

Zerlegung eines Graphen in 3-Zusammenhangskomponenten

Carsten Gutwenger

Vorlesung **Algorithm Engineering** WS 05/06
vom 2. und 3. Januar 2006



Universität Dortmund, FB Informatik, LS11

Motivation

Graph-Dekomposition ist ein klassisches Problem der Graphentheorie.

Aber warum *Algorithm Engineering*?

- „Implementierbarkeit“ von Algorithmen
- zahlreiche Anwendungen, insbesondere im Graph Drawing
 - Planarität, c-Planarität, Aufwärtsplanarität
 - praxisrelevant!
- Preprocessing

02./03.01.2006 — **Algorithm Engineering**: 3-Zusammenhangskomponenten

2

Literatur

W. T. Tutte, *Connectivity in graphs*, Vol. 15 of Mathematical Expositions, University of Toronto Press, 1966.

J. E. Hopcroft, R. E. Tarjan, *Dividing a graph into triconnected components*, SIAM Journal on Computing 2, 1973, pp. 135-158.

C. Gutwenger, P. Mutzel, *A linear time implementation of SPQR-trees*, in: J. Marks (ed.), Graph Drawing (GD 2000), LNCS 1984, pp. 77-90, Springer-Verlag, 2001.

02./03.01.2006 — **Algorithm Engineering**: 3-Zusammenhangskomponenten

3

k -Zusammenhang

Definition. Sei $k \in \mathbb{N}$.

$G=(V,E)$ ist k -zusammenhängend genau dann wenn

- $|V| > k$ und
- $G-X$ ist zusammenhängend für jedes $X \subseteq V$ mit $|X| < k$.

0-zshgd. $\Leftrightarrow G$ ist nicht leer.

1-zshgd. $\Leftrightarrow G$ ist zusammenhängend.

02./03.01.2006 — **Algorithm Engineering**: 3-Zusammenhangskomponenten

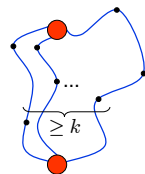
4

k -Zusammenhang (2)

Satz (Menger's Theorem).

Ein Graph ist genau dann k -zusammenhängend, wenn es zwischen jedem Paar von Knoten k unabhängige Pfade gibt.

unabhängige Pfade: keine internen Knoten gemeinsam



02./03.01.2006 — **Algorithm Engineering**: 3-Zusammenhangskomponenten

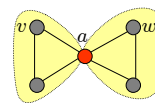
5

2-Zusammenhang

Bekannt:

a ist **Schnittknoten** (cutvertex) gdw. ex. Knoten v und w , so dass jeder Pfad von v nach w über a führt.

Block: maximaler zusammenhängender Teilgraph, der keinen Schnittknoten enthält.



02./03.01.2006 — **Algorithm Engineering**: 3-Zusammenhangskomponenten

6

Block-Graph

Definition. Seien $G=(V,E)$ und

- $C \subseteq V$ Menge der Schnittknoten von G ,
- $B \subseteq G$ Menge der Blöcke von G .

Dann ist der (bipartite) Graph

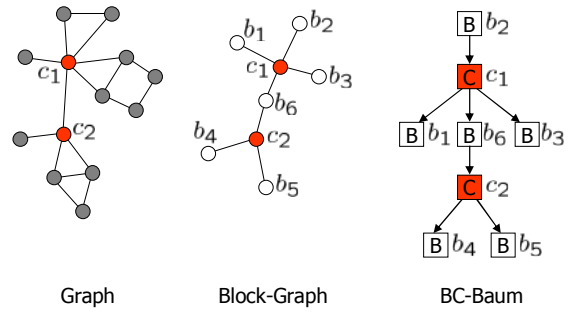
$$(C \cup B, \{(c, b) \mid c \in b\})$$

der **Block-Graph** von G .

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten

7

Block-Graph (2)



Graph

Block-Graph

BC-Baum

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten

8

Separationspaar

Sei $G=(V,E)$ 2-zusammenhängend.

$\{u,v\} \in V$ heisst **Separationspaar**, falls $G - \{u,v\}$ nicht zusammenhängend ist.

