

Algorithmen im Chip-Entwurf 6

Exakte Optimierungsverfahren

Andreas Koch
FG Eingebettete Systeme
und ihre Anwendungen
TU Darmstadt

Organisatorisches

- **Anmeldung zur Prüfung**
 - Auch im klausurlosen 4SWS-Modus!
- **Im 2SWS-Modus**
 - Zwei mündliche Prüfungen
 - Je 20 Minuten
 - Vormittags & Nachmittagstermine
 - Ausschuchen!

- **Probleme im VLSI CAD-Bereich**
 - Am Beispiel: Travelling Salesman (TSP)

- **Exakte Verfahren**
 - Backtracking
 - Branch-and-Bound
 - Dynamic Programming
 - Integer Linear Programming

- **Zusammenfassung**

Art der Probleme

- **Viele Probleme im Bereich VLSI CAD sind**
 - NP-vollständig
 - NP-hart
- **Exakt lösbar nur für kleine Problemgrößen**
- **Falls sub-optimale Lösungen akzeptabel**
 - **Näherungsverfahren**
 - ◆ Garantieren eine vorgegebene Lösungsqualität
 - **Heuristiken**
 - ◆ Schwankende Lösungsqualität

Exakte Lösungsverfahren

■ Erschöpfende Suche

- Durchlaufen des gesamten Lösungsraums
- Beispiel: Backtracking

■ Eliminierung "schlechter" Ansätze

- Abschätzung aus Teillösung
- Beispiel: Branch-and-Bound

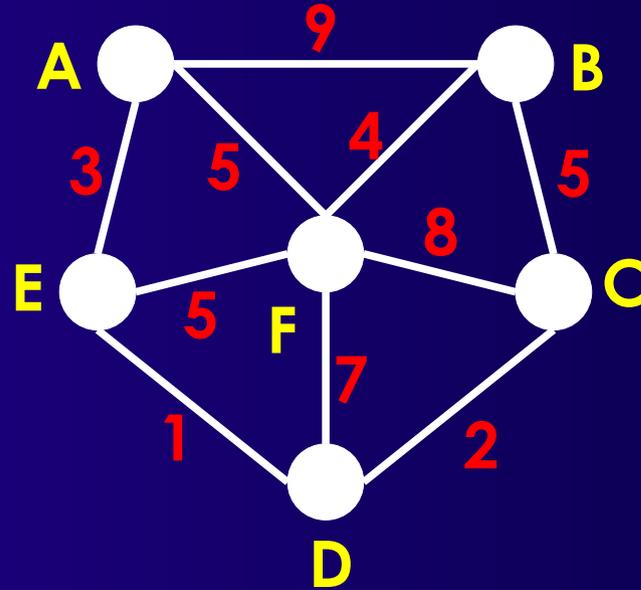
■ Wiederverwendung alter Ergebnisse

- Beispiel: Dynamic Programming

■ Mathematisches Modell

- Beispiel: (Integer) Linear Programming

Travelling Salesman Problem



■ TSP

■ *Einfacher* Zyklus durch alle Knoten mit minimaler Länge

- Jeder Knoten nur einmal besucht
- Minimale Kantengewichte

■ NP-vollständig

■ Instanz $I = (F, c)$

- Lösungsraum F
- Kostenfunktion $c: F \rightarrow \mathbb{R}$

■ Lösung $\underline{f} \in F: \underline{f} = [f_1, \dots, f_n]^T$

- Explizite Einschränkungen: Wertebereiche f_i
- Implizite Einschränkungen: Abhängigkeiten

■ Teillösung \underline{f}^{\sim}

- Einige f_i undefiniert
- Spannt Unterraum von F auf

Backtracking 1

- Systematisch durch ganzen Lösungsraum
- Beginne mit komplett undefinierter Teillösung, $k = 0$
- Weise f_k einen möglichen Wert zu
- Gehe zu nächstem f_k : $k = k + 1$
- Solange, bis
 - Komplette Lösung ($k = n$), neue beste Lösung?
 - oder implizite Einschränkungen greifen
- Zurück zu letztem änderbarem f_k

