

11. Branch and Bound

Einleitung

Kombinatorische Optimierung

Definition

- Ziel: Konstruktion einer Teilmenge aus einer (großen) Menge diskreter Elemente (Gegenstände, Orte usw.).
- Erfüllung gewisser Nebenbedingungen.
- Optimalität bezüglich einer Kostenfunktion (kleinstes Gewicht, kürzeste Strecken, ...).

Optimierung: Schwere Probleme

Schwere Probleme:

- In dieser Lehrveranstaltung wurden bereits Probleme behandelt, für die es **unwahrscheinlich** ist, dass Lösungsverfahren existieren, die **alle möglichen Instanzen** (siehe hier) eines bestimmten Problems in **polynomieller Zeit** lösen können.

Anwendung:

- In den folgenden Einheiten werden wir uns mit Verfahren beschäftigen, die bei solchen schweren Problemen grundsätzlich angewendet werden können.

Verfahren für Optimierungsprobleme:

- Branch-and-Bound
- Dynamische Programmierung
- Approximations(algorithmen)
- Heuristische Verfahren

Diese haben immer unterschiedliche Trade-offs

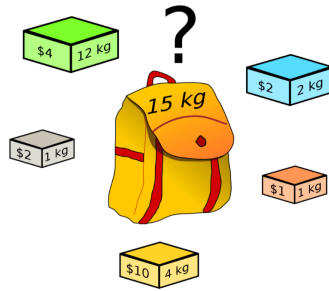
Branch and Bound

Beschränke eine auf Divide-and-Conquer basierende systematische Durchmusterung aller Lösungen mit Hilfe von Methoden, die **untere und obere Schranken** liefern, und ermittle eine optimale Lösung.

Trade-Off hier: **Verzicht auf Garantie der Polynomialzeit.**

Rucksackproblem

Gegeben: n Gegenstände mit positiven rationalen Gewichten g_1, \dots, g_n und Werten w_1, \dots, w_n und eine Kapazität G (auch positiv rational).



Gesucht: Teilmenge S der Gegenstände mit Gesamtgewicht $\leq G$ und maximalem Gesamtwert.

Man will eine Teilmenge der Gegenstände mit so viel Wert wie möglich mitnehmen, ohne die Gewichtsgrenze G zu überschreiten.

Welche Teilmenge ist die beste?

Mathematische Formulierung

Entscheidungsvariablen:

- Einführung von 0/1-Entscheidungsvariablen x_1, \dots, x_n für die Wahl der Gegenstände:

$$x_i = \begin{cases} 0 & \text{falls Gegenstand } i \text{ nicht gewählt wird} \\ 1 & \text{falls Gegenstand } i \text{ gewählt wird} \end{cases}$$

Mathematische Formulierung (für n Gegenstände):

- Maximiere** den Gesamtwert der ausgewählten Gegenstände:

$$\sum_{i=1}^n w_i x_i$$

wobei w_i der Wert des Gegenstands i ist.

Das ist unsere **Zielfunktion**.

- Nebenbedingung:** Das Gesamtgewicht der ausgewählten Gegenstände darf die Kapazität G des Rucksacks nicht überschreiten:

$$\sum_{i=1}^n g_i x_i \leq G$$

wobei g_i das Gewicht des Gegenstands i ist.

- Zusätzliche Bedingung:** Die Entscheidungsvariablen müssen binär sein:

$$x_i \in \{0, 1\} \quad \text{für } i = 1, \dots, n$$

Enumeration (Backtracking)

Enumeration:

- Eine Enumeration aller zulässigen Lösungen für das Rucksackproblem entspricht der Aufzählung aller Teilmengen der n -elementigen Menge (bis auf diejenigen Teilmengen, die nicht in den Rucksack passen).
- Da gibt's 2^n viele mögliche Lösungen

Lösungsvektor und Zielfunktion:

- Zu jedem aktuellen Lösungsvektor $\vec{x} = (x_1, \dots, x_n)$ gehört ein Zielfunktionswert (Gesamtwert von \vec{x}) w_{curr} und ein Gesamtgewicht g_{curr} .
- Die bisher beste gefundene Lösung wird in dem globalen Vektor \vec{x}_{best} und der zugehörige Lösungswert in der globalen Variablen w_{max} gespeichert.