$\{u,v\}$ heisst **Splitpaar**, falls

- $\{u,v\}$ Separationspaar ist, oder
- u und v sind adjazente Knoten.

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten

9

Splitklassen

Sei $\{u,v\}$ Splitpaar.

Splitklassen von $\{u,v\}$:

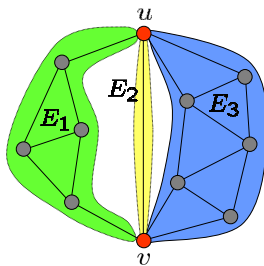
Partition E_1, \dots, E_k der Kantenmenge, so dass

$$e, f \in E_i \iff e \text{ und } f \text{ liegen auf einem Pfad, der } u \text{ und } v \text{ höchstens als einen Endpunkt enthält.}$$

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten

10

Splitklassen (2)



02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten

11

Split-Operation

Seien E_1, \dots, E_k die Splitklassen eines Splitpaares $\{u,v\}$.

Für ein j mit $1 \leq j < k$ seien

$$C = E_1 \cup \dots \cup E_j \text{ und } \bar{C} = E_{j+1} \cup \dots \cup E_k, \text{ so dass } |C| \geq 2 \text{ und } |\bar{C}| \geq 2.$$

Eine **Split-Operation** ersetzt G durch die **Split-Graphen**

$$G_1 = (V(C), C \cup e) \text{ und}$$

$$G_2 = (V(\bar{C}), \bar{C} \cup e),$$

wobei $e=(u,v)$ eine neue **virtuelle Kante** ist.

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten

12

Split-Operation (2)

$C = E_1$ $\bar{C} = E_2 \cup E_3$

G G_1 G_2

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten 13

Tutte-Split

Eine Split-Operation heißt **Tutte-Split**, falls

- $C = E_\alpha$ und
- $G(C)$ oder $G(\bar{C})$ enthält keinen Schnittknoten.

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten 14

Tutte-Split (2)

Tutte-Split ja oder nein?

Nein! Nein! Ja!

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten 15

Split-Graphen

Beobachtung. Jeder Split-Graph enthält keinen Schnittknoten und mindestens 3 Kanten.

Führe solange Split-Operationen durch wie möglich.

Lemma. Die Gesamtzahl aller Kanten in Splitgraphen ist beschränkt durch $3|E| - 6$.

Beweis. → Tafel

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten 16

3-Zusammenhangskomponenten

Definition. Wir erhalten die **3-Zusammenhangskomponenten** eines Graphen, indem wir solange Tutte-Splits durchführen wie möglich.

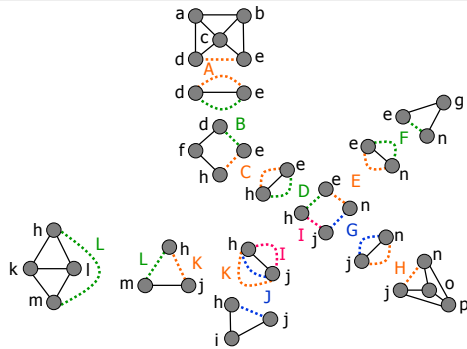
02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten 17

3-Zusammenhangskomponenten (2)

Beispiel:

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten 18

3-Zusammenhangskomponenten (3)

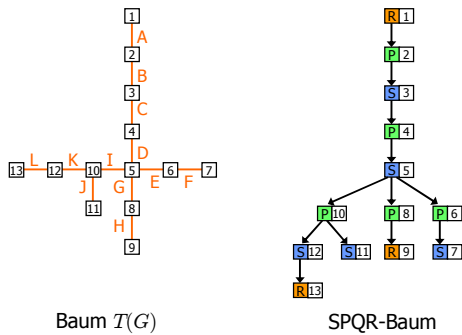


3-Zusammenhangskomponenten (4)

- Theorem.** Jede 3-Zusammenhangskomponente ist
- a) ein einfacher, 3-zshgd. Graph, 3-zshgd.
 - b) ein Kreis (Polygon) mit mind. 3 Kanten, oder seriell
 - c) ein Bündel von mind. 3 parallelen Kanten. parallel

Theorem. Die 3-Zusammenhangskomponenten eines Graphen sind eindeutig.

