NP-vollständige Probleme

- Keine polynomiellen Algorithmen, falls $P \neq NP$.
- Viele wichtige Probleme sind NP-vollständig.
- Irgendwie müssen sie gelöst werden.
- Es reicht nicht zu sagen, daß das Problem NP-schwer ist und daher nicht gelöst werden kann!

Einführendes Beispiel

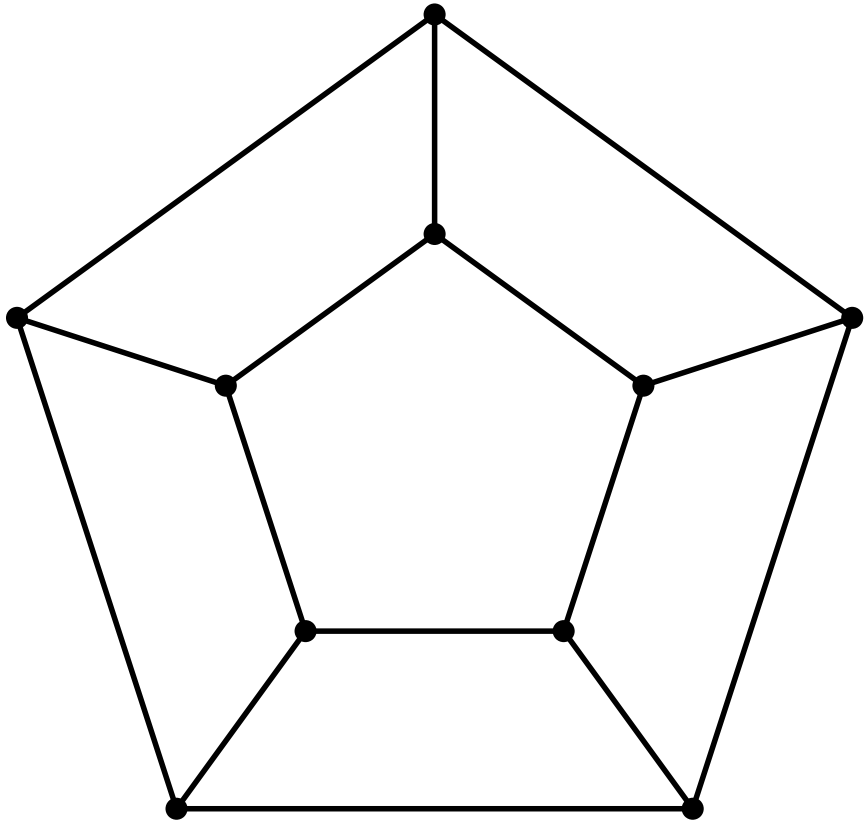
Gegeben: Ein Graph $G = (V, E)$.

Gesucht: Ein minimales *Vertex Cover* $C \subseteq E$.

