Grundlagen Theoretischer Informatik 3

SoSe 2012 in Trier

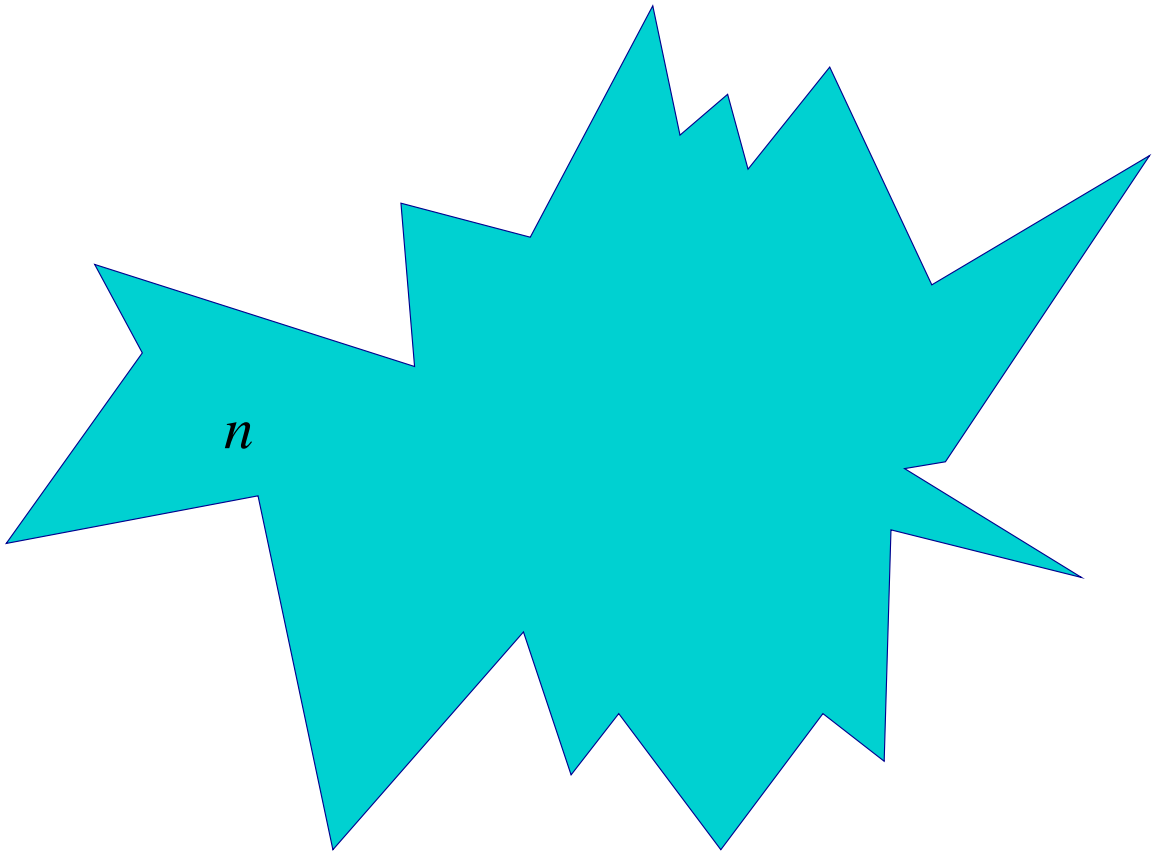
Henning Fernau
Universität Trier
fernau@uni-trier.de

Beispiele für *NP*-vollständige Probleme

- SATisfiability (Erfüllbarkeit logischer Formeln: Satz von Cook)

- Graphprobleme (*Standardparameterisierung* für Optimierungsprobleme)
  - Gibt es eine *Knotenüberdeckung* der Größe $\leq k$? (Vertex Cover VC)
  - Gibt es eine *unabhängige Menge* der Größe $\geq k$? (Independent Set IS)
  - Gibt es eine *dominierende Menge* der Größe $\leq k$? (Dominating Set DS)

The Curse of Combinatorics (Folklore)



Bekannte Auswege:

- Suchbäume (branch & bound):
  Exponentialzeit; Laufzeitgarantien?

- Näherungsalgorithmen:
  Polynomzeit; Güteschranken; nicht optimal

- Heuristiken, u.a. Reduktionsregeln

- Randomisierung

Exakte Wege: praktisch betrachtet

- Suchbäume (branch & bound):
  Exponentialzeit; Laufzeitgarantien?
- ILP-Techniken
- SAT-Löser
- (O)BDD-Techniken
- parameterisierte Algorithmen, u.a. Reduktionsregeln