SPQR-Baum



Baum $T(G)$

SPQR-Baum

3-Zusammenhangskomponenten (4)

Was passiert, wenn wir immer beliebige Split-Operationen durchführen?

→ Split-Komponenten

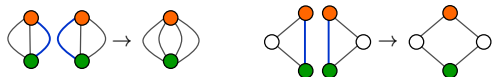
- Theorem.** Jede Split-Komponente ist
- a) ein einfacher, 3-zshgd. Graph,
 - b) ein Kreis mit **genau 3** Kanten, oder
 - c) ein Bündel von **genau 3** parallelen Kanten.

Theorem. Die Split-Komponenten eines Graphen sind **nicht** eindeutig.

3-Zusammenhangskomponenten (4)

Theorem. Man erhält die 3-Zusammenhangskomponenten aus den Split-Komponenten, indem man

- Dreiecke zu maximalen Polygonen, und
- 3-er Bündel zu maximalen Bündeln verschmelzt.



Der Algorithmus

1. Ersetze Multi-Kanten durch virtuelle Kanten.
→ C_1, \dots, C_p
2. Bestimme Split-Komponenten.
→ C_{p+1}, \dots, C_k
3. Verschmelze in C_1, \dots, C_k (solange wie möglich) je zwei Polygone bzw. Bündel, die die gleiche virtuelle Kante enthalten.

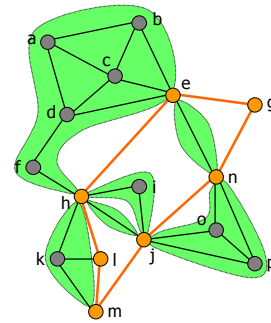
Segmente

Betrachte einen Kreis c im Graphen G .

Ein **Segment** relativ zu c ist ein Untergraph S von G , der entweder

- aus einer einzelnen Kante $e=(u,v)$ mit $u,v \in c$ und $e \notin c$, oder
- aus einer ZK K von $G \setminus c$ plus allen Kanten, die K mit c verbinden, besteht.

Segmente (2)



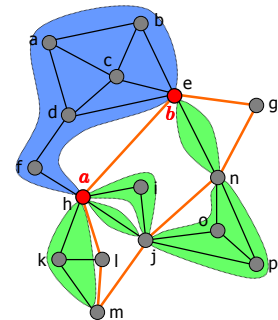
Separationspaare

Lemma. Seien S_1, \dots, S_n die Segmente relativ zu c . Ist $\{a,b\}$ ein Separationspaar von G , dann gilt:

- Die Knoten a und b liegen beide auf c , oder beide im gleichen Segment.
- Falls a und b auf c liegen, dann trifft (mind.) einer der folgenden Fälle zu:

Typ-1

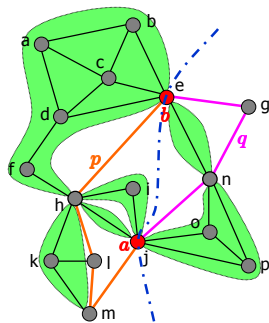
Ein Segment mit mind. 2 Kanten hat nur a und b mit c gemeinsam, und ein weiterer Knoten liegt nicht darin.



Typ-2

Seien p und q die beiden Pfade, in die c durch a und b geteilt wird.

- Kein Segment enthält ein $v \neq a, b$ in p und ein $w \neq a, b$ in q ;
- p und q enthalten beide einen Knoten $\neq a, b$.



