Parameterisierte Algorithmen

WS 2007/08 in Trier

Henning Fernau
fernau@uni-trier.de

Parameterisierte Algorithmen | Gesamtübersicht

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

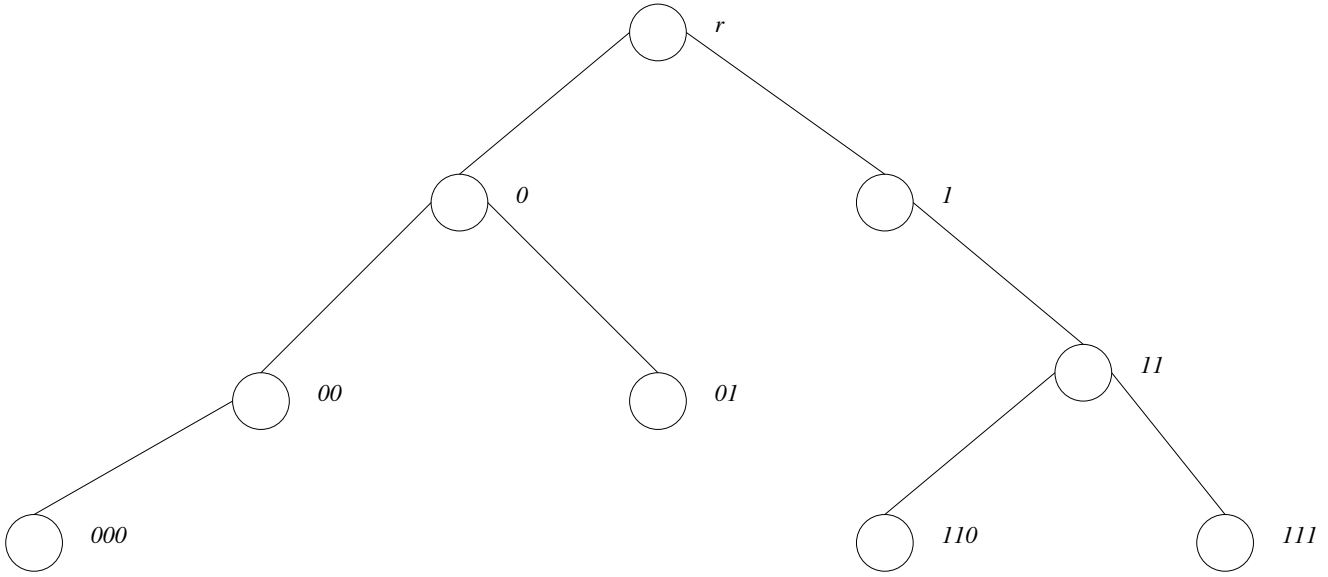
Grundidee: Bäume sind einfach!

Beispiel: Berechne kleinstmögliche dominierende Menge für Bäume!

Grundtechnik: dynamisches Programmieren

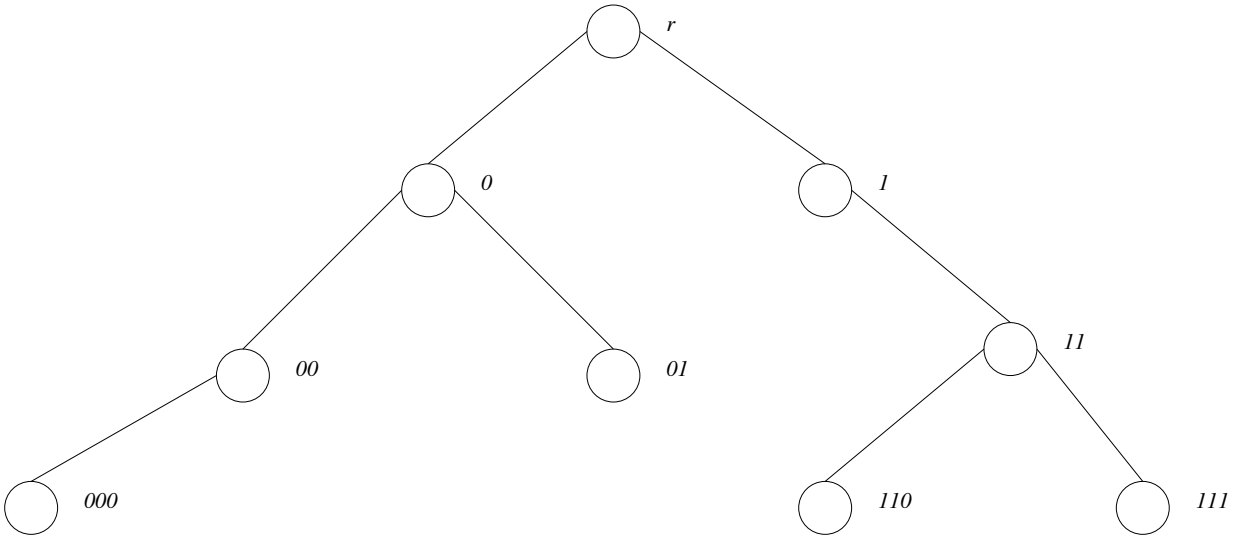
Für MDS sind dafür drei “Zustände” nötig:
— dominierend: 1
— nicht dominierend aber dominiert 0
— nicht dominierend und nicht dominiert $\hat{0}$

Unser Beispielbaum



Bearbeitung: von den Blättern zur Wurzel

Unser Beispielbaum: Wie geht es weiter ?



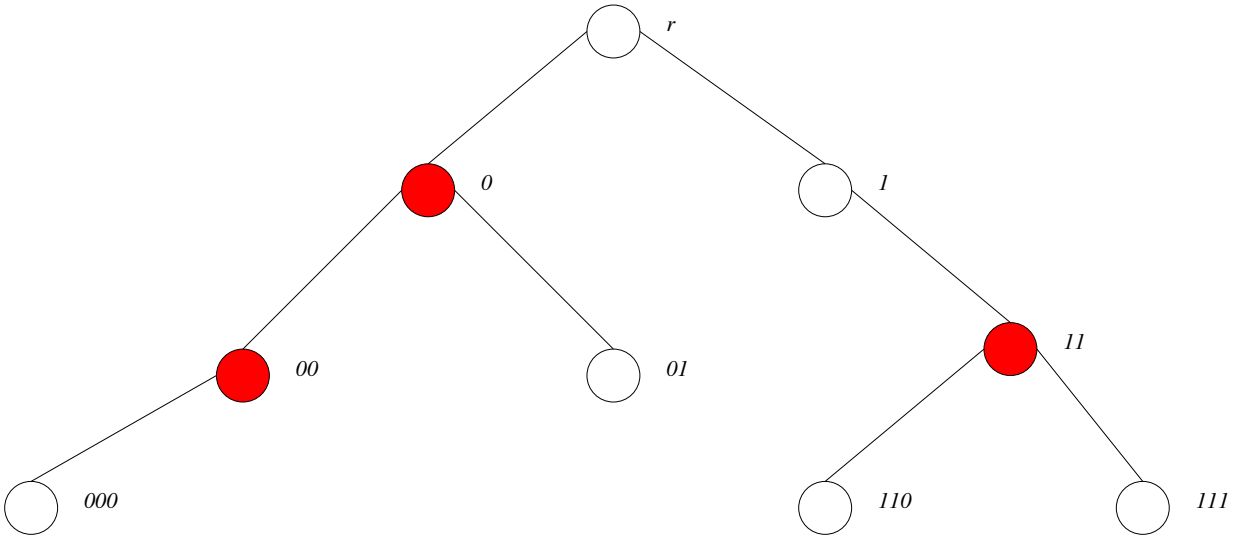
Blatt-Tabellen:

Zustand	Min.
1	1
0	∞
$\hat{0}$	0

$\rightsquigarrow$ Tabellen bei 00 und 11:

Zustand	Minimum
1	1
0	1/2 schlecht
$\hat{0}$	∞

Unser Beispielbaum: Eine kleinstmögliche Lösung ?



Tab. bei 00 & 11:

Zustand	Minimum
1	1
0	1/2 schlecht
$\hat{0}$	∞

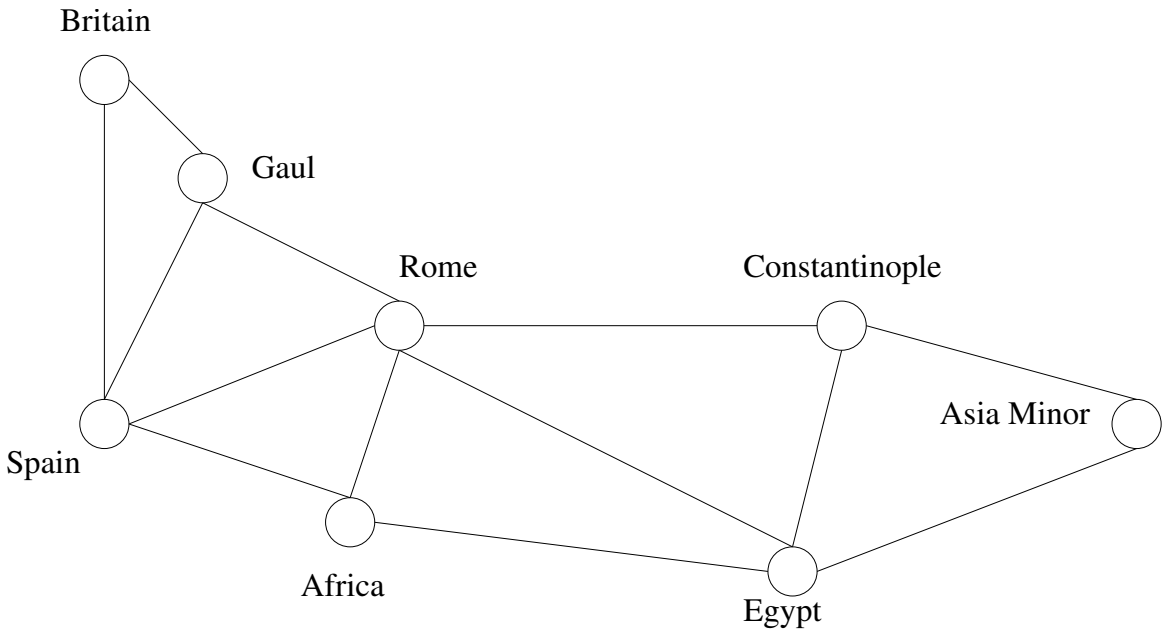
$\rightsquigarrow$

0	Min.
1	2
0	2
$\hat{0}$	∞

1	Min.
1	2
0	2
$\hat{0}$	1

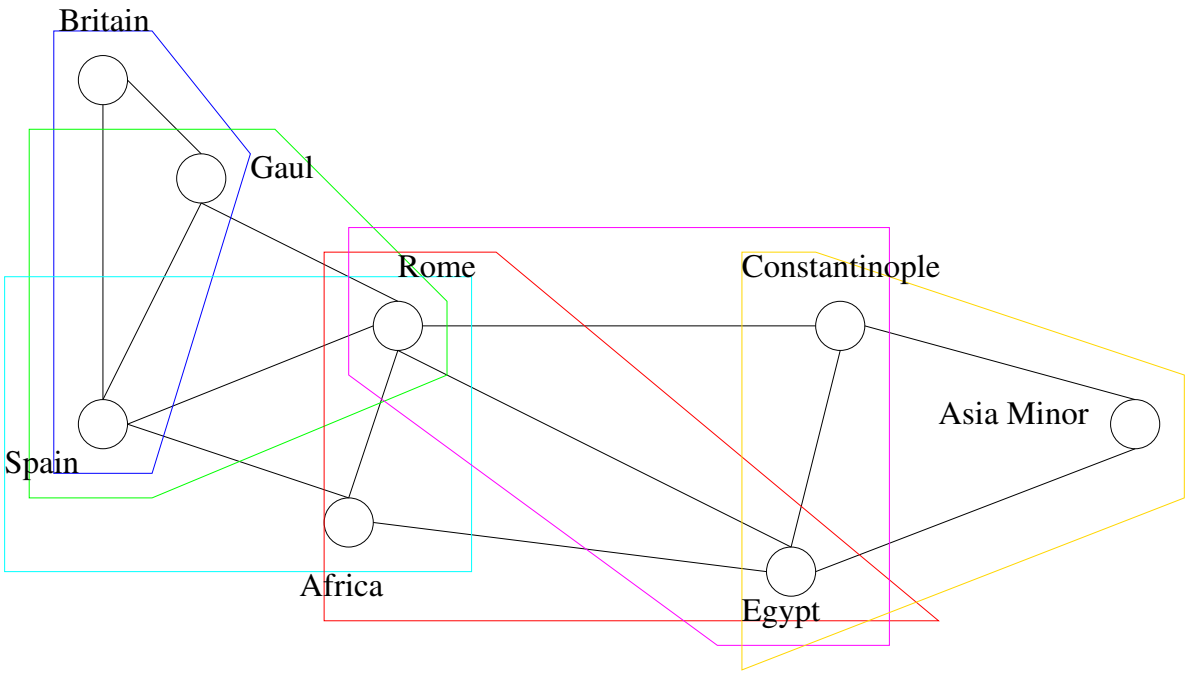
r	Min.
1	4
0	3
$\hat{0}$	∞

Verallgemeinerung: baum- oder pfad-artige Graphen



Aufgabe: Berechne kleinstmögliche Knotenüberdeckung!

Verallgemeinerung: baumartige Graphen



Aufgabe: Berechne kleinstmögliche Knotenüberdeckung!

Baumzerlegung: die Definition

Eine Baumzerlegung eines Graphen $G = (V, E)$ ist ein Paar $\langle \{X_i \mid i \in I\}, T \rangle$, wobei jedes $X_i \subseteq V$ Bag heißt, und T ist ein Baum mit den Elementen von I als Knoten. Es gelten folgende Bedingungen:

1. $\bigcup_{i \in I} X_i = V$; alle Knoten
2. $\forall \{u, v\} \in E \exists \in I : \{u, v\} \subseteq X_i$; alle Kanten
3. $\forall i, j, k \in I$: liegt j auf dem Weg von i nach k in T, so gilt $X_i \cap X_k \subseteq X_j$.

Algorithmik

$\max\{|X_i| \mid i \in I\} - 1$ ist die Weite der Baumzerlegung.

Die Baumweite $\mathrm{tw}(G)$ von G ist das kleinste k, sodass G eine Baumzerlegung der Weite k besitzt.

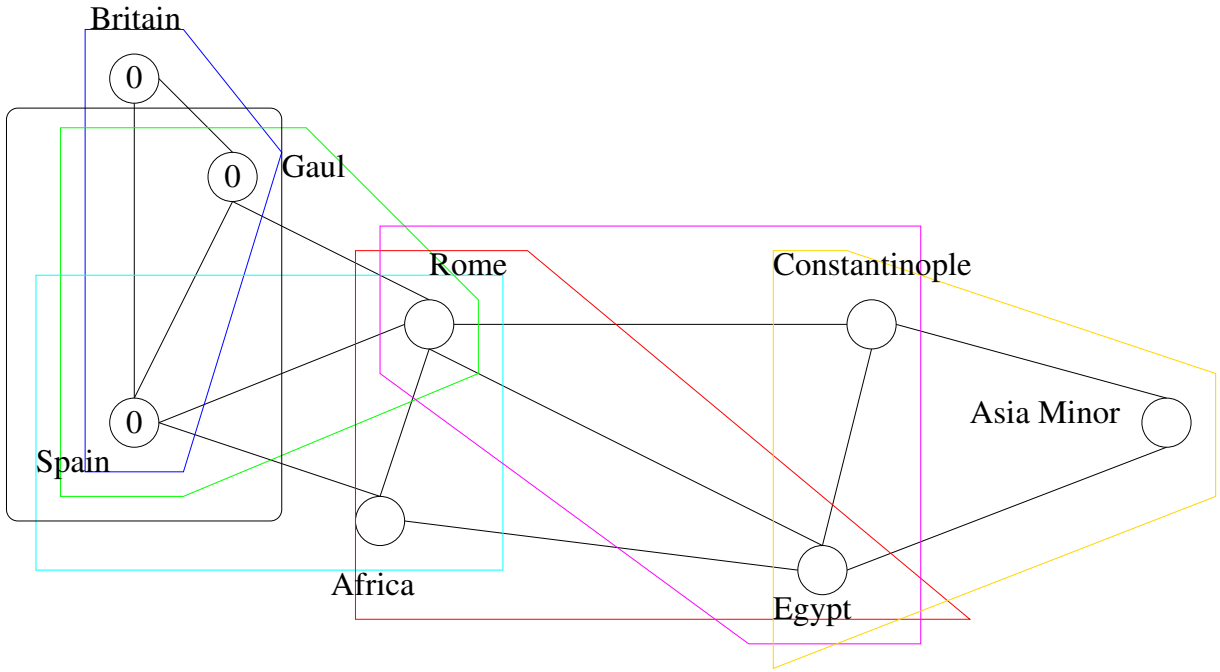
Etwas Strukturelles

Satz 1 *Besitzt* $G = (V, E)$ *eine Baumzerlegung der Weite* ℓ*, so auch jeder Teilgraph.*

Konsequenz: Beim Nachweis von Baumweitenschranken kann man sich manchmal auf Graphen mit möglichst vielen Kanten zurückziehen, z.B. auf triangulierte zusammenhängende Graphen im Falle planarer Graphen.

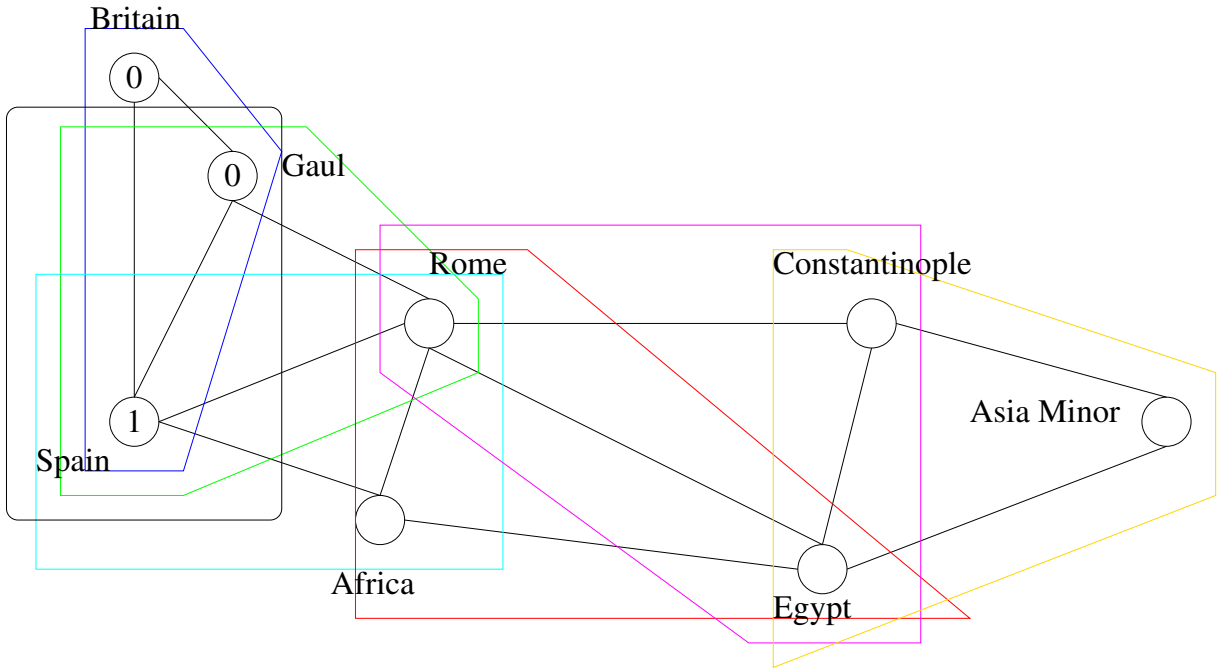
Beispiele: Bäume haben Baumweite 1, Kreise haben Baumweite 2, $\mathrm{tw}(K_n) = n$.

Teste alle Möglichkeiten: 000



Problem: ungültig, da nicht alle Kanten abgedeckt!

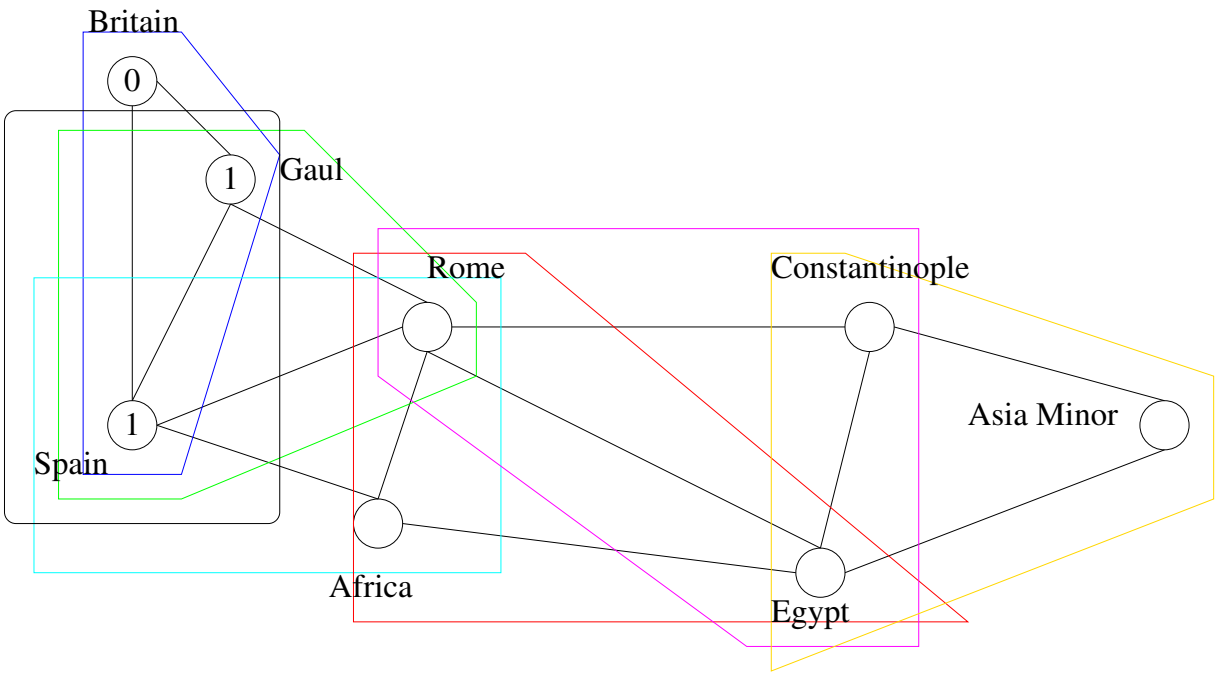
Teste alle Möglichkeiten: 001



Problem: ungültig, da nicht alle Kanten abgedeckt!