Prinzip:

- Wir folgen wiederum dem Prinzip des Divide-and-Conquer.

Enumerationsalgorithmus

Eingabe: Anzahl z der fixierten Variablen in \vec{x} ; Gesamtwert w_{curr} ; Gesamtgewicht g_{curr} ; aktueller Lösungsvektor \vec{x} .

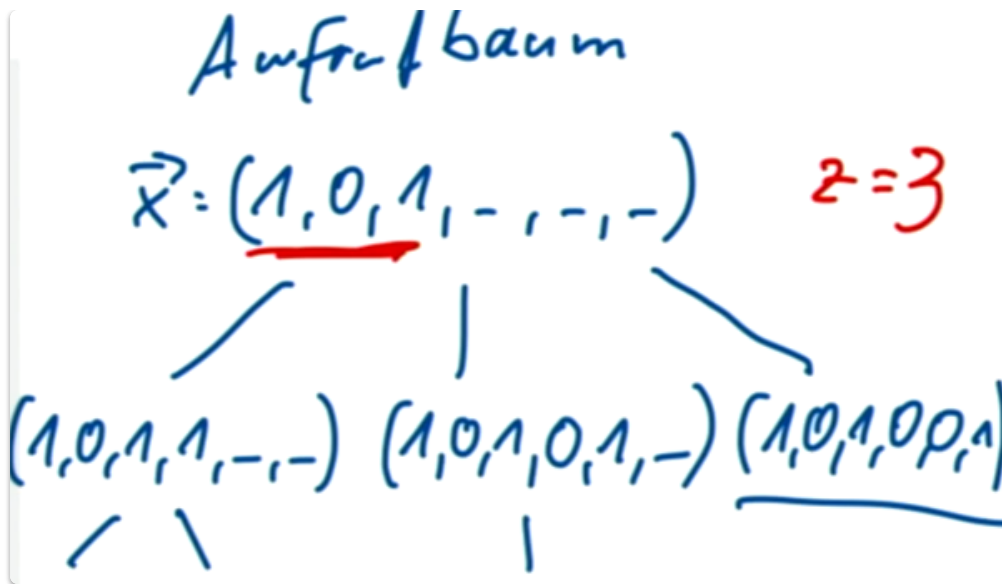
```
Enum( $z, w_{curr}, g_{curr}, \vec{x}$ ):
  if  $g_{curr} \leq G$ 
    if  $w_{curr} > w_{max}$ 
       $w_{max} \leftarrow w_{curr}$ 
       $\vec{x}_{best} \leftarrow \vec{x}$ 
    for  $i \leftarrow z + 1$  bis  $n$ 
       $x_i \leftarrow 1$ 
      Enum( $i, w_{curr} + w_i, g_{curr} + g_i, \vec{x}$ )
       $x_i \leftarrow 0$ 
```

Hinweis:

- w_{max} und \vec{x}_{best} sind globale Variablen.
- Initialisierung: $w_{max} = 0$ und $\vec{x}_{best} = \vec{0}$

1. Schauen ob wir noch Platz haben
2. Schauen ob unsere jetzige Lösung besser ist als die die wir bis jetzt hatten
 1. wenn ja update
3. Rekursiver Aufruf für jeden nächsten Gegenstand den es gibt.

Aufrufbaum:



ⓘ Ablauf des Enumerationsalgorithmus (Backtracking)

Start:

- Der Algorithmus wird mit dem Aufruf `Enum(0, 0, 0, \vec{0})` gestartet.

Rekursiver Aufruf:

- In jedem rekursiven Aufruf wird die aktuelle Lösung \vec{x} bewertet.
- Danach werden die Variablen x_1 bis x_z als fixiert betrachtet.
- Der dadurch beschriebene Teil des gesamten Suchraums wird weiter unterteilt.
- Wir betrachten alle möglichen Fälle, welche Variable x_i (mit $i = z + 1$ bis $i = n$) als nächstes auf 1 gesetzt werden kann.
- Die Variablen x_{z+1} bis x_{i-1} werden gleichzeitig auf 0 fixiert.
- Alle so erzeugten kleineren Unterprobleme werden durch rekursive Aufrufe gelöst.

