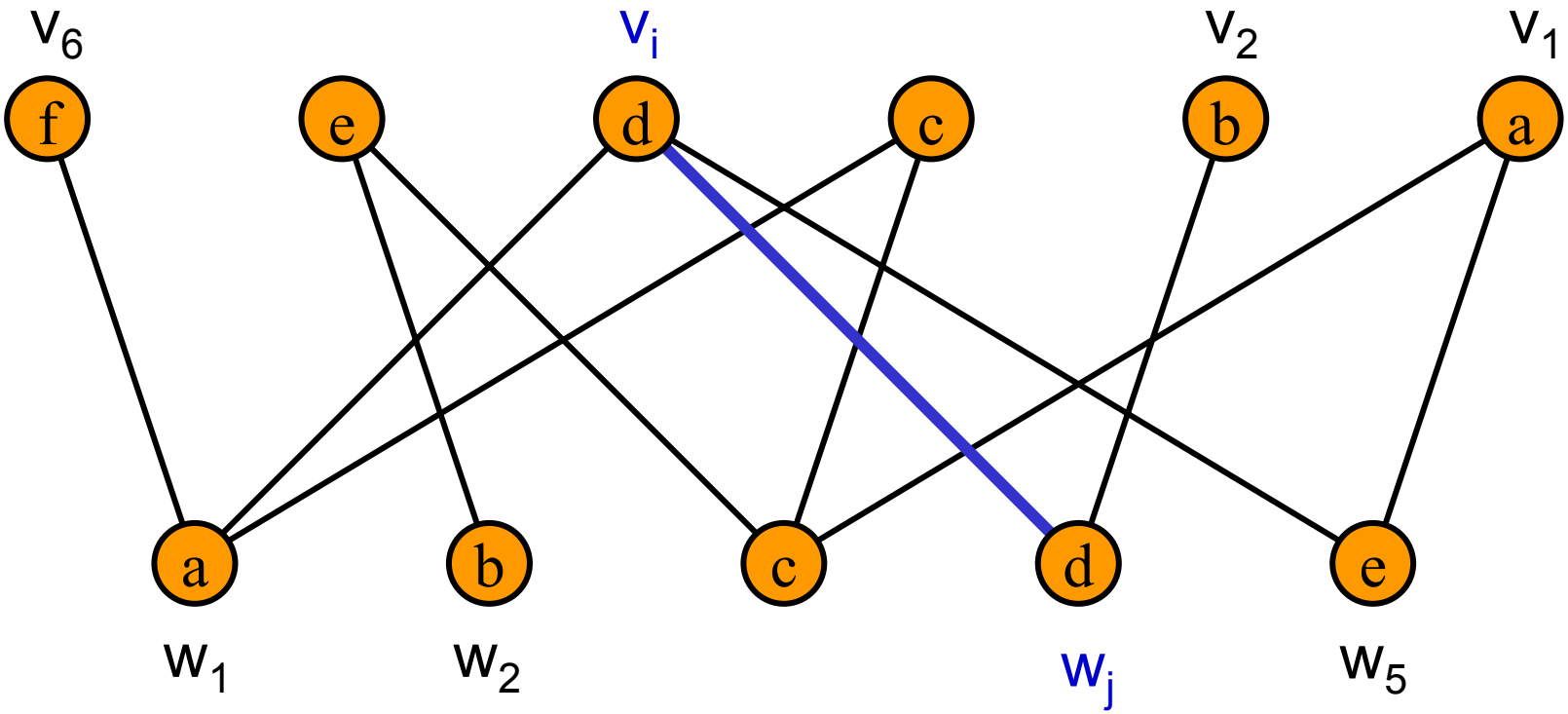
- Experimentelle Resultate

- Algorithmen von Nagamochi & Yamada NEU

Algorithmen von Nagamochi & Yamada

Idee: 2 Ansätze basierend auf **Dynamischer Programmierung** und **Divide & Conquer**

