Parameterisierte Algorithmen

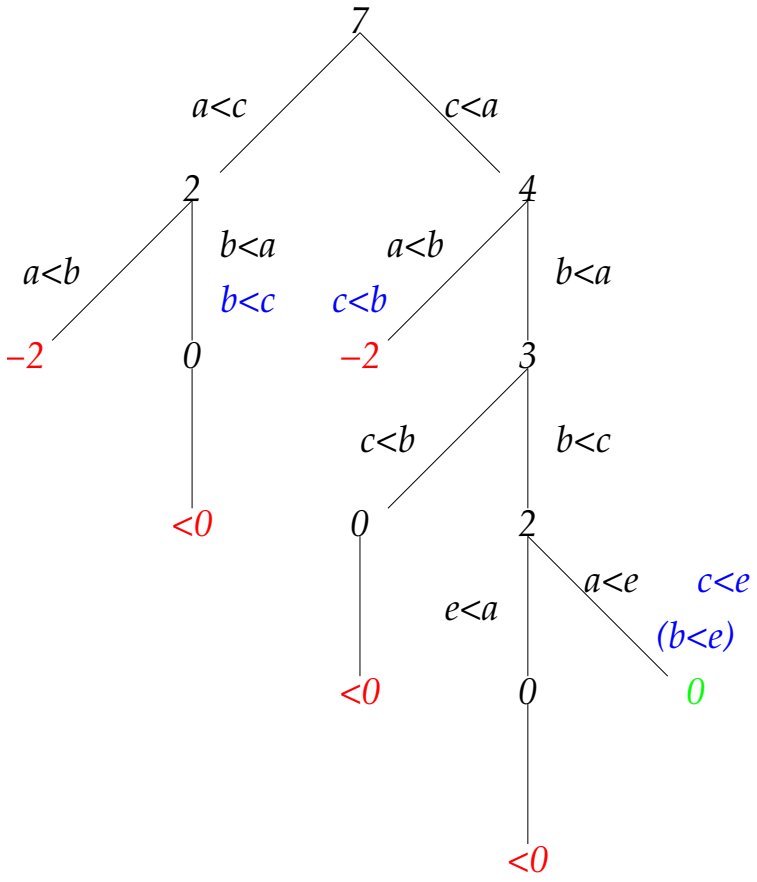
WS 2009/10 in Trier

Henning Fernau
fernau@uni-trier.de

Parameterisierte Algorithmen | Gesamtübersicht

- Einführung
- Grundbegriffe
- Problemkerne
- Suchbäume
- Graphparameter
- Weitere Methoden
- Komplexitätstheorie—parameterisiert

Suchbäume:



Suchbäume Aufgaben:

Arbeit, die in den Knoten zu tun ist, meist “trivial” (Polynomzeit).

$\rightsquigarrow$ exponentieller Aufwand lässt sich mit der Anzahl der Baumknoten abschätzen.

Dem “entspricht” auch “meist” die Anzahl der Blätter des Baumes. (Warum ?)

Aufbau der Suchbäume meist “rekursiv” (rekursive Prozedur).

Ist $\mathcal{T}(\mathrm{k})$ “typischer” Suchbaum für Parameter(wert) k, so wäre für folgendes Programmstück $\mathcal{T}(\mathrm{k}) = \mathcal{T}(\mathrm{k}-1) + \mathcal{T}(\mathrm{k}-1)$.

Algorithm 1 A simple search tree algorithm, called VCMH

Input(s): a graph $G = (V, E)$, a positive integer k

Output(s): YES if there is a vertex cover $C \subseteq V$, $|C| \leq k$, (and it will implicitly produce such a small cover then) or NO cover of size $\leq k$ exists.

if $k \leq 0$ and $E \neq \emptyset$ **then**
  return NO
else if $k \geq 0$ and $E = \emptyset$ **then**
  return YES
else
  Choose edge $e = \{x, y\} \in E$
  if VCMH$(G - x, k - 1)$ **then**
    return YES
  else
    return VCMH$(G - y, k - 1)$
  end if
end if

Algorithm 2 A simple search tree algorithm, called VCMH-C

Input(s): a graph $G = (V, E)$, a positive integer k

Output(s): a vertex cover $C \subseteq V$, $|C| \leq k$, or NO cover of size $\leq k$.

if $k \leq 0$ and $E \neq \emptyset$ **then**
  return NO
else if $k \geq 0$ and $E = \emptyset$ **then**
  return $\emptyset$
else
  Choose edge $e = \{x, y\} \in E$
  if $C :=$ VCMH-C$(G - x, k - 1) \neq$ NO **then**
    return $C \cup \{x\}$
  else if $C :=$ VCMH-C$(G - y, k - 1) \neq$ NO **then**
    return $C \cup \{y\}$
  else
    return NO
  end if{branching}
end if

Zur Korrektheit von VCMH-C

Ist $k \leq 0$ und $E \neq \emptyset$, so Fehlerfall ✓

Gilt andernfalls $E = \emptyset$, so ist $\emptyset$ eine gültige (kleinste) Überdeckung.
(Dies ist der Induktionsanfang für die Beh.: es wird stets eine gültige Überdeckung zurückgeliefert für Graphen mit m Kanten, die nicht größer ist, als der Parameterwert vorschreibt.)

Andernfalls gibt es eine Kante e und der Parameter k ist > 0.
$e = \{x, y\}$ muss durch entweder x oder y abgedeckt werden.
Die Graphen $G - x$ und $G - y$ haben wenigstens eine Kante weniger als G, somit ist die Induktionsannahme anwendbar, d.h.: wir können annehmen, dass C_x resp. C_y gültige hinreichend kleine Überdeckungen von $G - x$ resp. C_y sind und mithin $C_x \cup \{x\}$ und $C_y \cup \{y\}$ für G.