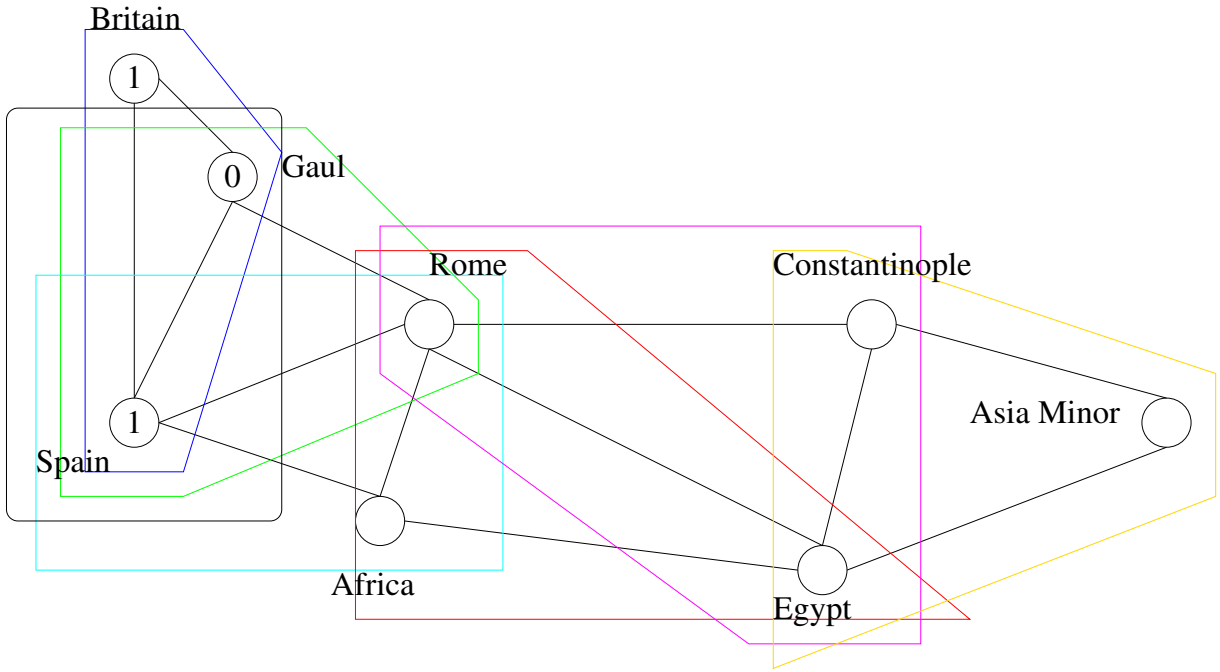
Entsprechend ungültig: 010, 100 (Dreieck)

Teste alle Möglichkeiten: 011



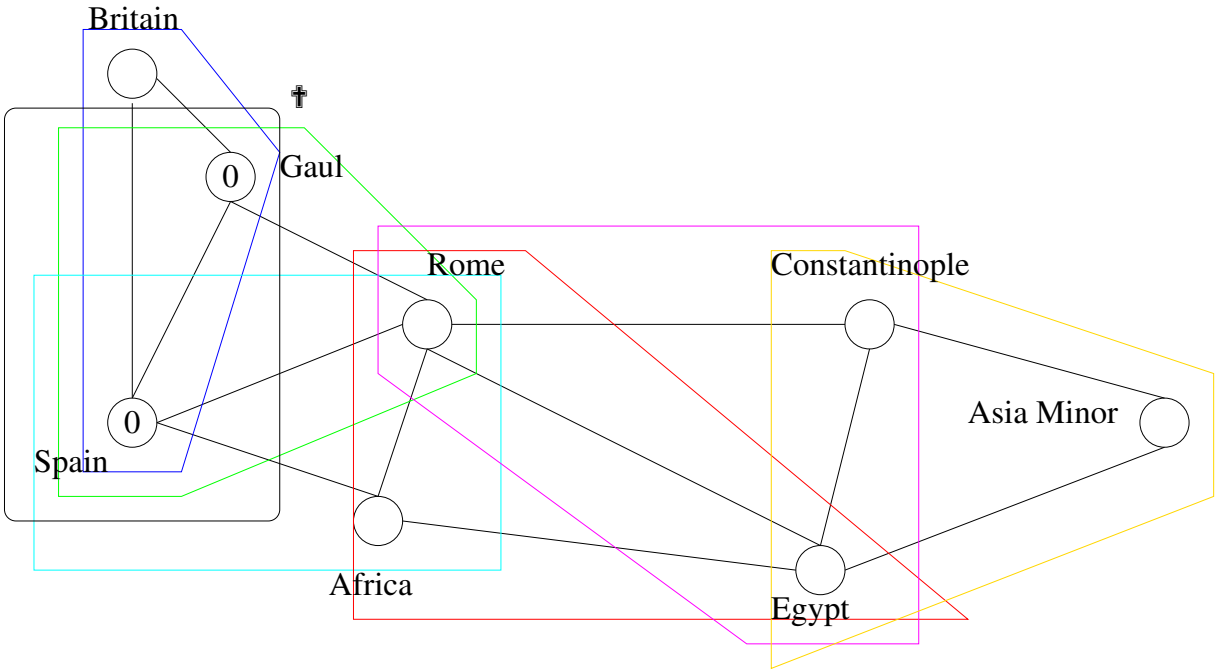
o.k.: Bemerke: besser als 111 !

Teste alle Möglichkeiten: 101



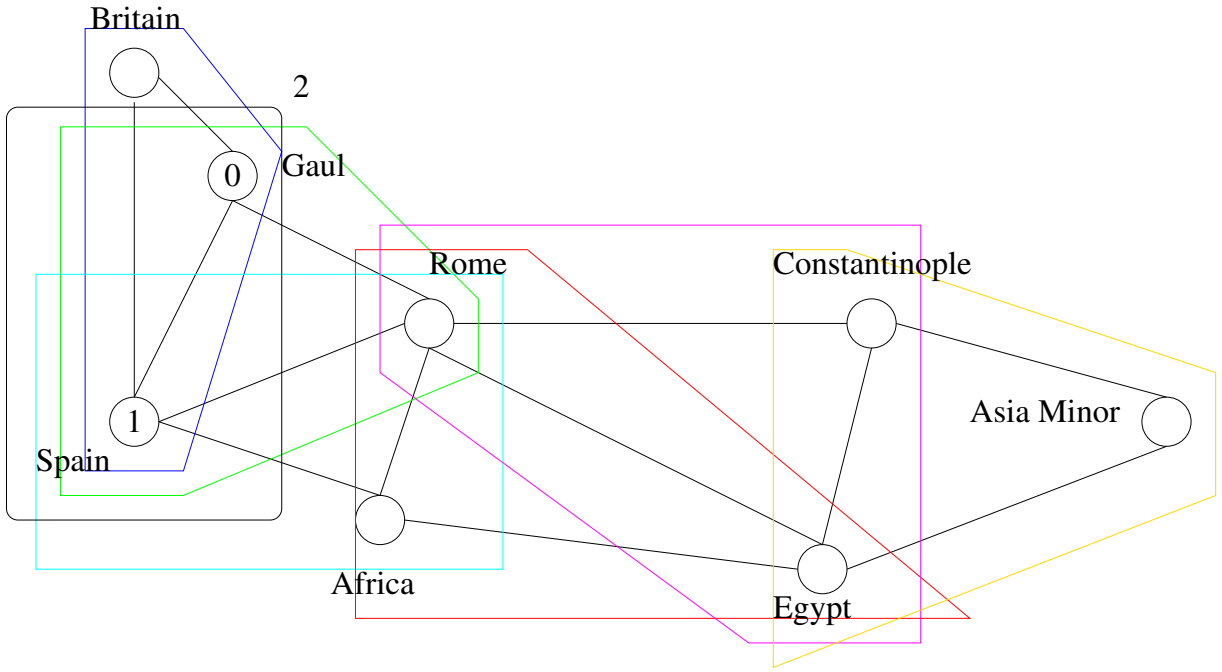
o.k.: Analog: 110

Teste alle Möglichkeiten: 00 (schwarz)



ungültig!

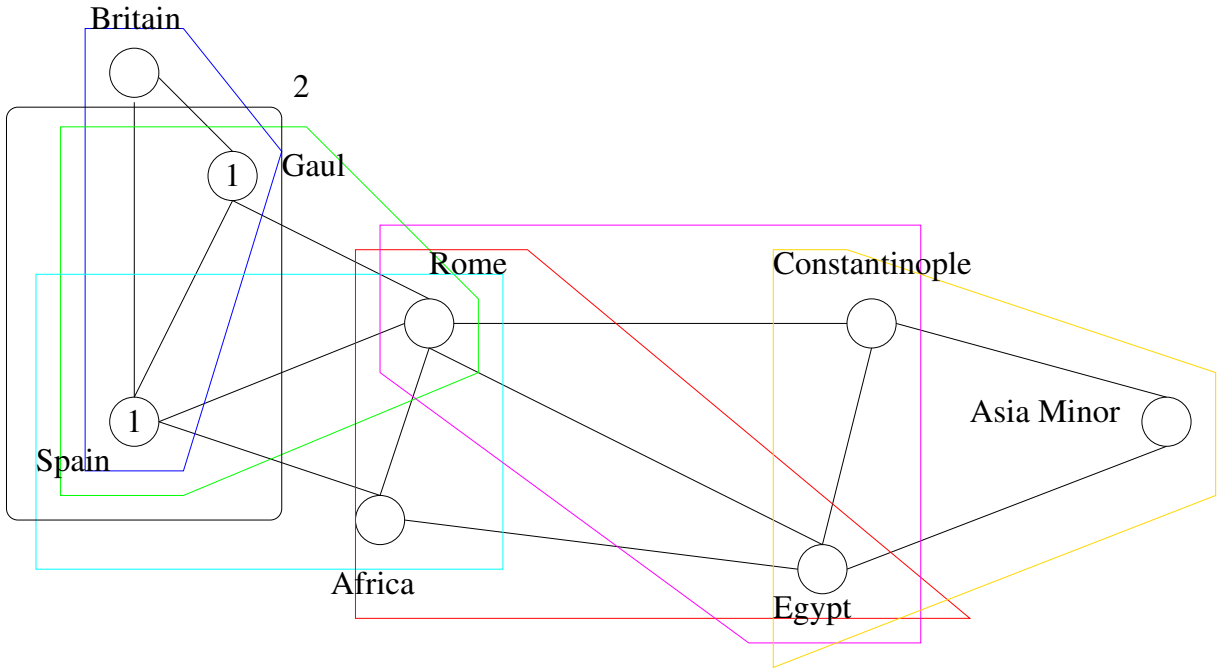
Teste alle Möglichkeiten: 01 (schwarz)



kleinstmögliche Knotenzahl in Überdeckung bislang: 2

10 analog!