ⓘ Komplexität

- Es gibt bis zu 2^n rekursive Aufrufe.
- Der Aufwand pro Aufruf (exklusive Rekursion) ist konstant.
- Daher liegt die Laufzeit in $O(2^n)$.

Branch and Bound

Rucksackproblem: Verbesserung der Enumeration

🔗 Idee zur Verbesserung

- Überprüfen von Zwischenlösungen mit $z < n$ fixierten Variablenwerten.

- Man überprüft, ob es noch möglich sein kann, aus dieser Lösung durch Hinzufügen weiterer Gegenstände eine zu erzeugen, die besser ist als die bisher beste gefundene.
- Wenn es **offensichtlich ist, dass keine neue beste Lösung abgeleitet werden kann**, dann sind weitere **rekursive Aufrufe nicht sinnvoll**.
- Das **frühzeitige Abbrechen** führt zu einer Beschneidung des rekursiven Aufrufbaums.
- Das kann eine **erhebliche Beschleunigung** bewirken.

Wie funktioniert das jetzt bei unserem Rucksackproblem

Ansatz

- Berechne obere Schranke U' , und führe den Aufruf nur durch, wenn der Wert $U' > w_{\max}$.
- Sortiere die Gegenstände nach nicht-steigenden Werten $\frac{w_i}{g_i}$.

Wenn w_{\max} größer oder gleich U' ist, dann wissen wir, dass wir nichts besseres mehr finden können und können abbrechen.

Wir wollen die Gegenstände mitnehmen, die vom Wert-Gewichtsquotienten sich am meisten lohnen.

```
Enum( $z, w_{\text{curr}}, g_{\text{curr}}, \vec{x}$ ):
if  $g_{\text{curr}} \leq G$ 
    if  $w_{\text{curr}} > w_{\max}$ 
         $w_{\max} \leftarrow w_{\text{curr}}$ 
         $\vec{x}_{\text{best}} \leftarrow \vec{x}$ 
    for  $i \leftarrow z + 1$  bis  $n$ 
         $U' \leftarrow w_{\text{curr}} + (G - g_{\text{curr}}) \cdot \frac{w_i}{g_i}$ 
        if  $U' > w_{\max}$ 
             $x_i \leftarrow 1$ 
            Enum( $i, w_{\text{curr}} + w_i, g_{\text{curr}} + g_i, \vec{x}$ )
             $x_i \leftarrow 0$ 
```

(Das Rote ist neu!)

Obere Schranke (U'):

- Berechnung der oberen Schranke für den Wert der optimalen Lösung in einem Teilproblem:

$$U' \leftarrow w_{\text{curr}} + (G - g_{\text{curr}}) \cdot \frac{w_i}{g_i}$$

- w_{curr} : Wert der bisherigen Zuteilung.

- $G - g_{curr}$: Verbleibende Kapazität im Rucksack.
- $\frac{w_i}{g_i}$: Wert pro Gewichtseinheit des aktuell untersuchten Gegenstands i .

Erläuterung:

- Die verbleibende Kapazität wird mit dem aktuell untersuchten Gegenstand i komplett (möglicherweise auch mehrmals) aufgefüllt.
- Hierbei kann es auch zu **teilweisen Zuteilungen** kommen (z.B. Gegenstand i wird 1.7 mal eingepackt).
- Da die Gegenstände nach **nicht-steigenden Werten** $\frac{w_i}{g_i}$ **sortiert** sind, haben alle Gegenstände $i + 1, i + 2, \dots, n$ einen relativen Wert kleiner oder gleich dem von i .
- Damit ist die obere Schranke U' **garantiert größer oder gleich** dem Wert der optimalen Lösung für dieses Teilproblem.

Prinzip von Branch-and-Bound: Maximierungsproblem

Branching

- Wie bei der Enumeration üblich wird das Problem rekursiv in kleinere Teilprobleme partitioniert → *Divide-and-Conquer-Prinzip*.

