Globální metody

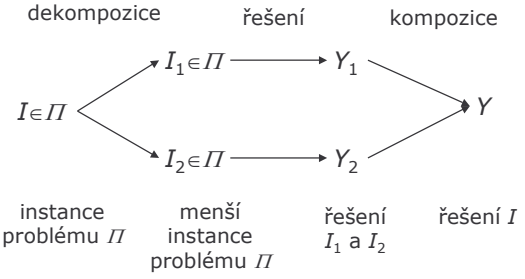
Terminologie
aneb co všechno je globální optimalizace

- techniky, které hledají globální optimum za přítomnosti lokálních optim
  - už jich známe spoustu
- techniky, které rovnoměrně prohledávají stavový prostor
  - náhodné starty lokálních metod
  - diverzifikace v lokálních metodách
  - atd.
- metody, které nejsou založeny na pojmu stavového prostoru

Globální a lokální metody

- lokální metody:
  - stav, stavový prostor
  - aktuální stav
  - okolí, prohledávání okolí
- globální metody
  - instance
  - dekompozice instance na podinstance
  - kompozice řešení podinstancí
  - rekurze
  - řešení triviálních instancí hrubou silou

Dekompozice a kompozice



Příklad: dynamické programování problému batohu

I=(*W, C, M*) ---------- instance problému batohu
X ---------- řešení problému batohu
cost (*X*) ---------- cena řešení
X=*KNAP*(*W, C, M*)----- procedura, řešící instanci *I*

exaktní metoda

triviální případ hrubou silou

```
function KNAP (W, C, M)
    if isTrivial (W, C, M)
       return trivialKNAP (W, C, M);
    X0 = KNAP (W-{w_n}, C-{c_n}, M);
    X1 = KNAP (W-{w_n}, C-{c_n}, M-w_n);
    if cost(X1)+c_n > cost(X0) return X1.1;
    return X0.0;

function isTrivial (W, C, M)
    return (isEmpty (W) or M=0 or M<0);
```

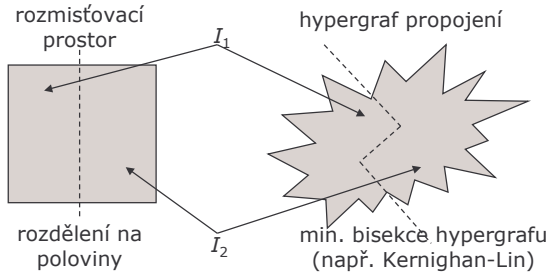
I_1 dekompozice I_2

kompozice

Příklad: řešení problému diskrétního rozmístění

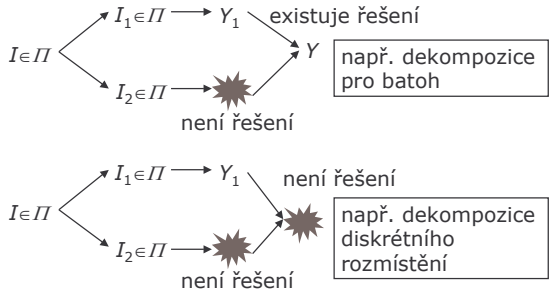
- Dáno:
  - množina *n* modulů $K=\{k_1, \ldots, k_n\}$
  - množina *m* pozic $P=\{p_1, \ldots, p_m\}$, $m \geq n$
  - propojení modulů jako hypergraf *G* (*K*, *N*), kde *N* je množina spojů *n*
  - cenová funkce *W* (*R, n*), která pro každé přiřazení *R*: $K \rightarrow P$ odhadne cenu realizace spoje *n*.
- Nalézt:
  - prosté přiřazení *R*: $K \rightarrow P$ (rozmístění), které minimalizuje součet ohodnocení *W* (*R, n*) přes všechny spoje.

