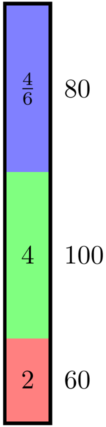
Übersicht



$= 240$

Greedy-Algorithmen

- Wie bei Dynamic Programming beinhalten optimale Lösungen optimale Teillösungen
- Anders als bei Dynamic Programming gilt die greedy choice property: eine lokal optimale Lösung ist stets Teil einer global optimalen Lösung
- Korrektheitsbeweise über Theorie der Matroide, Greedoide, Matroideinbettungen, ...

Übersicht

Flußalgorithmen

Modelliere Optimierungsproblem als Flußproblem.

Varianten:

- Maximaler Fluß
- Fluß mit maximalen Gewinn
- Matchings maximaler Kardinalität
- Matchings maximalem Gewichtes

Übersicht

6 Paradigmen
- Divide-and-Conquer
- Dynamisches Programmieren
- Greedy-Algorithmen
- Flüsse
- Lineares Programmieren
- Randomisierung
- Backtracking
- Branch-and-Bound
- A*-Algorithmus
- Heuristiken

Lineares Programmieren

- Viele Probleme können als Maximierung einer linearen Funktion mit linearen Ungleichungen als Nebenbedingungen ausgedrückt werden.
- Mehr dazu in der Vorlesung Effiziente Algorithmen

Übersicht

Randomisierte Algorithmen

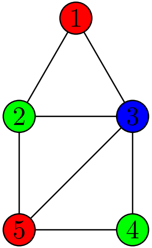
- Beispiele in der Vorlesung nutzen Zufall, um den *worst-case* in Form eines *adversary* mit hoher Wahrscheinlichkeit zu vermeiden, sowie um den Algorithmus und/oder die Analyse zu vereinfachen
- Eine Vielzahl von weiteren Techniken zum Entwurf in der Vorlesung Randomisierte Algorithmen

Übersicht

Backtracking

- Tiefensuche auf dem Raum der möglichen Lösungen
- ein natürlicher Weg, schwere Entscheidungsprobleme zu lösen
- Ausdruck "backtrack" von D.H. Lehmer in den 50er Jahren
- Verfeinerung der *brute force*-Suche
- In der Praxis gut bei Problemen wie 3-Färbbarkeit oder 0/1-Knapsack

Backtracking



Algorithmus

function backtrack(v1, ..., vi) :
if(v1, ..., vi) is a solution **then return**(v1, ..., vi) **fi**;
for each v **do**
  if(v1, ..., vi, v) is acceptable vector **then**
    sol := backtrack(v1, ..., vi, v);
    if sol $\neq$ () **then return** sol **fi**
  fi
od;
return()

Übersicht

Branch-and-Bound

- 1960 vorgeschlagen von A.H. Land und A.G. Doig für Linear Programming, allgemein nützlich für Optimierungsprobleme
- branching auf einem Suchbaum wie bei Backtracking
- bounding und pruning führt zum Auslassen von uninteressanten Zweigen
- Geeignet für Spiele wie Schach oder schwere Optimierungsprobleme

Beispiel: 0/1-Knapsack

- Verzweige, je nachdem, ob ein Gegenstand mitgenommen wird oder nicht, in einer festen Reihenfolge
- Eine Teillösung $I \subseteq \{1, \ldots, k\}$ ist zulässig, solange $\sum_{i \in I} c_i \leq C$
- Wenn I nicht zulässig ist, kann auch keine Lösung, die I enthält, zulässig sein
- Außerdem ist $\sum_{i \in I} v_i + \sum_{i \in \{k+1,\ldots,n\}} v_i$ eine obere Schranke für den Profit jeder Lösung, die I erweitert, wenn wir in I alle Gegenstände bis zum k. berücksichtigt haben
- Also können wir alle Zweige abschneiden, die entweder nicht zulässig sind oder deren obere Schranke unter dem bis jetzt optimalen Profit liegt.

Das Problem des Handlungsreisenden

Wir untersuchen das folgende Problem:

Definition (TSP)

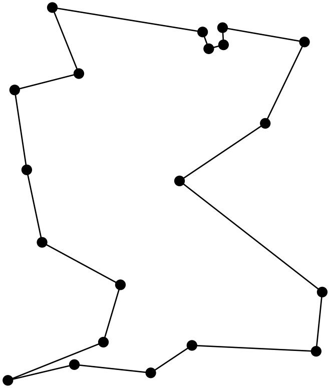
Eingabe: Ein Graph $G = (V, E)$ mit Kantengewichten $\mathit{length} : E \to \mathbf{Q}$
Ausgabe: Eine geschlossene Rundreise über alle Knoten mit minimaler Gesamtlänge.

Wir denken zum Beispiel an n Städte, die alle besucht werden müssen.