Bounding

- Für jedes Teilproblem wird berechnet:
 - Eine **lokale obere Schranke** U' (upper bound) (liefert uns einen **best case**).
 - Eine **lokale untere Schranke** L' (lower bound) (liefert uns einen **worst case**).
 - Zwischen U' und L' ist eine kleine Lücke gut für den Algorithmus

Abbruch

- Teilprobleme mit $U' \leq L$ (wobei L einer **globalen unteren Schranke** entspricht) brauchen nicht weiter verfolgt zu werden!

Schranken

- Der Wert jeder gültigen Lösung ist eine **untere Schranke**.
- Obere Schranken werden i. A. separat mit einer sogenannten **Dualheuristik** ermittelt.

Rucksackproblem: Verbesserte Schranken

ⓘ Verbesserte untere Schranke

Sortierung:

- Die Gegenstände werden nicht-steigend nach ihrem relativen Wert $\frac{w_i}{g_i}$ sortiert.

Untere Schranke:

- Man durchläuft alle Gegenstände, deren Variablen noch nicht festgelegt sind, in der sortierten Reihenfolge und packt den jeweils aktuellen Gegenstand ein, falls noch Platz im Rucksack ist (Greedy-Algorithmus).

ⓘ Verbesserte obere Schranke

Einfache obere Schranke (wurde weiter vorne beschrieben).

Mögliche Verbesserung:

- Alle Gegenstände, deren Variablen noch nicht festgelegt sind, werden in der sortierten Reihenfolge durchlaufen.
- Man packt alle Gegenstände ein, bis man zu dem ersten Gegenstand kommt, der nicht mehr in den Rucksack passt.
- Sei r die noch freie Kapazität des Rucksacks. Dann zählt man $r \cdot \frac{w_i}{g_i}$ noch zu dem Wert der Gegenstände im Rucksack dazu.
- Der letzte Gegenstand wird daher nur **teilweise** eingepackt.
- Alle verbleibenden Gegenstände werden ignoriert.

Lösung:

- Diese Vorgehensweise liefert in der Regel eine **obere Schranke**, die keine gültige Lösung des Rucksackproblems entspricht.
 - Das macht aber nichts, da wir trz dann eine echte obere Schranke haben
- Falls diese Vorgehensweise zu einer gültigen Lösung führt, dann ist die Lösung (für das betrachtete Teilproblem) **optimal**.

≡ Beispiel - Rucksackproblem

Gegeben: 4 Gegenstände, Rucksackkapazität = 100

Gegenstand	1	2	3	4
Gewicht g_i	32	16	21	50
Wert w_i	80	20	63	100
Verhältnis w_i/g_i	2.5	1.25	3	2

Sortierung:

- Für jeden Gegenstand i das Verhältnis $\frac{w_i}{g_i}$ berechnen.
- Sortierte Reihenfolge der Gegenstände: 3 (3), 1 (2.5), 4 (2), 2 (1.25)

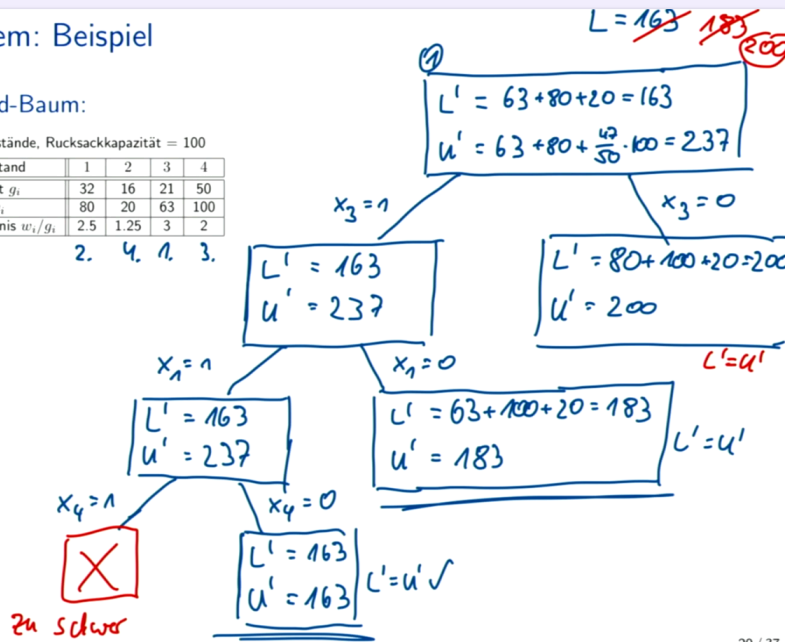
Rucksackproblem: Beispiel

Branch-and-Bound-Baum:

4 Gegenstände, Rucksackkapazität = 100

Gegenstand	1	2	3	4
Gewicht g_i	32	16	21	50
Wert w_i	80	20	63	100
Verhältnis w_i/g_i	2.5	1.25	3	2

2. 4. 1. 3.