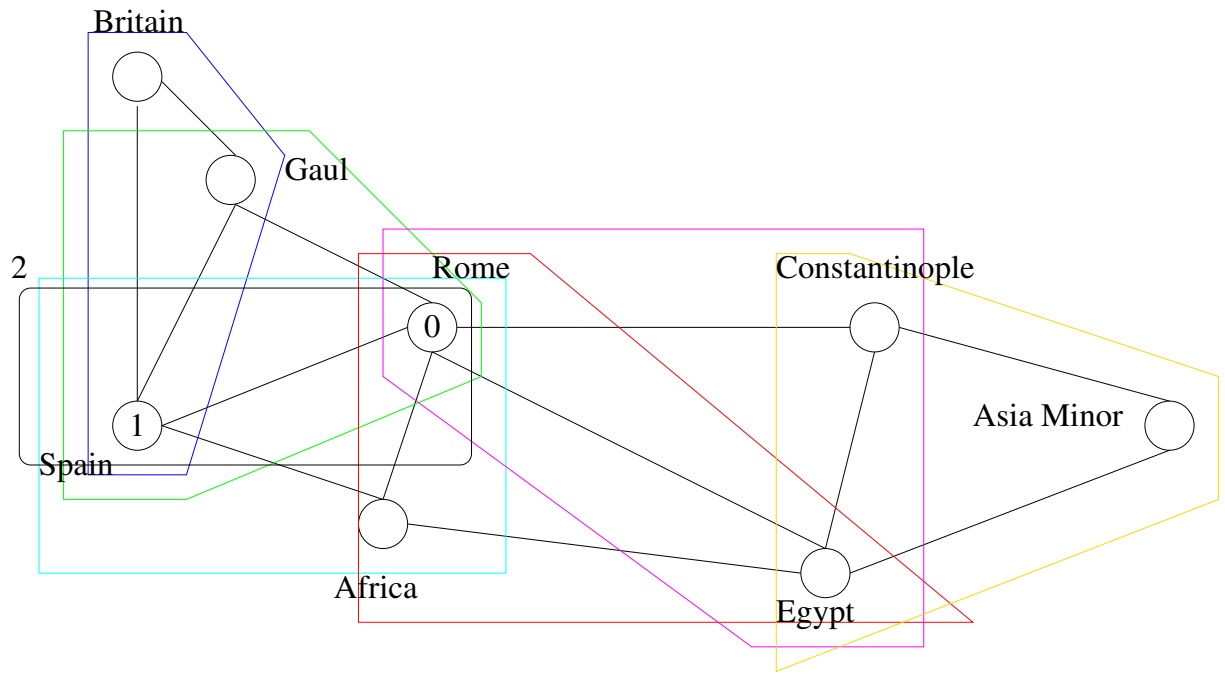
Teste alle Möglichkeiten: 11 (schwarz)



kleinstmögliche Knotenzahl in Überdeckung bislang: 2

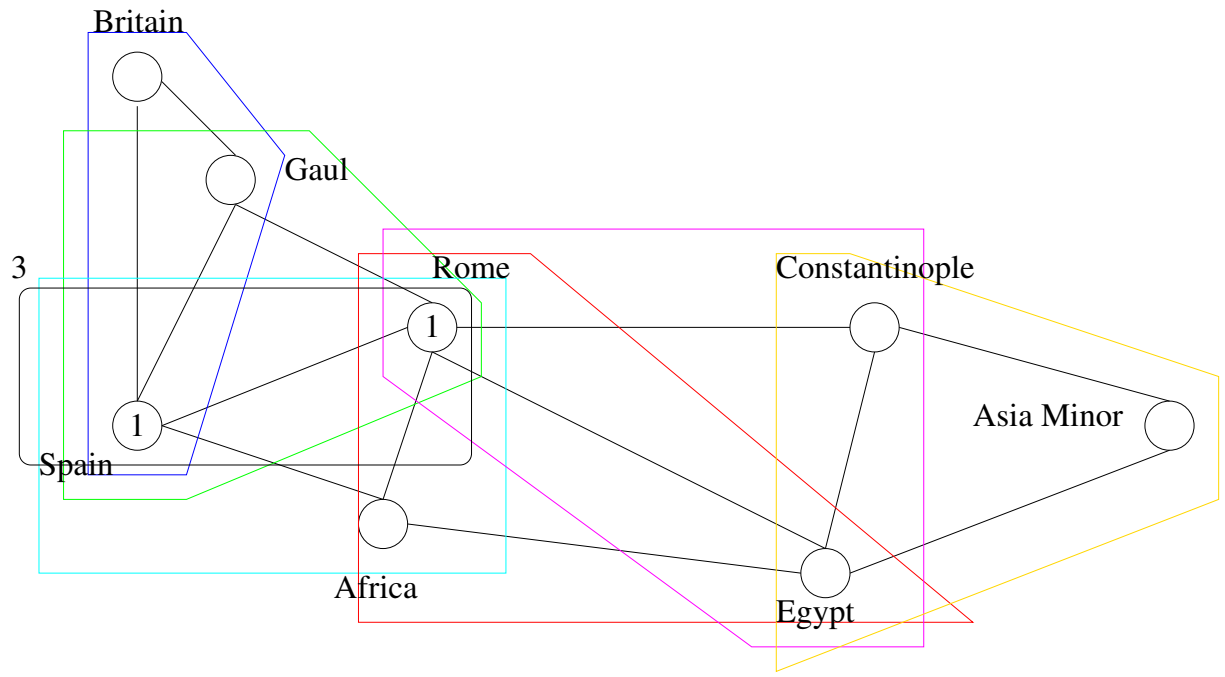
Dies ist die günstigste Möglichkeit ! Warum ?

Nächste Möglichkeiten: 01 (schwarz)



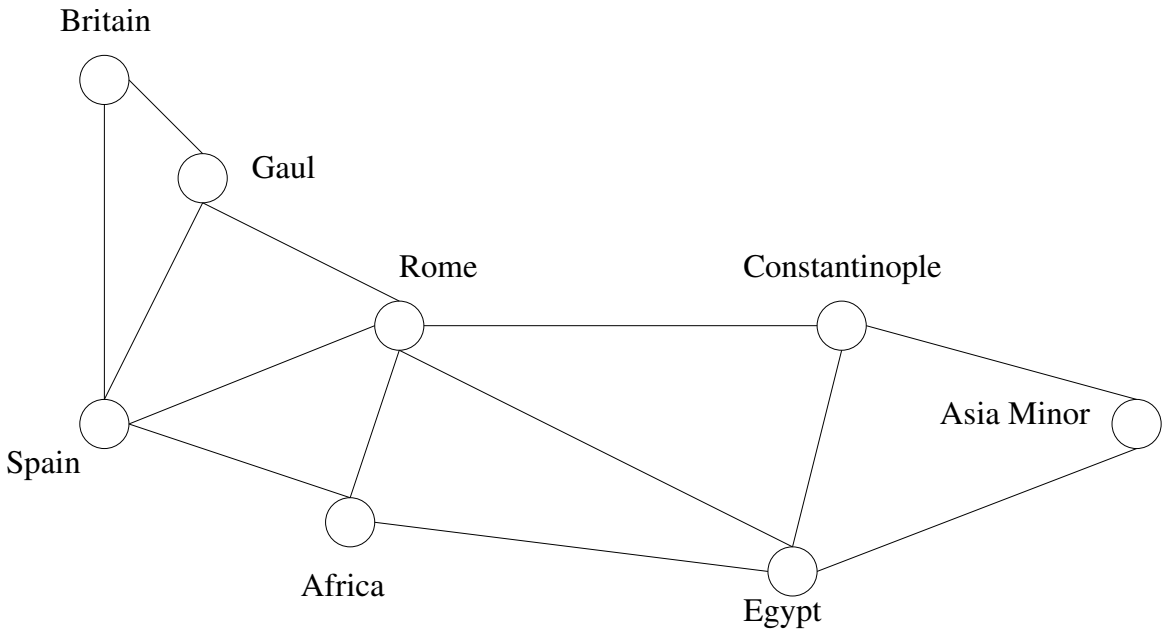
kleinstmögliche Knotenzahl in Überdeckung bislang: 2

Nächste Möglichkeiten: 11 (schwarz)



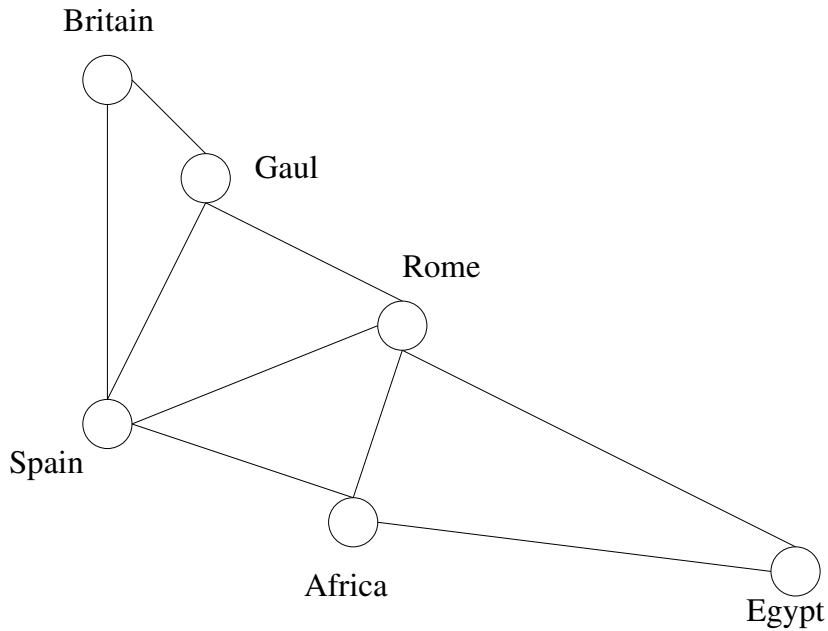
kleinstmögliche Knotenzahl in Überdeckung bislang: 3

Erinnerung: Faltungsregel als algorithmische Alternative



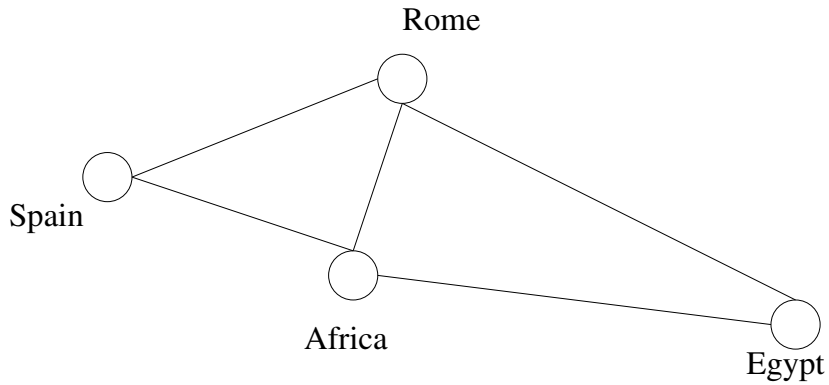
Regel: Falte Asia Minor $\Rightarrow$ verschmelze C und E

Erinnerung: Faltungsregel als algorithmische Alternative



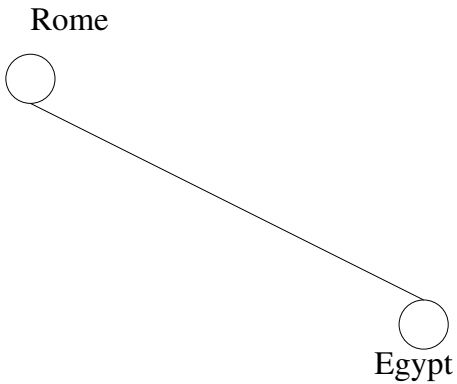
Regel: Falte Britannien $\Rightarrow$ verschmelze G und S

Erinnerung: Faltungsregel als algorithmische Alternative



Regel: Falte Spanien $\Rightarrow$ verschmelze R und A

Erinnerung: Faltungsregel als algorithmische Alternative



schon fertig!

