Das große "O"

- Aufwandsklasse $O(g)$ für $g: \mathbb{N} \longrightarrow \mathbb{R}^+$ enthält alle Funktionen $f: \mathbb{N} \longrightarrow \mathbb{R}^+$ mit

$$f(n) \leq c \cdot g(n) \text{ für alle } n \geq n_0$$

  c, n_0: konstant und größer als 0

- $O(g)$ beschreibt alle Probleme, die eine algorithmische Lösung op besitzen mit $T^{op} \in O(g)$
- Obere Schranke, bei der konstante Faktoren und kleine Eingaben nicht zählen.

Das große “O”

- Idee: wähle g einfach und einprägsam, z.B.

$$\log n, n, n \cdot \log n, n^2, n^3, \ldots 2^n, \ldots$$

  - $\log n$: logarithmisch
  - n: linear
  - n^2: quadratisch
  - n^3: kubisch
  - 2^n: exponentiell

Beispiele

- $T^{mergesort} \in O(n \cdot \mathrm{ld}\, n)$
- $T^{insort} \in O(n^2)$
- $T^{xmpl}(n) = 1/3n \cdot \mathrm{ld}\, n + 22n + 1/2n^2 + 7$

$$\rightsquigarrow \quad T^{xmpl} \in O(n^2),$$

d.h. $xmpl$ hat quadratischen Aufwand.

Eigenschaften

1. $f \in O(f)$.
2. $O(f + c) = O(f)$ (c konstant).
3. $O(c * f) = O(f)$ (c konstant).
4. $f_i \in O(g_i)$ $(i = 1, 2) \Longrightarrow$ $f_1 + f_2 \in O(g_1 + g_2)$ und $f_1 * f_2 \in O(g_1 * g_2)$.
5. $O(f_1 + f_2) = O(\max(f_1, f_2))$.
6. Falls f ein Polynom vom Grad k ist, gilt $f \in O(n^k)$.
7. $f \in O(g)$ und $g \in O(h) \Longrightarrow f \in O(h)$.

Zusammenhang zwischen verschiedenen Aufwandsklassen

$$O(1) \subset O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^2) \subset O(n^3) \subset \cdots \subset O(2^n) \subset O(3^n) \subset \cdots \subset O(n!) \subset O(n^n).$$

Linear, quadratisch und exponentiell im Vergleich

n	n^2	2^n
10	100	$1024 \sim 10^3$
20	400	$1048576 \sim 10^6$
30	900	$1073741824 \sim 10^9$
40	1600	$10999511627776 \sim 10^{13}$
50	2500	$1125899906842624 \sim 10^{15}$
60	3600	$1152921504600686976 \sim 10^{18}$
70	4900	$1180591620717411303424 \sim 10^{21}$

ohne Gewähr

Mit Supercomputer gegen Aufwand (?)

- Szenario: entwickle ein Programm, das bei Eingabegröße n gerade $n \mid n^2 \mid 2^n$ FLOPs benötigt und miete für 3 Stunden ($\sim 10^4$ Sek.) einen Supercomputer mit 100 Tera-FLOPs pro Sekunde (10^{14} FLOPs/Sek.).
- Welche Eingabegröße lässt sich bearbeiten?

n	n^2	2^n
10^{18}	10^9	60

Mit Supercomputer gegen Aufwand (?)

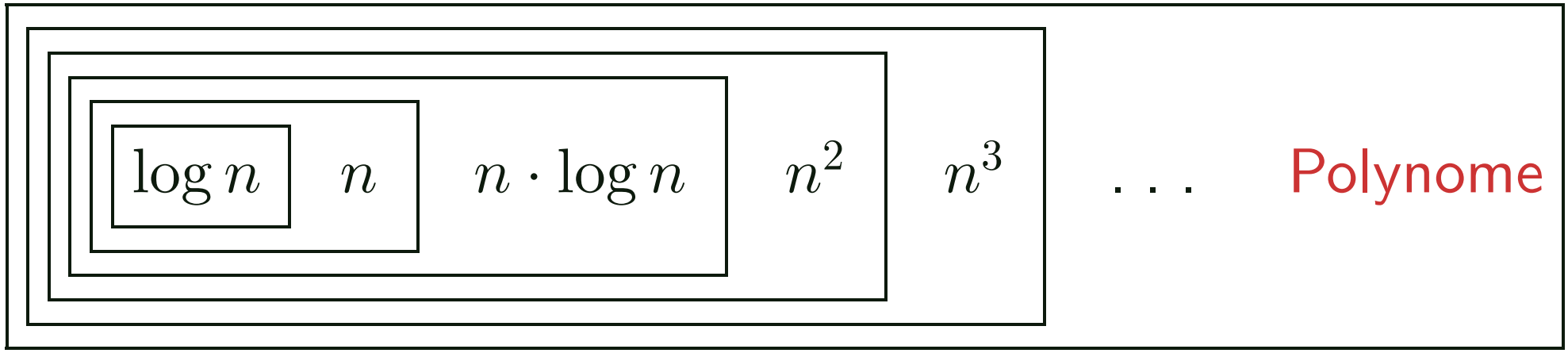
- Szenario: entwickle ein Programm, das bei Eingabegröße n gerade $n \mid n^2 \mid 2^n$ FLOPs benötigt und miete für 3 Stunden ($\sim 10^4$ Sek.) einen Supercomputer mit 100 Tera-FLOPs pro Sekunde (10^{14} FLOPs/Sek.).
- Welche Eingabegröße lässt sich bearbeiten?

n	n^2	2^n
10^{18}	10^9	60
10^{21}	$32 \cdot 10^9$	70 $(*)$

$(*)$: bei Rechenzeitverbesserung um Faktor 1000

Polynomieller Aufwand

- Probleme mit polynomiellem Aufwand können mit herkömmlichen Rechnern in verfügbarer Zeit gelöst werden (wenn der Exponent nicht zu groß ist).



Die Klassen P und NP

P enthält die Probleme, die mit herkömmlichen Computern zeitgerecht lösbar sind

NP enthält eine Vielzahl praktisch relevanter Probleme (wie Tourenplanung, Maschinenbelegung, Stundenplanung, Lagerhaltung, . . .)