Erläuterung:

- **1 (Start):** Noch keine Variablen fixiert. Der gesamte Suchraum wird betrachtet.
- **2:** Es wird angenommen, dass Gegenstand 3 (x_3) fixiert ist (z.B. auf 1 gesetzt), und nur die anderen Gegenstände können ausgewählt werden. Dies führt zu einem Unterbaum des Suchraums.
- Eine Fixierung von Gegenstand 3, 1 und 4 würde zu einer unmöglichen Lösung führen, da sie eine Kapazität von 103 benötigen würde (Annahme: Rucksackkapazität ist geringer). Dieser Pfad im Suchbaum würde frühzeitig verworfen (implizites Bounding).
- **4 und 5:** In diesen Knoten gilt $L = U$ (lokale untere Schranke gleich lokaler oberer Schranke). Dies bedeutet, dass die bestmögliche Lösung in diesem Unterbaum bereits gefunden wurde. Daher brauchen in diesem Unterbaum keine weiteren Gegenstände hinzugefügt werden → **Beschneidung des rekursiven Aufrufbaums**.
- **6:** Hier gilt ebenfalls $L = U$. Der Unterbaum braucht nicht weiter untersucht zu werden. Der Wert der Lösung in diesem Unterbaum (L bzw. U) ist am größten im Vergleich zu anderen Endknoten mit $L = U$.

- **Ergebnis:** Da in Knoten 6 die beste Lösung mit $L = U$ gefunden wurde, und in diesem Fall $x_3 = 0$ war (Gegenstand 3 wurde nicht eingepackt), werden die Gegenstände 1, 2 und 4 eingepackt.

Maximierungsproblem: Vorgehen

Allgemeines Vorgehen


- durch das Fixieren von Variablen oder Hinzufügen von Randbedingungen in Unterprobleme zerteilt --> Lösungsraum wird partitioniert.
- Ist für eine (oder mehrere) dieser Teilmengen die für sie berechnete **obere Schranke** U' **nicht größer** als die **beste überhaupt bisher gefundene untere Schranke** L (= Wert der bisher besten Lösung), braucht man die Lösungen in dieser Teilmenge nicht mehr beachten (**Pruning**).
- Ist die obere Schranke U' **größer** als die beste gegenwärtige untere Schranke L , muss man die Teilmenge weiter zerkleinern (**Branching**).
- Man fährt solange mit der Zerteilung fort, bis für alle Lösungsteilmengen die obere Schranke U' nicht mehr größer ist als die (globale) beste untere Schranke L .

Allgemeiner Algorithmus

Eingabe: Instanz I

```

Branch-and-Bound-Max( $I$ ):
 $L \leftarrow -\infty$  oder Wert einer initialen heuristischen Lösung
 $U \leftarrow \infty$  oder obere Schranke für  $I$  aus Dualheuristik
 $\Pi \leftarrow \{(I, L, U)\}$ 
while  $\exists (I', L', U') \in \Pi$ 
    Entferne  $(I', L', U')$  aus  $\Pi$ 
    if  $U' > L$ 
        Partitioniere  $I'$  in Teilinstanzen  $I_1, \dots, I_k$ 
        Berechne zugehörige gültige heuristische Lösungen  $\rightarrow$  untere Schranken  $L_1, \dots, L_k$ 
        Berechne zugehörige lokale obere Schranken  $U_1, \dots, U_k$  mit Dualheuristik
         $\Pi \leftarrow \Pi \cup \{(I_1, L_1, U_1), \dots, (I_k, L_k, U_k)\}$ 
    if  $\max\{L_1, \dots, L_k\} > L$ 
         $L \leftarrow \max\{L_1, \dots, L_k\}$ 
return beste gefundene Lösung mit Wert  $L$ 
  