Zur Zeitkomplexität von VCMH-C

Die meisten “Elementaroperationen” gehen in $\mathcal{O}(1)$,
mit Ausnahme der Erstellung von $G - x$ benötigt $\mathcal{O}(n)$ Zeit.

Durch Induktion über die Suchbaumtiefe k folgt eine Laufzeit von $\mathcal{O}(2^k n)$.

Für die Suchbaumgröße $T(k)$ gilt nämlich: $T(k) \leq 2T(k-1)$.

Zusammenfassung für VC

Satz 1 *(Buss und Mehlhorn)* k*-VC kann in Zeit* $\mathcal{O}(nk + 2^k k^2)$ *gelöst werden.*

Exkurs Analytische Methodik—Erzeugende Funktionen / z-Transformation

Einen Zugang zur Lösung von Rekursionen gestattet die Analysis:

Interpretiere Folgen als Koeffizienten einer Potenzreihe.

Der Rekursionsgleichung (unter Berücksichtigung der Anfangsbedingungen) entspricht dann eine Funktionalgleichung für die erzeugende Funktion.
Dabei Konvention: $T(n) = 0$ für $n < 0$.
Nützliche Schreibweise: $[P(n)]$ für Prädikat P auf $\mathbb{Z}$ mit

$$[P(n)] = \begin{cases} 1 & P(n) \\ 0 & \neg P(n) \end{cases}$$

Beispiel: Rekursiongleichung $T(n) = 2T(n-1)$ mit Anfangsbedingung $T(0) = 1$

$\rightsquigarrow T(n) = 2T(n-1) + [n = 0]$ (eine Gleichung !)

$$\begin{aligned} G(z) &= \sum_{n\in\mathbb{Z}} T(n)z^n = \sum_{n\in\mathbb{Z}} (2T(n-1) + [n=0])z^n \\ &= 2z \sum_{n\in\mathbb{Z}} T(n-1)z^{n-1} + 1 = 2z \sum_{n\in\mathbb{Z}} T(n)z^n + 1 \\ &= 2zG(z) + 1 \end{aligned}$$

$\rightsquigarrow G(z) = \frac{1}{1-2z}$

$\rightsquigarrow$ Potenzreihenentwicklung $G(z) = \sum_{n\in\mathbb{N}} (2z)^n$, denn $\frac{1}{1-y} = \sum_{n\in\mathbb{N}} y^n$

$\rightsquigarrow T(n) = 2^n$

Achtung: Anfangsbedingungen

Betrachte:

$$T(n) = \begin{cases} 1 & n = 0 \\ 3 & n = 1 \\ 5 & n = 2 \\ T(n-1) + T(n-2) + T(n-3) & n \geq 3 \end{cases}$$

Ausdrücken in einer Rekursionsgleichung:

$$T(n) = T(n-1) + T(n-2) + T(n-3) + c_2[n = 2] + c_1[n = 1] + c_0[n = 0]$$

Wegen $T(n) = 0$ für $n < 0$ gilt hier: $T(0) = T(0-1) + T(0-2) + T(0-3) + c_0 = 1$ mit $c_0 = 1$.
Weiterhin: $T(1) = T(1-1) + T(1-2) + T(1-3) + c_1 = 1 + c_1 = 3$ mit $c_1 = 2$.
Schließlich: $T(2) = T(2-1) + T(2-2) + T(2-3) + c_2 = 1 + 3 + c_2 = 5$ mit $c_2 = 2$.

Ein hilfreiches Lemma

Ist $q(z) = 1 + q_1 z + \cdots + q_d z^d$, $q_d \neq 0$ so bezeichnet $q^R(z) = z^d + q_1 z^{d-1} + \cdots + q_d$ das reflektierte Polynom.

Lemma 2 *Sind $\alpha_1, \ldots, \alpha_d$ die Nullstellen des reflektierten Polynoms, so gilt:*

$$q(z) = \prod_{j=1}^{d} (1 - \alpha_j z).$$

Beweis: $q(z) = z^d \prod_{j=1}^{d} (1/z - \alpha_j)$ gilt, denn $q^R(y) = y^d q(1/y)$.

Fibonaccis Rekursion

geschlossene Rekursion: $F_n = F_{n-1} + F_{n-2} + [n = 1]$.

Daraus Funktionalgleichung:

$$F(z) = \sum F_n z^n = \sum F_{n-1} z^n + \sum F_{n-2} z^n + \sum [n = 1] z^n = zF(z) + z^2 F(z) + z.$$

$$\rightsquigarrow F(z) = \frac{z}{1 - z - z^2}$$

Partialbruchzerlegung mit dem hilfreichen Lemma und dem Ansatz:

$$\frac{1}{(1 - \alpha z)(1 - \beta z)} = \frac{a}{1 - \alpha z} + \frac{b}{1 - \beta z}$$

Nullstellen von $q^R(z) = z^2 - z - 1$: $\alpha = \frac{1+\sqrt{5}}{2} = \phi$ und $\beta = \frac{1-\sqrt{5}}{2} = 1 - \phi =: \hat{\phi}$.
Unser Ansatz liefert:

$$\frac{1}{(1-\phi z)(1-\hat{\phi}z)} = \frac{a}{1-\phi z} + \frac{b}{1-\hat{\phi}z} = \frac{(a+b) - (a\hat{\phi} + b\phi)}{(1-\phi z)(1-\hat{\phi}z)}$$

und damit das lineare Gleichungssystem mit den Bedingungen
$a + b = 1$ und $\hat{\phi}a + \phi b = 0$, also $a = \frac{\phi}{\sqrt{5}}$ und $b = -\frac{\hat{\phi}}{\sqrt{5}}$.
So erhalten wir die folgende schon bekannte Beziehung als explizite Darstellung der F_n:

$$F_n = \frac{\phi}{\sqrt{5}}\phi^{n-1} - \frac{\hat{\phi}}{\sqrt{5}}\hat{\phi}^{n-1}.$$