Backtracking 2

```
float best_cost;
solution_element cur_sol[n], best_sol[n];

backtrack(int k) {
    float new_cost;
    solution_element sol_el;
    if (k == n){
        new_cost := cost(cur_sol);
        if (new_cost < best_cost) {
            best_cost := new_cost;
            best_sol := copy(cur_sol);
        }
    } else
        foreach (sol_el ∈ allowed(cur_sol, k)) {
            cur_sol[k] := sol_el;
            backtrack(k+1);
        }
}

main {
    best_cost := ∞
    backtrack(0);
    report(best_sol);
}
```

TSP via Backtracking 1

■ Lösung: Folge von Knoten mit

- Kanten zwischen benachbarten Elementen
- Erstes und letztes Element sind gleich
- Alle anderen Elemente sind unterschiedlich

■ Modell

- Folge $f = (f_1, \dots, f_n)$ mit Knoten als f_i
- $f_1 = f_n = v, f_i \in V \setminus \{v\}$ für $i \notin \{1, n\}$ (explizit)
- $f_{i+1}: (f_i, f_{i+1}) \in E$ (implizit)
- $f_i \neq f_j$ für $i \neq j \wedge i, j \notin \{1, n\}$ (implizit)

Branch-and-Bound 1

- Teillösung $\underline{f}^{(k)}$
- $D(\underline{f}^{(k)})$: Menge aus $\underline{f}^{(k)}$ herleitbarer Lsgen.
- **Abschätzung**
 - Gegeben $\underline{f}^{(k)}$
 - Kosten $c^{\sim}(\underline{f}^{(k)})$ der besten vollständigen Lösung in $D(\underline{f}^{(k)})$?
- **Verwerfe $\underline{f}^{(k)}$, falls $c^{\sim}(\underline{f}^{(k)}) > \text{best_cost}$**
 - Suchbaum wird gestutzt
- **Aus Teillösung Endkosten „erraten“**
 - Abschätzung!

Branch-and-Bound 2

```
float best_cost;
solution_element cur_sol[n], best_sol;

b_and_b(int k) {
    float new_cost;
    solution_element sol_el;
    if (k == n){
        new_cost := cost(cur_sol);
        if (new_cost < best_cost) {
            best_cost      := new_cost;
            best_sol       := copy(cur_sol);
        }
    } elseif (lower_bound_cost(cur_sol, k) >= best_cost)
        /* tu nix, stütze baum */;
    else foreach (sol_el ∈ allowed(cur_sol, k)) {
        cur_sol[k] := sol_el;
        b_and_b(k+1);
    }
}

main {
    best_cost := ∞
    b_and_b(0);
    report(best_sol);
}
```

Branch-and-Bound 3

■ Effekt der Abschätzung

- Reale Kosten höher als geschätzte Kosten
 - ◆ Zu optimistisch ("ja, es lohnt sich weiterzumachen")
 - ◆ Überflüssige Schritte
- Reale Kosten niedriger als geschätzte Kosten
 - ◆ Zu pessimistisch ("nein, das bringt nichts mehr")
 - ◆ Optimum wird möglicherweise übersehen!
 - ❖ Keine exakte Lösung mehr!

➤ $c_{\sim}(f_{\sim}^{(k)})$ sollte möglichst genau sein

- Darf aber Kosten keinesfalls überschätzen!

■ Wunsch: Schneller als vollständige Suche
→ Abwägen!