Hübsche Baumzerlegungen

Eine Baumzerlegung $\langle\{X_i \mid i \in I\}, T\rangle$ mit ausgezeichnetem Wurzelknoten r heißt hübsche Baumzerlegung falls gilt:

1. Jeder innere Knoten von T hat höchstens 2 Kinder.

2. Hat n zwei Knoten n' and n'', dann ist $X_n = X_{n'} = X_{n''}$ (join node).

3. Hat n nur ein Kind n', gilt entweder

   (a) $|X_n| = |X_{n'}| + 1$ und $X_{n'} \subset X_n$ (insert node) oder

   (b) $|X_n| = |X_{n'}| - 1$ and $X_n \subset X_{n'}$ (forget node).

Satz 2 *Zu jeder Baumzerlegung kann in Linearzeit eine "äquivalente" hübsche Baumzerlegung des selben Graphens gefunden werden.*

Ist der ursprüngliche Baum ein Pfad, so erhalten wir sogar eine "hübsche Pfadzerlegung."

Konsequenz: Beim Entwickeln von Algorithmen für Graphen mit gegebener Baumzerlegung kann man sich auf hübsche Baumzerlegungen beschränken.

Satz 3 *Kleinstmögliche Knotenüberdeckungen können für $\mathfrak{n}$-Knoten-Graphen mit gegebener Baumzerlegung der Weite ℓ in Zeit $\mathcal{O}(2^{\ell}\mathfrak{n})$ berechnet werden.*

Beweis: Der Algorithmus arbeitet eine hübsche Baumzerlegung des Graphen $G = (V, E)$ "von den Blättern zur Wurzel" ab; die Wahl der Wurzel ist dabei willkürlich. Die Einzelheiten sind im Folgenden beschrieben.

Tabellenaufbau

Jedem Knoten X der Baumzerlegung wird eine Tabelle der Größe $2^{\ell+1}$ zugeordnet, in der die bisherigen "Rekorde" eingetragen sind.
Ein Binärwort der Länge $\ell + 1$ wird wie folgt interpretiert.
Sei $V = \{v_1, \ldots, v_n\}$; für $X \subseteq V$ können wir nun die Reihenfolge der Knoten übernehmen und so eindeutig vom i-ten Knoten in X sprechen.

Nun ist für jedes Binärwort vermerkt, welches der "Wert" der kleinstmöglichen Knotenüberdeckung ist für den von denjenigem Knoten induzierten Teilgraphen ist, welche im Bag X oder in einem seiner Kinder vorkommen, vorausgesetzt, der $i$te Knoten in X ist in der Überdeckung gdw. an der $i$ten Stelle des Binärwortes steht eine Eins.
Der "Wert" ist dabei entweder die Größe der kleinstmöglichen Überdeckung oder ∞ im Falle von Inkonsistenz.

Initialisierung (an den Blättern)

Man kann sich leicht überlegen, dass wir sogar voraussetzen können, dass alle Bags an den Blättern nur einen Knoten enthalten.
Damit wird die Initialisierung für das dynamische Programmieren trivial.
Genau genommen entspricht dies nun dem Vorgehen an den Blättern beim Lösen desselben Problems auf Bäumen.

insert node: Einfügeknoten

Wir wollen die Tabelle für Knoten X berechnen, der nur einen Sohn Y hat mit $Y \subset X$.
Es gibt also eine Abbildung $\mathfrak{h}$, die jedem Binärwort w der Länge $k \leq \ell + 1$ "von X" in eindeutiger "knotentreuer" Weise ein Binärwort $\mathfrak{h}(w)$ der Länge $k - 1$ "von Y" zuordnet.
Ist das "neue Bit" eine Eins, so erhöht sich der Wert (im Vergleich zu dem bei $\mathfrak{h}(w)$ um Eins.
Ist das "neue Bit" eine Null, so wird der Wert von $\mathfrak{h}(w)$ übernommen, es sei denn, es entstehen Inkonsistenzen (Wert ∞).

forget node: Löschknoten

Wir wollen die Tabelle für Knoten X berechnen, der nur einen Sohn Y hat mit $Y \supset X$.

Es gibt also eine Abbildung h, die jedem Binärwort w der Länge $k \leq \ell$ “von X” in eindeutiger “knotentreuer” Weise zwei Binärworter $h_1(w), h_2(w)$ der Länge $k+1$ “von Y” zuordnet.
Der Wert von w ist dann das Maximum der Werte von $h_1(w)$ und $h_2(w)$.

join node: Verbindungsknoten

Hat X zwei Kinder X' and X'', dann ist $X = X' = X''$.

Der Wert $\mu(w)$ eines Binärwortes in der Tabelle von X errechnet sich gemäß

$$\mu(w) = \mu'(w) + \mu''(w) - |w|_1.$$

Dabei ist μ' bzw. μ'' die Wertefunktion für Knoten X' bzw. X''.
$|w|_1$ zählt die Einsen in Wort w.

Algorithm 1 A schematics for algorithms based on a given tree decomposition.

Input(s): a graph G, together with a tree decomposition
Output(s): Whatever to be solved...

Find a nice tree decomposition $\mathcal{TD} = \langle \{X_i \mid i \in I\}, T \rangle$ of G.
Perform a depth-first search along T, choosing an arbitrary root r for T as starting point; i.e., call: dfs$(G, \mathcal{TD}, r)$.
Read off a solution from the table that belongs to r.

subroutine dfs$(G, \mathcal{TD}, n)$
if current node n is a leaf **then**
    perform leaf node actions and return corresponding table
else if current node n has one child n' **then**
    table$_{n'}$:=dfs$(G, \mathcal{TD}, n')$
    if n is forget node **then**
        perform forget node actions and return corresponding table, based on table$_{n'}$
    else
        perform insert node actions and return corresponding table, based on table$_{n'}$
    end if
else
    {Current node n has two children n' and n''}
    table$_{n'}$:=dfs$(G, \mathcal{TD}, n')$
    table$_{n''}$:=dfs$(G, \mathcal{TD}, n'')$
    perform join node actions and return corresponding table, based on table$_{n'}$ and on table$_{n''}$
end if

Zu dominierenden Mengen

Wir brauchen jetzt drei “Zustände”: 1, 0, $\hat{0}$.
Lösch- und Einfügeknoten werden ähnlich wie bei VC behandelt (Zeit 3^ℓ).
Schwierigkeit: Join-Knoten mit Zustand 0: “Woher” kommt denn die Dominierung ?!
$\rightsquigarrow$ Naiv 9^ℓ Vergleiche bei Baumweite ℓ.
Monotonie-Trick: Re-Interpretation von $\hat{0}$: Unklar, ob bereits Dominierung vorliegt.
Das bedeutet: betrachten wir Zuweisungen α und α' derart, dass α aus α' durch Ersetzung einiger 0 durch $\hat{0}$ entsteht, so ist der Tabelleneintrag von α größer oder gleich dem von α'.
$\rightsquigarrow$ Bei Eintragung 0 in Vatertabelle muss man in die entsprechenden Kinder-Tabellen mit 0 links und $\hat{0}$ rechts und umgekehrt schauen.
Für eine Tabelle der Größe 3^ℓ erhält man als Laufzeit $T(\ell)$ für die Tabellenaktualisierung die Rekurrenz $T(\ell) = 4T(\ell - 1)$ und daraus die Laufzeit 4^ℓ.
Durch Mengenfaltung lässt sich dies weiter auf 3^ℓ drücken.

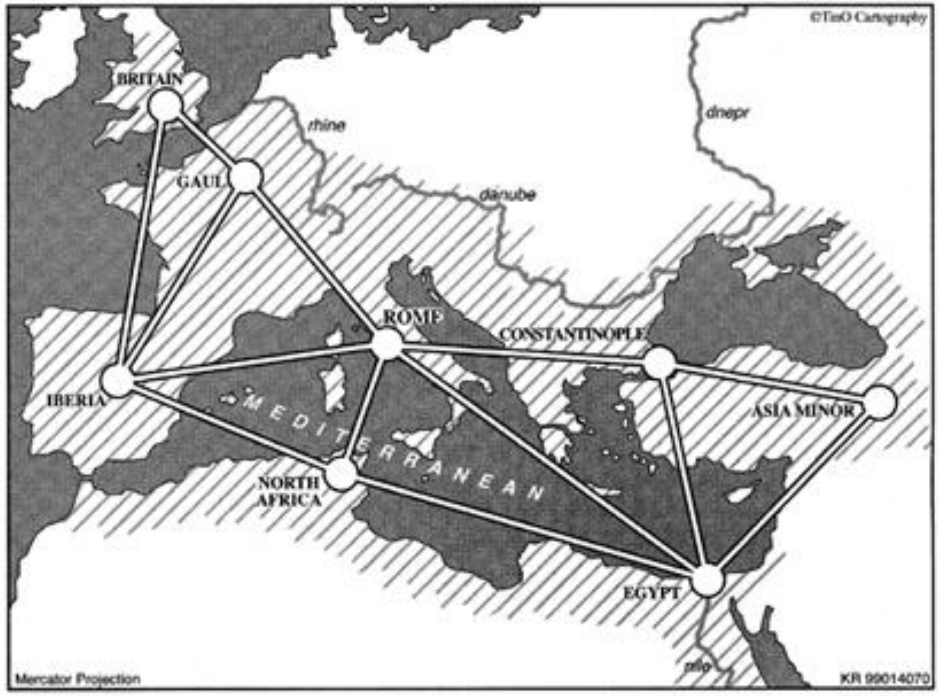
Satz 4 *Kleinstmögliche dominierende Mengen können für n-Knoten-Graphen mit gegebener Baumzerlegung der Weite ℓ in Zeit $\mathcal{O}(3^\ell n)$ berechnet werden.*

Ein Beitrag zum Konstantin-Jahr 2007 ?