Entscheidungsprobleme

Entscheidungsproblem

Abbildung der Form $dp\colon A^* \to BOOL$

Lösung von dp

CE-S-Operation $sol\colon A^* \to BOOL$, so dass für alle $w \in A^*$ gilt:

$$dp(w) = T \text{ g.d.w. } sol(w) \stackrel{*}{\longleftrightarrow} T$$

(aber: $dp(w) = F$ g.d.w. $sol(w) \stackrel{*}{\longleftrightarrow} T$ nicht gilt)

Definition von NP und P

$dp \in NP$, falls eine Lösung sol und ein $k \in \mathbb{N}$ existieren mit $T^{sol} \in O(n^k)$

$dp \in P$, falls zusätzlich für alle $w \in A^*$ gilt:

$$dp(w) = T \text{ und } sol(w) \overset{*}{\longleftrightarrow} t \text{ impl. } t \overset{*}{\longleftrightarrow} T$$

offensichtlich: $P \subseteq NP$

offen: $NP \subseteq P$

Beispiele

Problem aus P	Aufwand
Suchen in balancierten Bäumen	$\log n$
Suchen in Wörtern, Transponieren, Zählen, Filtern	n
Sortieren durch Mischen	$n \cdot \log n$
Sortieren durch Einsortieren, $quicksort$	n^2
Matrizenmultiplikation, Wortproblem kontextfreier Sprachen	n^3

Das $P = NP$-Problem

Besitzen (Entscheidungs-)Probleme mit einer **N**icht-deterministischen **P**olynomiellen Lösung immer auch eine deterministische **P**olynomielle Lösung?

- Eines der bekanntesten offenen Probleme der Informatik
- Nobelpreis-verdächtig

Anmerkungen zum Nichtdeterminismus

- Manches ist nichtdeterministisch

  Suchen in Bereichen, unsicheres und unvollständiges Wissen, Expertensysteme, Spiele

- Gleichwertigkeit ist nichtdeterministisch
- Manchmal bequem (und schadet nicht)
- Einschränken, wenn gewünscht und nötig

Beispiel: Little-Solitaire

solitaire

opns: $solitaire, won\colon \{0,1\}^* \to BOOL$

$move\colon \{0,1\}^* \to \{0,1\}^*$

vars: $u, v, w \in \{0,1\}^*$

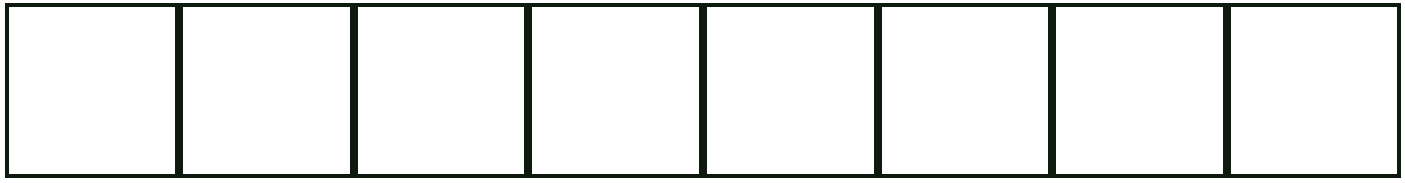
eqns: $solitaire(w) = won(move(w))$

$won(w) = count(1,w) = 1$

$move(u110v) = move(u001v)$

$move(u011v) = move(u100v)$

$move(w) = w$



Anmerkungen zu P

- Die meisten praktisch interessanten Probleme in P lassen sich in $O(n^3)$ Schritten oder schneller lösen.
- Das Laufzeitverhalten polynomieller Algorithmen lässt sich häufig weiter verbessern (Beispiel: Matrizenmultiplikation).

- Wenn jede Gleichungsanwendung in CE-S auf dem Computer polynomiellen Aufwand hat, können wir Gleichungsanwendung als konstant zählen, ohne dabei die Klasse P zu verlassen.

  Aber:

  Es kann sich bei einer CE-S-Spezifikation ein anderer Aufwand ergeben (als bei einem entsprechenden Computer-Programm), wenn der Aufwand einer Gleichungsanwendung nicht konstant ist.

- Gleichungsanwendungen haben nicht immer konstanten Aufwand.

Ein Problem in NP

- SAT (Erfüllbarkeitsproblem der Aussagenlogik)
  - Eingabe: Aussagenlogische Formel f.
  - Ausgabe: T gdw eine Belegung der Variablen in f mit 1 oder 0 existiert, so dass f gilt.
  - Lösung 1: Probiere alle 2^k Belegungen $\in O(2^k)$
  - Lösung 2: Rate richtige Belegung $\in O(k)$

Anwendungsgebiete: Entwurf und Analyse von Schaltkreisen, Model Checkers, Robotik, . . .

Aussagenlogische Formel

Sei Var eine Menge von Variablen.

Menge WFF aller aussagenlogischen Formeln

1. $v \in Var \Longrightarrow v \in WFF$
2. $S_1, S_2 \in WFF \Longrightarrow$
   $(S_1 \vee S_2), (S_1 \wedge S_2), \neg S_1 \in WFF$

Beispiel

Ein Gerät mit vier $0/1$-Schaltern befindet sich bei folgenden Schalterstellungen in einem betriebssicheren Zustand:

1. Wenn A und B gleich 0, dann C gleich 1
2. A oder C gleich 1
3. A, B, C gleich 1 oder B, C, D gleich 1 oder A, D gleich 1

$P = NP$-Problem

- Lässt sich das Erfüllbarkeitsproblem der Aussagenlogik deterministisch in polynomieller Zeit lösen?
- Entsprechende Frage für viele praktisch relevante Probleme (Maschinenbelegung, Tourenplanung, Handelsreisende, Färbung, Lagerhaltung, . . .)

---

* P: alle polynomiell lösbaren (Entscheidungs-)Probleme
NP: alle nichtdeterministisch polynomiell lösbaren (Entscheidungs-)Probleme

Anmerkungen zu NP

- Zur Lösung von Problemen aus NP muss meist eine Anzahl von$Lösungskandidaten$ durchsucht werden.
- Diese Anzahl steigt mit wachsender Eingabe exponentiell an.
- Ein nichtdeterministischer Algorithmus löst diese Probleme in polynomieller Zeit durch Raten einer richtigen Lösung.
- Die Überprüfung, ob ein gegebener $Lösungskandidat$ eine korrekte Lösung ist, kann mit einem deterministischen Algorithmus in $O(n^k)$ Schritten gelöst werden.
- Es wird allgemein vermutet, dass NP größer als P ist.

