Fachhochschule
Bonn-Rhein-Sieg
University
of Applied Sciences

Studiengang:	Master of Science in Computer Sciences
Fach:	Parallel Algorithms
Betreut durch:	Prof. Dr. Berrendorf
Referentin:	Dagmar Schmitz

Kantenfärbung bei Partiellen k-Trees

Edge-Coloring Partial k-Trees

1. Einleitung

1.1.Allgemeine Problemstellung

Der Regelfall der Kantenfärbung eines Graphen besteht darin, jeder an den selben Knoten angrenzenden Kante eine unterschiedliche Farbe zuzuweisen, dabei jedoch, den Graphen als Ganzes betrachtet, nur die minimal mögliche Anzahl an Farben zu verwenden. Es handelt sich demnach um ein kombinatorisches Problem.
Zu diesem Problem existieren viele unterschiedliche Lösungen, die zu einem großen Teil auf heuristischen Verfahren beruhen. Einige, wie z.B. „Simulated Annealing“[*] benutzen eine Perturbationsfunktion, die ähnlich wie ein evolutiver genetischer Algorithmus eine Verbesserung des Ergebnisses durch schrittweise zufällige Modifikationen erreicht, dabei insgesamt jedoch nach dem „trial and error“ (Versuchund Irrtum)-Prinzip vorgeht. Der Vorteil einer solchen Funktion ist es, daß sie auf die unterschiedlichsten Typen von Graphen erfolgreich angewendet werden kann.[**] Der Nachteil solcher Verfahren ist es jedoch, daß die hohe Komplexität (meist gleich dem Produkt aus Knoten- und Kantenzahl bei „einfachen“[***] Graphen) eine große Rechenzeit mit sich bringt, und ein optimales Ergebnis, d.h. die Übereinstimmung der minimal nötigen Farbenzahl mit der Zahl der tatsächlich vergebenen Farben, nicht in jedem Fall garantieren kann.
Einige Autoren gehen daher den Weg, maßgeschneiderte Lösungen für Spezialfälle von Graphen zu konstruieren. Solche Spezialfälle enthalten immer ein limitierendes Element, meist eine Beschränkung des maximalen Grades ☚ des Graphen.

---

[*] meines Wissens nach existiert keine adäquate deutsche Übersetzung dieses Fachterminus. „Annealing“ ist ein Begriff aus der Metallvearbeitung und bezeichnet das schrittweise Ausglühen eines Arbeitsstückes - „stählen“ in der figurativen Übersetzung.

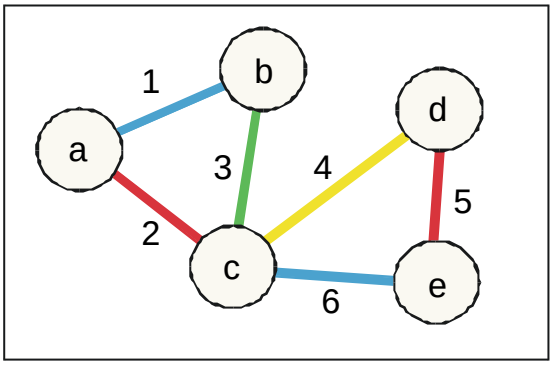
[**] ENOCHS & WAINWRIGHT 2001[2] geben 62 verschiedene Graphen unterschiedlicher Komplexität an, auf welche der von ihnen entwickelte lineare Algorithmus erfolgreich angewendet wurde. Im „worst case“, einem vollständigen Graphen mit 200 Knoten und 19900 Kanten, benötigte „Modified Simulated Anneling“ 174191 Iterationen bei 3341 sec. Laufzeit auf einem Intel Pentium III 700Mhz Prozessor.

[***] „Einfach“ (simple) bedeutet hier, daß zwischen zwei adjazenten Knoten des Graphen nur eine einzige (gerichtete) Kante existiert. Außerdem darf es keine Rückkanten („self-loops“) zum Knoten selbst geben. Das Gegenbeispiel ist ein sog. Multigraph, der beides –Rückkanten und Mehrfachkanten- enthalten kann.

1.2.Sonderfälle von Graphen

ZHOU *ET AL.*1994[1] schreiben, daß sehr viele effiziente Lösungen für kombinatorische Probleme auf sog. partiellen k-trees und seriell-parallelen Multigraphen beruhen. Der Grund hierfür liegt in der Beschränkung der gültigen Subgraphen („forbidden subgraphs“) bei diesen beiden Graphentypen.
Am Beispiel eines einfachen Graphen *G* mit 5 Knoten und 6 Kanten wird der Zusammenhang klar:

Abb.1: Kantenfärbung und Graph-Partitionierung



Tab.1: Farbtabelle

Kante	1	2	3	4	5	6
Farbe	3	1	2	4	1	3

Die eingefärbten Kanten lassen sich folgendermaßen in 4 Sets (E_1-E_4) einteilen:

$M = \{E_1 = \{1,6\}; E_2 = \{2,5\}; E_3 = \{3\}; E_4 = \{4\}\}$

Die Menge aller Kanten E in *G* ist somit in 4 Untermengen E_i von jeweils nicht-adjazenten Kanten (Kanten, die sich nicht denselben Knoten als Endpunkt teilen) aufgeteilt. Jede Kante $e \in E - E_i$ (eine Kante, die nicht in der betrachteten Untermenge enthalten ist) muß dabei an eine in E_i enthaltene Kante angrenzen. Ist diese Bedingung erfüllt, spricht man von einem „**Maximal Matching“**.
Der chromatische Index *X‘*(G), welcher die minimal mögliche Anzahl an verschiedenen Farben zur Kantenfärbung eines Graphen *G* angibt, entspricht demzufolge auch der Anzahl der Untermengen: 4 Untermengen, 4 Farben.