Da $|\hat{\phi}| = \left|\frac{1-\sqrt{5}}{2}\right| < 1$, ist F_n die zu $\frac{\phi^n}{\sqrt{5}}$ nächstgelegene ganze Zahl ist.
Daraus folgt unmittelbar: $F_n = \Theta(\phi^n)$.

z-Transformation allgemein (hier speziell für lineare Rekursionen)

1. Darstellung der Rekursion in einer Gleichung
   $f(n) = q_1 f(n-1) + q_2 f(n-2) + \cdots + q_d f(n-d) +$ *Anfangsbedingungen*.

2. Darstellung als erzeugende Funktion $F(z)$ und Auflösen der Funktionalgleichung als $F(z) = \frac{p(z)}{q(z)}$, wobei p und q Polynome sind vom Grad höchstens d.

3. Partialbruchzerlegung liefert

$$F(z) = \sum f(n) z^n = \sum_{i=1}^{k} \frac{g_i(z)}{(1-\alpha_i z)^{d_i}}$$

   Die α_i sind Nullstellen des (komplexen) Polynoms $q^R(z)$ der Vielfachheit d_i, g_i sind Polynome vom Grad kleiner d_i.

4. Explizite Darstellung der Rekursion: $f(n) = \sum_{i=1}^{k} p_i(n) \alpha_i^n$

Insbesondere gilt: $f(n) = \Theta(p_j(n)\alpha^n)$ mit $\alpha = \alpha_j = \max_{i=1}^{k} |\alpha_i|$.
Beweis: siehe Aigner, S. 62

Zurück zu VC

Geht es besser?

Einfache Beobachtung: Wenn ein gewisser Knoten definitiv nicht in die (zu konstruierende) Knotenüberdeckung kommt, so müssen *alle* seine Nachbarn in die Knotenüberdeckung.

Algorithm 3 A simple search tree algorithm, called VCMH′

Input(s): a graph $G = (V, E)$, a positive integer k

Output(s): YES if there is a vertex cover $C \subseteq V$, $|C| \leq k$, (and it will implicitly produce such a small cover then) or
NO if no vertex cover of size at most k exists.

if $k \leq 0$ and $E \neq \emptyset$ **then**
  return NO
else if $k \geq 0$ and $E = \emptyset$ **then**
  return YES
else
  Choose edge $e = \{x, y\} \in E$
  if VCMH′$(G - x, k - 1)$ **then**
    return YES
  else
    return VCMH′$(G - N(x), k - \deg(x))$
  end if
end if

Geht es besser?

Einfache Beobachtungen: 1. Wenn ein gewisser Knoten definitiv nicht in die (zu konstruierende) Knotenüberdeckung kommt, so müssen *alle* seine Nachbarn in die Knotenüberdeckung.

2. Ein Graph mit Maximalgrad Eins kann trivialerweise in Polynomzeit optimal gelöst werden.

Algorithm 4 A still simple search tree algorithm, called VCMH-TL

Input(s): a graph $G = (V, E)$, a positive integer k

Output(s): YES if there is a vertex cover $C \subseteq V$, $|C| \leq k$, or
NO if no vertex cover of size at most k exists.

if $k \leq 0$ and $E \neq \emptyset$ **then**
  return NO
else if $k \geq 0$ and $E = \emptyset$ **then**
  return YES
5: **else if** possible **then**
  Choose vertex $x \in V$ such that $\deg(x) \geq 2$.
  if VCMH-TL$(G - x, k - 1)$ **then**
    return YES
  else
10: return VCMH-TL$(G - N(x), k - \deg(x))$
  end if
else
  resolve deterministically
end if

Zur Laufzeit

Wir haben jetzt die folgende Rekursion zu lösen:

$$\begin{aligned} T(k) &= 1 \text{ falls } k < 1 \\ T(k) &\leq T(k-2) + T(k-1) \end{aligned}$$

Ansatz: $T(k) \leq c^k$ (Begründung s.o.)

$$c^k = c^{k-2} + c^{k-1}$$

Teilen durch c^{k-2} liefert:

$$c^2 = 1 + c$$

Daher: $c = 1.618...$

So gelangt man sehr einfach zu den Wurzeln des reflektierten Nennerpolynoms.

Weiter geht's: Triviality last

Graphen vom Maximalgrad zwei lassen sich auch deterministisch lösen
(gilt nicht mehr für Maximalgrad drei...)
Warum sind beide Aussagen richtig ?

$$\rightsquigarrow T(k) \leq T(k-3) + T(k-1) \approx 1.4656^k$$

Das umgekehrte Prinzip: Triviality first

Wir kennen Reduktionsregeln, die Knoten vom Grad Null und Eins “erledigen”

$\rightsquigarrow$ ein reduzierter Graph hat Minimalgrad zwei

$\rightsquigarrow$ alternativer Algorithmus für VC

Bem.: Faltungsregel erlaubt auch, Grad-2 Knoten zu behandeln.

Algorithm 5 Yet another simple search tree algorithm, called VCMH-TF

Input(s): a graph $G = (V, E)$, a positive integer k

Output(s): YES if there is a vertex cover $C \subseteq V$, $|C| \leq k$, or
NO if no vertex cover of size at most k exists.