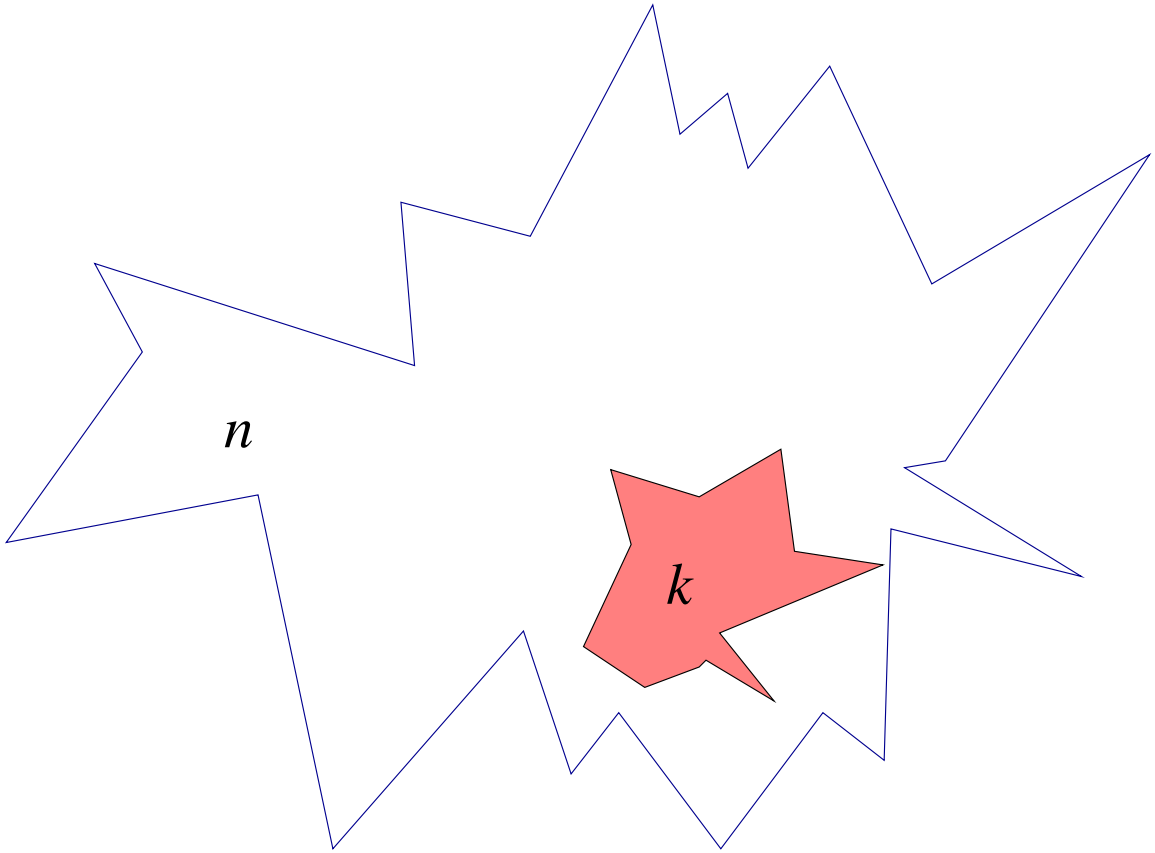
Wie “schlimm” ist Exponentialzeit?

Vergleichen wir exponentielle mit polynomiellen Laufzeiten für $n = 50$ auf einem Rechner, der 10^8 Operationen je Sekunde bearbeitet.

Polynomiell		Exponentiell	
Komplexität	Laufzeit	Komplexität	Laufzeit
n^2	25 µs	1.2^n	91 µs
n^3	1 ms	1.5^n	6 s
n^5	3 s	2^n	130 Tage
n^{100}	$9.13 \cdot 10^{156}$ Jahre	3^n	$228 \cdot 10^6$ Jahre

$\rightsquigarrow$ “Moderate” Exponentialzeit ist nicht “schlimmer” als große Polynome.

The Curse of Combinatorics Eine zweidimensionale Sicht



Ziel:

- exakte Algorithmen mit
- Laufzeitgarantien

Methoden: (u.a.)

- “Problemkerne” (Datenreduktion)
- Suchbäume
- Wohlquasiordnungen
- Dynamisches Programmieren
- Iteratives Erweitern
- Farbkodieren

Parameterisierte Algorithmik

Die Klasse *FPT* (fixed parameter tractable) enthält Sprachen $L \subseteq \Sigma^* \times \mathbb{N}$, für die es einen Algorithmus gibt, der die Frage "$(x, k) \in L$?" in Zeit

$$f(k) \cdot p(|x|)$$

entscheidet (f bel., p Polynom).
Alternative Notation der Laufzeit: $\mathcal{O}^*(f(k))$.

Satz 1 *Gleichwertig mit der gegebenen FPT-Def. ist:*
Es gibt einen Problemkern (x', k') *zu Instanz* (x, k),
d.h. $k', |x'| \in \mathcal{O}(g(k))$, g *beliebig.*
Die zugehörige Problemkernreduktion ist in Polynomzeit berechenbar.

Wahl des Parameters ist zunächst willkürlich.