DFS-Baum

Betrachte DFS Baum von G :

$\text{num}(v)$ DFS-Nummer von v

$v \rightarrow w$ Baumkante

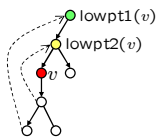
$v \leftarrow w$ Rückwärtskante

$D(v) = \{w \mid v \xrightarrow{*} w\}$ Menge der Nachfolger von v

Lowpt-Werte

$$\text{lowpt1}(v) = \min(\{\text{num}(v)\} \cup \{\text{num}(w) \mid v \overset{*}{\leftrightarrow} w\})$$

$$\text{lowpt2}(v) = \min(\{\text{num}(v)\} \cup (\{\text{num}(w) \mid v \overset{*}{\leftrightarrow} w\} \setminus \{\text{lowpt1}(v)\}))$$



Nummerierung

Aber: Wir benötigen eine spezielle Nummerierung $\text{num}(v)$:

Sei $\text{Adj}(v)$ die Adjazenzliste von v .

(P1) Die Wurzel hat Nummer 1.

(P2) Seien w_1, \dots, w_n die Kinder von v gemäß $\text{Adj}(v)$.

Dann ist

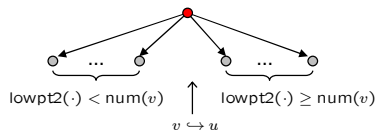
$$\text{num}(w_i) = \text{num}(v) + |D(w_{i+1}) \cup \dots \cup D(w_n)| + 1$$

Nummerierung (2)

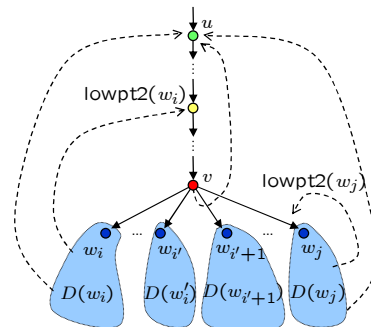
(P3) Die Kanten e in $\text{Adj}(v)$ sind aufsteigend sortiert

$$\text{nach } \begin{cases} \text{lowpt1}(w) & \text{falls } e = v \rightarrow w \\ \text{num}(w) & \text{falls } e = v \leftrightarrow w \end{cases}$$

Und: Knoten w_i, \dots, w_j mit gleichem lowpt1 -Wert u sind wie folgt gemäß lowpt2 -Wert sortiert:

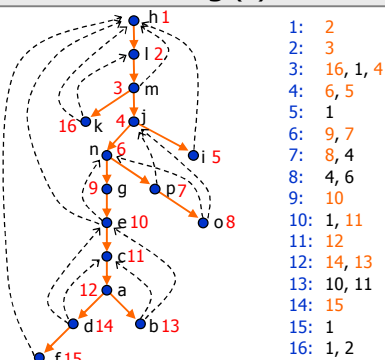


Nummerierung (3)



Nummerierung (4)

Beispiel:



Nummerierung (5)

Wie kann man diese Nummerierung effizient bestimmen?

1. Sortiere Adjazenzlisten aufsteigend gemäß

$$\Phi(e) = \begin{cases} 3\text{lowpt1}(w) & \text{if } e = v \rightarrow w \text{ and } \text{lowpt2}(w) < v \\ 3w + 1 & \text{if } e = v \leftrightarrow w \\ 3\text{lowpt1}(w) + 2 & \text{if } e = v \rightarrow w \text{ and } \text{lowpt2}(w) \geq v \end{cases}$$

→ Bucket-Sort

2. Bestimme damit Nummerierung gemäß (P1) und (P2).

3. lowpt1 und lowpt2 müssen neu berechnet werden!

Pfade

Betrachte DFS-Traversierung gemäß $\text{Adj}(v)$.

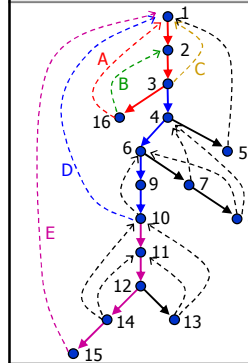
→ Menge von Pfaden der Form $v \xrightarrow{*} w \leftrightarrow u$

Jeder Pfad

- endet am Knoten mit der kleinsten möglichen Nummer;
- hat nur Anfangs- und Endknoten mit vorangehenden Pfaden gemeinsam.