Probleme in NP

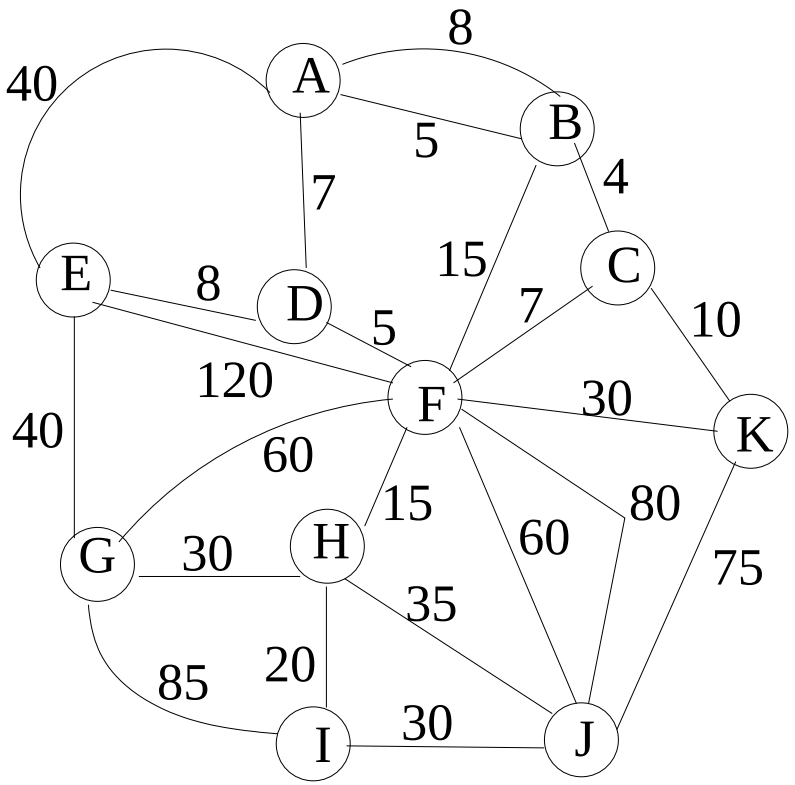
- 3SAT (Spezialfall von SAT)
  - Eingabe: Aussagenlogische Formel f der Form

$$c_1 \wedge c_2 \wedge \cdots \wedge c_n,$$

    so dass
    - c_i: Klausel der Form $L_1 \vee L_2 \vee L_3$,
    - L_j: Literal der Form x oder $\neg x$
    - x: Variable.
  - Ausgabe: T gdw eine Belegung der Variablen in f mit 1 oder 0 existiert, so dass f gilt.

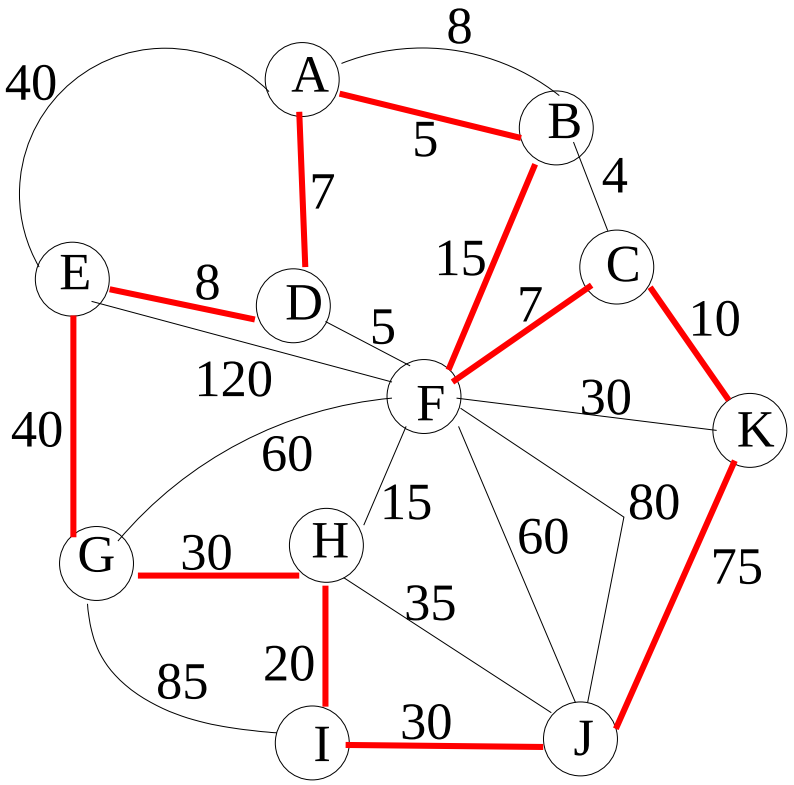
- TSP (Traveling Salesperson Problem, Entscheidungsproblemvariante)
  - Eingabe: Ungerichteter Graph G mit natürlichen Zahlen als Kantenmarkierungen und eine Zahl k.

    Beispiel für G:



- Ausgabe: T gdw G einen Rundweg besitzt, der jeden Knoten genau einmal besucht und höchstens k lang ist.

  Beispiel: T für k $= 250$



Rundweg: 247

Ordnung auf NP durch Reduktion

Eine Reduktion von

$$dp_1 \colon A_1^* \to BOOL \quad \text{auf} \quad dp_2 \colon A_2^* \to BOOL$$

ist eine CE-S-Operation $red \colon A_1^* \to A_2^*$, so dass

1. $T^{red} \in O(n^l)$ für ein $l \in \mathbb{N}$
2. für alle $w \in A_1^*$ gilt: $dp_1(w) = dp_2(red(w))$.

Schreibweise: $dp_1 \leq dp_2$

Theorem

$dp_1 \leq dp_2$ und $dp_2 \in P$ impl. $dp_1 \in P$.

NP-Vollständigkeit

Definition

$dp_0 \in NP$ heißt NP-vollständig, falls $dp \leq dp_0$ für alle $dp \in NP$.

Theorem

dp_0 NP-vollständig und $dp_0 \in P$ impl. $NP \subseteq P$.

Beweis der NP-Vollständigkeit eines Problems dp mittels Reduktion

Theorem

dp ist NP-vollständig.

Beweis:

1. Zeige, dass dp in NP liegt.
2. Wähle ein NP-vollständiges Problem und reduziere es auf dp.

Anmerkung: Ist nur möglich, wenn man bereits ein NP-vollständiges Problem hat.

NP-vollständige Probleme

Theorem (Cook und Levin)
Das Erfüllbarkeitsproblem der Aussagenlogik ist NP-vollständig.
Beweisidee: Reduziere jedes Problem aus NP auf SAT.

Theorem
3SAT ist NP-vollständig.
Beweisidee: Reduziere SAT auf 3SAT.

Theorem

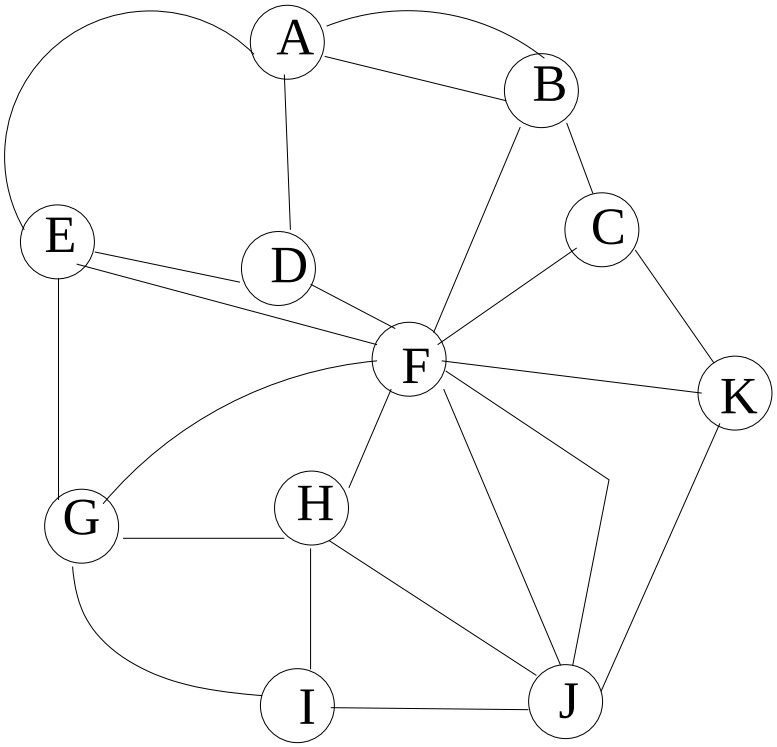
TSP ist NP-vollständig.

