IV126 Umělá inteligence II

Hana Rudová

Fakulta informatiky, Masarykova univerzita

8. ledna 2021

Online výuka

Web předmětu: `http://www.fi.muni.cz/~hanka/ai`

- materiály průběžně aktualizovány na webu předmětu
- zveřejnění cca týden před odpovídající přednáškou

Průsvitky (slides)

Videa ke každé přednášce

- ze starších běhů předmětu nebo
- nově připravené s pomocí OBS Studio

Videokonference v době přednášky

- Zoom (link e-mailem)
- sada otázek předem + případné další dotazy
- odpovědi studentů nebo demonstračně
- bonusové body za aktivní účast pro vylepšení hodnocení
  - až 24 bodů ke 100 běžným

Hodnocení

Online závěrečná písemná práce: 80 bodů

- vyžadována nadpoloviční znalost látky (více než 40 bodů)
- pravděpodobně (především vzhledem k počtu zapsaných)

Domácí úkoly: 20 bodů

- 2 domácí úkoly po 10 bodech
- minimální počet bodů za domácí úkoly: 8 bodů
- úkol zadán na přednášce, řešení do přednášky za 2 týdny
- postup řešení musí být součástí odevzdaného úkolu

Bonusové body: až 2 body za aktivní účast na 1 videokonferenci

- 1 bod: reakce na více jednoduchých dotazů nebo dotazy studentky/a na vyjasnění látky, reakce na jeden složitější dotaz
- 2 body: větší interakce

Hodnocení: A 90 a více, B 80-89, C 70-79, D 60-69, E 50-59

Obsah přednášky

Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, fourth edition. Prentice Hall, 2020. `http://aima.cs.berkeley.edu/`

- Prohledávání a heuristiky: koncepty, práce s jedním řešením, s populací řešení
- Plánování: Klasické plánování, reprezentace problému. Plánování se stavovým prostorem, plánování s prostorem plánů
- Neurčitost: Bayesovské sítě, exaktní a aproximační odvozování. Čas a neurčitost. Užitek a rozvhodování. Sekvenční rozhodovací problémy, Markovské rozhodovací procesy.
- Robotika: Robot a jeho hardware. Vnímání robota, lokalizace a mapování Plánování pohybu robota, plánování s neurčitostí

PB016 Umělá inteligence I

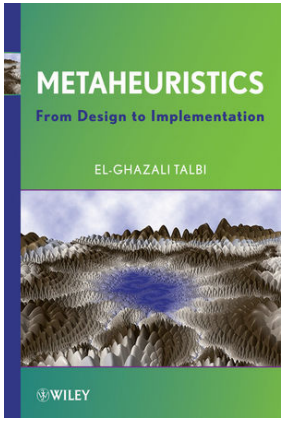
- IV126 volně navazuje na PB016, absolvování PB106 není podmínkou

Metaheuristiky: úvod

8. ledna 2021

1. Přehled optimalizačních metod
2. Hlavní koncepty
3. Použití a implementace algoritmů

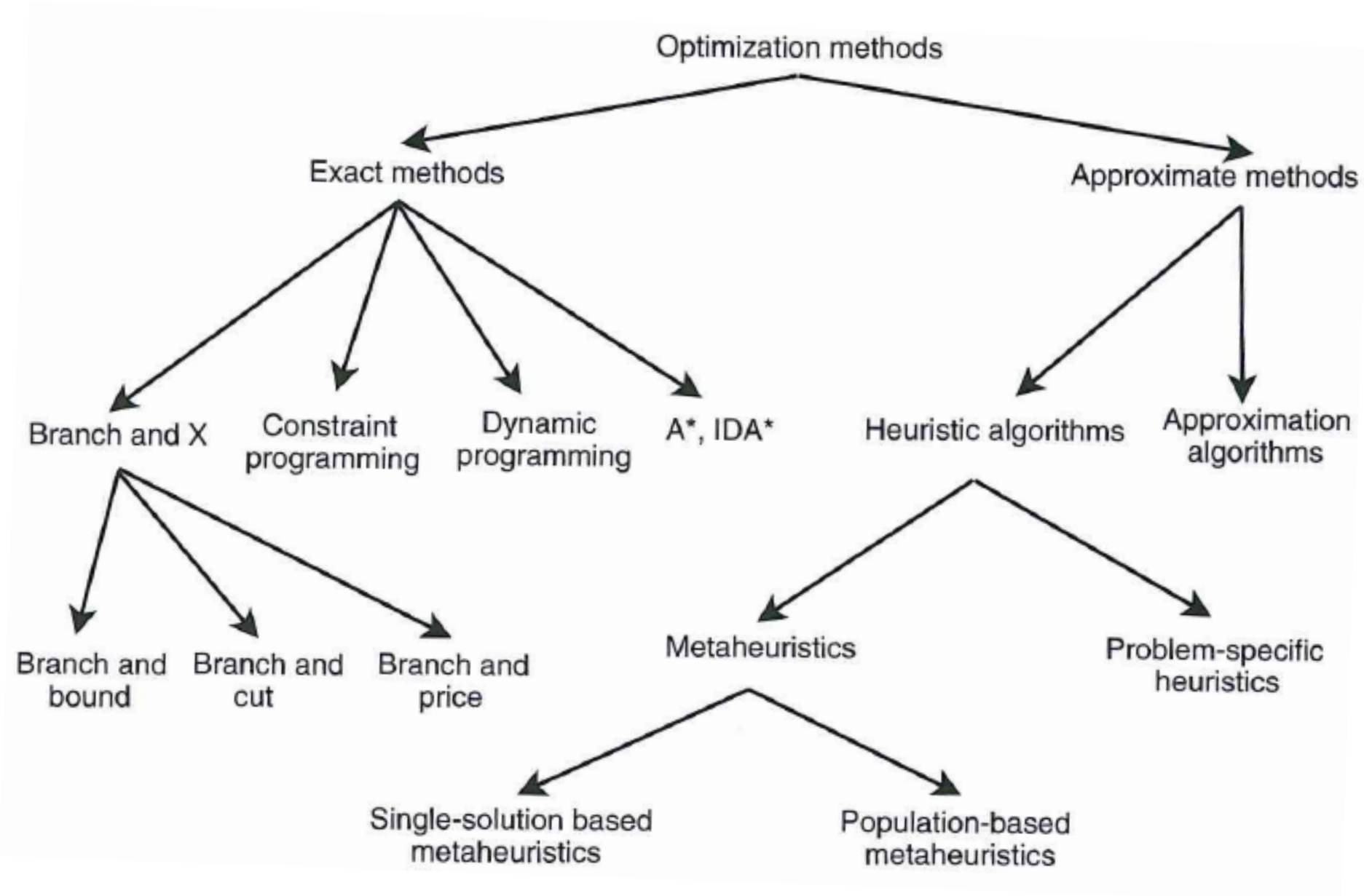
Zdroj: *El-Ghazali Talbi, Metaheuristics: From Design to Implementation. Wiley, 2009.*
http://www.lifl.fr/~talbi/metaheuristic/



Literatura

- Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, fourth edition. Prentice Hall, 2020.
  `http://aima.cs.berkeley.edu/`
- El-Ghazali Talbi, *Metaheuristics: From Design to Implementation*. Wiley, 2009. `http://www.lifl.fr/~talbi/metaheuristic/`
- FI:PA184 Heuristic Methods for Search and Optimization
  jaro 2010, `https://is.muni.cz/auth/predmet/fi/jaro2010/PA184`
- Holger H. Hoos und Thomas Stützle, *Stochastic Local Search*. Morgan Kaufmann Publishers, 2005.
  `http://iridia.ulb.ac.be/~stuetzle/Teaching/HO14/`
- Sörensen K., Sevaux M., Glover F., *A History of Metaheuristics*. In Handbook of Heuristics. Springer, Cham, 2018.
  `https://arxiv.org/abs/1704.00853`

Optimalizační metody



Přehled optimalizačních metod

Exaktní metody

- výpočet (globálně) optimálního řešení, garance optimality
- nevhodné pro velké problémy

Aproximační metody → aproximační algoritmy

- garance na mez kvality vypočítaného řešení
- ϵ-aproximace
  - př. problém plnění košů (bin packing) – stejné koše, různé předměty
    First Fit (FF) $\frac{17}{10}opt + 1$ (opt: počet košů v optim.řešení)
    First Fit Decreasing (FFD) $\frac{11}{9}opt + 1$

Aproximační metody → heuristické algoritmy

- ze staré řečtiny slovo „heuriskein" znamená:
  umění objevovat nové strategie/pravidla pro řešení problémů
- cíl generovat pro praktické použití „vysoce kvalitní" řešení
  v rozumném čase
- žádná garance na kvalitu (globálního) optima

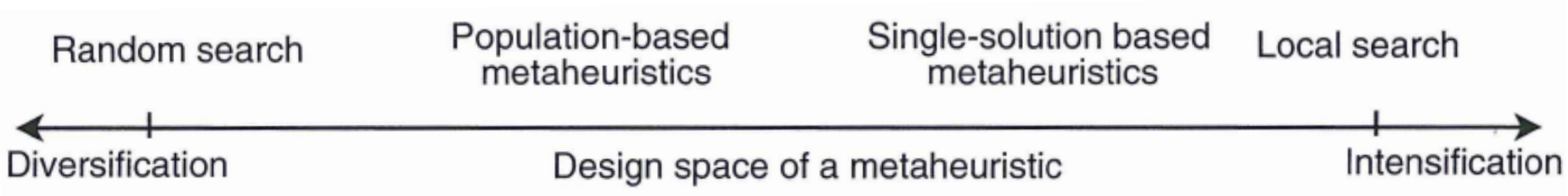
Heuristické algoritmy → metaheuristiky

Metaheuristiky

- předpona meta opět z řečtiny: metodologie vyšší/obecnější úrovně
- představeny F. Gloverem v roce 1986
- metaheuristika: metodologie (templát/šablona) vyšší obecnější úrovně, která může být použita jako řídící strategie pro návrh základní heuristiky pro řešení specifických optimalizačních problémů
  - př. rozvrhování předmětů: simulované žíhání a výměna dvou předmětů
- reprezentují obecný přístup aplikovatelný na různé problémy

Při návrhu metaheuristik nutno aplikovat dva protichůdné principy:

- průzkum (exploration), diversifikace vs.
  zužitkování (exploitation), intensifikace



Klasifikace metaheuristik

(Ne)inspirované přírodou

- přírodní procesy: evoluční algoritmy
- sociální vědy: mravenci, včelstva, roje
- fyzika: simulované žíhání

(Ne)používající paměť

- tabu prohledávání (tabu seznam jako paměť) vs. simulované žíhání

Deterministické vs. stochastické

- deterministická rozhodnutí (tabu prohledávání)
- použití náhodnosti (simulované žíhání)

Založené na populaci vs. jednom řešení

- genetické algoritmy vs. simulované žíhání

Iterativní vs. hladové

- iterativní: začneme s úplným řešením (nebo jejich populací) a provádíme jeho/jejich transformace
- hladové: začneme z prázdného řešení a v každém kroku přiřadíme jednu rozhodovací proměnnou, dokud nenalezneme úplné řešení
- většina metaheuristik iterativní

Lokální prohledávání (local search, hill climbing, horolezecký algoritmus)

- Nejstarší a nejjednodušší metaheuristika
- Nahrazuje aktuální řešení zlepšujícím řešením

$s := s_0$; *(generuj iniciální řešení s_0)*
while není splněna podmínka ukončení **do**
    generuj($N(s)$); *(generování sousedů z okolí)*
    if neexistuje lepší soused **konec**;
    $s := s'$; *(vyber lepšího souseda $s' \in N(s)$)*
end while
výstup: finální nalezené řešení (lokální optimum)

Hlavní koncepty pro metaheuristiky

Reprezentace (kódování)

- jak reprezentovat řešení/přiřazení
- př. posloupnost: úlohy na jednom stroji naplánovány v pořadí (1,2,3)

Účelová funkce (objektivní funkce, případně optimalizační kritérium)

- př. chceme minimalizovat čas dokončení poslední úlohy

Manipulace s omezeními (constraint handling)

- omezení/podmínky musí být splněny, aby bylo řešení validní, tj. konzistentní
- konzistentní vs. nekonzistentní řešení
- př. úlohy naplánované na jednom stroji se nesmí překrývat

Reprezentace

Nutné charakteristiky reprezentace problému

- úplnost
  - lze reprezentovat všechna řešení
- dosažitelnost
  - ve stavovém prostoru musí existovat cesta mezi každými dvěma řešeními
  - každé (zejména optimální) řešení je dosažitelné
- efektivita
  - s reprezentací lze snadno manipulovat prohledávacími operátory
  - minimalizace časové a prostorové složitosti operátorů

Typy reprezentací

- lineární reprezentace
  - řetězce symbolů dané abecedy
  - binární kódování, diskrétní kódování, permutace, vektor reálných čísel
- nelineární reprezentace
  - založené často na grafových strukturách, zejména použité stromy

Binární kódování

- S každou rozhodovací proměnnou je spojena binární hodnota
- Řešení reprezentováno jako vektor bitových hodnot

Příklad: problém batohu

- Máme n předmětů zadané velikosti a ceny. Vyberte do batohu velikosti m předměty tak, aby se do něj vešly a jejich cena byla maximální.
- Řešení reprezentuje vektor $s = (s_1, s_2, \ldots, s_n)$

$$s_i = \begin{cases} 1 & \text{pokud je předmět } i \text{ v batohu} \\ 0 & \text{jinak} \end{cases}$$

Další příklady:

- SAT problém
- problémy celočíselného programování s binárními proměnnými

Diskrétní kódování

- Zobecnění binárního kódování
- Řešení reprezentováno jako vektor diskrétních hodnot, tj. proměnné nabývají hodnoty z n-ární abecedy

- Pro problémy, kde proměnné nabývají hodnot z konečné domény, např. kombinatorické problémy
- Řada reálných optimalizačních problémů (např. alokace zdrojů) může být redukovaná na problém přiřazení
- Problém přiřazení: Máme zadánu množinu n úloh, kterým má být přiřazeno m agentů tak, abychom maximalizovali celkový zisk. Úloha může být přiřazena libovolnému agentovi.
- Řešení reprezentuje vektor s velikosti n, kde s_i reprezentuje agenta přiřazeného úloze i

$$s_i = j \qquad \text{pokud je agent } j \text{ přiřazen úloze } i$$

Reprezentace permutací

Permutační problémy

- seřazení (sequencing), směrování, plánování
- každý element problému se musí v reprezentací objevit pouze jednou

Příklad: problém obchodního cestujícího (traveling salesman problem TSP)

- n měst, která musí obchodní cestující projít
- řešení reprezentuje permutace měst udávající pořadí jejich procházení

Příklad: plánování úloh bez překrývání na jeden stroj

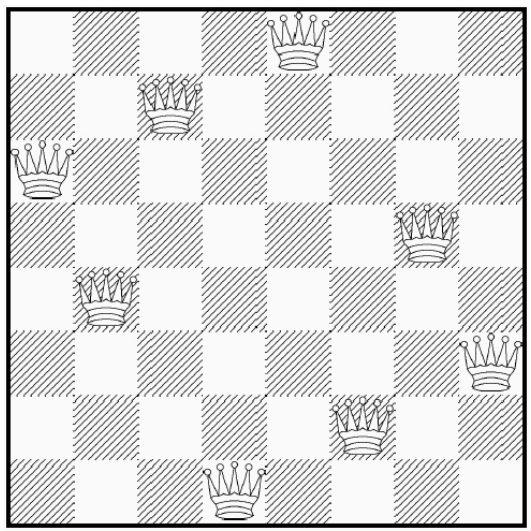
- řešení reprezentuje permutace úloh udávající pořádí prováděných úloh

Redukce stavového prostoru vhodnou reprezentací

Příklad: umístit 8 královen na šachovnici tak, aby na sebe neútočily

Souřadnice v prostoru

- vektor $(s_1, \ldots, s_8)$ pozic na šachovnici, kde $s_i = (x_i, y_i)$
- počet možností 64^8, tj. přes 4 miliardy



Jedna královna ve sloupci

- vektor $(y_1, \ldots, y_8)$ umístění královny i v i-tém sloupci
- zakazuje více královen v jednom slouci
- počet možností 8^8, tj. přes 16 miliónů

Permutace

- pokud zakážeme dvě královny ve stejném sloupci a řádku, pak se reprezentace redukuje na permutaci královen
- královna na sloupec, pro každou královnu různý řádek $\Rightarrow$ permutace
- počet možností 8!, tj. 40 320 možností

Účelová funkce

Soběstačná účelová funkce

- účelová funkce problému lze rovnou použít při optimalizaci
- ideální situace
- soběstačná funkce: TSP $min \sum dist(x, y)$
- ne-soběstačná funkce: SAT splnitelná formule (true/false)

Směrující účelová funkce

- směrující funkce lze použít při optimalizaci
  - soběstačná funkce lze přímo použít
- někdy je nutné účelovou funkci transformovat kvůli lepší konvergenci metaheuristiky
  - např. ne-soběstačné funkce je nutné transformovat
- nová účelová funkce bude směřovat k efektivnějšímu prohledávání
- k-SAT: konjunkce m klauzulí (disjunkcí) nad k proměnnými

  $(x_1 \vee x_2) \wedge (\overline{x_1} \vee x_2) \wedge (x_1)$ pro (x_1, x_2) porovnej (0,0) a (1,0)

  nová účelová funkce: počet splněných klauzulí, cíl: maximalizace

Další typy účelových funkcí

Relativní a kompetitivní účelové funkce

- pokud nelze přiřadit funkcí absolutní hodnotu ke všem řešením
- příklad: teorie her, strategie $A, B, C : A < B, B < C, C < A$

Meta-modelování

- při extrémně výpočetně náročné účelové funkci je navržen přibližný model a přibližná účelová funkce
- př. návrh telekomunikační sítě

A další ...

