Parameterisierte Algorithmen

SoSe 2013 in Trier

Henning Fernau
fernau@uni-trier.de

Parameterisierte Algorithmen | Gesamtübersicht

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

Organisatorisches

Vorlesung: Montag 10-12 im H6
Vorschlag: 10.06-11.35 ab 2. VL-Woche

Übungen (Daniel Meister): Mittwoch 8-10 im H7

Meine Sprechstunde: DO, 13-14 Uhr

Kontakt: fernau,daniel.meister@uni-trier.de

Hausaufgaben / Schein ?! n.V. (Master ?! $\rightarrow$ mündliche Prüfung)

Einführung 1: Motivation

Das “Gute”: *P* (klassische Sicht)

Erinnerung: *P* ist die Klasse von (Kodierungen von) Entscheidungs-Problemen, die von *deterministischen* Turing-Maschinen in Polynomzeit akzeptiert werden können.

Beispiele: (Achtung: nicht alles Entscheidungs-Probleme!)

- Sortieren von n Gegenständen: $\mathcal{O}(n \log(n))$ (nicht bei TM-Modell).
- Wortproblem für kontextfreie Sprachen: $\mathcal{O}(n^3)$.
- Matrixmultiplikation: $\mathcal{O}(n^3)$.

Das “Böse”: *NP*-Härte

Erinnerung: *NP* ist die Klasse von (Kodierungen von) Entscheidungs-Problemen, die von *nichtdeterministischen* Turing-Maschinen in Polynomzeit akzeptiert werden können.

Ein *NP*-Entscheidungsalgorithmus arbeitet oft wie folgt:

1. eine Lösung wird geraten;
2. die Gültigkeit der gefundenen Lösung *überprüft* ein *deterministischer* Algorithmus.

Beispiel: Erfüllbarkeitsproblem SAT: (1) Rate Belegung, (2) prüfe, ob Formel erfüllt ist.
Daraus deterministischer Alg.: Ersetze (1) durch systematisches Durchprobieren.

Noch bekannt? *NP*-vollständig versus *NP*-hart?

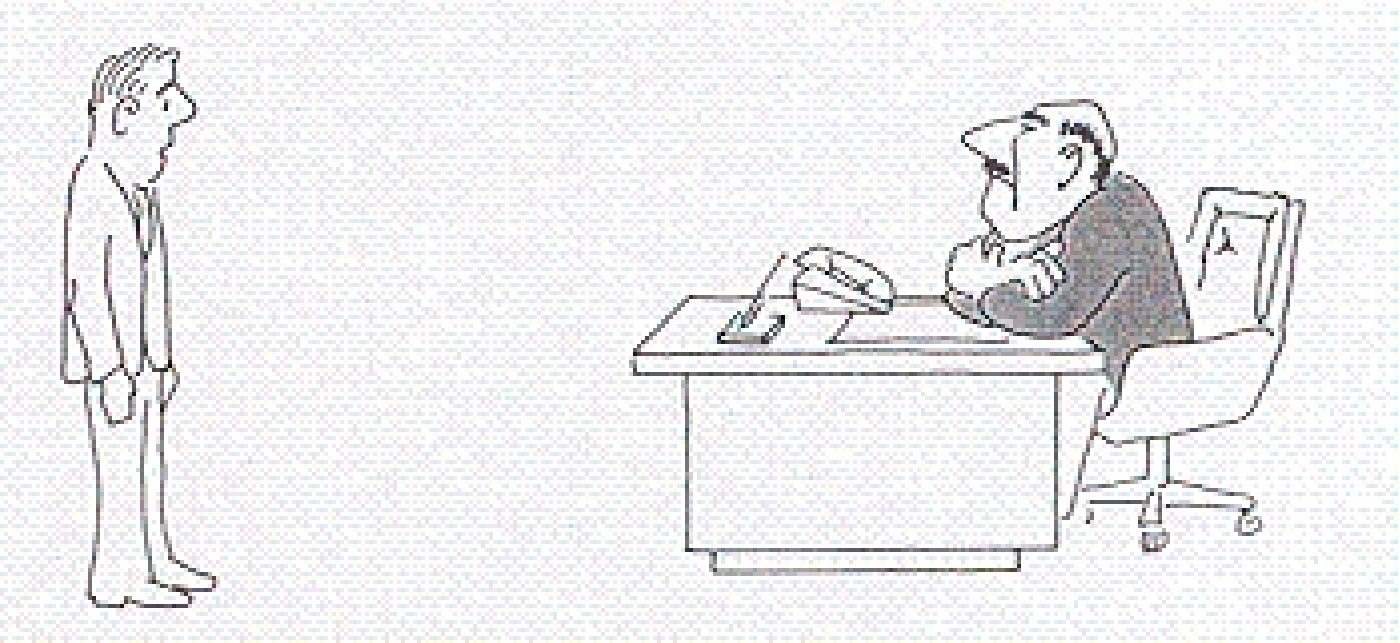
Motivation

Viele interessante Probleme (aus der Praxis!) sind NP-hart

$\Rightarrow$ wohl keine Polynomialzeitalgorithmen sind zu erwarten.

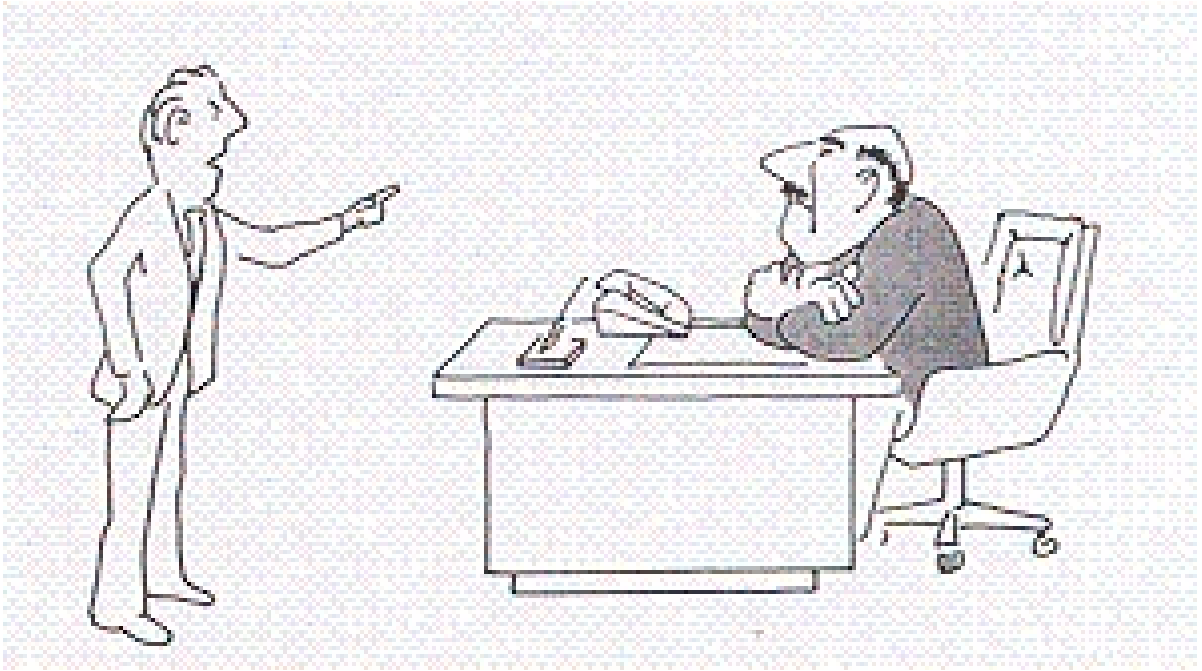
Motivation

siehe `http://max.cs.kzoo.edu/~kschultz/CS510/ClassPresentations/NPCartoons.html` wiederum aus Garey / Johnson