Beweisidee: Reduziere HC auf TSP (HC ist ein Spezialfall von TSP und NP-vollständig, wie wir noch sehen werden...)

HC

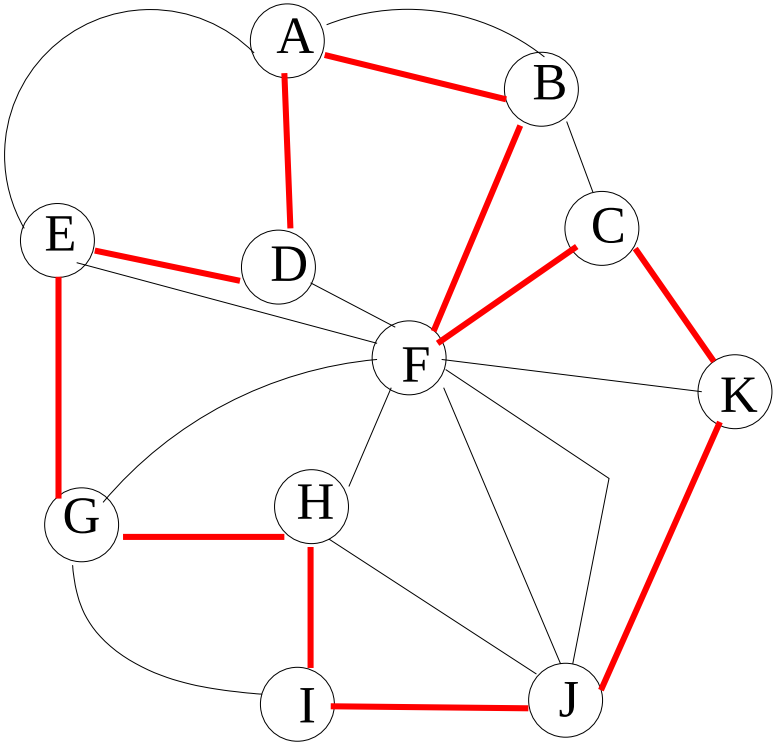
- HC (Hamiltonian Circuit Problem)
  - Eingabe: Ein ungerichteter Graph G.

    Beispiel:



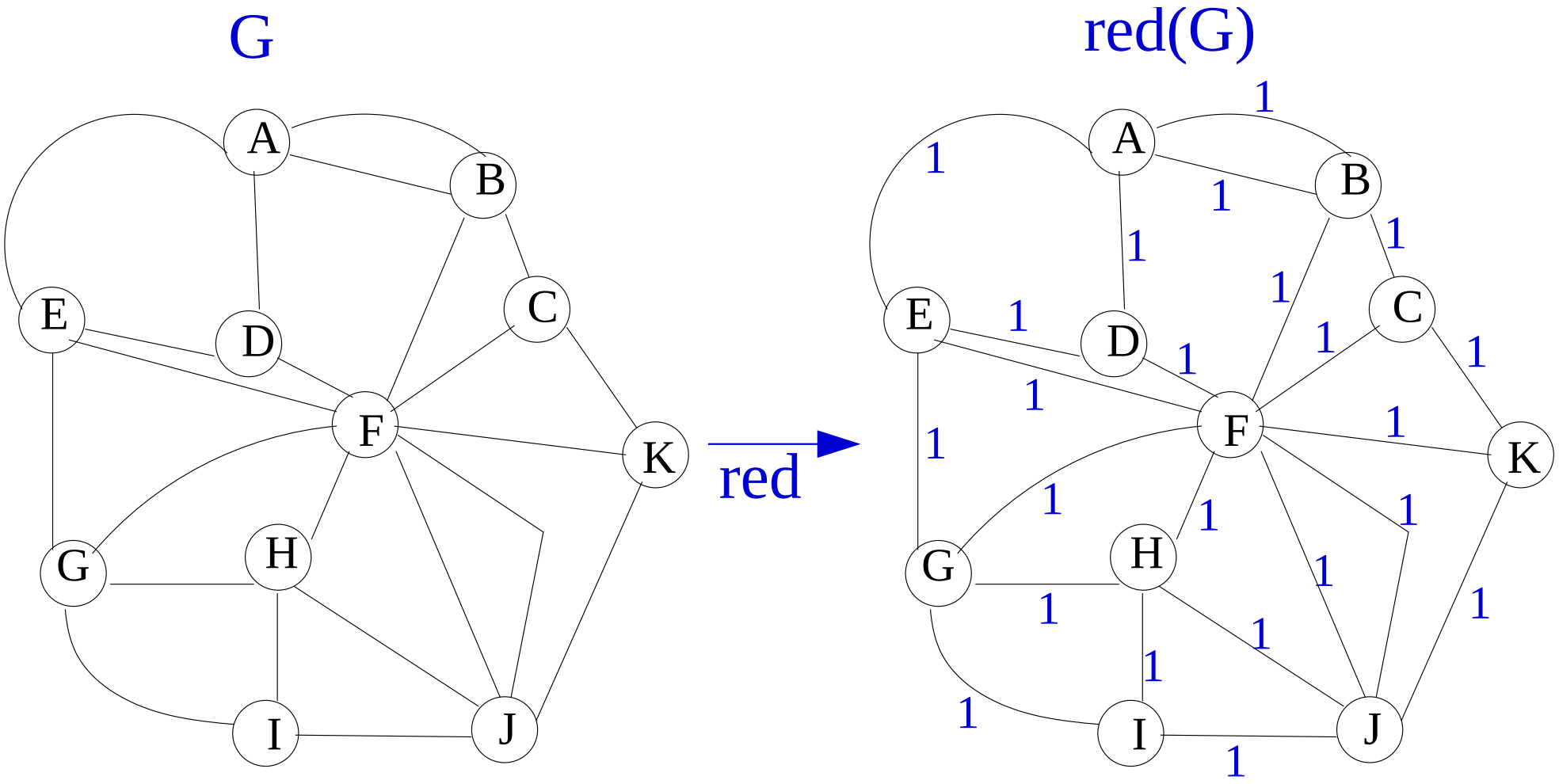
- Ausgabe: T gdw G einen Rundweg besitzt, der jeden Knoten genau einmal besucht.