Exhaustively apply the reduction rules, yielding $G = (V, E)$.
if $k \leq 0$ and $E \neq \emptyset$ **then**
  return NO
else if $k \geq 0$ and $E = \emptyset$ **then**
  return YES
else
  Choose some vertex $x \in V$
  if VCMH-TF$(G - x, k - 1)$ **then**
    return YES
  else
    return VCMH-TF$(G - N(x), k - \deg(x))$
  end if
end if

Korrektheit? Klar aus früheren Überlegungen

Zeit? Mit den bisherigen Regeln (wie oben berechnet) $\mathcal{O}^*(1.61..^k)$

Reduktionsregel 1 *Sei* (G, k) *eine VC-Instanz. Sei* $v \in V(G)$ *mit* $N(v) = \{u, w\}$.

- *Angenommen* $u \notin N(w)$. *Konstruiere* $G' = G[u = w] - v$. *Die neue Instanz ist* $(G', k-1)$.

- *Gilt* $u \in N(w)$, *so füge* u, w *in die Überdeckungsmenge ein und reduziere zu* $(G - \{u, v, w\}, k-2)$.

Damit erhalten wir sogar dieselbe Laufzeit wie bisher.

Häufig am günstigsten: kombiniere beide Ansätze des Umgangs mit Trivialitäten.

Andere (ähnliche) Probleme I:

GEWICHTETES KNOTENÜBERDECKUNGSPROBLEM (WVC)
Eingabe: ein Graph $G = (V, E)$ mit Knotengewichten $\omega : V \to \mathbb{R}_{\geq 1}$
Parameter: eine natürliche Zahl k
Frage: Gibt es eine Knotenüberdeckung $C \subseteq V$ mit $\omega(C) \leq k$?

Problem: Es gibt keine einfache Grad-1-Regel !

Andere (ähnliche) Probleme II:

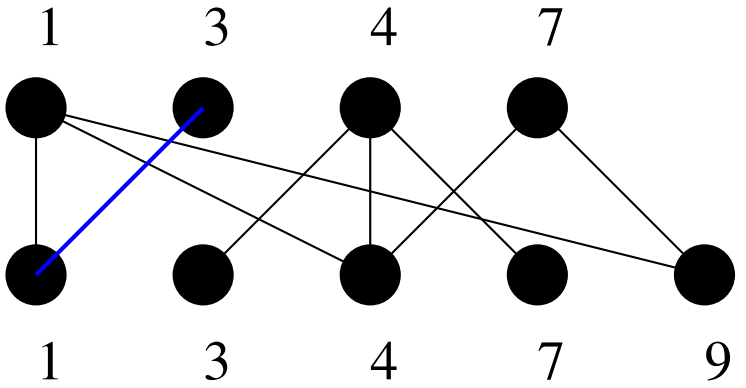
EINGESCHRÄNKTES BIPARTITES KNOTENÜBERDECKUNGSPROBLEM (CBVC)
Eingabe: ein bipartiter Graph $G = (V_1, V_2, E)$
Parameter: natürliche Zahlen k_1, k_2
Frage: Gibt es eine Knotenüberdeckung $C \subseteq V_1 \cup V_2$ mit $|C \cap V_i| \leq k_i$ für $i = 1, 2$?

Anwendung: Chip-Produktion

Hinweis: Unlängst abgeschlossene Diplomarbeit

Motivation CBVC

	1	2	3	4	5	6	7	8	9
1	?			?					?
2									
3	?								
4			?	?			?		
5									
6									
7				?					?



Die Fragezeichen markieren fehlerhafte Bauelemente.

Erinnerung:

“Fehlergraph” entspricht paarem Graphen zu gegebener binärer Relation.

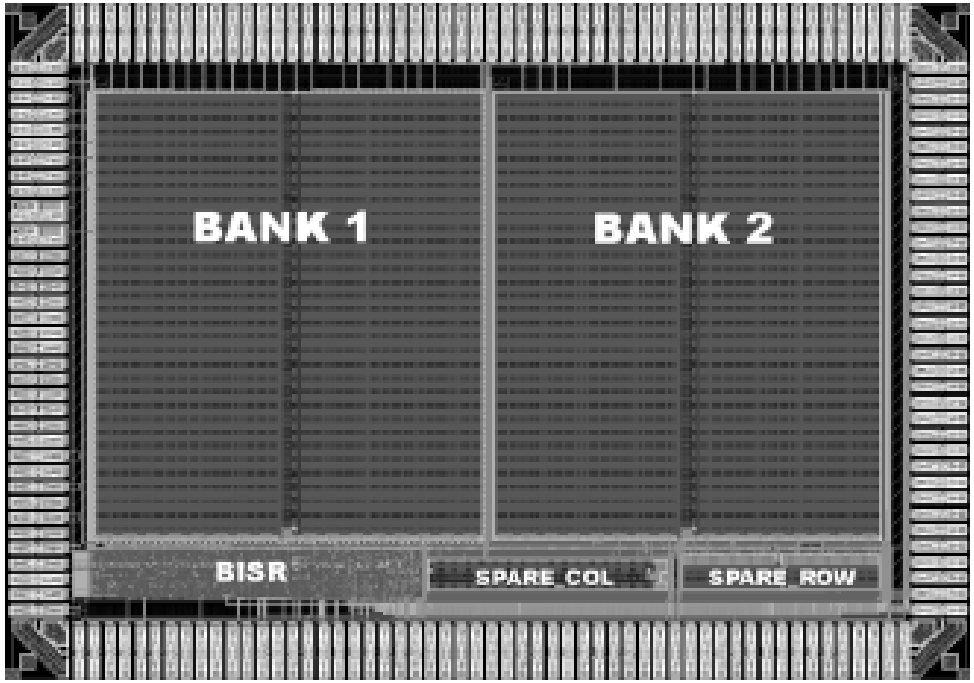
Speicherrekonfigurierung

- Physikalische Probleme bei der Fertigung großer Speicher-Matrizen
  $\rightsquigarrow$ verschlechterte Ausbeute