```

 *Bounding – Fall $U' \leq L$ nicht weiter interessant.*

 *Branching.*

Maximierungsproblem: Allgemeines Verfahren

Allgemeines Verfahren

- Branch-and-Bound = allgemeines **Metaverfahren**
- Anwendbar auf viele diskrete Optimierungsprobleme

Effizienzfaktoren:

- Wahl der Heuristiken für obere Schranke U' und untere Schranke L'
- Branching-Strategie (wie erfolgt die Zerteilung in Teilprobleme)

- Auswahlregel für nächste Teilinstanz (z.B. nach bestem U')

Branch-and-Bound: Auswahl des nächsten Teilproblems

Auswahl des nächsten Teilproblems:

- Auswahl aus offenen Problemen hat **keinen Einfluss auf Korrektheit** von Branch-and-Bound
- Aber: **großer Einfluss auf praktische Laufzeit**

Beispiele für Strategien:

- **Best-first:** Auswahl des Teilproblems mit der besten (z.B. höchsten bei Maximierung) oberen Schranke.
- **Depth-first:** Auswahl des zuletzt erzeugten Teilproblems.

Best-first

- Wählt Teilproblem mit **bester dualer Schranke** (größte obere Schranke bei Maximierung)
- Ziel: **minimale Anzahl** bearbeiteter Teilprobleme bis zur Optimalität

Depth-first

- Bearbeitet **zuletzt erzeugtes** Teilproblem (analog zu Tiefensuche)
- Liefert oft **schnell gültige Näherungslösung** → dient als initiale untere Schranke
- Kombination möglich: zuerst Depth-first (für L'), dann Best-first (für Effizienz)

Branch and Bound für Minimales Vertex Cover

Allgemeiner Algorithmus

Eingabe: Instanz I

```

Branch-and-Bound-Min( $I$ ):
 $U \leftarrow \infty$  oder Wert einer initialen heuristischen Lösung
 $L \leftarrow -\infty$  oder untere Schranke für  $I$  aus Dualheuristik
 $\Pi \leftarrow \{(I, L, U)\}$ 
while  $\exists (I', L', U') \in \Pi$ 
    Entferne  $(I', L', U')$  aus  $\Pi$ 
    if  $L' < U$ 
        Partitioniere  $I'$  in Teilinstanzen  $I_1, \dots, I_k$ 
        Berechne zugehörige gültige heuristische Lösungen  $\rightarrow$  obere Schranken  $U_1, \dots, U_k$ 
        Berechne zugehörige lokale untere Schranken  $L_1, \dots, L_k$  mit Dualheuristik
         $\Pi \leftarrow \Pi \cup \{(I_1, L_1, U_1), \dots, (I_k, L_k, U_k)\}$ 
        if  $\min\{U_1, \dots, U_k\} < U$ 
             $U \leftarrow \min\{U_1, \dots, U_k\}$ 
return beste gefundene Lösung mit Wert  $U$ 
  
```

- Bounding - Fall $L' \geq U$ nicht weiter interessant.
- Branching.

Minimales Vertex Cover

Branch-and-Bound: Minimales Vertex Cover

Eingabe: Graph $G = (V, E)$ und Knotenmenge $C = \emptyset$

Knoten fürs Vertex Cover



```

MinVertexCover-BranchAndBound( $G, C$ ):
 $U \leftarrow$  gültige heuristische Lösung für  $(G, C)$  mit Greedyheuristik
 $L \leftarrow$  untere Schranke für  $(G, C)$  mit Matchingheuristik
 $\Pi \leftarrow \{((G, C), L, U)\}$ 
while  $\exists I' \in \Pi$ 
    Entferne  $I' = ((G', C'), L', U')$  aus  $\Pi$ 
    if  $L' < U$ 
         $u_{\max} \leftarrow$  Knoten mit maximalem Grad in  $G'$ 
        Erzeuge Teilinstanzen  $I_1 = (G' - \{u_{\max}\}, C' \cup \{u_{\max}\})$  und
         $I_2 = (G' - \{u_{\max}\} - N(u_{\max}), C' \cup N(u_{\max}))$   $\rightarrow$  wähle alle Nachbarn von  $u_{\max}$ 
        Berechne für  $I_1, I_2$  gültige heuristische Lösungen mit Greedyheuristik  $\rightarrow$  obere Schranken  $U_1, U_2$ 
        Berechne für  $I_1, I_2$  lokale untere Schranken  $L_1, L_2$  mit Matchingheuristik
         $\Pi \leftarrow \Pi \cup \{(I_1, L_1, U_1), (I_2, L_2, U_2)\}$ 
        if  $\min\{U_1, U_2\} < U$ 
             $U \leftarrow \min\{U_1, U_2\}$ 
return beste gefundene Lösung mit Wert  $U$ 
  