Auch die mit den Kanten verbundenen Knoten des Beispielgraphen können so in 4 Sets partitioniert werden:

$V_1 = \{a,b,c,e\}; V_2 = \{a,c,d,e\}; V_3 = \{b,c\}; V_4 = \{c,d\};$

Die Anzahl der Zusammenhangskomponenten in jedem durch die Partition erzeugten Subgraphen entspricht dabei der Anzahl der in ihnen enthaltenen Kanten. (Die Kanten einer Menge in dieser Partition tragen jeweils die gleiche Farbe.)

Bei dem verwendeten Beispielgraphen ist die optimale Lösung für das Kantenfärbungsproblem intuitiv ersichtlich, doch schon bei nur wenigen Kanten (und Knoten) mehr beginnen die Schwierigkeiten. Wie gezeigt liegt dem Problem der Kantenfärbung ein Partitionierungsproblem zugrunde. Und, um auf den oben angekündigten Zusammenhang zurückzukommen: Je mehr mögliche Partitionierungen eines Graphen von vorneherein ausgeschlossen werden können („forbidden subgraphs“), desto einfacher gestaltet sich der Weg zu einer optimalen Partitionierung.
Nach ZHOU *ET AL.*1994[1] existieren bereits einige effiziente Algorithmen für die genannten Sondertypen von Graphen. Ihre Charakteristika sind in der folgenden Tabelle zusammengefaßt:

Tab2: Eigenschaften von ausgewählten Kantenfärbungs-Algorithmen:

Lösungen zur Kantenfärbung bei speziellen Graphentypen							
Autor	Lösung	Typ Algorithmus	"Komplexität" Graph	Anzahl Farben	Komplexität Algorithmus	Komplexität Algorithmus für 3-trees	Anzahl Prozessoren
Diverse	vollständig	sequentiell	"simple"	Δ+1	O (m*n)	O (m*n)	1
Terada & Nishizeki	partiell	sequentiell	"series-parallel simple Graph" = partial 2-tree	Δ oder Δ+1	O (n^2)	O (n^2)	1
Zhou *et al.*	partiell	sequentiell	"series-parallel Multigraph"	Δ oder Δ+1	"linear"	"linear"	1
Bodlaender	partiell	sequentiell	"partial k-tree"	Δ oder Δ+1	O(n(Δ+1)^2 ^^2(k+1))	O($n^{25/}$)	1
Zhou *et al.*	vollständig	sequentiell	"partial k-tree"	X'(G)	"linear"	O(n)	1
Zhou *et al.*	optimal	parallel	"partial k-tree" mit "tree decomposition*"	X'(G)	logarithmisch	O(logn)	O(n/logn)**
Zhou *et al.*	optimal	parallel	*"tree decomposition" von G für treewidth(G) </= k		logarithmisch	O($\log^2$n)	O(n/logn)**
"simple" Graph: ohne Mehrfachkanten und "self-loops"; Multigraph: mit Mehrfachkanten/"self-loops" **CREW PRAM							

Im Folgenden wird die Entwicklung der optimalen Problemlösung für partielle k-Trees (partial k-Trees), zunächst mit einem seriellen, dann mit einem auf Ersterem basierenden parallelen Algorithmus nach ZHOU *ET AL.*1994[1] beschrieben.

1.3.Die Eigenschaften von partiellen k-Trees

Bei einem k-Tree ist die Breite jedes Knotens, d.h. die Anzahl der Kanten zu seinen Nachbarn, durch den Faktor k festgelegt. (Ein vollständiger Graph mit k Knoten ist somit immer ein k-Tree). Der Grad eines Graphen ist entsprechend gleich dem Maximum aller Breiten seiner Knoten. Bei einem partiellen k-Tree entspricht k der *maximalen* Breite, d.h. von einem Knoten können auch weniger als k Kanten ausgehen, auf keinen Fall jedoch mehr. Daraus folgt, daß ein partieller k-Tree ein Subgraph eines k-Trees ist, und daß die Anzahl der in ihm enthaltenen Knoten kleiner als das Produkt von k und der Anzahl der Kanten ist:

$$\xi E \xi < kn^{*}$$

Die folgenden Abbildungen zeigen Beispiele:

Abb.2: Entwicklung eines 3-Trees

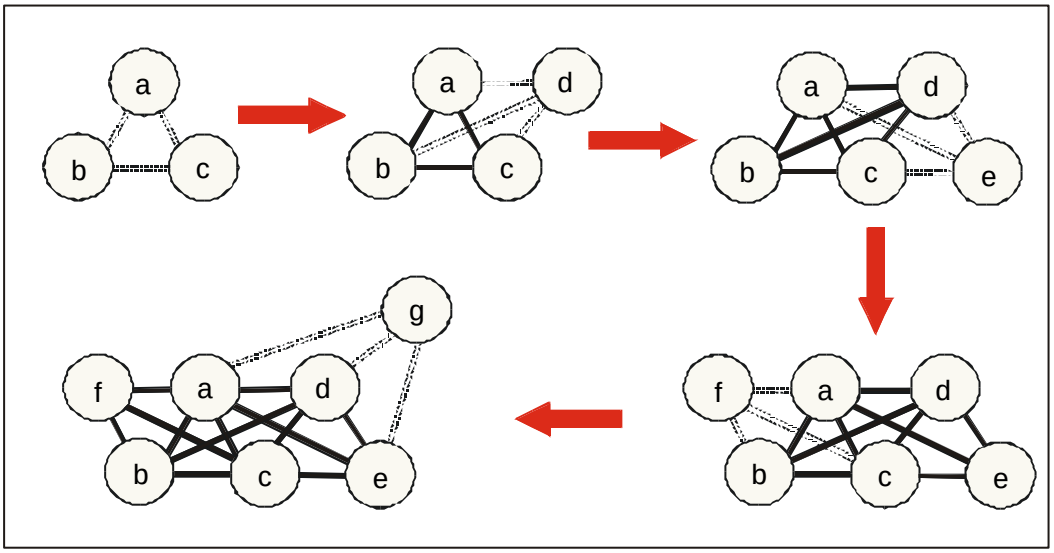
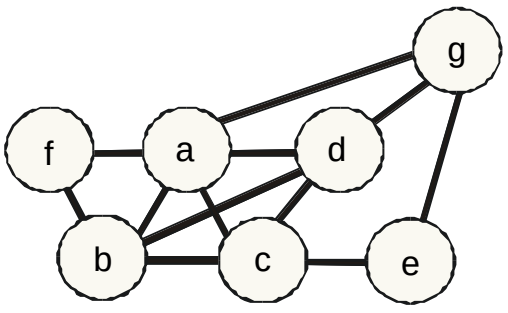


Abb.3: Ein partieller 3-Tree



* Formeln und Abkürzungen siehe Anhang

2. Hauptteil

2.1. Tree-Decomposition: Der Weg zur idealen Partitionierung eines k-Trees

Um die Partitionierung des Graphen in der oben genannten Weise durchzuführen, bedient man sich einer Baumstruktur, deren Knoten (X_1, X_2, X_3 usw.) jeweils eine Untermenge von Knoten aus G repräsentieren und in hierarchischer Beziehung zueinander stehen.
Das Ergebnis der Zerlegung eines k-Trees G = (V, E) in eine solche „Tree-Decomposition“ T = (V_T, E_T) muß für die Untermenge V_T der Knotenmenge V folgende Bedingungen erfüllen:

a) Die Vereinigung aller Knoten-Untermengen X_1-X_j ergibt wieder V

b) für jede Kante e = (v,w) χ E existiert ein Knoten X_i , welcher v und w enthält

c) wenn ein Knoten X_2 auf einem Weg in T zwischen X_1 und X_3 liegt, so ergibt die Schnittmenge von X_1 und X_3 wieder den Knoten X_2 oder ist zumindest eine Teilmenge von ihm.

Aus den Eigenschaften eines k-Trees ergeben sich einige wichtige Eigenschaften für eine Tree-Decomposition:

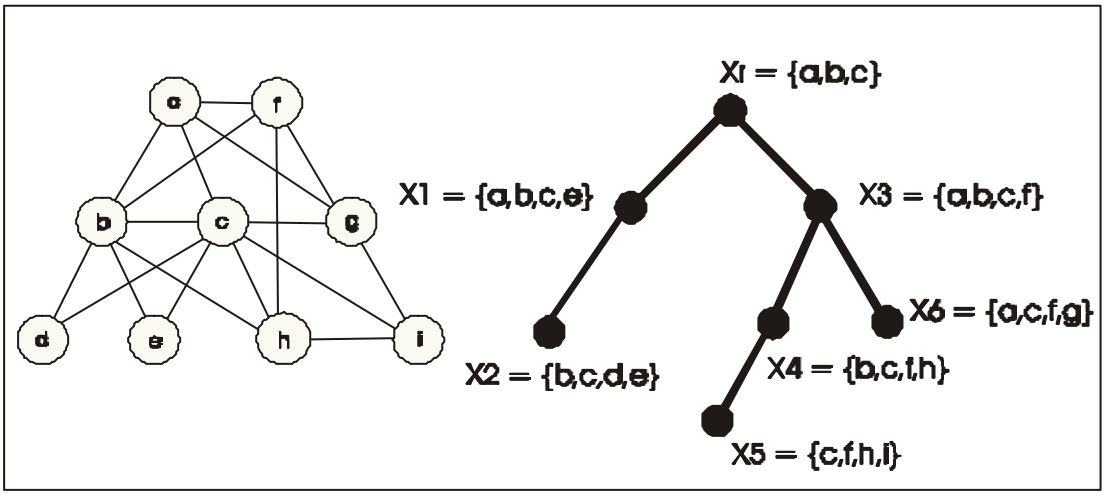
I) Die Breite einer Tree-Decomposition T = (V_T, E_T) ist maximal = $\boxtimes$ [X_i] –1.