Üblich bei Optimierungsproblemen: Standardparameter:
Schranke auf die Größe der Lösungsmenge oder die Kosten

Bei Graphproblemen möglich:
Knotenzahl, Kantenzahl, Baumweite, ...

Wir werden dies im Folgenden am unterschiedlichen Problemen studieren.

Parameter Knotenzahl $n \rightsquigarrow$ exakte Algorithmen

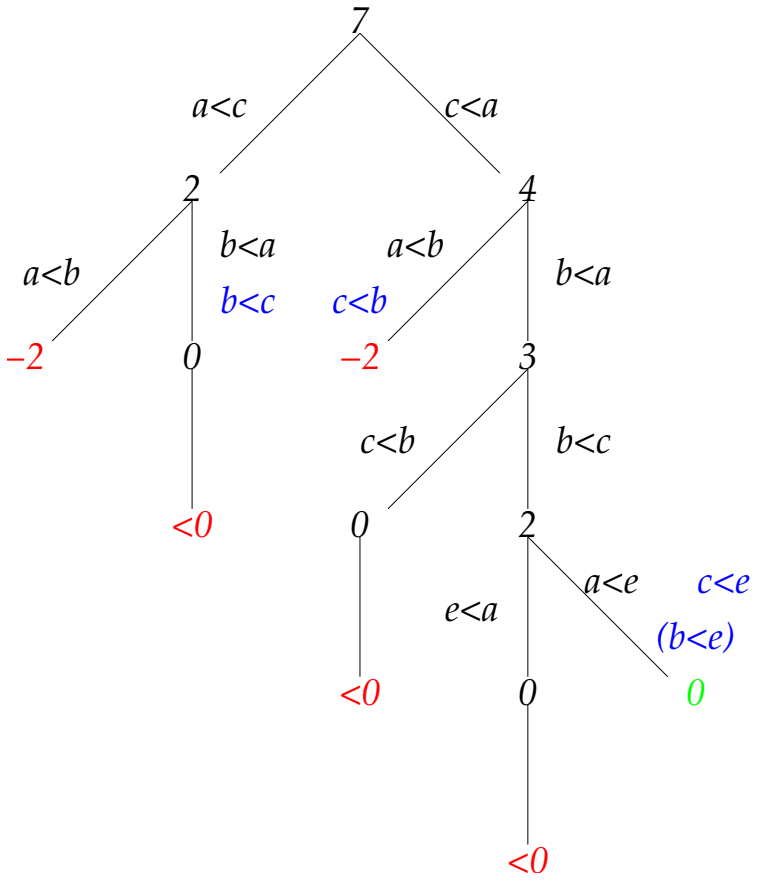
Meistens trivial: Laufzeit $\mathcal{O}^*(2^n)$.
Beispiel: Auffinden kleinstmöglicher dominierender Mengen (MDS).

Bis vor wenigen Jahren galt diese Grenze bei MDS als scharf.

"Mittlerweile" kann man "fast" auf $\mathcal{O}^*(1.5^n)$ abschätzen.

Solche exakten Algorithmen können von praktischem Interesse sein.

Sehr wichtig: Komplexitätsabschätzung bei Suchbäumen:



Das Knotenüberdeckungsproblem

Es sei $G = (V, E)$ ein (Hyper-)Graph.

$C \subseteq V$ heißt *(Knoten-)Überdeckung (vertex cover)* gdw. $\forall e \in E \exists v \in C : v \in e$.

VERTEX COVER
Eingabe: Ein ungerichteter Graph $G = (V, E)$
Parameter: Eine natürliche Zahl k
Frage: Gibt es eine Knotenüberdeckung $C \subseteq V$ von G mit $|C| \leq k$?

Satz 2 (Mehlhorn 1984) *VC kann in Zeit $\mathcal{O}^*(2^k)$ gelöst werden.*

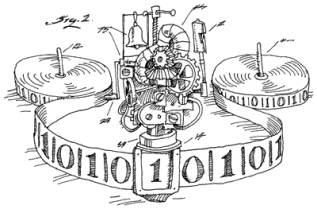
Hinweis: Mit verfeinerter Analyse lassen sich (kompliziertere) Suchbaumalgorithmen angeben, die $\mathcal{O}^*(1.27^k)$ erreichen.

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$(G - y, k - 1)$



Das Halteproblem auf Turing-Maschinen
ist einer der natürlichen Kandidaten,
um “harte Probleme” zu definieren.

Klassische Berechenbarkeitstheorie: Das Halteproblem ist unentscheidbar.

Klassische Komplexitätstheorie: Schränkt man die Schrittzahl einer nichtdeterministischen TM ein, so gelangt man zu typischen *NP*-harten Problemen.

parameterisierte Sicht $\rightsquigarrow$

KURZE NTM-AKZEPTANZ

Eingabe: Einband-NTM M
Parameter: $k \in \mathbb{N}$
Frage: Gibt es Berechnung von M auf leerer Eingabe, die in $\leq k$ Schritten hält?