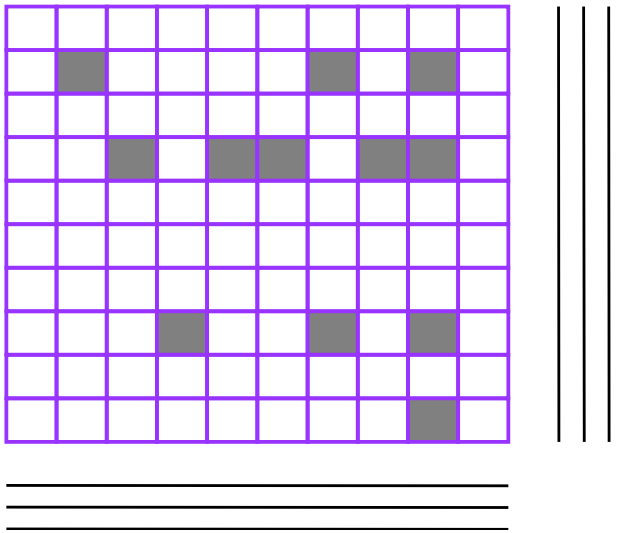
- Häufige Strategie:
  Bevorraten mit Ersatz-Zeilen und -Spalten.
  Diese kommen zum seit den 70er Jahren mit Variationen zum Einsatz.
  Die (für uns hier unbedeutende) eigentliche Reparaturtechnik ändert sich.

- Naheliegende Formalisierungen sind *NP*-vollständig.
  Klar: Streben nach wenig Redundanz
  $\rightsquigarrow$ Es wird nur wenig Ersatz bereitgestellt.
  $\rightsquigarrow$ In der Praxis kleiner Parameter $\mathrm{k} < 50$!

BISR: Built-in self repair: Heutige Methodik bei Speicherkomponenten



Speicherrekonfigurierung an einem Beispiel

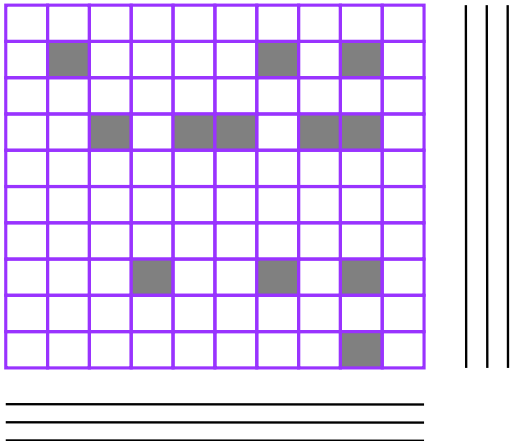


Quadrate: Speicherzellen
Linien: Austauschstücke.
Graue Zellen seien fehlerhaft.
Höchstens drei Zeilen
und drei Spalten
stehen als Ersatz
zur Verfügung.

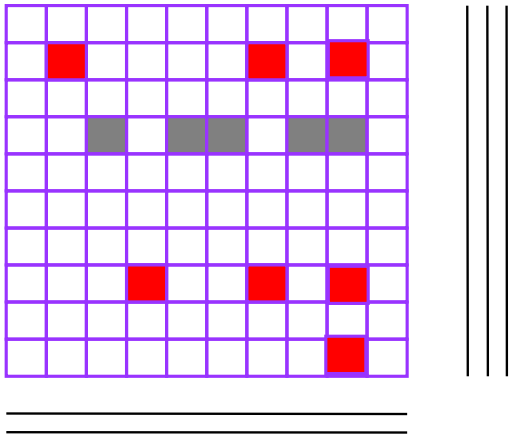
Speicherrekonfigurierung: Auswahl einer Zeile zum Ersetzen

$\rightsquigarrow$ nun eine Zeile weniger zum Rekonfigurieren zur Verfügung

Ausgangssituation



Nach Austausch der vierten Zeile



Speicherrekonfigurierung formal beschrieben

SPEICHERREKONFIGURIERUNG SAP

Eingabe: Eine binäre $n \times m$ Matrix A (fehlerbehafteter Chip) $A[r, c] = 1 \iff$ der Chip is an Stelle $[r, c]$ fehlerhaft

Parameter: natürliche Zahlen k_1, k_2

Frage: Gibt es eine *Reconfigurationsvorschrift*, die alle Fehler behebt und dazu höchstens k_1 Ersatzzeilen und höchstens k_2 Ersatzspalten benötigt?

Testergebnisse Blough's Speicherfehlermodell in Testläufen

m=n	$k_1=k_2$	p_1	p_2	r	success (%)	without solution				with solution			
						Alg 2	# br	Alg 3	# br	Alg 2	# br	Alg 3	# br
1024	32	0.000007	0.8	5	29	0.1003	12	0.1225	12	0.4440	99	0.4416	98
1024	32	0.000004	0.8	9	11	0.1341	6	0.1337	6	0.2184	32	0.2212	32
1024	32	0.000002	0.8	15	100	----------	---	----------	---	0.0663	3	0.0660	3
1024	36	0.000003	0.8	15	8	0.0402	0	0.0407	0	0.0475	0	0.0488	0
1024	36	0.000005	0.7	9	6	0.0445	1	0.0457	1	0.3705	37	0.3255	37
2048	64	0.000003	0.7	7	3	0.0441	0	0.0425	0	5.2006	574	5.0236	573
2048	64	0.000002	0.5	9	10	0.0345	0	0.0322	0	12.6780	1875	10.6960	1875
4096	128	0.000001	0.7	7	30	0.0714	0	0.0728	0	11.2197	576	11.216	574

Andere (ähnliche) Probleme III:

3-SAT
Eingabe: Eine Boolesche Formel F in konjunktiver Normalform (CNF)mit Variablen X, jede Klausel mit höchstens drei Literalen
Parameter: eine natürliche Zahl k, die die Zahl der Klauseln mit drei Literalen beschränkt
Frage: Gibt es eine erfüllende Belegung $\alpha : X \to \{0, 1\}$ für F?