Branch-and-Bound 4

■ Aufbau der Abschätzungsfunktion

$$\tilde{c}(\tilde{f}^{(k)}) = \tilde{g}(\tilde{f}^{(k)}) + \tilde{h}(\tilde{f}^{(k)})$$

Basiert auf bekannten $f_i, i \leq k$

TSP: Länge des bekannten Pfades

Basiert auf unbekanntem $f_i, i > k$

TSP: Verbinde verbliebene Knoten irgendwie

→ **Minimaler überspannender Baum (MST)**

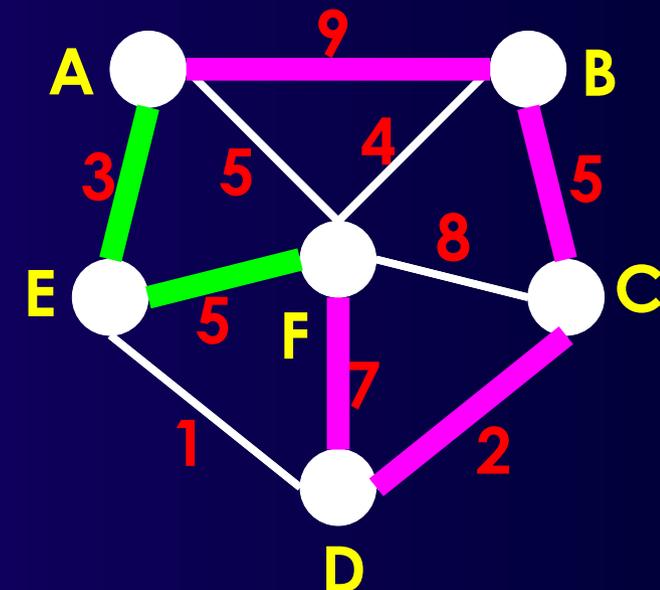
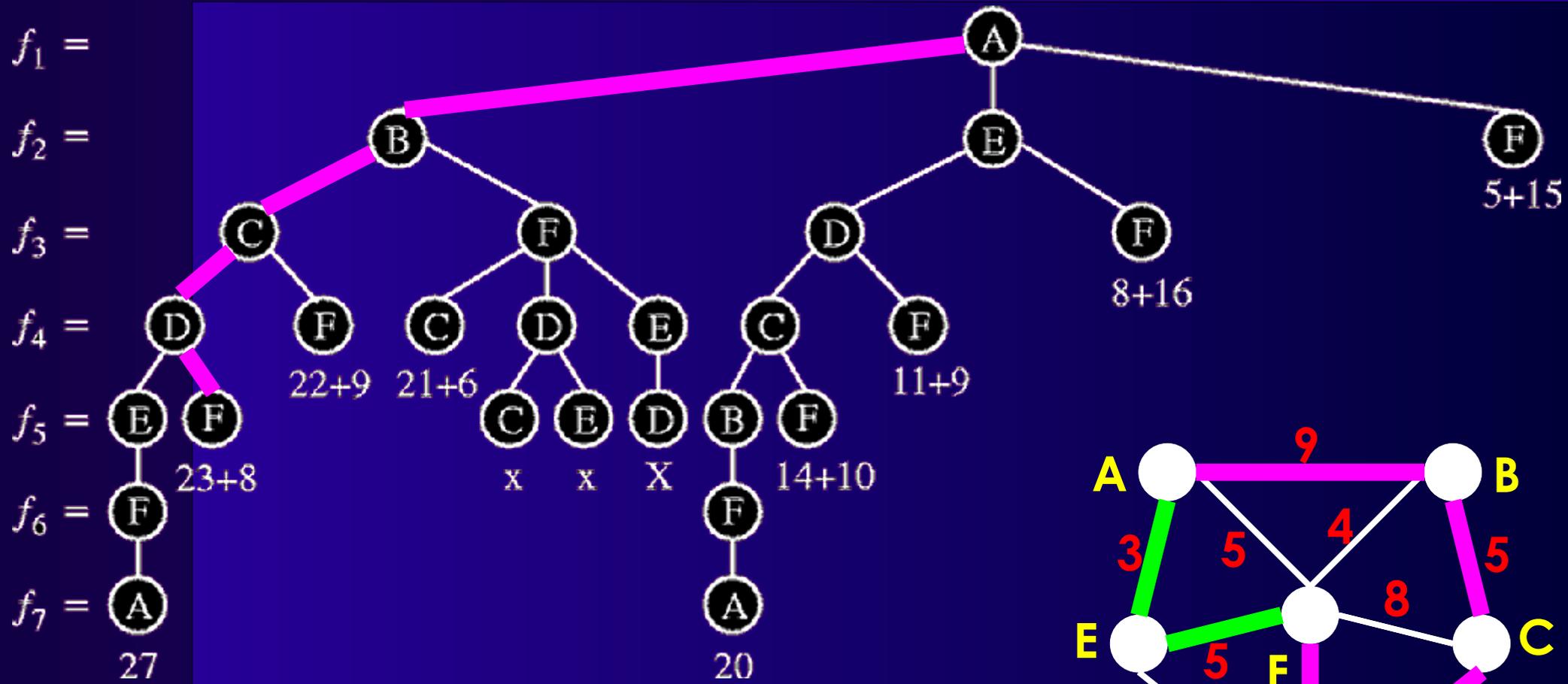
■ MST