Das Halteproblem auf Turing-Maschinen: Glaubensfragen

Wie ?! Simulation einer nichtdeterministischen TM durch eine deterministische
Am einfachsten: systematisches Durchprobieren aller Verzweigungen.
$\rightsquigarrow$ Für KURZE NTM-AKZEPTANZ (deterministische) Laufzeit von n_M^k.
Jedenfalls "auf diese Weise" liegt das Problem nicht in *FPT*.

$\rightsquigarrow$ *W[1]*: Klasse von parameterisierten Problemen (P, k) mit
$(P, k) \leq^{FPT}$ KURZE NTM-AKZEPTANZ.

Vermutung: $FPT \neq W[1]$ (ähnlich zu $P \neq NP$)

Hinweis: Es gibt eine ganze Hierarchie $FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots$ "immer schwierigerer" parameterisierter Problemklassen.

Ein Beispiel für ein W[1]-Problem

I heißt *unabhängige Menge* gdw. $N(I) \cap I = \emptyset$.

INDEPENDENT SET

Eingabe: Ein ungerichteter Graph $G = (V, E)$
Parameter: Eine natürliche Zahl k
Frage: Gibt es eine unabhängige Menge $I \subseteq V$ von G mit $|I| \geq k$?

Lemma 3 INDEPENDENT SET *liegt in W[1].*

Speichere G in der Übergangstabelle von M_G. Generiere für jeden Knoten Extra-Eingabezeichen.
M_G rät in den ersten k Schritten k Knoten und schreibt sie auf sein Arbeitsband.
In den folgenden $\mathcal{O}(k^2)$ Schritten überprüft M_G (deterministisch!), ob die geratene Knotenmenge eine unabhängige Menge bildet.
Mitteilung: INDEPENDENT SET ist W[1]-vollständig.
Folgerung: CLIQUE ist W[1]-vollständig.

Anwendungsprobleme: Zum Sinn der parameterisierten Sicht

1. Speicherrekonfigurierung: $\leadsto$ bipartite Variante von VC
   Kleiner Parameter: bereitzuhaltende Redundanz

2. Graphenzeichnen: z.B. hierarchisches Zeichnen
   $\leadsto$ Zeichnen bipartiter Graphen
   Kleiner Parameter: Kreuzungsanzahl