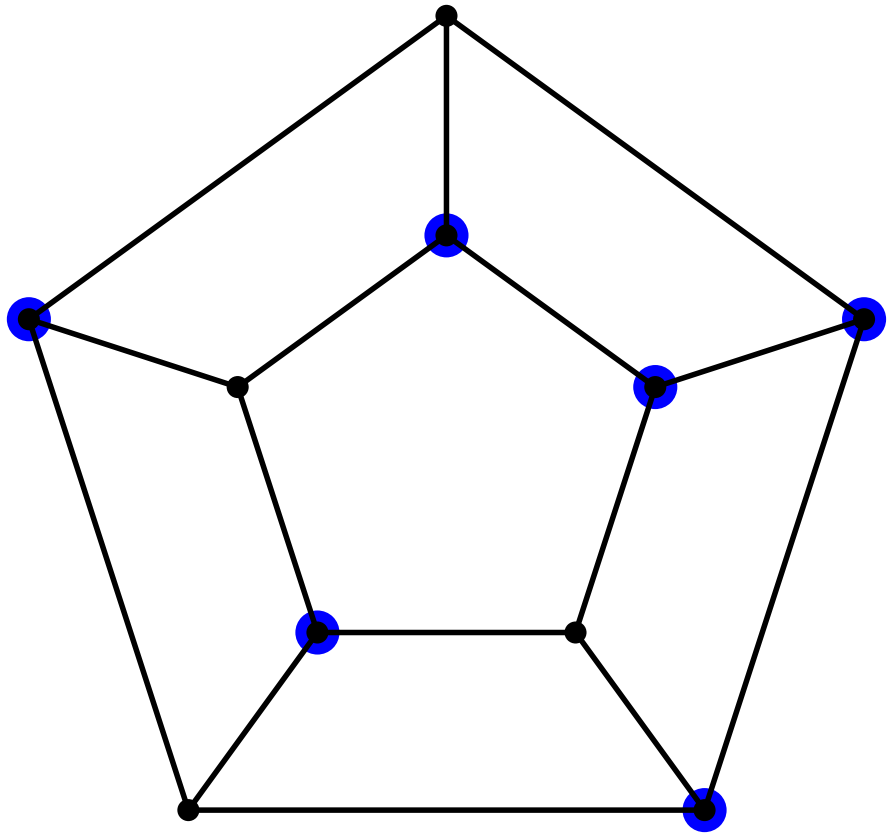
Definition

Ein Menge $C \subseteq V$ ist ein *Vertex Cover* von $G = (V, E)$, falls mindestens ein Endpunkt jeder Kante in E in C enthalten ist.

Beispiel



Beispiel



Modellierung als ILP

Gegeben ist $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}$.

$$\begin{aligned} &\text{Minimiere } v_1 + \ldots + v_n \\ &\quad \text{unter } 0 \leq v_i \leq 1 \text{ für } i = 1, \ldots, n \\ &\quad\quad v_i + v_j \geq 1 \text{ für } \{v_i, v_j\} \in E \\ &\quad\quad v_i \in \mathbf{Z} \text{ für } i = 1, \ldots, n \end{aligned}$$

Jedes *NP*-vollständige Problem läßt sich auf ein ILP zurückführen (aber meist ist das keine gute Idee).

British Museum Method

Wichtige NP-vollständige Probleme sind oft Suchprobleme. In einem (sehr großen) Suchraum verbergen sich die Lösungen.

Eine mögliche Lösungsstrategie ist es daher, den Lösungsraum vollständig zu durchsuchen.

Für Vertex Cover bedeutet dies, alle $C \subseteq V$ durchzuprobieren.

Das sind $2^{|V|}$ Möglichkeiten.

Die Laufzeit beträgt $O(|E|2^{|V|})$.

Backtracking

Wir betrachten einen Knoten $v \in V$.

Es gibt zwei Möglichkeiten $v \in C$ oder $v \notin C$.

Falls $v \notin C$, dann $N(v) \subseteq C$, denn die zu inzidenten Kanten müssen ja abgedeckt werden.

(Mit $N(v)$ bezeichnen wir die zu v adjazenten Knoten.)

Aus diesen Beobachtungen ergibt sich ein einfacher Algorithmus.