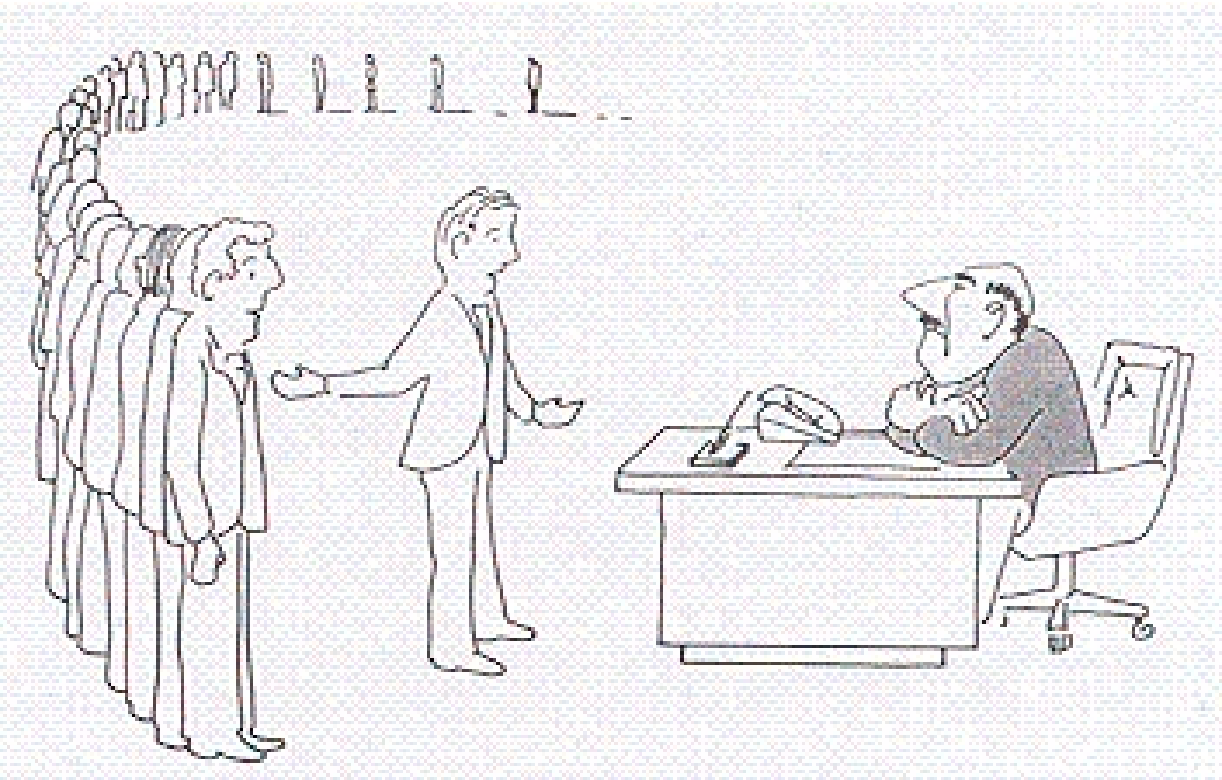
Sorry Chef, aber ich kann für das Problem keinen guten Algorithmus finden. . .

Die beste Antwort wäre hier aber. . .



. . . Ich kann aber beweisen, dass es für das Problem keinen guten Algorithmus geben kann !

Was die Komplexitätstheorie statt dessen liefert. . .



. . . Ich kann aber beweisen, dass das alle anderen auch nicht können !

Das Credo: $\mathrm{P} \subsetneq \mathrm{NP}$

Folgerung: Für *NP*-harte Probleme “glaubt man” nicht an Polynomzeitalgorithmen zu ihrer Lösung.

$\rightsquigarrow$ Exponentialzeitalgorithmen erscheinen unvermeidlich für

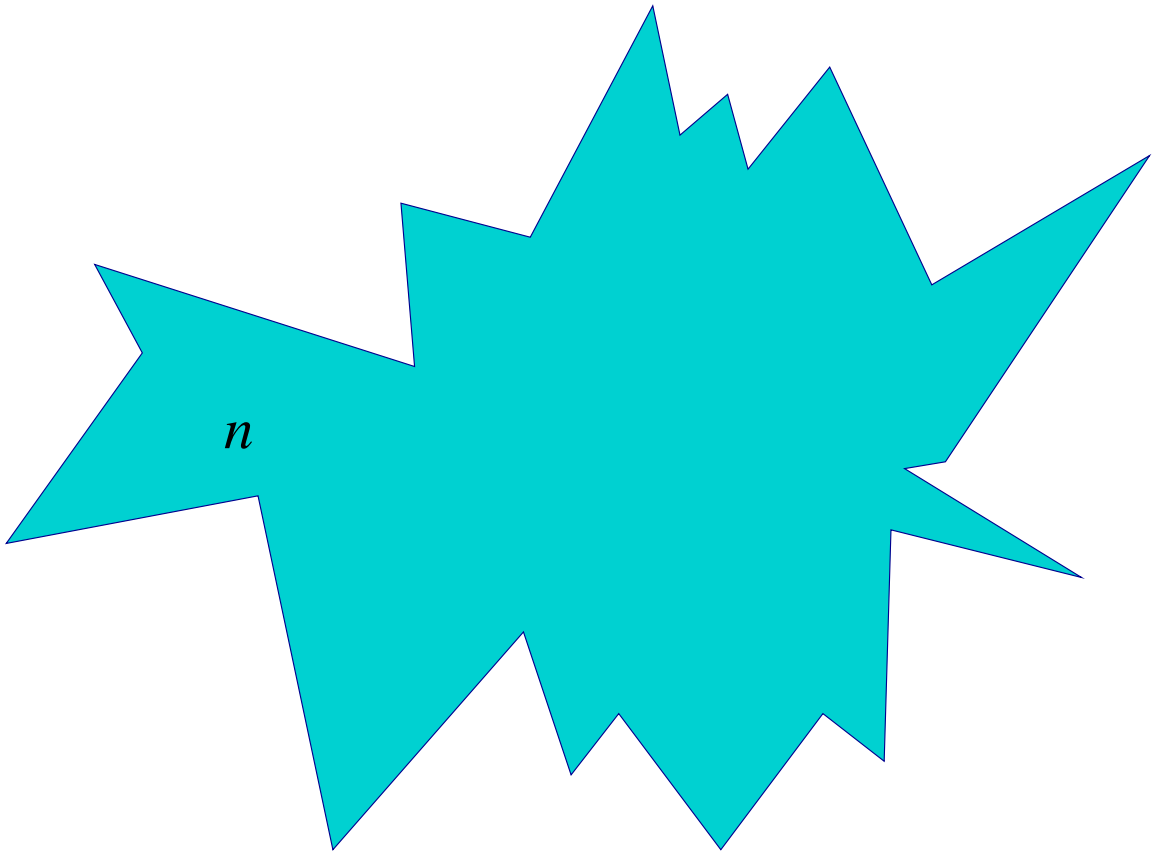
exakte Lösungen *NP*-harter Probleme.

Beispiele für *NP*-vollständige Probleme

- SATisfiability (Erfüllbarkeit logischer Formeln: Satz von Cook)

- Viele Graphprobleme:
  - 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)



Auswege:

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

- Näherungsalgorithmen:
  Polynomzeit; Güteschranken; nicht optimal

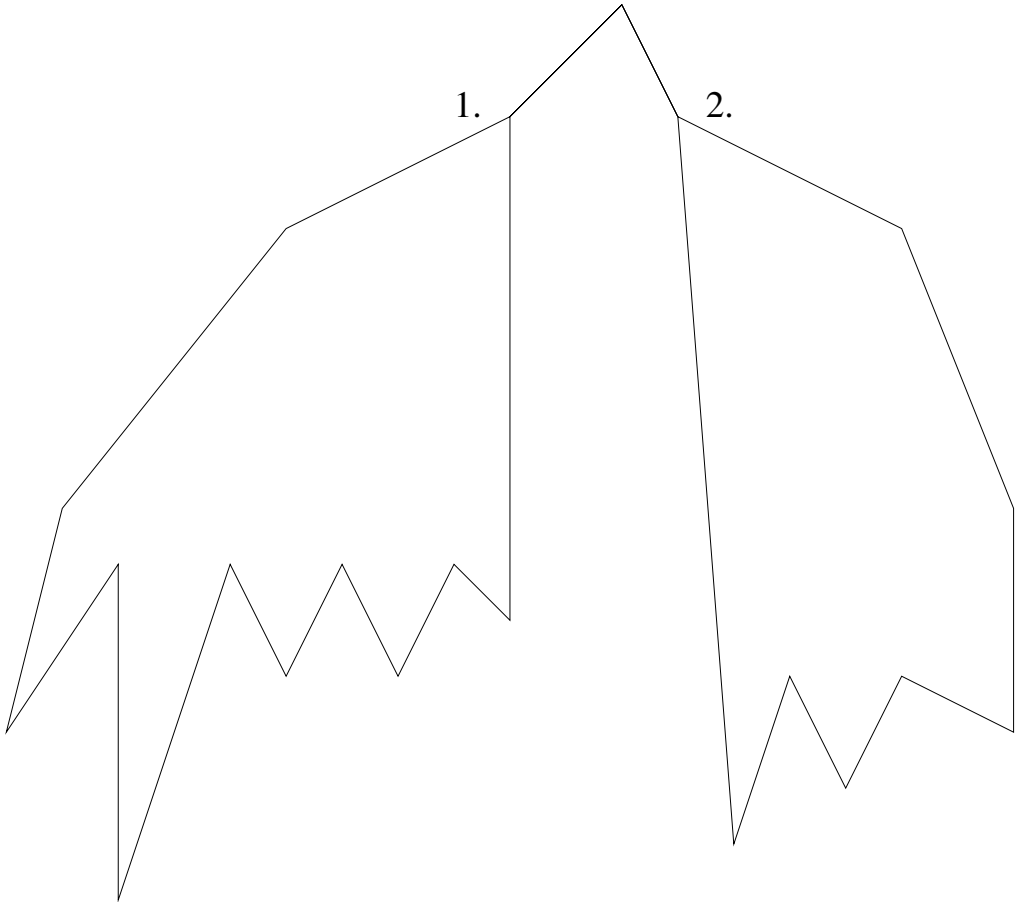
- Heuristiken, u.a. Reduktionsregeln

- Randomisierung

Exakte Wege: praktisch betrachtet

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

Suchbäume:



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

Sehr wichtig: Komplexitätsabschätzung

Thm. 3-HITTING SET ist lösbar in Zeit $\mathcal{O}(2.17\ldots^k + kn)$.

$k = \ldots$	10	15	20	25	30
3^k	60 ms	15 s	3487 s $\approx$ 1 h	847289 s $\approx$ 10 d	57192 h $\approx$ 6.5 y
2.18^k	3 ms	0.12 s	6 s	290 s $\approx$ 4 min	4 h

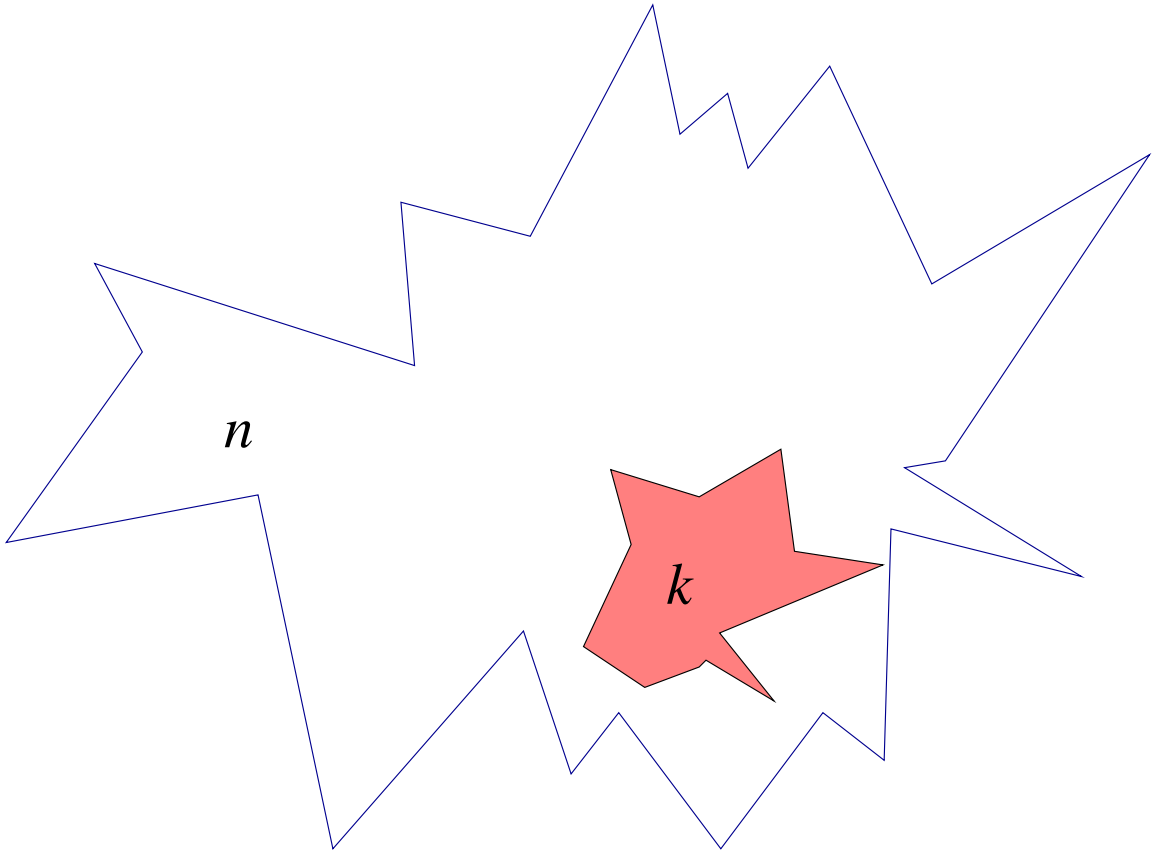
(Annahme: 10^6 Suchbaumknoten pro Sek.)

Einführung 2: Parameterisierte Algorithmen im Überblick

Parameterisierte Algorithmen

- Die parameterisierte Sicht
- (Sprechweisen aus der Graphentheorie)
- Einführungsbeispiele
- Beispiel HITTING SET

The Curse of Combinatorics Eine zweidimensionale Sicht



Ziel:

- exakte Algorithmen mit
- Laufzeitgarantien

Methoden: (u.a.)

- “Problemkerne” (Datenreduktion)
- Suchbäume

Parameterisierte Algorithmik: kurz gefasste Idee

Ein **parameterisiertes Problem** ist ein Entscheidungsproblem, das eine explizite Parameterisierung besitzt, (strenger) formalisiert durch eine Funktion, die jeder gültigen Eingabe x eine Zahl k (ihren Parameter) zuordnet.

Anders ausgedrückt ist ein parameterisiertes Problem eine Sprache $L \subseteq \Sigma^* \times \mathbb{N}$ (bei geeigneter fester Kodierung).
$(x, k) \in L$ gdw. die Instanz x mit Parameter k ist eine JA-Instanz für das Problem.

Parameterisierte Algorithmik: kurz gefasste zentrale Definition

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

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

entscheidet (f bel., p Polynom).

Satz. Gleichwertig hiermit ist: Es gibt einen Problemkern (x', k') zu Instanz (x, k) mit $k', |x'| \in \mathcal{O}(g(k))$ für g beliebig.
Die zugehörige Problemkernreduktion ist in Polynomzeit berechenbar.

Wahl des Parameters ist zunächst willkürlich für ein Entscheidungsproblem.

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.

Kurzer Exkurs: Graphen

Graph: $G = (V, E)$
V: (endliche) Knotenmenge (vertices)
E: Kantenmenge (edges)
E ist binäre Relation auf V, d.h., $E \subseteq V \times V$.
Ist E symmetrisch, so ist G ein ungerichteter Graph.
Dann E auffassbar als Teilmenge von 2^V (siehe Hypergraphen).
Übliche Kantennotationen: (x, y), xy, $x\vec{y}$, $\{x, y\}$.
Nicht erfasst in dieser Formalisierung: Mehrfachkanten.
Schlingen: eine Kante xx.
(Ein Graph ohne Schlingen und Mehrfachkanten heißt auch schlicht.)
Ein Graph mit Mehrfachkanten heißt auch Multigraph.

Nachbarschaft und Grad

$N(v)$: offene Nachbarschaft,
d.h., Menge aller Nachbarn von v,
d.h., $N(v) = \{u \mid \exists e \in E(G) : \{u, v\} \subseteq e\}$.
$N[v] := N(v) \cup \{v\}$ ist die abgeschlossene Nachbarschaft von v.

$\deg(v)$ bezeichnet den Grad (degree) (oder die Valenz) des Knotens v, d.h., $\deg(v) = |N(v)|$.
Für $X \subseteq V$ benutzen wir auch $N(X) = \bigcup_{x \in X} N(x)$ und $N[X] = N(X) \cup X$.

Spezielle Mengen

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

$C \subseteq V$ heißt (Knoten-)Überdeckung (vertex cover) gdw. $\forall e \in E \exists v \in C : v \in e$.
$C \subseteq E$ heißt Kantenüberdeckung (edge cover) gdw. $\forall v \in V \exists e \in C : v \in e$.
$I \subseteq V$ heißt unabhängig (independent, stable) gdw. $I \cap N(I) = \emptyset$.
$D \subseteq V$ ist eine dominierende (Knoten-)Menge gdw. $N[D] = V$.