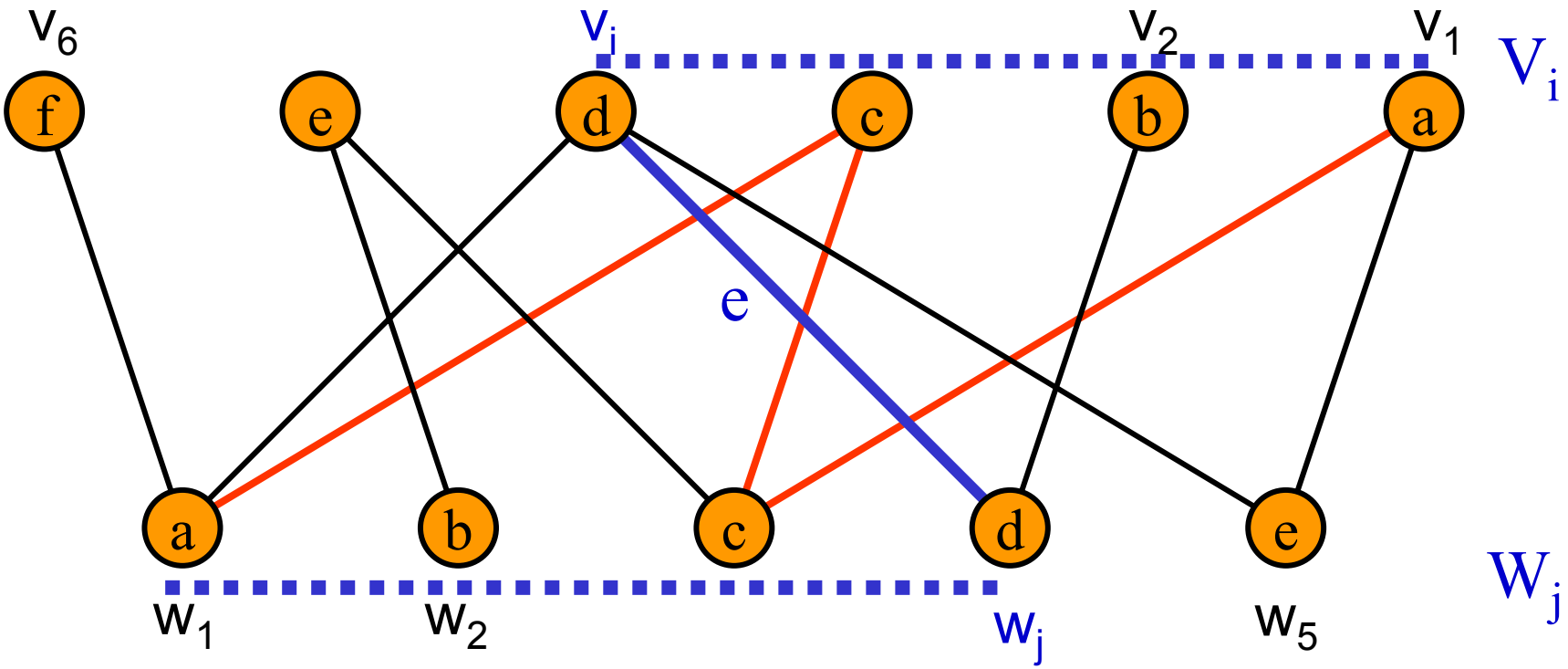
- Voraussetzungen: Seien V und W die beiden Knotenmengen
- $V = \{v_1, v_2, \ldots, v_{n1}\}$ geordnet von **rechts nach links**
- $W = \{w_1, w_2, \ldots, w_{n2}\}$ geordnet von links nach rechts



NY Dynamic Programming

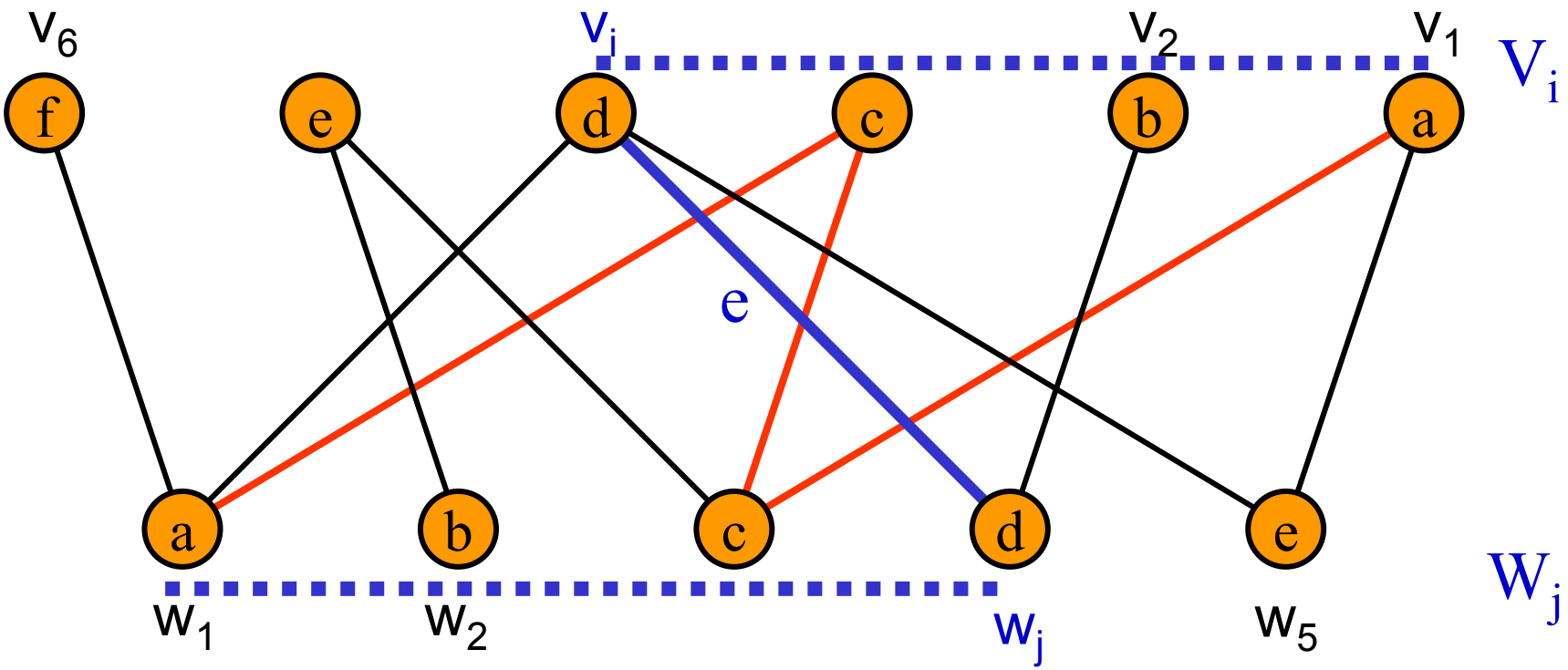
- $V_i=\{v_1,v_2,\ldots,v_i\}$ $(1\le i\le n_1=|V|)$
- $W_j=\{w_1,w_2,\ldots,w_j\}$ $(1\le j\le n_2=|W|)$
- Für jede Kante $e=(v_i,w_j)$ betrachte die Anzahl an Kreuzungen zwischen e und Kanten $e'\in E(V_{i-1},W_{j-1})$