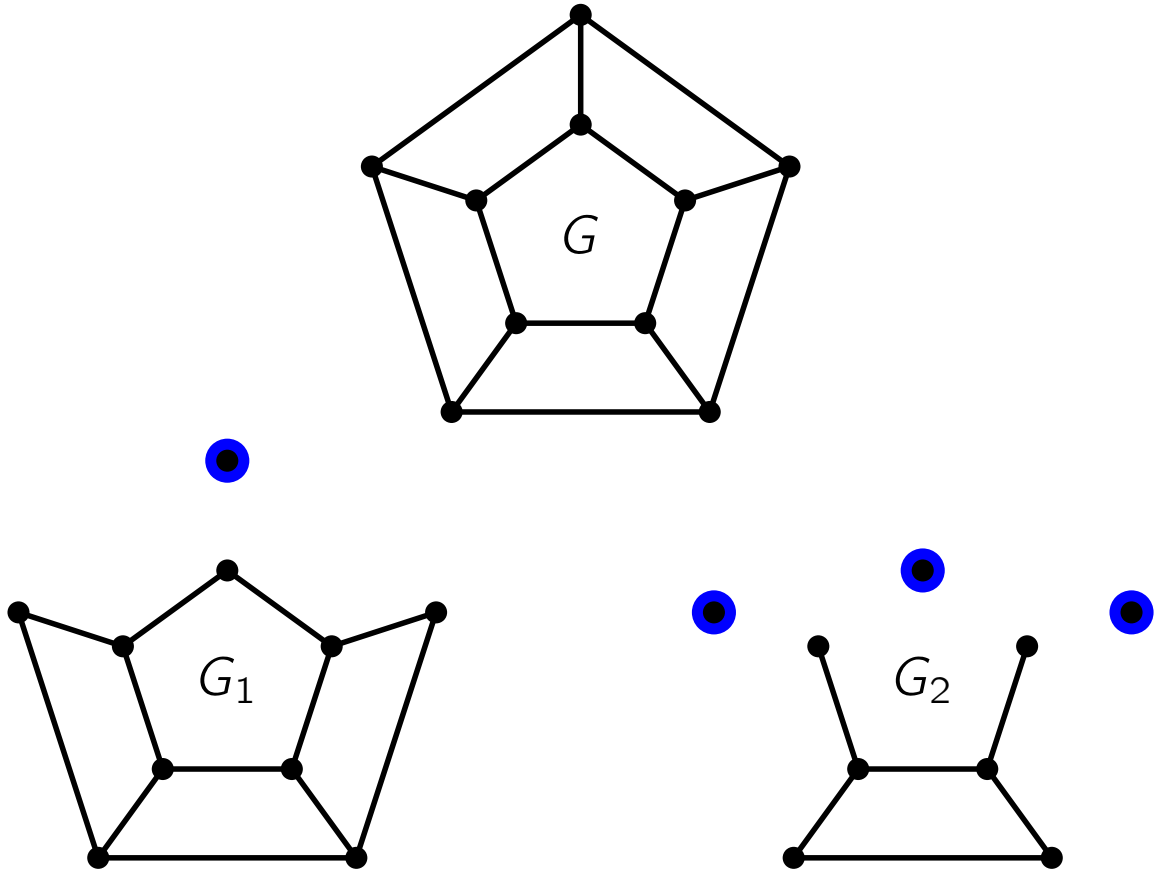
Backtracking

Eingabe: $G = (V, E)$

Ausgabe: Optimales Vertex Cover $VC(G)$

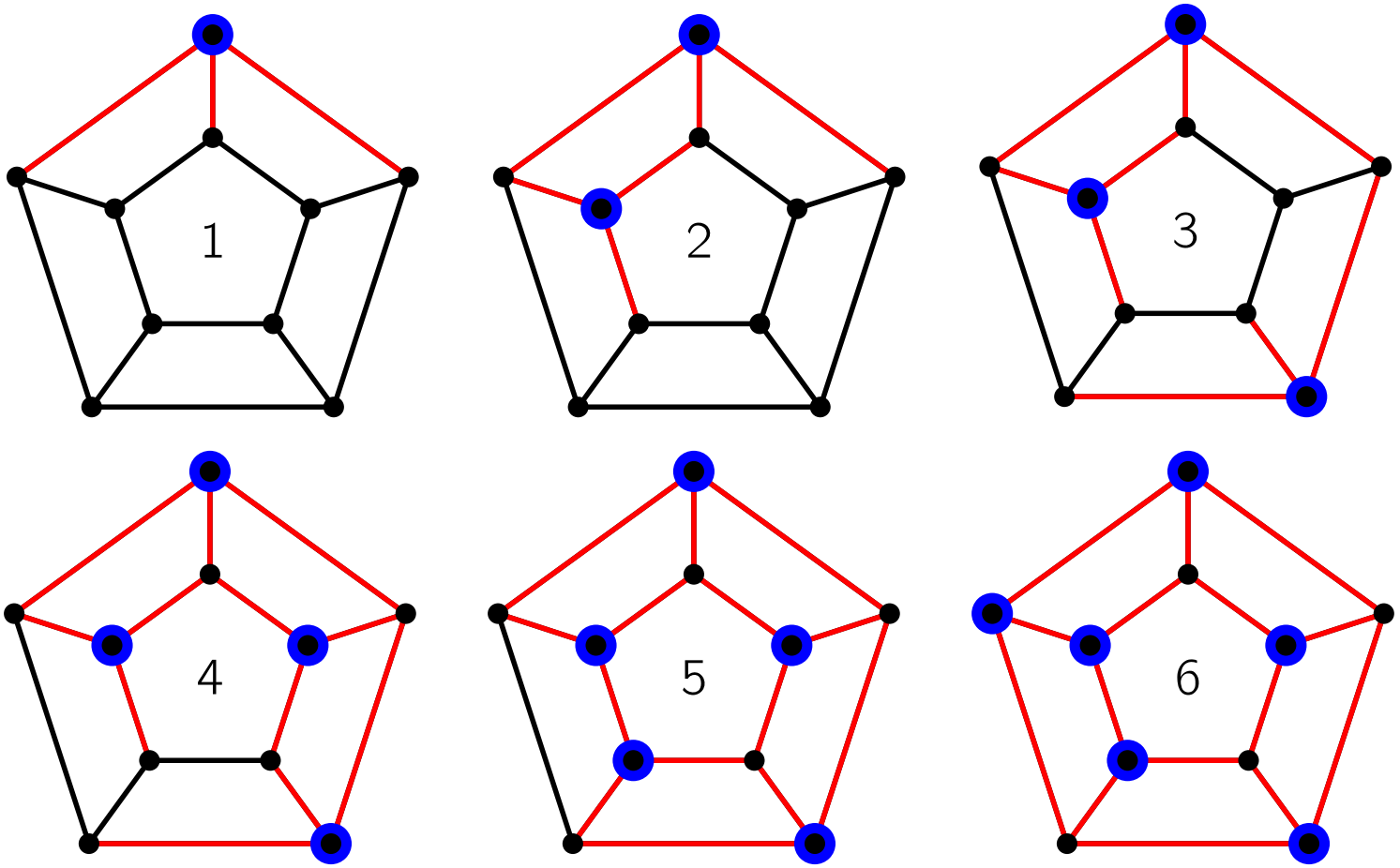
$G_1 := (V - \{v\}, \{\, e \in E \mid v \notin e \,\})$
$G_2 := (V - \{v\} - N(v), \{\, e \in E \mid e \cap N(v) = \emptyset \,\})$
if $|\{v\} \cup VC(G_1)| \leq |N(v) \cup VC(G_2)|$
then return $\{v\} \cup VC(G_1)$
else return $N(v) \cup VC(G_2)$