Manipulace s omezeními

Zamítací strategie

- uchováváme pouze konzistentní řešení

Penalizující strategie

- původní účelová funkce rozšířena o penalizaci nekonzistentních řešení
- uvažovány váhy nesplněných omezení
- řádově vyšší váha za nesplněná omezení ve srovnání s hodnotou účelové funkce

Opravné strategie

- opravují nekonzistentní řešení
- aplikovány opravné strategie na vygenerované nekonzistentní přiřazení, které tedy opravíme, a pokračujeme s konzistentním řešením

A další ...

Ladění parametrů a analýza výkonosti

Při návrhu algoritmů je důležité ladění parametrů algoritmů

- metaheuristiky mají mnoho parametrů
- optimální ladění záleží na problému a instanci

Offline inicializace parametrů

- nastavíme parametry a spustíme algoritmus, příklady:
  - nastavování jednoho parametru za druhým (neoptimální)
  - *Design of Experiments (DoE)*: k faktorů/parametrů
    s n možnými úrovněmi/hodnotami: n^k experimentů (!)
    Přírodovědecká fakulta: FX003 Plánování a vyhodnocování experimentu
  - Improving your statistical inferences
    `https://www.coursera.org/learn/statistical-inferences/`
  - strojové účení

Online inicializace parametrů

- (paralelní) restartování algoritmu s různými parametry

Velkou roli hraje experimentální analýza algoritmů

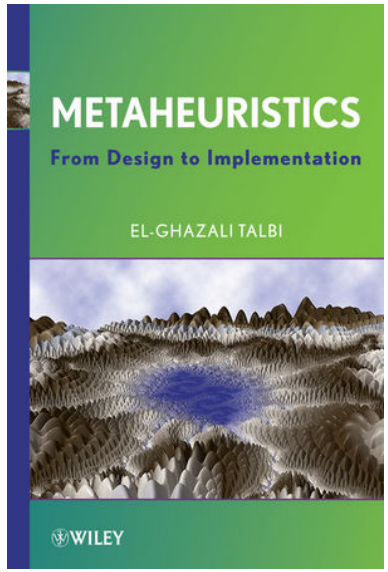
Software

- Metaheuristic optimization frameworks: a survey and benchmarking, Soft Computing, 2012, Volume 16, Issue 3, pp 527-561. http://link.springer.com/article/10.1007/s00500-011-0754-8
- PA184 Heuristic Methods for Search and Optimization, 2010 přednáška Sotware Libraries for Heuristics, http://is.muni.cz/el/1433/jaro2010/PA184/um/09_Lecture-Software-Libraries.pdf
- ParadisEO, A Software Framework for Metaheuristics http://paradiseo.gforge.inria.fr
- EasyLocal++ knihovna pro modelování a řešení kombinatorických optimalizačních problémů pomocí metaheuristic https://bitbucket.org/satt/easylocal-3
- OptaPlanner sofware pro plánování, který využívá různé algoritmy lokálního prohledávání https://www.optaplanner.org/

Metaheuristiky pracující s jedním řešením

8. ledna 2021

Zdroj: *El-Ghazali Talbi, Metaheuristics: From Design to Implementation. Wiley, 2009.*
http://www.lifl.fr/~talbi/metaheuristic/



Kostra algoritmu pracující s jedním řešením

Vstup: iniciální řešení s_0
$t = 0$;
repeat
  (generuj množinu kandidátních řešení – částečné nebo úplné okolí – z s_t)
  generate($C(s_t)$);
  (vyber řešení z $C(s)$ pro náhradu aktuálního řešení s_t)
  $s_{t+1} = \text{select}(C(s_t))$;
  $t = t + 1$;
until splněna podmínka ukončení
Výstup: nejlepší nalezené řešení

Iniciální řešení

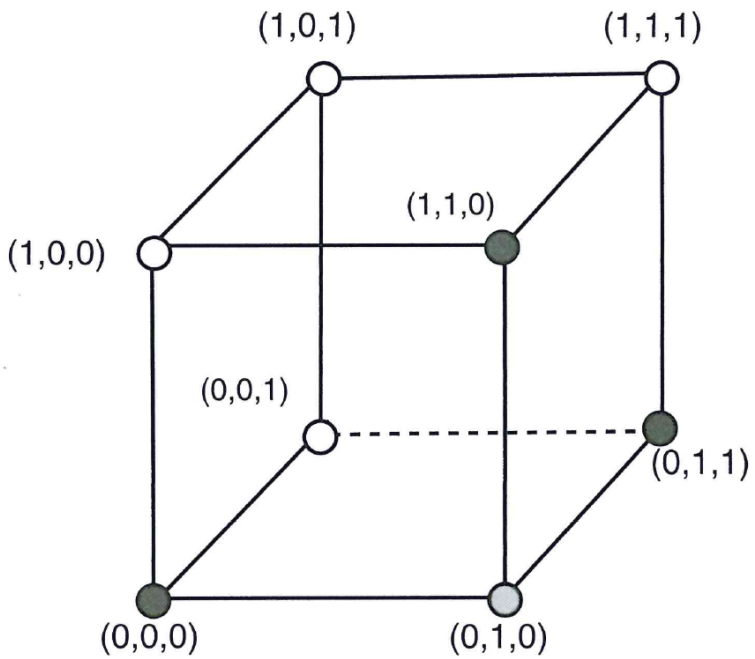
- Hlavní strategie
  - náhodné řešení
  - heuristické řešení (např. pomocí hladového algoritmu)
  - částečně nebo úplně inicializované uživatelem
- Kvalita vs. výpočetní čas
- Použití lepších iniciálních řešení nepovede vždy k lepšímu lokálnímu optimu
- Generování konzistentních náhodných řešení může být obtížné pro vysoce omezené problémy

Okolí

S: množina řešení

Definice Okolí. Funkce okolí N je mapování $N : S \to 2^S$, která přiřadí každému řešení s z S množinu řešení $N(s) \subset S$.

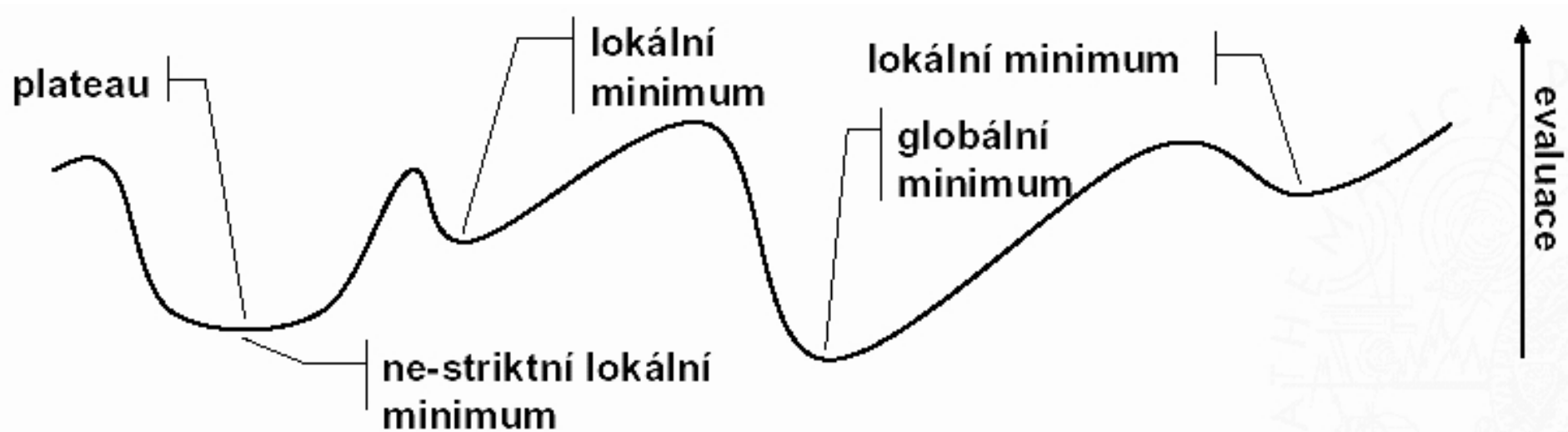
Definice V diskrétním optimalizačním problému je okolí $N(s)$ řešení s reprezentováno množinou $\{s' \in S \mid d(s', s) \leq \epsilon\}$, kde d reprezentuje vzdálenost svázanou s operátorem změny.



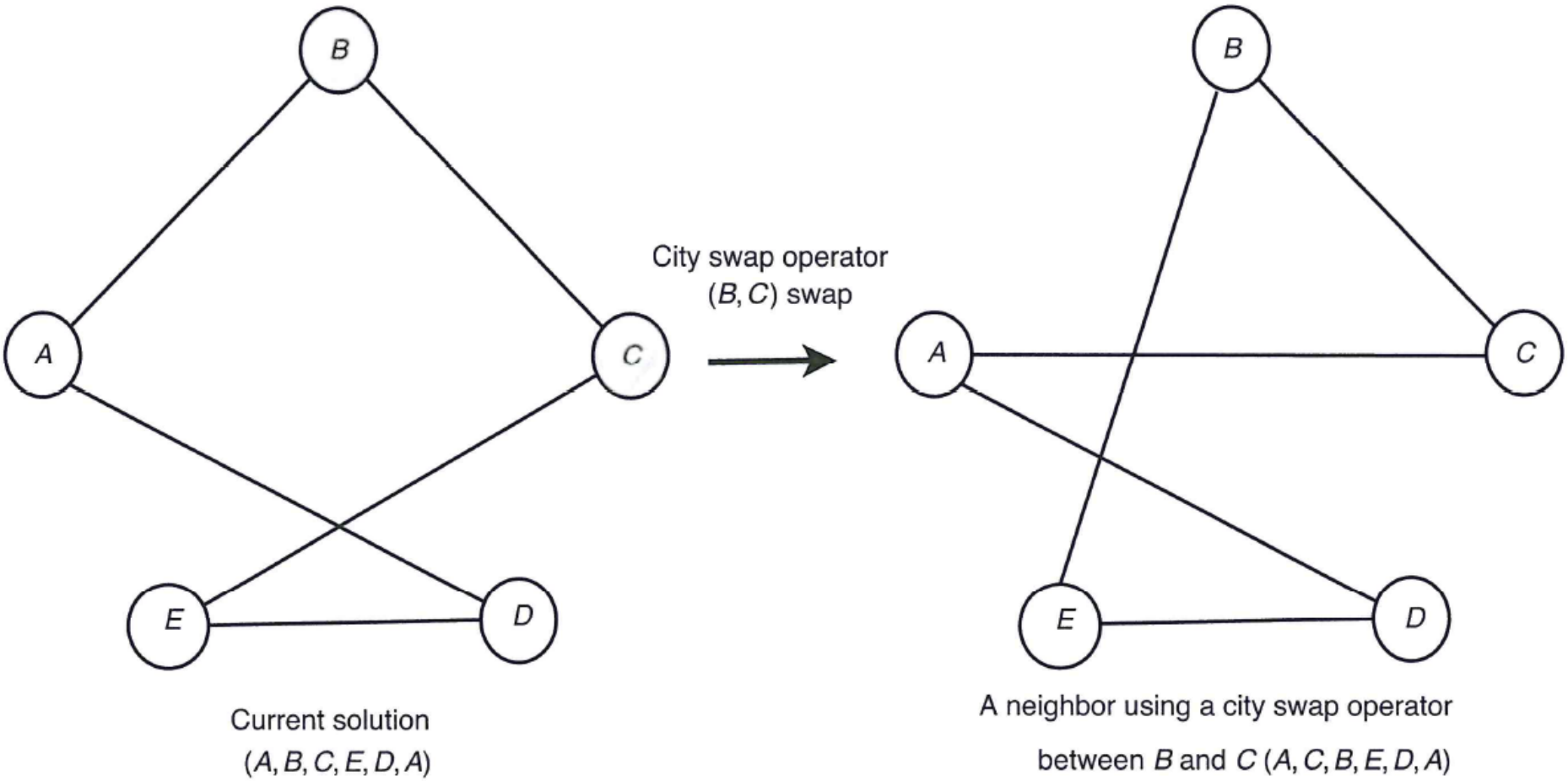
- Uzly reprezentují řešení problému
- Okolí řešení jsou sousední vrcholy v grafu
  - $d(s, s') \leq 1$: měníme hodnotu jedné proměnné
- Pro $(0, 1, 0)$ máme okolí $\{(0, 0, 0), (1, 1, 0), (0, 1, 1)\}$

Lokální optimum

Definice Lokální optimum. Řešení $s \in S$ je lokální optimum vzhledem k dané funkci okolí N, pokud má lepší kvalitu než všichni jeho sousedi, tj. $f(s) \leq f(s')$ pro všechna $s' \in N(s)$. (pro minimalizační problém)

