CARL VON OSSIETZKY universität OLDENBURG

Sommersemester 2020

Objektorientierte Modellierung und Programmierung

Dr. Christian Schönberg

CARL VON OSSIETZKY universität OLDENBURG

Lösungsstrategien II

CARL VON OSSIETZKY universität OLDENBURG

Lernziele

- TSP
- Aufzählungsmethoden
  - Backtracking (Wiederholung)
  - Branch & Bound
- Relaxation, Approximation
- Heuristiken
  - Lokale Suche
  - Tabu Suche
- Evolutionäre Algorithmen

Optimierungsprobleme (Wiederholung)

- Optimierungsprobleme
  - gegeben: ein Problem O
  - gesucht: die beste (optimale) Lösung für O
- Formal:
  - Lösungsraum S: Menge aller Lösungen für O
  - Gütefunktion q: $S \rightarrow R^+$ ($q(x)$ = Qualität der Lösung x aus S)
  - Finde die Lösung $x \in S$, so dass $q(x)$ optimal (maximal oder minimal) ist
    - $\max\{ q(x) \mid x \in S \}$ oder $\min\{ q(x) \mid x \in S \}$
- Für viele Optimierungsprobleme gibt es **keine** polynomiale Lösung
- Beispiel: Route des Paketlieferanten, die an allen Adressen vorbeikommt und im Lager anfängt und aufhört

Travelling Salesman Problem (TSP)

- Gegeben
  - Liste von Orten
  - paarweise Kosten zwischen Orten (Entfernung, Reisezeit, Reisekosten, …)
- Gesucht
  - billigste Route
  - von einem Startpunkt,
  - die jeden anderen Ort besucht und
  - wieder am Startpunkt endet

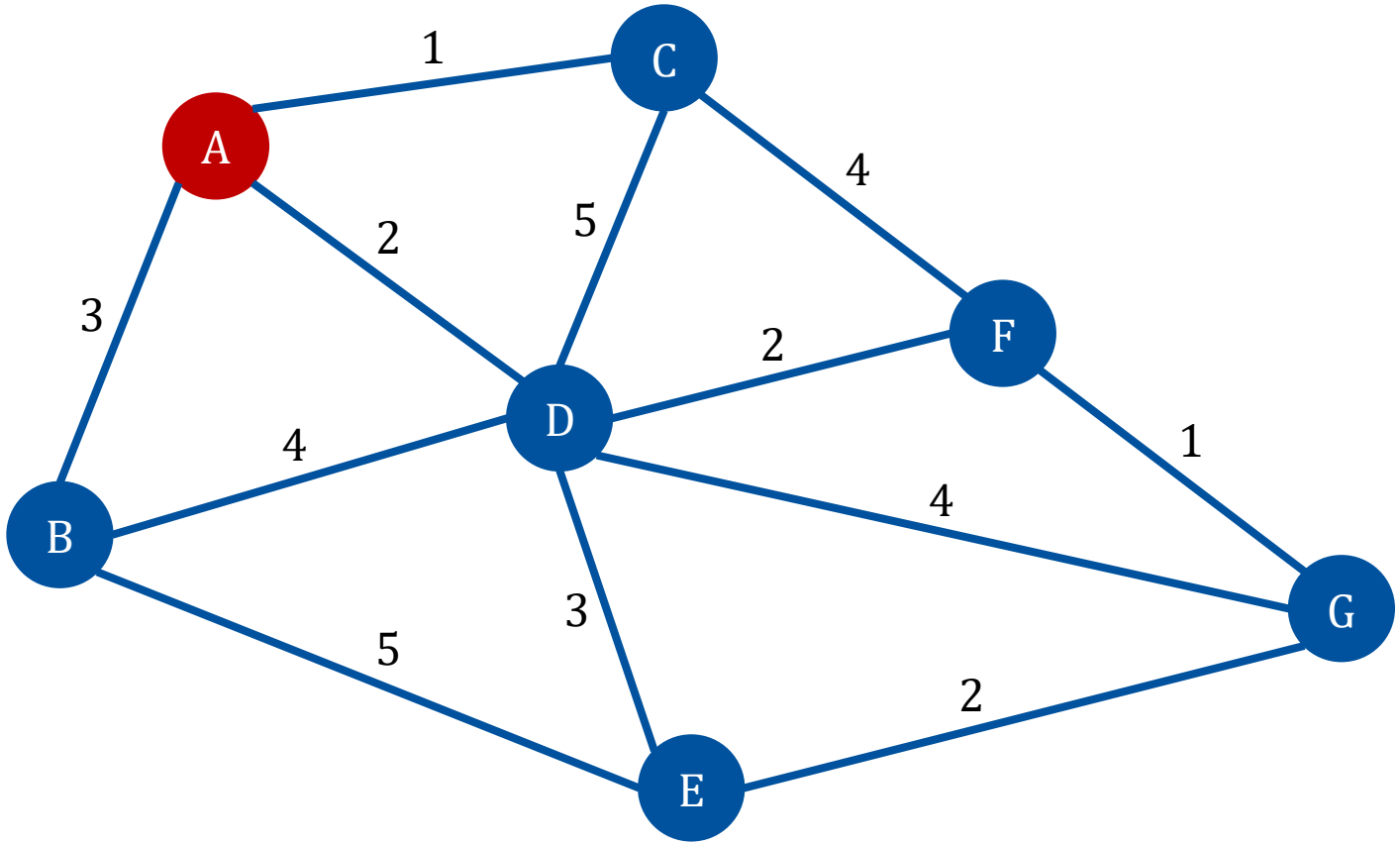
  ➢Rundreise
- NP-vollständig
- Weite Übertragbarkeit
  - Planung, Logistik, Chipdesign, …

TSP (formal)

- Gegeben sei ein ungerichteter, gewichteter Graph $G = (V, E)$ mit Kantengewichten $c(e) = c(u, v)$ und $|V| = n$
- gesucht ist der Pfad $(v_1, v_2, \ldots, v_n, v_1)$, der
  - jeden Knoten $v_i \in V$ genau einmal enthält
  - geschlossen ist (Anfangsknoten = Endknoten)
  - so dass $\sum_{i=1}^{n-1} c(v_i, v_{i+1}) + c(v_n, v_1)$ minimal unter allen möglichen Pfaden ist
- Verschärfung: für alle Gewichte gilt zusätzlich die Dreiecksungleichung, d.h. für alle $i, j, k \in \{ 1, \ldots, n \}$: $c(v_i, v_j) \leq c(v_i, v_k) + c(v_k, v_j)$

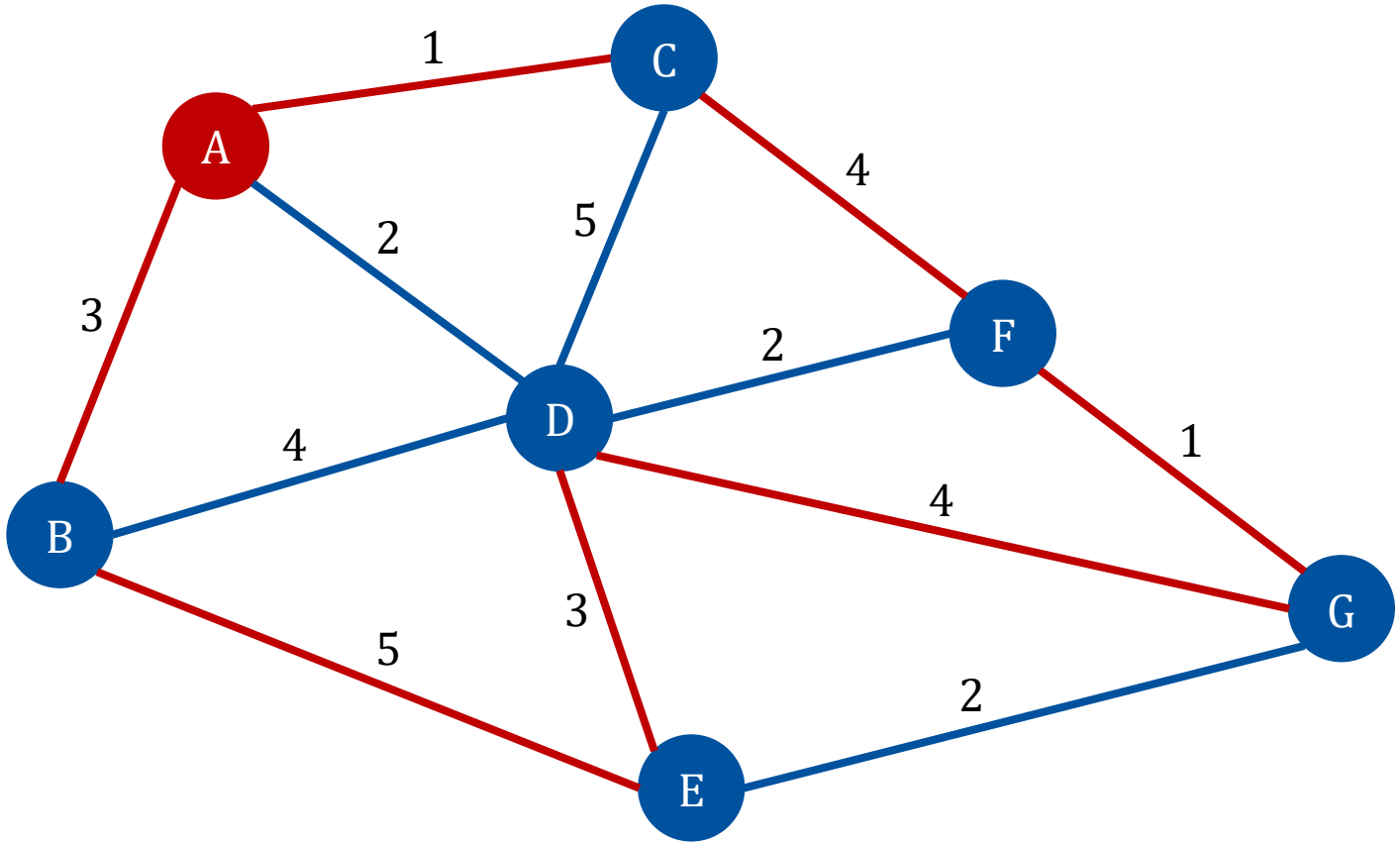
Beispiel: TSP

Gesucht: Kürzester Weg von A nach A über jeden anderen Knoten



Beispiel: TSP

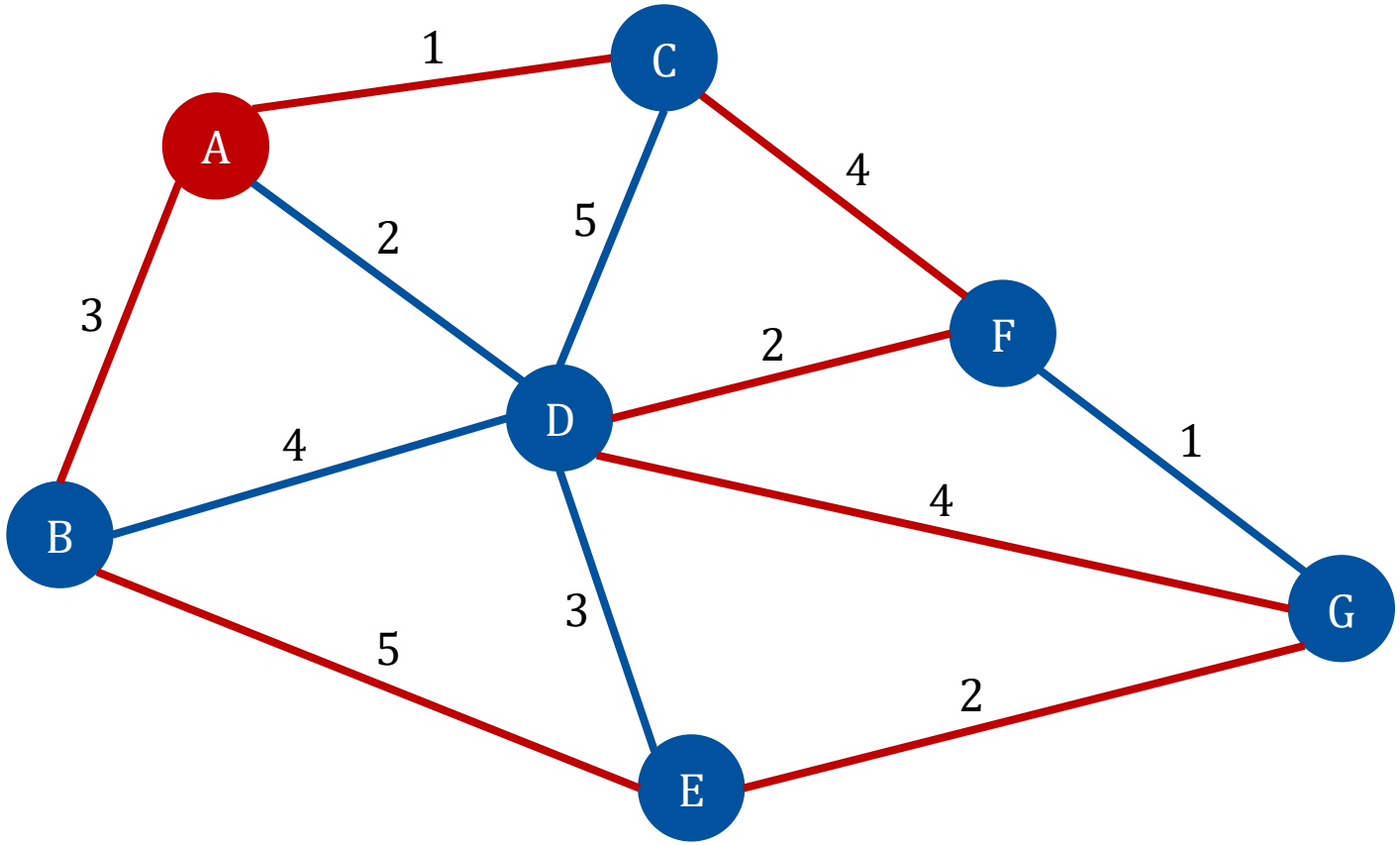
Gesucht: Kürzester Weg von A nach A über jeden anderen Knoten
Kandidat 1: A, B, E, D, G, F, C, A
Kosten: $3 + 5 + 3 + 4 + 1 + 4 + 1 = 21$



Beispiel: TSP

Gesucht: Kürzester Weg von A nach A über jeden anderen Knoten
Kandidat 1: A, B, E, D, G, F, C, A
Kosten: $3 + 5 + 3 + 4 + 1 + 4 + 1 = 21$

Kandidat 2: A, C, F, D, G, E, B, A
Kosten: $1 + 4 + 2 + 4 + 2 + 5 + 3 = 21$

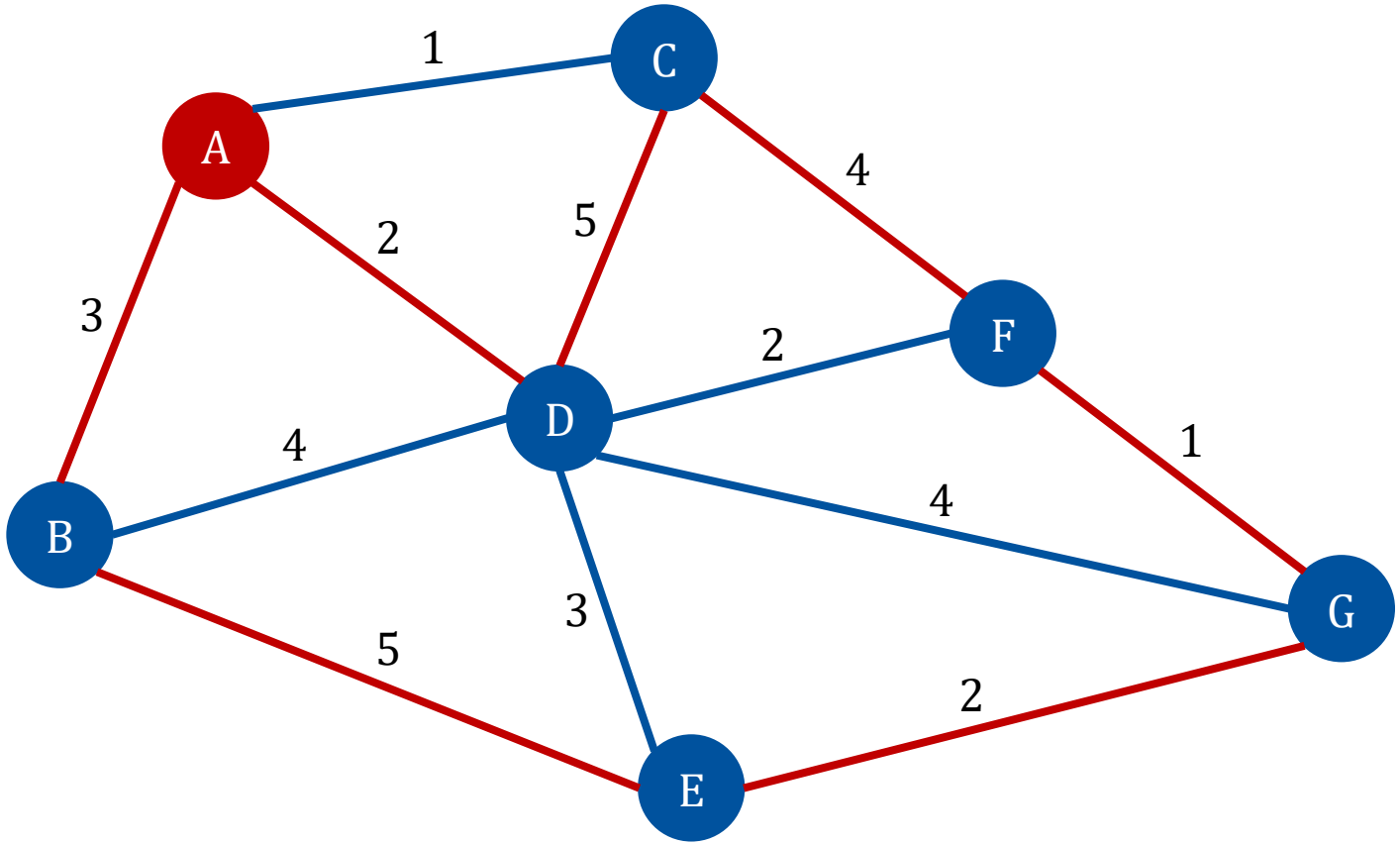


Beispiel: TSP

Gesucht: Kürzester Weg von A nach A über jeden anderen Knoten
Kandidat 1: A, B, E, D, G, F, C, A
Kosten: $3 + 5 + 3 + 4 + 1 + 4 + 1 = 21$

Kandidat 2: A, C, F, D, G, E, B, A
Kosten: $1 + 4 + 2 + 4 + 2 + 5 + 3 = 21$

Kandidat 3: A, D, C, F, G, E, B, A
Kosten: $2 + 5 + 4 + 1 + 2 + 5 + 3 = 22$



CARL VON OSSIETZKY universität OLDENBURG

Aufzählungsmethoden

Aufzählungsmethoden

- Optimierungsprobleme: Suche die beste Lösung in einem (großen) Lösungsraum
- Aufzählungsstrategien
  - zähle alle Lösungen auf, prüfe deren Güte und wähle die beste aus
  - möglich in kleinem Lösungsraum
- Problem: Lösungsräume können schnell sehr groß werden
  - Beispiel: $n!$ mögliche Permutationen von n Zahlen
    - $n = 5$ → 120 Lösungen
    - $n = 30$ → 10^{32} Lösungen
      (bei 1000 Lösungen/Sekunde: 1 Trilliarde Jahre)

Aufzählungsmethoden (2)

- Vorgehen
  - strukturiere den Suchprozess in Teilzustände und mögliche Entscheidungsalternativen
    → Entscheidungsbaum
  - feste Ordnung auf den Entscheidungsalternativen, d.h. auf den Nachfolgern eines Knotens im Entscheidungsbaum
- Mögliche Ziele
  - Konstruktion einer oder aller Lösungen
- Backtracking-Strategie
  - Systematische Suche im gesamten Lösungsraum
- Branch & Bound-Strategie
  - Wie Backtracking, aber vorzeitiger Abbruch der Suche in Teilbäumen, die keinen Erfolg versprechen

Backtracking-Strategie

- Beginne mit trivialem Teilproblem mit Lösung s_n
- Wähle die erste Entscheidung d_n und erweitere s_n durch d_n zu s_{n-1}
- Fahre fort, bis durch die resultierende Entscheidungsfolge $(d_n, \ldots, d_1)$ eine vollständige Lösung s_0 konstruiert wurde
- Falls auf dem Weg keine Lösung konstruiert werden kann oder falls alle Lösungen konstruiert werden sollen:
  - revidiere die letzte Entscheidung
    (kehre von s_k zurück zu s_{k+1})
  - falls nun eine andere Entscheidung möglich ist
    → wähle nächste mögliche Entscheidung
    sonst: wiederhole Revidierungsschritt

Beispiel: Permutationen

- Gegeben: n Zahlen, z.B. $1, \ldots, n$
- Erzeuge alle Permutationen $p = (z_1, \ldots, z_n)$
- Vorgehen
  - gegeben: Teillösung $s_i = (z_1, z_2, \ldots, z_i)$
    → Entscheidung: Füge nächste, unbenutzte Zahl z_{i+1} an
  - Ordnung auf Entscheidungen durch Wert von z
    ← revidieren: Lösche die letzte Zahl aus s_i
- Initiale Teillösung: $s_n = ()$