Backtracking



Heuristiken

Zum Beispiel *Greedy*.



Wähle immer einen Knoten mit maximalem Grad.

Approximationsalgorithmen

Jede Kante muß von mindestens einem ihrer Knoten abgedeckt werden.

Problem: Von welcher?

Lösung: Wir nehmen beide.

- Natürlich gibt es so keine Garantie, daß wir eine optimale Lösung finden.
- Das so gefundene Vertex Cover kann aber höchstens doppelt so groß sein wie ein optimales.

Approximationsalgorithmen

Der Algorithmus könnte so aussehen:

$C := \emptyset$;
while $E \neq \emptyset$ **do**
  choose some $e \in E$;
  $V := V - e$;
  $C := C \cup e$;
  $E := \{ e' \in E \mid e \cap e' = \emptyset \}$
od;
return C

Backtracking

Der Suchraum bei einem NP-vollständigen Problem, läßt sich oft so schreiben:

$$S_1 \times S_2 \times \cdots \times S_n$$

Jeder Punkt im Suchraum ist dann ein Vektor

$$(x_1, x_2, \ldots, x_n)$$

mit $x_i \in S_i$.

Die Grundidee ist es, nacheinander $x_1, x_2, \ldots, x_n$ zu wählen.

Bounding Functions

Sind bereits $x_1, \ldots, x_m$ gewählt, können wir manchmal bereits leicht entscheiden, daß es keine Lösung geben kann, die so beginnt.

Wir nehmen an, es gibt Prädikate $B_1, \ldots, B_n$, so daß wenn

$$B_k(x_1, x_2, \ldots, x_k)$$

falsch ist, dann gibt es keine Lösung $(y_1, \ldots, y_n)$ mit $y_i = x_i$ für $i = 1, \ldots, k$.

Der Algorithmus kann diese Prädikate verwenden, um den Suchraum einzuschränken.

Beispiel

Beim 8-Damenproblem können wir als Suchraum

$$\{1, \ldots, 8\}^8$$

wählen, denn in jeder Zeile steht eine Dame und sie muß in einer der 8 Spalten sein.

Wir setzen $B_k(x_1, \ldots, x_k) = \mathit{false}$, wenn sich bereits zwei der ersten k Damen in den ersten k Zeilen auf den Positionen $x_1, \ldots, x_k$ bedrohen.

Rekursiver Algorithmus

```
Backtrack(k) :
for each x_k ∈ S_k do
   if B_k(x_1, ..., x_k) then
      if (x_1, ..., x_k) ist eine Lösung
      then write (x_1, ..., x_k) fi
      if k < n then Backtrack(k + 1) fi
   fi
od
```

Die Variablen $x_1, \ldots, x_n$ sind global.

Der Algorithmus muß mit $Backtrack(1)$ aufgerufen werden.