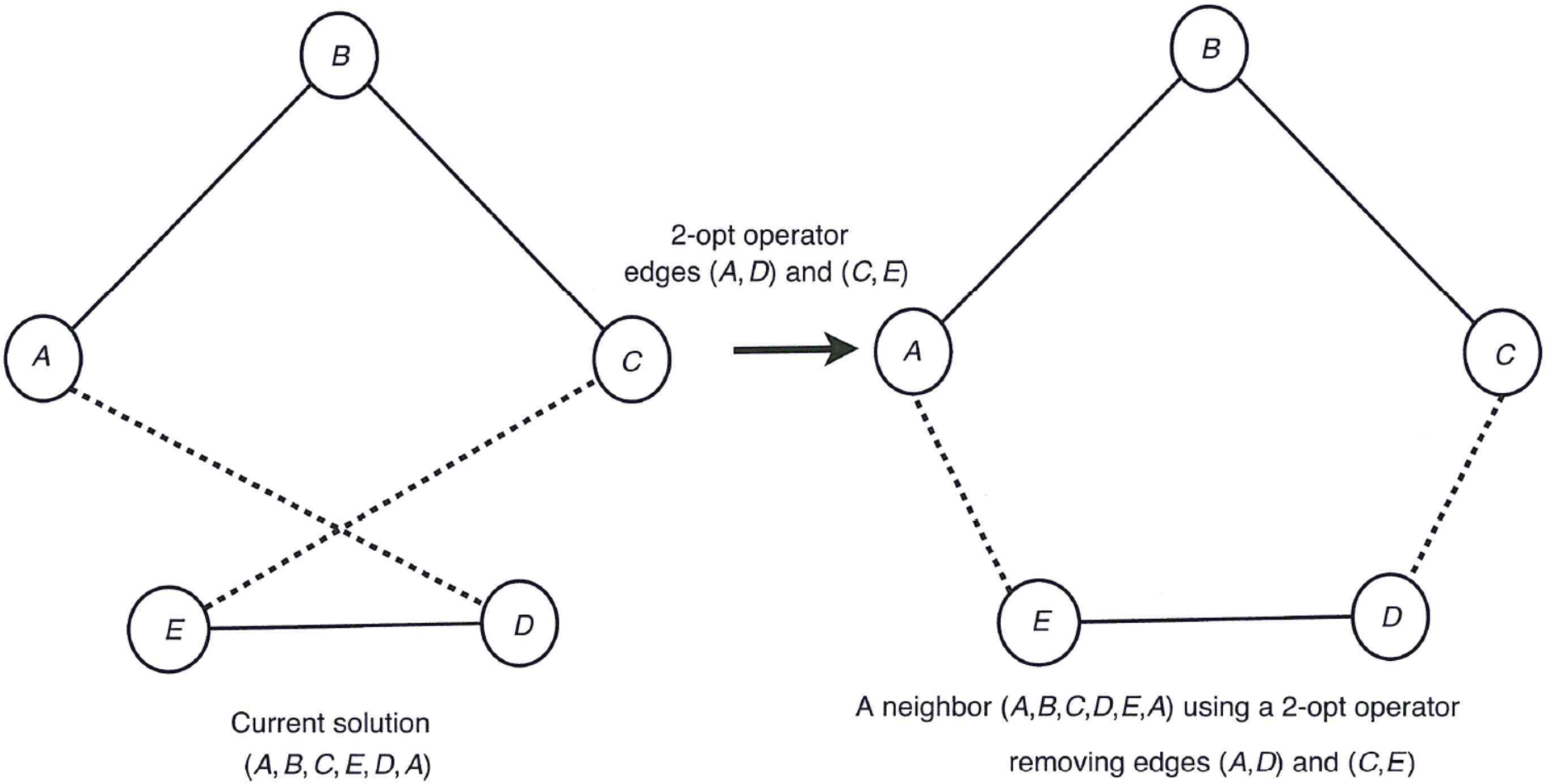


Příklad okolí: k-vzdálenost (k-distance)



- Permutační problémy, např. TSP: 2-distance vymění dvě města (k-distance vymění k měst)
- Velikost okolí pro 2-distance: $n(n-1)/2$, kde n je počet měst

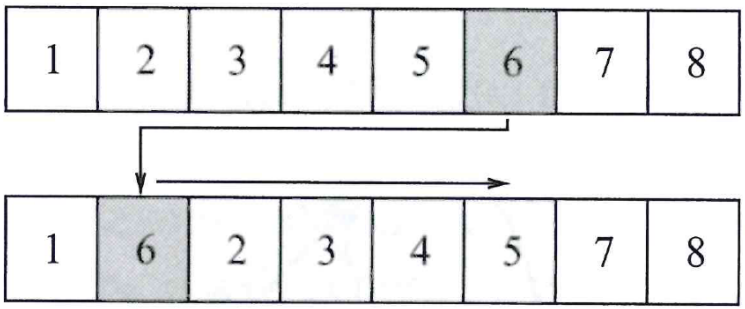
Příklad okolí: k-výměna (k-opt)



- Permutační problémy, např. TSP
- k-opt smaže k hran a nahradí je dalšími k hranami
- Velikost okolí pro 2-opt: $[n(n-1)/2 - n]$, kde n je počet měst
  - uvažujeme všechny páry hran kromě sousedních párů

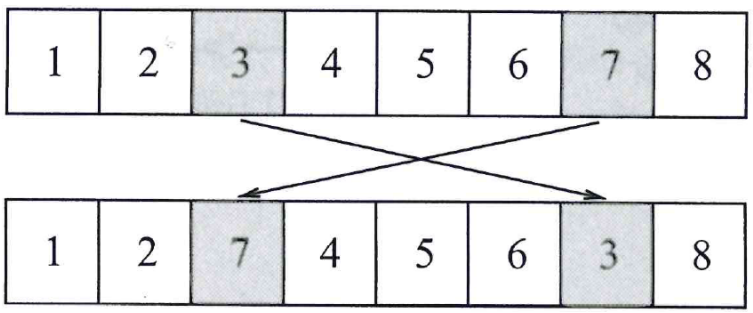
Permutační okolí pro rozvrhovací problémy

Okolí založené na pozici

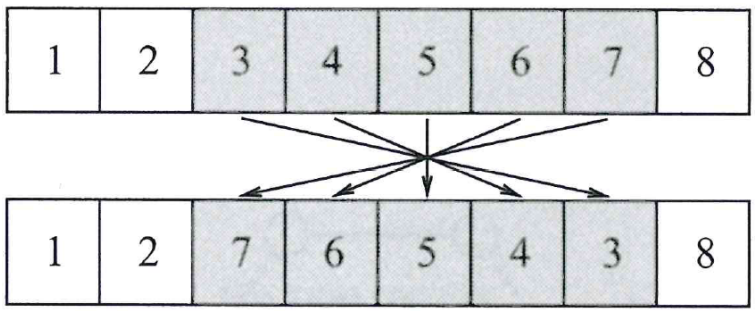


operátor vložení

Okolí založená na pořadí



operátor výměny



operátor inverze

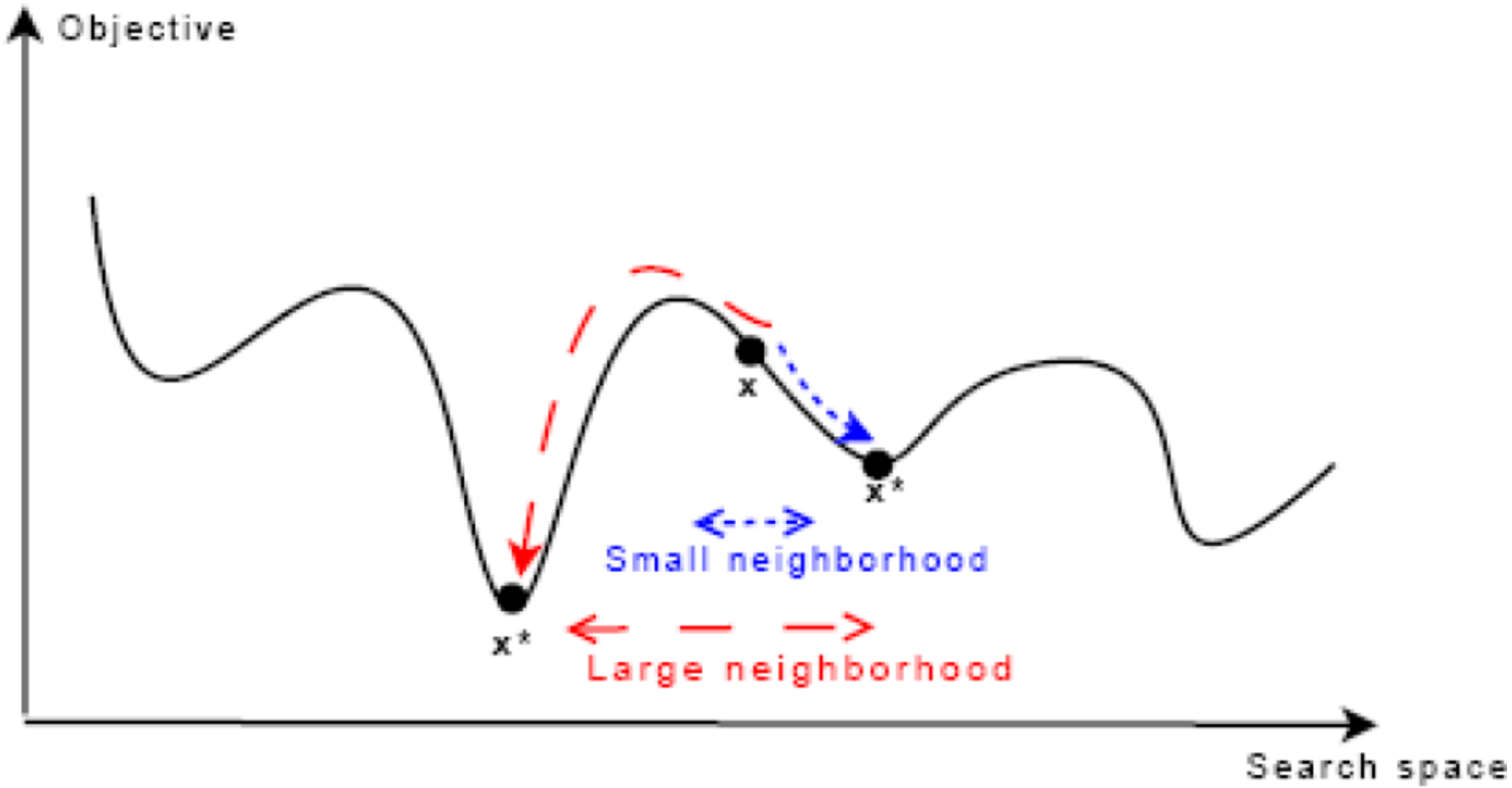
Přehled okolí

- Okolí, lokální optimum
- Malá okolí
  - k-distance (k-vzdálenost)
  - k-opt (k-výměna)
  - permutační okolí
    - operátor vložení
    - operátor výměny
    - operátor inverze
- Rozsáhlá okolí
  - řetězec odstranění
  - cyklická výměna
- Inkrementální evaluace okolí

Rozsáhlá okolí (very large neighborhoods)

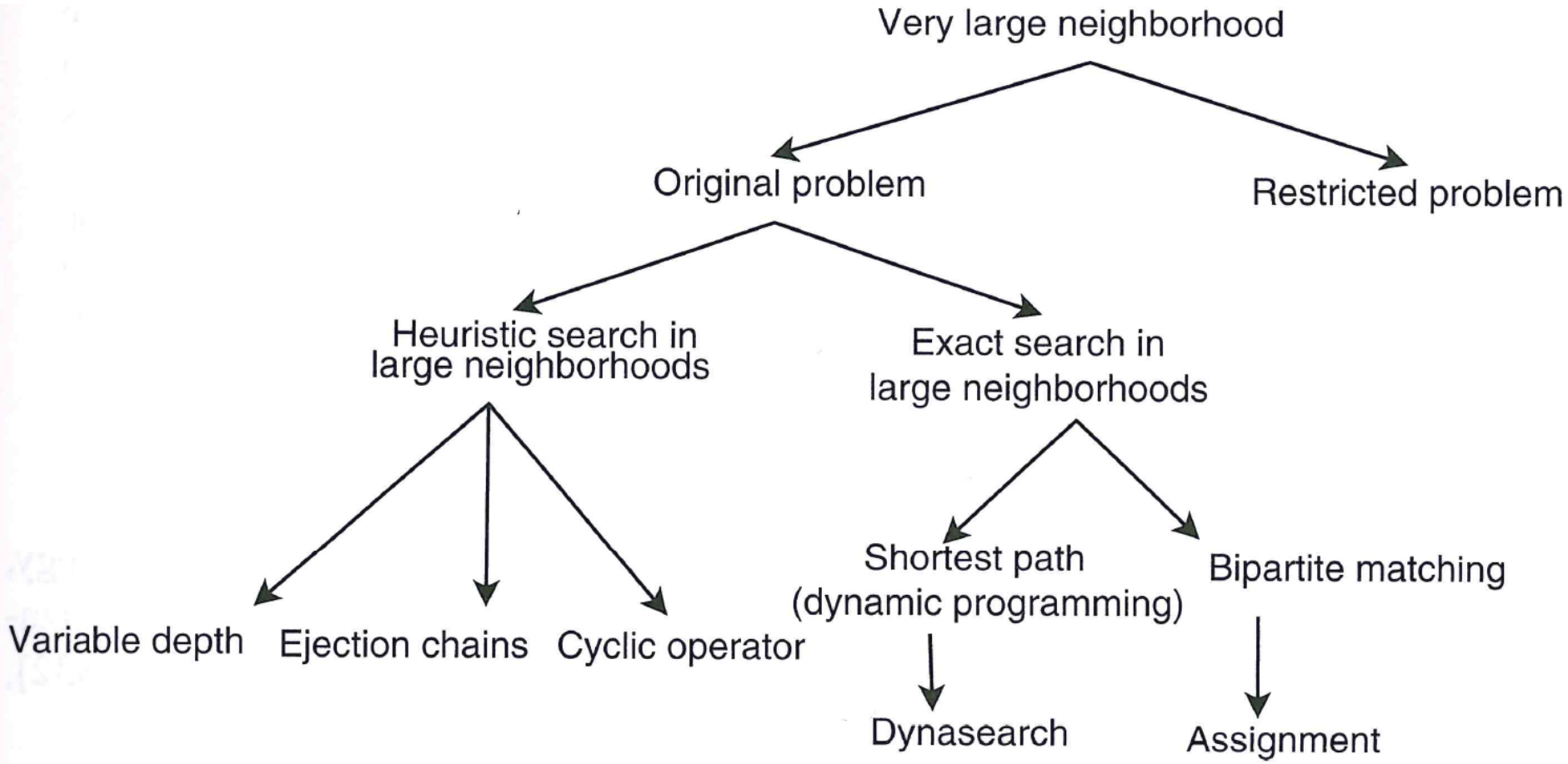
Kompromis:

- velikost (diametr) okolí (výpočetní složitost na jeho prozkoumání) vs. kvalita řešení



Efektivní algoritmy na prozkoumání rozsáhlých okolí

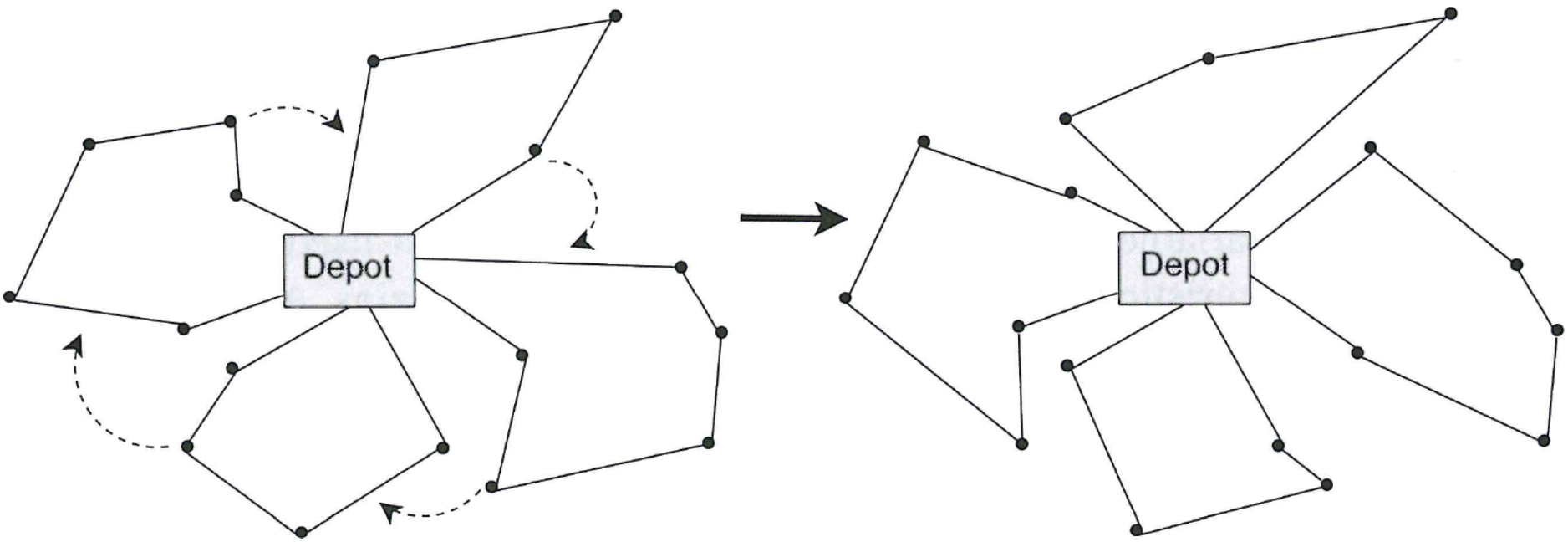
- Velikost okolí: polynom vyššího řádu (n>2) n. exponenciální
- Hlavní problém: identifikace zlepšujících okolí nebo nejlepšího souseda bez enumerace celého okolí