Schwierige Graphprobleme

VC Ggb. Graph $G = (V, E)$ und Zahl k, gibt es eine Knotenüberdeckung C für G mit $|C| \leq k$?

DS Ggb. Graph $G = (V, E)$ und Zahl k, gibt es eine dominierende Menge D für G mit $|D| \leq k$?

IS Ggb. Graph $G = (V, E)$ und Zahl k, gibt es eine unabhängige Menge I in G mit $|I| \geq k$?

Parameter Knotenzahl $n \rightsquigarrow$ "exakte Exponentialzeit-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 "ungefähr" auf $\mathcal{O}^*(1.5^n)$ abschätzen.

Solche exakten Algorithmen können von praktischem Interesse sein.

Parameter Größenschranke k für die fragliche Menge $\leadsto$ Standardparameter

Beispiel VC (rekursiv $\leadsto$ Suchbaum):

SolveVC Eingabe: $G = (V, E), k$
Falls $E = \emptyset$: fertig! (Lösung gefunden)
Falls $k < 0$: fertig! (keine Lösung).
Wähle $e = xy \in E$.
Betrachte (rekursiv) zwei Fälle:
(a) x ist in Überdeckung: SolveVC$(G - x, k - 1)$.
(b) y ist in Überdeckung: SolveVC$(G - y, k - 1)$.

Literatur (besondere Empfehlungen, da sehr algorithmisch ausgerichtet +)

R. Downey / M. Fellows: Parameterized Complexity. Springer, 1999.

H. Fernau (+): Parameterized Algorithmics: A Graph-Theoretic Approach (auf meiner Homepage “insgeheim” unter habil.pdf zu erhalten)

J. Flum / M. Grohe: Parameterized Complexity Theory. Springer, 2006.

R. Niedermeier (+): Invitation to Fixed-Parameter Algorithms. Oxford University Press, 2006.

Skript: Parametrisierte Algorithmen von R. Niedermeier / J. Alber (leicht verändert von J. Gramm) in Tübingen.

Foliensatz von P. Rossmanith in Aachen.

Parameterisierte Algorithmen

- Die parameterisierte Sicht
- (Sprechweisen aus der Graphentheorie)
- Einführungsbeispiele
- Beispiel HITTING SET

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

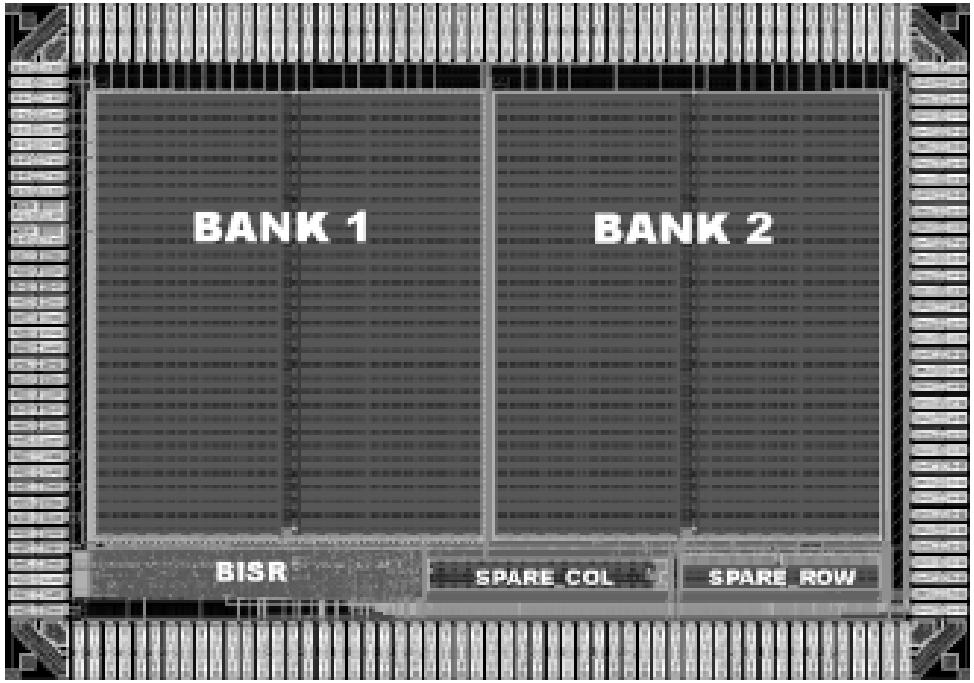
Speicherrekonfigurierung

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

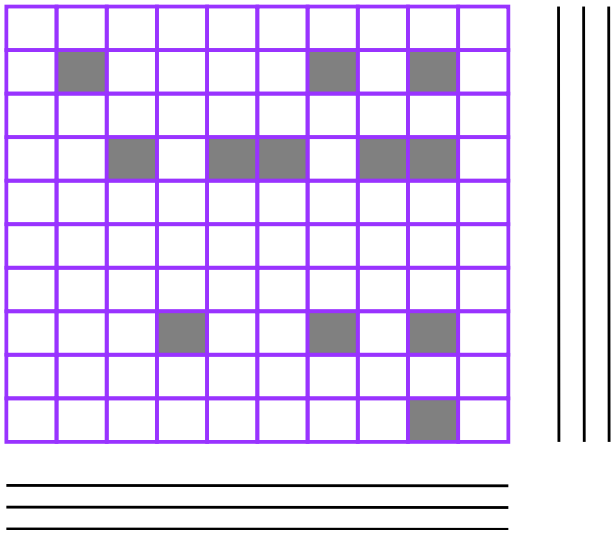
- Häufige Strategie:
  Bevorraten mit Ersatz-Zeilen und -Spalten.
  Diese kommen zum seit den 70er Jahren mit Variationen zum Einsatz.
  Die (für uns hier unbedeutende) eigentliche Reparaturtechnik ändert sich.

- 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

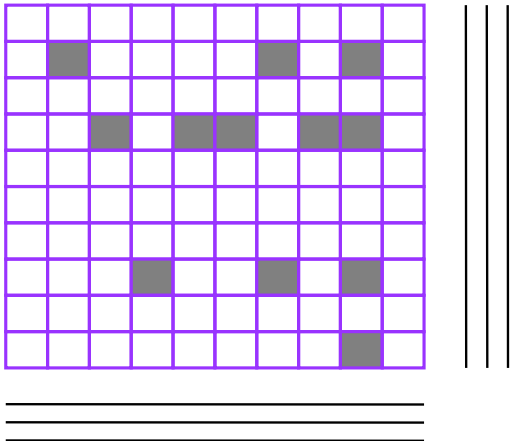


Quadrate: Speicherzellen
Linien: Austauschstücke.
Graue Zellen seien fehlerhaft.
Höchstens drei Zeilen
und drei Spalten
stehen als Ersatz
zur Verfügung.

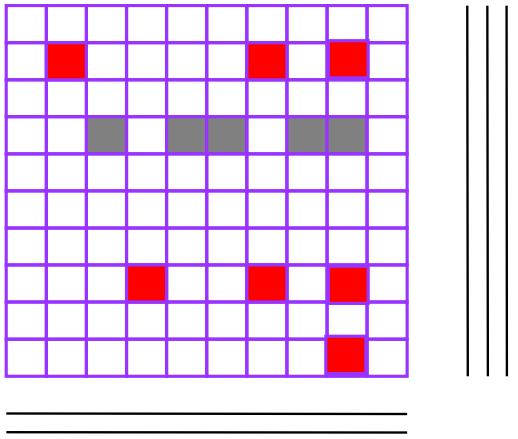
Speicherrekonfigurierung: Auswahl einer Zeile zum Ersetzen

$\rightsquigarrow$ 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?

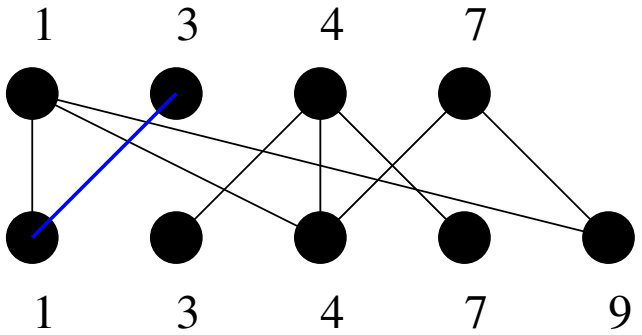
Alternative Formalisierung

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 $i = 1, 2$?