Menge aller Kanten von V_{i-1} nach W_{j-1}



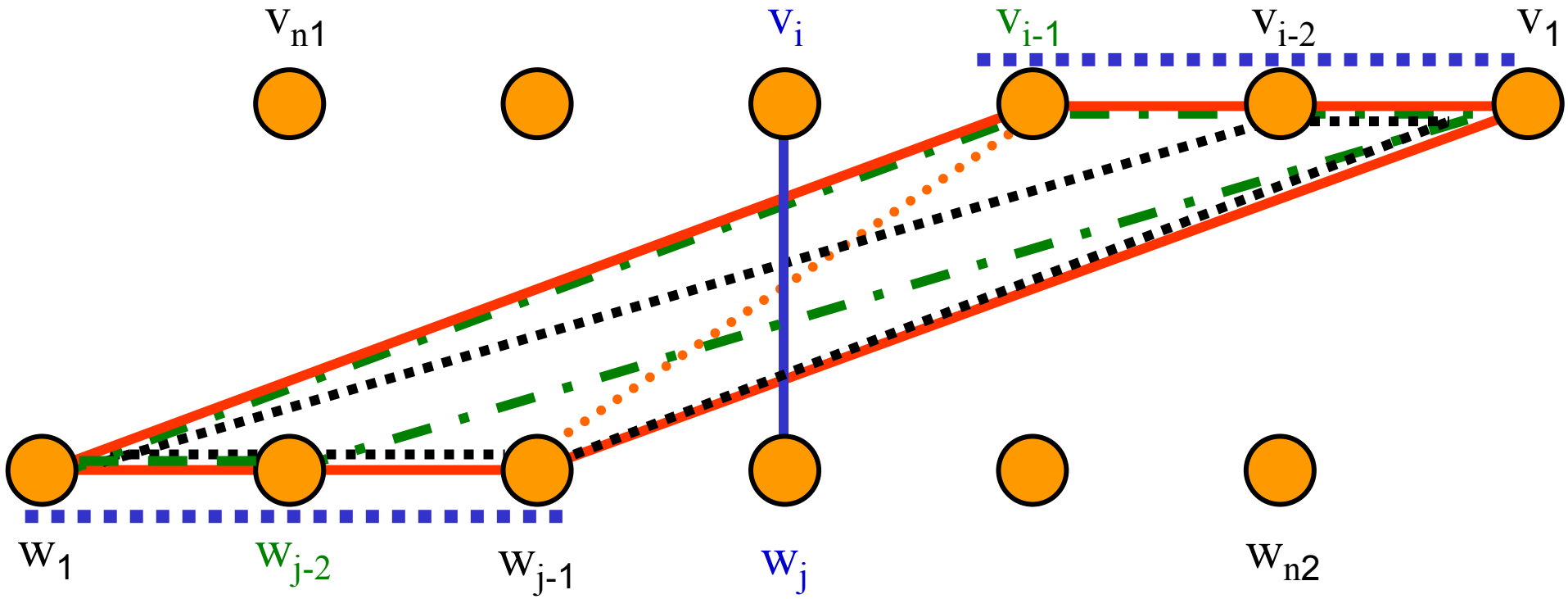
NY Dynamic Programming

- Für jede Kante $e=(v_i,w_j)$ betrachte die Anzahl an Kreuzungen zwischen e und Kanten $e' \in E(V_{i-1},W_{j-1})$
- Anzahl der Kreuzungen: $\sum_{e=(v_i,w_j)\in E} \left|E(V_{i-1},W_{j-1})\right|$



NY Dynamic Programming

- Es gilt: $$\left|E(V_{i-1},W_{j-1})\right| = \left|E(V_{i-1},W_{j-2})\right| + \left|E(V_{i-2},W_{j-1})\right| - \left|E(V_{i-2},W_{j-2})\right| + \left|E(\{v_{i-1}\},\{w_{j-1}\})\right|$$