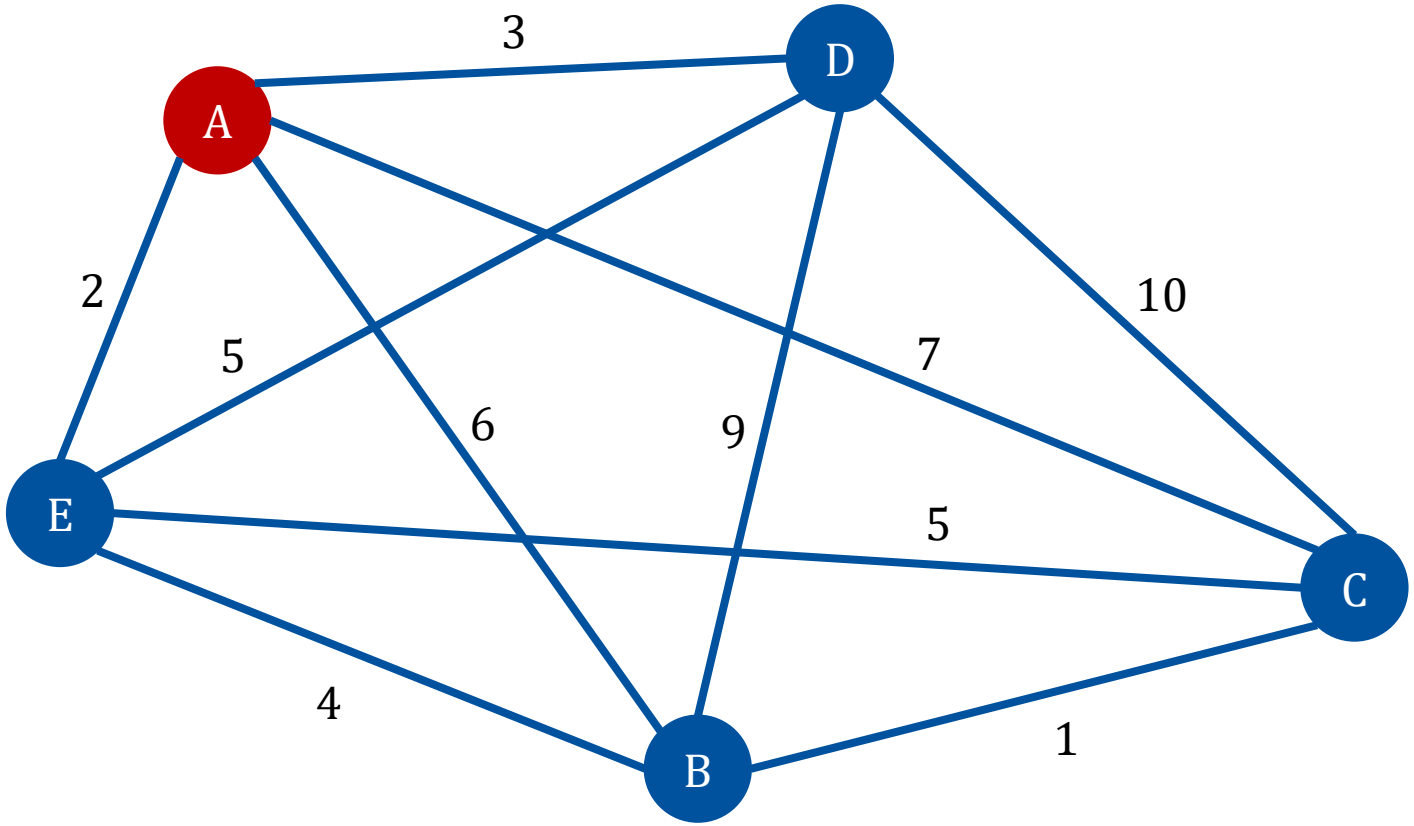
- Sei $n = 3$:
  - Init $s = ()$
  - → 1 $s = (1)$
  - → 2 $s = (1, 2)$
  - → 3 $s = (1, 2, 3)$ OK
  - ← 3 $s = (1, 2)$
  - ← 2 $s = (1)$
  - → 3 $s = (1, 3)$
  - → 2 $s = (1, 3, 2)$ OK
  - ← 2 $s = (1, 3)$
  - ← 3 $s = (1)$
  - ← 1 $s = ()$
  - → 2 $s = (2)$
  - → 1 $s = (2, 1)$
  - …

Beispiel: TSP

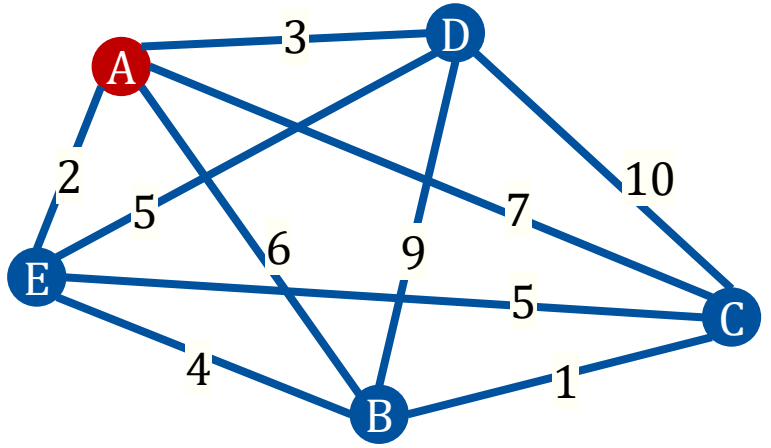
- Festlegungen
  - feste Ordnung auf den zu besuchenden Städten
    - z.B. alphabetisch
  - Start- und Endstadt: A
  - Initiale Lösung: $s_n = (A)$, Weglänge $= 0$
- In jedem Schritt
  - konstruiere nächste Teillösung s_{i-1}:
    Hänge die nächste, nicht besuchte Stadt an die Folge s_i
    - Rundtour komplett? Güte berechnen

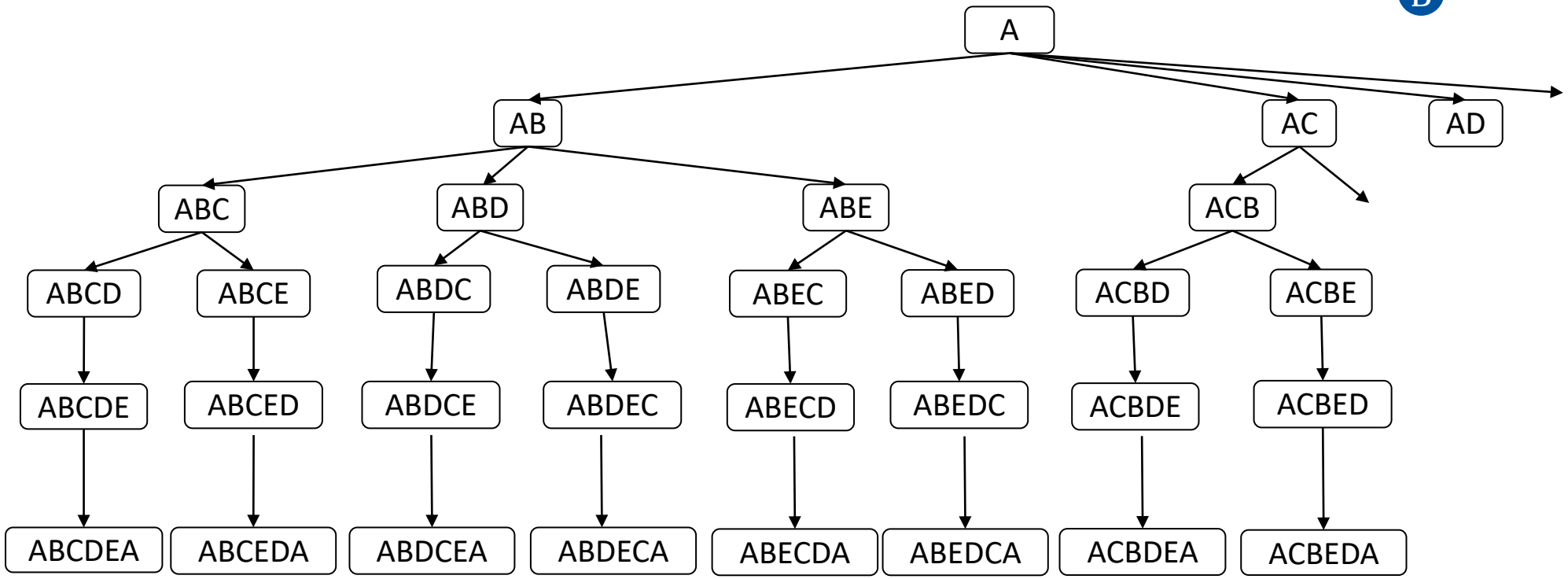
    ➢ ggf. als beste Tour merken
  - falls keine Entscheidung mehr möglich ist oder alle Lösungen konstruiert werden sollen
    - revidiere die letzte Entscheidung: entferne die letzte Stadt

Beispiel: TSP mit $n = 5$



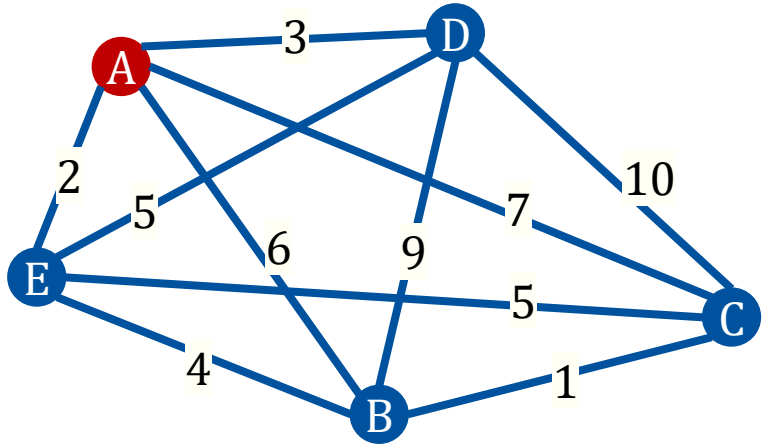
Beispiel: TSP mit n = 5

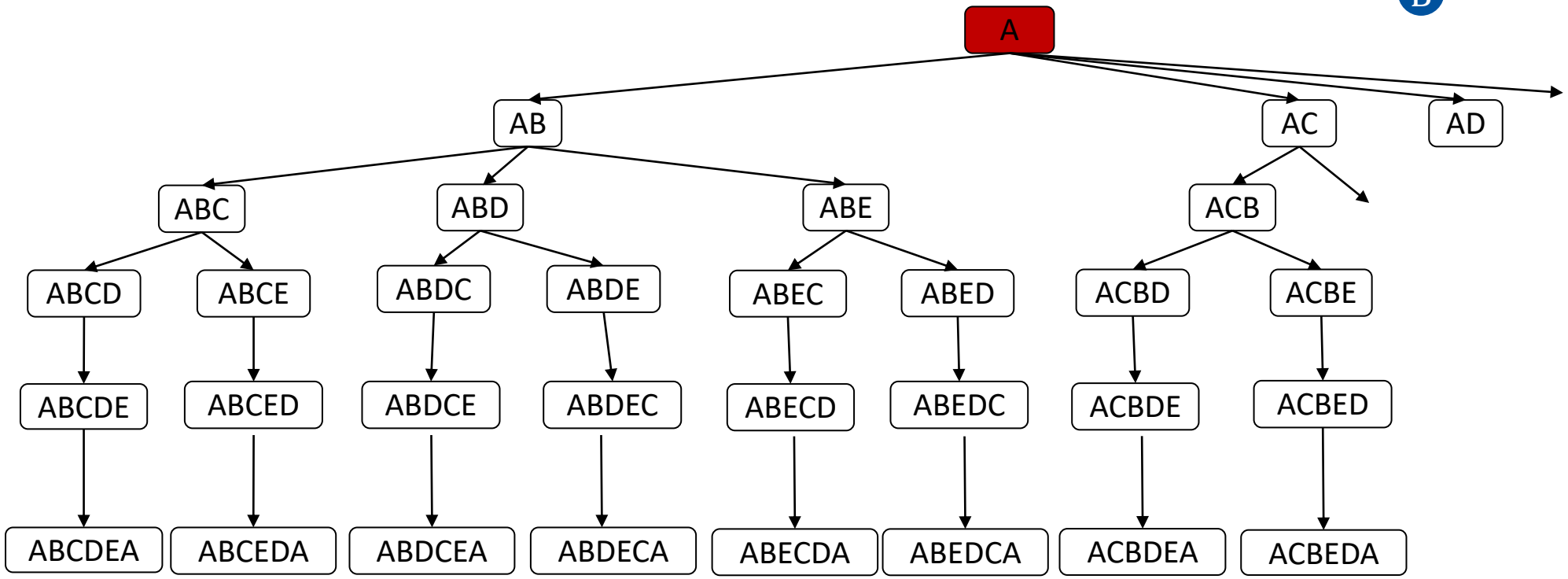




Länge:

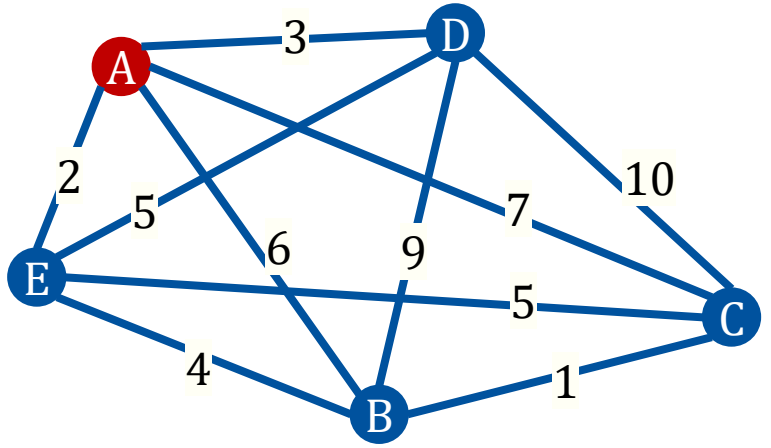
Beispiel: TSP mit n = 5

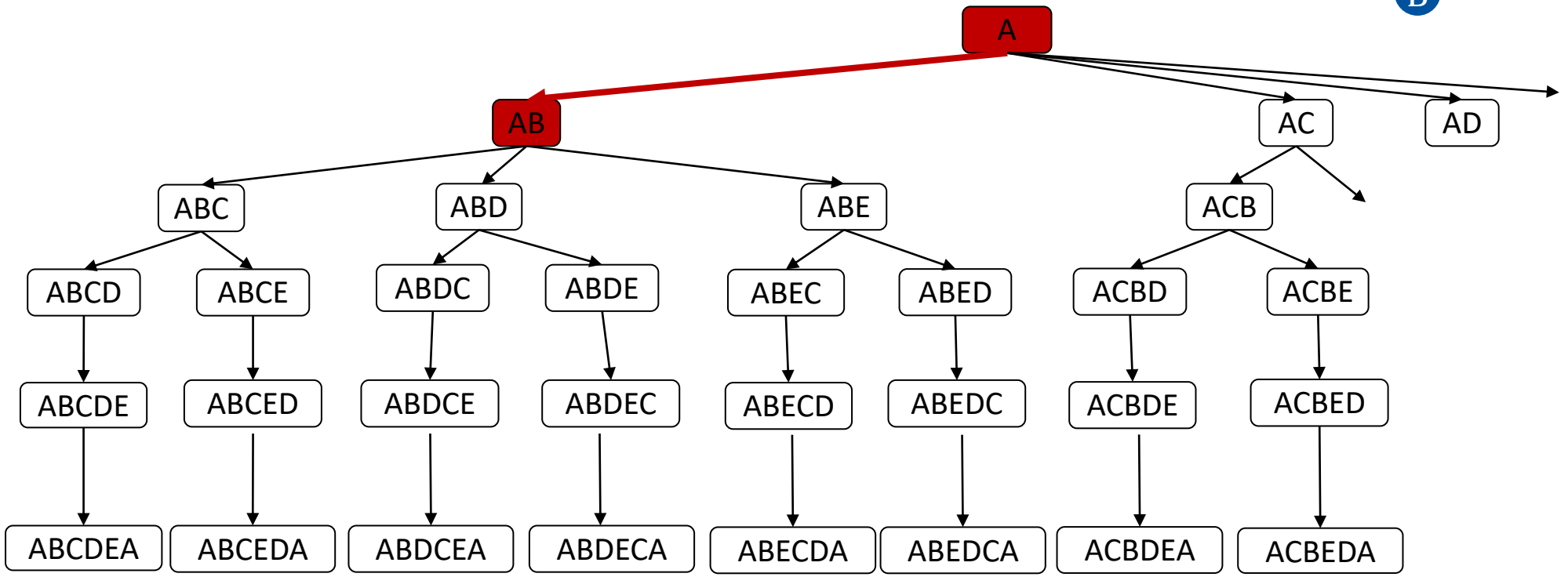




Länge:

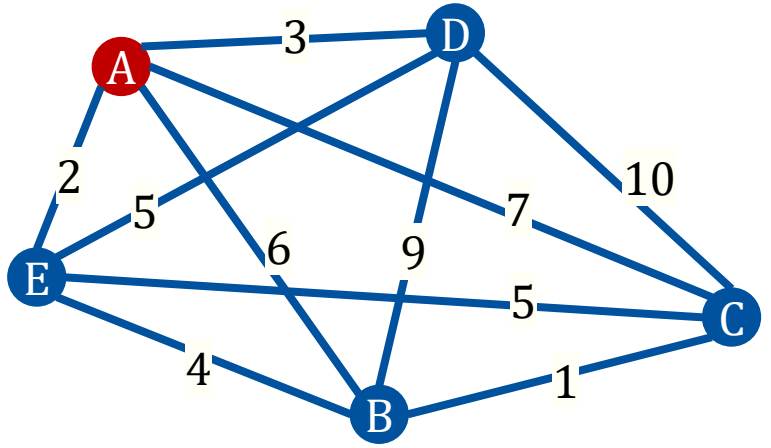
Beispiel: TSP mit n = 5

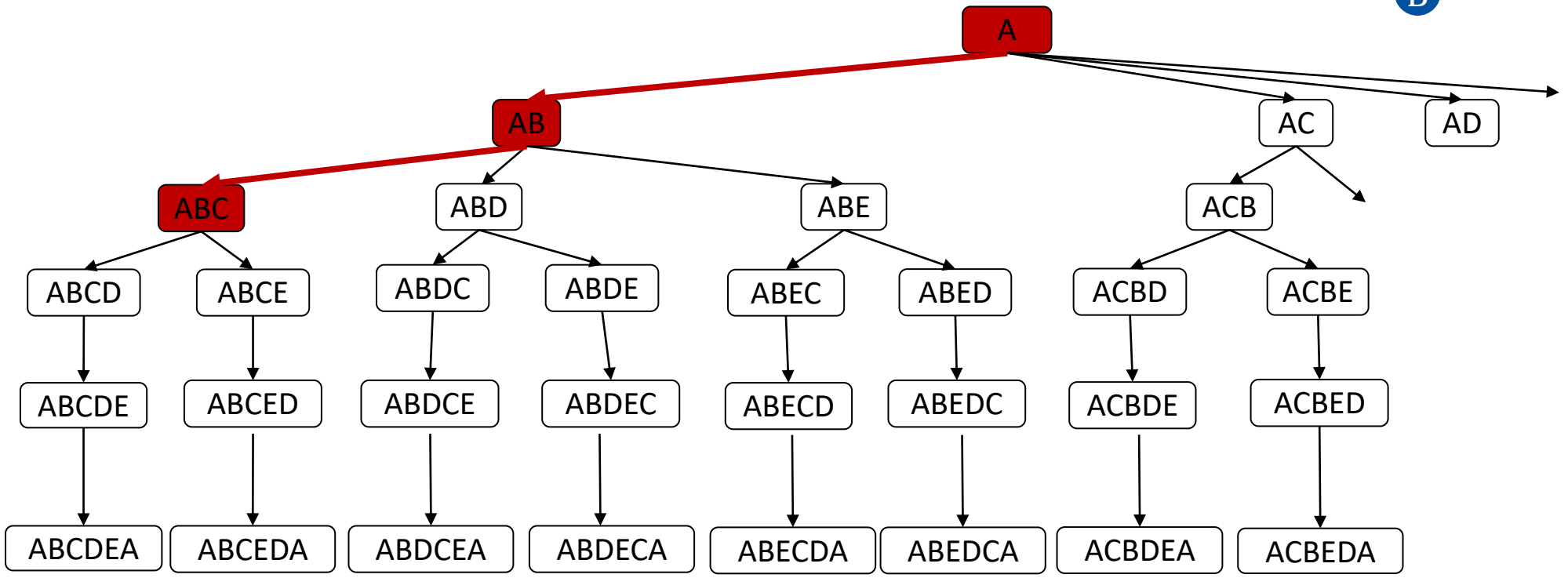




Länge:

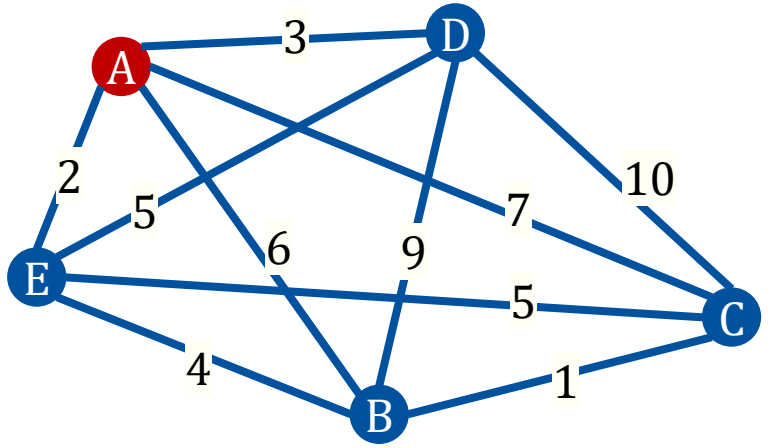
Beispiel: TSP mit n = 5

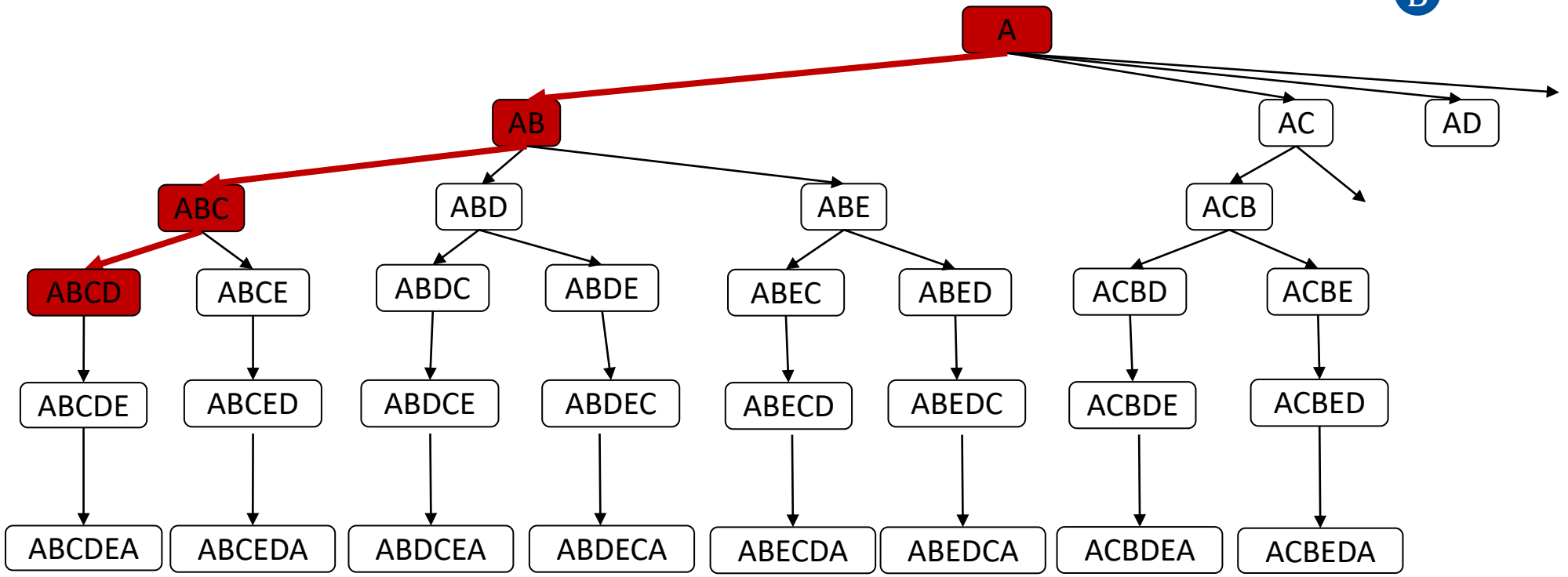




Länge:

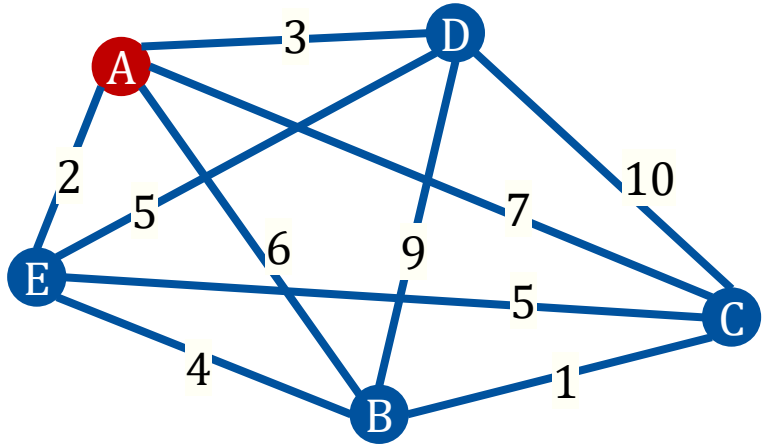
Beispiel: TSP mit n = 5

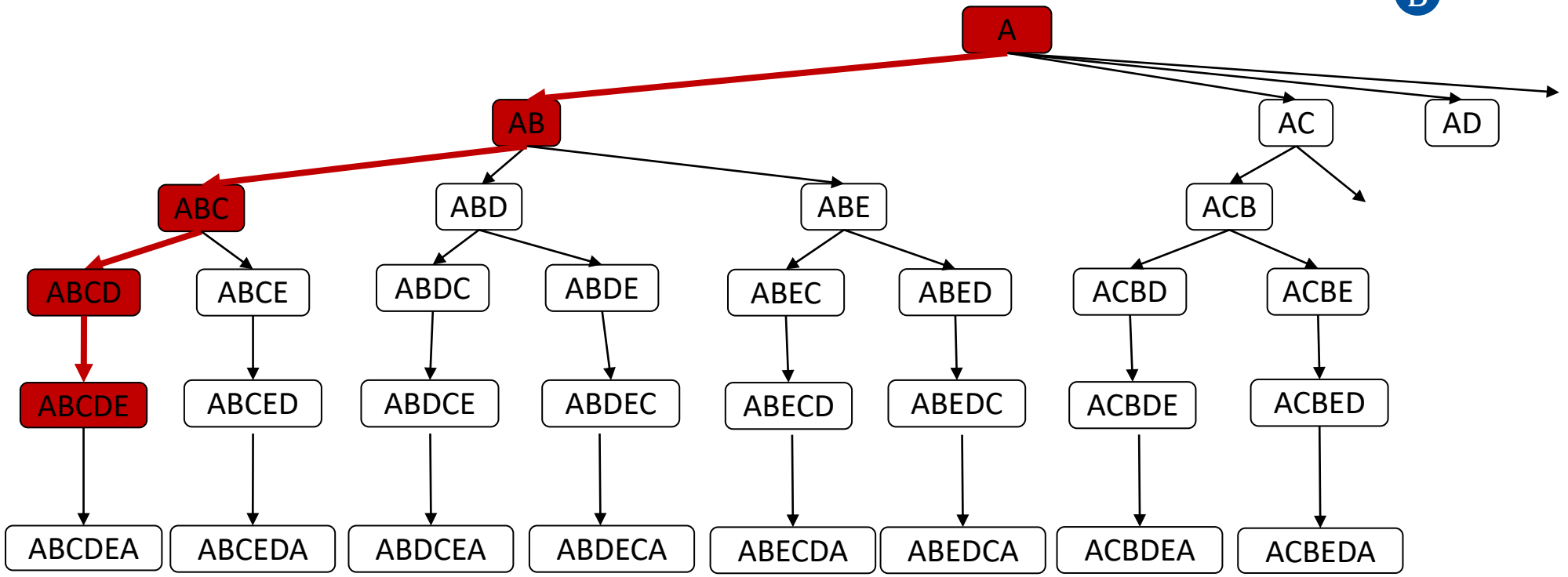




Länge:

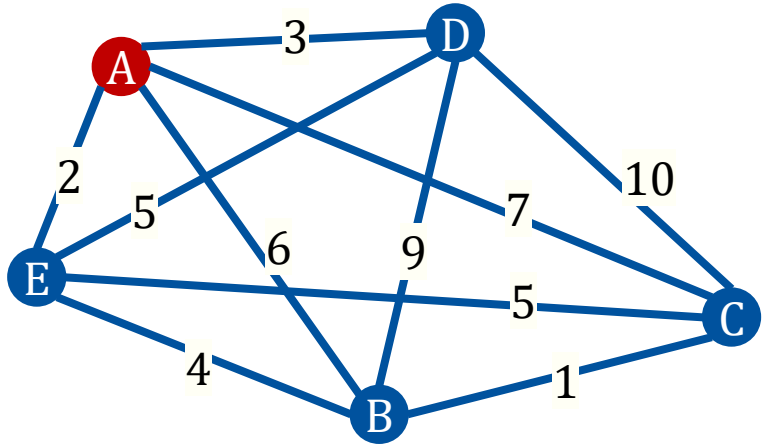
Beispiel: TSP mit n = 5

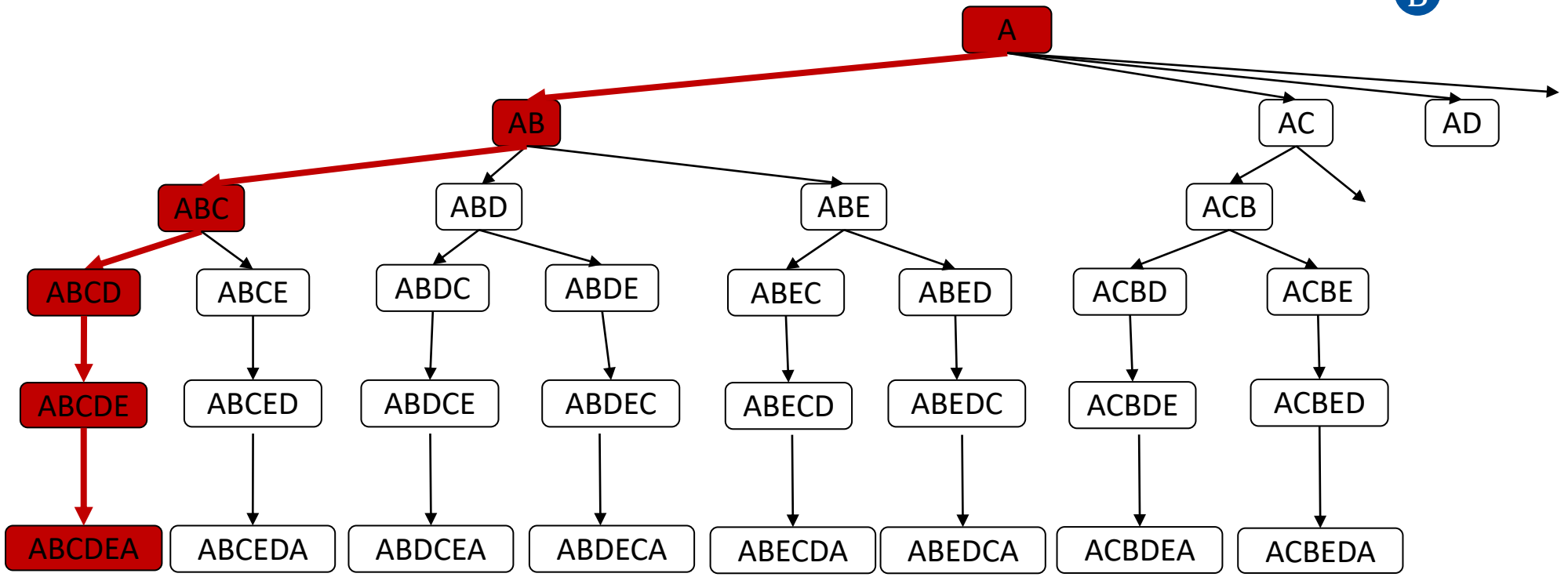




Länge:

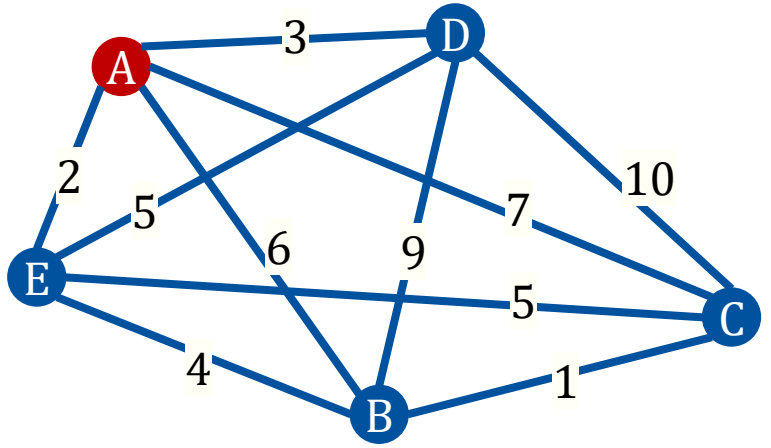
Beispiel: TSP mit n = 5

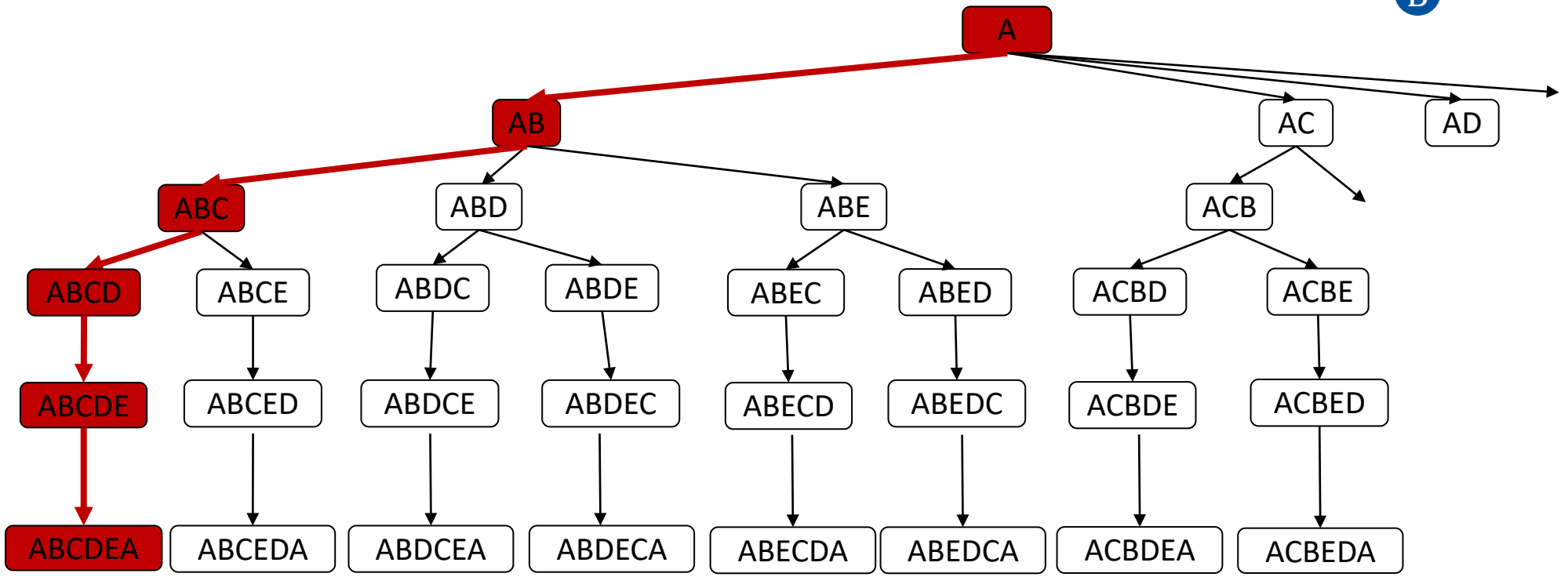




Länge:

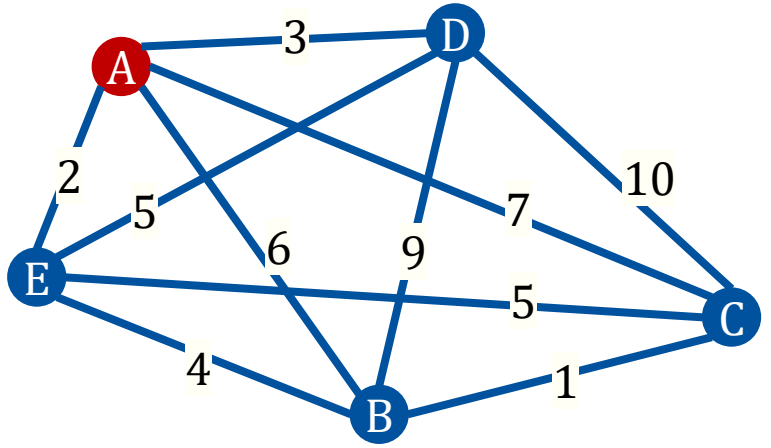
Beispiel: TSP mit n = 5

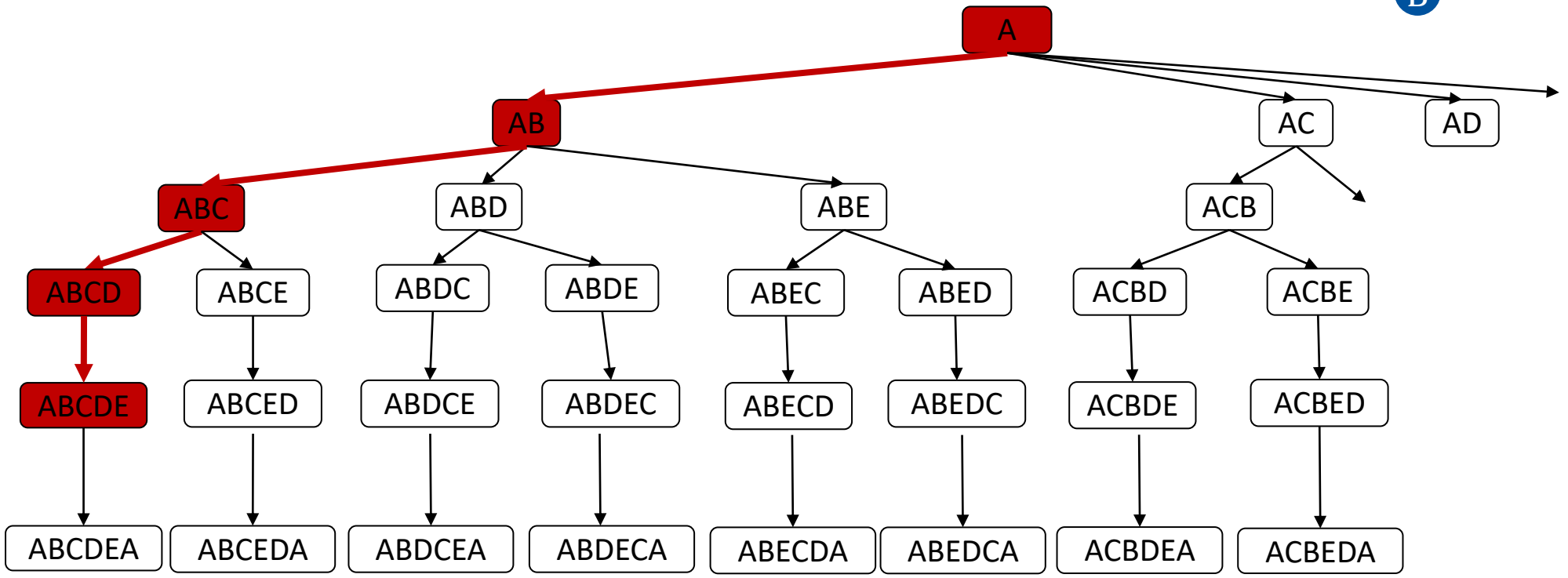




Länge: *24*

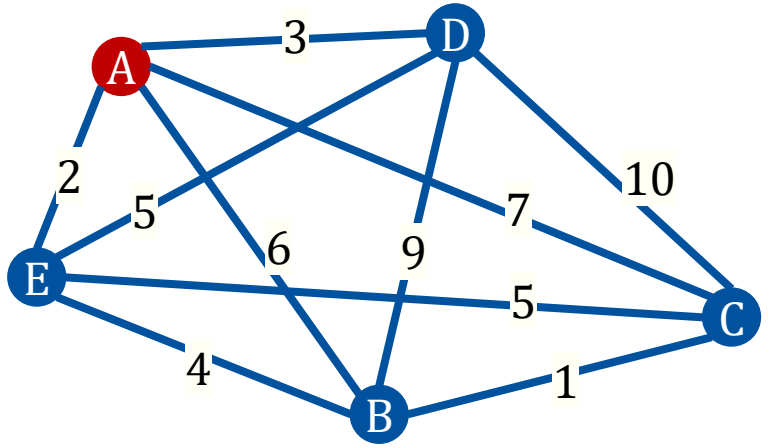
Beispiel: TSP mit n = 5

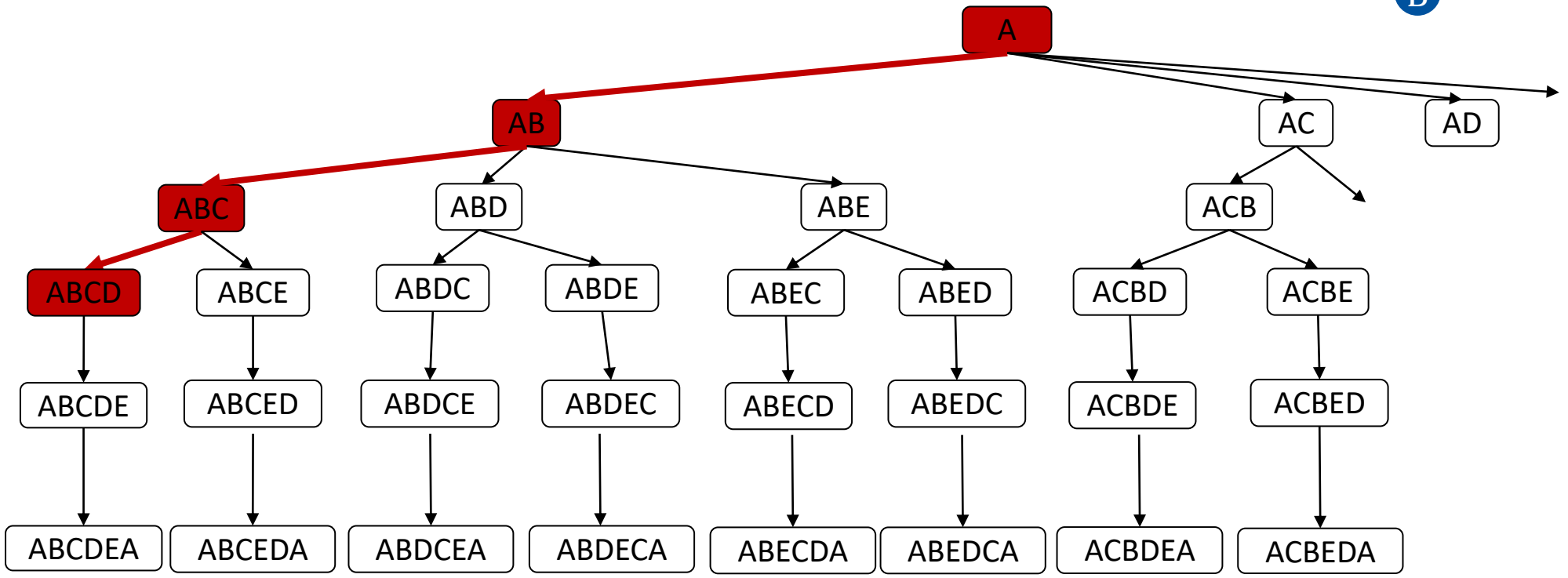




Länge: *24*

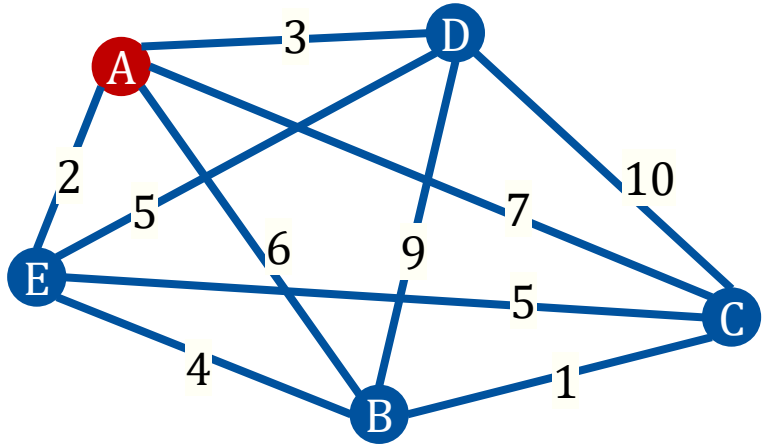
Beispiel: TSP mit n = 5

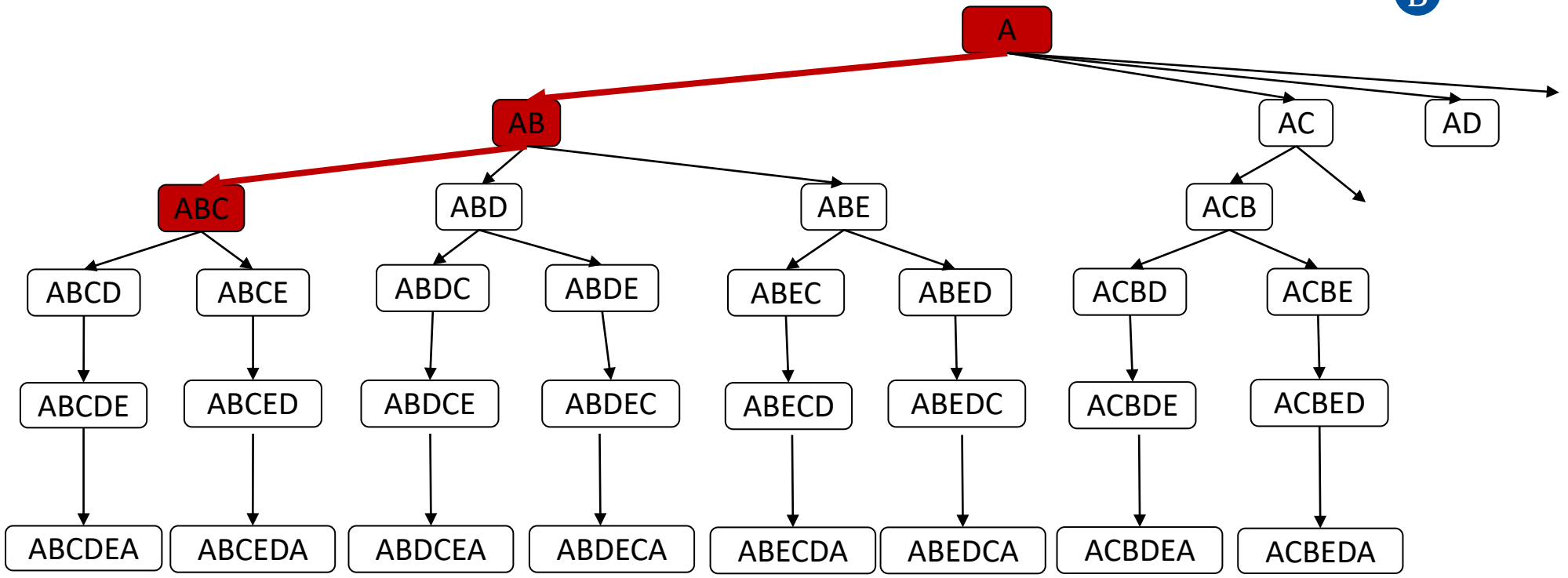




Länge: *24*

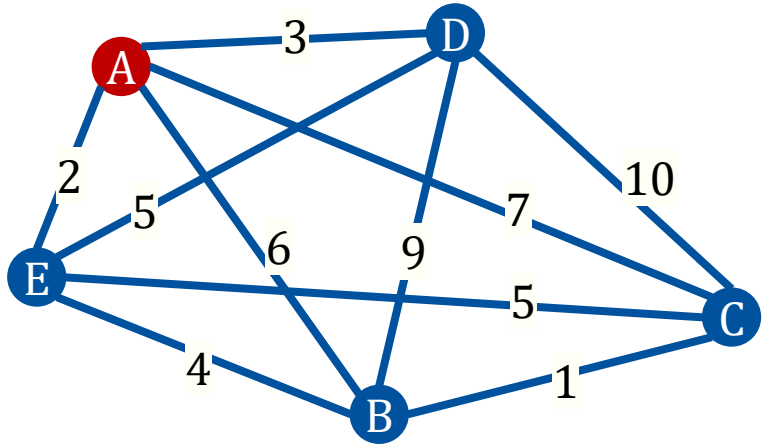
Beispiel: TSP mit n = 5

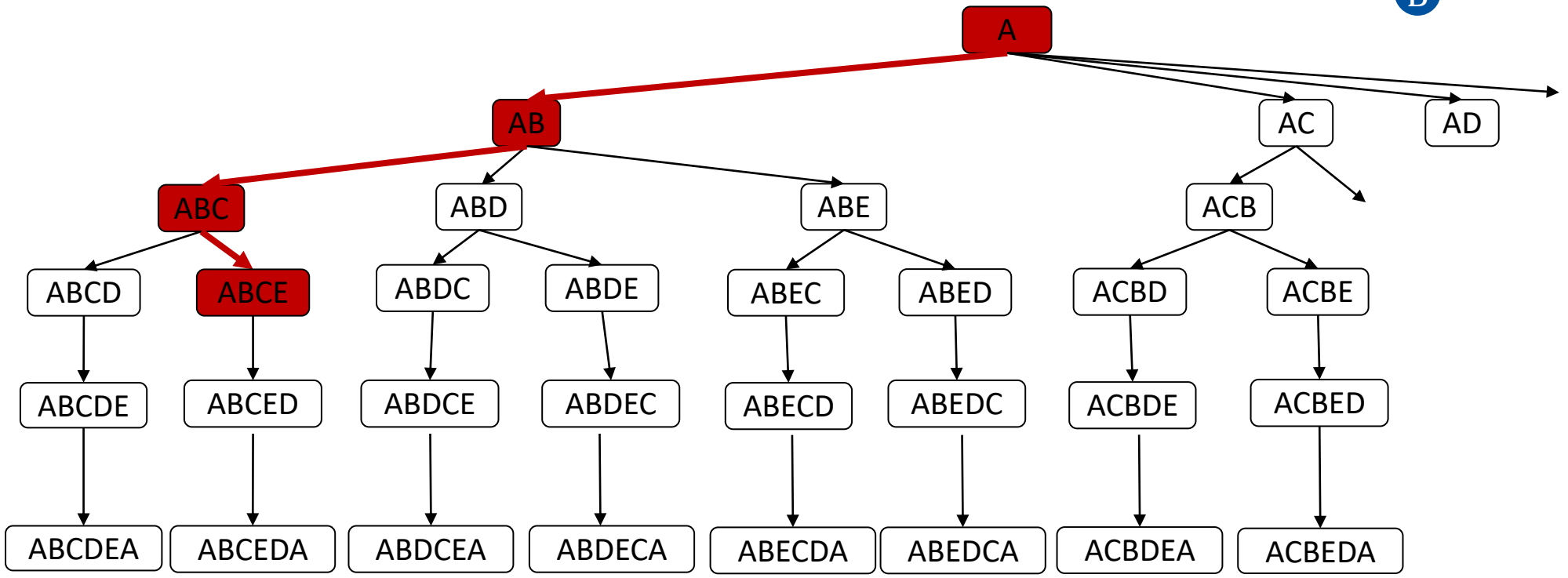




Länge: *24*

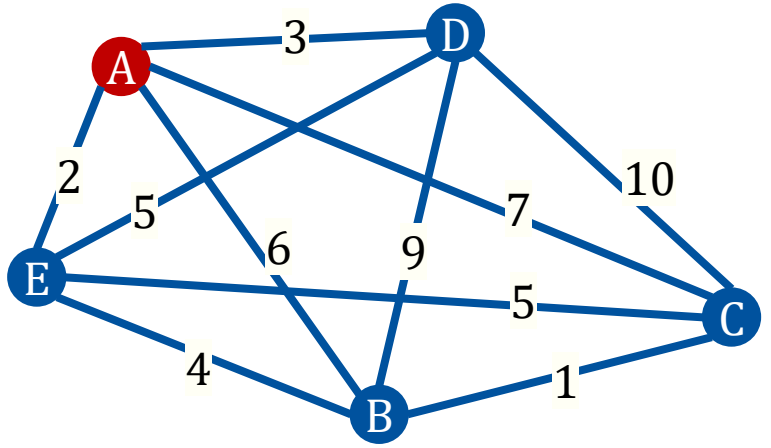
Beispiel: TSP mit n = 5

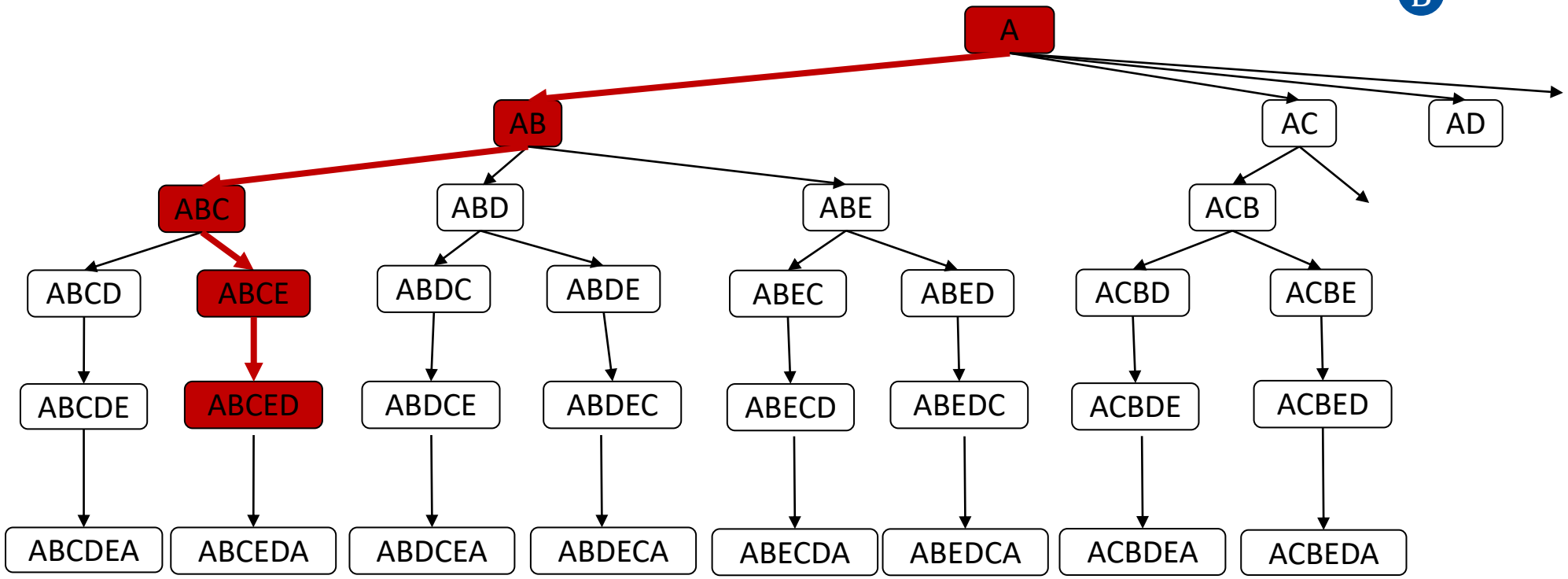




Länge: *24*

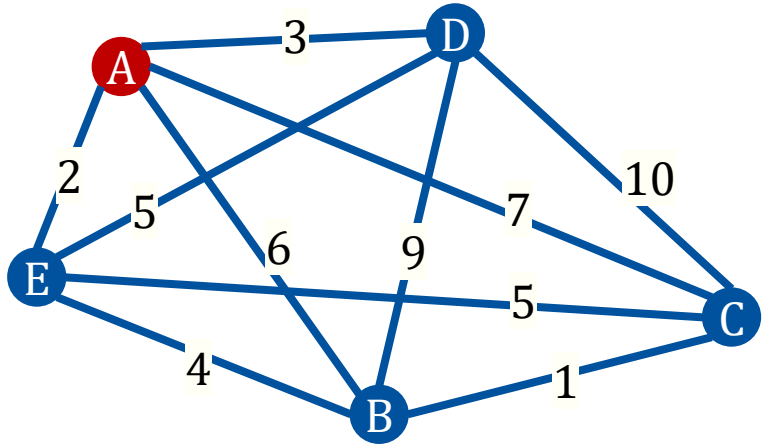
Beispiel: TSP mit n = 5

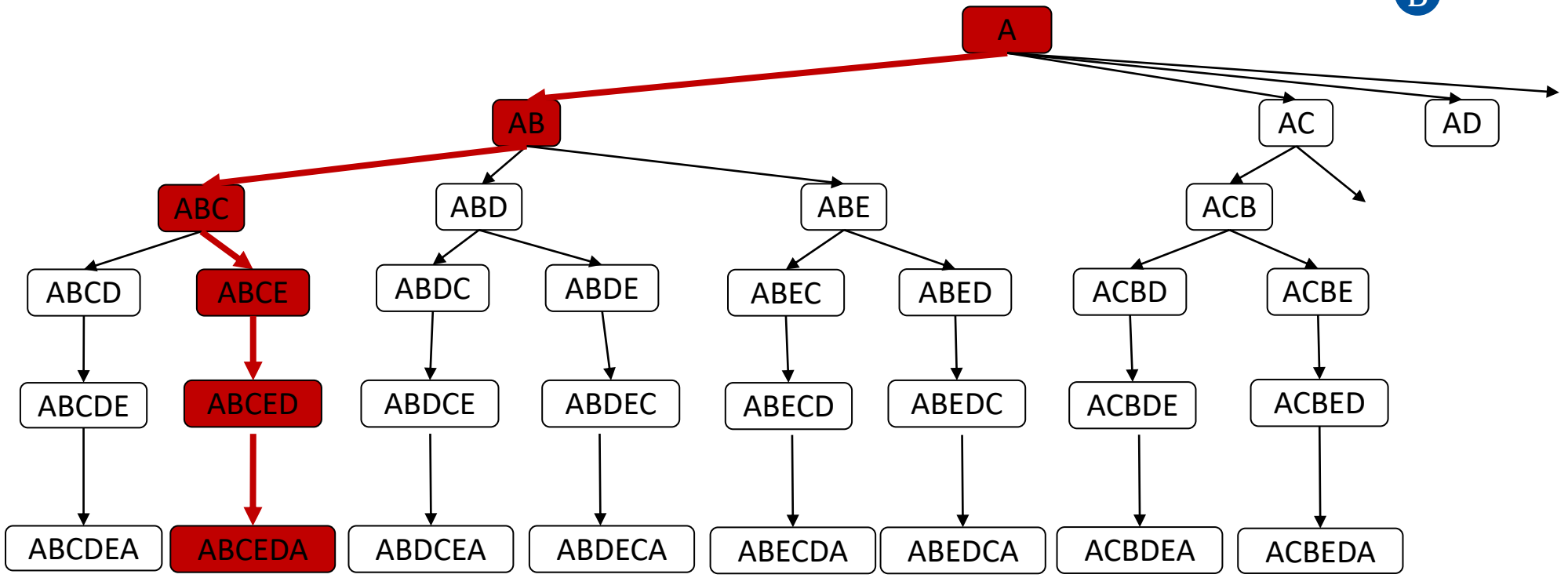




Länge: *24*

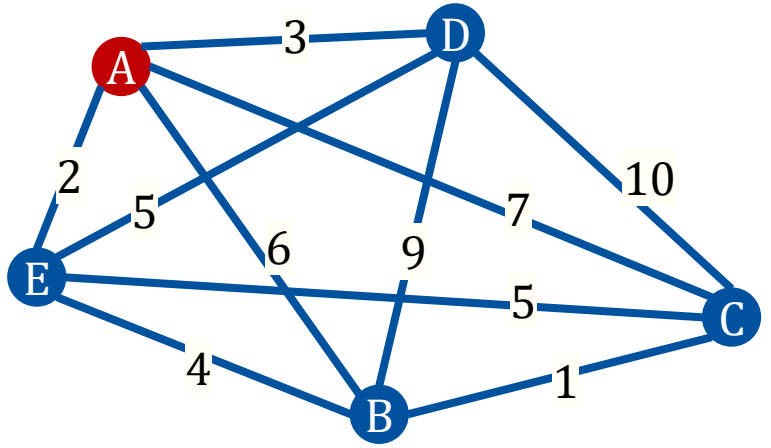
Beispiel: TSP mit n = 5

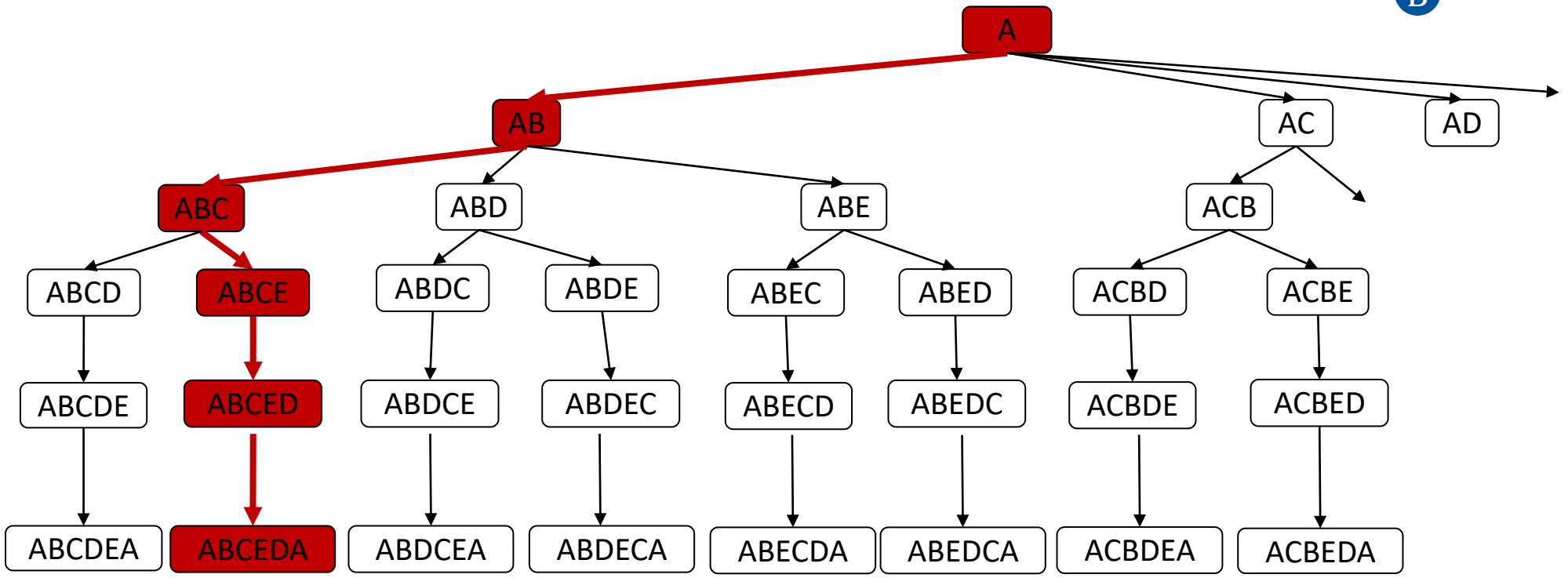




Länge: *24*

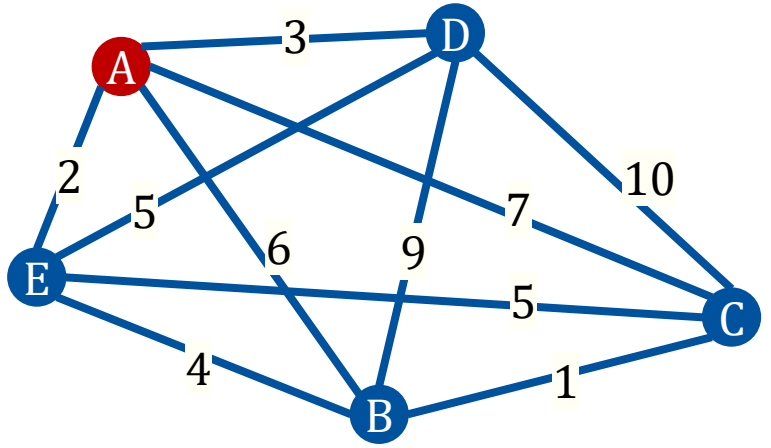
Beispiel: TSP mit n = 5

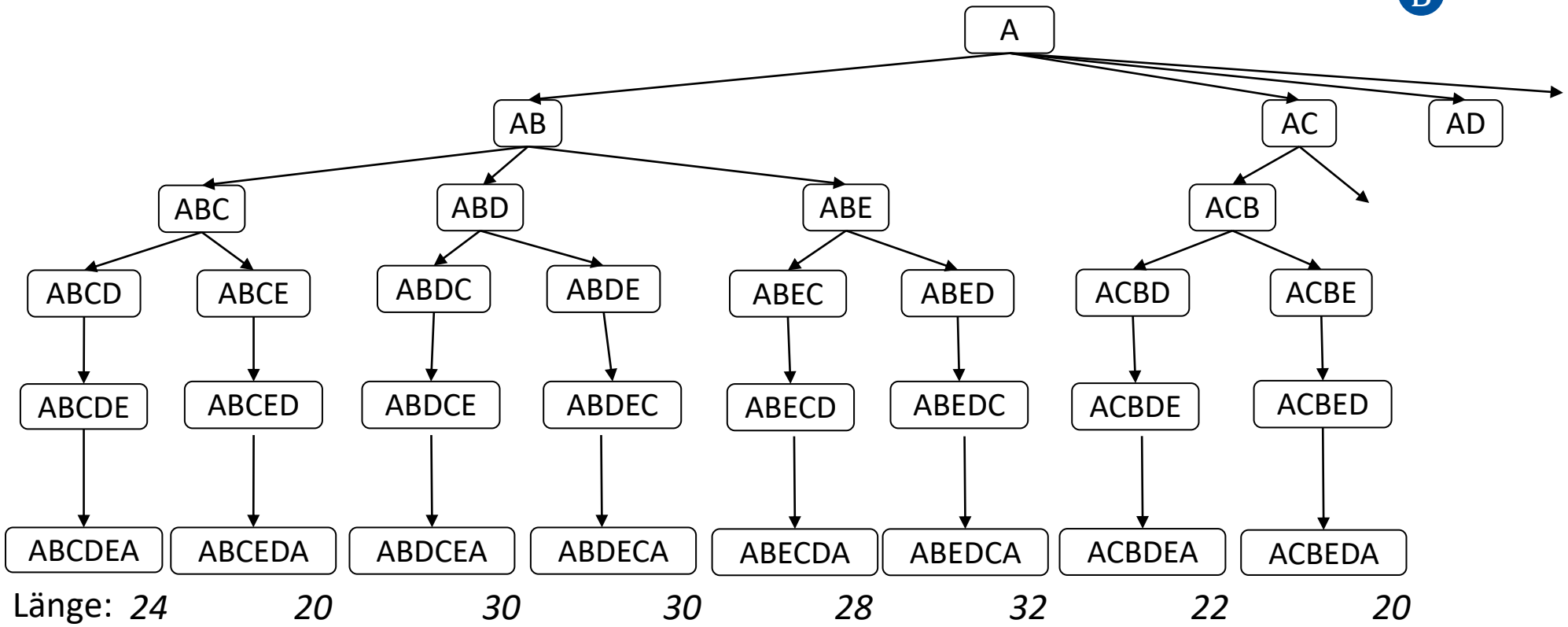




Länge: *24* *20*

Beispiel: TSP mit n = 5

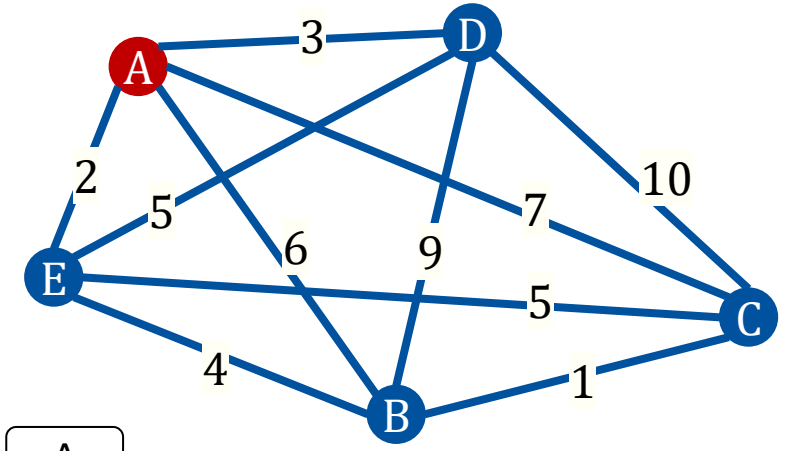


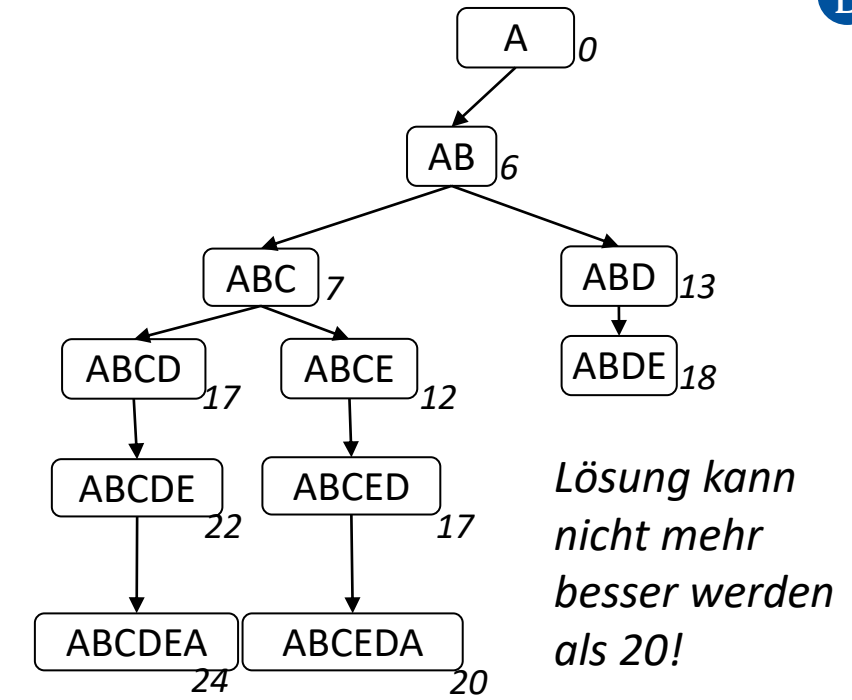


Backtracking = Tiefensuche im Entscheidungsbaum

Branch & Bound: TSP

- Vermindern des Rechenaufwandes durch frühzeitiges Erkennen aussichtsloser Teilbäume
- TSP: Untere Schranke (Bound) für die Länge der noch fehlenden Kanten

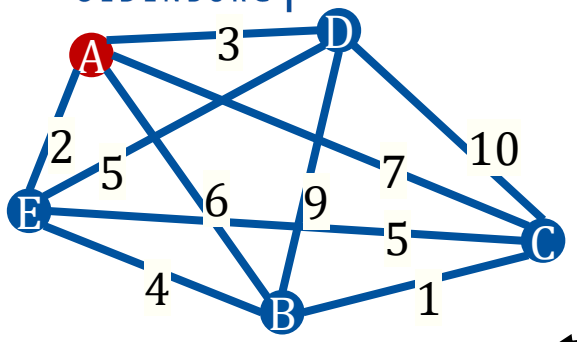


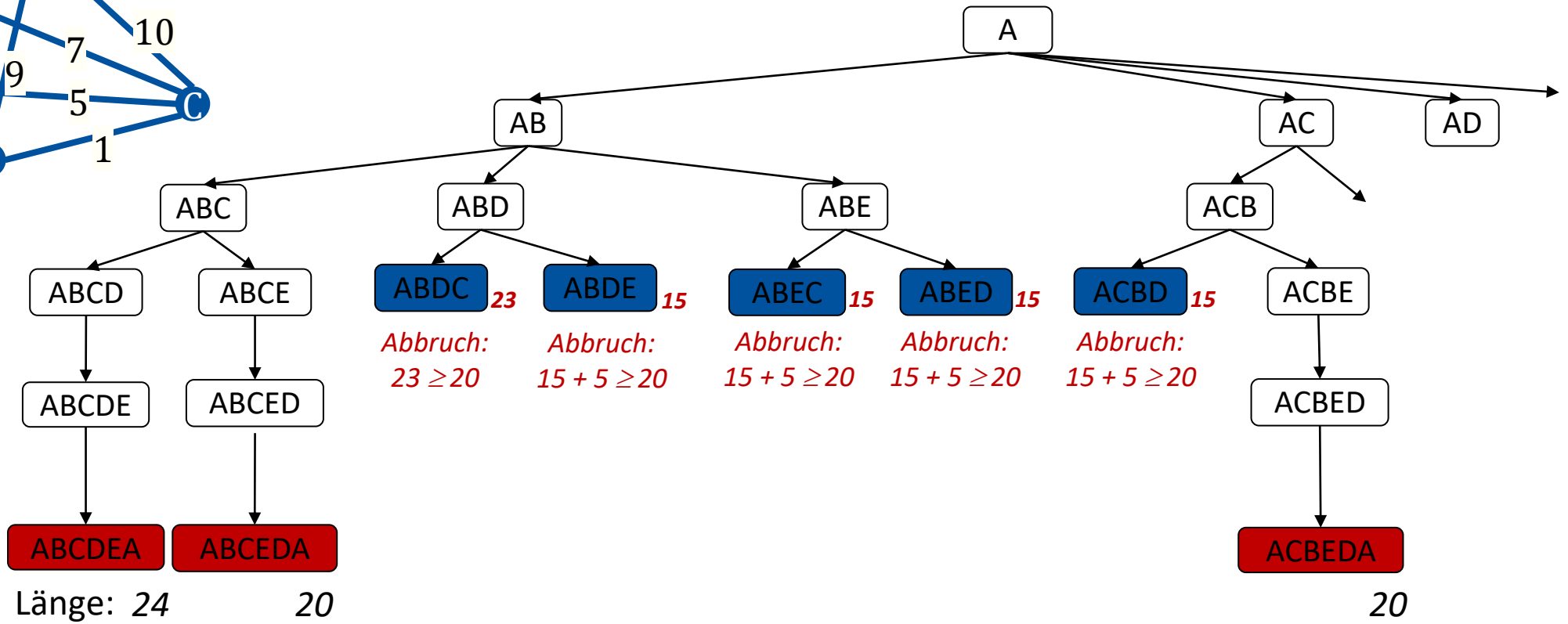


Branch & Bound-Strategie

- Branch-Schritt
  - systematische Suche wie beim Backtracking
- Bound-Schritt
  - bei jeder Teillösung: berechne untere Schranke (Bound) für den noch fehlenden Teil der Lösung
  - Abbruch der Suche in einem Zweig, falls
    Güte der Teillösung + Bound
    schlechter als die bisher gefundene Lösung ist
- Ziele
  - möglichst frühzeitiger Abbruch
  - möglichst einfach zu berechnender Bound

Beispiel: TSP mit Branch & Bound





Bisher beste Lösung: 24

Bisher beste Lösung: 20

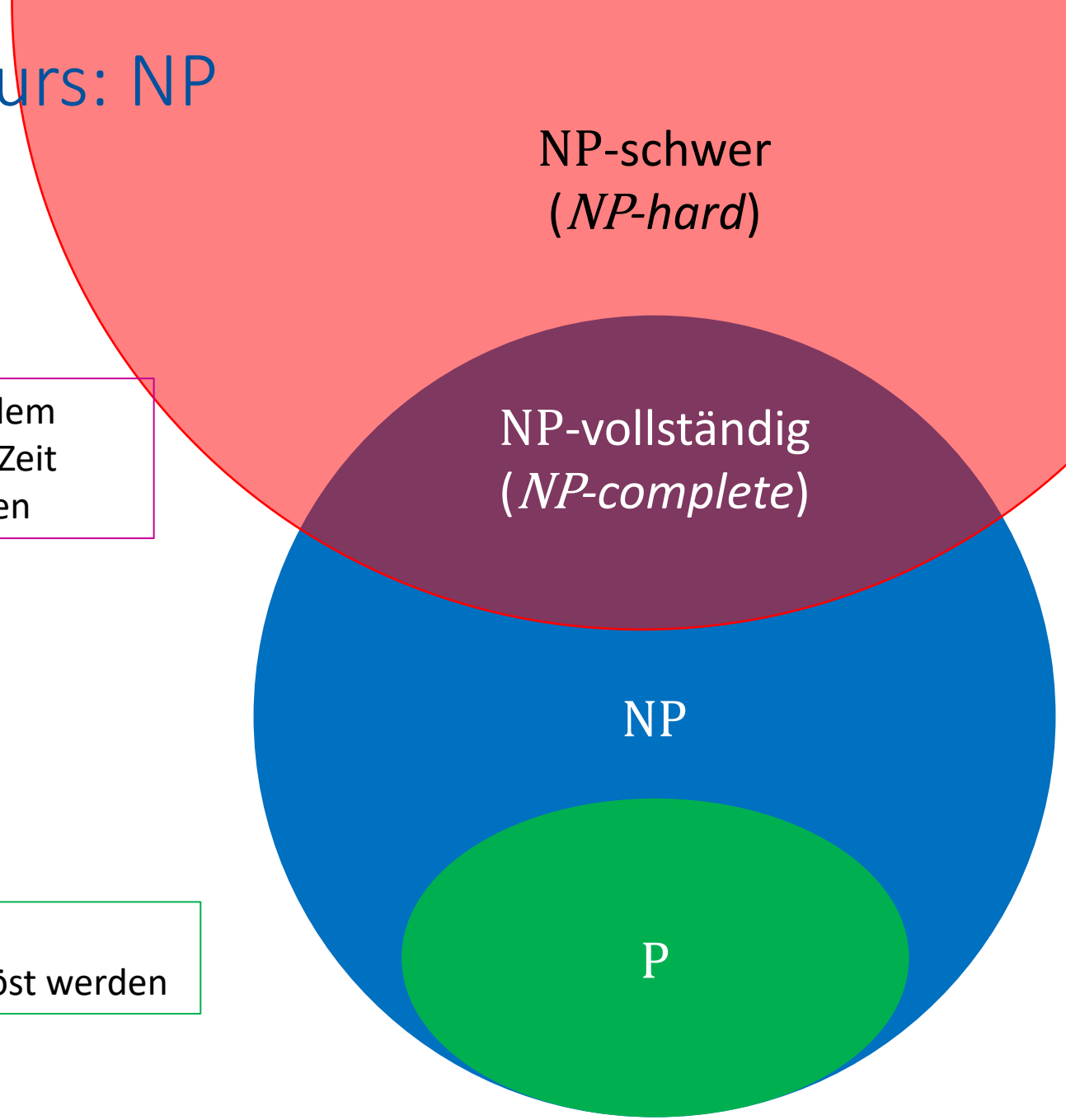
Zwischenfazit

- Strategien
  - Divide & Conquer
    - Rekursion
    - Dynamische Programmierung
  - Aufzählung
    - Backtracking