Kaiser Konstantins Thronsaal

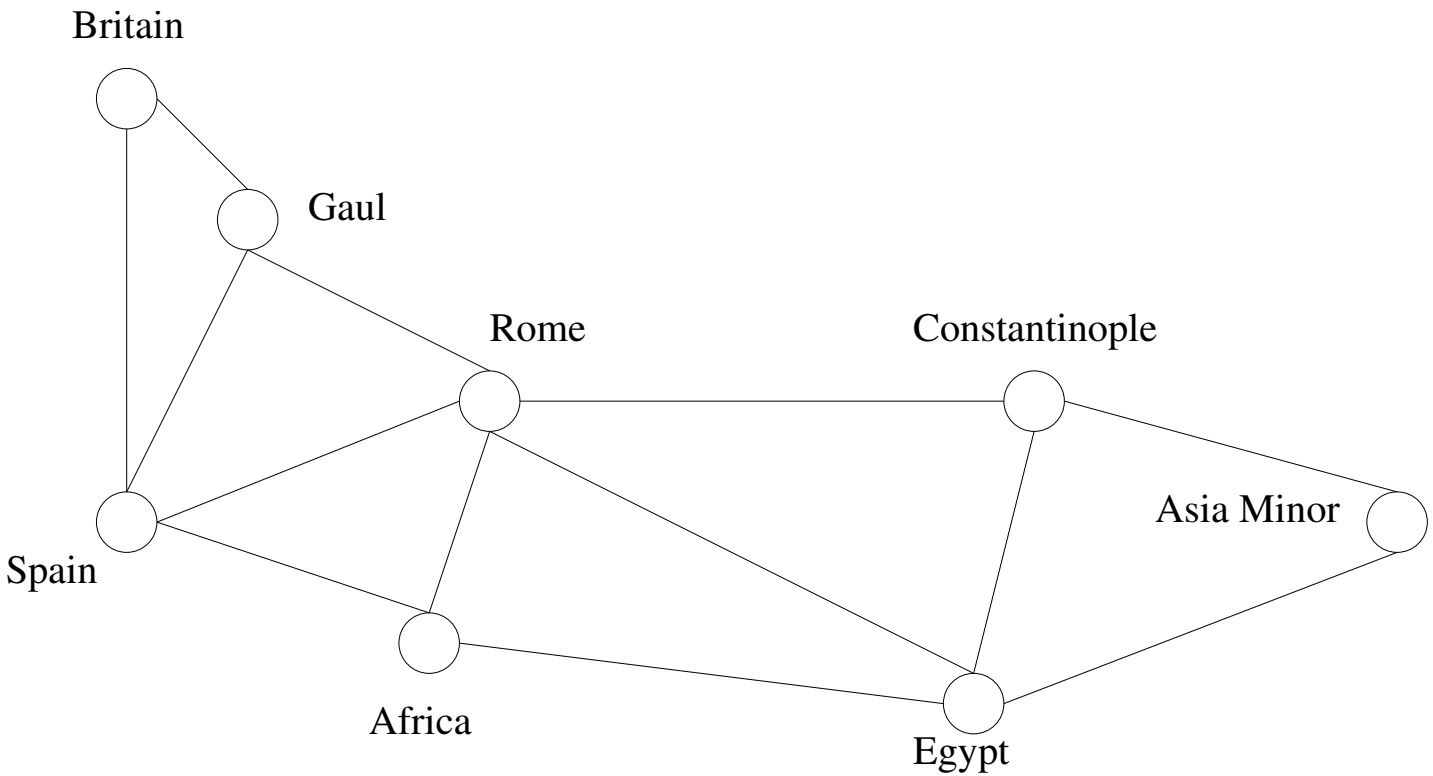
Ein historische Anwendung

Das Römische Reich zur Zeit Konstantins des Großen

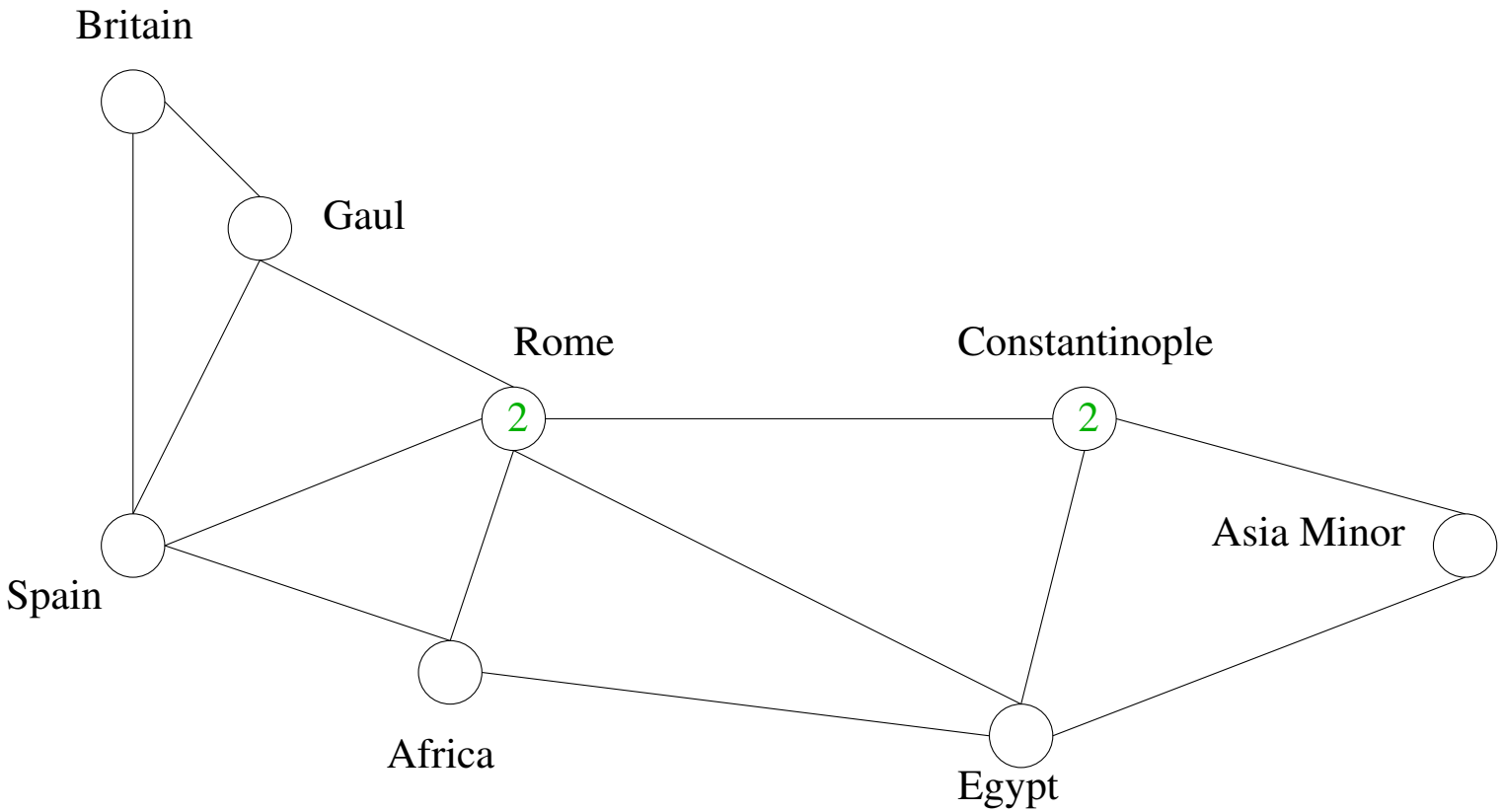


Ein reines Graphenmodell für Konstantins Problem:

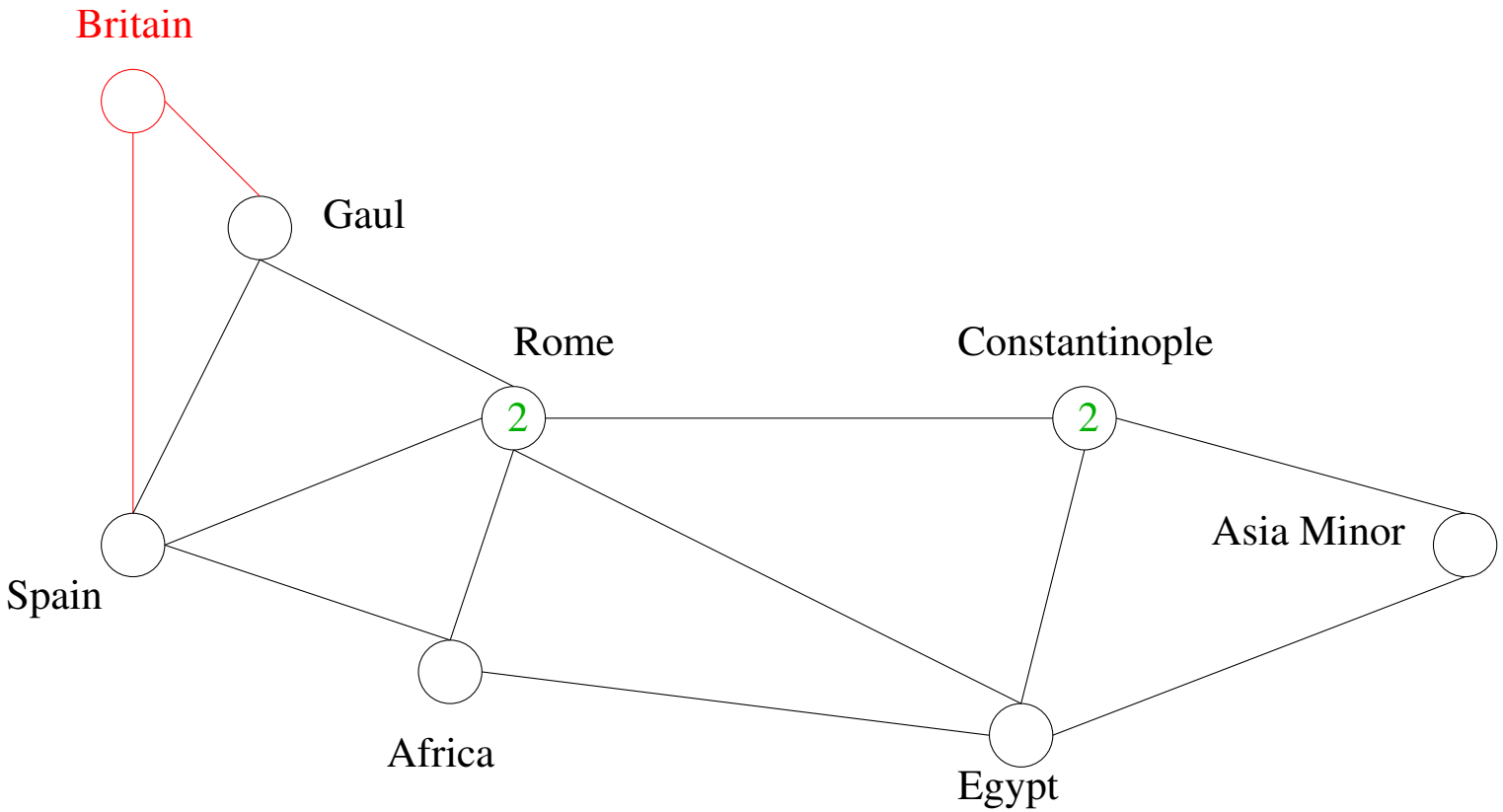
Wie verteile ich vier Hauptarmeen auf die Gebiete, sodass alles “gesichert” ist ?



Konstantins Lösung



Britannien in Gefahr



Etwas genauer bitte: Was bedeutet “gesichert” ?

– Eine Armee auf einem Gebiet sichert (nur) das Gebiet, auf dem sie steht.
– Zwei Armeen auf einem Gebiet sichern auch die angrenzenden Gebiete, denn eine der Armeen kann abziehen, ohne das Gebiet völlig schutzlos zu hinterlassen.
– In der Graphenmodellierung sind die Gebiete Knoten des Graphen, und Kanten geben die “direkte Nachbarschaft” der Gebiete an.

Könnte Informatik Kaiser Konstantin helfen ?

Könnte Informatik Kaiser Konstantin helfen ?

Könnten wir Informatiker also ein Programm für das Problem schreiben ?

Etwas genauer bitte: Wie lautet das römische Dominierungsproblem (formal) ?

Eingabe: Ein Graph $G = (V, E)$ (Knotenmenge, Kantenmenge), eine Schranke k für die Gesamtzahl der Armeen
Gesucht: Eine Zuordnung $f : V \to \{0, 1, 2\}$ (römische Dominierungsfunktion), die für jeden Knoten $v \in V$ angibt, wieviele Armeen (nämlich $f(v)$) auf diesen stationiert werden sollten.
Also gilt: Zu jedem Knoten $v \in V$ mit $f(v) = 0$ (also ohne Armee) gibt es eine Kante $\{v, w\}$ (also: einen Nachbarn w von v) mit $f(w) = 2$.