NY Dynamic Programming

DPCOUNT

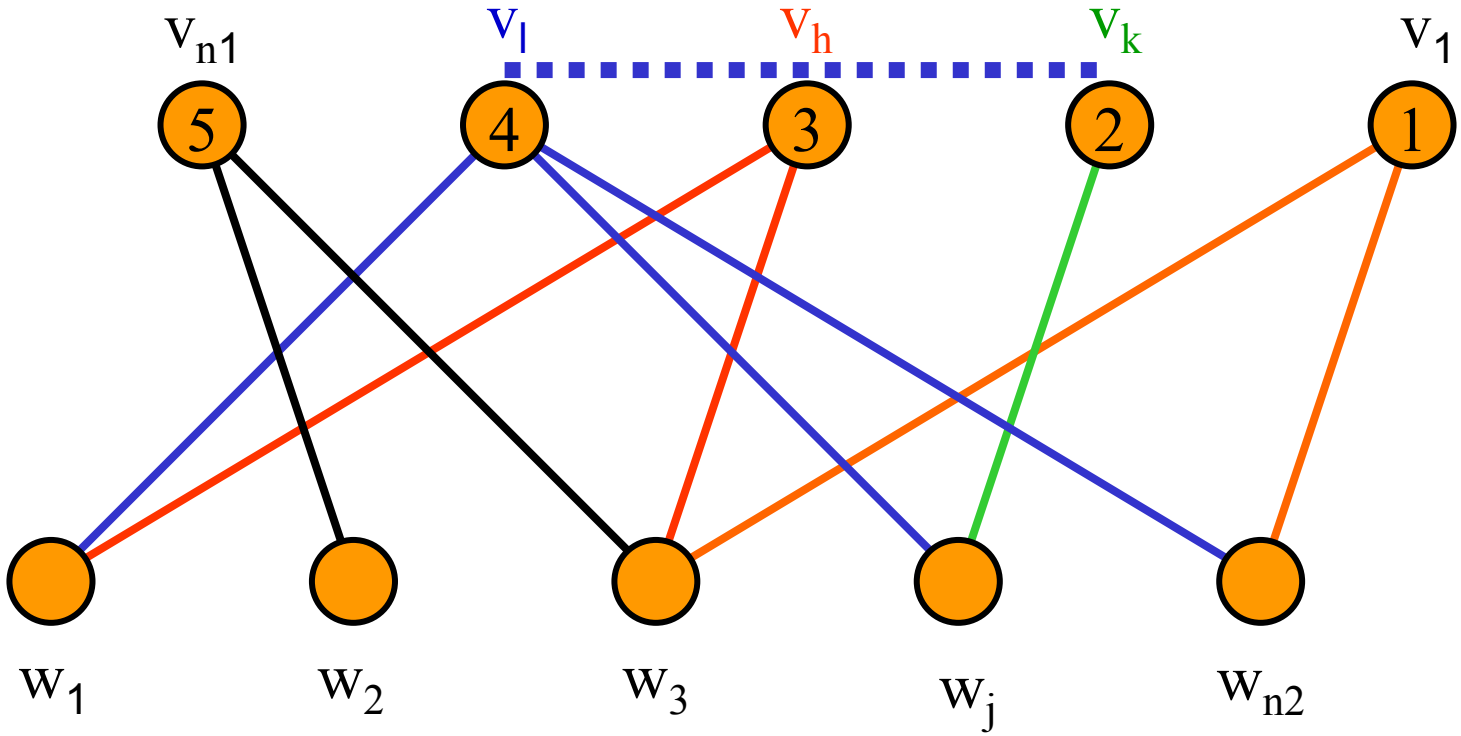
1. sum=0; initcvw←0
2. **for** i=2,3,…,n_1 **do**
3. **for** j=2,3,…,n_2 **do**
4. $c(v_{i-1},w_{j-1})=c(v_{i-1},w_{j-2})+c(v_{i-2},w_{j-1})-c(v_{i-2},w_{j-2})$
5. **falls** $(v_{i-1},w_{j-1})\in E$ **dann**
6. $c(v_{i-1},w_{j-1})=c(v_{i-1},w_{j-1})+1$
7. **falls** $(v_i,w_j)\in E$ **dann** $sum=sum+c(v_{i-1},w_{j-1})$
8. }
9. }

NY Dynamic Programming: Laufzeit

- Speicherung mit Adjazenzmatrix:
  - Zeit: $O(n_1 n_2)$
  - Platz: $O(n_1 n_2)$
- Speicherung mit Adjazenzlisten:
  - Zeit: $O(n_1 n_2)$
  - Platz: $O(m)$
    - denn wir benötigen nicht ganze Tabelle, sondern nur die letzten beiden Zeilen
    - für Abfrage $(v,w) \in E$: in $O(n_2)$ Platz mit Aufbau eines „membership“ Feldes für v_i bei jedem Aufruf der äußerer Schleife

NY Divide & Conquer

- Sei $n_1 \leq n_2$. Für Knoten $v_k, v_l \in V$ mit $k \leq l$ sei
- $V[k,l]$ die Menge der Knoten v_i mit $k \leq i \leq l$, und
- $c[k,l]$ die Anzahl der Kreuzungen im Subgraph induziert durch $V[k,l] \cup W$.