    - Branch & Bound
- Ergebnis
  - exakte Lösungen, keine Näherungen → Optimum
  - Problem: zu hoher Aufwand bei sehr komplexen Problemen

Grundlegende Methoden

- Optimierungsprobleme
  - sind oft NP-schwer,
    d.h. bisher bekannte Algorithmen $\in O(2^n)$
  - ➢ sind nicht in akzeptabler Zeit exakt lösbar
- Möglichkeiten
  - Relaxation – Lockerung von Nebenbedingungen
  - Approximation – Näherungsweise Lösung
  - Heuristiken – Intelligente Suchstrategien im Lösungsraum

Exkurs: NP



Ist mindestens so „schwer“ wie ein NP-Problem

Jedes andere NP-Problem kann in polynomieller Zeit darauf reduziert werden

Lösungen können in polynomieller Zeit überprüft werden

Probleme können in polynomieller Zeit gelöst werden

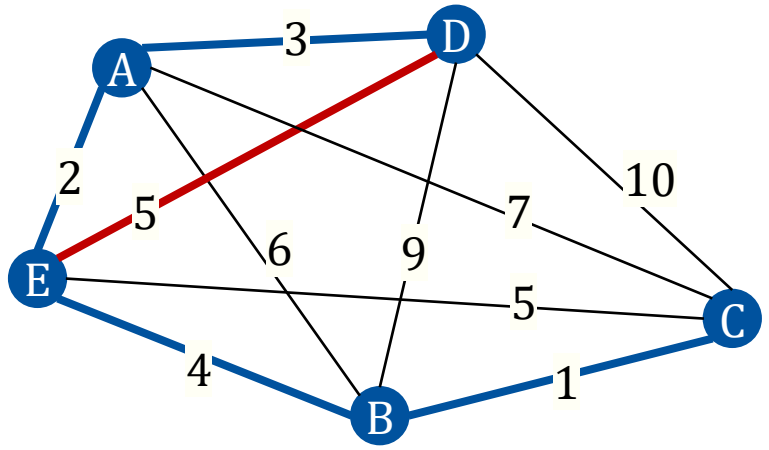
Annahme: P≠NP

Relaxation (Lockerung)

- Abschwächung der Nebenbedingungen
- Lösung des hierdurch vereinfachten Problems
- Übertragung der vereinfachten Lösung auf das komplexe Problem
- Lagrange-Relaxation
  - gewichte die Nebenbedingungen durch „Strafwerte“
  - erweitere die Gütefunktion um die Berücksichtigung dieser Strafwerte

Beispiel: Relaxation TSP

- Bestimmung einer unteren Schranke für die Länge der kürzesten Rundreise
- Verzicht auf die Nebenbedingung eines geschlossenen Weges
- Lösung
  - Konstruktion des MST
  - Erweiterung um die Kante kürzester Länge
  - Ergebnis ist keine zulässige Rundtour
  - ➢ Mindestlänge für eine Rundtour (z.B. für Branch & Bound)



Kante des MST

Hinzugefügte Kante kürzester Länge

Approximation (Näherung)

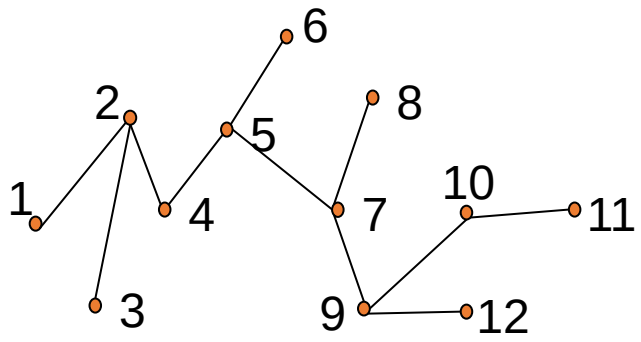
- Approximative Verfahren
  - keine Garantie für exakte/optimale Lösung
  - aber Fehlerschranken möglich

Beispiel: Approximation TSP

- Phase 1: Bestimme MST des Graphen
  (Aufwand ~$O(|E| \log |V|)$ mit Algorithmus von Prim)
- Phase 2: Bestimme Weg auf dem MST, der ggf. Knoten mehrfach besucht, z.B. durch Preorderdurchlauf
- Phase 3: Streiche mehrfach besuchte Knoten aus der Liste
  - funktioniert nur bei vollständigen Graphen, da sonst nach Streichen eines Knotens ggf. keine Rundtour mehr gebildet werden kann

Beispiel: Approximation TSP (2)