- Weniger Einschränkungen als TSP
 - ◆ Kein Zyklus
 - ◆ Kein einfacher Pfad
- MST immer kürzer oder gleich TSP
 - ◆ Optimistische Abschätzung

■ MST läuft in $O(n^2)$: Prim's Algorithmus

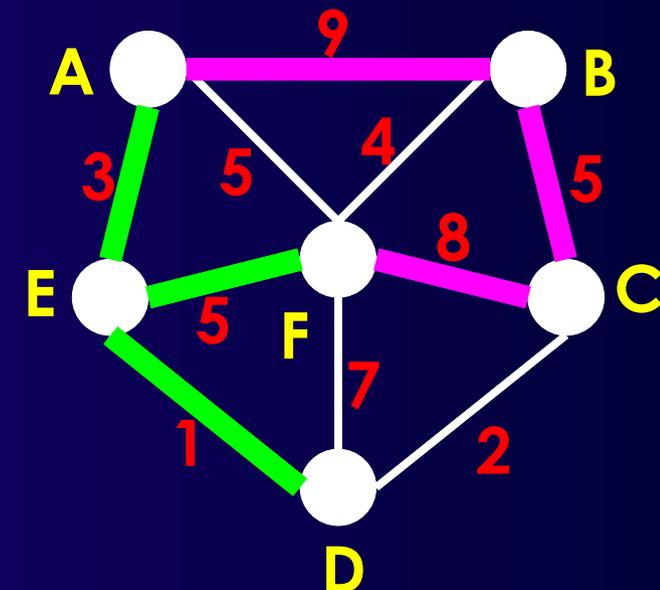
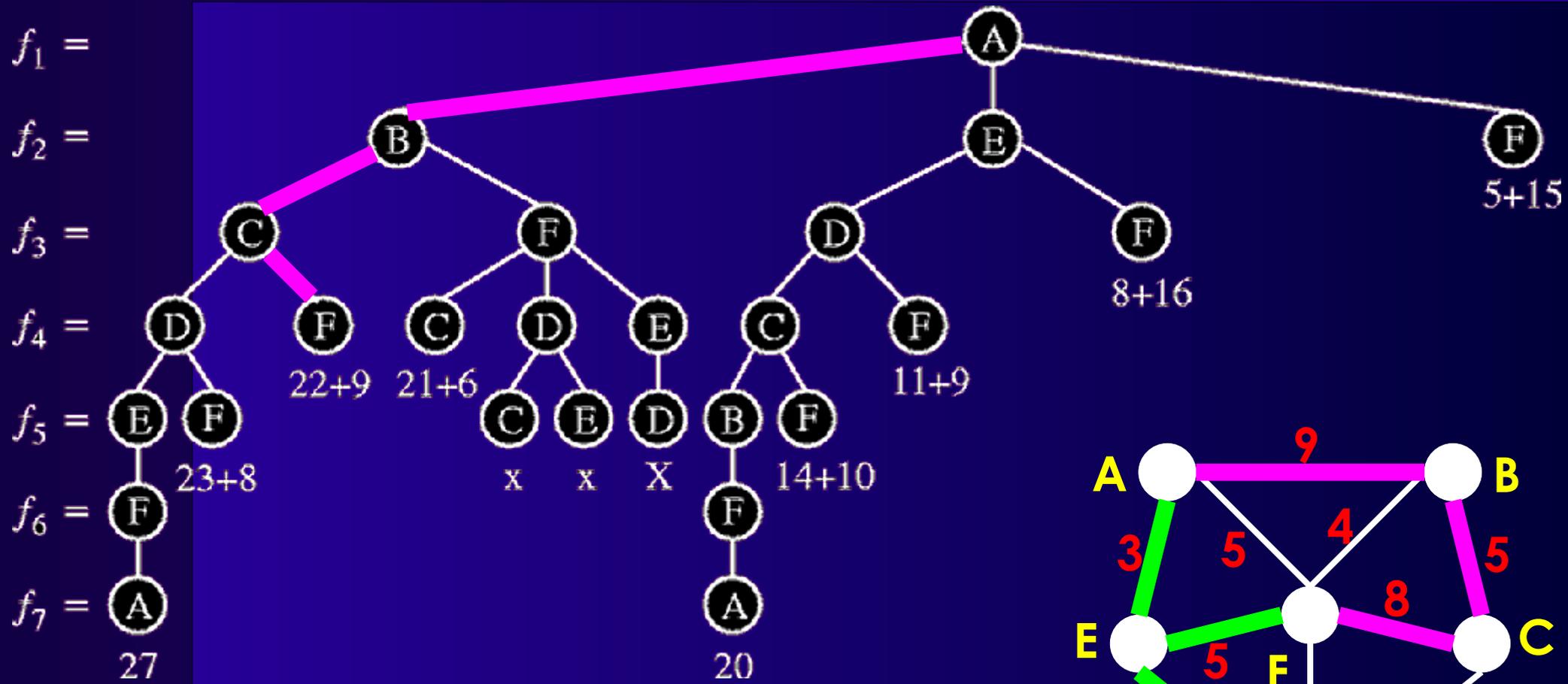
- Besser als NP-vollständig für TSP