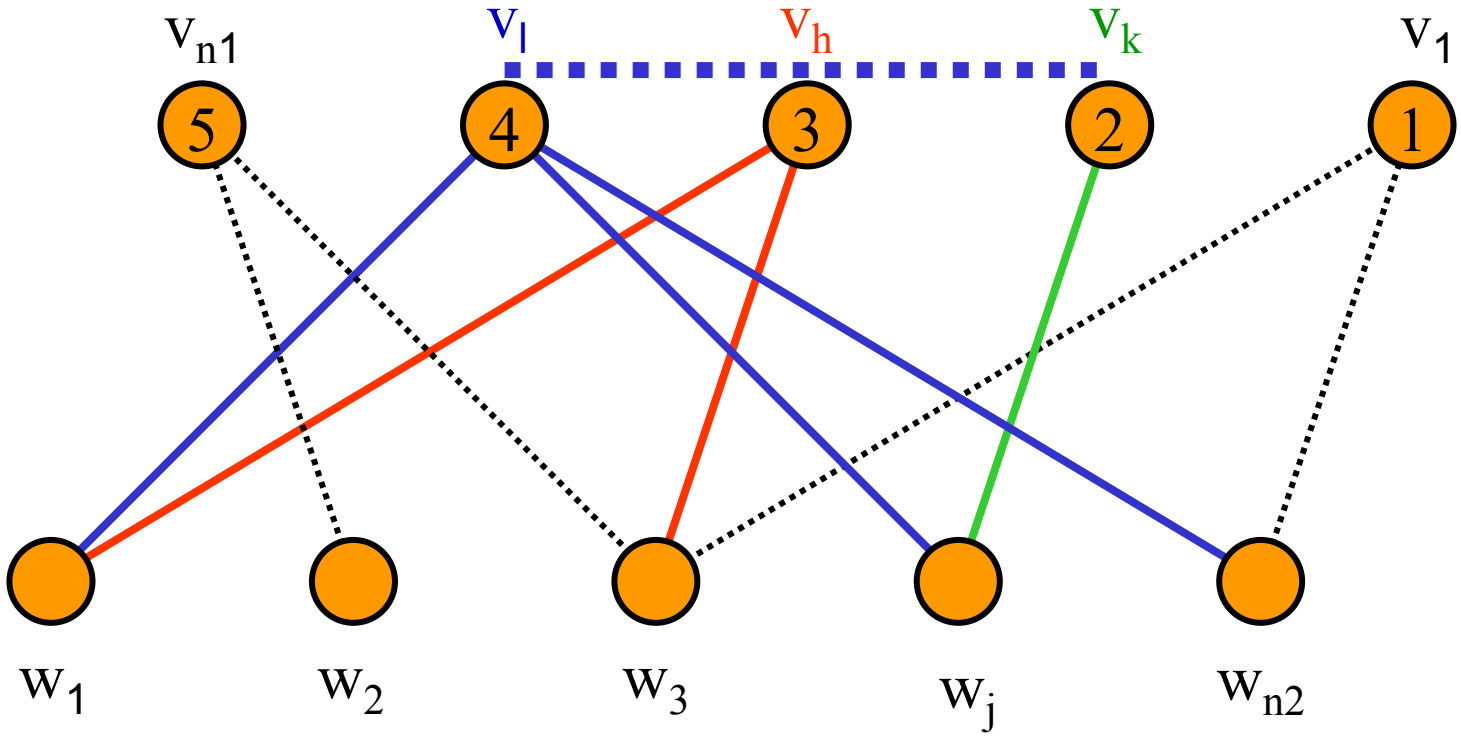


c[2,4]=5

NY Divide & Conquer

Lemma: Für k,h und l ($1 \le k \le h < l \le n_1$) ergibt sich $c[k,l]$ durch die Summe von $c[k,h]+c[h+1,l]$ und die Anzahl an Kreuzungen durch je eine Kante aus $E(V[k,h],W)$ und $E(V[h+1,l],W)$.