Phase 1: MST

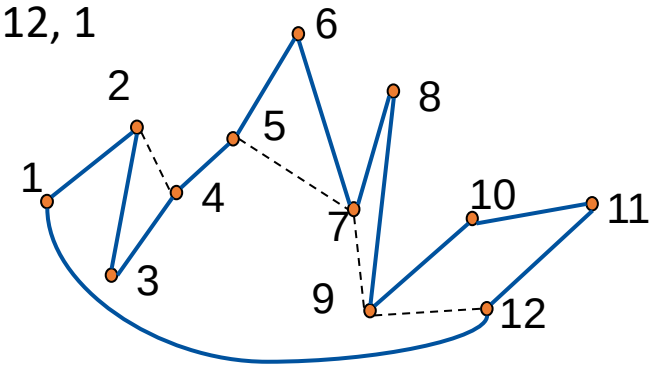


Phase 2: Preorder Weg

1, 2, 3, 2, 4, 5, 6, 5, 7, 8, 7, 9, 10, 11, 10, 9, 12, 9, 7, 5, 4, 2, 1

Phase 3: Streichen von mehrfach besuchten Knoten

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1



CARL VON OSSIETZKY universität OLDENBURG

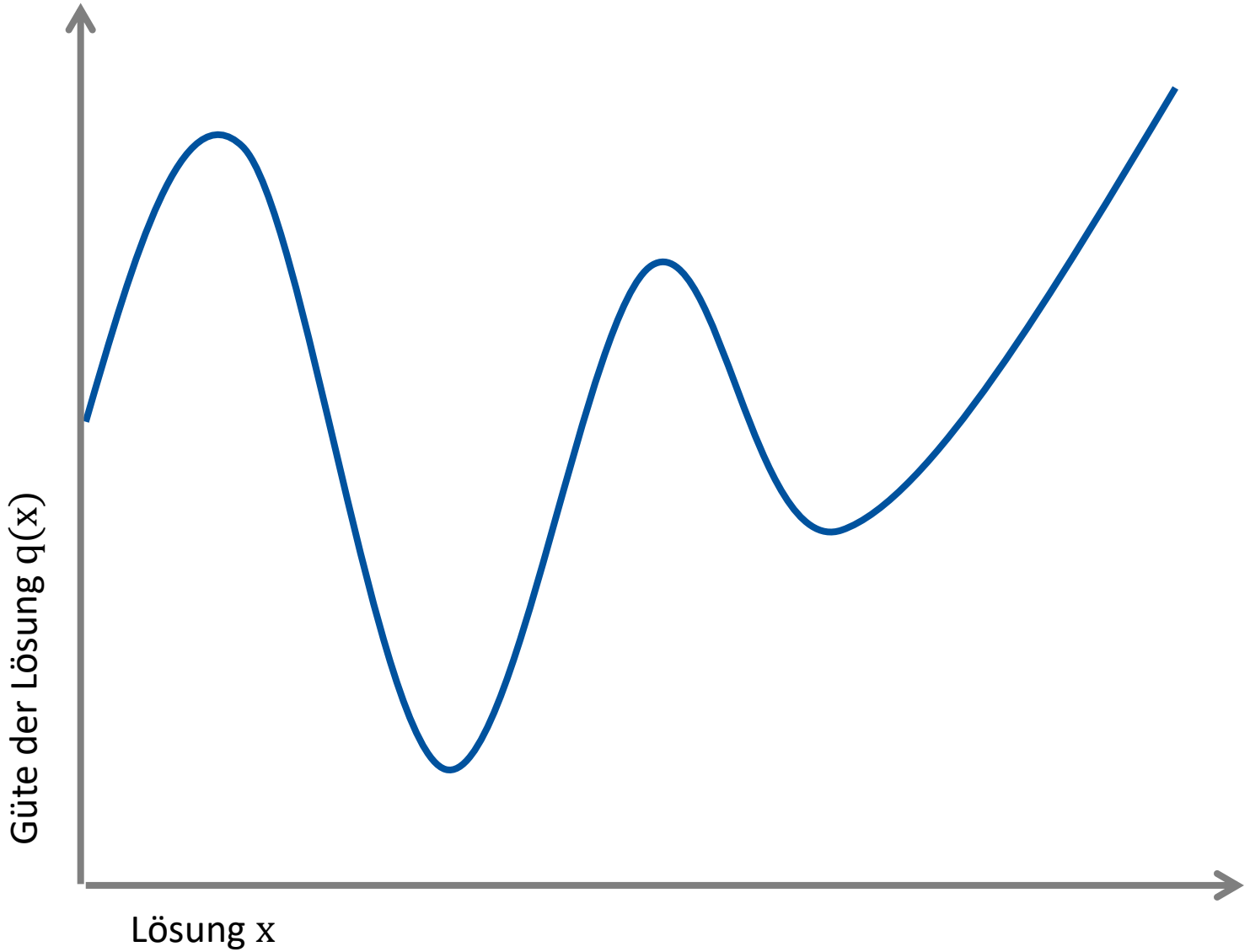
Heuristiken

CARL VON OSSIETZKY universität OLDENBURG

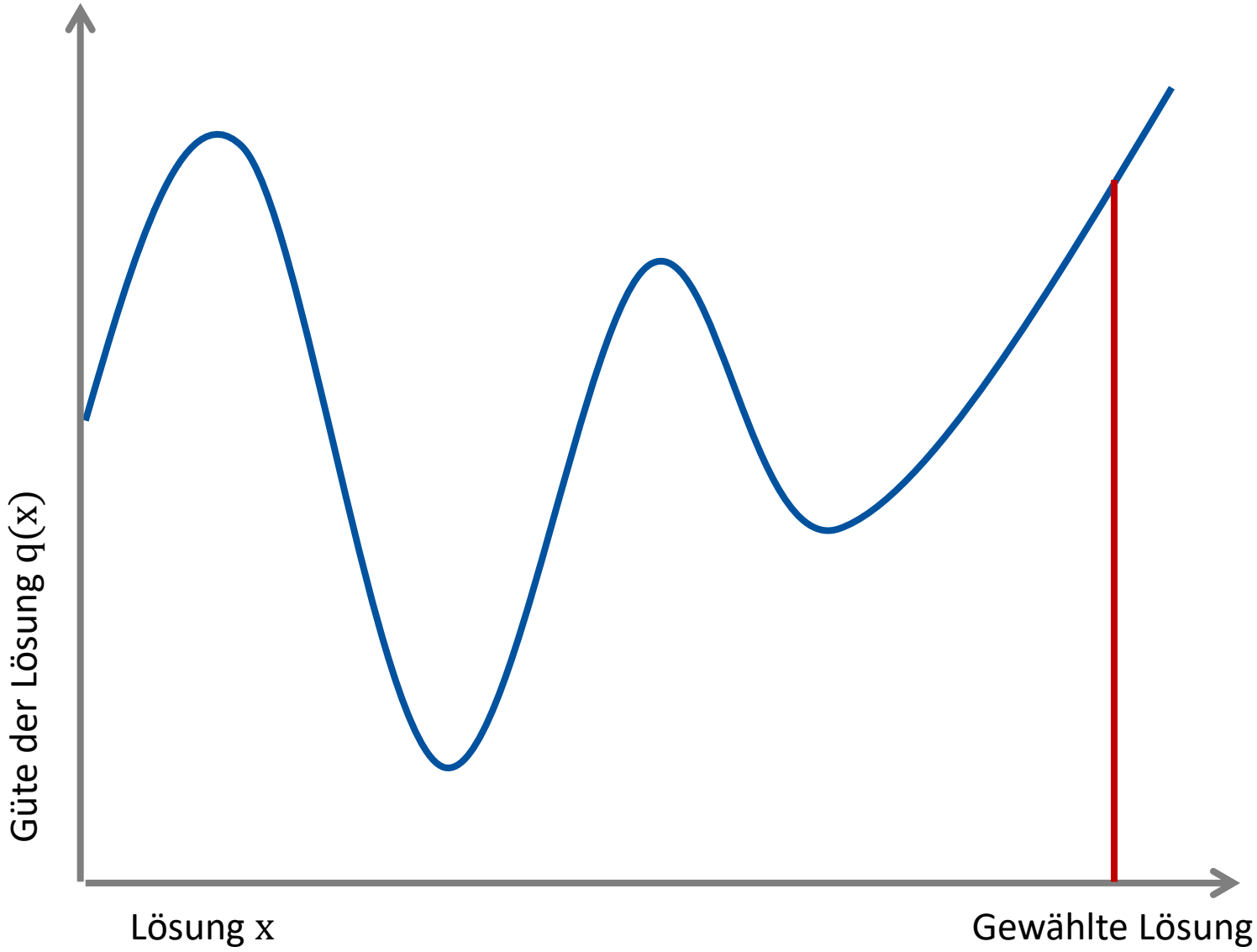
Heuristiken

- Intelligente Suchstrategie im Lösungsraum
- Verwendet nur lokales Wissen über den Suchraum
- Keine Garantie für optimale Lösung, aber in akzeptabler Zeit eine relativ gute Lösung
- Kompromiss zwischen Rechenzeit und Güte der Lösung
- Meta-Heuristik
  - abstrakte Heuristik, die für unterschiedliche Anwendungsbereiche konkretisiert werden kann

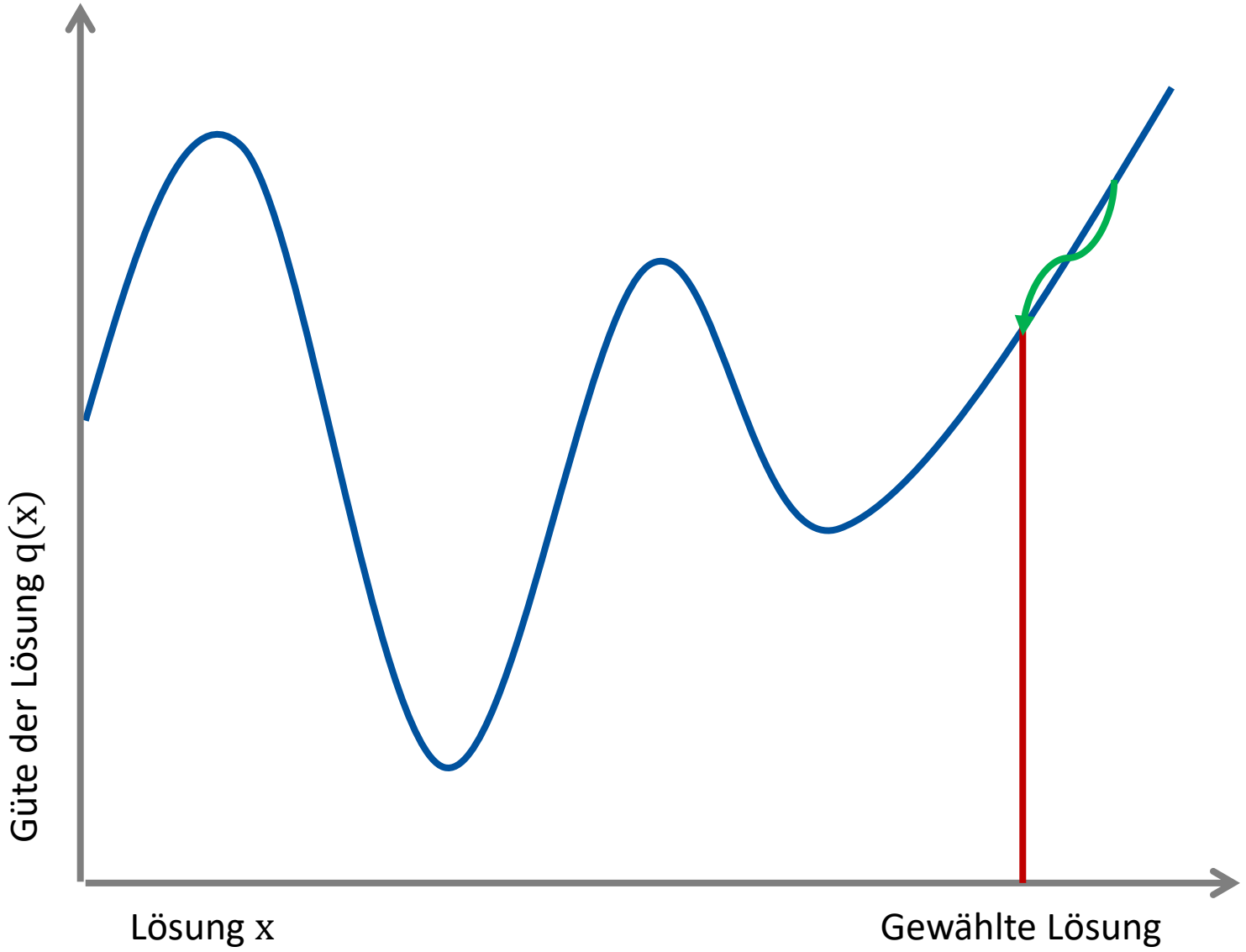
Heuristiken: Lösungsraum



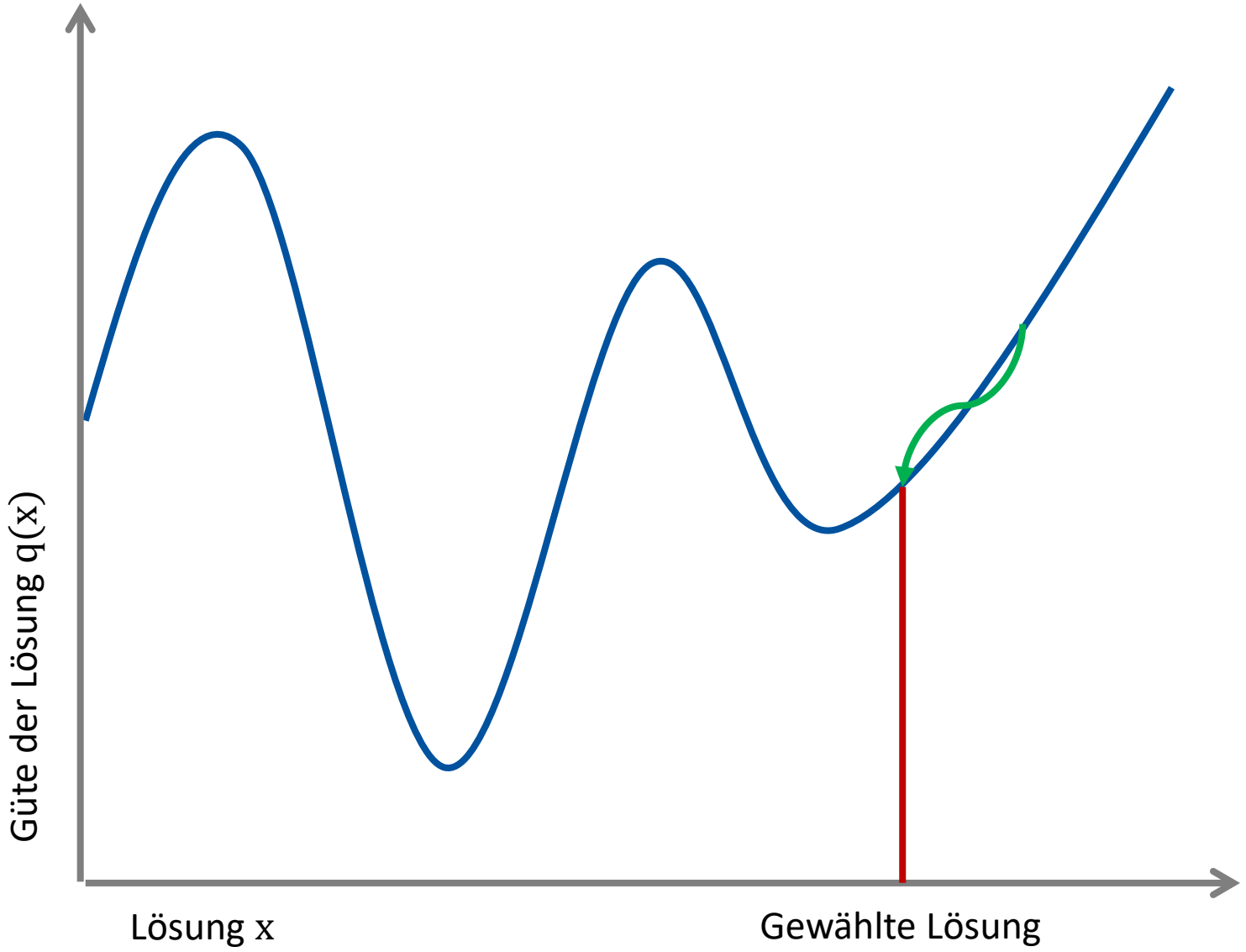
Heuristiken: Lösungsraum



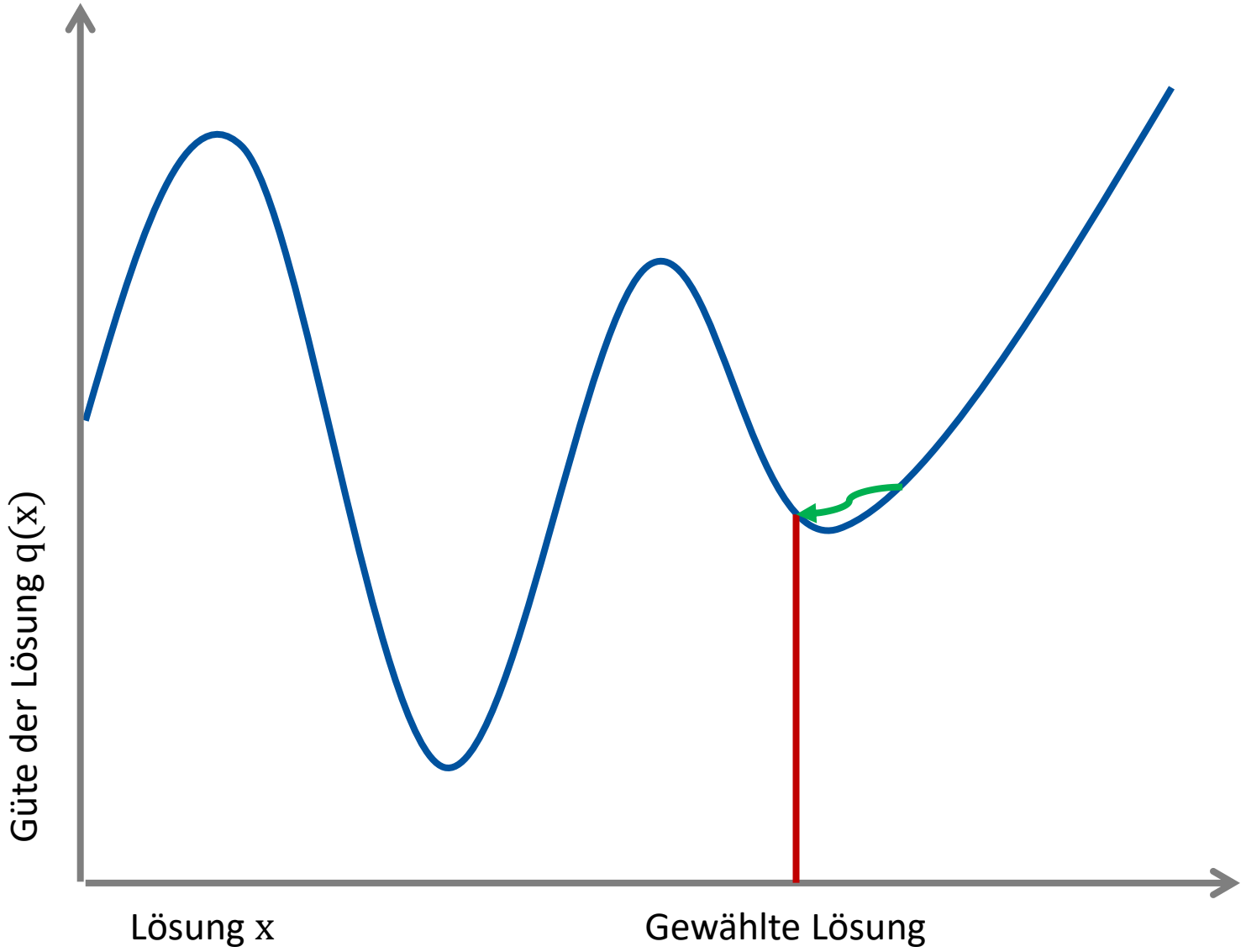
Heuristiken: Lösungsraum



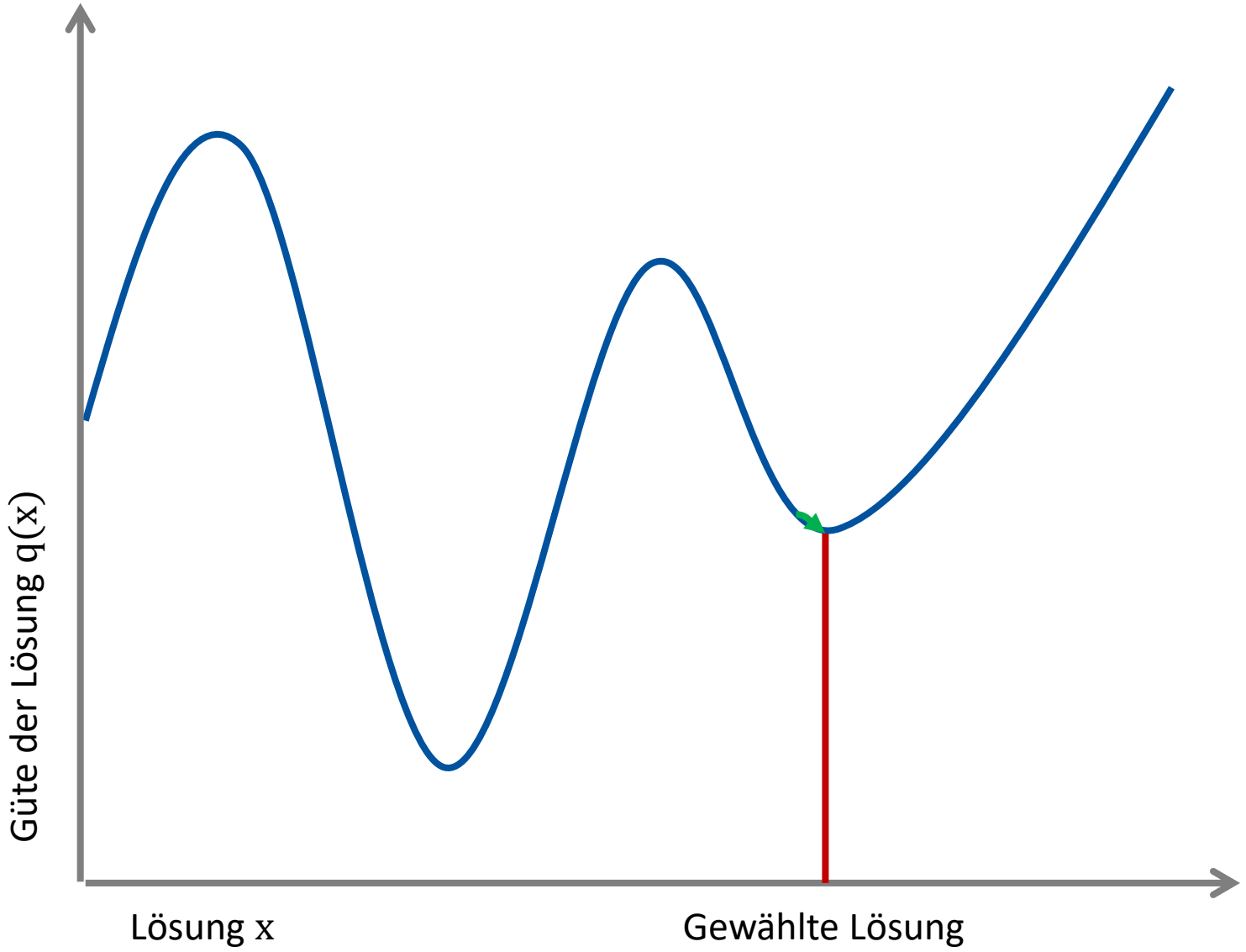
Heuristiken: Lösungsraum



Heuristiken: Lösungsraum



Heuristiken: Lösungsraum



Meta-Heuristiken

- Lokale Suche
- Tabu-Suche
- Genetische Algorithmen
- Durchgehendes Beispiel: TSP
  - NP-vollständig
  - auf viele Optimierungsprobleme übertragbar
  - TSP mit Dreiecksungleichung: Weltrekord für exakte Lösung liegt bei nur ca. 10.000 Knoten