Řetězec odstranění (ejection chain)

(Capacitated) vehicle routing problem

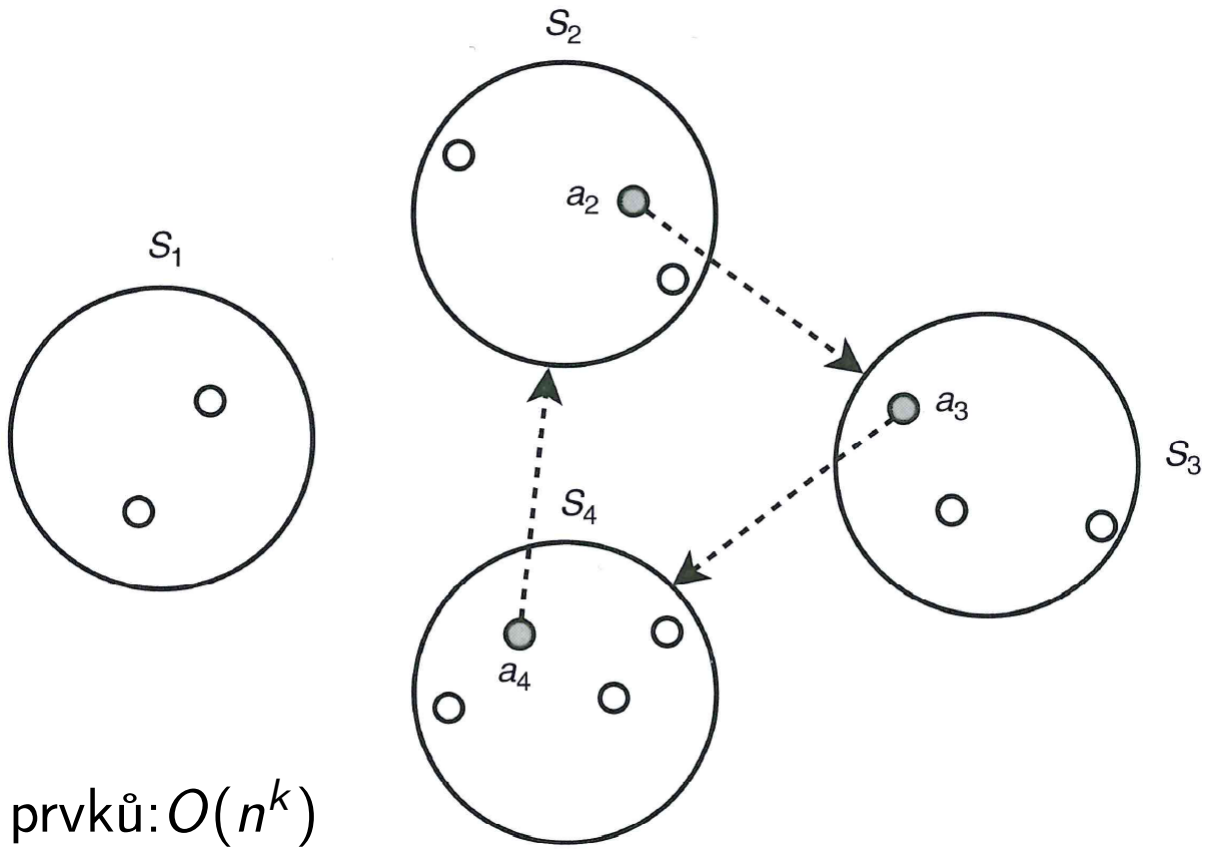
- řetězec odstranění: posloupnost přesunů zákazníka z jednoho okruhu na následující
  - každý řetězec odstranění zahrnuje k úrovní začínajících na okruhu 1 a končících na okruhu k
  - vrchol je odstraněn z okruhu 1 a přesunut na okruh 2, vrchol z okruhu 2 přesunut na okruh 3, atd. až vrchol z k-1 na k, poslední vrchol z okruhu k přesunut do okruhu 1
- úspěšná změna: žádný vrchol není v řešení více než jednou



Cyklická výměna (cyclic exchange) ve skupinách

Problémy rozdělení n prvků do q skupin (např. při vyvažování zátěže)

- prvky mají být rozděleny do skupin $S = \{S_1, \ldots, S_q\}$
- cyklická výměna 3 prvků (3-okolí): prvek $a_2 \in S_2$ je přesunut do S_3, $a_3 \in S_3$ je přesunut do S_4, $a_4 \in S_4$ přesunut do počáteční skupiny S_2

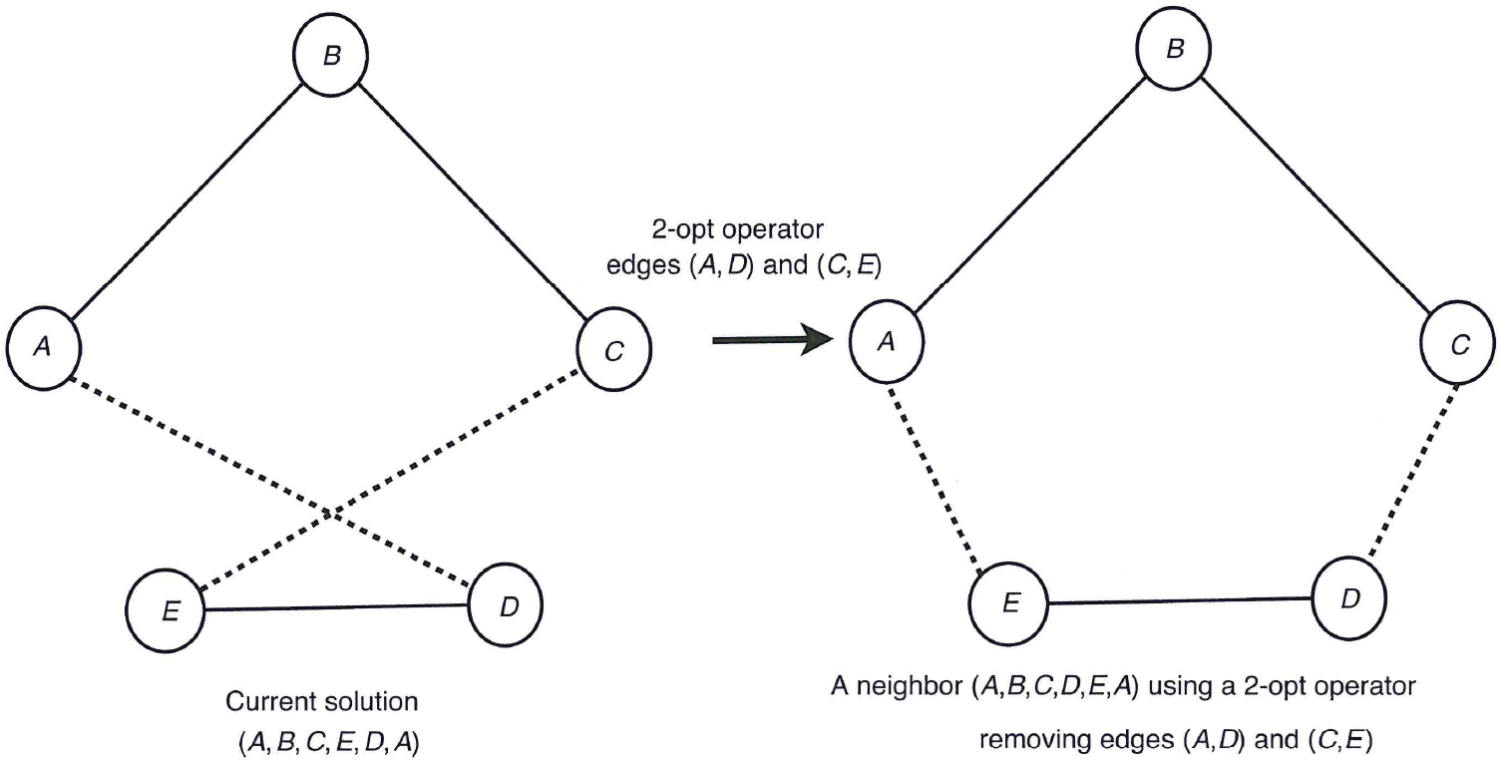


- velikost k-okolí pro n prvků: $O(n^k)$
  - pro výměnu dvojice prvků máme $O(n^2)$

Inkrementální evaluace okolí

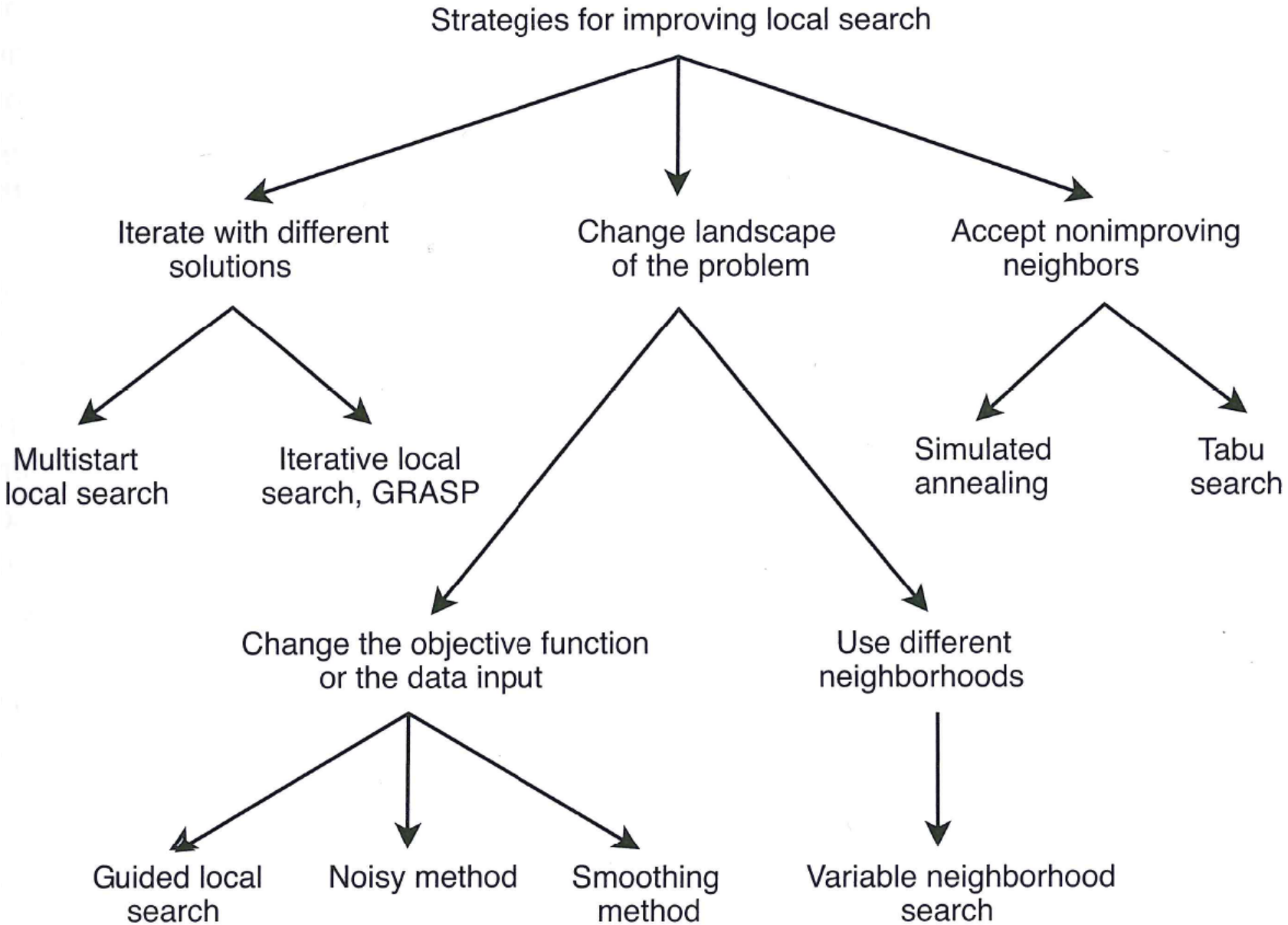
- Evaluace řešení: nejdražší část výpočtu
- Naivní evaluace: úplná evaluace každého řešení v okolí
- Cíl: navrhnout vhodnou inkrementální evaluaci okolí

Příklad:



$$\Delta f = distance(A, E) + distance(C, D) - distance(A, D) - distance(C, E)$$

Pokročilé lokální prohledávání



Lokální prohledávání (local search, hill climbing, horolezecký algoritmus)

- Opakování z minulé přednášky
  - hill climbing, inicializace řešení, okolí
- Nejstarší a nejjednodušší metaheuristika
- Nahrazuje aktuální řešení zlepšujícím řešením

$s := s_0$; *(generuj iniciální řešení s_0)*
while není splněna podmínka ukončení **do**
    generuj($N(s)$); *(generování sousedů z okolí)*
    if neexistuje lepší soused **konec**;
    $s := s'$; *(vyber lepšího souseda $s' \in N(s)$)*
end while
výstup: finální nalezené řešení (lokální optimum)

Lokální prohledávání: výběr souseda

Výběr souseda z okolí:

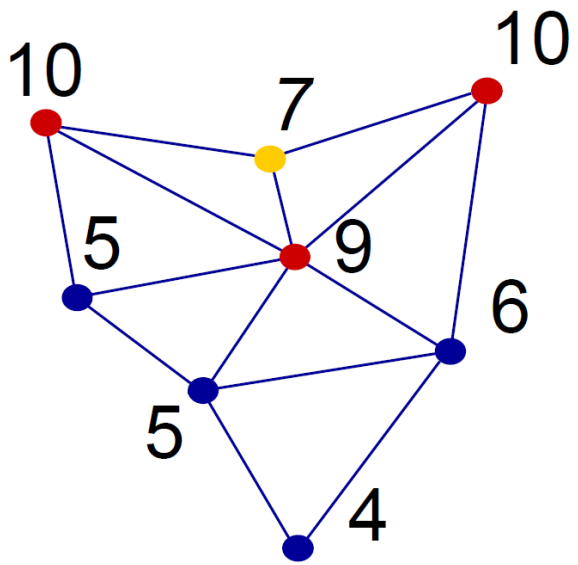
- nejlepší zlepšující (steepest descent, metoda největšího spádu): výběr nejlepšího souseda (tj. nejvíce zlepšuje účelovou funkci)
- první zlepšující: systematický/deterministický výběr prvního souseda, který je lepší než aktuální řešení
- náhodný výběr: náhodný výběr mezi zlepšujícími sousedy

Prohledávací prostor a krajina (fitness landscape)

Definice: Prohledávací prostor je definován orientovaným grafem $G = (S, E)$, kde množina vrcholů S odpovídá řešením problému definovaným jejich reprezentací a množina hran E je určena operátorem změny použitým pro generování sousedů z okolí.