→ erhalten Kreise $u \xrightarrow{*} v \xrightarrow{*} w \leftrightarrow u$

Pfade (2)



- A: $1 \rightarrow 2 \rightarrow 3 \rightarrow 16 \leftrightarrow 1$
- B: $16 \leftrightarrow 2$
- C: $3 \leftrightarrow 1$
- D: $3 \rightarrow 4 \rightarrow 6 \rightarrow 9 \rightarrow 10 \leftrightarrow 1$
- E: $10 \rightarrow 11 \rightarrow 12 \rightarrow 14 \rightarrow 15 \leftrightarrow 1$
- F: $14 \leftrightarrow 10$
- G: $14 \leftrightarrow 11$
- H: $12 \rightarrow 13 \leftrightarrow 10$
- I: $13 \leftrightarrow 11$
- J: $10 \leftrightarrow 6$
- K: $6 \rightarrow 7 \rightarrow 8 \leftrightarrow 4$
- L: $8 \leftrightarrow 6$
- M: $7 \leftrightarrow 4$
- N: $4 \rightarrow 5 \leftrightarrow 1$

Separationspaare: Typ-1

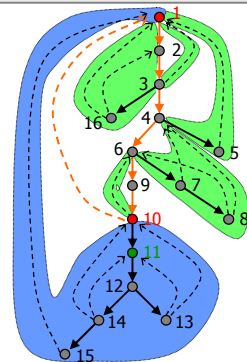
$\{a,b\}$ mit $\text{num}(a) < \text{num}(b)$ ist Separationspaar, falls

Typ-1:

Ex. $r \neq a, b$ und $s \neq a, b$ mit

- $b \rightarrow r$
- $\text{lowpt1}(r) = \text{num}(a)$
- $\text{lowpt2}(r) \geq \text{num}(b)$
- $s \notin D(r)$

Separationspaare: Typ-1 (2)



- $a = 1$
- $b = 10$
- $r = 11$
- $s = 2$
- $\text{lowpt1}(r) = 1$
- $\text{lowpt2}(r) = 10 \geq \text{num}(b)$

Separationspaare: Typ-2

$\{a,b\}$ mit $a \neq 1$ und $\text{num}(a) < \text{num}(b)$ ist Separationspaar, falls

Typ-2:

Ex. $r \neq b$ mit

- $a \rightarrow r \xrightarrow{*} b$
- b ist ein „erster“ Nachfahre von r
- Für alle $x \leftrightarrow y$ mit $\text{num}(r) \leq \text{num}(x) < \text{num}(b)$ gilt: $y \geq a$
- Für alle $x \leftrightarrow y$ mit $\text{num}(a) < \text{num}(y) < \text{num}(b)$ und $b \rightarrow w \xrightarrow{*} x$ gilt: $\text{lowpt1}(w) \geq \text{num}(a)$

Separationspaare: Typ-2

Intuitiv:

- Löschen von a und b trennt die Knoten $\text{parent}(a)$ und r voneinander.
- Die Bedingungen für Rückwärtskanten bedeuten:
 - Es darf keine Rückwärtskante geben, die aus dem Bereich zwischen r und x über a hinausläuft, denn diese würde $\text{parent}(a)$ mit r verbinden.
 - Gibt es ein Kind w von b , aus dessen Teilbaum eine Rückwärtskante in den Bereich zwischen r und x läuft, dann ist der Teilbaum an w auch nach dem Löschen von a und b mit r verbunden. Daher muss ausgeschlossen werden, dass eine Rückwärtskante aus diesem Teilbaum in den Bereich über a führt, also wird $\text{lowpt1}(w) \geq a$ gefordert.

Separationspaare: Typ-2 (2)

$a = 4$
 $b = 10$
 $r = 6$

relevante Rückwärtskanten:
 $8 \leftarrow 6$
 $8 \leftarrow 4$
 $7 \leftarrow 4$

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten 43

Separationspaare

Typ-1:
 $\{1,4\}, \{1,6\}, \{4,6\}$
 $\{1,10\}, \{1,3\}$

Typ-2:
 $\{6,10\}, \{4,10\}, \{10,15\}$
 $\{10,14\}$

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten 44

Test auf Separationspaar

Typ-1:
→ Test einer einfachen Bedingung für jede Baumkante.