  Beispiel:



Beobachtung: HC ist in NP.

Reduktion von HC auf TSP



k:= Anzahl der Knoten in G

Theorem

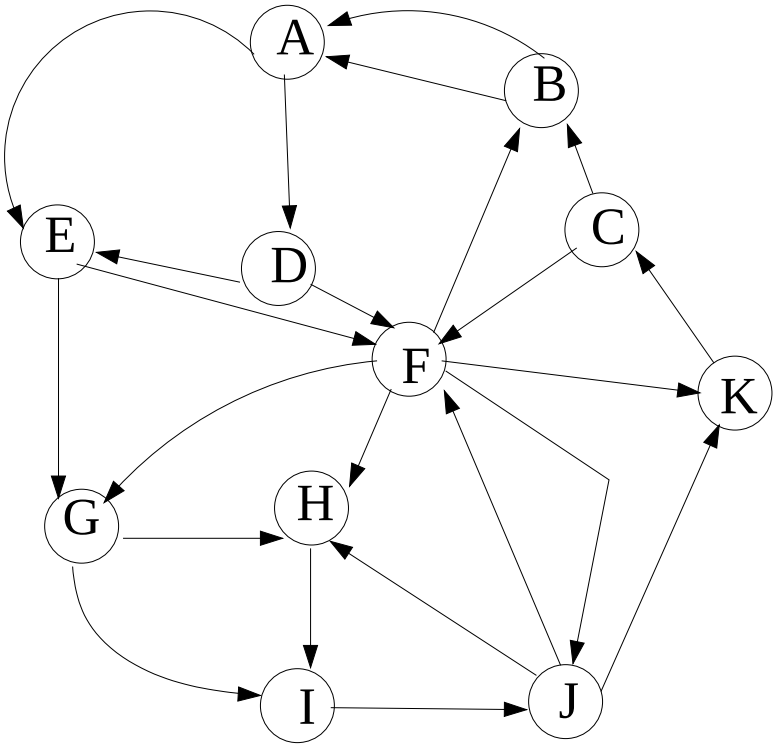
HC ist NP-vollständig.

Beweisidee: Reduziere DHC auf HC. (DHC ist HC für gerichtete Graphen und NP-vollständig, wie wir noch sehen werden...)

DHC

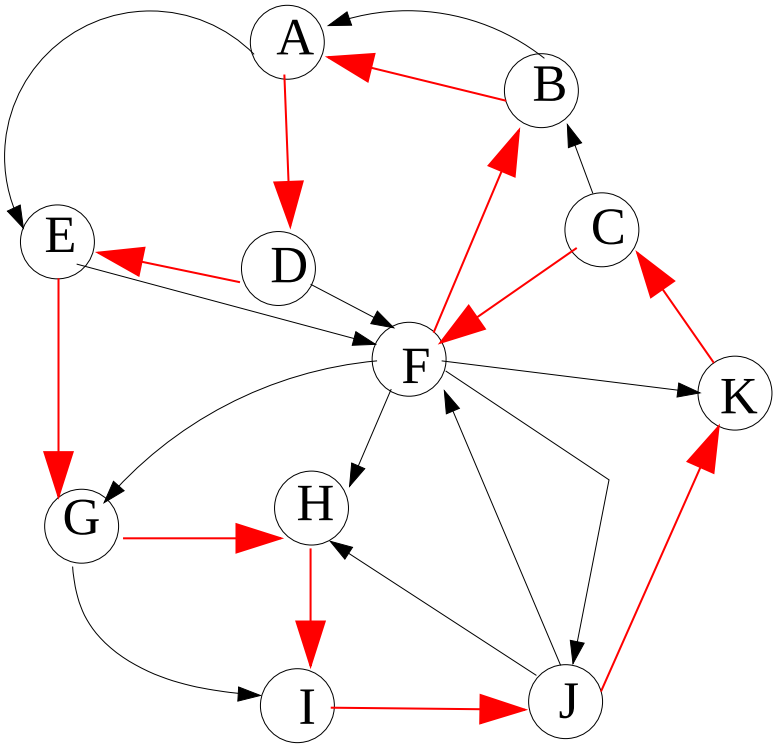
- DHC (Directed Hamiltonian Circuit Problem)
  - Eingabe: Ein gerichteter Graph G.

  Beispiel:



- Ausgabe: T gdw G einen Rundweg (in Pfeilrichtung) besitzt, der jeden Knoten genau einmal besucht.

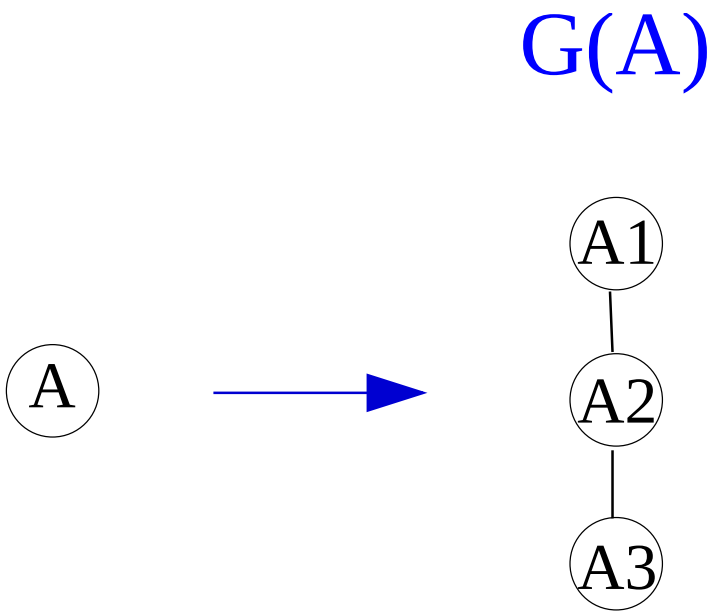
  Beispiel:



Beobachtung: DHC ist in NP.