Bem.: 2-SAT liegt in P, sodass DSAT mit "triviality last" in $\mathcal{O}^*(2^k)$ lösbar ist.

Reduktionsregeln für DSAT I:

Reduktionsregel 2 *Sei* (F, X, k) *eine Instanz von DSAT und* $x \in X$*.*

- *Erscheint* x *nur als positives Literal in* F*, so setze* x *"wahr" und lösche alle Klauseln mit* x *aus* F*,* x *selbst aus* X *und reduziere den Parameter um die Zahl der Klauseln der Größe drei in* F*, in welchen* x *auftauchte.*

- *Erscheint* x *nur als negatives Literal in* F*, so setze* x *"falsch" und lösche alle Klauseln mit* $\bar{x}$ *aus* F*,* x *selbst aus* X *und reduziere den Parameter um die Zahl der Klauseln der Größe drei in* F*, in welchen* $\bar{x}$ *auftauchte.*

Reduktionsregeln für DSAT II:

Reduktionsregel 3 *Sei* (F, X, k) *eine Instanz von DSAT und* $x \in X$*. Ist* C *eine Klausel mit allein der Variablen* x*, dann tue:*

- *Sind* x *und* $\bar{x}$ *beide in* C *enthalten, so lösche* C *aus* F*.*
- *Ist nur* x *in* C *enthalten, setze* x *"wahr", lösche* C *aus* F *und* x *aus* X*.*
- *Ist nur* $\bar{x}$ *in* C *enthalten, setze* x *"falsch", lösche* C *aus* F *und* x *aus* X*.*

Dies betrifft nur den Parameter, falls C *drei Literale enthält.*

Reduktionsregeln für DSAT III:

Reduktionsregel 4 *Sei* (F, X, k) *eine Instanz von DSAT und* $x \in X$*.*

Annahme, Regel 3 wäre nicht anwendbar. Sind x *und* $\bar{x}$ *in* C *enthalten, so lösche* x *und* $\bar{x}$ *aus* C*, entsprechend* F *modifizierend. Dekrementiere den Parameter.*

Mitteilung: Die Reduktionsregeln sind korrekt.

Damit: DSAT in Zeit $\mathcal{O}^*(1.61^k \dots)$

Zeitanalyse Vor der Verzweigung ist F reduziert, d.h., für jede Variable x gilt: es gibt eine Klausel C mit $x \in C$ und eine Klausel $\bar{C}$ mit $\bar{x} \in \bar{C}$. Ferner ist $|C|, |\bar{C}| \geq 2$, sowie o.E. $|C| + |\bar{C}| \geq 5$.

Falls $|C| = |\bar{C}| = 3$, so Rekursionsformel $T(k) \leq 2T(k-2)$ für Suchbaumgröße.

Falls o.E. $|C| = 3$ und $|\bar{C}| = 2$, so beobachte:
Setzen wir x "wahr", so wird $\bar{C} = \bar{x} \vee \ell$ nur dann wahr, wenn das Literal ℓ wahr ist durch entsprechendes Setzen der zu ℓ gehörigen Variablen y.
Da F reduziert, gibt es eine weiter Klausel C_y, in der $\bar{\ell}$ als Literal auftaucht.
(Unterfall $C_y = C$ trivial!)
Gilt $|C_y| = 2$, so wiederholt sich das Argument, bis entweder die gesamte Formel deterministisch gelöst wird oder bis o.E. $|C_y| = 3$ gilt.
Nach dem Verzweigen gilt hat C_y nur noch zwei Literale.
Setzen wir x "falsch", "gewinnen" wir nichts weiter.
Zusammen erhalten wir $T(k) \leq T(k-2) + T(k-1)$.

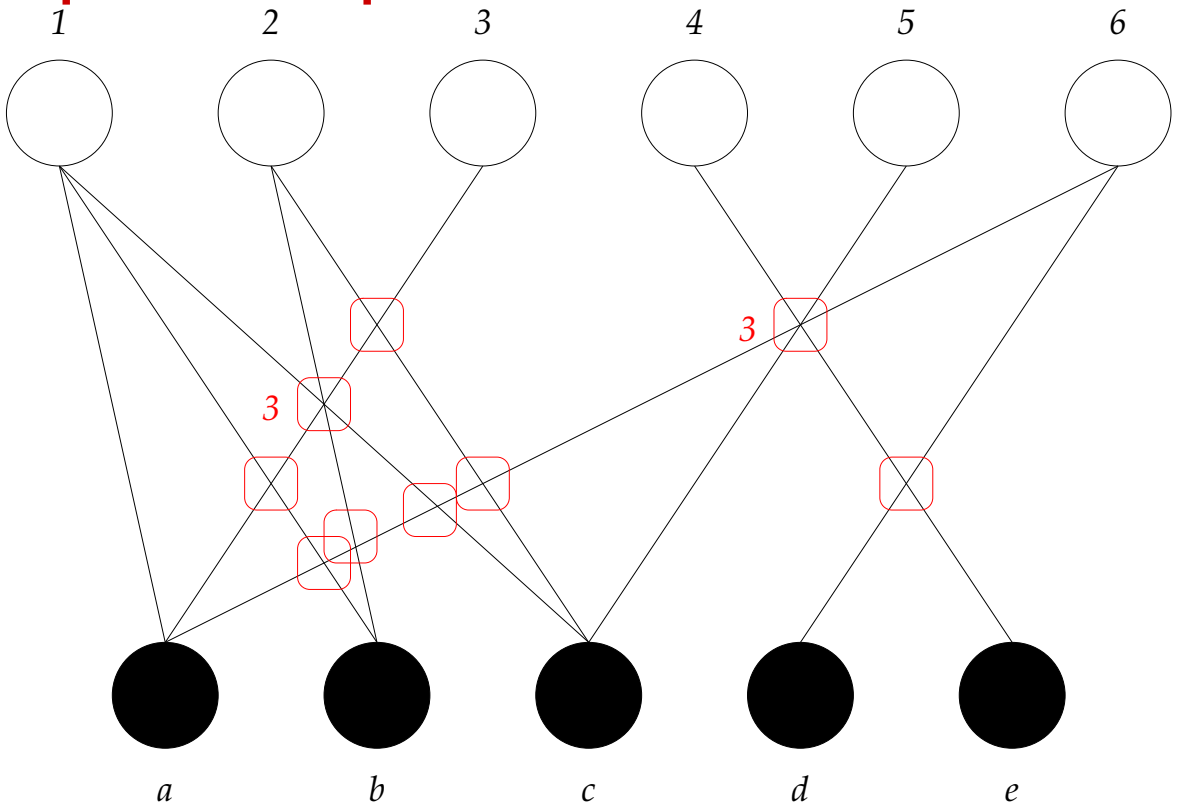
Algorithm 6 A simple search tree algorithm, called 3SAT