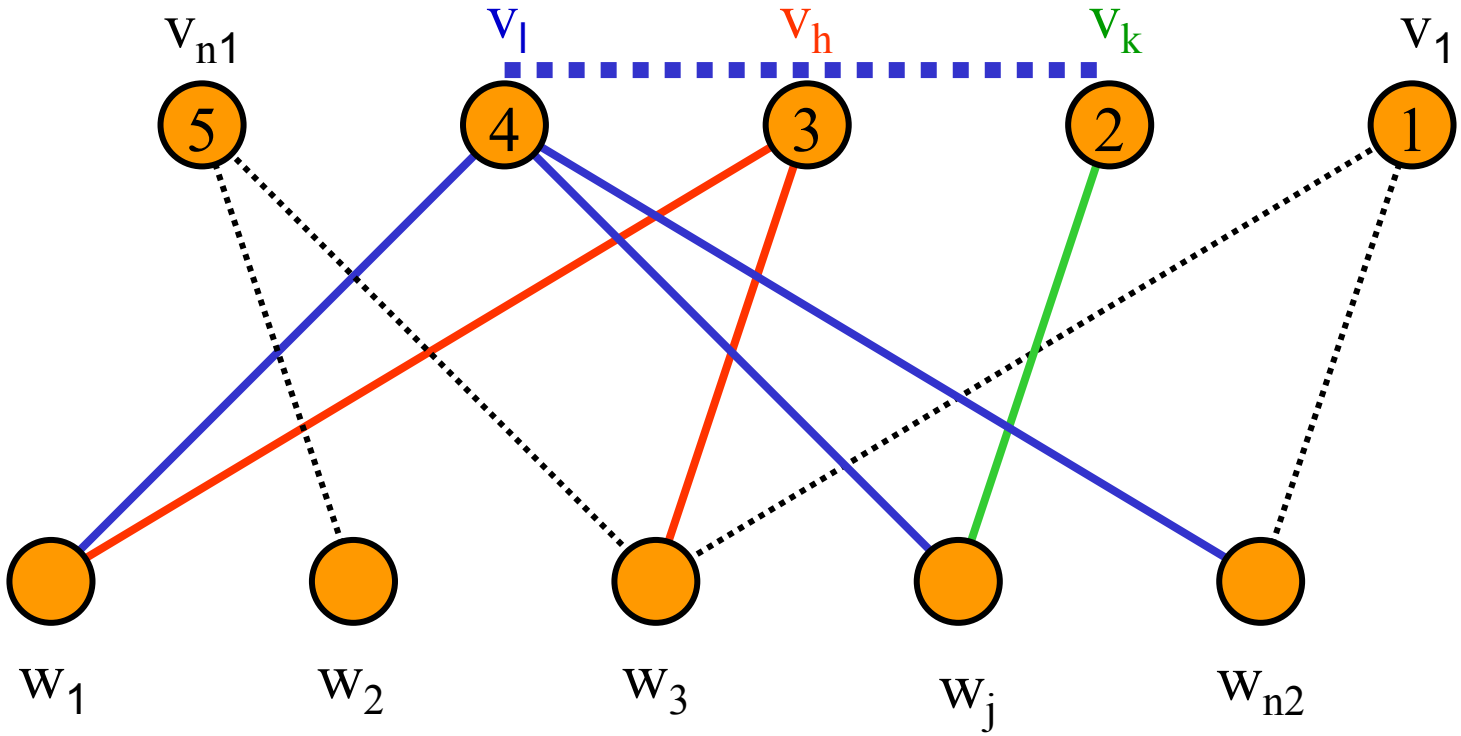
$c[2,4]=c[2,3]+c[4,4]+X=0+5=5$



NY Divide & Conquer

Idee: Anwendung des Lemmas auf $c[1,n_1]$, dann rekursiv auf $c[1,n_1/2]$, $c[n_1/2+1,n_1]$ etc.

Problem also reduziert auf: Zähle die Anzahl an Kreuzungen durch je eine Kante aus $E(V[k,h],W)$ und $E(V[h+1,l],W)$.



Lösung des Teilproblems

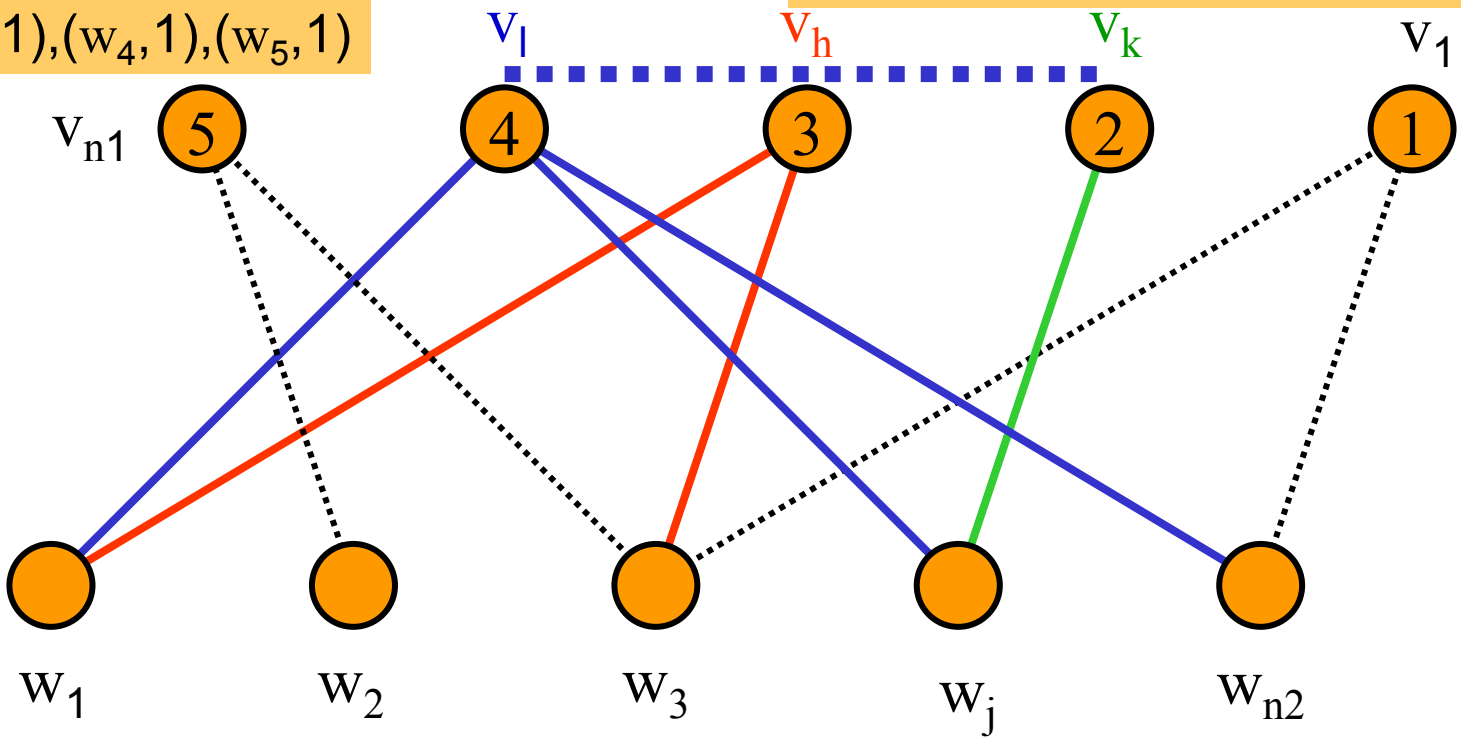
Idee: Sei $L[k,l]=(w_{i1},d_1),(w_{i2},d_2),\ldots,(w_{ip},d_p)$ mit $w_{i1},\ldots,w_{ip}$ $(i1<i2<\ldots<ip)$ sortierte Nachbarliste von $V[k,l]$ und d_j ist die Anzahl der Kanten zwischen $V[k,l]$ und w_{ij}.