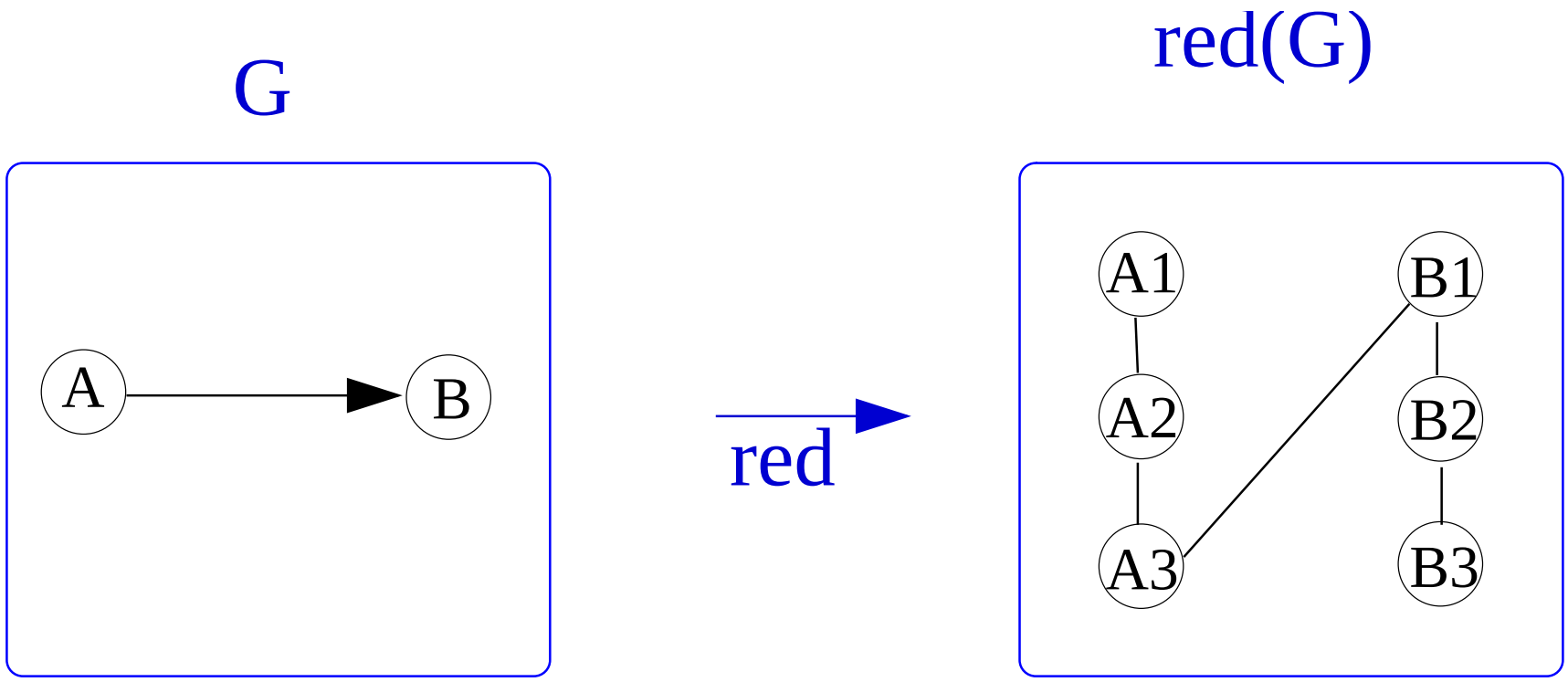
Reduktion von DHC auf HC

1. Für jeden Knoten in G konstruiere drei miteinander verbundene Knoten wie folgt:



2. Für jede Kante von A nach B ziehe eine Kante in $red(G)$ von $A3$ nach $B1$.

Skizze



Die Klasse P

Die Klasse P enthält alle (Entscheidungs-)Probleme mit einer deterministischen polynomiellen Lösung.

Beispiele

$insort$, $quicksort$, $mergesort$, Matrizenmultiplikation, Wortproblem für kontextfreie Sprachen, . . .

Die Klasse NP

Die Klasse NP enthält alle (Entscheidungs-)Probleme mit einer nichtdeterministischen polynomiellen Lösung.

Beispiele

Traveling Salesperson, Hamiltonsche Kreise, Rucksack, Stundenplanung, Maschinenbelegung, Sudoku, . . .

Theorem

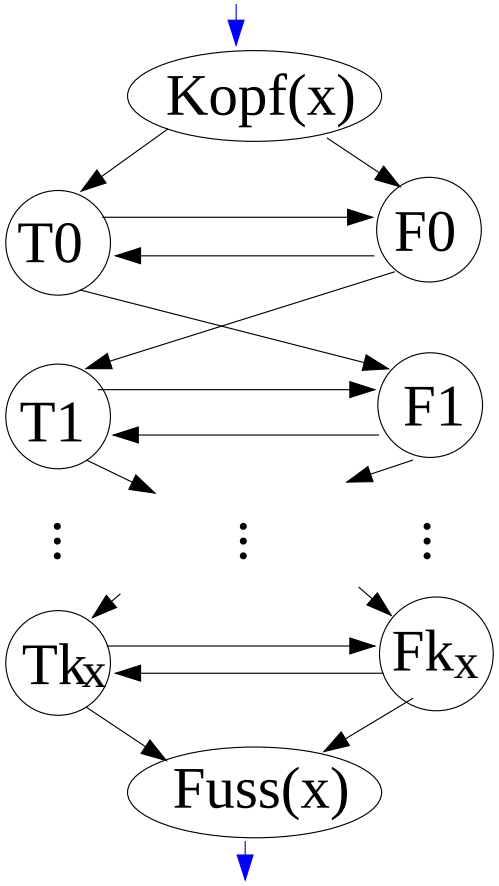
DHC ist NP-vollständig.

Beweisidee: Reduziere 3SAT auf DHC.

Reduktion von 3SAT auf DHC

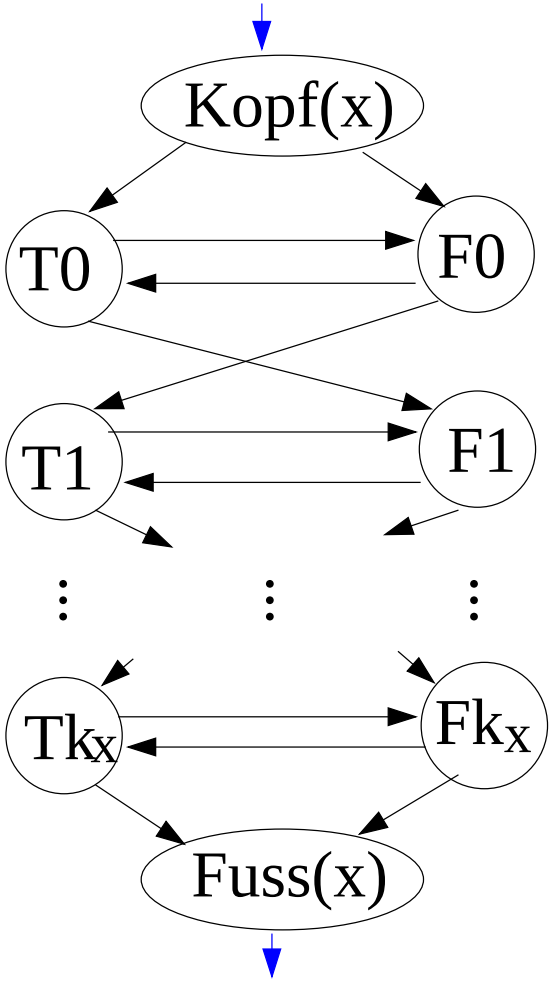
- f: Eingabeformel für 3SAT mit den Variablen $\{x_1, \ldots x_n\}$