TSP via Branch-and-Bound 1



- Erste Lösung wird *immer* berechnet
- Dann „bound“

TSP via Branch-and-Bound 2



■ Erste Lösung wird *immer* berechnet

- **Verschiedene Sucharten**
 - Welche Teillösung weiter verfeinern?
- **Bisher DFS**
- **Alternative Vorgehensweise**
 - BFS
 - Greedy
 - ◆ Schnelles Finden einer Lösung
 - ◆ Maximales Stutzen

Dynamic Programming 1

- **Wiederverwendung von Lösungen**
- **Prinzip der Optimalität**
 - Lösung eines komplexen Problems kann optimal aus den optimalen Lösungen von Teilproblemen zusammengesetzt werden
 - Gilt nicht für alle Probleme!
- **p: Parameter für Problemlkomplexität**
 - $p = k$: Gesamtproblem
 - $p < k$: Teilproblem
 - $p = 0$ oder $p = 1$: Kleinstes Problem

Dynamic Programming 2

■ Fibonacci-Zahlen: 0,1,1,2,3,5,8,13,21,34,...

- $F_n = F_{n-1} + F_{n-2}$ mit $F_0=0, F_1=1$

```
fib(int n) {  
  if (n=0)  
    return(0);  
  else if (n=1)  
    return(1);  
  else  
    return(fib(n-1)+fib(n-2));  
}
```

- $F_n/F_{n+1} \simeq 1,6 \Rightarrow F_n > 1,6^n$
- Summiere 0 und 1
⇒ $1,6^n$ Schritte

```
int Fib[MAXFIB];
```

```
fib(int n) {  
  Fib[0] := 0;  
  Fib[1] := 1;  
  for (i=2; i<=n; ++i)  
    Fib[i] := Fib[i-1]+Fib[i-2];  
  return(Fib[n]);  
}
```