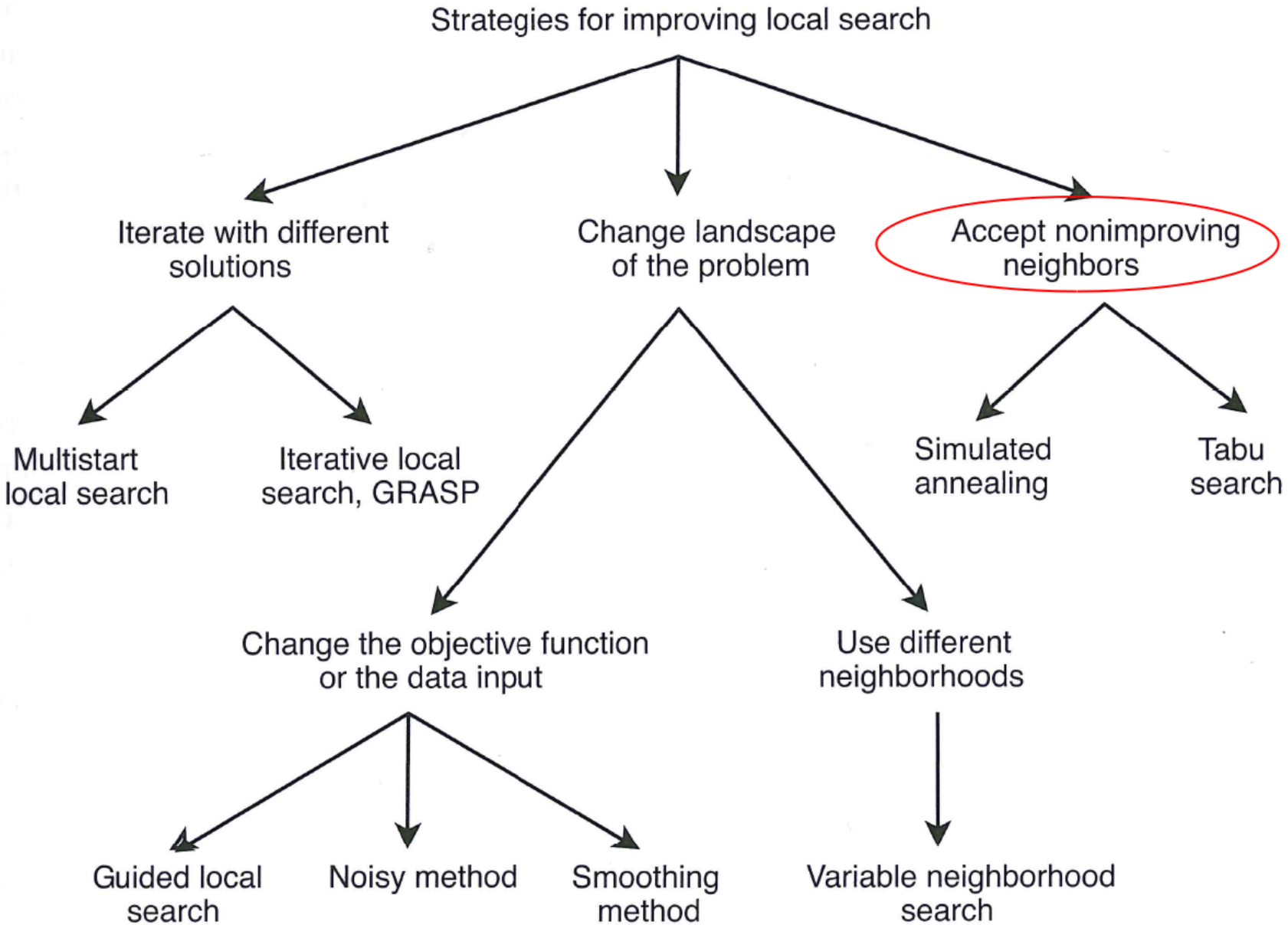
Poznámka: Sousedním vrcholem řešení v G jsou sousedé z okolí řešení.

Definice: Krajina (daná vhodností) I je definována dvojicí $\langle G, f \rangle$, kde G je prohledávací prostor a f účelová funkce, která řídí prohledávání.



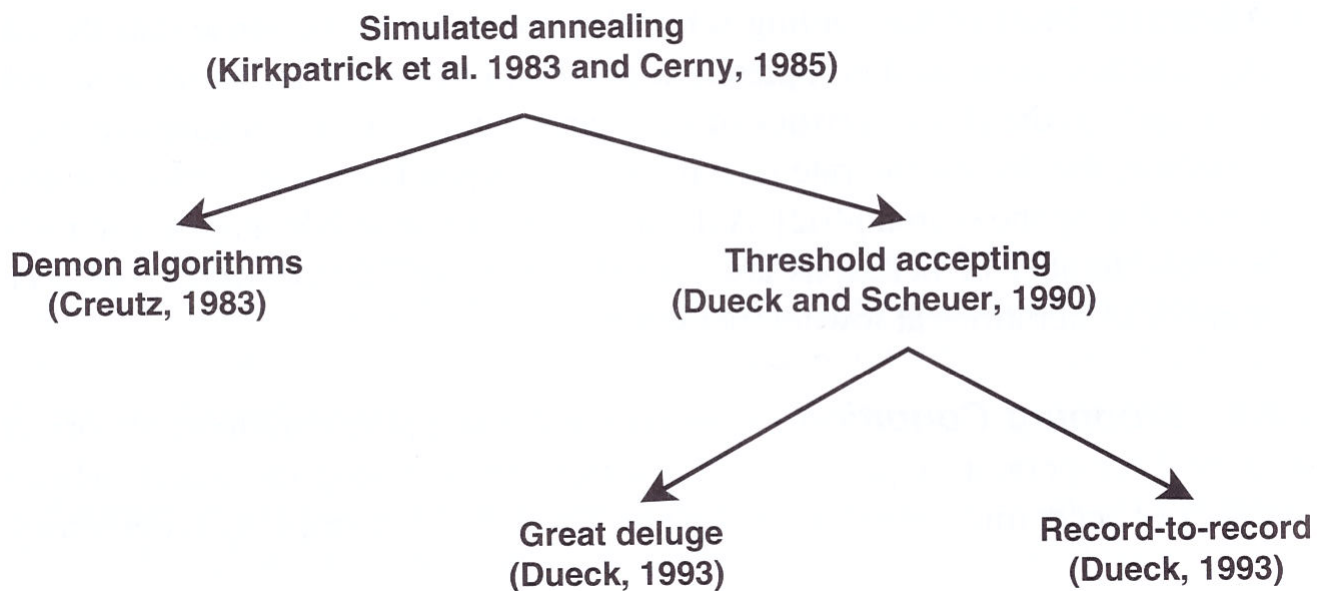
řešení s_0 s $f(s_0) = 7$ má v okolí
3 sousedy s hodnotou účelové funkce
10,9,10

Pokročilé lokální prohledávání



Simulované žíhání (simulated annealing, SA) a spol.

- Klasický reprezentant lokálního prohledávání
  - FI: IA101, PV027, PA167, PA163
  - viz také Talbi(ho) kniha a její průsvitky ke kapitole 2
- Náhodný výběr souseda z okolí
- Výběr lepšího souseda je vždy akceptován jako u hill climbing
- Akceptuje i výběr horšího souseda
- Na začátku prohledávání je povolena větší pravděpodobnost zhoršení vybraného řešení, postupně se povoluje menší a menší pravděpodobnost zhoršení vybraného řešení



Algoritmy založené na přijetí prahem (threshold accepting)

- Deterministická varianta simulovaného žíhání
- Akceptována i horší řešení
  s maximálními zhoršením o prahovou hodnotu Q

Algoritmus přijetí prahem

$s = s_0$; *(generování iniciálního řešení)*
$Q = Q_{\max}$; *(počáteční nastavení prahu)*
repeat
  repeat *(při pevném prahu)*
    generuj náhodně souseda $s' \in N(s)$;
    $\Delta E = f(s') - f(s)$;
    if $\Delta E \leq Q$ **then** $s = s'$ *(akceptování souseda)*
  until splněna podmínka rovnováhy
    (např. po pevném počtu iterací při každém Q)
  $Q = g(Q)$; *(aktualizace prahu, typicky zmenšení)*
until splněna podmínka ukončení *(např. při $Q \leq Q_{\min}$)*
výstup: nejlepší nalezené řešení

Great Deluge algoritmus (alg. velké potopy)

- Analogie s potopou a stoupající vodní hladinou
- Horolezec (kvalita řešení) „vždy zůstane nad vodní hladinou"
- Pozor, algoritmus uveden klasicky pro minimalizační problém, tj. hladinu (*LEVEL*) musíme redukovat a snažíme se zůstat pod *LEVEL*

$s = s_0$; *(generování iniciálního řešení)*
zadej rychlost deště *UP*; *($UP > 0$)*
zadej počáteční úroveň vodní hladiny *LEVEL*;
repeat
    generuj náhodného souseda $s' \in N(s)$;
    if $f(s') < LEVEL$ **then** $s = s'$; *(akceptuj řešení)*
    $LEVEL = LEVEL - UP$; *(aktualizuj úroveň hladiny)*
until splněna podmínka ukončení
výstup: nejlepší nalezené řešení

UP je kritický parametr

- příliš vysoký $\Rightarrow$ rychle nalezeno nekvalitní řešení
- nízký $\Rightarrow$ lepší výsledky se zvyšujícími se časovými nároky

Tabu prohledávání

- Deterministické rozšíření základního algoritmu lokálního prohledávání
  - často kombinováno i s dalšími algoritmy
- Umožňuje akceptovat i horší řešení
  - řešení horší/stejné kvality akceptováno, pokud neexistuje lepší
  - cyklení je zabráněno prostřednictvím tzv. tabu seznamu
- Udržován tabu seznam
  - seznam několika posledních změn, které jsou zakázány (typicky 5-9)
  - např. problém obchodního cestujícího, 2-výměna měst,
    v tabu seznamu uloženy dvojice měst z N posledních změn
- Aspirační kritérium
  - podmínka, za které je možné realizovat i změny z tabu seznamu
  - např. povolení změn, které vedou k lepší hodnotě účelové funkce

Algoritmus tabu prohledávání

$s := s_0$; *(generuj iniciální řešení s_0)*
inicializace tabu seznamu;
repeat
    nalezni nejlepší přípustné $s' \in N(s)$;
        (přípustné řešení: není v tabu seznamu n. platí asp. kritérium)
    $s := s'$;
    aktualizace tabu seznamu a aspiračního kritéria;
until splněna podmínka ukončení
výstup: nejlepší nalezené řešení

Poznámka:

- nejlepší přípustné řešení nemusí být nutně zlepšující
- je to „jen" nejlepší řešení z $N(s)$

Iterativní lokální prohledávání (iterated local search)

Multistart local search (opakované lokální prohledávání)

- opakování lokálních prohledávání
- iniciální řešení je generováno při další iteraci náhodně

Iterativní lokální prohledávání

- vylepšení multistart local search
- nalezené lokální optimum změněno a použito při dalším lokálním prohledávání
- obecná kostra: lze použít libovolný alg. lokálního prohledávání

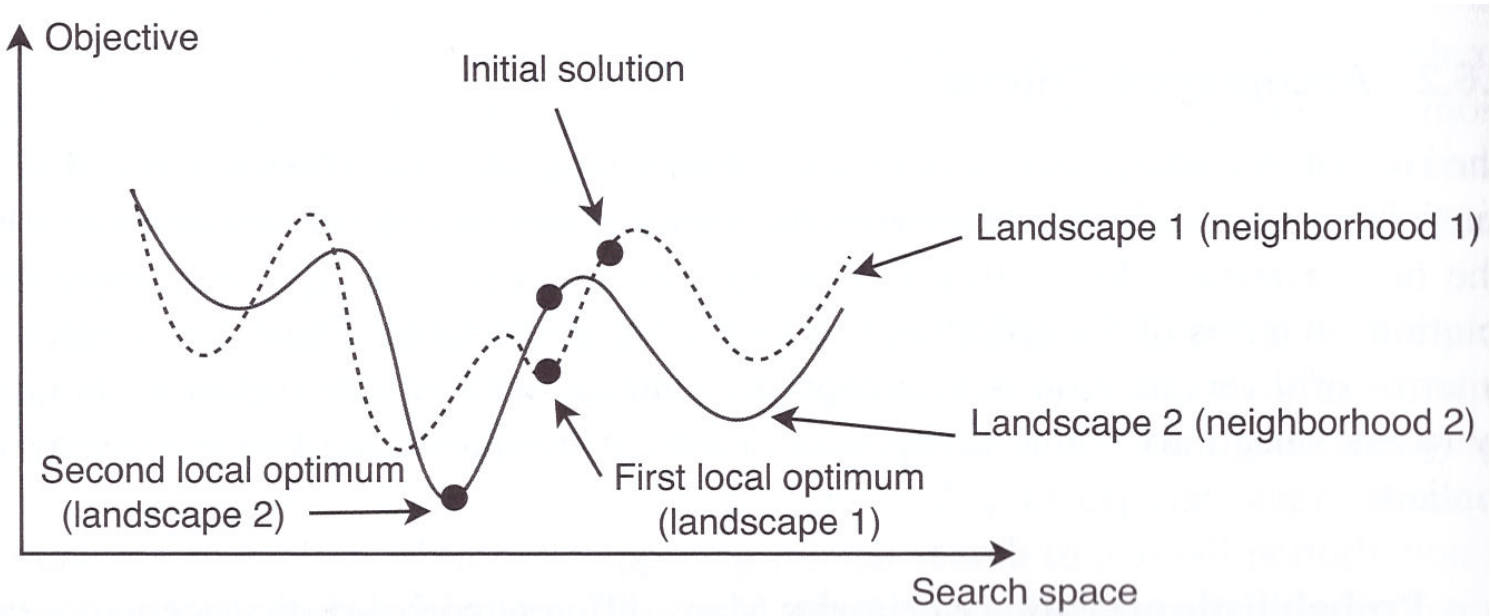
Algoritmus iterativního lokálního prohledávání

$s = s_0$; *(generuj iniciální řešení)*
$s_* =$ lokální_prohledávání(s); *(aplikuj daný alg. lok. prohledávání)*
repeat
  $s' =$ změna(s_*, historie prohl.); *(změň spočítané lokální minimum)*
  $s'_* =$ lokální_prohledávání(s'); *(aplikuj lok. prohl. na změněné řešení)*
  $s_* =$ akceptuj(s_*,s'_*, paměť prohl.); *(kritérium přijetí řešení)*
until splněna podmínka ukončení
výstup: nejlepší nalezené řešení

Změna

- velká náhodná změna v řešení
- část řešení zachována, část změněna
- použití efektivnějšího lokálního prohledávání vyžaduje větší změnu
- příliš velká změna povede k opakovanému lokálnímu prohledávání, kde při opakované iteraci začínáme s náhodným řešením

Prohledávání s proměnlivým okolím (variable neighborhood search, VNS)



Použití různých okolí:

- Komplementární okolí:

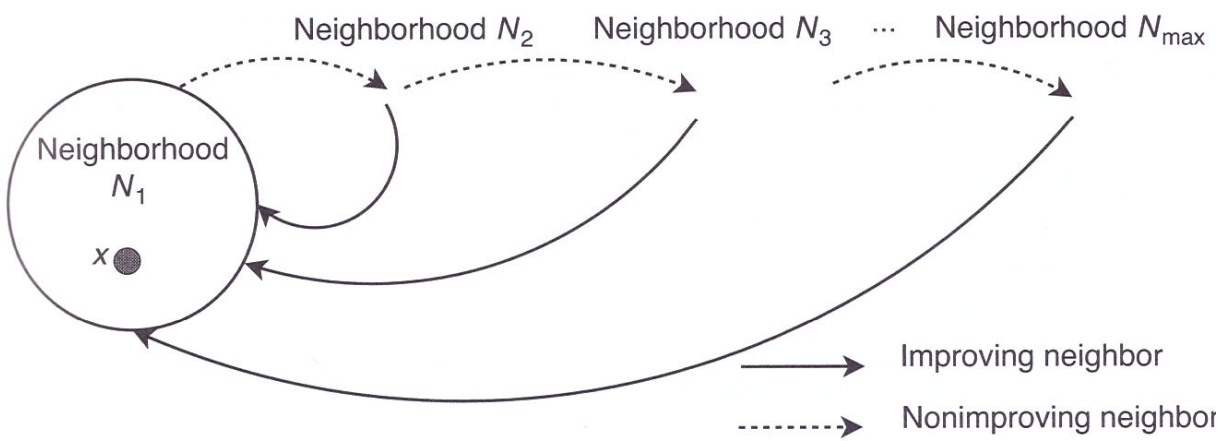
  lokální optimum okolí N_i nebude lokálním optimem okolí N_j

- (1) První lokální optimum získáno dle okolí 1
  (2) Abychom z něj unikli, prohledáváme okolí 2
  (3) Pomocí okolí 2 se dostaneme do lokálního optima okolí 2

Zlepšující prohledávání s proměnlivým okolím (variable neighborhood descent algorithm, VND)

- VNS založeno na zlepšujícím prohl. s proměnlivým okolím (VND)
- VND je deterministická verze VNS

$s = s_0$; *(generuj iniciální řešení)*
$l = 1$;
while $l \leq l_{max}$ **do**
    nalezni nejlepšího souseda $s' \in N_l(s)$;
    if $f(s') < f(s)$ **then** $s = s'$; $l = 1$; *(máme lepší řešení)*
    else $l = l + 1$;
výstup: nejlepší nalezené řešení



Základní verze algoritmu s proměnlivým okolím

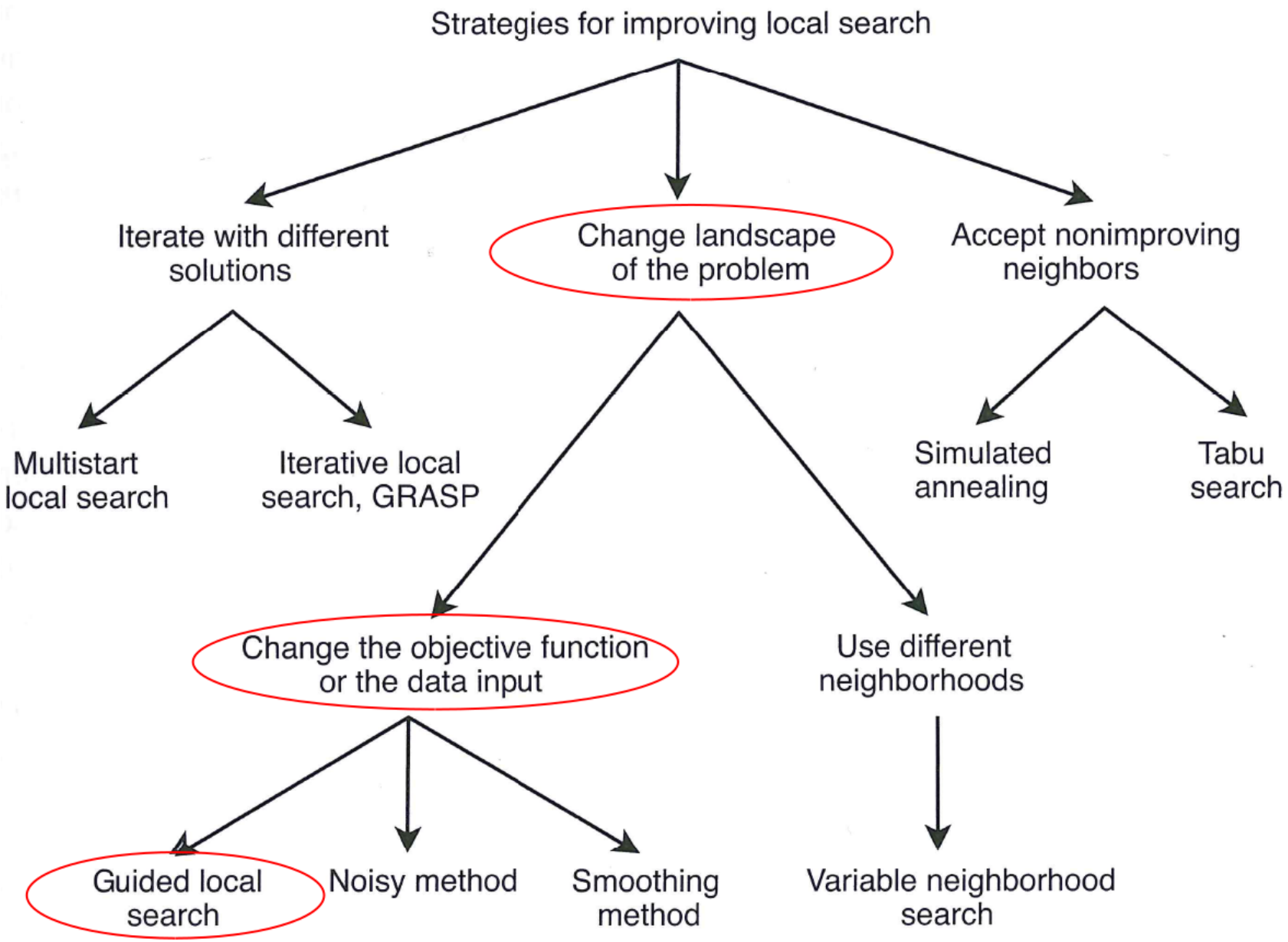
$s = s_0$; *(generuj iniciální řešení)*
repeat
  $k = 1$;
  repeat
    vyber náhodně $s' \in N_k(s)$; *(zatřepání/shaking)*
    $s'' =$ lokální_prohledávání(s');
    if $f(s'') < f(s)$ **then** $s = s''$; $k = 1$; *(máme lepší řešení)*
    else $k = k + 1$;
  until $k = k_{max}$;
until splněna podmínka ukončení
výstup: nejlepší nalezené řešení

Okolí pro zatřepání

- typicky jsou postupně prozkoumávána vnořená okolí
  $N_1(s) \subset N_2(s) \cup \cdots \cup N_k(s), \forall s \in S$
- příklad: problém obchodního cestujícího a k-opt (k-výměna) měst

Komentář: v obecné verzi algoritmu je lokální prohledávání nahrazeno VND

Pokročilé lokální prohledávání



Řízené lokální prohledávání (guided local search, GLS)

- Opakovaná lokální prohledávání s upravenou hodnotou účelové funkce
- Každé řešení $s \in S$ má množinu m charakteristik i s cenou c_i

$$I_i(s) = \begin{cases} 1 & \text{pokud } s \text{ má charakteristiku } i \\ 0 & \text{jinak} \end{cases}$$

  - snaha o prohledávání částí stav. prostoru s nižší cenou charakteristik
  - př. problém obchodního cestujícího: zda je hrana (a, b) v řešení, cenu určuje délka hrany (vzdálenost měst)
- Nová účelová funkce

$$f'(s) = f(s) + \lambda \sum_{i=1}^{m} p_i I_i(s)$$

  - penalizace p_i: význam charakteristiky (vysokou penalizaci nechceme) p_i inicializováno na 0 pro všechna i
  - λ se používá pro normalizaci
- Charakteristika i s nejvyšším užitkem $u_i(s) = I_i(s)\frac{c_i}{1+p_i}$ penalizována

užitek/hodnota

  - chceme se příště vyhnout charakteristikám s vysokou cenou
  - charakteristiky použité v řešení penalizovány pro podporu diversifikace

Algoritmus řízeného lokálního prohledávání

$s = s_0$; *(generuj iniciální řešení)*
forall i **do** $p_i = 0$; *(inicializace penalizací)*
repeat
  $s' =$ lokální_prohledávání(s); *(s upravenou účelovou funkcí)*
  forall i **do** $u_i(s') = I_i(s')\frac{c_i}{1+p_i}$; *(spočítej užitek pro všechny char.)*
  $u_j = \max_{i=1,\ldots m}(u_i(s'))$; *(urči charakteristiku s max. užitkem)*
  $p_j = p_j + 1$; *(změň účelovou funkci penalizací char. j)*
until splněna podmínka ukončení
výstup: nejlepší nalezené řešení

- Intensifikace
  - GLS prohledává stavový prostor s nižšími cenami c_i
- Diversifikace
  - GLS penalizuje charakteristiky použité ve vygenerovaném lokálním optimu, abychom se jim vyhnuli

Příklad: symetrický problém obchodního cestujícího

- $d_{i,j} = d_{j,i}$ udává vzdálenost mezi městy i, j (symetrická matice)
- Město $t(i)$ následuje po trase za městem i
- Řešení zakódováno jako permutace měst
- Původní účelová funkce

$$f(s) = \sum_{i=1}^{n} d_{i,t(i)}$$

- Indikace charakteristiky

$$I_{(i,j)}(s) = \begin{cases} 1 & \text{pokud hrana } (i,j) \in s \\ 0 & \text{jinak} \end{cases}$$

- Nová účelová funkce

$$f'(s) = \sum_{i=1}^{n} d_{i,t(i)} + \lambda p_{i,t(i)}$$

  - $p_{i,j}$ udává penalizaci za hranu (i,j)

- Užitek

$$u_i(s,(i,j)) = I_{(i,j)}(s)\frac{d_{ij}}{1+p_{ij}}$$

Příklad: TSP a GLS

$$u_i(s) = l_i(s)\frac{c_i}{1+p_i}$$

- TSP s hranami 1, 2, 3, 4 a cenou 10, 6, 3, 2 v řešení
- Funkce užitku pro hrany: 10/1, 6/1, 3/1, 2/1
  - p_i iniciálně 0
- Penalizována hrana s užitkem 10/1
- Nové řešení s hranami 2, 3, 4, 5 a cenou 6, 3, 2, 3
- Funkce užitku pro hrany: 6/1, 3/1, 2/1, 3/1
- Penalizována hrana s užitkem 6/1
- Nové řešení s hranami 1, 3, 4, 6 a cenou 10, 3, 2, 7
- Funkce užitku pro hrany: 10/2, 3/1, 2/1, 7/1
- Penalizována hrana s užitkem 7/1
  - hrana s cenou 10 se opakuje, je možná důležitá, proto ji zatím nepenalizujeme
- ...

Hyper-heuristiky

Hyper-heuristiky

- prvotní myšlenka: heuristiky na výběr heuristik
- rozšířeno na: prohledávací metoda nebo učící mechanismus pro výběr nebo generování heuristik pro řešení obtížných prohledávacích problémů

Kategorie hyper-heuristik

- výběr heuristik: snaha o nalezení vhodné sekvence aplikace existujících heuristik pro efektivní řešení problému
- generování heuristik: snaha o vývoj nových heuristik na základě komponent známých heuristik
  - učení heuristik pomocí neuronových sítí, evolučních algoritmů, ...

Hyper-heuristiky pracují nad prohledávacím prostorem heuristik
(a ne přímo nad prohledávacím prostorem řešeného problému)

> *Hyper-heuristics: a survey of the state of the art, Burke, Gendreau, Hyde, Kendal, Ochoa, Özcan, Qu, Journal of the Operational Research Society (2013), 64, 1695-1724.*

Problém: rozvrhování událostí jako barvení grafu

Barvení grafu

- vrcholy = události
- hrany mezi událostmi, které nesmí být ve stejném časovém slotu
- barva říká, ve kterém čase událost rozvrhneme
  sousední vrcholy musí mít různou barvu / musí být v různém čase

Jednoduché heuristiky, které konstruují řešení seřazením událostí:

Stupeň saturace

- události seřazeny (1) ve vzrůstajícím pořadí podle počtu časových slotů, kam je lze umístit v aktuálním částečném rozvrhu (saturace), a pak (2) dle klesajícího stupně vrcholu v neobarveném podgrafu

Počet účastníků (zapsaných studentů)

- události seřazeny v klesajícím pořadí podle počtu účastníků, resp. zapsaných studentů

...

Náhodné pořadí

- dosud nerozvržené události seřazeny náhodně

Grafová hyper-heuristika pro rozvrh událostí

inicializace seznamu heuristik $hl = \{h_1, \ldots, h_k\}$;
for $i = 0$ **to** počet_iterací **do**
    $h = hl$ se dvěma vyměněnými heuristikami; *(lokální změna v tabu prohl.)*
    if h není v „neúspěšném seznamu"
        if h není v tabu seznamu
            for $j = 0$ **to** $j = k$ **do** *(h použito pro konstrukci řešení)*
                rozvrhuj prvních e událostí v seznamu událostí seřazených dle h_j
                if neexistuje žádné řešení
                    ulož h do „neúspěšného seznamu" *(aktualizace „neúsp. seznamu")*
                else if cena řešení $c < c_{best}$
                        ulož nejlepší řešení; $c_{best} = c$;
                smaž nejstarší položku z tabu seznamu, pokud je plný;
                přidej h do tabu seznamu;
    $hl = h$;
    hill climbing s výběrem nejlepšího souseda aplikovaný na nalezené řešení;
výstup: nejlepší nalezené řešení s cenou c_{best}

Poznámky k algoritmu

- Přístup:
  - heuristika na výběr heuristik
  - tabu prohledávání nad stavovým prostorem heuristik
- $e \times k$ odpovídá počtu událostí v rozvrhu
  - k délka heuristického seznamu
  - e událostí rozvrhnuto najednou stejnou heuristikou
    - je známo, že heuristiky se při řešení po sobě často opakují
    - ušetří čas na řešení
  - v každé iteraci tak sestrojíme úplné řešení
- „Neúspěšný seznam" obsahuje začátek heuristického seznamu, kde došlo k chybě (nenalezeno řešení)
  - př. při neúspěchu u seznamu „h1h2h3..." při aplikaci h3 zakážeme heuristické seznamy „h1h2h3h4...", „h1h2h3h5...", ...

A graph-based hyper-heuristic for educational timetabling problems, E. Burke, B. McCollum, A. Meisels, S. Petrovic, R. Qu. European Journal of Operational Research 176 (2007) 177-192.

Shrnutí

4. Základní algoritmus
5. Iniciální řešení
6. Okolí
   - Malá okolí
   - Rozsáhlá okolí
7. Lokální prohledávání
   - Horolezecký algoritmus (hill climbing)
8. Algoritmy s výběrem horšího řešení
   - Simulované žíhání
   - Algoritmus přijetí prahem (threshold accepting)
   - Algoritmus velké potopy (great deluge)
   - Tabu prohledávání
9. Iterace s různými řešeními
   - Opakované (multistart) a iterativní (iterated) lokální prohledávání
10. Použití různých okolí
    - Prohledávání s proměnlivým okolím (variable neighborhood search)
      - Zlepšující prohledávání s proměnlivým okolím (variable neighborhood descent)
11. Změna účelové funkce

Cvičení

Máme zadán problém rozvrhování n předmětů do m učeben v p časových slotech. Každý předmět je vyučován v jednom slotu a v jedné učebně, přičemž dva předměty nesmí být ve stejném čase a místě.