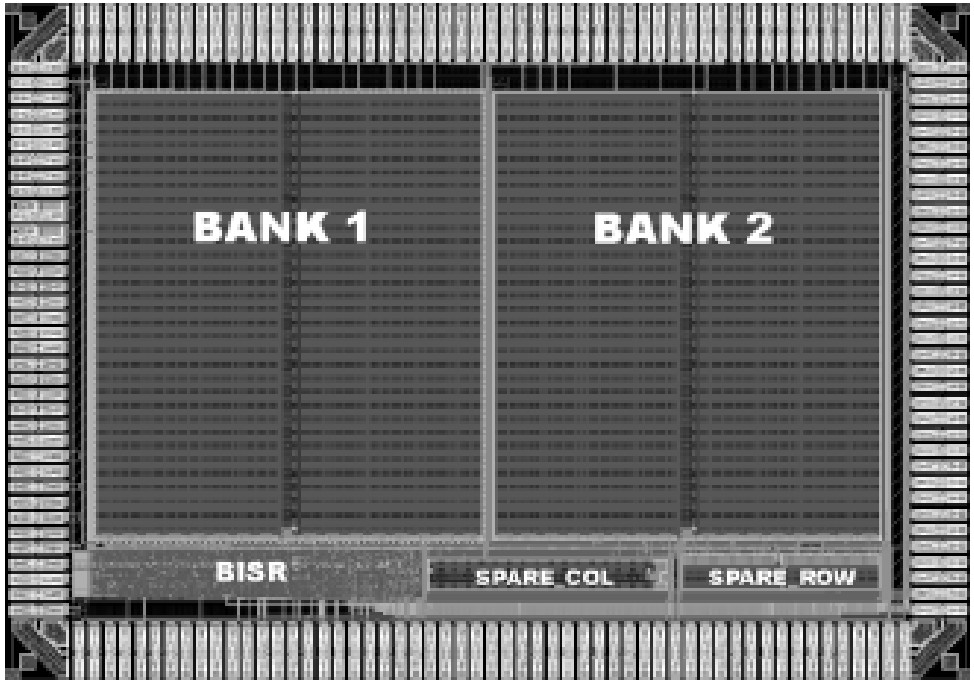
Speicherrekonfigurierung

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

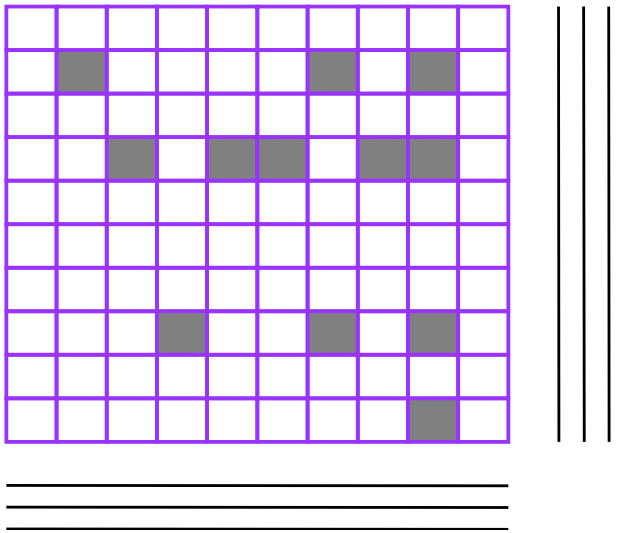
- Häufige Strategie:
  Bevorraten mit Ersatz-Zeilen und -Spalten.
  Diese kommen zum seit den 70er Jahren mit Variationen zum Einsatz.
  Die (für uns hier unbedeutende) eigentliche Reparaturtechnik ändert sich.

- 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

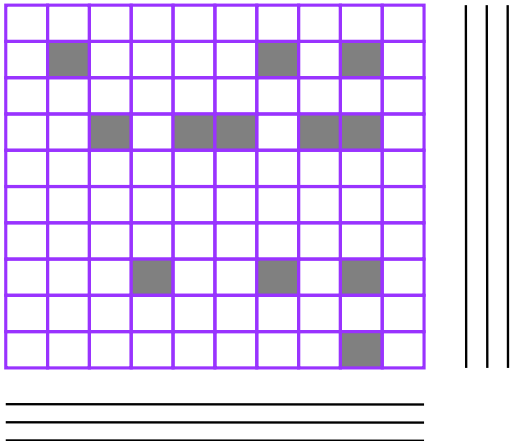


Quadrate: Speicherzellen
Linien: Austauschstücke.
Graue Zellen seien fehlerhaft.
Höchstens drei Zeilen
und drei Spalten
stehen als Ersatz
zur Verfügung.

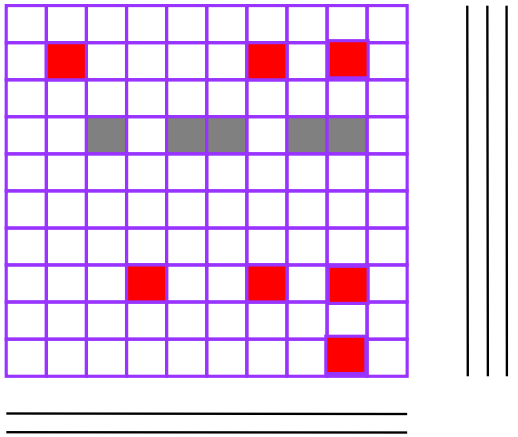
Speicherrekonfigurierung: Auswahl einer Zeile zum Ersetzen

$\rightsquigarrow$ 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 *Rekonfigurationsvorschrift*, die alle Fehler behebt und dazu höchstens k_1 Ersatzzeilen und höchstens k_2 Ersatzspalten benötigt?

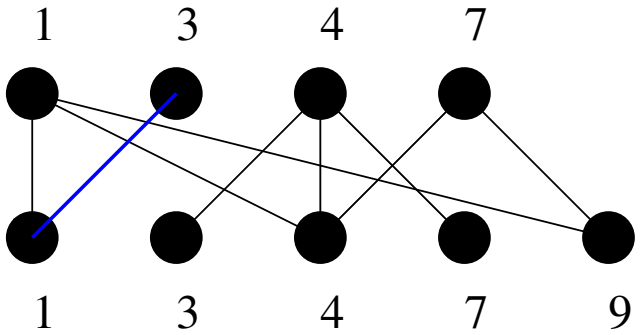
Alternative Formalisierung

BIPARTITES ZWEIPARAMETRIGES 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 $i = 1, 2$?

Beie Probleme sind “offenbar” äquivalent.
$\rightsquigarrow$ VC-Suchbaumalgorithmus ist auch hier anwendbar
$\rightsquigarrow$ CBVC in Zeit $\mathcal{O}^*(2^{k_1+k_2})$ lösbar (lässt sich auch weiter verbessern)