Příklad: řešení problému diskrétního rozmístění

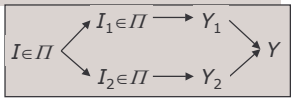


kompozice: sjednocení množin

Vlastnosti dekompozice

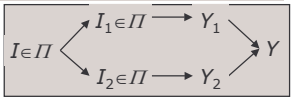


Přesná dekompozice



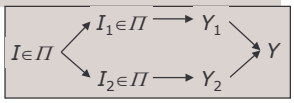
přesná dekompozice (proper decomposition):
čistá dekompozice taková, že každé optimální řešení Y se dá složit z nějakých optimálních řešení Y_1, Y_2
(ze všech optimálních Y_1, Y_2 se dají složit všechny optimální Y)

Čistá dekompozice



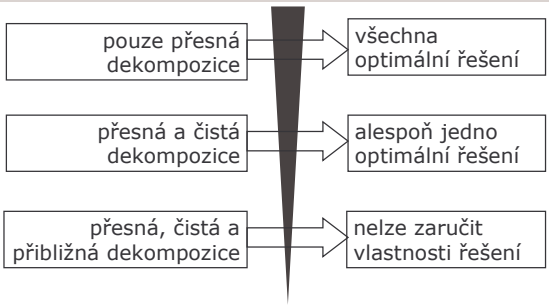
čistá dekompozice (pure decomposition):
přibližná dekompozice taková, že
- Y_1, Y_2 optimální řešení I_1, $I_2 \Rightarrow Y$ je optimální řešení I
- Y_1, Y_2 neexistují $\Rightarrow Y$ neexistuje

Přibližná dekompozice



přibližná dekompozice (approximate decomposition):
- Y_1, Y_2 optimální řešení I_1, $I_2 \Rightarrow Y$ je řešení I
- Y_1, Y_2 neexistují $\Rightarrow$ nevíme nic

Vlastnosti globálních metod



Redukce

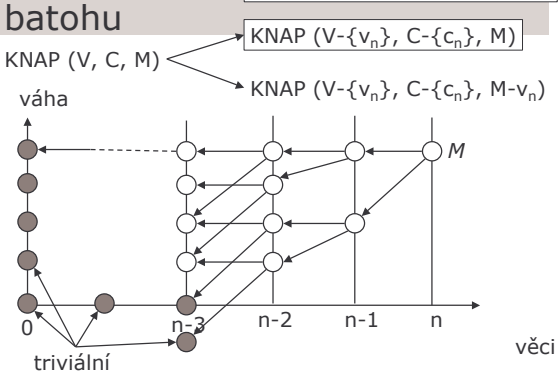
- Dekompozice taková, že jedna z dekomponovaných instancí je triviální: redukce
- přesná redukce
- čistá redukce
- přibližná redukce

Dynamické programování

- čistá dekompozice
- dekomponované instance se dají charakterizovat malým objemem hodnot
- řešení dekomponovaných instancí se dají indexovat těmito hodnotami
- dekomponované instance se dají rozdělit do disjunktních tříd (stupňů) tak, že k výpočtu instancí jednoho stupně je třeba jen instancí právě jednoho jiného stupně

Problém batohu

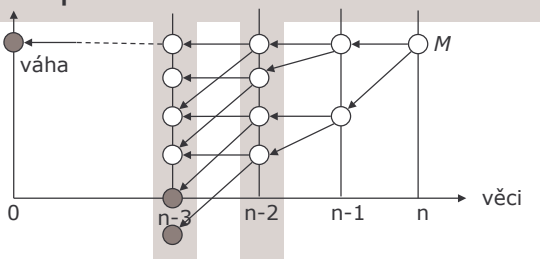
charakteristika podinstance: n, M



Formulace dynamického programování

- Rekurzivní:
  - vyjít ze zadané instance
  - stanovit, které podinstance je třeba řešit, až se dosáhne triviálních instancí
  - jasná souvislost s globálními metodami
  - ale v praxi se nepoužívá
- Dopředná
  - vyjít ze zadané instance
  - spočítat všechny složitější podinstance, až se dosáhne zadané instance
  - a tak se to dělá