- Jakým způsobem zakódujete řešení problému?
- Pro každý předmět je určena penalizace za přiřazení do daného času. Jak definujete účelovou funkci?
- Jak vygenerujete iniciální řešení?
- Jak spočítáte inkrementálně účelovou funkci, pokud je lokální změna dána výměnou dvou předmětů?
- Jaké další typy lokálních změn byste mohli aplikovat?
- Jak lze aplikovat změnu pomocí řetězce odstranění (ejection chain) na tento problém? A jak bude vypadat inkrementální výpočet účelové funkce v tomto případě?
- Jak velké je okolí u horolezeckého algoritmu (hill climbing), pokud hledáte nejlepšího zlepšujícího souseda (při výměně 2 předmětů)?
- Jak navrhnete prvky tabu seznamu, pokud je lokální změna dána výměnou dvou předmětů?

Metaheuristiky s populacemi

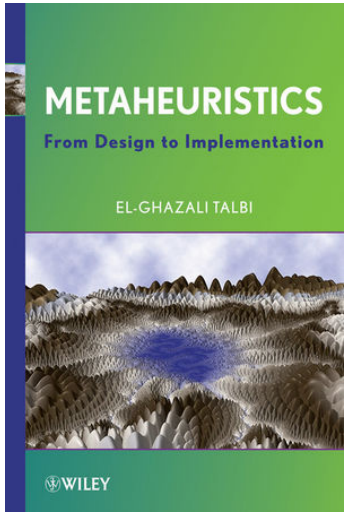
8. ledna 2021

13 Společné koncepty metaheuristik s populacemi

14 Evoluční algoritmy

15 Optimalizace mravenčí kolonie

Zdroj: *El-Ghazali Talbi, Metaheuristics: From Design to Implementation. Wiley, 2009.* `http://www.lifl.fr/~talbi/metaheuristic/`



Metaheuristiky s populacemi (population-based metaheuristics)

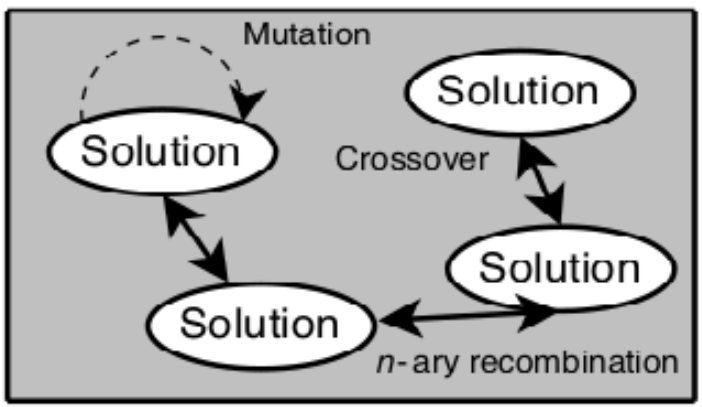
Genetické algoritmy, evoluční strategie, genetické programování,
optimalizace mravenčí kolonie (ant colony optimization, ACO),
optimalizace jedinců hejna (particle swarm optimization, PSO),
včelí úl (bee colony), umělé imunitní systémy (artificial immune systems),
odhad distribučními algoritmy (estimation of distribution algorithms, EDA),
...

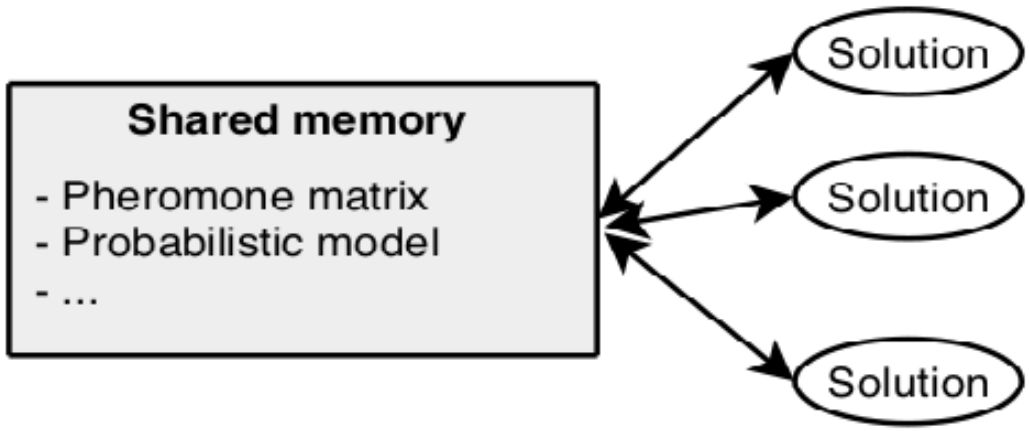
$P = P_0$; *(generuj počáteční populaci)*
$t = 0$;
repeat
    generuj(P'_t); *(generuj novou populaci)*
    P_{t+1} = vyber_populaci($P_t \cup P'_t$); *(vyber novou populaci)*
    $t = t + 1$;
until splněna podmínka ukončení
výstup: nejlepší nalezené/á řešení

Populační metaheuristiky: evoluce vs. paměť

Základní rozdělení algoritmů

- algoritmy využívající evoluci
  - řešení v populaci vybírána a reprodukována s pomocí operátorů (zejména: mutace a křížení)
  - př. genetické algoritmy, evoluční strategie
- algoritmy s pamětí (blackboard)
  - řešení v populaci se účastní na konstrukci paměti, pomocí níž se vytváří noví jedinci
  - př. feromonová matice (ACO), pravděpodobnostní model učení (EDA)





Společné koncepty metaheuristik s populacemi: generování počáteční populace

Náhodné generování

Sekvenční diversifikace

- řešení generována postupně s maximální odlišností
- př. simple sequential inhibition (SSI) process
  - každé následující řešení generováno tak, aby vzdálenost od všech předchozích řešení byla minimálně Δ
  - výpočetně náročné

Paralelní diversifikace

- řešení generována nezávisle paralelně se snahou o celkovou maximální odlišnost řešení v populaci
- může být obtížnější než řešení původního problému!

Heuristická inicializace

- jednotlivá řešení generována libovolnými heuristickými algoritmy (např. lokální prohledávání)
- nebezpečí v malé odlišnosti řešení v populaci

Společné koncepty: podmínky ukončení

Statická procedura

- konec prohledávání předem znám
- př. pevný počet iterací

  limit na CPU zdroje,

  maximální počet vyhodnocení účelové funkce

Adaptivní procedura

- konec prohledávání předem neznámý
- př. pevný počet iterací bez zlepšení (populace),

  vypočítáno dostatečně kvalitní řešení,

  malá odlišnost řešení v populaci

Základní koncepty evolučních algoritmů

Reprezentace

- populace: množina řešení (cca 20-100)
- chromozom/jedinec: zakódované řešení
- gen: rozhodovací proměnná v rámci řešení
- alely: možné hodnoty rozhodovací proměnné

Vhodnost (fitness)

- používaný termín pro účelovou funkci

Strategie výběru

- rodičů (řešení) pro vytváření další populace

Strategie reprodukce

- křížení a mutace: operace vytvářející nové jedince (potomky)

Strategie náhrady

- výběr jedinců do nové populace

Kostra evolučních algoritmů

generování(P_0); *(generuj počáteční populaci)*
$t = 0$;
while není splněna podmínka ukončení **do**
    vyhodnocení(P_t);
    P'_t = výběr(P_t); *(strategie výběru)*
    P'_t = reprodukce(P'_t); *(strategie reprodukce)*
    vyhodnocení(P'_t);
    P_{t+1} = nahrazení(P_t, P'_t); *(strategie náhrady)*
    $t = t + 1$;
end while
výstup: nejlepší nalezené/á řešení

Evoluční algoritmy

Na různých školách se vyvíjely různé typy evolučních algoritmů:

Genetické algoritmy (Holland, Michigan, USA)

Evoluční strategie (Rechenberg & Schwefel, Berlín, Německo)

Evoluční programování (Fogel, San Diego, USA)

- spojitá optimalizace
- menší použití pro velkou podobnost s evolučními strategiemi

Genetické programování (Koza, Stanford, USA)

- jedinci jsou programy (nelinerální reprezentace založená na stromech)
- automatické generování programů řešících danou úlohu
- př. nalezení programu odpovídajícího matematické rovnici
  minimalizace odchylek součtů čtverců od testovacích bodů

Genetické algoritmy

- Autor: Holland, Michigan, USA, sedmdesátá léta
- Prvotní reprezentace: binární
  - dnes i další typy
- Typicky použití operátoru křížení nad dvěma řešeními
- Mutace využívána pro zlepšení diversifikace
- Pevná pravděpodobnost mutace p_{m} a křížení p_{c}
- Náhrada bývá generační: rodiče systematicky nahrazeni potomky

Evoluční strategie (ES)

- Autoři: Rechenberg & Schwefel, Berlín, Německo, 1964
- Většinou pro: spojité optimalizace, vektory reálných hodnot
- Křížení využito zřídka
- Obvykle: náhrada nejlepšími jedinci (elitismu)
- Populace rodičů velikosti μ, populace potomků velikosti $\lambda \geq \mu$

---

inicializace populace μ jedinců
vyhodnocení μ jedinců
repeat generuj λ potomků z μ rodičů
vyhodnocení λ potomků
náhrada populace μ jedinců rodiči a potomky
until splněna podmínka ukončení
výstup: nejlepší jedinec nebo nalezená populace

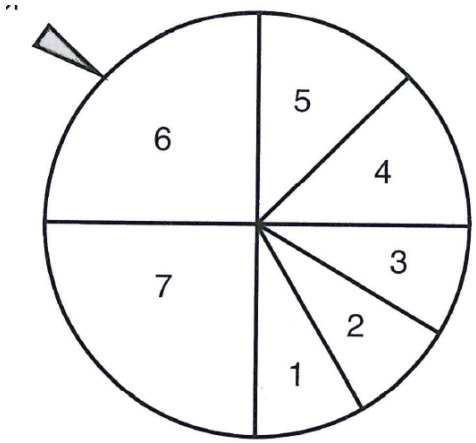
- $(1 + 1)$-ES: jednoduchá verze, 1 rodič nahrazen 1 potomkem
- $(\mu + \lambda)$-ES
  - opakuj λ-krát: (náhodný výběr rodiče, vygenerován potomek)
  - seřazení μ rodičů a λ potomků dle vhodnosti, náhrada nejlepšími
- (μ, λ)-ES: pro novou populaci se vybírá jen mezi λ potomky

Výběr ruletovým kolem (roulette wheel selection)

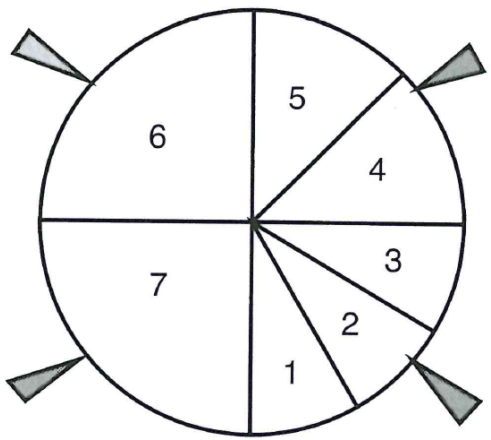
Výběr ruletovým kolem

- nejpoužívanější strategie výběru
- f_i vhodnost jedince i v populaci
- pravděpodobnost výběru jedince dána jako $p_i = f_i/(\sum_{j=1}^{n} f_j)$
- analogie: ruletové kolo s díly pro všechny jedince v populaci
  velikost dílu ruletového kola pro jedince odpovídá p_i
- Problémy:
  příliš velká snaha vybírat kvalitní jedince $\Rightarrow$ předčasná konvergence

Jedinci	1	2	3	4	5	6	7
Vhodnost	1	1	1	1.5	1.5	3	3



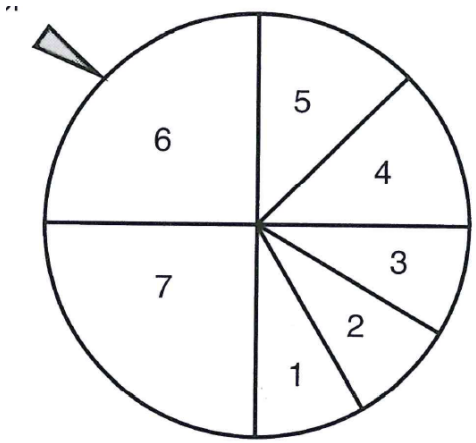
Roulette selection



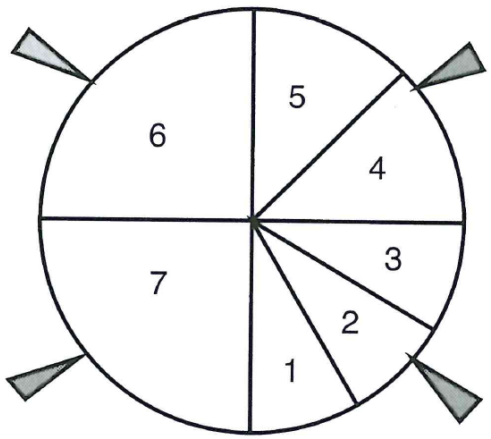
Stochastic universal sampling

Pravděpodobnostní univerzální vzorkování (stochastic universal sampling)

Jedinci	1	2	3	4	5	6	7
Vhodnost	1	1	1	1.5	1.5	3	3



Roulette selection



Stochastic universal sampling

Pravděpodobnostní univerzální vzorkování (řeší problémy rulet.kola)

- u ruletového kola dáme μ rovnoměrně rozložených ukazatelů
- jedno otočení ruletového kola vybírá μ jedinců

Turnajový výběr, výběr rankováním

Turnajový výběr

- náhodný výběr k jedinců
- turnaj: z těchto k jedinců je výbrán nejlepší jedinec
- pro výběr μ jedinců aplikujeme turnaj μ-krát

Výběr rankováním (rank-based selection)

- pro každého jedince spočítán rank a dle něj jsou vybíráni jedinci
- rank může např. škálovát linerárně se závislostí na snaze o výběr nejlepšího jedince
- rank použit pro výpočet pravděpodobnosti a aplikován stejně jako u ruletového kola

Mutace

Vlastnosti mutace

- operátor mutace mění jedince v populaci a způsobí jeho malou změnu
- pravděpodobnost mutace genu $p_{\mathrm{m}} \in [0.001, 0.01]$
- př. inicializace p_{m} na $1/k$, kde k je počet genů (rozhodovacích proměnných), tj. v průměru zmutovaná 1 proměnná

Mutace v binární reprezentaci

- prohození (flip) hodnoty binární proměnné

Mutace v diskrétní reprezentaci

- změna hodnoty prvku za jinou hodnotu v abecedě

Mutace v permutacích

- vložení, výměna, inverze hodnot(y)
- př. viz permutační okolí pro rozvrhovací problémy (1.přednáška)

Křížení

Vlastnosti křížení

- binární (někdy n-ární) operátor
- cíl: zdědit vlastnosti rodičů potomkem
- pravděpodobnost křížení rodičů $p_c \in [0, 1]$, běžně $p_c \in [0.45, 0.95]$

Linerární reprezentace (vyjma permutací)