- n Schritte
- Teillösungen bei $p = i$
- Lösung bei $p = n$

Dynamic Programming 3

■ Dijkstras Kürzester Pfad - Algorithmus

- "Finde kürzesten Pfad über die an v_s nahegelegensten k Knoten"
 - Komplexitätsparameter: $p = k$
 - $p = 0$: $u.dist = w((v_s, u))$ für alle $u \in V$
 - ◆ Finde Knoten u mit min. $dist$ ← Teillösungen in $dist$
 - ◆ Transferiere u von V nach T
 - ◆ Aktualisiere $dist$ aller Knoten in V
 - $p = k+1$: Finde Knoten $u \in V$ mit min. $dist$
 - ◆ Transferiere u von V nach T
 - ◆ Aktualisiere $dist$ aller Knoten in V
- 

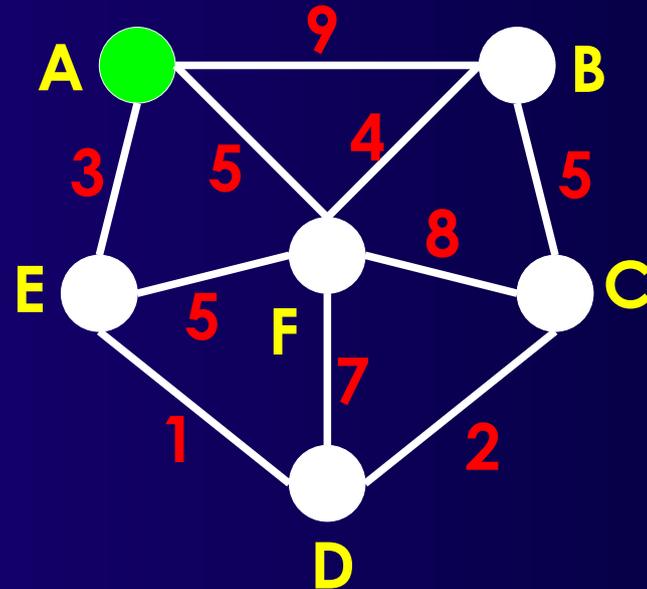
TSP via Dynamic Programming 1

- $G(V,E)$, Kanten gewichtet mit w
- Wähle beliebiges $v_s \in V$
- $p = k$: "finde kürzeste Pfade von v_s zu $v \in V \setminus \{v_s\}$ durch k Zwischenknoten"
- $C(S,v)$: Länge des kürzesten Pfades von v_s nach v durch die Knoten in S
- TSP ist $C(S \setminus \{v_s\}, v_s)$: $p=k+1$

$$C(S, v) = \min_{m \in S} [C(S \setminus \{m\}, m) + w((m, v))]$$

TSP via Dynamic Programming 2

- TSP mit $v_s=A$
- $p=0$: $C(\emptyset, v) = w((v_s, v))$ für $(v_s, v) \in E$,
sonst ∞
 - $C(\emptyset, B) = 9$
 - $C(\emptyset, C) = \infty$
 - $C(\emptyset, D) = \infty$
 - $C(\emptyset, E) = 3$
 - $C(\emptyset, F) = 5$



TSP via Dynamic Programming 3

■ Zwischenergebnisse für $p = 0$

- $C(\emptyset, B) = 9, C(\emptyset, C) = \infty, C(\emptyset, D) = \infty,$
 $C(\emptyset, E) = \infty, C(\emptyset, F) = 5$

■ $p=1$: Berechne alle Kombinationen von $|S|=1$ und v (5 x 4 Mögl.). Auszug:

- $C(\{B\}, C) = C(\emptyset, B) + w((B, C)) = 9 + 5 = 14$
- $C(\{B\}, F) = C(\emptyset, B) + w((B, F)) = 9 + 4 = 13$
- $C(\{F\}, B) = C(\emptyset, F) + w((F, B)) = 5 + 4 = 9$
- ...

■ min entfällt bei $|S|=1$ (nur ein Element!)