```

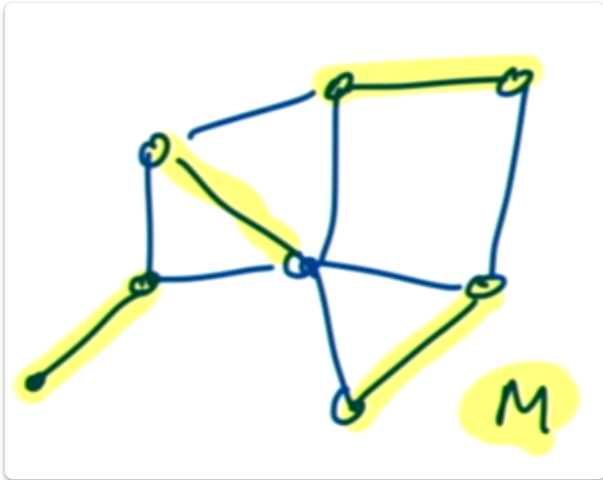
- alle Nachbarknoten von u_{\max}

Untere Schranke: Wird mit Hilfe eines Matchings bestimmt.

- Sei ein Graph $G = (V, E)$ gegeben.
- Eine Menge $M \subseteq E$ heißt **Matching**, wenn keine zwei Kanten aus M einen Knoten gemeinsam haben.

Nicht erweiterbares Matching (maximales Matching):

- Ein Matching M ist **nicht erweiterbar** (maximal), wenn es keine Kante $e \in E \setminus M$ gibt, sodass $M \cup \{e\}$ ein gültiges Matching ist.
- Ein nicht erweiterbares Matching ist **nicht notwendigerweise** ein größtes Matching (Maximum Matching).
- Ein nicht erweiterbares Matching kann mit einem Greedy-Verfahren gefunden werden.

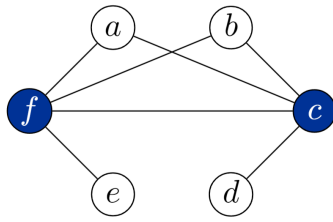


Berechnung der unteren Schranke L' für die Instanz (G', C') :

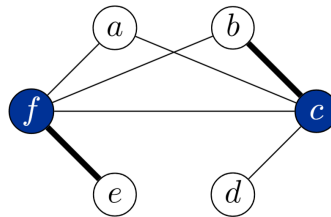
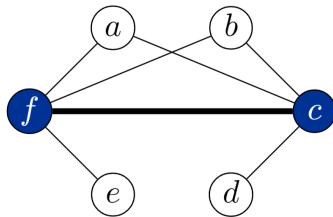
- Man wählt für L' die Größe eines nicht erweiterbaren Matchings.
 - Dabei wird zunächst eine beliebige Kante $e = (u, v)$ gewählt und dann die Knoten u und v und ihre inzidenten Kanten aus G' entfernt.
 - Man fährt mit dieser Prozedur fort, bis keine Kante mehr vorhanden ist.
 - Die Anzahl der gewählten Kanten entspricht der Größe des Matchings.
- Kanten in einem Matching haben keine Knoten gemeinsam.
- Ein Vertex Cover muss zumindest einen Knoten für jede Kante in einem Matching wählen.
- Daher ist die Größe eines Matchings von G' eine untere Schranke für die Größe eines Vertex Covers der Instanz (G', C') .