Was ist die minimale Länge einer Tour, die alle Städte besucht und im gleichen Ort beginnt und endet?

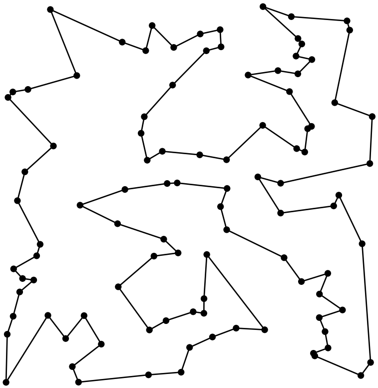
Das Problem des Handlungsreisenden

Ein kleines Beispiel:



Das Problem des Handlungsreisenden

Ein mittelgroßes Beispiel:



Der naive Ansatz

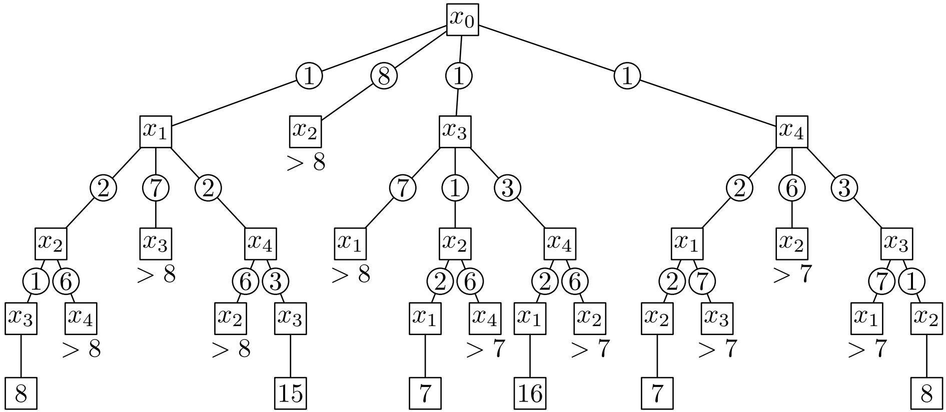
Der einfachste Ansatz dieses Problem zu lösen besteht darin, alle Möglichkeiten durchzuprobieren.

Wieviele Möglichkeiten gibt es?

Es gibt $(n-1)!$ mögliche Rundreisen, da es $n!$ Permutationen gibt.
Wir könnten alle Rundreisen durchprobieren, ihre Kosten berechenen und die billigste auswählen.

Es eine schnellere Lösung mit Hilfe von dynamischer Programmierung. Wir versuchen es mit Branch-and-Bound.

Branch and Bound - Beispiel TSP



Übersicht

A*-Suche

- 1968 von Peter Hart, Nils Nilsson, and Bertram Raphael als A-search (A*-search mit optimaler Heuristik)
- Graphsuchverfahren mit Anwendungen in KI (Planung) und Suche in großen Suchräumen
- Generalisierung von Dijkstras Algorithmus und best-first search
- Knotenexpansion in Reihenfolge von $f(x) = g(x) + h(x)$ mit tatsächlichen Kosten g und Heuristik h

Eigenschaften der A*-Suche

Theorem

Der A-Algorithmus findet einen optimalen Pfad, wenn h optimistisch ist.*

Beweis.

Wenn der Algorithmus terminiert, ist die Lösung billiger als die optimistischen Schätzungen für alle offenen Knoten. $\square$

Eigenschaften der A*-Suche

Theorem

Für jede Heuristik h betrachtet die A-Suche die optimale Anzahl von Knoten unter allen zulässigen Algorithmen.*

Beweis.

Nimm an, Algorithmus B mit Heuristik h betrachtet einen Knoten x nicht, der in A* einmal offen war. Dann beträgt $h(x)$ höchstens Kosten des Lösungspfades von B und kann B nicht ausschließen, daß es einen Pfad über x gibt, der billiger ist. Also ist B nicht zulässig. □

A* - Kosten

- Laufzeit: Im allgemeinen Anzahl der betrachteten Knoten exponentiell in der Länge des optimalen Pfades
- stark abhängig von der Güte der Heuristik: polynoniell wenn $|h(x) - h^*(x)| \in O(\log h^*(x))$
- Problem: Speicherplatz ebenfalls exponentiell
- Varianten/Verbesserungen: IDA*,MA*,SMA*,AO*, Lifelong Learning A*, ...

Übersicht

Heuristiken

- Hier: Verfahren, die in der Praxis oft funktionieren, aber für die man keine worst-case-Schranken zeigen kann
- Beispiele: Lokale Suche, Simulated Annealing, Genetische Algorithmen

Lokale Suche

- Heuristik für Optimierungsprobleme
- Definiere Nachbarschaft im Suchraum
- Gehe jeweils zum lokal optimalen Nachbarn
- Problem: lokale Minima

Lokale Suche für TSP