Könnte Informatik Kaiser Konstantin helfen ?

Mögliche Lösung: Durchprobieren sämtlicher möglicher Zuordnungen.

Bei n Knoten des Graphen gibt es potentiell 3^n Möglichkeiten.

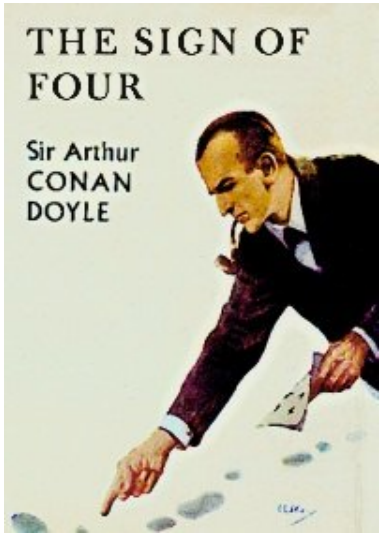
Die Rechenzeit wächst daher exponentiell mit der Eingabegröße.

Theoretische Informatik: Es geht “wahrscheinlich” nicht wirklich besser (NP-Härte).

Was bedeutet diese Erkenntnis ?

Könnte vielleicht Scotland Yard helfen ?

Könnte vielleicht Scotland Yard helfen ?

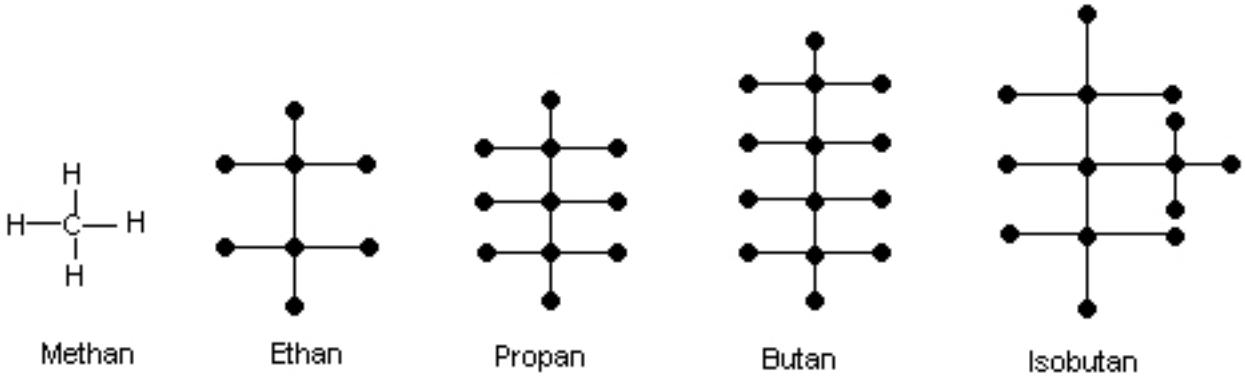


Was macht Scotland Yard ?

Verbrecherjagd

Beobachtung: Der Stadtplan ist abstrakt ein Graph !

Verbrecherjagd Auf welchen Graphen ist Verbrecherjagd einfach ?
Wenn keine Auswege mehr vorhanden sind, also:
Wenn keine Kreise im Graphen existieren.
Kreisfreie (zusammenhängende) Graphen heißen auch Bäume.



Vorgehensweise: systematisches Absuchen mit zwei Polizisten
Beobachtung: Noch einfacher wäre es, wenn es keine Abzweigungen gäbe (Pfade statt Bäume)

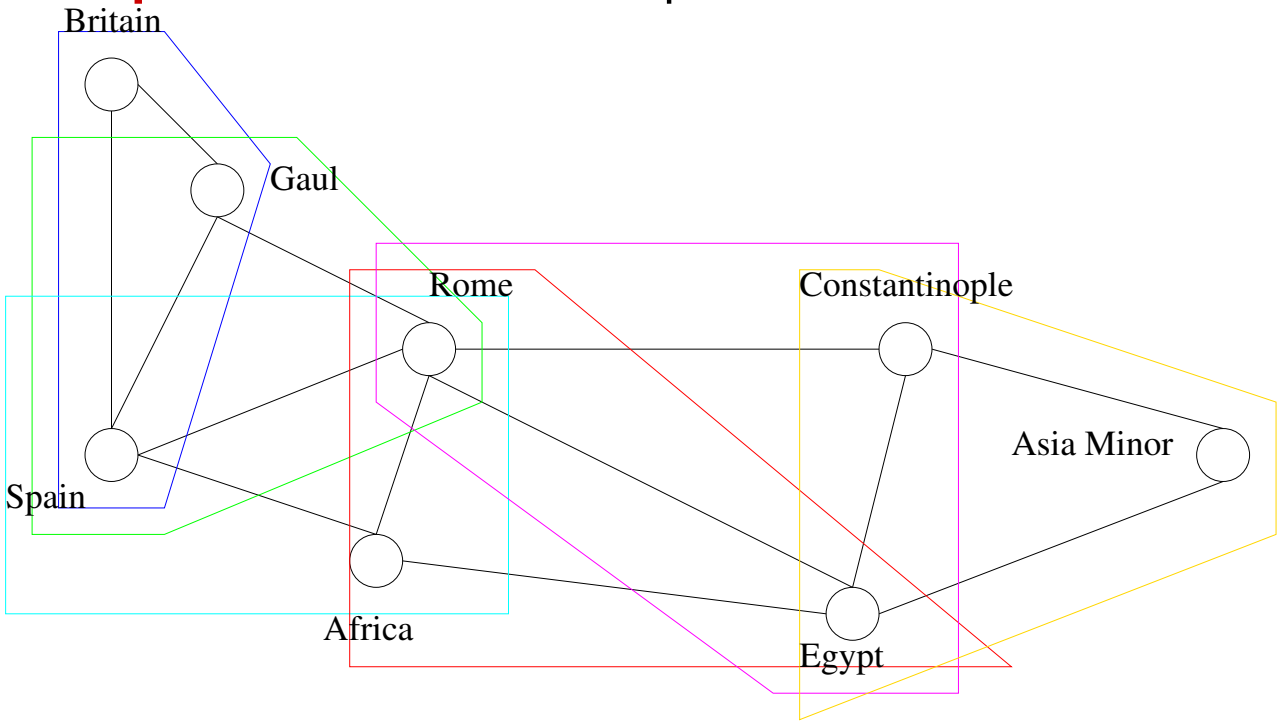
Verbrecherjagd und Algorithmen

Vorgelegt: ein “schwieriges” Problem auf Graphen, jetzt speziell auf Bäumen (oder noch besser auf Pfaden).

Vorgehensweise: systematisches Absuchen mit zwei “Polizisten”,
die sich Teillösungen für bereits abgesuchte Baumteile merken.
Beobachtung: Viele Graph-Probleme lassen sich so “effizient” auf Pfaden oder auch auf Bäumen lösen.

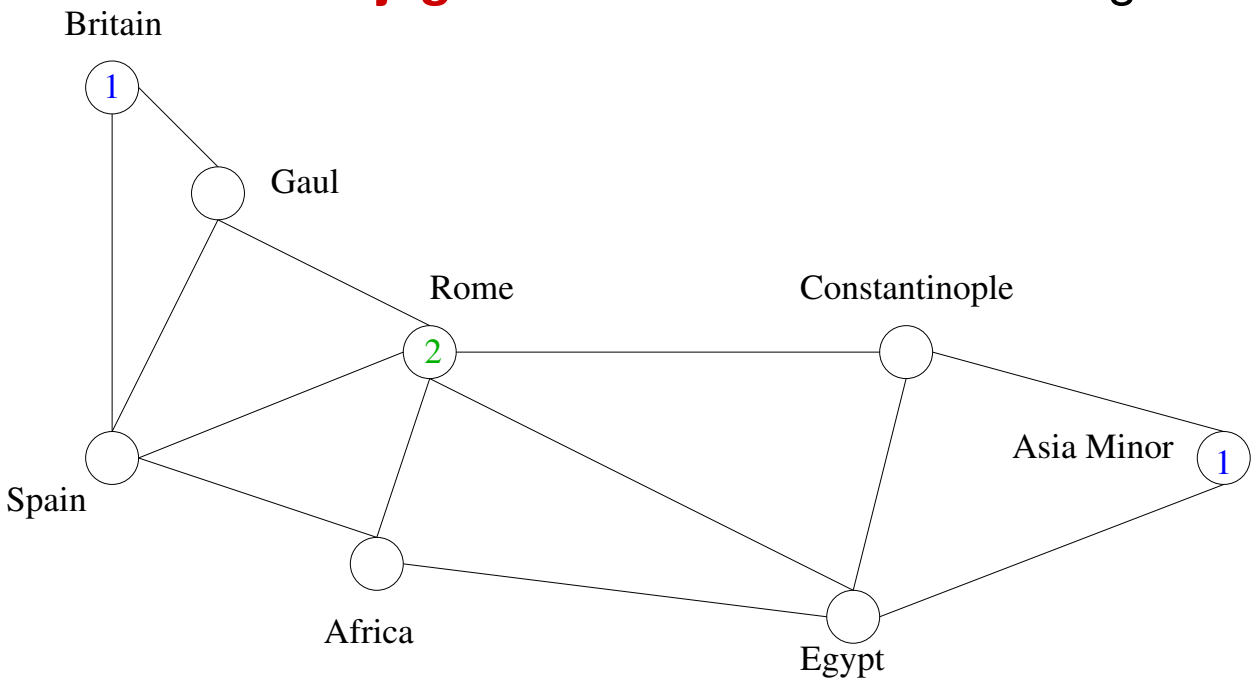
Hilft dies Kaiser Konstantin ?

Konstantins Graph ist kein Pfad, aber “pfadähnlich”



Auch auf pfadähnlichen Graphen lassen sich Verbrecher mit wenigen Polizisten jagen und daher graphentheoretische Probleme effizient lösen.