1. Konstruiere für jede Variable x in f den Graphen $G(x)$:



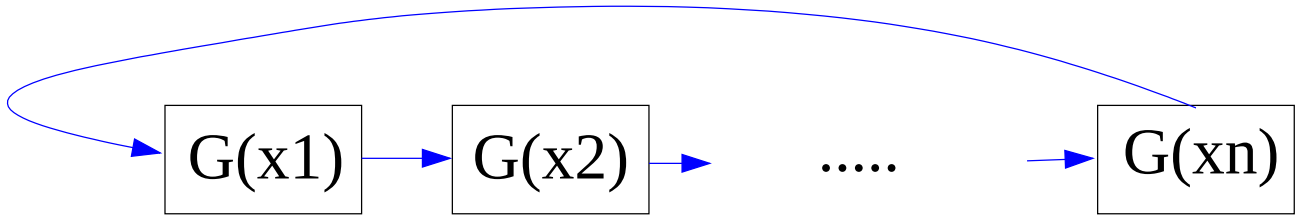
mit $k_x = max(count(x, f), count(\neg x, f))$

Idee



Durchlaufe den Graphen auf einem Weg von Kopf bis Fuß. Besuche dabei jeden Knoten genau einmal. Gehe nach dem Kopf zuerst zu T_0, falls x mit 1 belegt ist; sonst gehe zuerst zu F_0.

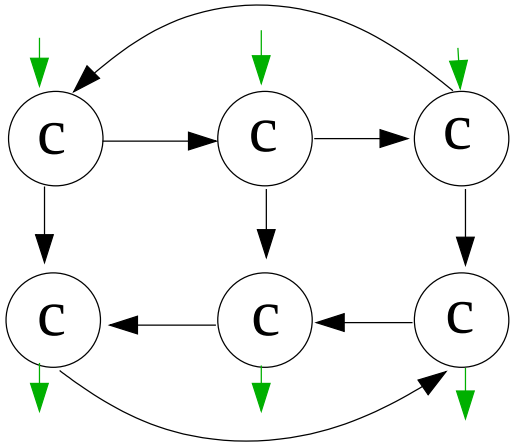
2. Verbinde diese Graphen miteinander zu G':



Beobachtung

Für jede Belegung der Variablen existiert ein Hamiltonscher Kreis in G' (d.h. ein Rundweg, der jeden Knoten genau einmal besucht).

3. Konstruiere für jede Klausel c in f den Klausel-Graphen $G(c)$:



Beobachtung

Läuft man in den i-ten oberen Knoten hinein, muss man aus dem i-ten unteren Knoten wieder herauslaufen, wenn man jeden Knoten aus $G(c)$ genau einmal besuchen will ($i = 1, 2, 3$).

4. Konstruiere $G(f)$, indem jeder Graph $G(c)$ mit dem Rest wie folgt verbunden wird:

$$\text{Sei } c = L_1 \vee L_2 \vee L_3.$$

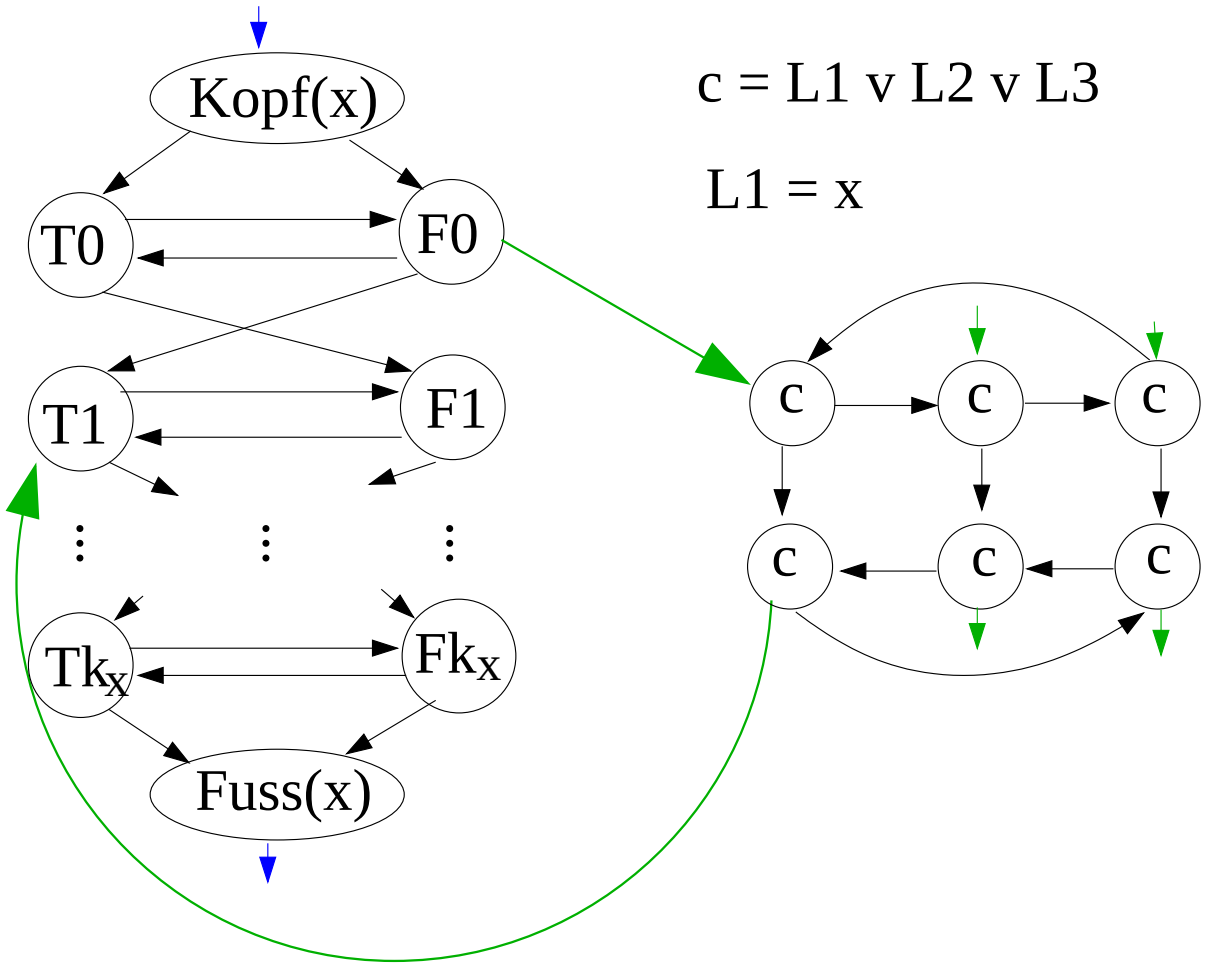
Falls $L_i = x$:

(a) Wähle ein F_j aus $G(x)$, für das gilt:

- ▷ Es gibt keine Kante zwischen F_j und einem Klausel-Graphen.
- ▷ F_j ist nicht der unterste F-Knoten in $G(x)$.