≡ Beispiel für untere Schranke

Vertex Cover: Minimales Vertex Cover mit $k = 2$



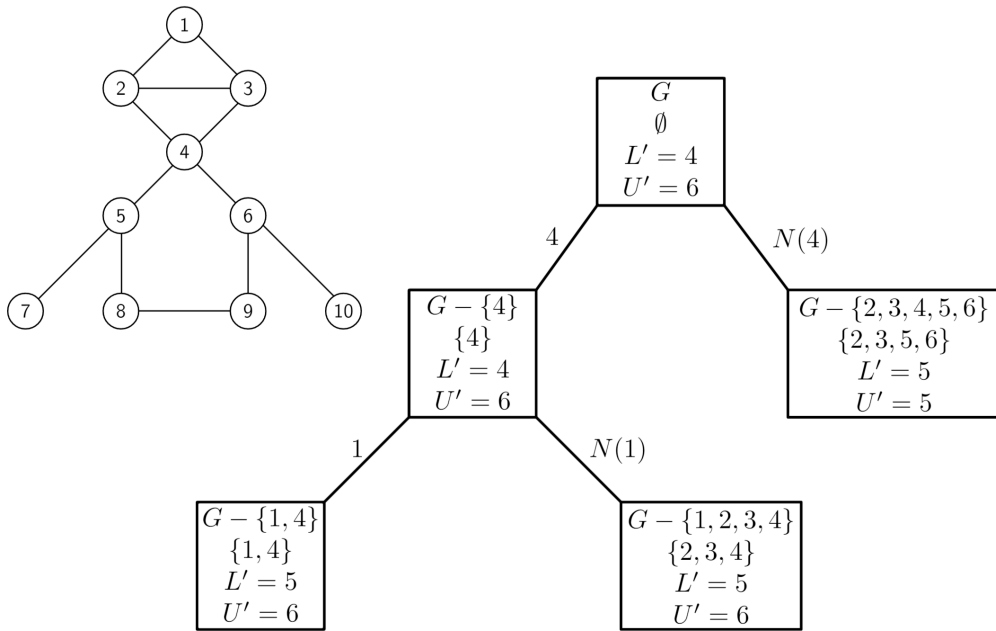
Greedy-Matching: 2 Beispiele für Greedy-Matching (fett eingezeichnet Kanten).



Obere Schranke U' : Wird mit Hilfe eines Greedy-Algorithmus bestimmt.

- Sei ein Graph $G' = (V', E')$ und eine Knotenmenge C' gegeben (die bereits im Vertex Cover enthaltene Knoten der aktuellen Teilinstanz).
- Initialisiere eine Menge $S \leftarrow \emptyset$.
- Sortiere die Knoten in V' nicht-steigend nach dem Knotengrad (Anzahl der inzidenten Kanten).
- Durchlaufe V' in dieser Reihenfolge, solange der Graph G' noch Kanten enthält.
 - Füge den Knoten u mit dem höchsten Knotengrad zu S hinzu.
 - Entferne u und alle seine inzidenten Kanten aus G' .
 - Passe die Reihenfolge der verbleibenden Knoten in V' gegebenenfalls an (da sich Knotengrade ändern können).
- Die Menge S ist ein Vertex Cover für den verbleibenden Graphen G' .
- Daher ist $|C'| + |S|$ eine obere Schranke für die Größe eines minimalen Vertex Covers der Teilinstanz (G', C') (und damit auch des Eingabegraphen G).

≡ Minimales Vertex Cover Beispiel



Branch-and-Bound: Zusammenfassung

- Branch-and-Bound ist eine allgemein für kombinatorische Optimierungsprobleme einsetzbare Technik zur Berechnung exakter (optimaler) Lösungen.
- Sie funktioniert sowohl für Maximierungs- als auch für Minimierungsprobleme.
- Praktisch lassen sich oft hohe Beschleunigungen erreichen, die worst-case Laufzeit bleibt jedoch wie bei der Enumeration aller möglichen Lösungen (oft exponentiell).

Vorgehen beim Entwurf von Branch-and-Bound Algorithmen

- Wie lassen sich (Teil-)Instanzen des Problems ausdrücken?
- Was sind gute Heuristiken für untere und obere Schranken?
- Wie wird eine (Teil-)Instanz in weitere Teilinstanzen partitioniert (Branching)?
- Welche Teilinstanz wird im nächsten Schritt ausgewählt?