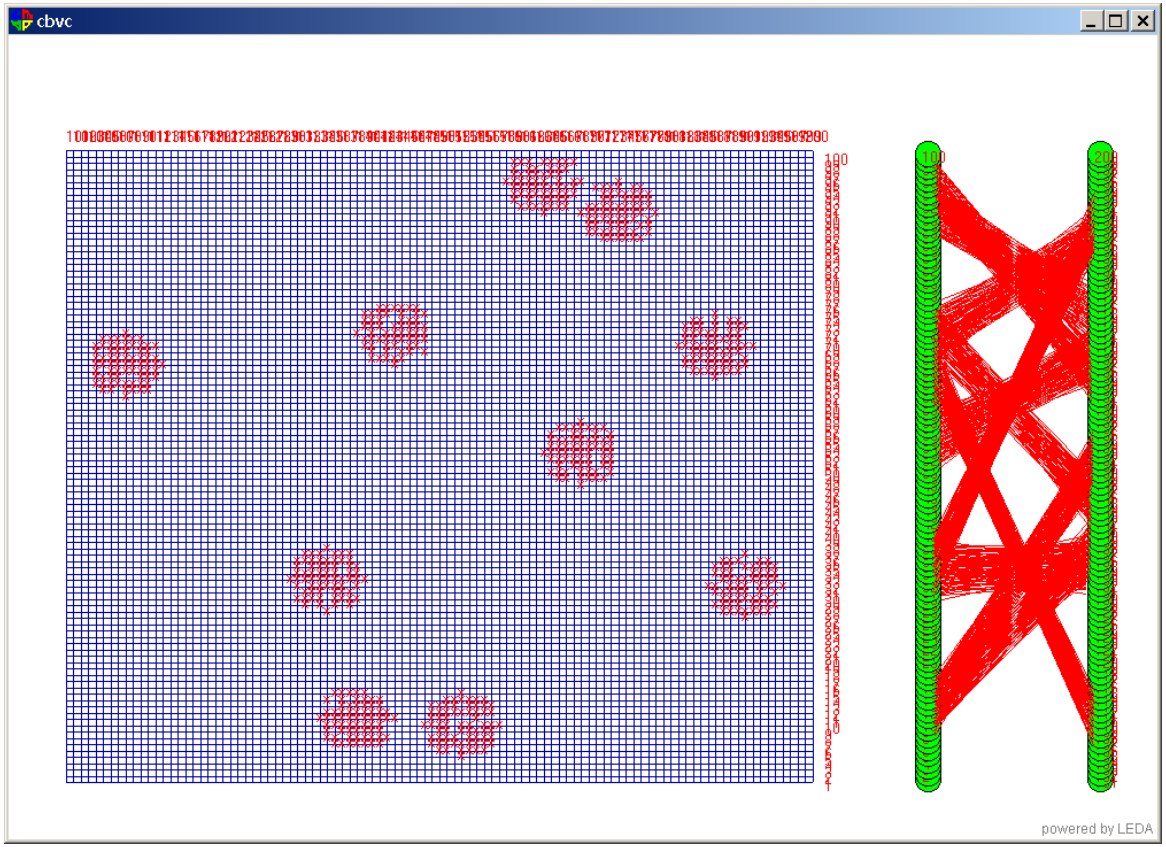
Beie Probleme sind “offenbar” äquivalent.
$\rightsquigarrow$ VC-Suchbaumalgorithmus ist auch hier anwendbar
$\rightsquigarrow$ CBVC in Zeit $\mathcal{O}^*(2^{k_1+k_2})$ lösbar (lässt sich auch weiter verbessern)

Ein kleines Beispiel

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



Blough's Speicherfehlermodell



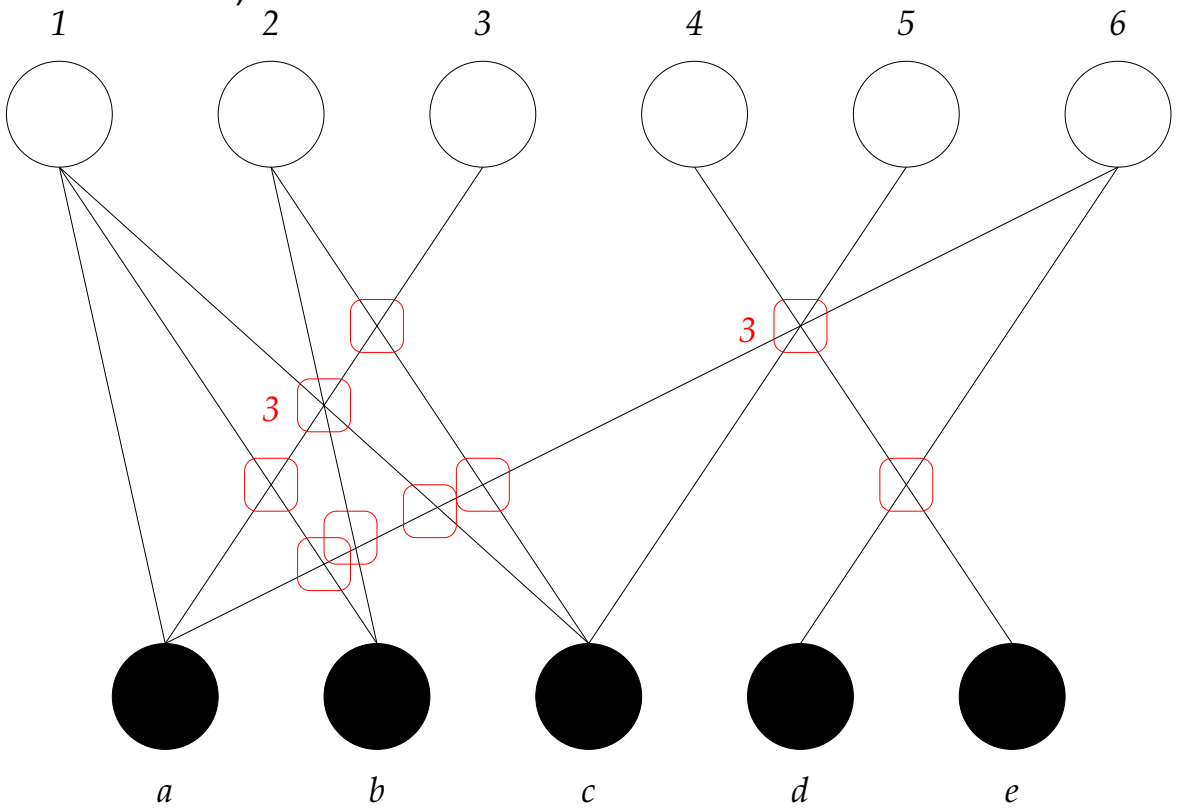
Testergebnisse Blough's Speicherfehlermodell in Testläufen (Bai, F 2008)

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

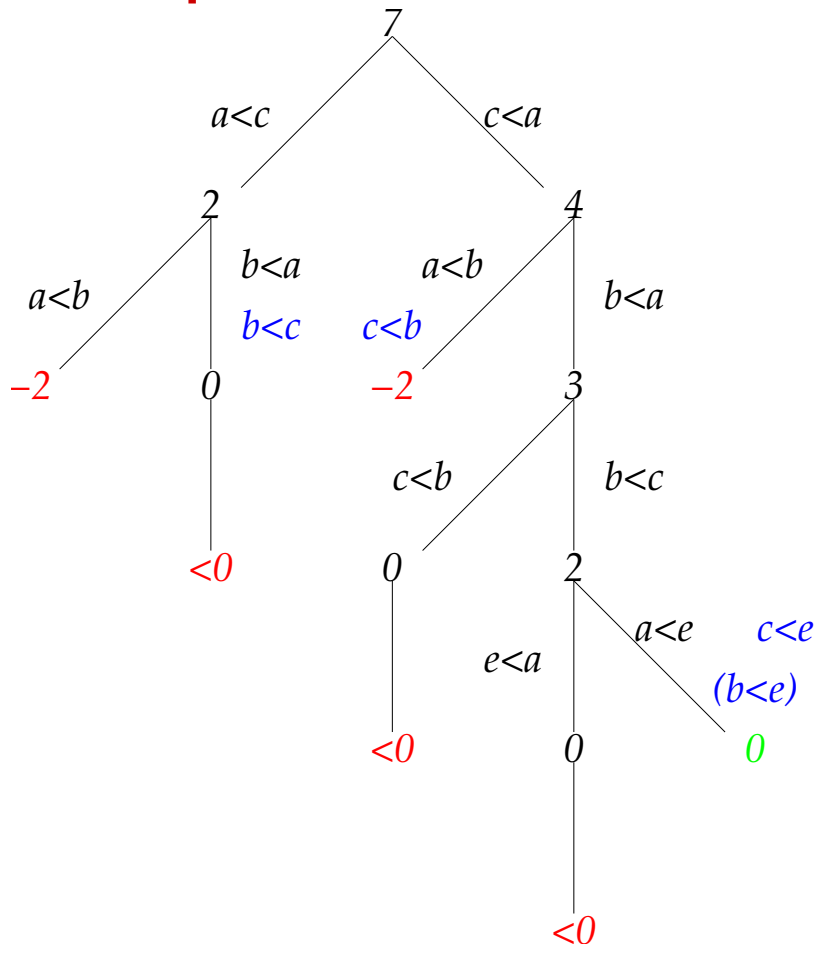
Alg.2: $\mathcal{O}^*(1.46^{k_1+k_2})$ Alg.3: $\mathcal{O}^*(1.40^{k_1+k_2})$