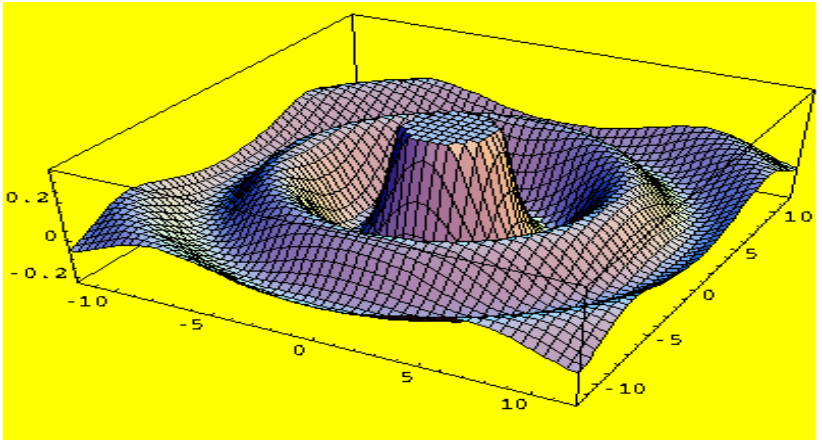
CARL VON OSSIETZKY universität OLDENBURG

Lokale Suche

Lokale Suche

- Lokal: nächste Zwischenlösung wird nur lokal in der Nachbarschaft der aktuellen Lösung gesucht

```
x = startLoesung;
while (!abbruch) {
    waehle y aus Nachbarschaft N(x)
    if q(y) besser als q(x)
        x = y
}
```



Voraussetzungen für Lokale Suche

- Initiale Startlösung?
  - Berechnung einer zulässigen suboptimalen Lösung, z.B. durch einfaches Greedy-Verfahren
- Nachbarschaft $N(x)$ einer Lösung x?
  - Menge der zulässigen Lösungen, die sich „wenig“ von x unterscheiden
- Abbruchbedingung?
  - kein Nachbar mit besserer Güte vorhanden?
  - Lösungsgüte erreicht?
  - maximale Anzahl an Iterationen oder Rechenzeit

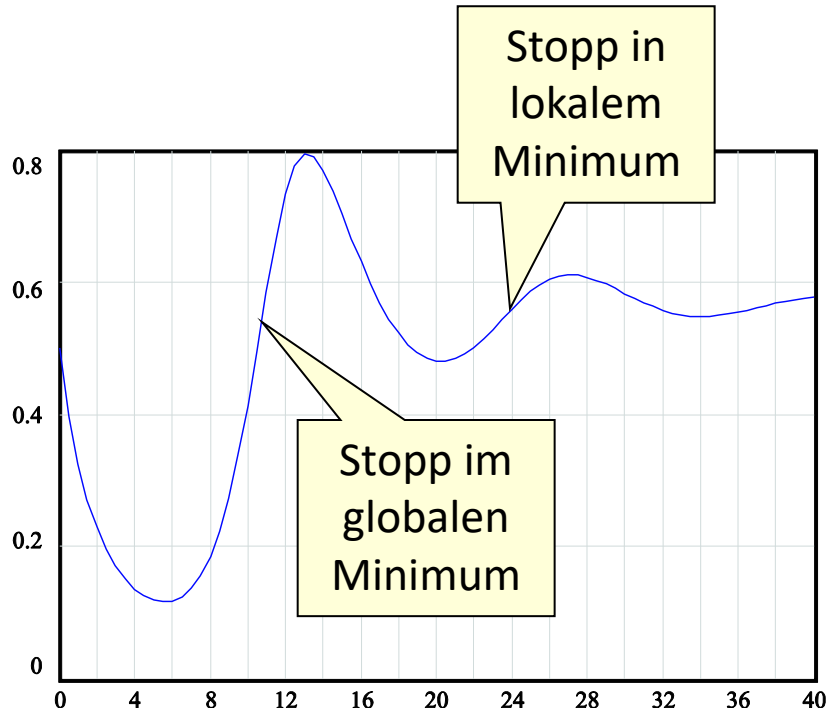
➢ alle problemabhängig zu definieren

Auswahl der neuen Lösung

- Aktuelle Lösung x
- Auswahl der Nachbarlösung y aus $N(x)$
  - Wahl des Nachbarn, der die größte Verbesserung der Gütefunktion mit sich bringt
  - Wahl des ersten Nachbarn, der eine Verbesserung der Güte zur alten Lösung liefert
  - zufällige Auswahl eines Nachbarn, falls mehrere zur selben (besten) Verbesserung führen

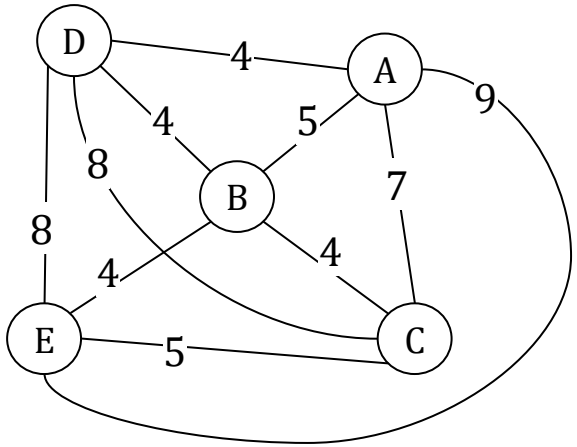
Konvergenz

- Gütefunktion im Lösungsraum ist nicht (streng) monoton
- Globales Optimum wird nicht für jede Startlösung gefunden



Beispiel: Lokale Suche TSP

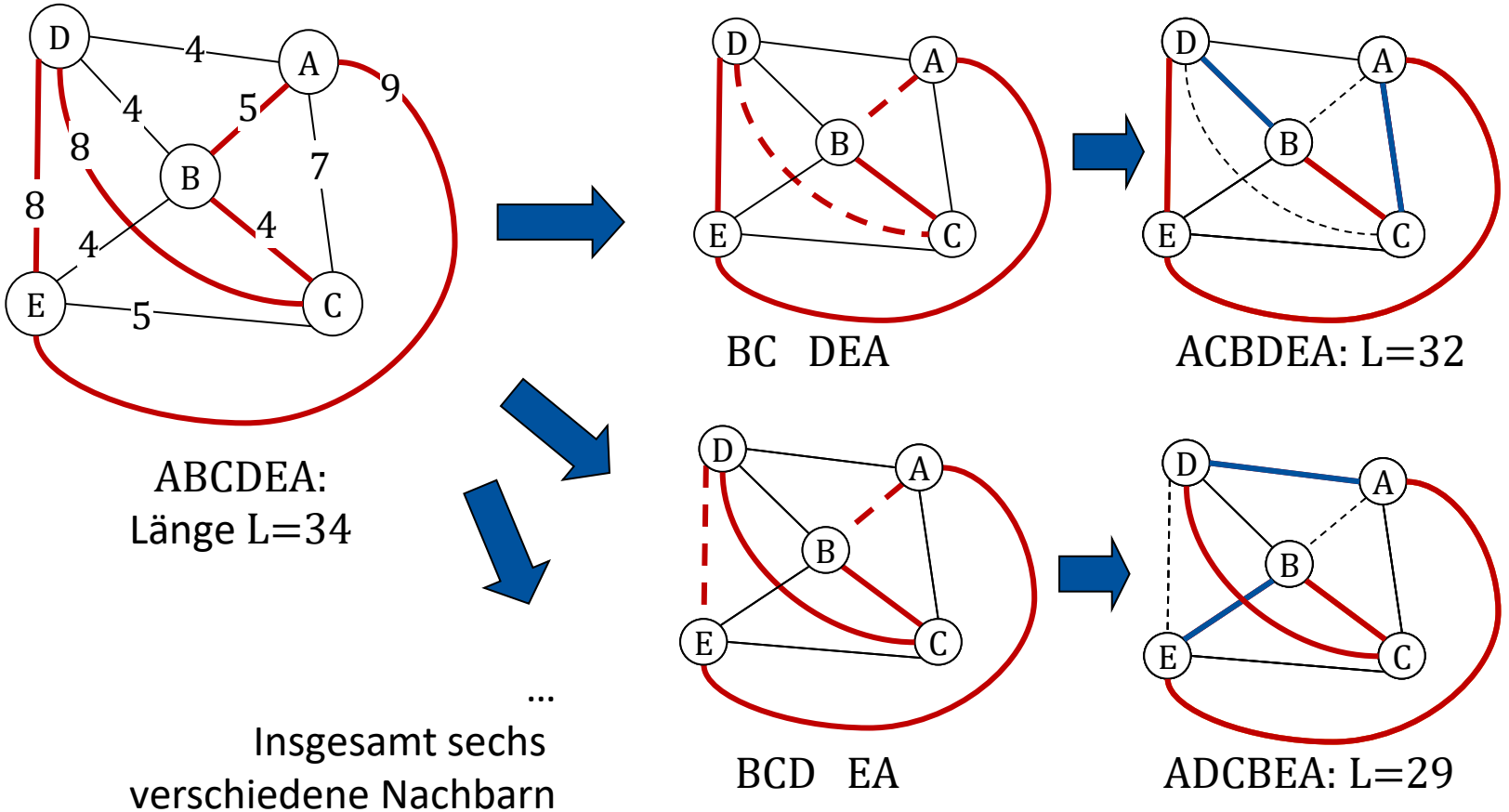
- Aufgabe: Finde die kürzeste Rundreise mittels lokaler Suche
- Zu klären
  - initiale Lösung
    - z.B. (A, B, C, D, E)
      → Länge 34
  - Nachbarschaft einer Lösung
    - z.B. Konstruktion der Nachbarn von $x = (v_1, v_2, \ldots, v_n)$
      durch Austausch zweier disjunkter Kanten



Konstruktion der Nachbarschaft

Rundtour zerfällt durch gewählte, zu löschende Kanten

Einfügen neuer Kanten → neue Rundtour



...

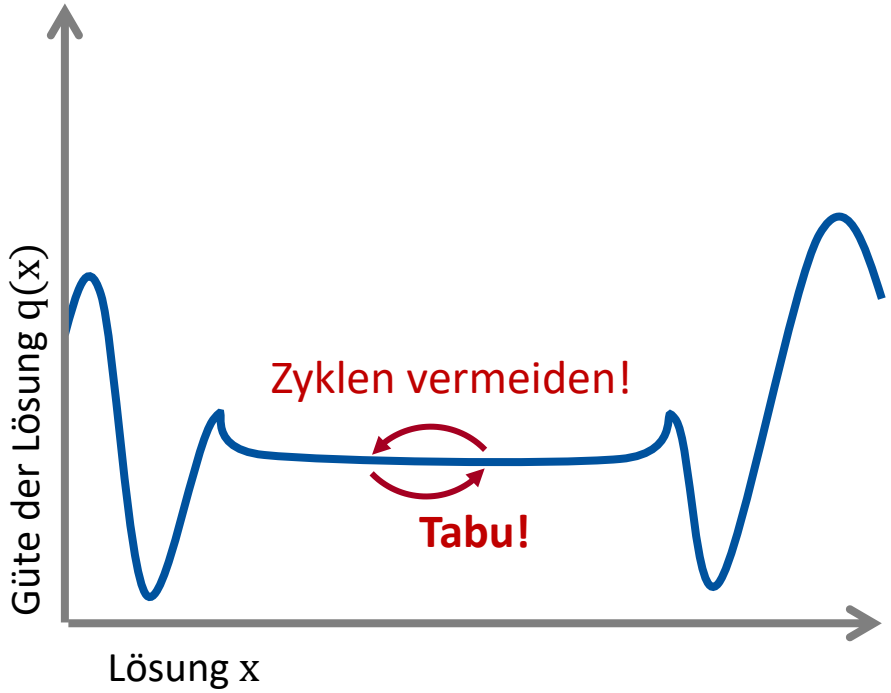
Insgesamt sechs verschiedene Nachbarn

CARL VON OSSIETZKY universität OLDENBURG

Tabu-Suche

Tabu-Suche

- Lokales Suchverfahren, das temporäre Verschlechterungen der Zwischenlösungen erlaubt
- Tabuliste
  - zur Vermeidung von Zyklen
  - hat feste Länge M
  - bereits besuchte Lösungen sind für M Schritte tabu

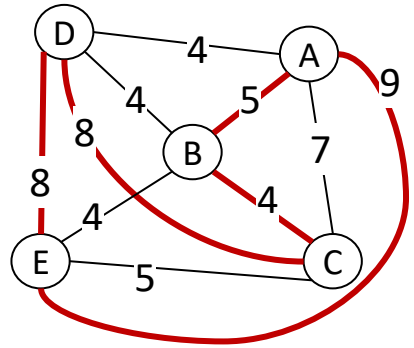


Tabu-Suche (2)

```
x = startLoesung;
best = x;                    // bisher beste Lösung
(x, wartezeit) → Tabu-Liste
while (!abbruch) {
     bestimme N = N(x);    // Nachbarschaft der Lösung x
     entferne Tabu-Liste aus N;
     waehle y mit der besten Guete q(y) aus N
     reduziere die Wartezeit aller Eintraege der Tabu-Liste um 1
     (y, wartezeit) → Tabu-Liste
     if q(y) besser als q(best) {
           best = y;       // neue beste Lösung
     }
     x = y;
}
```

Beispiel: Tabu-Suche TSP

Tabulistenlänge = Wartezeit = w = 3
x = ABCDEA
best = ABCDEA
q(ABCDEA) = 34

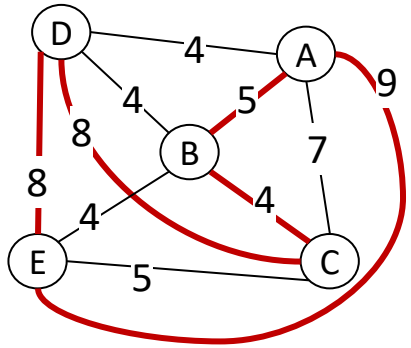


Tabu-Liste:

ABCDEA, 3

Beispiel: Tabu-Suche TSP

Tabulistenlänge = Wartezeit = w = 3
x = ABCDEA
best = ABCDEA
q(ABCDEA) = 34
N(x) = { ACBDEA, ADCBEA, AECDBA, ABDCEA, ABEDCA, ABCEDA }
q(ACBDEA) = 32
q(ADCBEA) = 29
q(AECDBA) = 31
q(ABDCEA) = 31
q(ABEDCA) = 32
q(ABCEDA) = 26

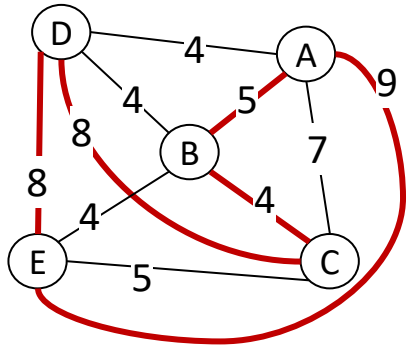


Tabu-Liste:

ABCDEA, 3

Beispiel: Tabu-Suche TSP

Tabulistenlänge = Wartezeit = w = 3
x = ABCDEA
best = ABCDEA
q(ABCDEA) = 34
N(x) = { ACBDEA, ADCBEA, AECDBA, ABDCEA, ABEDCA, ABCEDA }
q(ACBDEA) = 32
q(ADCBEA) = 29
q(AECDBA) = 31
q(ABDCEA) = 31
q(ABEDCA) = 32
q(ABCEDA) = 26

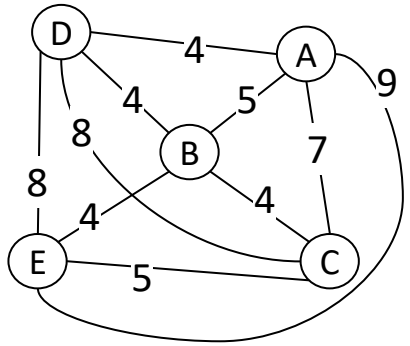


Tabu-Liste:

ABCDEA, 3

Beispiel: Tabu-Suche TSP

Tabulistenlänge = Wartezeit = w = 3
x = ABCEDA
best = ABCEDA
q(ABCEDA) = 26

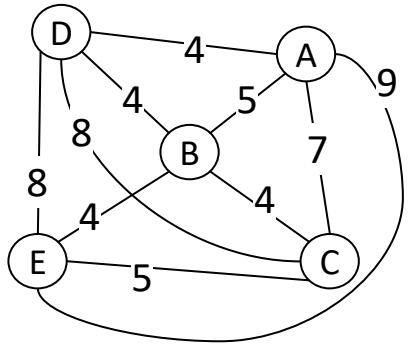


Tabu-Liste:

ABCDEA, 2
ABCEDA, 3

Beispiel: Tabu-Suche TSP

Tabulistenlänge = Wartezeit = w = 3
x = ABCEDA
best = ABCEDA
q(ABCEDA) = 26
N(x) = { ACBEDA, AECBDA, ADCEBA, ABECDA, ABDECA, ~~ABCDEA~~ }
q(ACBEDA) = 27
q(AECBDA) = 26
q(ADCEBA) = 26
q(ABECDA) = 26
q(ABDECA) = 29
~~q(ABCDEA) = 34~~

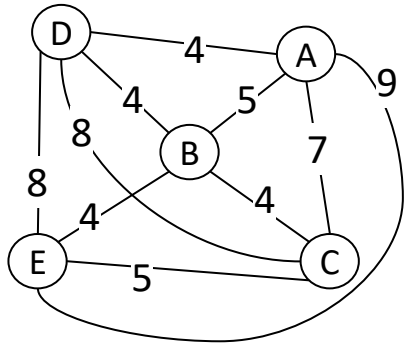


Tabu-Liste:

ABCDEA, 2
ABCEDA, 3

Beispiel: Tabu-Suche TSP

Tabulistenlänge = Wartezeit = w = 3
x = AECBDA
best = ABCEDA
q(ABCEDA) = 26

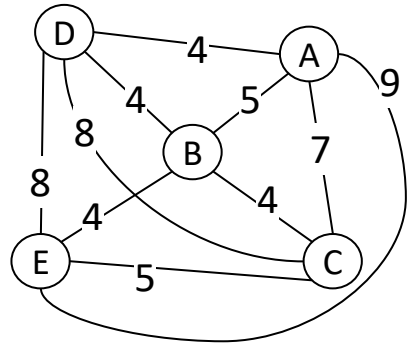


Tabu-Liste:

ABCDEA, 1
ABCEDA, 2
AECBDA, 3

Beispiel: Tabu-Suche TSP

Tabulistenlänge = Wartezeit = w = 3
x = AECBDA
best = ABCEDA
q(ABCEDA) = 26
N(x) = { ACEBDA, ~~ABCEDA~~, ADCBEA, AEBCDA, AEDBCA, AECDBA, }
q(ACEBDA) = 24
~~q(ABCEDA) = 26~~
q(ADCBEA) = 29
q(AEBCDA) = 29
q(AEDBCA) = 32
q(AECDBA) = 31

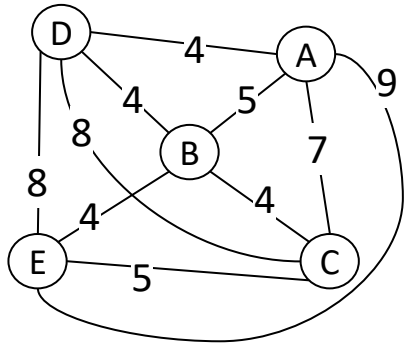


Tabu-Liste:

ABCDEA, 1
ABCEDA, 2
AECBDA, 3

Beispiel: Tabu-Suche TSP

Tabulistenlänge = Wartezeit = w = 3
x = ACEBDA
best = ACEBDA
q(AECBDA) = 24



Tabu-Liste:

ABCEDA, 1
AECBDA, 2
ACEBDA, 3

ACEBDA ist zwar eine optimale Route. Aber das Tabu-Verfahren kennt den Wert des Minimums nicht und läuft daher bis zum Abbruchkriterium!