Ein kleines Beispiel

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

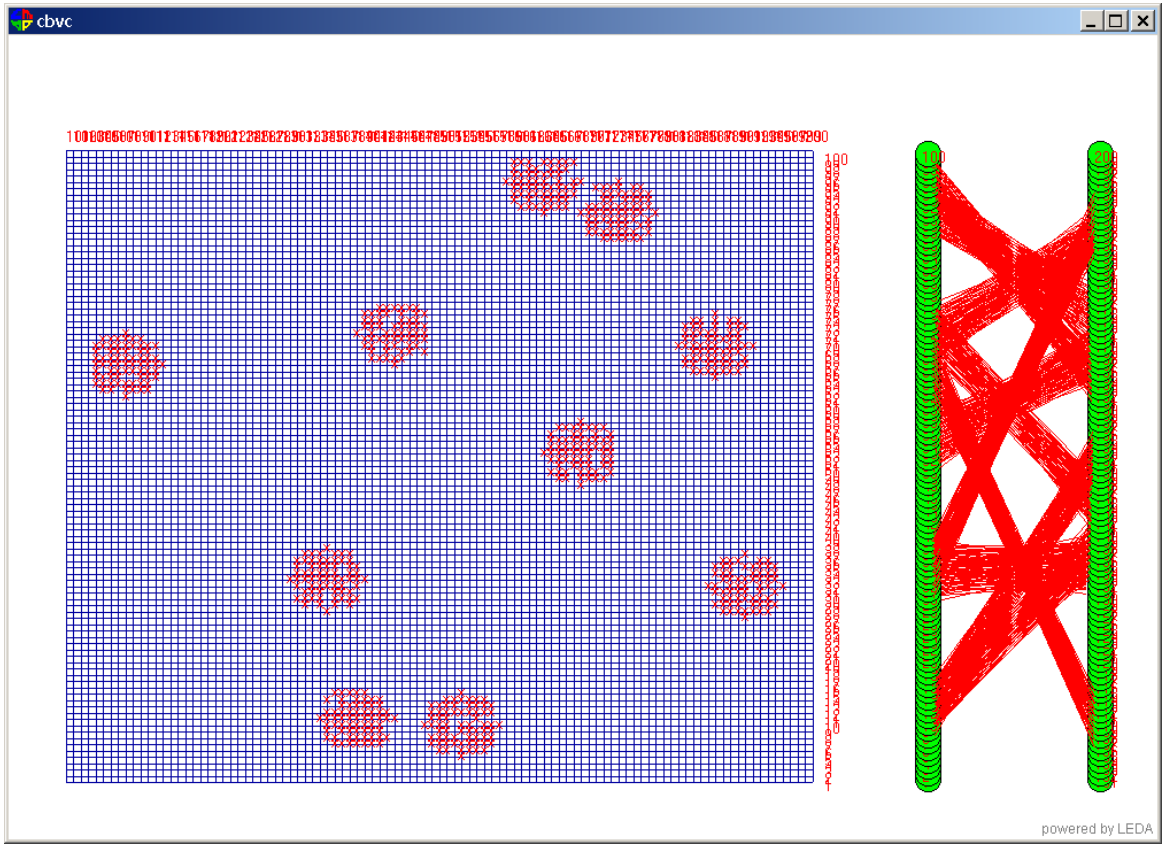


Zur *NP*-Härte

Wir zeigen: CLIQUE $\leq_p$ CBVC.
Sei Clique-Instanz $G = (V, E)$, k gegeben.
Konstruiere $G' = (V', E')$ mit:
$V' = V \cup E$ und
$xy \in E'$ gdw. $x \in V$, $y \in E$ und $x \in y$ (in G).
Die Bipartition $A \cup B$ von V' ist def. durch $A = V$ und $B = E$.
Ferner sei: $k_A = k$ und $k_B = |B| - (k(k-1)/2)$.

Übung: Zeige Korrektheit der Konstruktion: G besitzt k-Clique gdw. G' besitzt (k_A, k_B)-CBVC.

Blough's Speicherfehlermodell



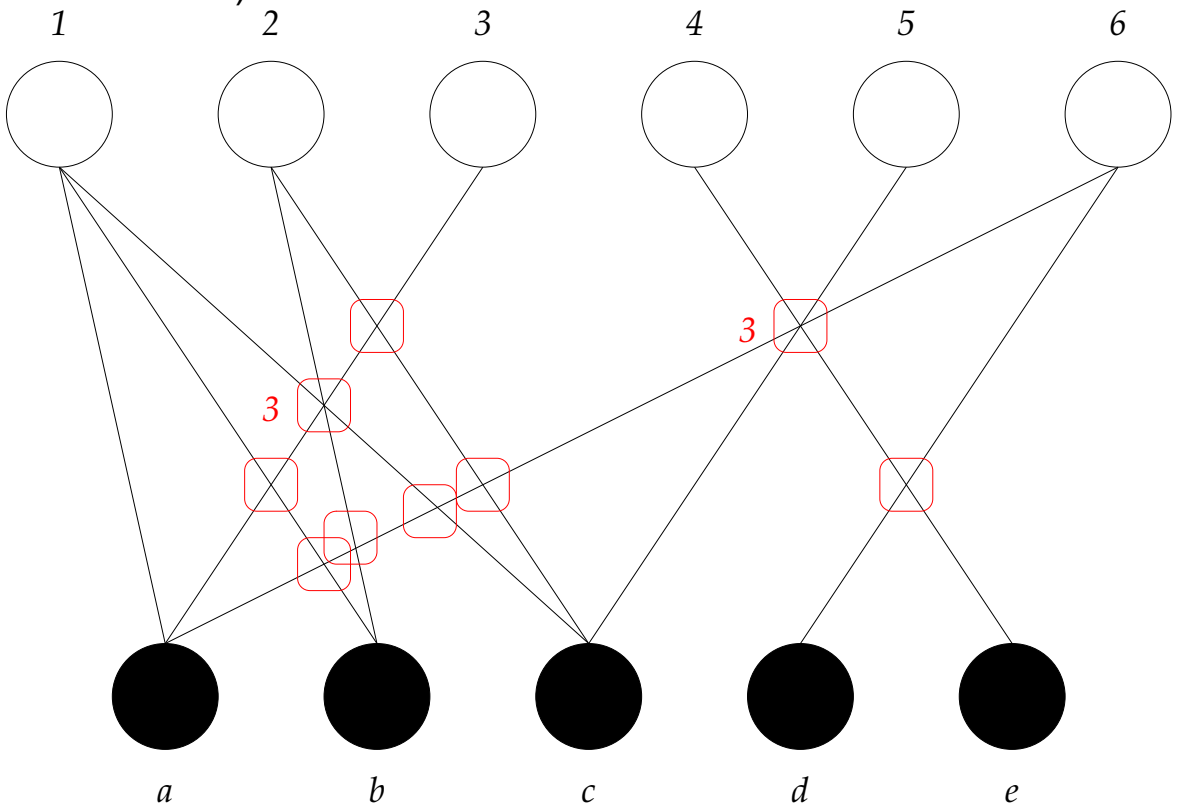
Testergebnisse Blough's Speicherfehlermodell in Testläufen (Bai, F 2008)