Bounding Functions

Wie wählen wir die Prädikate B_k?

Ein Extremfall ist

$$B_k(x_1, \ldots, x_k) = \mathit{true}$$

für alle x_i. Hier wird der Suchbaum niemals abgeschnitten und der gesamte Suchraum durchsucht.

Der andere Extremfall ist

$$B_k(x_1, \ldots, x_k) = \begin{cases} \mathit{true} & \text{falls eine Lösung mit } x_1, \ldots, x_k \text{ beginnt,} \\ \mathit{false} & \text{sonst.} \end{cases}$$

Hier werden alle Teilbäume abgeschnitten, die keine Lösung enthalten.

Beide Extremfälle sind oft ungünstig. Entweder wird zuwenig abgeschnitten oder der Berechnungsaufwand für B_k wird zu groß.

Iterativer Algorithmus

Backtrack :
$k := 1$; $R_k := S_k$
while $k > 0$ **do**
  if $\{ x \in R_k \mid B_k(x_1, \ldots, x_{k-1}, x) \} \neq \emptyset$ **then**
    choose $x_k \in \{ x \in R_k \mid B_k(x_1, \ldots, x_{k-1}, x) \}$
    $R_k := R_k - \{x_k\}$;
    if $(x_1, \ldots, x_k)$ ist eine Lösung **then write** $x_1, \ldots, x_k$;
    $k := k + 1$; $R_k := S_k$
  else $k := k - 1$
od

Abschätzung der Suchbaumgröße

Die Größe des (abgeschnittenen) Suchbaums kann grob abgeschätzt werden, bevor der Algorithmus abläuft.

Falls sich der Suchbaum im Level i um m_i verzweigt, dann ist die Anzahl der Blätter im Suchbaum etwa

$$\prod_{i=1}^{n} m_i.$$

Die Verzweigungsgrade m_i lassen sich leicht für einen zufälligen Pfad im Suchbaum leicht bestimmen.

Experimente zeigen, daß die Schätzungen im allgemeinen ganz gut sind.

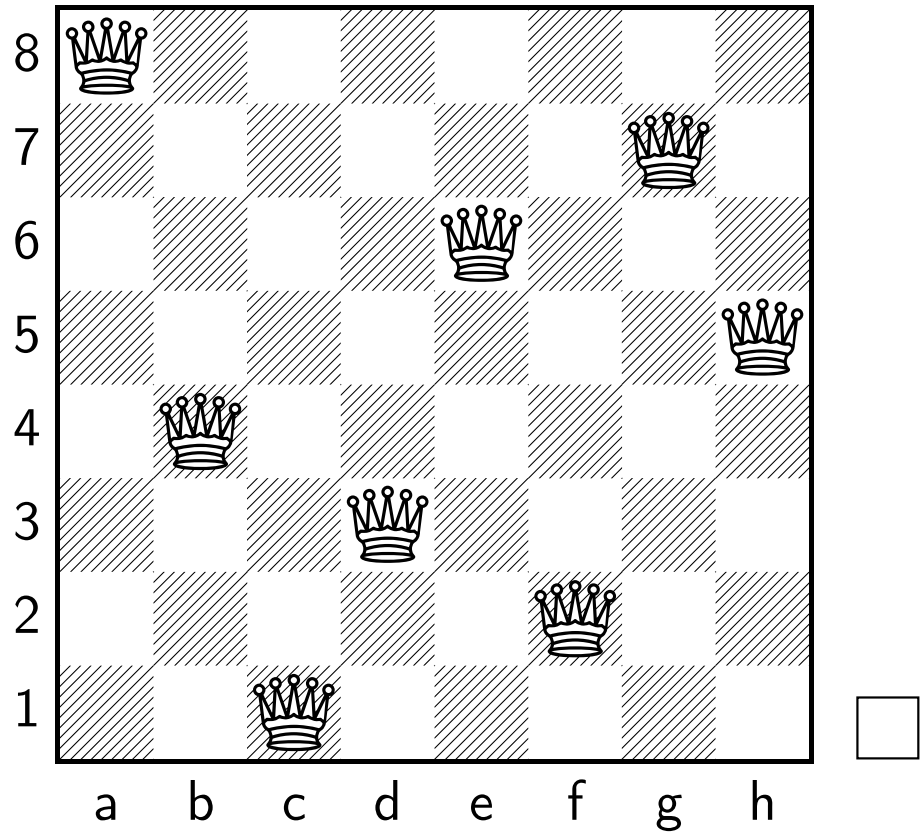
Es können mehrere Schätzungen kombiniert werden (aber nicht zu viele).