Bewertung der Tabu-Suche

- Vorteile
  - viele Variationsmöglichkeiten (Nachbarschaft, Gütefunktion, Tabulistenlänge)
  - ➢auf andere Problemstellungen übertragbar
  - liefert bei vielen Anwendungen schnell gute Lösungen
  - Berechnet oft sogar die besten bekannten Lösungen
- Nachteile
  - viele Variationsmöglichkeiten erfordern ggf. zeitaufwändige Parametrisierung
  - $N(x)$ kann zu groß für eine effiziente Berechnung sein
    - Tabu-Suche definiert für diese Fälle keine Vorgehensweise
  - Sicherstellung der Suche im gesamten Lösungsraum schwierig
  - keine Garantie für Auffindung des globalen Optimums

Anwendung der Tabu-Suche

- Frameworks (z.B. OpenTS)
  - implementieren generelle Vorgehensweise
  - anwendungsspezifische Aspekte müssen implementiert bzw. spezifiziert werden
    - Nachbarschaft
    - Gütefunktion (Evaluation)
    - Datenstruktur von Tabulisteneinträgen
    - Länge der Tabulisten
    - Abbruchkriterium

CARL VON OSSIETZKY universität OLDENBURG

Evolutionäre Algorithmen

Idee: Evolutionäre Algorithmen

- Hier: Genetische Algorithmen
- Simulation der Evolution von Lebewesen
  - gute Eigenschaften setzen sich durch (fitness)
  - schlechte Eigenschaften sterben aus (survival of the fittest)
- Beispiel: Züchten von Hunden mit Schlappohren
  - wähle Hunde mit den längsten Ohren
  - paare sie
  - wiederhole, bis die Ohren lang genug sind
  - Idee: Langohr-Eigenschaft setzt sich durch

Vorgehen

- Population
  - Menge der Lösungen (Individuen)
- Individuum
  - eine Lösung, kodiert als Sequenz von Genen (z.B. Bits)
- Fitness
  - Gütefunktion

Vorgehen (2)

- Beginne mit Startpopulation
- Erzeuge daraus die nächste Generation
  - durch Selektion, Mutation, Rekombination
- Wiederhole bis Abbruchkriterium erfüllt
- Ergebnis: Individuum mit höchster Fitness nach x Generationen

CARL VON OSSIETZKY universität OLDENBURG

Startpopulation

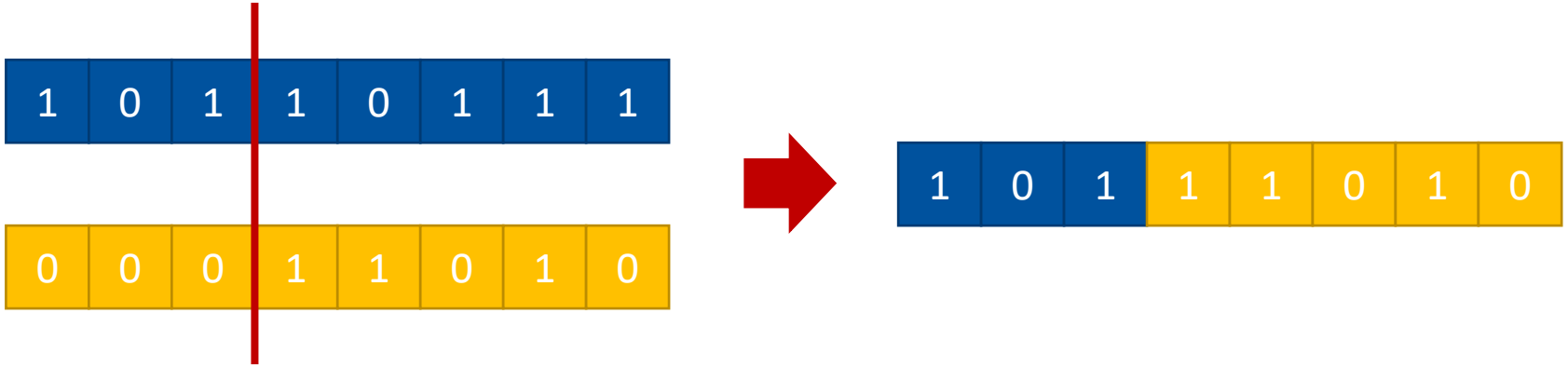
- Möglichst heterogen
- Soll die ganze Breite an möglichen Eigenschaften abdecken
- Oft: zufällig gewählt

Selektion

- Wähle Individuen aus der aktuellen Generation für die Erzeugung der nächsten Generation
- Auswahlwahrscheinlichkeit direkt proportional zur Fitness

Rekombination

- Wähle zwei zufällige Eltern (Selektion)
- Bestimme zufällig einen Schnittpunkt
- Erzeuge neues Individuum durch Rekombination



- Problem: nicht jede Rekombination ergibt ein gültiges Individuum (z.B. bei TSP)

➢komplexere Operationen

Mutation

- Verändere zufällig (aber mit geringer Wahrscheinlichkeit) einzelne Gene
- Z.B. durch Bit-Swap

Abbruchkriterium

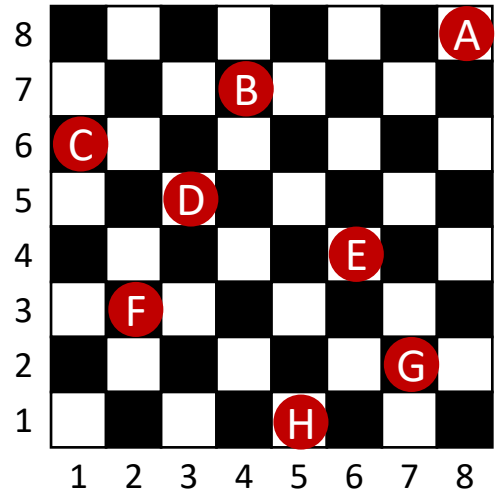
- Nach fester Zahl von Generationen
- Bei vorgegebener Mindestgüte
- Nach nur noch geringer Verbesserung

Anwendungsfall: Damenproblem

- Wie können auf einem Schachbrett acht Damen positioniert werden, so dass keine Dame eine andere schlagen kann?
  - jede Dame darf jede andere schlagen (keine Farben)
  - übliche Bewegungsmuster (horizontal, vertikal, diagonal)

➢keine zwei Damen in der gleichen Zeile, Spalte oder Diagonale

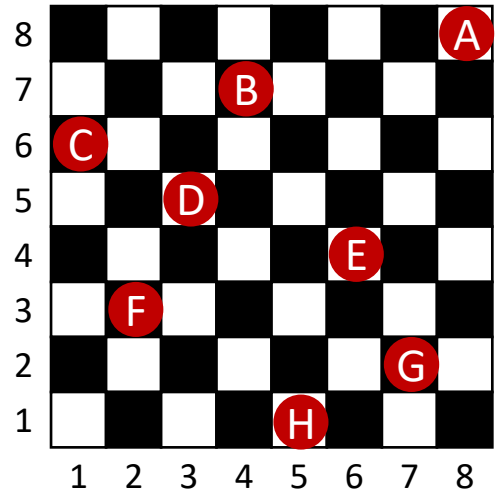
  - erweiterbar: n Damen auf $n \times n$ Feld



Fitness

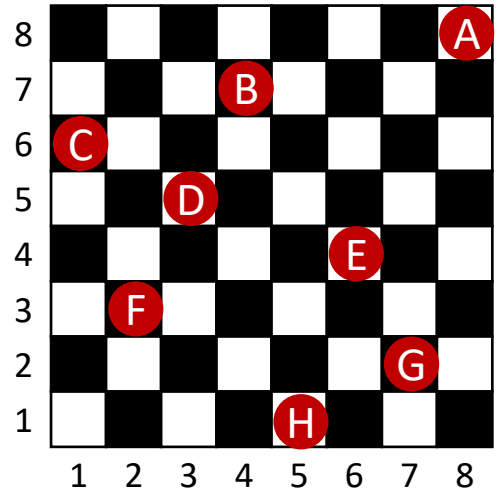
- Zahl der Damen, die sich paarweise nicht bedrohen
- AB, AC, AD, AE, AF, AG, AH, BC, BD, BE, …
- Minimum: 0
  (alle Damen bedrohen sich gegenseitig)
- Maximum: $\binom{n}{2} = \frac{n \cdot (n-1)}{2}$

  (keine Damen bedrohen sich gegenseitig)



Individuen

- Sequenz von n Positionen:
  Zahl an Stelle i: Position der Dame in Zeile i
  (eindeutig, da keine zwei Damen pro Zeile möglich sind)

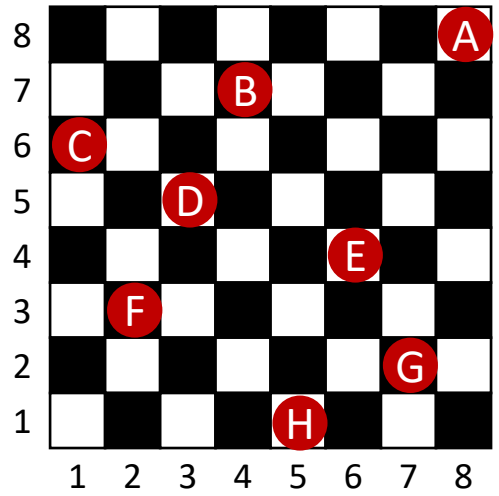


$$(5, 7, 2, 6, 3, 1, 4, 8)$$

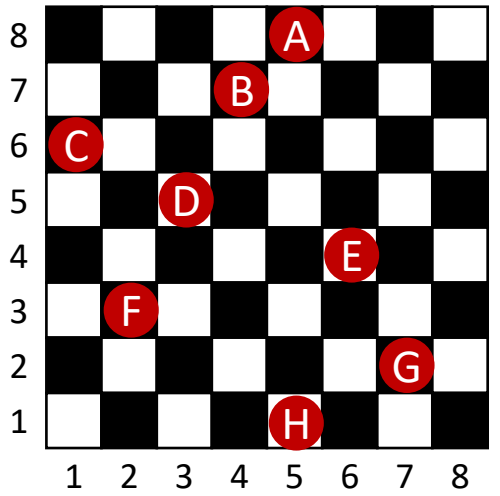
Mutation

- Ändere einen zufälligen Wert aus der Sequenz von Positionen

➢verschiebe Dame i in eine andere Spalte

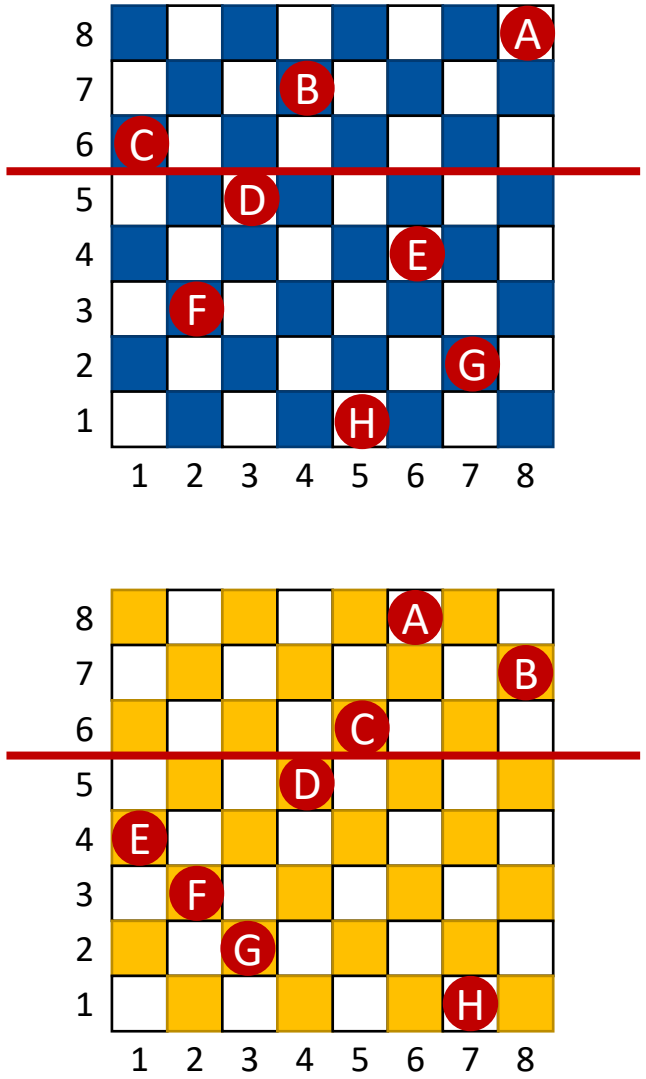


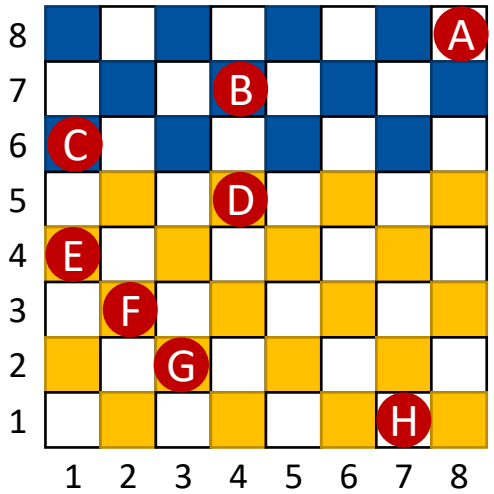
$(5, 7, 2, 6, 3, 1, 4, 8)$



$(5, 7, 2, 6, 3, 1, 4, 5)$

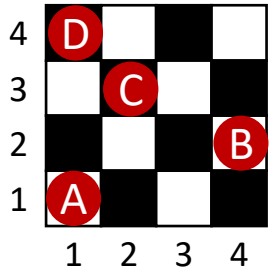
Rekombination





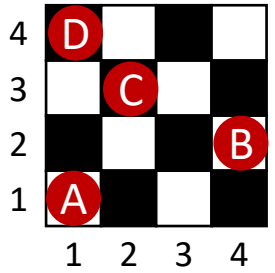
Beispiel: 4-Damenproblem

Startpopulation: { (1, 4, 2, 1) }
Fitness: q(1, 4, 2, 1) = 4



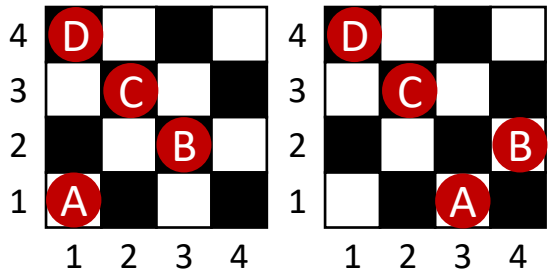
Beispiel: 4-Damenproblem

Startpopulation: { (1, 4, 2, 1) }
Fitness: q(1, 4, 2, 1) = 4
Mutationen: (1, 4, 2, 1) → (1, 3, 2, 1), (1, 4, 2, 1) → (3, 4, 2, 1)



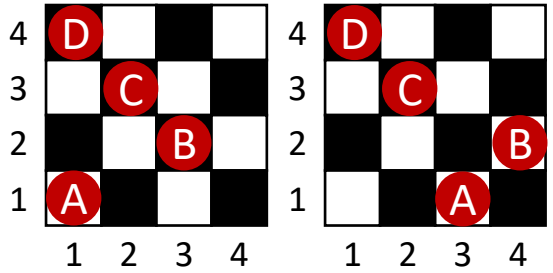
Beispiel: 4-Damenproblem

Startpopulation: { (1, 4, 2, 1) }
Fitness: q(1, 4, 2, 1) = 4
Mutationen: (1, 4, 2, 1) → (1, 3, 2, 1), (1, 4, 2, 1) → (3, 4, 2, 1)
Population (Gen. 2): { (1, 3, 2, 1), (3, 4, 2, 1) }
Fitness: q(1, 3, 2, 1) = 2, q(3, 4, 2, 1) = 4



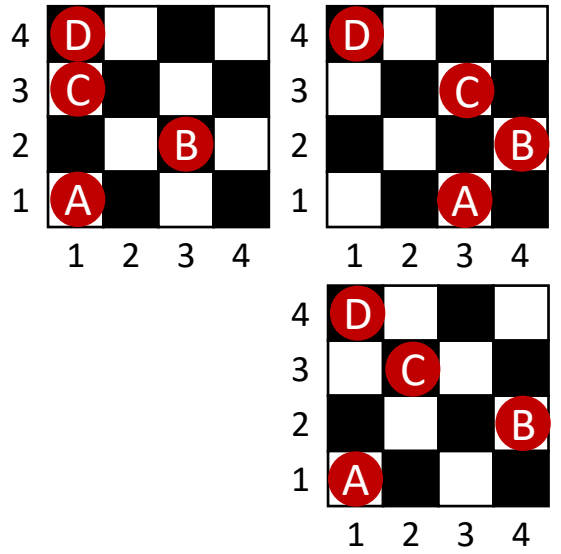
Beispiel: 4-Damenproblem

Startpopulation: { (1, 4, 2, 1) }
Fitness: q(1, 4, 2, 1) = 4
Mutationen: (1, 4, 2, 1) → (1, 3, 2, 1), (1, 4, 2, 1) → (3, 4, 2, 1)
Population (Gen. 2): { (1, 3, 2, 1), (3, 4, 2, 1) }
Fitness: q(1, 3, 2, 1) = 2, q(3, 4, 2, 1) = 4
Mutationen: (1, 3, 2, 1) → (1, 3, 1, 1), (3, 4, 2, 1) → (3, 4, 3, 1)
Rekombination: (1 | 3, 2, 1) + (3 | 4, 2, 1) → (1, 4, 2, 1)



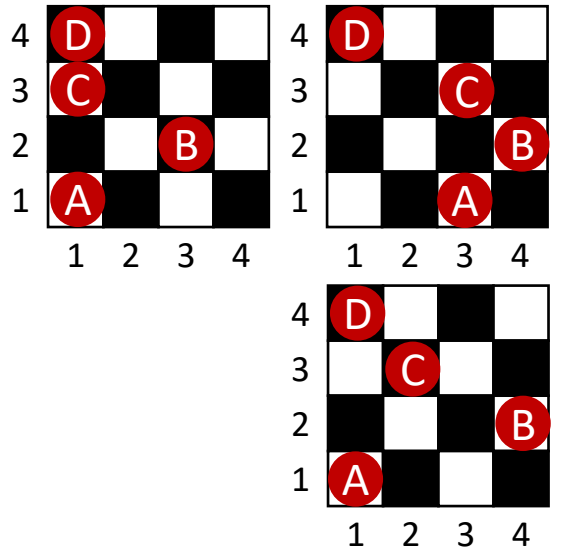
Beispiel: 4-Damenproblem

Startpopulation: { (1, 4, 2, 1) }
Fitness: q(1, 4, 2, 1) = 4
Mutationen: (1, 4, 2, 1) → (1, 3, 2, 1), (1, 4, 2, 1) → (3, 4, 2, 1)
Population (Gen. 2): { (1, 3, 2, 1), (3, 4, 2, 1) }
Fitness: q(1, 3, 2, 1) = 2, q(3, 4, 2, 1) = 4
Mutationen: (1, 3, 2, 1) → (1, 3, 1, 1), (3, 4, 2, 1) → (3, 4, 3, 1)
Rekombination: (1 | 3, 2, 1) + (3 | 4, 2, 1) → (1, 4, 2, 1)
Population (Gen. 3): { (1, 3, 1, 1), (3, 4, 3, 1), (1, 4, 2, 1) }



Beispiel: 4-Damenproblem

Startpopulation: { (1, 4, 2, 1) }
Fitness: q(1, 4, 2, 1) = 4
Mutationen: (1, 4, 2, 1) → (1, 3, 2, 1), (1, 4, 2, 1) → (3, 4, 2, 1)
Population (Gen. 2): { (1, 3, 2, 1), (3, 4, 2, 1) }
Fitness: q(1, 3, 2, 1) = 2, q(3, 4, 2, 1) = 4
Mutationen: (1, 3, 2, 1) → (1, 3, 1, 1), (3, 4, 2, 1) → (3, 4, 3, 1)
Rekombination: (1 | 3, 2, 1) + (3 | 4, 2, 1) → (1, 4, 2, 1)
Population (Gen. 3): { (1, 3, 1, 1), (3, 4, 3, 1), (1, 4, 2, 1) }
…



CARL VON OSSIETZKY universität OLDENBURG

Anwendung der Gen. Algorithmen

- Frameworks (z.B. JGAP)
  - implementieren generelle Vorgehensweise
  - anwendungsspezifische Aspekte müssen implementiert bzw. spezifiziert werden
    - Kodierung der Lösungen (Individuen)
    - Fitness (Gütefunktion)
    - Mutations- und Rekombinationsoperator
    - Selektionsstrategie
    - Abbruchkriterium (oft Anzahl der Generationen)
    - Größe der Population

CARL VON OSSIETZKY universität OLDENBURG

Lernziele

- TSP
- Aufzählungsmethoden
  - Backtracking (Wiederholung)
  - Branch & Bound
- Relaxation, Approximation
- Heuristiken
  - Lokale Suche
  - Tabu Suche
- Evolutionäre Algorithmen