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

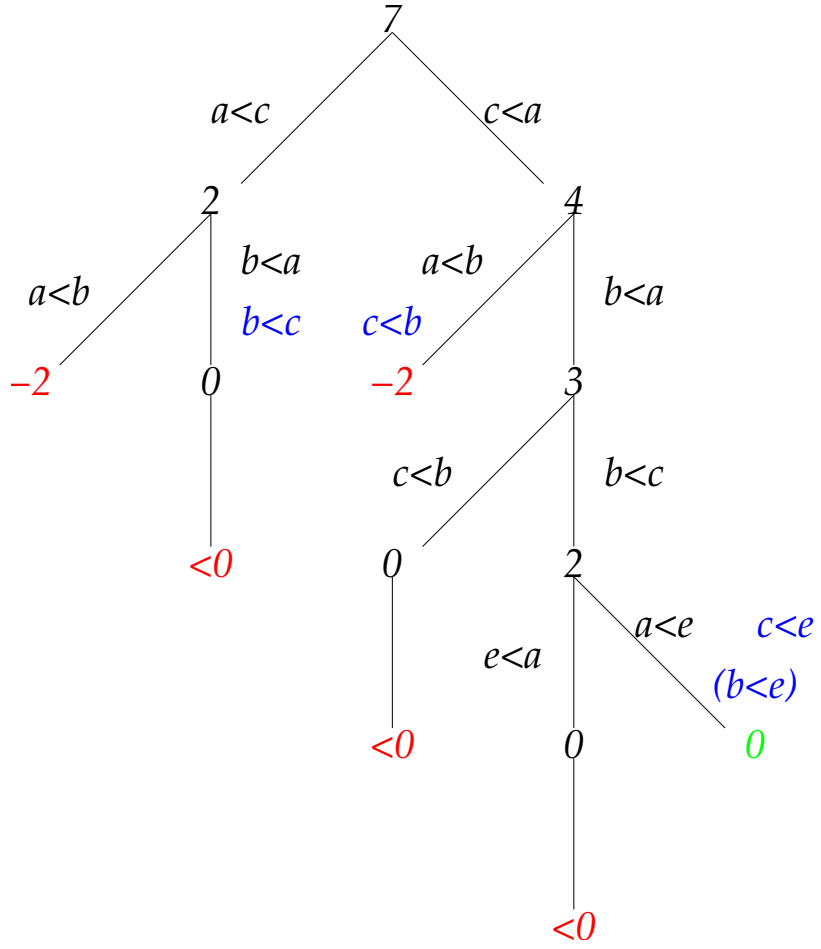
Alg.2: $\mathcal{O}^*(1.46^{k_1+k_2})$ Alg.3: $\mathcal{O}^*(1.40^{k_1+k_2})$ Ergebnisse aus Diplomarbeit Bai.
Hinweis: Schneller Abbruch möglich durch Bestimmung einer kleinstmöglichen Knotenüberdeckung im bipartiten Graphen über Matching-Techniken, s. VL Näher. Dies ist wiederum eine Relaxation.