Zeichnen von bipartiten Graphen in der Ebene

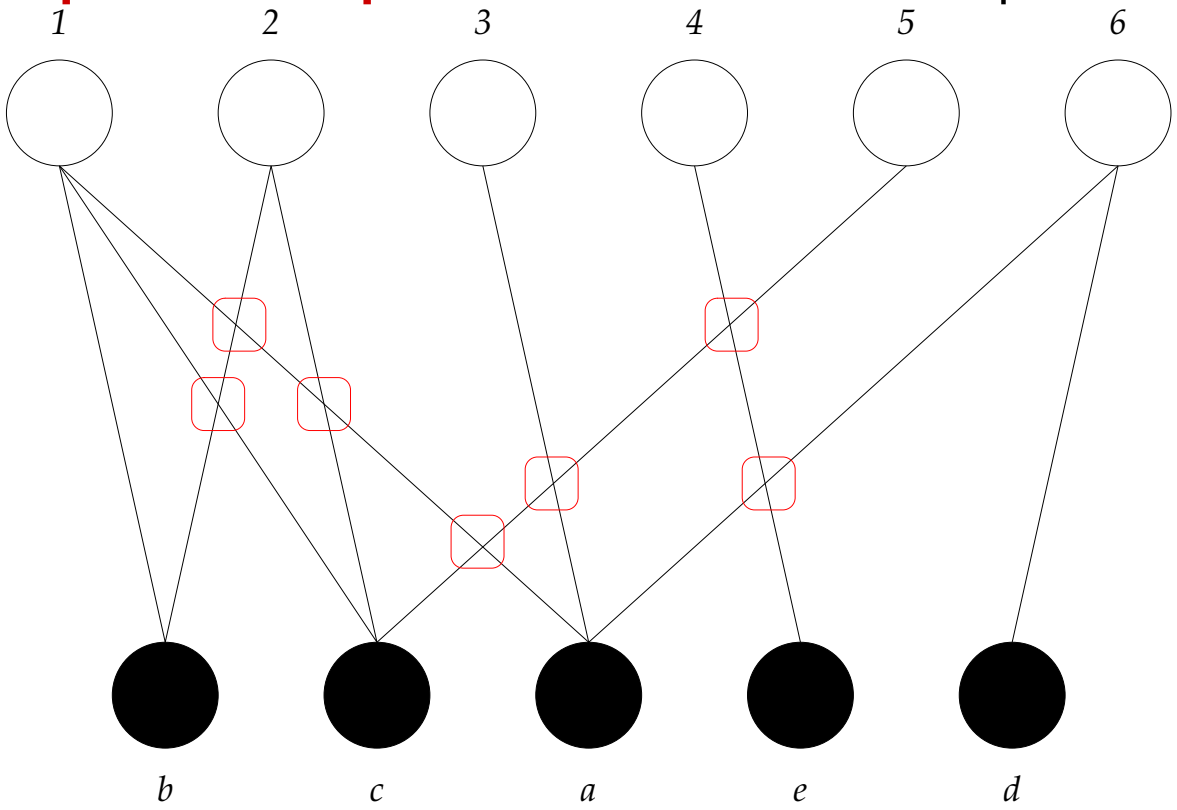
(Dujmović, F, Kaufmann 2008)



Zeichnen von bipartiten Graphen in der Ebene: Wieder binäres Verzweigen



Zeichnen von bipartiten Graphen in der Ebene: Das Optimum



Abschließende Diskussion

Die parameterisierte Sicht ermöglicht
manchmal praktikable Ansätze für NP-harte Probleme.

Was sind “gute Parameterisierungen” ?

Vergleich mit anderen Ansätzen

Praktische Erfahrungen

Parameterisierte Algorithmen

- Die parameterisierte Sicht
- (Sprechweisen aus der Graphentheorie)
- Einführungsbeispiele
- Beispiel HITTING SET

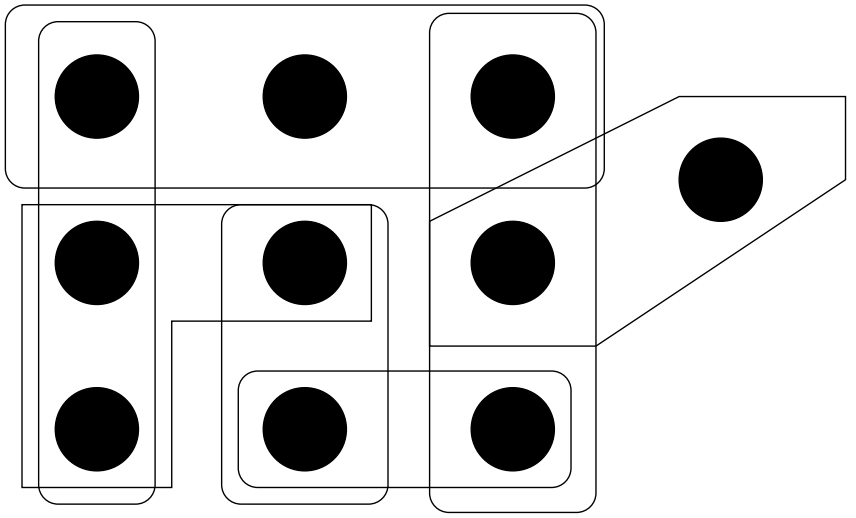
Exkurs: Hypergraphen

Zur Definition

Hypergraph $G = (V, E)$:
V: Knotenmenge
$E \subseteq 2^V$: Kantenmenge

Spezialfall: $\forall e \in E : |e| \leq 2 \rightsquigarrow$
(ungerichteter) Graph

Ein Beispiel



3-HITTING SET: Knotenüberdeckung auf Hypergraphen

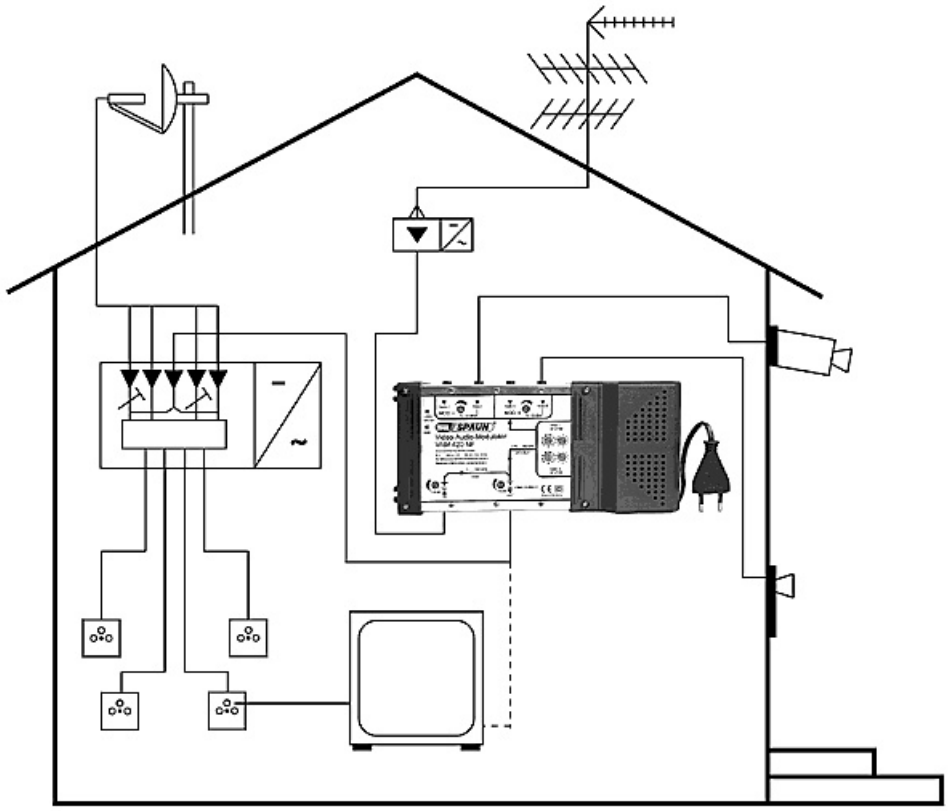
Eingabe: Hypergraph $G = (V, E)$ mit *Kantengröße* höchstens 3: $\forall e \in E(|e| \leq 3)$