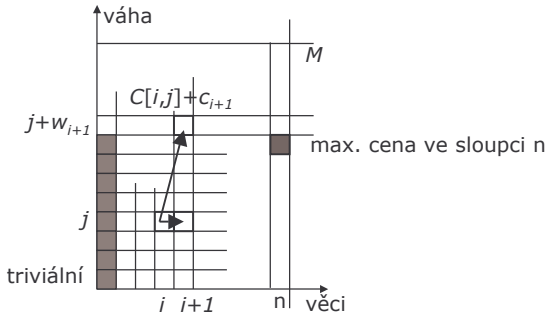
Stupně



stupeň n-3

stupeň n-2, k výpočtu instancí stačí instance stupně n-3, není nutno skladovat více

Dopředný výpočet



Použití dynamického programování

- Formulovat řešení jako rekurzivní algoritmus
- Stanovit minimální charakterizaci podinstancí a organizaci paměti mezihodnot
- Tak ukázat, že podinstancí je polynomiálně mnoho
- Stanovit stupně a pořadí řešení tak, aby všechny mezihodnoty byly k dispozici

Co může být výhodné řešit dynamickým programováním

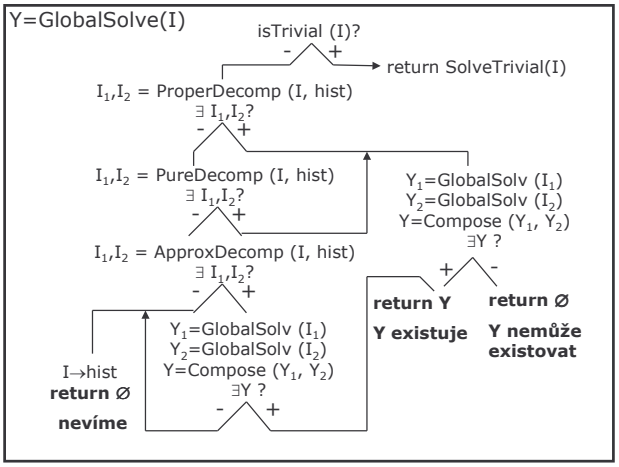
- Problémy, které mají omezený počet mezihodnot
  - $n!$ permutací měst TSP, ale
  - $n(n\text{-}1)/2$ souvislých podřetězů řetězu, určených počátkem a koncem
  - $n(n\text{-}1)/2$ podstromů binárního vyhledávacího stromu, určených minimálním a maximálním klíčem
- Problémy s přirozeným uspořádáním prvků

Rozděl a panuj

- Algoritmy založené na přibližné dekompozici
- Opakované řešení dekomponovaných instancí je řídké
- Zvláštní případ: je potřeba řešit jen jednu z dekomponovaných instancí - zmenši a panuj (např. binární hledání), někdy jen použití přibližné redukce

Globální metody, složené z různých dekompozic

- Princip: rekurzivní procedura
- V každé úrovni se použije nejlepší možná dekompozice (přesná, čistá, přibližná)
- Možné výsledky:
  - nalezeno řešení
  - řešení neexistuje
  - nevíme nic
- Tam, kde je možno provést více dekompozic stejného druhu, udržujeme seznam těch, které nevedly k výsledku