II) Die „Baum-Breite“ (Tree-Width) eines Graphen G ist gleich der minimalen Breite seiner Tree-Decomposition, gerechnet über alle möglichen verschiedenen Dekompositionen von G.

III) Da jeder Graph G mit einer maximalen Breite von k ein partieller k-Tree ist, (Begründung s.o.) folgt, daß jede Tree-Decomposition von G eine Breite ≥ k besitzt.

Die folgende Abbildung illustriert eine solche Tree-Decomposition:

Abb.4: Tree-Decomposition eines partiellen 4-Trees:



2.2. Tree-Decomposition: Der Algorithmus von Bodlaender

Der in Tabelle Nr. 2 aufgeführte sequentielle Algorithmus von BODLAENDER1992[3] erzeugt eine Tree-Dekomposition eines Graphen für festgelegte k. Man beachte, daß k hierbei *nicht* der maximalen Breite des Graphen entspricht, sondern der maximalen Breite der gewünschten Tree-Decomposition des Graphen. Durch geeignete Wahl von k kann dieses Verfahren die Partitionierung so vornehmen, daß die Anzahl der erzeugten Untermengen (die nur nicht-adjazente Kanten enthalten dürfen) gleich dem chromatischen Index X‘ von G ist. (Dazu im folgenden Abschnitt mehr.) Hier eine kurze Skizze der Funktionsweise:

BODLAENDER:

Graph *G (V,E)*
begin
if Treewidth (*G*) > k (i.e. ξ*E*ξ k * ξ*V*ξ- ½k*(k+1)) **then**
stop
else
if Treewidth (*G*) [k **then**
begin
find a tree-decomposition of *G* with width [k;
begin
1 count the number of friendly vertices N
if N ¥ ξ*V*ξ/ ($4k^2$ + 12k + 16) **then**
2 find a maximal matching M r E in G
3 compute *G‘ (V‘,E‘)* by contraction of every edge e ? M
4 Rekursion: BODLAENDER (*G‘*)
if Treewidth (*G‘*) > k **then stop** // weil G dann auch größer als k ist
else output G‘
else 5 compute the improved graph of G*
6 Rekursion: BODLAENDER (improved G)

end

friendly vertices:
Der Knotengrad-Faktor d errechnet sich aus: d = k^2 + 4k + 4. Jeder Knoten mit einem Grad kleiner gleich d ist ein *low degree vertex*, andernfalls ein *high degree vertex*. Ein Knoten ist *friendly*, „freundlich“, wenn er selbst ein *low degree vertex* ist und an mindestens einen anderen *low degree vertex* angrenzt.

contraction:
Eine *contrction*-Operation entfernt zwei adjazente Knoten *v,w* und ersetzt sie durch einen einzigen neuen Knoten, der zu allen Knoten adjazent ist, die zuvor adjazent zu *v* und *w* waren.

* Bodlaenders Algorithmus zur Graphenverbesserung beruht auf einer Zuweisungsoperation auf den Knoten des Graphen in Stacks und Queues, auf welche dann mehrere Sortierverfahren angewendet werden. Eine detailliertere Aufführung des Verfahrens ist an dieser Stelle leider nicht möglich.

2.3. Tree-Decomposition: Der sequentielle Algorithmus von Zhou et al.

Laut ZHOU *ET AL.*1994[1] arbeitet Bodlaenders Algorithmus angewendet auf das Problem der Kantenfärbung nur dann effizient, wenn $\Delta(G)$ kleiner ist als 6k (siehe Tabelle Nr.2).
Für $\Delta \infty$ 6k geben ZHOU *ET AL.*1994[1] zunächst einen sequentiellen (*linear time*) Algorithmus an, der dann in einem weiteren Schritt in einen parallelen Algorithmus umgewandelt wird.
Die Grundidee dabei ist es, in einem solchen Fall den Graphen G (*V,E*) in mehrere Subgraphen G_1, G_2, … G_s umzuwandeln, so daß die Summe der chromatischen Indices der Subgraphen den chromatischen Index des Ursprungsgraphen ergibt:

$$X'(G) = \boxtimes^{s}_{j=1} X'(Gj) \text{ für } \Delta(G) \infty 6k$$

Für die Partition E_1, E_2, ... E_s der Kantenmenge E von G gelte folgendes:
c sei eine feste Positive Konstante. E_1, E_2, ... E_s ist eine **(Δ,c)-Partition** von E wenn für den Untergraphen Gj, erzeugt aus G[Ej] (1 j s) gilt:

a) $\Delta(G) = \boxtimes^{s}_{j=1} \Delta(Gj)$ // Der Grad von G entspricht der Summe der Grade seiner Subgraphen
b) $\Delta(G_j) = c$ für alle j, 1 j s – 1 und c $\Delta(G_s) < 2c$ // Grad (Subgraph) = c