Parameter: k

Frage: Gibt es eine Knotenüberdeckung (hitting set) mit höchstens k Knoten:

$$\exists C \subseteq V\,(|C| \leq k \wedge \forall e \in E(C \cap e \neq \emptyset))\,?$$

Motivation: Systemanalyse á la Reiter



Was ist ein System? (nach R. Reiter)

- Systembestandteile (Komponenten) C

- Systembeschreibung (wie? $\leadsto$ Logik) SD:
  Aussagen über erwartetes Systemverhalten,
  d.h., Beziehungen zwischen den Komponenten.

- beobachtetes Systemverhalten (Observationen) OBS

Was ist ein fehlerbehaftes System?

- spezielles Prädikat $\mathrm{ab}(\mathrm{c})$ für jede Komponente $\mathrm{c} \in \mathrm{C}$:
  kennzeichnet abnormes Verhalten (Fehler)
  SD enthält auch Aussagen der Form:
  "Wenn $\mathrm{ab}(\mathrm{c})$, dann gilt:..." bzw.
  "Wenn $\neg\mathrm{ab}(\mathrm{c})$, dann gilt:..."

- ein System $(\mathrm{C}, \mathrm{SD}, \mathrm{OBS})$ ist *fehlerbehaftet*, wenn in

$$\mathrm{SD} \cup \mathrm{OBS} \cup \{\neg\mathrm{ab}(\mathrm{c}) \mid \mathrm{c} \in \mathrm{C}\}$$

  ein Widerspruch zu erkennen ist.

Konfliktmengen und Diagnosen

Eine *Konfliktmenge* ist eine (minimale) Menge C' von Komponenten, so dass in

$$SD \cup OBS \cup \{\neg ab(c) \mid c \in C'\}$$

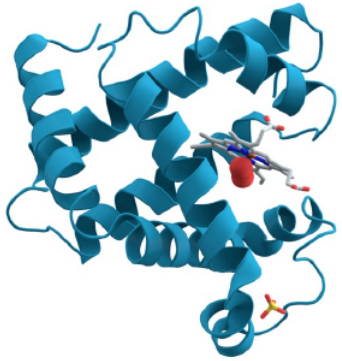
ein Widerspruch zu erkennen ist.

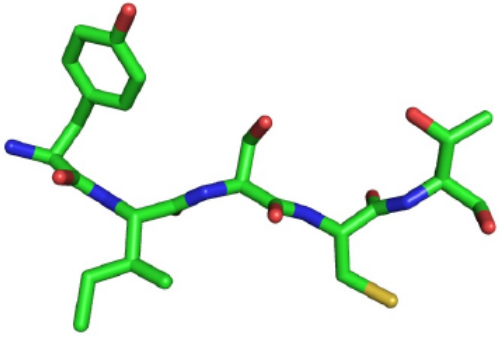
Eine *Diagnose* ist eine möglichst kleine Menge C' von Komponenten, so dass $C \setminus C'$ *keine* Konfliktmenge ist.

Übersetzung in Hitting Set:
Die Hypergraphknoten sind die Komponenten,
die Konfliktmengen sind die Kanten,
die Diagnose die Überdeckungsmenge.

Motivation: Analyse von Proteinmischungen

Möchte man wissen, welche Proteine in einer Mischung enthalten sind, so zergliedert man die (langen) Proteine und stellt fest, welche Peptide (Proteinbestandteile) in der Mischung enthalten waren.





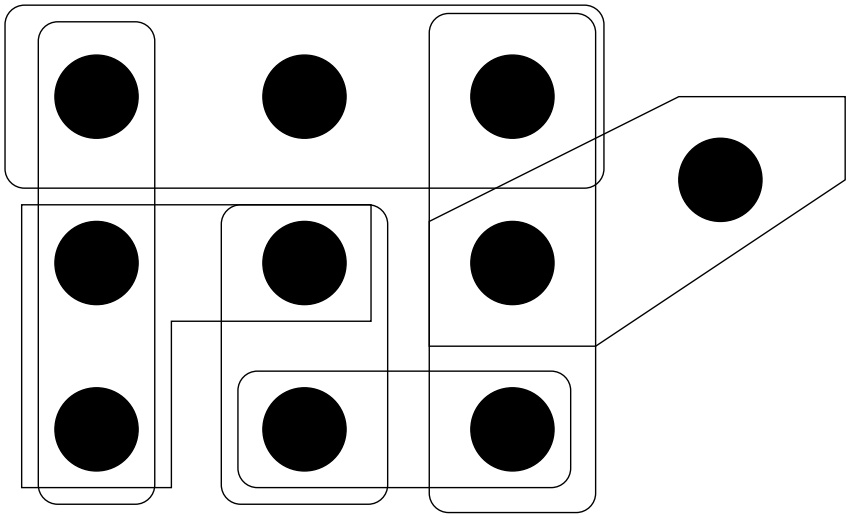
Wie kann man nun die Proteine rekonstruieren?

In einer Datenbank sind die Peptide verschiedenster Proteine gespeichert.

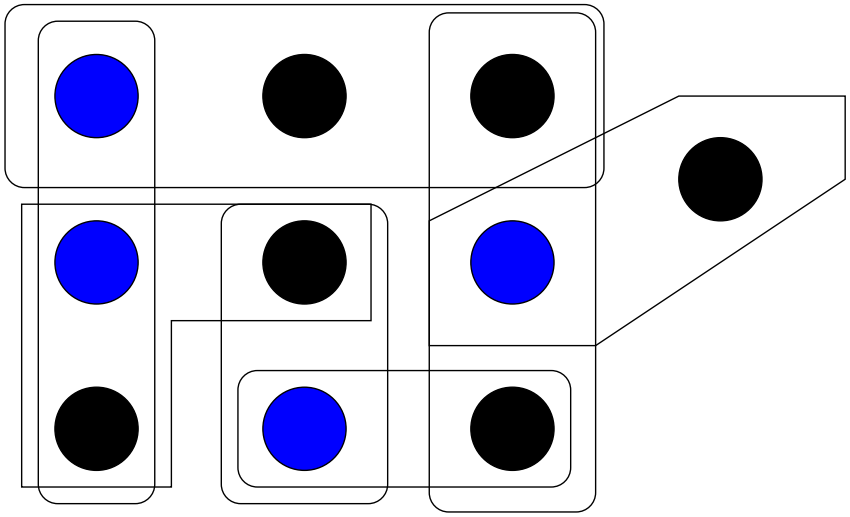
"Occam's Razor" $\rightsquigarrow$ Suche nach der "kleinsten Erklärung", d.h.

finde kleinstmögliche Menge von Proteinen, sodass jedes Peptid (aufgefasst als Menge derjenigen Proteine, die das Peptid enthalten) "getroffen" wird.