Berechnung: Merge der Listen $L[k,h]$ und $L[h+1,l]$ zu $L[k,l]$

$L[2,3]=(w_1,1),(w_3,1),(w_4,1)$

$L[4,4]=(w_1,1),(w_4,1),(w_5,1)$

$L[2,4]=(w_1,2),(w_3,1),(w_4,2),(w_5,1)$



NY Divide & Conquer Algorithmus

MERGE(L[k,h],L[h+1,l],L[k,l],subsum)

INPUT: Sei L[h+1,l]=$(w_{i1},d_1),(w_{i2},d_2),\ldots,(w_{ip},d_p)$ und

sei L[k,h]=$(w_{i'1},d'_1),(w_{i'2},d'_2),\ldots,(w_{i'q},d'_q)$

OUTPUT: L[k,l]=$(w_{i''1},d''_1),(w_{i''2},d''_2),\ldots,(w_{i''t},d''t)$

- subsum=0; accum=summe aller d_i aus L[h+1,l]
- **for** j=1,2,…,t **do {**
- $d''_j=0$

1. **falls** $w_{i''j}$ ist gleich einem $w_{j'a}\in$ L[k,h] **dann** {

2. subsum = subsum+accum •d'_a

3. $d''_j = d''_j + d'_a$ }

4. **falls** $w_{i''j}$ ist gleich einem $w_{jb}\in$ L[h+1,l] **dann** {

5. accum=accum-d_b

6. $d''_j = d''_j + d_b$ }

7. **}**

NY Divide & Conquer Algorithmus

Sei L_i=L[i,i]

DCCOUNT(sum)