Wählt man nun c = 3k so ergibt sich: $\Delta(Gj) < 2c = 6k$ und, durch eine Folgerung aus dem Satz von Vizing: $\boldsymbol{X'(G) = \Delta(G_j)}$
Das Theorem von *Vizing* wurde bereits im einleitenden Vortrag zu dieser Reihe vorgestellt, hier nur eine kurze Wiederholung:
Vizing´s Theorem: $X'(G) = \Delta(G)$ oder $\Delta+1(G)$*
Sprich: Der chromatische Index eines Graphen entspricht dessen maximalem Grad (evtl. +1, nach ZHOU *ET AL.*1994[1] jedoch nicht für Graphen mit $\Delta(G) > 6k$).
Somit läßt sich mit Hilfe der **(Δ,c)-Partition** der chromatische Index des Graphen G bestimmen!
Hier der auf diesem Prinzip beruhende, in linearer Zeit berechenbare sequentielle Algorithmus:

EDGE-COLOR(G)

```
begin
    if Δ(G) < 6k then
        begin
    1   find a tree-decomposition of G with width ⌈ k; // siehe BODLAENDER
    2   find an edge-coloring of G with X'(G) colors;
end
else
begin
    3   find a (Δ, 3k)-partition E1, E2, ..., E5 of E;
    4   find tree-decompositions of G1, G2, ..., Gs with widths ⌈ k;
    5   find an edge-coloring of Gj with X'(Gj) = Δ(Gj) colors for each j, 1 ⌈ j ⌈ s;
    6   extend these edge-colorings of G1, G2,..., Gs to an edge-coloring of G with
        X'(G) = Δ(G) colors;
```

* Auch: $X'(G) = \Delta$ oder $\Delta+1$

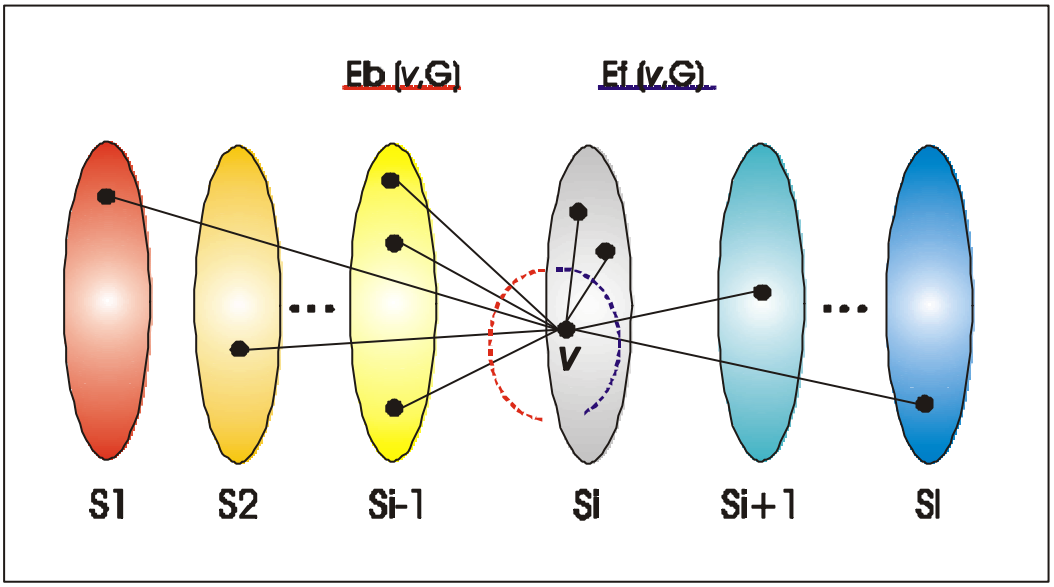
end
end.
Wie wird nun diese (ε, 3k)-Partitionierung der Kantenmenge E durchgeführt?

(ε, c)-Partition einer Kantenmenge *E* aus einer Partition (S_1, S_2, ..., S_l) von einer Knotenmenge *V*.
Die Partitionierung von *V* wird so durchgeführt, daß für jede Kante *v* aus einer beliebigen Untermenge S_i (1 i l) gilt:

backward edges (E_b): E_b (*v*, G) = {(*v*,*w*) χ *E* ξ *w* χ S_j und j < i},
foreward edges (E_f): E_f (*v*, G) = {(*v*,*w*) χ *E* ξ *w* χ S_j und j < i},
backward degree (d_b):d_b (*v*, G) =ξ E_b (*v*, G) ξ,
foreward degree (d_f): d_f (*v*, G) =ξ E_f (*v*, G) ξ.

⇨ d (*v*, G) = d_b (*v*, G) + d_f (*v*, G) ! (siehe Abb.5)

Abb.5: Partitionierung der Knotenmenge V von G



Die erzeugte Knoten-Partition S_1, S_2, ..., S_l wird (d_f, c)-Partition genannt, wenn für jeden Knoten v aus v gilt:

d_f (*v*, G) ε c, wobei c eine Konstante ist.

Durch die Wahl von c = 3k kann nun aus der (d_f, c)-Knoten-Partition eine **(ε,3k)-Kanten-Partition** hergestellt werden.