TSP via Dynamic Programming 4

■ Einige Zwischenergebnisse für $p=1$:

- $C(\{B\}, F) = 13, C(\{F\}, B) = 9$

■ $p = |S| = 2$: $\{5 \times 4\} \times 3$ Möglichkeiten, *Auszug:*

$$C(\{B, F\}, C) =$$

$$\min [C(\{B\}, F) + w((F, C)), C(\{F\}, B) + w((B, C))]$$

$$13 + 8$$

$$9 + 5$$

$$\rightarrow C(\{B, F\}, C) = 14$$

TSP via Dynamic Programming 5

■ Ein Zwischenergebnis für $p=2$:

- $C(\{B, F\}, C) = 14$
- "Kürzester Pfad von A nach C über $\{B, F\}$ hat Länge 14"

■ Ad nauseam bis $p=|S|=n-1$

- $C(\{B, C, D, E, F\}, A) = 20$
- "Kürzester Pfad von A nach A über $\{B, C, D, E, F\}$ hat Länge 20"

→ TSP

■ Immer noch NP-hart: 2^n Untermengen

- Untersuche aber nur optimale Teillösungen

Lineare Programmierung 1

- Mathematische Modelle als Grundlage
- Beispiel: Optimiere auf max. Umsatz

	Preis	Rohstoff A	Rohstoff B
Produkt 1	550	42	23
Produkt 2	250	14	53
Liefermenge		100	200

- Modell: x_1 Produkt 1, x_2 Produkt 2

$$\max : 550 x_1 + 250 x_2$$

$$42 x_1 + 14 x_2 \leq 100$$

$$23 x_1 + 53 x_2 \leq 200$$

Lineare Programmierung 2

■ Lösbar in P

- Ellipsoid Verfahren (1979)

■ Praktisch schneller sind Verfahren in NP

- Simplex (1947)

■ Theoretisch besser (auch in P)

- Interior Point, projective method (1984)

■ Lösung durch "LP Solver"

- Ip_solve, GLPK, CLP (public domain)
- CPLEX (kommerziell, aber viel schneller!)

■ Beispiel: $x_1=1,31303$, $x_2=3,20378$

- Umsatz 1523,11

■ Problem

- Häufig nur ganzzahlige Variablen erlaubt

➤ Integer Lineare Programmierung (ILP)

■ Lösungsmethoden komplizierter

- Rundung *nicht* sinnvoll
 - ◆ Sub-optimal
 - ❖ Beispiel $x_1' = 1, x_2' = 3$: Gewinn 1300
 - ◆ Ungültige Werte

■ Beispiel: $x_1 = 2, x_2 = 1$, Gewinn 1350

→ Lösungsverfahren jetzt NP-vollständig

■ Häufig weitere Einschränkung

- Variablen nur 0 oder 1
- 0-1 ILP

■ Lösungsverfahren

- LP kombiniert mit branch-and-bound
- SAT (Erfüllbarkeitsproblem), nur bei 0-1

TSP via 0-1 ILP - 1

- Pro Kante $e_i \in E$ ein x_i , $1 \leq i \leq |E| = k$
 - $x_i = 1$ wenn x_i im optimalen Zyklus, 0 sonst
 - Entscheidungsvariablen
 - Zykluslänge (Gewicht)

$$\sum_{i=1}^k w(e_i) x_i$$

■ Zyklus (I)