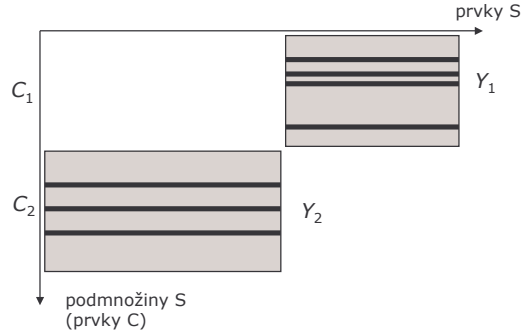
Problém pokrytí

- Dáno: $I=(C,S)$ kolekce C podmnožin konečné množiny S.
- Zkonstruovat:
  - podmnožinu $Y \subseteq C$ takovou, že každý prvek S patří do alespoň jedné podmnožiny z Y, a dále $|Y|$ = min. (unátní pokrytí)
  - podmnožinu $Y \subseteq C$ takovou, že každý prvek S patří do právě jedné podmnožiny z Y, a dále $|Y|$ = min. (binátní pokrytí)

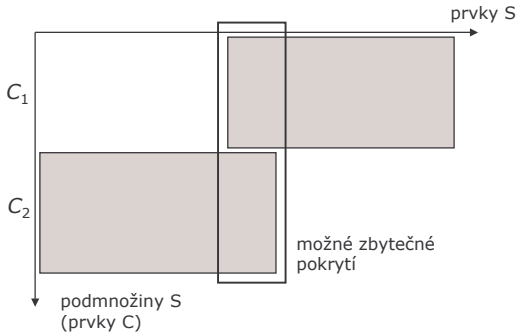
Dekompozice

- $C = C_1 \cup C_2$, nikoliv nutně disjunktní
- dekompozice C indukuje instance
  $I_1=(C_1, S_1)$
  $I_2=(C_2, S_2)$
- $Y_1 \cap C_2 = Y_2 \cap C_1 \Rightarrow Y=Y_1 \cup Y_2$ je řešení I.
- $Y_1 \cap C_2 = Y_2 \cap C_1$ a Y_1, Y_2 jsou jediná min. řešení $\Rightarrow$ dekompozice je přibližná.
- $Y_1 \cap Y_2 = C_2 \cap C_1$ a Y_1, Y_2 jsou jediná min. řešení $\Rightarrow$ dekompozice je přesná.

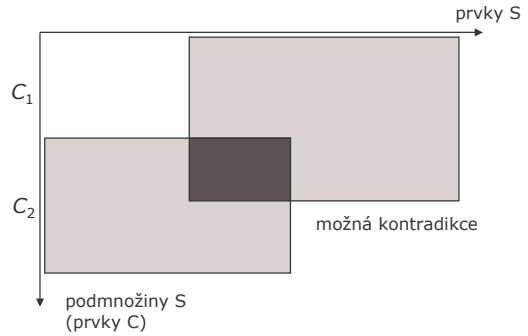
Dekompozice
(zřejmý případ)



Dekompozice
(méně zřejmý případ)



Dekompozice
(obecný případ)



Redukce:
sloupcová dominance

- prvek $s \in S$ je obsažen pouze v množině $c \in C \Rightarrow c \in Y$
  - dekompozice na $I_1=(\{c\},\{s\})$ a zbytek I_2 instance
  - I_1 je triviální, $Y_1=\{c\}$
  - c musí být v každém, tedy i optimálním řešení
  - všechna opt. řešení I dostaneme složením všech opt. řešení I2 a $\{c\}$
- sloupcová dominance je přesná redukce

Redukce:
řádková dominance

- množina $c_1 \in C$ je podmnožinou $c_2 \in C \Rightarrow c_1 \notin Y$
  1. Nechť Y_{opt} je optimální řešení, nechť $c_2 \in Y_{opt}$. Pak $c_1 \notin Y_{opt}$, neboť je zbytečná.
  2. Nechť Y_{opt} je optimální řešení, nechť $c_2 \notin Y_{opt}$. Pak je buď
     - $c_1 \notin Y_{opt}$
     - $c_1 \in Y_{opt}$; pak se dá nahradit množinou c_2 (optimalizační kritérium $|C|$ se nezmění); nedostaneme všechna optimální řešení
- řádková dominance je čistá redukce