EDGE-PARTITION (G)
(Voraussetzung: G = (V, E) ist ein partieller k-Tree mit maximalem Grad ☚ ∞ 6k)

begin

```
1   find a (df, 3k)-vertex-partition S1, S2, ..., Sl of V; // siehe vorherige Seite
2   s:= [☚/3k]; // s = Anzahl der Untermengen Ei
3   for each v χ V do
4   begin
5       number the backward edges in Eb (v, G) from 1 to db (v, G);
6       let Eb (v, G) = { evqξ 1   q   db (v, G)}
7   end;
8   for each j, 1   j   s, do Ej := ❷; // Initialisiere Kanten-Sets Ej ❷ = empty
9   for each v χ V do p(v):= [db (v, G)/3k]; // p(v)   s
    // Konstruiere E1, E2, ..., Es-1
10  for each v χ V and each j, 1   j   min {p(v), s-1}, do
11  Ej := Ej 4 {ev4 χ Eb (v, G) ξ 3k (j-1) < q   3kj};
// weise Ej genau 3k backward-edges, die an Knoten v grenzen, zu, so daß db (v, G[Ej]) = 3k
//obwohl eventuell df (v, G[Ej]) > 0 und somit d (v, G[Ej]) = df (v, G[Ej]) + db (v, G[Ej]) > 3k!
12  for each v χ V with p(v)   s-2 do
13  Ep(v)+1 := Ep(v)+1 4 {ev4 χ Eb (v, G) 3kp (v) < q   db (v, G)};
    // weise Ep(v)+1 die restlichen backward edges an Knoten v zu, so daß
    //db (v,G[Ep(v)+1]) < 3k
    // passe d(v, G[Ej]) so an, daß 3k nicht überschritten werden
14  for i := l downto 1 do
15  for each v χ Si and each j, 1   j   s-1, do
    // d (v, G[Ej]) –3k   db (v, G[Ej])
16  if d (v, G[Ej]) > 3k then
17  delete d (v, G[Ej]) –3k backward edges in Eb (v, G[Ej]) from Ej
    // d (v, G[Ej]) = 3k τ ☚ (G[Ej])   3k
18  Es := Ej –E1 4 E2 4 ... 4 Es-1 // Erzeuge Es
```

end

2.4. Kantenfärbung: Der parallele Algorithmus von Zhou et al.

In ihrem Artikel „Edge Coloring Partial k-Trees“ geben ZHOU *ET AL.* 1994[1] keine konkrete Implementierung für einen parallelen Kantenfärbungs-Algorithmus an, zeigen jedoch, daß sowohl die Knoten-Partitionierung als auch der Algorithmus EDGE-PARTITION und die darauf folgende Durchführung der Tree-Decomposition für einen Graphen mit Kantenzahl n auf n/logn Prozessoren mit einer Zeitkomplexität von O(log n) für CREW-PRAM ausgeführt werden kann.

a) Kanten-Partitionierung (Vertex-Partitioning)

Da für die Anzahl l der (d_f, c)-Knoten-Partition S_1, S_2, ..., S_l aus G gilt: l = O(logn), kann die Partitionierung in O(n) Zeitkomplexität bei O(logn) paralleler Rechenzeit ausgeführt werden:

```
begin
    i := 0; // O(1)
    while V(G) ! ❷ do //V(G) not empty // O(log n)
        begin
        i := i+1;
        Si := {v χ V (G) ξd (v,G)   3k};
        V(G) := V (G) – Si ;
        end
    l := i; // O(1)
end.
```

Die Operationen innerhalb der while-Schleife können auf l = logn Prozessoren parallel ausgeführt werden.

b) EDGE-PARTITION

Betrachtet man den Algorithmus EDGE-PARTITION genauer, so stellt man fest, daß sämtliche Operationen in O(logn) oder O(1) paralleler Rechenzeit ausgeführt werden können, wobei die Gesamtkomplexität der Operationen gleich O(n) ist:

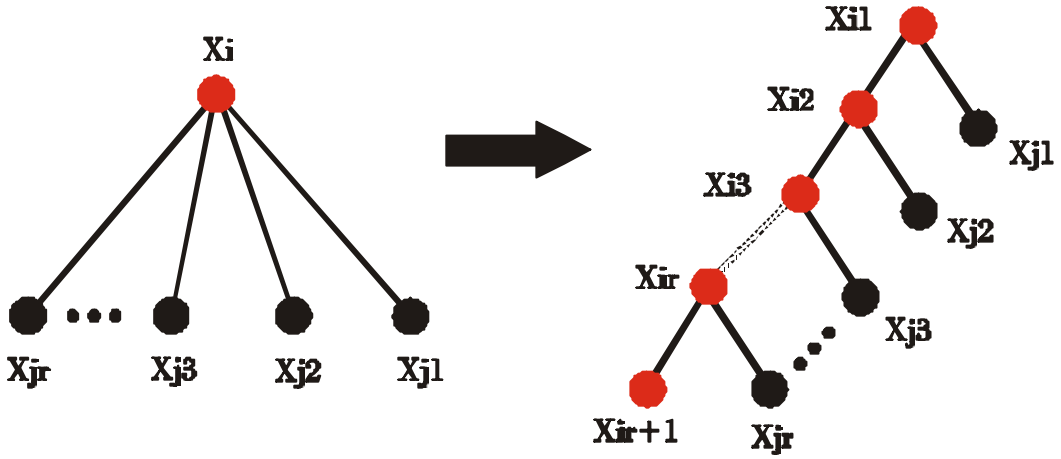
```
begin
1     find a (df, 3k)-vertex-partition S1, S2, ..., Sl of V; // O(log n) s. oben
2     s:= [❜/3k]; // O(1) lineare Zeit pro Prozessor

3     for each v χ V do // for-Schleife: O(logn)
4     begin
5           number the backward edges in Eb (v, G) from 1 to db (v, G);
6           let Eb (v, G) = { evqξ 1   q   db (v, G)}
7     end;

8     for each j, 1   j   s, do Ej := ❷; // O(1) lineare Zeit pro Prozessor


9     for each v χ V do p(v):= ⌈db (v, G)/3k⌉; // p(v)   s
      // O(1) lineare Zeit pro Prozessor

10    for each v χ V and each j, 1   j   min {p(v), s-1}, do // for-Schleife: O(logn)
11    Ej := Ej 4 {ev4 χ Eb (v, G) ξ 3k (j-1) < q   3kj};
12    for each v χ V with p(v)   s-2 do
13    Ep(v)+1 := Ep(v)+1 4 {ev4 χ Eb (v, G) 3kp (v) < q   db (v, G)};

14    for i := l downto 1 do // O(logn) da l = (logn) durch d(v, G [Ej])   6k

15    for each v χ Si and each j, 1   j   s-1, do // O(1) da d(v, G [Ej])   6k
```

16 **if** *d* (*v*, G[E_j]) > 3k then // O(1) da d(*v*, G [E_j]) 6k
17 **delete** *d* (*v*, G[E_j]) –3k backward edges in E_b (*v*, G[E_j]) from E_j // O(1) s.15/16
// *d* (*v*, G[E_j]) = 3k τ ☚ (G[E_i]) 3k
18 E_s := E_j –E_1 4 E_2 4 ... 4 E_{s-1} // O(1) lineare Zeit pro Prozessor
end

c) Tree-Decomposition

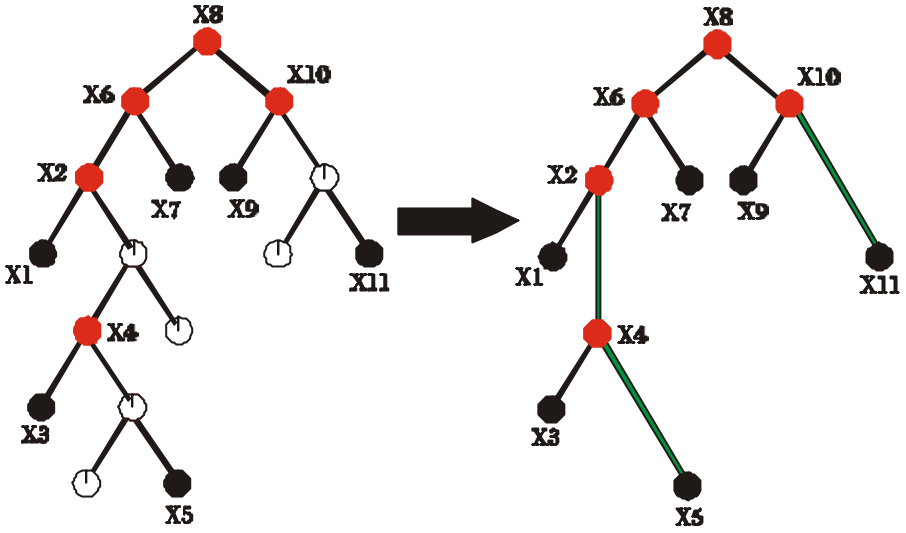
Angenommen E sei eine Partition eines partiellen k-Trees G mit Breite 6k. Aus dieser Partition wird nun die erste Untermenge der Partition als Wurzelknoten ausgewählt, und wie in der untenstehenden Abbildung als Wurzelknoten X_i eines Baumes T benutzt, wobei die restlichen Mengen die Blätter bilden.
T kann nun, wie im folgenden Schaubild zu sehen, in einen Binärbaum T_b umgewandelt werden, wozu O(n) Operationen in O(logn) paralleler Zeit benötigt werden:

Abb.6: Binärbaum-Transformation eines Graphen



X_i1 bis X_ir sind dabei Kopien des ursprünglichen Wurzelknotens X_i. Bei dieser Transformation wird die Ausgangsbreite, die ja kleiner gleich k sein muß, nicht überschritten. In einem weiteren Schritt wird der so erzeugte Graph vereinfacht: Und zwar werden alle Kopien eines Knotens X_i, die sich in der Baumstruktur auf tieferen Ebenen als X_i selbst befinden (in der folgenden Abbildung durch weiße Kreise dargestellt), gelöscht, so daß letztendlich eine Tree-Decomposition entsteht, die den in den vorherigen Kapiteln genannten Regeln entspricht.