- An jedem Knoten müssen zwei Kanten selektiert werden

$$v_1: x_1 + x_2 + x_3 + x_4 = 2$$

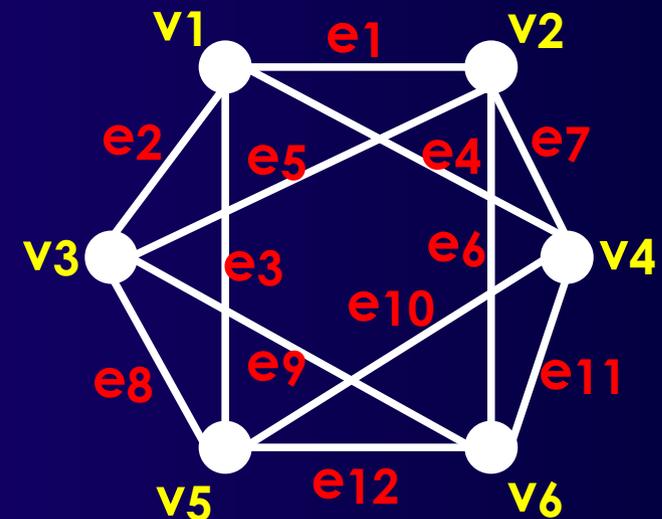
$$v_2: x_1 + x_5 + x_6 + x_7 = 2$$

$$v_3: x_2 + x_5 + x_6 + x_7 = 2$$

$$v_4: x_4 + x_7 + x_{10} + x_{11} = 2$$

$$v_5: x_3 + x_8 + x_{10} + x_{12} = 2$$

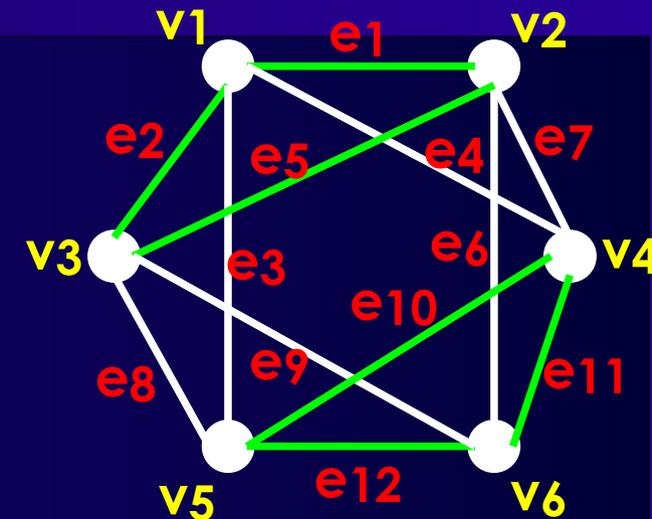
$$v_6: x_6 + x_9 + x_{11} + x_{12} = 2$$



TSP via 0-1 ILP - 3

■ Zyklus (II)

- Vermeide unverbundene Zyklen
- Kleinster Zyklus: 3 Knoten
 - ◆ Da 2 Kanten pro Knoten
- Bestimme alle unverbundenen Zyklen
- Fordere Verbindungen dazwischen



$$\{v_1, v_2, v_3\} + \{v_4, v_5, v_6\} : x_3 + x_4 + x_6 + x_7 + x_8 + x_9 \geq 2$$

$$\{v_1, v_3, v_5\} + \{v_2, v_4, v_6\} : x_1 + x_4 + x_5 + x_9 + x_{10} + x_{12} \geq 2$$

$$\{v_1, v_2, v_4\} + \{v_3, v_5, v_6\} : x_2 + x_3 + x_5 + x_6 + x_{10} + x_{11} \geq 2$$

$$\{v_1, v_4, v_5\} + \{v_3, v_2, v_6\} : x_1 + x_7 + x_{11} + x_{12} + x_8 + x_2 \geq 2$$

(I) Lineare Programmierung

- **Modelle werden i.d.R. generiert**
 - Spezielle Sprachen: z.B. AMPL, GNU MathProg
 - Mittels konventioneller Programme
 - Textdatei
 - API in eingebundene Bibliothek
- **I(LP) Parameter**
 - Anzahl Variablen
 - Anzahl Bedingungen
- **Mixed ILP: LP und ILP**
- **Non-Linear Programming (NLP)**

- Nächste Vorlesung: Dienstag
- Kapitel 9 bis einschließlich 9.3
 - Verdrahtung

Zusammenfassung

- **Exakte Lösungsverfahren**
- **Backtracking**
 - Erschöpfende Suche
- **Branch-and-Bound**
 - Stutzen des Suchbaumes
- **Dynamic Programming**
 - Wiederverwendung von Teillösungen
- **(Integer) Lineare Programmierung**
 - Mathematische Modelle