Ein abstrakteres Beispiel



Eine kleinste Überdeckung



Datenreduktionsregeln

1. Kantendominierung: $f \subset e. \rightsquigarrow$ entferne e

2. Kleine Kanten: $e = \{v\} \rightsquigarrow v$ kommt ins HS; entferne alle Kanten mit v.

3. Knotendominierung: Ein Knoten x heiße *dominiert* durch einen Knoten y, falls $\{e \in E \mid x \in e\} \subseteq \{e \in E \mid y \in e\} \rightsquigarrow$ entferne x

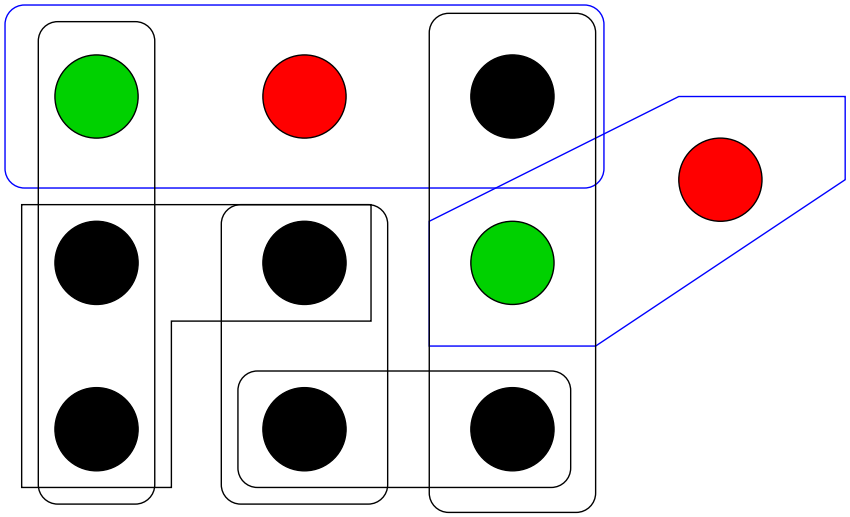
R. S. Garfinkel and G. L. Nemhauser. *Integer Programming*. John Wiley & Sons, 1972.
Oft wiederentdeckt: K. Weihe (Zugnetzoptimierung), R. Niedermeier & P. Rossmanith (param. HS, 2003), ebenso: R. Reiter (Theory of Diagnosis $\rightsquigarrow$ HS Bäume, 1987)

Einschub: Korrektheit von Reduktionsregeln

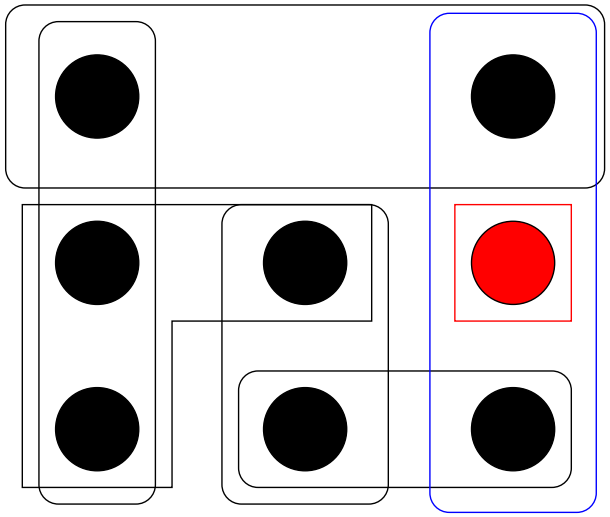
Was muss man beweisen?
Beispiel Kantendominierung.
Lemma: Es sei $G = (V, E)$ ein Hypergraph mit Kanten f, e, sodas $f \subset e$.
Dann gilt:
G hat eine Knotenüberdeckungsmenge der Größe höchstens k gdw.
$G' = (V, E \setminus \{e\})$ hat eine Knotenüberdeckungsmenge der Größe höchstens k.

Aufgabe: Beweisen Sie dieses Lemma und formulieren Sie ähnliche Aussagen, die die Korrektheit der anderen HS-Regeln belegen.

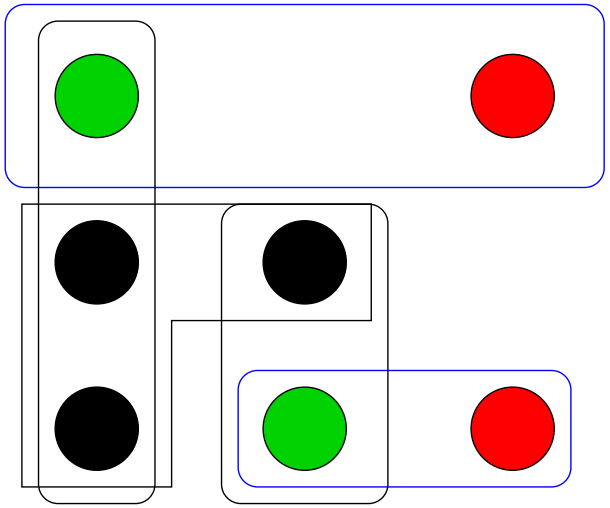
Knotendomination



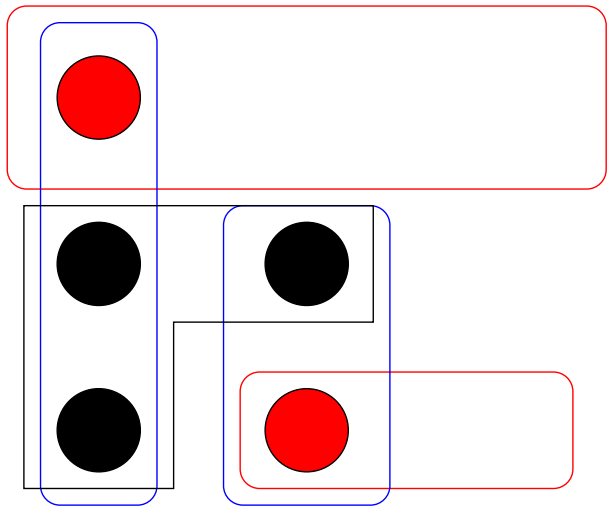
Kantenregeln



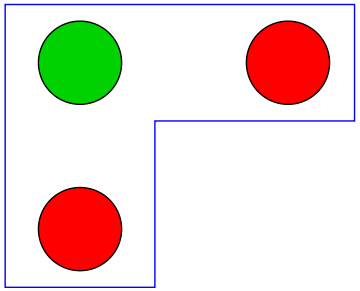
Knotendomination



Kantenregeln



Knotendomination



Verzweigungsregeln (heuristic priorities)

Bevorzuge Verzweigungen bezüglich

1. kleiner Kanten
2. Knoten hohen Grades
3. ...

Der Algorithmus HS(G, k)

1. Wende erschöpfend alle Reduktionsregeln an $\rightsquigarrow (G', k')$, $G' = (V', E')$.

2. **if** $k' \leq 0$ **then** return $(k' = 0 \& E' = \emptyset)$

3. **if possible:** Wähle $x \in V(G')$ gemäß Verzweigungsregeln

4. **if** HS$(G' - E'[x], k' - 1)$ **then** return YES

5. **else** return HS$(G' - x, k')$

Einige Erläuterungen

- $G' - x$ entsteht aus Hypergraph $G' = (V', E')$ durch Löschen von x aus der Knotenmenge V' sowie aus allen Kanten in $E'[x]$.
- $E'[x]$ sammelt alle Kanten aus E', die x enthalten.
- $G' - E'[x]$ löscht aus E' alle Kanten, die x enthalten.
- Die Fallunterscheidung betrachtet, ob x zum HS zählt oder nicht.
- Falls Algorithmus kleine Kante e (der Größe 2) zum Verzweigen findet, wird $x \in e$ gewählt (gemäß Verzweigungsregel). Falls x nicht ins HS aufgenommen wird (2. Fall), so wird insbesondere e zu $e \setminus \{x\}$. Also werden in der Rekursion Reduktionsregeln ausgelöst.

Verzweigungsalgorithmen / Suchbaumalgorithmen

sind immer dann geeignet, FPT-Mitgliedschaft zu zeigen,
wenn stets eine “kleine” Menge von “Kandidaten” gewählt werden kann,
auf denen anschließend verzweigt wird.

In jeder Verzweigung sinkt das Parameterbudget dabei wenigstens um Eins, weil ein Kandidat in die Zielmenge aufgenommen wird.

Haben die Kandidatenmengen stets höchstens d Elemente, so hat ein derartiger Suchbaum höchstens d^{k} Blätter, wenn k das ursprüngliche Parameterbudget angibt.

Diese Vorgehensweise “erklärt” Verzweigungsalgorithmen für das Knotenüberdeckungsproblem in Hypergraphen mit einer Schranke auf deren Kantengröße.

Cluster Editing: Eine abschließende Aufgabe

Ein Graph G heißt Clustergraph, wenn alle seine Zusammenhangskomponenten Cliquen sind.

Zeigen Sie: G ist Clustergraph gdw. es gibt keine Menge von drei Knoten, die einen Weg der Länge zwei induziert.
Benutzen Sie dieses Ergebnis, um einen Suchbaumalgorithmus für das folgende Problem zu entwerfen:

CLUSTER EDIT

Eingabe: ungerichteter Graph $G = (V, E)$
Parameter: ganze Zahl k
Frage: Kann man höchstens k_1 Kanten aus G löschen und höchstens k_2 Kanten in G hinzufügen, sodass $k = k_1 + k_2$ und der entstehende Graph ein Clustergraph ist?