Abb.7: Transformation eines Binärbaumes in eine Tree-Decomposition



Da die Anzahl der verwendeten ursprünglichen Knoten einer Komplexität von O(n) entspricht, können die gezeigten Transformationen in O(logn) paralleler Rechenzeit ausgeführt werden, wobei die Suche nach dem höchsten gemeinsamen Vorfahren gleichen Inhaltes ebenfalls eine Komplexität von O(logn) besitzt.

d) EDGE-COLOR

Schlußendlich muß noch gezeigt werden, daß die Komplexität des finalen Algorithmus EDGE-COLOR ebenfalls jene der bereits abgehandelten Operationen nicht übersteigt:

EDGE-COLOR(G)

begin
  if ☚(G) < 6k **then** // für den betrachteten Fall nicht relevant da ☚(G) ∞ 6k
    begin
  1 find a tree-decomposition of G with width [k; // siehe BODLAENDER
  2 find an edge-coloring of G with X‘(G) colors;
end
else begin
  3 *find a (☚, 3k)-partition $E_1, E_2, ..., E_5$ of E;* // bereits gegeben, also O(1)
  4 find tree-decompositions of $G_1, G_2, ..., G_s$ with widths [k;
  // bereits in Abschnitt c) gezeigt: O(logn)
  5 find an edge-coloring of G_j with X‘(G_j) = ☚(G_j) colors for each j, 1 [j [s;
  // da ☚(G_i) [6k sein muß, besitzt diese Operation für n_j Knoten eine
  // Komplexität von O(logn)
  6 extend these edge-colorings of $G_1, G_2,..., G_s$ to an edge-coloring of G with X‘(G) = ☚(G) colors;
  // da ☚(G_i) = ☒ (☚(G_i) [6k) ist, besitzt diese Operation für n Knoten ebenfalls
  // eine Komplexität von O(logn)
end

end.
Zusammenfassend läßt sich somit feststellen:

„Das Problem der Kantenfärbung eines k-Trees G kann in O(logn) paralleler Rechenzeit auf O(n/logn) Prozessoren bei gegebener Tree-Decomposition von G (mit einer Breite ≤ k) gelöst werden.“

3. Zusammenfassung

Die gezeigten Algorithmen von BODLAENDER 1992 und ZHOU *ET AL.* 1994 unterscheiden sich stark hinsichtlich ihrer Anwendbarkeit. Bodlaenders Algorithmus löst Partitionierungs, und damit verbunden auch Kantenfärbungs-Probleme bei einer Vielzahl von unterschiedlichen Graphentypen. Bei den Algorithmen Zhou et al. handelt es sich um maßgeschneiderte Lösungen für einen Spezialfall der Kantenfärbung, welche dabei in optimaler Rechenzeit sowohl sequenziell als auch parallel berechnet werden können. Eine weite Verbreitung dieser Lösungen wird jedoch dadurch garantiert, daß die Eigenschaften von k-Trees auf sehr viele Graphen zutreffen, die im „time-scheduling“, der Planung von Arbeitsabläufen in der Industrie, Verwendung finden. Bei sehr komplexen Arbeitsabläufen, z.B. aus der Verfahrenstechnik der chemischen Industrie, kann hier die Verwendung von parallel berechenbaren Algorithmen von entscheidender Bedeutung sein.

4. Literatur

BODLAENDER, H.L. „A linear time algorithm for finding tree-decompositions of small treewidth“ 1992, Archiv der Utrecht University, Department of Computer Science, Utrecht.

ENOCHS, B.P, WAINWRIGHT, R.L. „“Forging“ optimal solutions to the edge-coloring problem“ 2001, Proceedings of GECCO 2001 San Francisco, CA, Morgan Kauffmann, pp. 313-319.

ZHOU, X., NAKANO, S.-I., NISHIZEKI, T. „Edge-coloring partial k-Trees“ 1996 Journal of Algorithms 21, 598-617 (1996) Article No. 0061

Abkürzungen und Zeichen	
Kürzel	**Bedeutung**
^	"hoch" (folgendes Zeichen ist Exponent)
^^	"hoch" (folgendes Zeichen ist Exponent eines Exponenten)
c	Konstante
X'(G)	chromatischer Index von G
Δ	maximaler Grad
Δ (G)	*maximum degree* (maximaler Grad) von G
d (*V*)	*degree* (Grad) eines Knotens *v*
d (*V*,G)	*degree* (Grad) eines Knotens *v*
e	Kante
E(G)	Kantenmenge von G
E'(G)	Teilmenge von *E(G)*
E_T	Kantenmenge von T
G	Graph
G[*E'*]	Subgraph (Teilgraph) von G aus den Kanten von *E'*
i, ,j	ganzzahlige Werte
k	begrenzender Faktor für die "treewidth" eines Graphen
k-Tree	Graph mit einer durch k begrenzten (bounded) "tree-width" = vollständiger Graph mit k Knoten
m	Anzahl Kanten
n	Anzahl Knoten
partial k-Tree	Subgraph (Teilgraph) eines k-Trees
T	tree (Baum) aus Graph G
u,v	Knoten in G
X_i, X_j, X_l, X_1, ...	Knoten in einer Tree-Decomposition von G
V(G)	Knotenmenge von G
V'(G)	Teilmenge von *V(G)*
V_T	Knotenmenge von T