- sum=0

1. **for** t=0,1,2,…,$\lceil \log n_1 \rceil$-1 **do**

2. **for** j=0,1,2,…, $\lfloor (n_1-2^t-1)/2^{t+1} \rfloor$ **do**

3. // j Läufe bis max. j so dass $(2j+1)2^t+1 \leq n_1$

4. MERGE (L_{2j2^t+1}, $L_{(2j+1)2^t+1}$, L_{2j2^t+1},subsum)

5. sum+=subsum

6. }

7. }

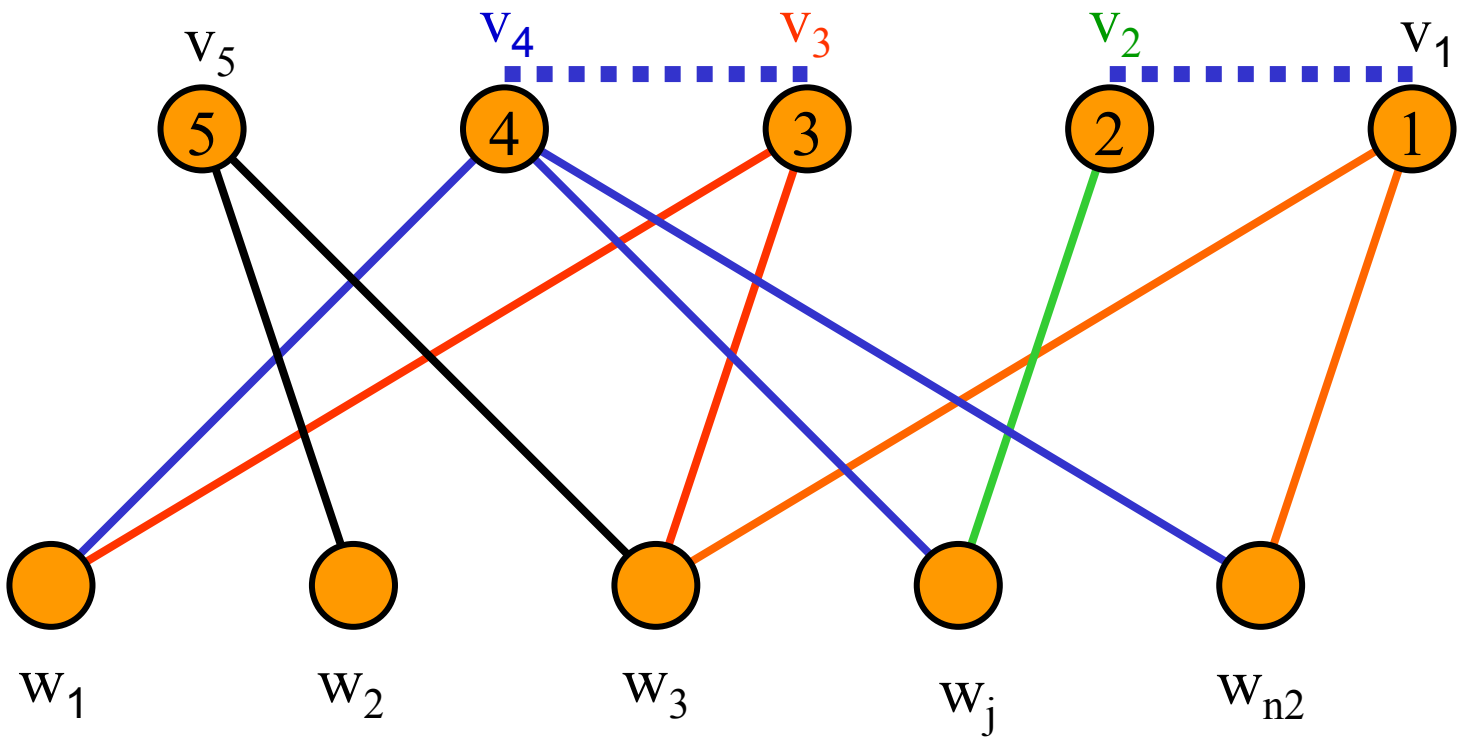
Beispiel

t=0,1,2:

t=0, j=0,1: $L_1 \cup L_2$ und $L_3 \cup L_4$ → 1+4 Kreuzungen

t=1, j=0: $L_1 \cup L_3$ → + 3 Kreuzungen

t=2, j=0: $L_1 \cup L_5$ → + 4 Kreuzungen



Analyse NY Divide & Conquer

Theorem: DCCOUNT berechnet die Anzahl an Kreuzungen in Zeit $O(\min\{n_1 n_2, m \log(\min\{n_1 n_2\})\})$ und $O(m)$ Platz.

Beweis Korrektheit: Für $t=0$: $L_i = L[i,i]$ für alle i

Induktion nach t: Mergen der Listen

$L_{2j2^t+1} = L[2j2^t+1,(2j+1)2^t]$ und

$L_{(2j+1)2^t+1} = L[(2j+1)2^t+1, \min\{n_1,(2j+2)2^t]$ zu Liste

$L[2j2^t+1, \min\{n_1,2(j+1)2^t\}]$, die als Liste

$L[j\ 2^{t+1}+1, \min\{n_1,(j+1)2^{t+1}\}]$ in Iteration $t+1$ benutzt wird.

Die letzte Liste L_h wird evtl. in Iteration t nicht gemischt, aber sie hat Index $h=j'2^{t'}+1$ $(0 \le j' \le \lfloor (n_1-2^{t'}-1)/2^{t'+1} \rfloor)$ in jeder folgenden Iteration für t' bis es gemischt wird.

Analyse NY Divide & Conquer

Weil es genau zwei Indizes von Listen in der letzten Iteration für $t=\lceil \log n_1 \rceil -1$ gibt, werden alle Listen zu $L[1,n_1]$ gemischt.

Beweis Laufzeit:

- DCCOUNT: $O(n_1)$ MERGE Aufrufe
- MERGE: kann in $O(n_2)$ implementiert werden
- MERGE zweier Listen $L[h,j]$ und $L[k,l]$ benötigt Zeit $O(|E(V[h,l]),W)|)$ Zeit,
- jede Iteration der äußeren FOR Schleife benötigt insgesamt $O(m)$ Zeit.
- Es gibt $O(\log n_1)$ Iterationen $\rightarrow$ Laufzeit $O(m \log n_1)$
- Speicherplatz $O(m)$

Übersicht aller Laufzeiten

SAN Sander [1995] $O(|E| + |C|)$

WAM Waddle & Malhotra [1999] $O(|E| \log |V|)$

INS Insertion Sort $O(|E| + |C|)$

MER Merge Sort $O(|E| \log run(p))$

BJM Neuer Algorithmus $O(|E| \log |V_{small}|)$

NY Dynamic Programming $O(|V_1||V_2|)$

NY Divide & Conquer $O(\min\{|V_1||V_2|, |E| \log |V_{small}|)$

Ende Kap. 3.3