Input(s): a set F of clauses, at most k of them having three literals

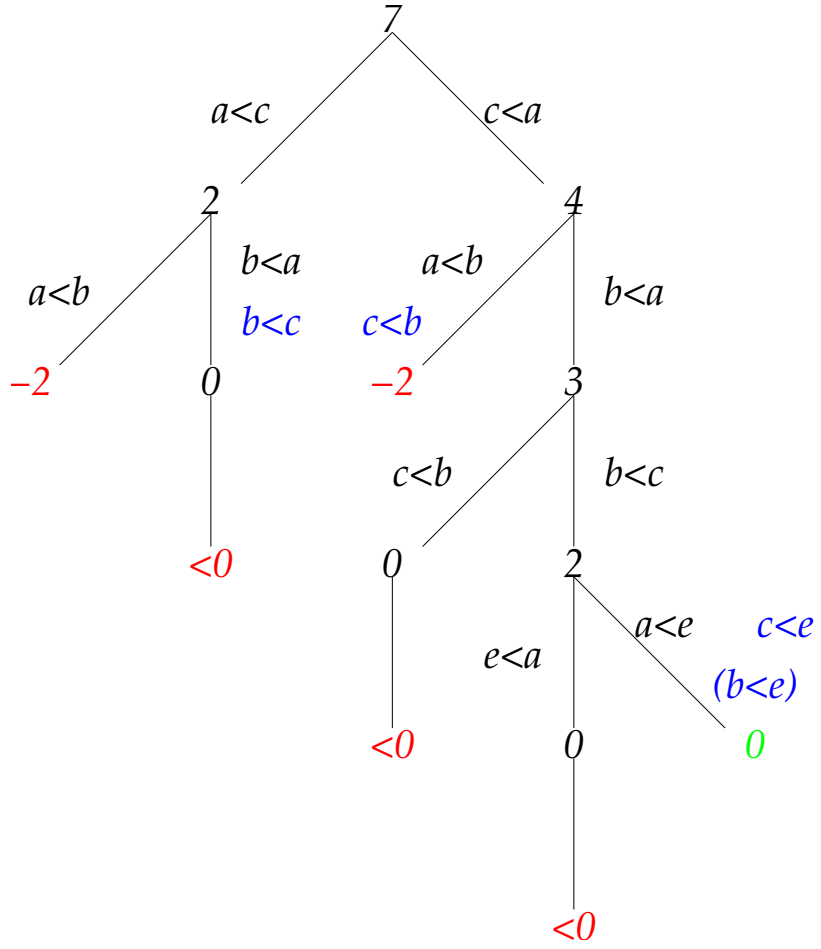
Output(s): YES if there is a satisfying assignment for F; NO otherwise

```
    Exhaustively apply rules 2, 3 and 4;
    if k ≤ 0 then
        return 2-SAT(F)
    else
5:      Choose variable x that occurs in some clause c with three literals.
        ℓ := # clauses in which x occurs either as positive or negative literal.
        Let F′ be the formula obtained from F by setting x true.
        Let F″ be the formula obtained from F by setting x false.
        if 3SAT(F′, k − ℓ) then
10:         return YES
        else
            return 3SAT(F″, k − ℓ)
        end if
    end if
```