Abschätzung der Suchbaumgröße

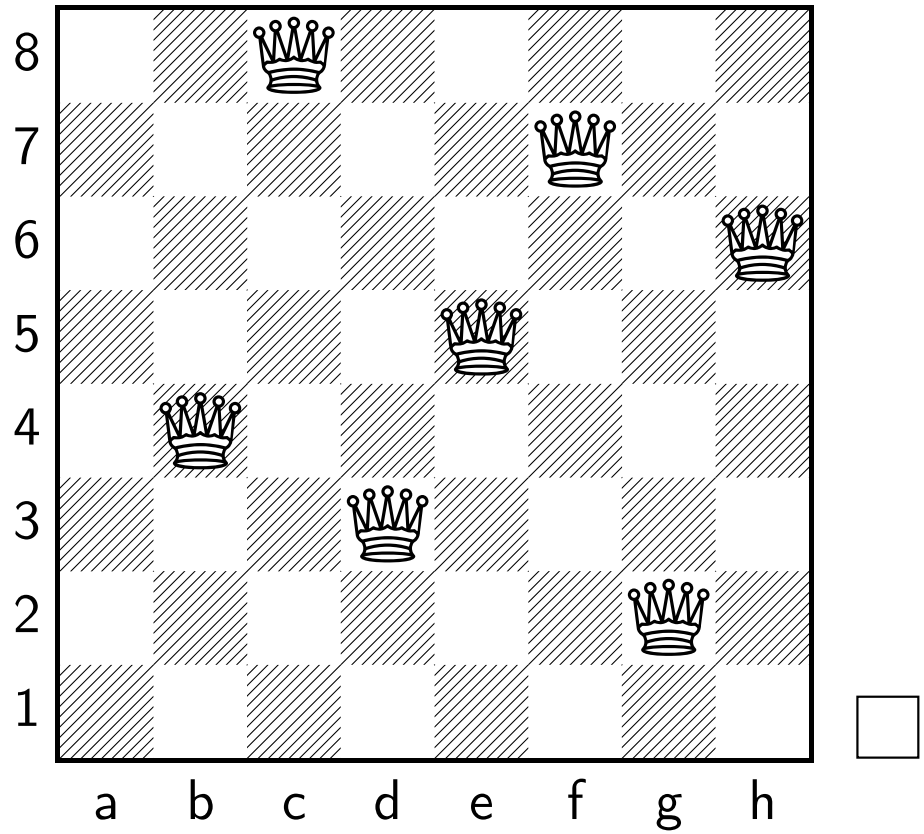
Estimate:
$k := 1$; $m := 1$; $r := 1$;
repeat forever do
  $T_k := \{\, x_k \in S_k \mid B_k(x_1, \ldots, x_k) \,\}$
  if $|T_k| = 0$ **then return** m;
  $r := r \cdot |T_k|$;
  $m := m + r$;
  $k := k + 1$;
  $x_k :=$ zufälliges Element $\in T_k$
od

Dieser Algorithmus schätzt die Suchbaumgröße der rekursiven oder iterativen Backtrackalgorithmen.

Beispiel



Das 8-Damen-Problem



Wir verwenden den Abschätzungsalgorithmus. Es gibt jeweils 8, 5, 3(2?), 3, 2, 1, 1 Möglichkeiten, die nächste Dame zu setzen. Das ergibt geschätze 2689 Positionen im Suchbaum.

Ein Programm für das 8-Damen-Problem mit Backtracking:

```
#include⟨stdio.h⟩
#define N 8
int x[N], c[N], d1[2 * N − 1], d2[2 * N − 1];

int main(void)
{
  int k = 0, i;
  while(k ≥ 0) {
    i = x[k];
    if(i == N) k−−, i = x[k]++, c[i] = d1[k − i + N − 1] = d2[i + k] = 0;
    else if(!c[i] && !d1[k − i + N − 1] && !d2[i + k])
      if(k == N − 1) {
        for(i = 0; i ≤ k; i++) printf("%d ", x[i]);
        printf("\n");
        x[k]++;
      }
      else c[i] = d1[k − i + N − 1] = d2[i + k] = 1, k++, x[k] = 0;
    else x[k]++;
  }
  return 0;
}
```

Ist das Programm korrekt?