Scotland Yards Verbrecherjagdstechnik hätte Konstantin geholfen:



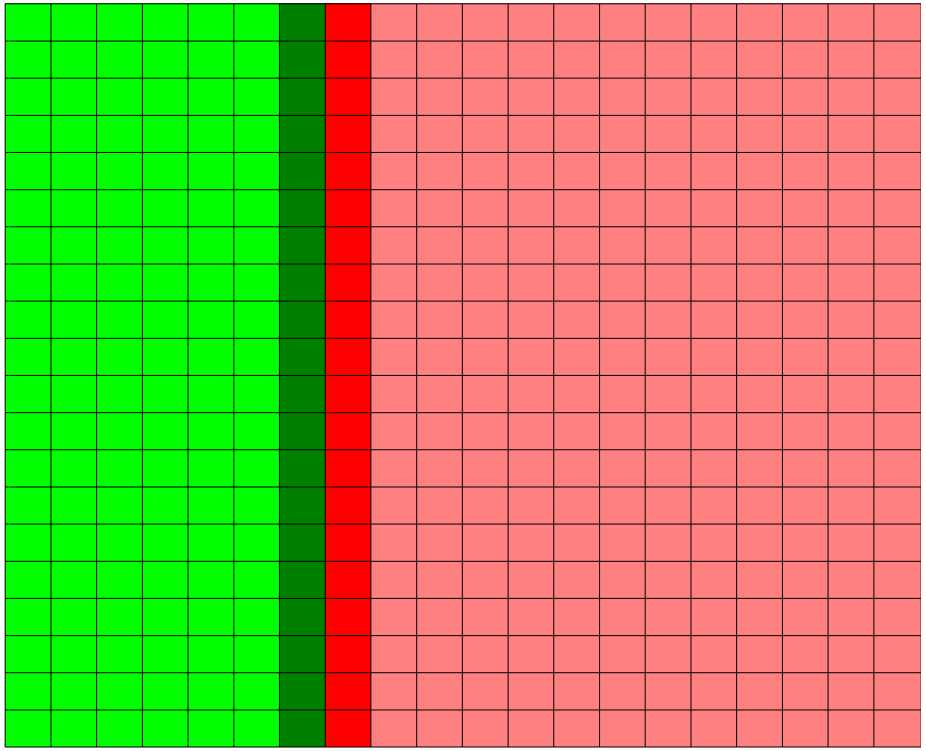
Verbrecherjagd formalisiert

Wir betrachten das folgende induktiv definierte Räuber-und-Gendarm-Spiel mit k Gendarmen (Polizisten) und einem Räuber auf einem Graphen $G = (V, E)$:
In Runde Null wählen die Gendarme eine Teilmenge X_0 von V mit $|X_0| \leq k$. Der Räuber wählt $v_0 \in V \setminus X_0$.
In Runde $i > 0$ wählen die Gendarme eine Teilmenge X_i von V mit $|X_i| \leq k$. Wenn möglich, wählt der Räuber nun einen Knoten $v_i \in V \setminus X_i$ so, dass es es einen (möglicherweise leeren) Weg von v_{i-1} nach v_i gibt, der keine Knoten aus $X_{i-1} \cap X_i$ benutzt.
Die Polizisten gewinnen, sobald der Räuber nicht mehr ziehen kann.
Kann der Räuber stets entwischen, gewinnt der Räuber.

Satz 5 *(Seymour / Thomas 1993) Es sei $k > 1$. Ein Graph G hat Baumweite höchstens $k - 1$ gdw. k Gendarme eine Gewinnstrategie auf G im Räuber-und-Gendarm-Spiel besitzen.*

Machen Sie sich das klar für Bäume ! (Und was ist mit einem einzigen Polizisten ?)

Zurück zur Damendominierung / Eine Sweep-Line Darstellung



Zustandsinformation: Zu jedem Feld auf der Sweep-Line wird ein 9-Bit Vektor beigeordnet

$b = (b_{\swarrow}, b_{\leftarrow}, b_{\nwarrow}, b_{\uparrow}, b_{\nearrow}, b_{\rightarrow}, b_{\searrow}, b_{\downarrow}, q)$.

$b_{\swarrow} = 1$ gdw. Feld wird von Südwest dominiert,

$b_{\leftarrow} = 1$ gdw. Feld wird von West dominiert,

...

$q = 1$ gdw. Dame steht auf dem Feld.

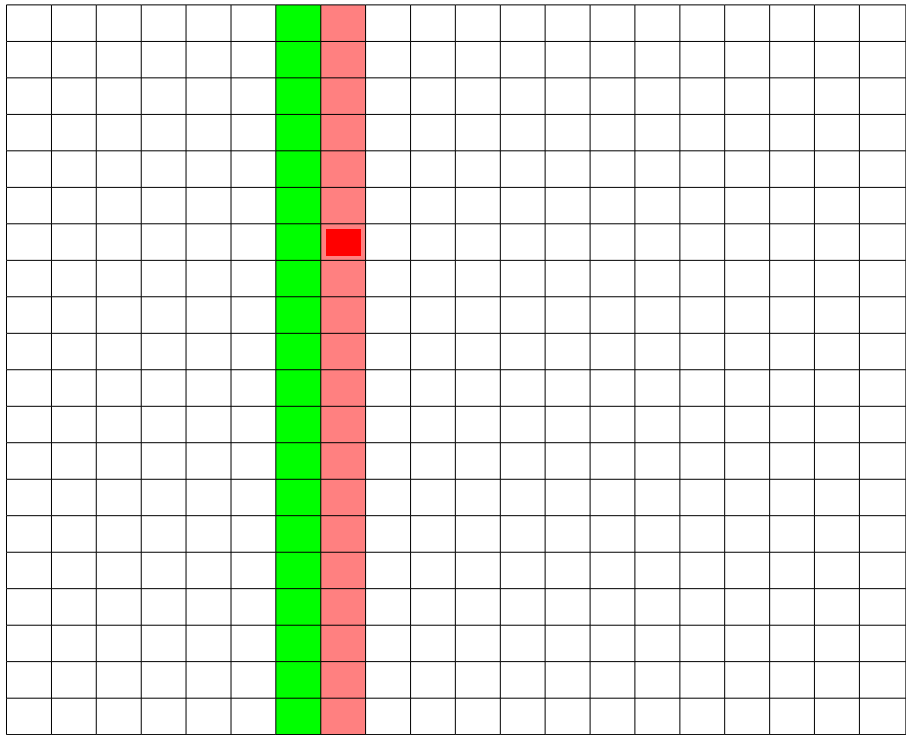
$\rightsquigarrow$ Eine Sweep-Line-Konfiguration (SLC) kann man durch 2^{9n} Bits beschreiben.

Für jede SLC muss die kleinste Anzahl von Damen, die konsistent mit dieser Konfiguration auf das Brett gestellt werden können, gespeichert werden.

Idee: Ein $\mathcal{O}^*(2^{9n})$-Algorithmus.

Wie geht man mit der Sweep-Line um ?:

Betrachten wir ein einzelnes Feld auf der neuen Sweep-Line



Wie geht man mit der Sweep-Line um ?:

Betrachten wir ein einzelnes Feld auf der neuen Sweep-Line

Betrachte den folgenden Zustand.

$$b = (b_{\swarrow} = 0, b_{\leftarrow} = 1, b_{\nwarrow} = 1, b_{\uparrow} = 0, b_{\nearrow} = 0, b_{\rightarrow} = 0, b_{\searrow} = 0, b_{\downarrow} = 0, q = 0),$$

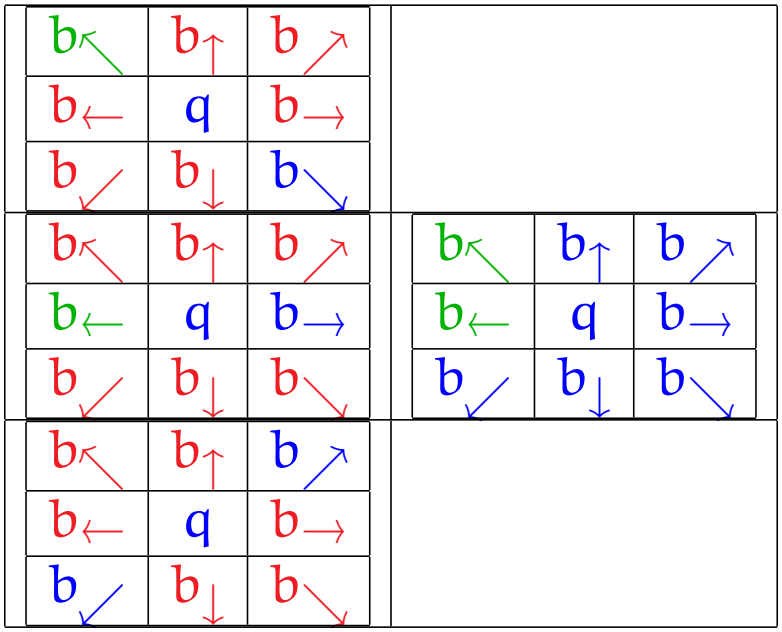
oder bildlich:

$b_{\nwarrow}$	$b_{\uparrow}$	$b_{\nearrow}$
$b_{\leftarrow}$	q	$b_{\rightarrow}$
$b_{\swarrow}$	$b_{\downarrow}$	$b_{\searrow}$

Wie geht man mit der Sweep-Line um ?:

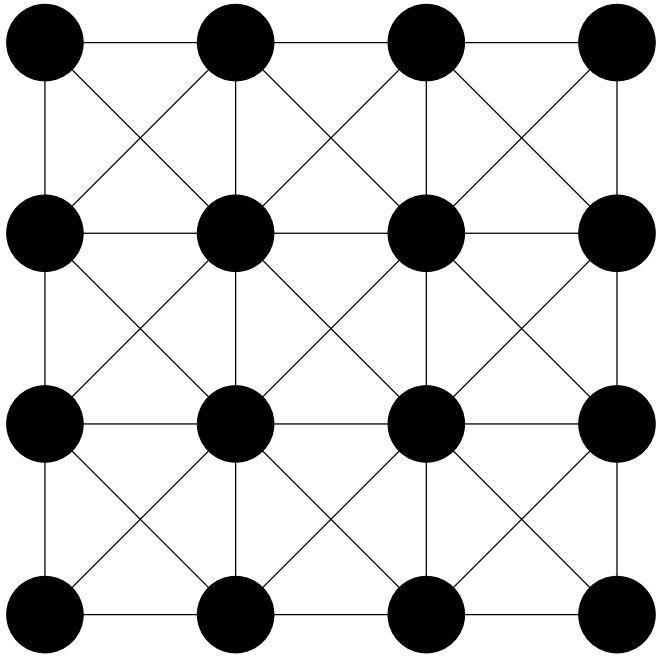
Betrachten wir ein einzelnes Feld auf der neuen Sweep-Line

Der interessante Teil der alten Sweep-Line:

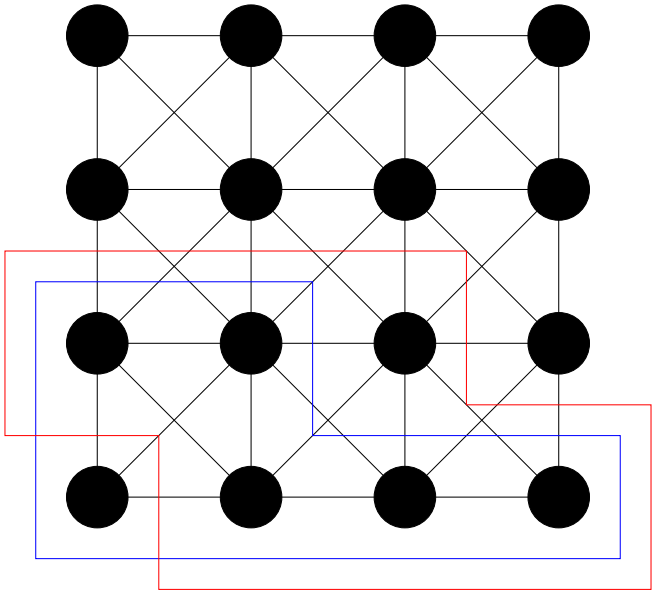


Re-Interpretation: Pfadweiten-Technik

Consider the $n \times n$ Gridgraph mit Diagonalen.



Re-Interpretation: Pfadweiten-Technik



Hinweise

Auch wenn es noch schlimm aussieht. Man kann einiges an den Informationen (Zuständen) einsparen, sodass der Pfadweitenansatz besser wird als das dynamische Mengenprogrammieren aus der vorigen Vorlesung.

Wir arbeiten noch daran. . .

Beachten Sie: Wir haben NICHT den vorher besprochenen Algorithmus fürs Auffinden kleinstmöglicher dominierender Mengen auf Graphen mit beschränkter Baumweite verwendet. Warum ?!

Schöne Weihnachtszeit