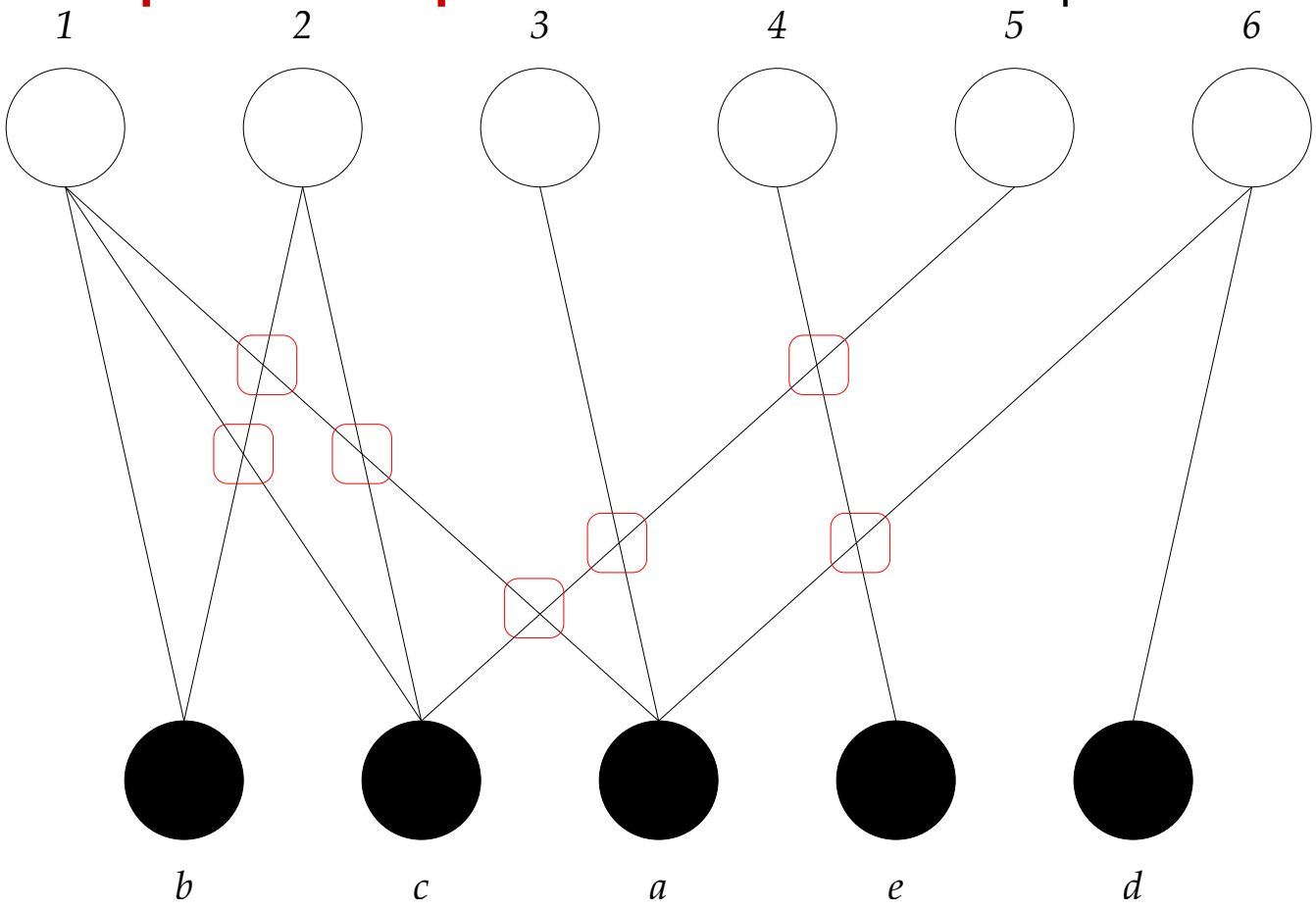
Andere (ähnliche) Probleme IV:
Zeichnen von bipartiten Graphen in der Ebene



Zeichnen von bipartiten Graphen in der Ebene: Wieder binäres Verzweigen



Zeichnen von bipartiten Graphen in der Ebene: Das Optimum



Weitere Techniken I: Regularität

wieder VC mit TF (Reduktionsregeln!):
Was ist der schlimmste Fall in unserer Verzweigung ?
Ein 3-regulärer Graph (alle Knoten Grad drei).

Dann werden aber Grad-2 oder Grad-1 Knoten “erzeugt” und daher Reduktionsregeln gezündet. Daher Laufzeit:

$$T(k) \leq 1.3803^k$$

Weitere Techniken II: Komponenten

“Meistens” lassen sich Graphprobleme für unzusammenhängende Graphen dadurch lösen, dass Zusammenhangskomponenten getrennt behandelt werden.

Beobachtung: Das “Parameter-Budget” kann man verrechnen, und man kann meist einfach prüfen, ob eine Komponente überhaupt irgendetwas vom Budget benötigt.

Damit ergibt sich als Schranke für die Gesamtlaufzeit in $\mathcal{O}^*$ die Laufzeit der größten Komponente.

Folgerung 3 *Die Regularitätsbeobachtung verbessert die Abschätzung der Laufzeit.*

Genauer: Sei T_c die Laufzeit für einen Graphen mit c Komponenten, so gilt:
$T_c(k) \leq c \cdot (T_1(k - rc)) \leq T_1(k)$ für eine geeignet gewählte Konstante r;
bis zur Parametergröße werden Probleme in Polynomzeit gelöst (Grad von r abhängig).
Aufgabe: Konkretes Argument für VC ?!

Weitere Techniken III: exponentieller Platz

Grundidee:
"auf Vorrat" oder "bei Bedarf" werden optimale (z.B.) Knotenüberdeckungen für induzierte Teilgraphen bis zu einer gewissen Größe berechnet und tabelliert.

Evtl. sinnvoll bei Problemen mit linearen Kernen ($\leq d \cdot k$) mit guten Algorithmen für "kleine Probleme":
Ist nämlich Parameter von k auf $\alpha \cdot k$ gefallen, so hat Problem Größe $\leq \alpha dk$. In der vorher berechneten Tabelle könnte dieser Wert stehen.
Mit $T(k) \leq c^k$ wäre dann nur noch ein Aufwand von $c^{(1-\alpha)k}$ zu erledigen.
Die Tabellengröße ist so zu wählen, dass deren Berechnungsaufwand ebenfalls $c^{(1-\alpha)k}$ entspricht.

Hinweis: Platz "teurer" als Zeit; es gibt Exp.-Zeitverfahren, die inhärent exp. Platz brauchen.