Typ-2:
→ Einfach, falls $a \rightarrow v \rightarrow b$ und $\text{deg}(v)=2$.
Sonst: Benutze TSTACK

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten 45

Test auf Separationspaar (2)

TSTACK

- Enthält Tripel (h, a, b)
 - h ist höchste Nummer eines Knotens in zugehöriger Split-Komponente
 - $\{a, b\}$ ist potentielles Typ-2 Separationspaar
- $(h_1, a_1, b_1), \dots, (h_k, a_k, b_k)$
→ $\text{num}(a_k) \leq \text{num}(a_{k-1}) \leq \dots \leq \text{num}(a_1) \leq \text{num}(v_i) \leq \text{num}(b_1) \leq \dots \leq \text{num}(b_k)$
- Alle a_i, b_i sind auf aktuellem Kreis

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten 46

Test auf Separationspaar (3)

Nach Bearbeitung von $v_i \rightarrow v_{i+1}$

- Sei (h, a, b) oberstes Element auf TSTACK.
Dann ist $\{a, b\}$ Typ-2 Separationspaar, falls
 - $a = v_i$
 - $\text{num}(v_i) \neq 1$
 - $a \neq \text{parent}(b)$
- Eigenschaften bzgl. Rückwärtskanten werden durch Update von TSTACK sichergestellt!

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten 47

Test auf Separationspaar (4)

Multi-Kanten:

- Können beim Berechnen der Split-Komponenten auftreten.
- Werden beim Erzeugen einer virtuellen Kante erkannt.
- Wichtig: Ordnung der Adjazenzlisten!

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten 48

Split-Komponenten

Berechnung der Split-Komponenten:

- Verwalte aktuellen Graphen und DFS-Baum.
→ Update beim Abspalten von Split-Komponenten.
- Abspalten = Löschen einiger Kanten und ersetzen durch neue virtuelle Kante.
→ **ESTACK**

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten

49

Split-Komponenten (2)

Verwaltung von ESTACK:

- Bearbeitung von $v \rightarrow w$:
→ Push auf ESTACK **nach** rekursivem Aufruf für w .
- Bearbeitung von $v \leftrightarrow w$:
→ Push auf ESTACK falls keine Multi-Kante.

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten

50

Korrekturen

- Funktion ϕ zum Sortieren der Adjazenzlisten
→ Erkennen von Multi-Kanten
- Erzeugen der letzten Split-Komponente
- Update von TSTACK
- Test auf Typ-2 Separationspaare
- Verschiedene Updates: $A1(v)$, $DEGREE(v)$, $HIGHTP(v)$

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten

51

Laufzeit Aufbau SPQR-Baum

Testgraphen:

- zufällig generiert
- planar, 2-zusammenhängend, simple
- $n = 1000, \dots, 10000$
- Dichte: $m = 1.5n$, $m = 2n$
- je 25 Graphen pro Datenpunkt

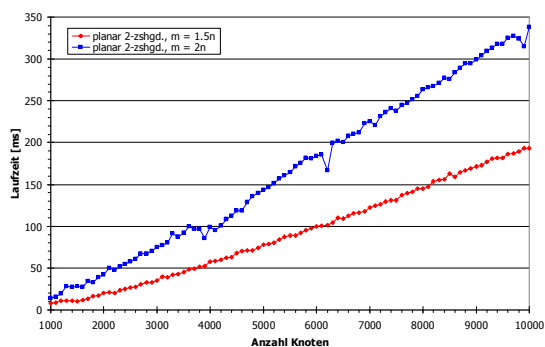
System & Implementierung:

- Pentium 4, 3.4 GHz, 1 GB
- Windows XP, MinGW g++ 3.4.4
- OGDF (*Open Graph Drawing Framework*)

02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten

52

Laufzeit Aufbau SPQR-Baum



02./03.01.2006 — Algorithm Engineering: 3-Zusammenhangskomponenten

53