(b) Ziehe eine Kante von F_j zu dem i-ten oberen Knoten in $G(c)$ und eine Kante von dem i-ten unteren Knoten in $G(c)$ zu T_{j+1} in $G(x)$.

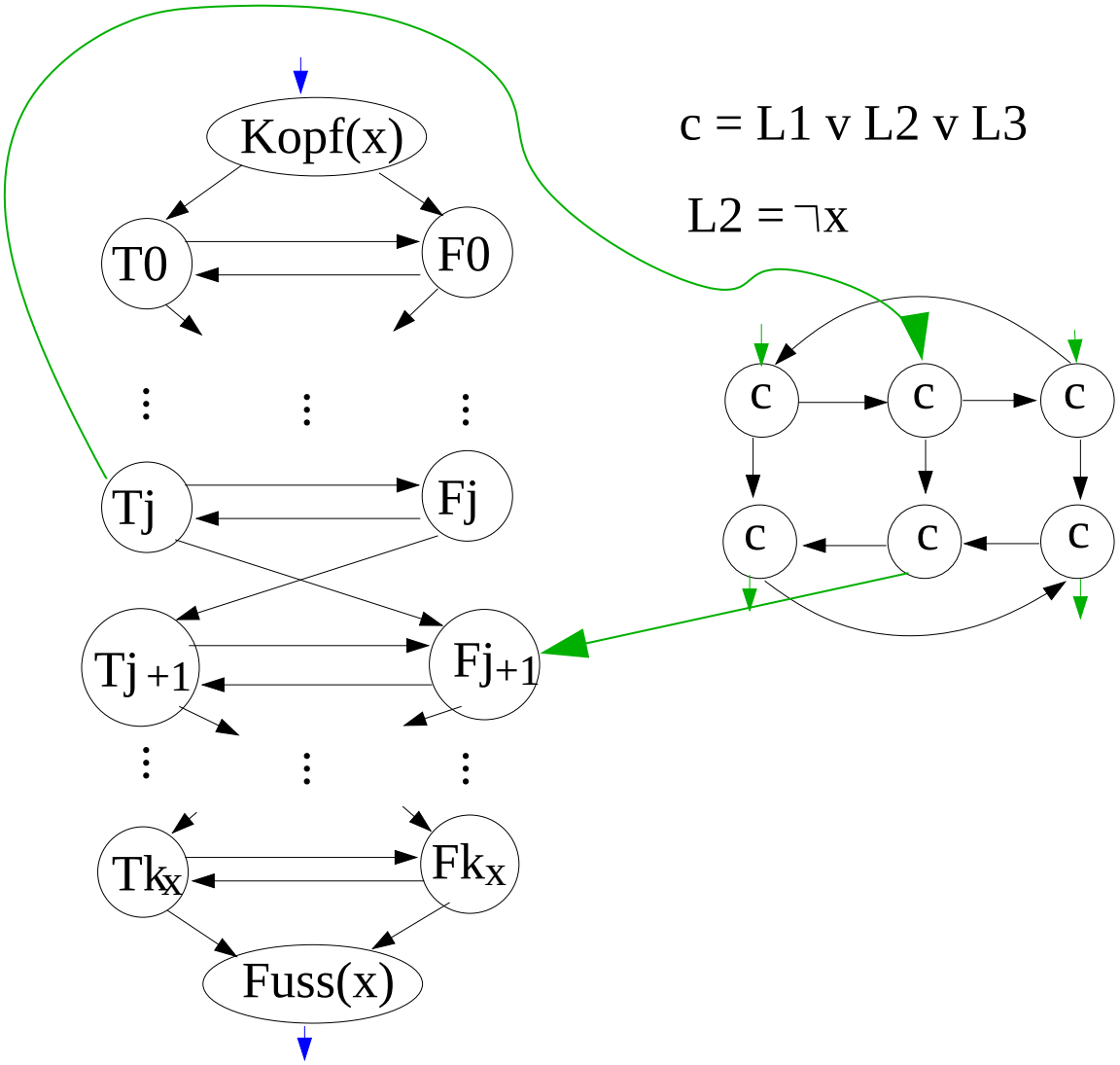
Beispiel ($L_1 = x$)



Falls $L_i = \neg x$:

1. Wähle ein T_j aus $G(x)$, für das gilt:
   - Es gibt keine Kante zwischen T_j und einem Klausel-Graphen.
   - T_j ist nicht der unterste T-Knoten in $G(x)$.
2. Ziehe eine Kante von T_j zu dem i-ten oberen Knoten in $G(c)$ und eine Kante von dem i-ten unteren Knoten in $G(c)$ zu F_{j+1} in $G(x)$.

Beispiel ($L_2 = \neg x$)



Beobachtung

Der Hamiltonsche Kreis in G' für eine Belegung der Variablen kann einen Umweg über $G(c)$ machen gdw c T ist.

Beobachtung

- DHC ist in NP.
- f ist erfüllbar gdw es einen Hamiltonschen Weg durch $G(f)$ gibt.
- Die Konstruktion von $G(f)$ hat einen polynomiellen Zeitaufwand.

Polynomieller Platzbedarf

NPSPACE:

Probleme mit nichtdeterministischen Lösungsalgorithmen, die polynomiellen Speicherplatz brauchen (im Verhältnis zur Größe der Eingabe)

PSPACE:

Analog für deterministische Lösungen

Zusammenhänge zwischen Komplexitätsklassen

$$P \subseteq NP \subseteq_{(1)} NPSPACE =_{(2)} PSPACE$$

(1) Konstanter Speicherplatz pro Rechenschritt

(2) Backtracking

Offenes Problem: $NPSPACE \subseteq NP$

Wie weit reicht P und was kommt dahinter?

$$\boxed{\boxed{\boxed{P} =^{?} NP} =^{?} \begin{matrix} PSPACE = \\ NPSPACE \end{matrix} \neq EXP} \quad \ldots \text{ Berechenbares}$$

- NP: Erfüllbarkeitsproblem, TSP u.v.a.m.
- $PSPACE$: Wortproblem monotoner Sprachen
- EXP: Drachenkurve, hoffnungslos für große Eingaben
- Berechenbares: Interpreter für CE-S u. ä.; Aufwand oft nicht definiert (Berechnung unendlich)
- Unberechenbares: Halteproblem u.v.a.m.