Zeichnen von bipartiten Graphen in der Ebene

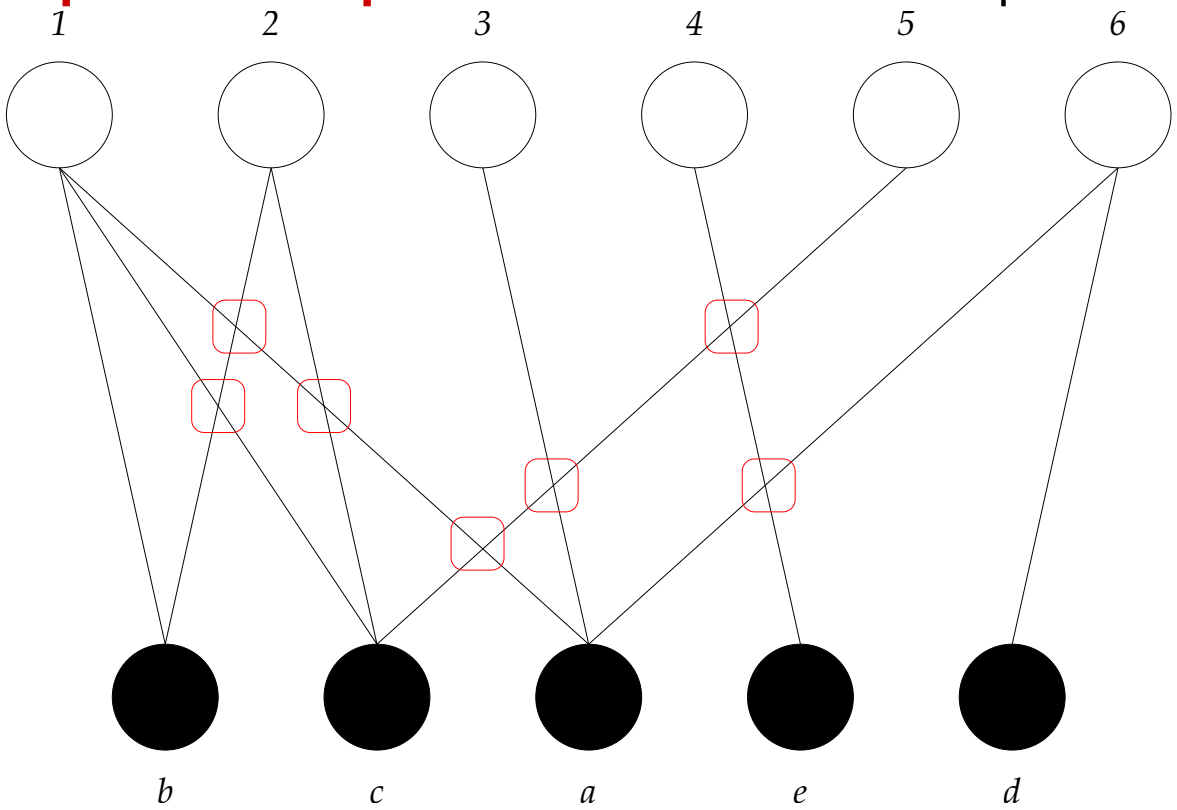
(Dujmović, F, Kaufmann 2008)



Zeichnen von bipartiten Graphen in der Ebene: Wieder binäres Verzweigen



Zeichnen von bipartiten Graphen in der Ebene: Das Optimum



Hinweis: Hierarchisches Graphenzeichnen Thema bei Prof. Diehl.

Bessere Verzweigungen am Beispiel VC:

Beobachte: Beim Verzweigen an Kante $e = \{x, y\}$ wird zwar zunächst der Fall $x \in C$ vollständig betrachtet, sodann aber im anderen Fall $y \in C$ "tiefer" im Suchbaum der Fall $x \in C$ möglicherweise nochmals.

$\rightsquigarrow$ Alternative Interpretation der Verzweigung: 1. Fall $x \in C$, 2. Fall $x \notin C$.

Beobachte: Hat x einen Grad $\delta(x) > 1$, so ergibt sich eine bessere Rekursion. Für Graphen mit Maximalgrad Eins gibt es dann einen Polynomzeitalgorithmus. Ebenso für Graphen mit Maximalgrad Zwei (wie sehen diese Graphen aus?)!

$\rightsquigarrow$ Blattanzahlabschätzung: $T(k) \leq T(k-1) + T(k-3)$, d.h., $T(k) \in \mathcal{O}(1.4656^k)$.

Datenreduktionen am Beispiel VC:

Es gibt vier einfache Regeln:

- Lösche Knoten vom Grad Null.
- Hat x Grad Eins, so lösche $N[x]$ und reduziere k um Eins.
- Hat x Grad größer k, so lösche x und reduziere k um Eins.
- Ist keine der obigen Regeln anwendbar und hat der Graph mehr als k^2 Kanten, so antworte NEIN.

Die Regeln lassen sich erschöpfend in Polynomzeit anwenden und führen zu einem Problemkern. Kombiniert mit dem Suchbaum ergeben sich (noch) bessere Abschätzungen, insbesondere wenn man noch die (kompliziertere) Grad-2-Reduktionsregel hinzunimmt.