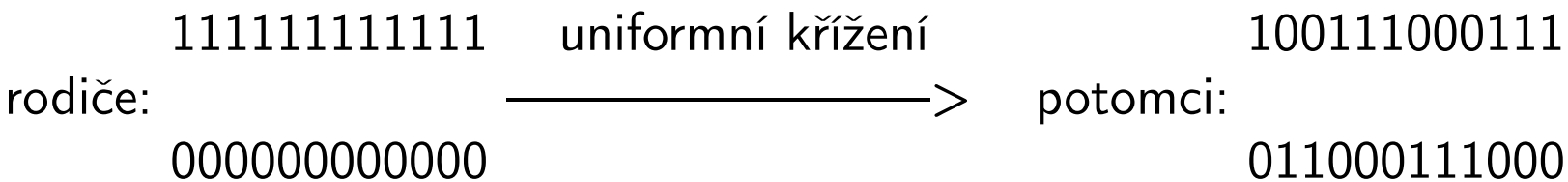
- 1-bodové křížení (1-point crossover)
  - podle vybrané pozice k v potomcích prohozeny hodnoty dvou rodičů

- 2-bodové (a n-bodové) křížení
  - vybrány dvě (n) pozice a provedeno prohození hodnot

Křížení (pokračování)

Linerární reprezentace (vyjma permutací)

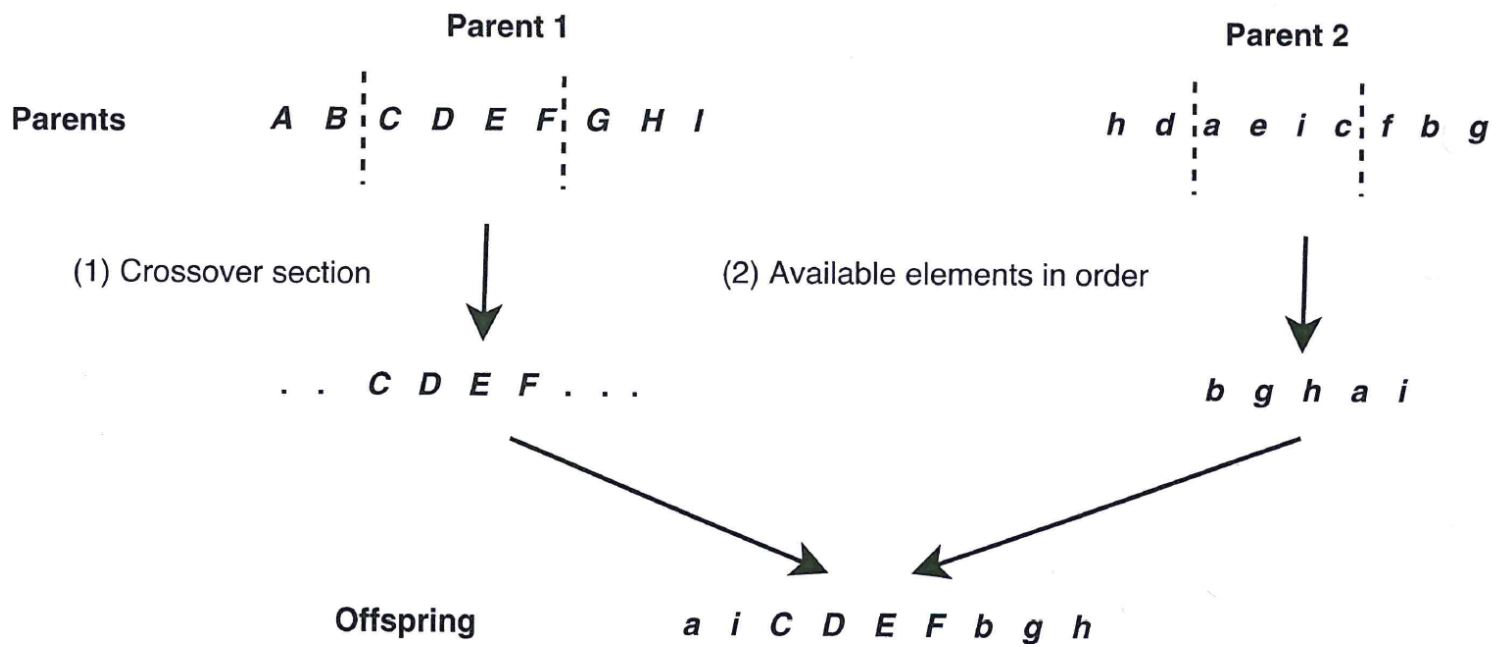
- uniformní křížení (uniform crossover)
  - jedinci kombinováni bez ohledu na velikost segmentů
  - každý gen potomka náhodně vybrán z rodiče
  - každý rodič rovnoměrně přispívá ke generování potomků stejně



Křížení (dokončení)

Reprezentace permutacemi

- křížení je složitější, jedince nelze takto jednoduše kombinovat, protože každá alela (hodnota) se musí výskytnout v jedinci právě jednou (používány různé formy mapování)
- Křížení dané pořadím (Order crossover, OX)
  - vybrány náhodně dva body křížení
  - z rodiče 1 hodnoty mezi nimi zkopírovány na stejné pozice v potomkovi
  - z rodiče 2 začneme od druhého bodu křížení vybírat prvky, které již nebyly vybrány z rodiče 1, a dávame je do potomka od 2. bodu křížení



Cvičení: strategie reprodukce

Plánování úloh na jednom stroji bez překrytí v čase

- každá úloha má určen: nejdřívější čas provádění, dobu provádění
- minimalizujeme čas dokončení poslední úlohy

Reprezentace jedince

- permutace úloh určuje pořadí provádění na stroji
  - pozor: nejdřívější čas příchodu nutno dodržet

Křížení

- 2 jedinci: 12|34567|89, 13|57924|68
- potomci pomocí OX křížení: 365792481, 923456781

Mutace

- vložení úlohy 123456789 $\rightarrow$ 124567389
- vyměna úloh 123456789 $\rightarrow$ 127456389

Strategie náhrady

Vybíráme mezi rodiči a potomky další populaci

Extrémní strategie náhrady (nepříliš používané)

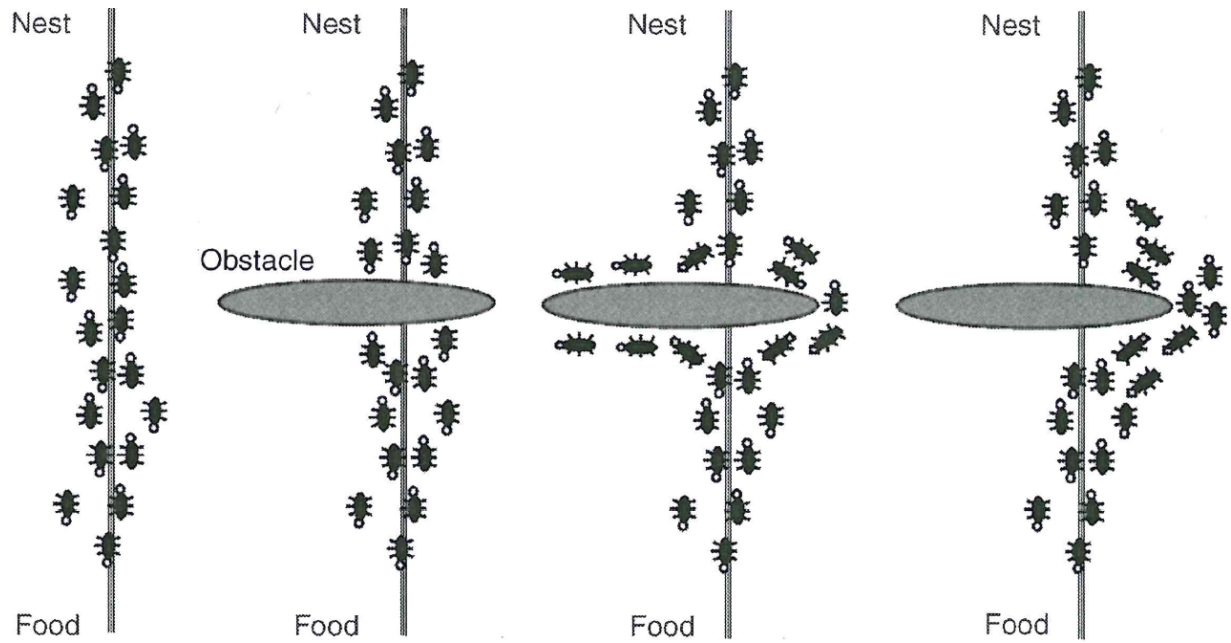
- úplná náhrada (generational replacement)
  - potomky bude nahrazena systematicky celá populace rodičů
  - ztrácíme i nejlepší jedince
- náhrada jednotlivce (steady-state replacement)
  - bude vytvořen pouze jediný potomek, který nahradí např. nejhoršího jedince populace
  - tj. degradace genetických algoritmů, otázka smyslu práce s populací

Používány strategie na pomezí mezi těmito krajními přístupy, např.

- náhrada pevného množství jedinců
  - pro náhradu vybíráno λ jedinců populace při velikosti populace μ
  - $1 < \lambda < \mu$
- elitářský model
  - výběr nejlepších jedinců mezi rodiči a potomky
  - rychlá avšak předčasná konvergence

Inteligence hejna (swarm intelligence)

- Algoritmy inspirované skupinových chováním druhů jako jsou mravenci, včely, vosy, termiti, ryby nebo ptáci
- Původ v sociální chování těchto druhů při hledání potravy
- Základní charakteristika algoritmů
  - jedinci jsou jednodušší nesofistikovaní agenti
  - jedinci kooperují nepřímo pomocí média



Optimalizace mravenčí kolonie (Ant Colony Optimization, ACO)

- Myšlenky inspirující algoritmus
  - mravenčí kolonie schopna najít nejkratší cestu mezi dvěma body
  - mravenci během cesty nechávají na zemi chemickou stopu (feromony)
  - feromony vedou mravence v cíli
  - feromony se postupně vypařují
- Algoritmus
  - inicializace feromonů
  - iterace: konstrukce řešení mravencem, aktualizace feromonů

---

inicialiace feromonové stopy;
repeat
  for každého mravence **do**
    konstrukce řešení pomocí feromonové stopy;
    (aktualizace feromonové stopy:)
      vypařování;
      zesílení feromonové stopy;
until splněna podmínka ukončení
výstup: nejlepší nalezené řešení nebo množina řešení

Optimalizace mravenčí kolonie (pokračování)

Feromonové informace

- τ typicky jako matice/vektor hodnot obsahující feromonovou stopu
- př. matice jako reprezentace grafu obsahuje feromony na hranách

Vypařování feromonů

- $\tau_{ij} = (1 - \rho)\tau_{ij}$ realizuje pro každé i, j vypařování feromonů
- $\rho \in [0, 1]$

Zesilování feromonů

- online aktualizace: τ_{ij} aktualizováno v každém kroku
- online pozdržená aktualizace:
  - τ aktualizováno po nalezení úplného řešení *jedním* mravencem
    - př. čím lepší řešení, tím více feromony zesíleny
- off-line aktualizace:
  - τ aktualizováno po nalezení úplného řešení *všemi* mravenci
  - nejpopulárnější přístup
  - př. aktualizace feromonů dle kvality
    - feromony aktualizovány dle nejlepšího (nebo několika nejlepších) řešení
    - $\forall i, j$ v řešení: $\tau_{i,j} = \tau_{i,j} + \Delta$

Optimalizace mravenčí kolonie pro problém obchodního cestujícího

---

inicialiace feromonové stopy;
repeat
  for každého mravence **do**
    (konstrukce řešení pomocí feromonové stopy)
    $S = \{1, 2, \ldots, n\}$; *(množina měst na výběr)*;
    náhodně vyber město i; $S = S - \{i\}$;
    repeat
      vyber město j s pravděpodobnosti p_{ij};
      $S = S - \{j\}$; $i = j$;
    until $S = \emptyset$
  end for
  (aktualizace feromonové stopy)
  for $i, j \in [1, n]$ **do** $\tau_{ij} = (1 - \rho)\tau_{ij}$; *(vypařování)*
  for i, j v nejlepším řešení iterace **do** $\tau_{ij} = \tau_{ij} + \Delta$; *(zesílení feromonů)*
until splněna podmínka ukončení
výstup: nejlepší nalezené řešení nebo množina řešení

Pravděpodobnost p_{ij} výběru dalšího města na cestě

Základní výpočet pravděpodobnosti:

$$p_{ij} = \frac{\tau_{ij}}{\sum_{k \in S} \tau_{ik}} \quad \forall j \in S$$

- př. jsme ve městě 1 ($i = 1$) a můžeme pokračovat do měst $S = \{2, 3, 4, 5, 6\}$
  $\tau_{12} = 3, \tau_{13} = 2, \tau_{14} = 2, \tau_{15} = 1, \tau_{16} = 2$
  $p_{12} = \frac{3}{3+2+2+1+2} = 0.3,\ p_{13} = \frac{2}{3+2+2+1+2} = 0.2, \ldots$

Problémově závislá heuristika:

- využití hodnot $\eta_{ij} = 1/d_{ij}$, kde d_{ij} udává vzdálenost mezi městy i, j

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \times \eta_{ij}^{\beta}}{\sum_{k \in S} \tau_{ik}^{\alpha} \times \eta_{ik}^{\beta}} \quad \forall j \in S$$

  kde α a β určují relativní vliv feromonové hodnoty a heuristické hodnoty η
  - $\alpha = 0$: stochastický hladový algoritmus
  - $\beta = 0$: základní výpočet pravděpodobností pouze pomocí feromonů

Cvičení: ACO

Aplikujte ACO algoritmus na problém obchodního cestujícího s n městy se vzdálenosti mezi nimi určenou úplným hranově ohodnoceným grafem.

Předpokládejte, že

- pracujete s m mravenci
- iniciální feromonová stopa je matice, jejíž hodnoty jsou nepřímo úměrné vzdálenosti mezi městy,
- při náhodném výběru měst vyberte vždy město s nejmenším indexem,
- koeficient vypařování je $\rho = 0.01$,
- koeficient pro zesílení feromonů je $\Delta = 1$

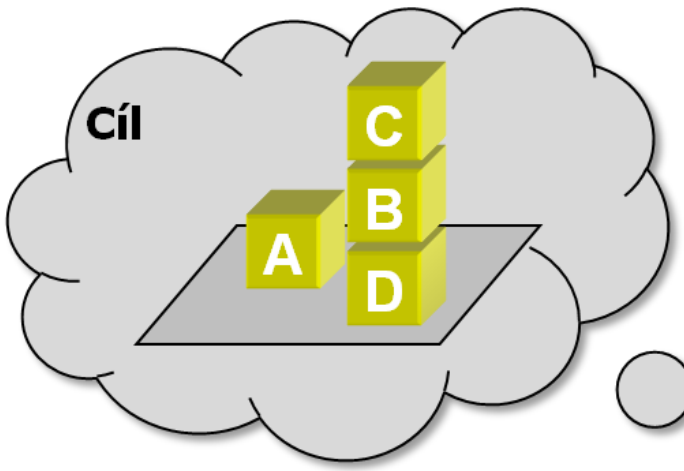
a aplikujte k kroků algoritmu.

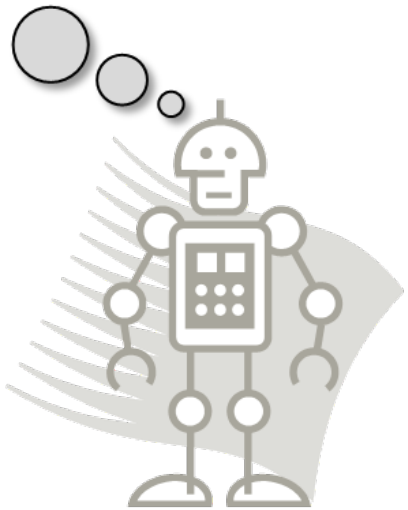
Plánování: reprezentace problému

8. ledna 2021

Zdroj: *Roman Barták, přednáška Plánování a rozvrhování, Matematicko-fyzikální fakulta, Karlova univerzita v Praze, 2014.*
http://kti.ms.mff.cuni.cz/~bartak/planovani

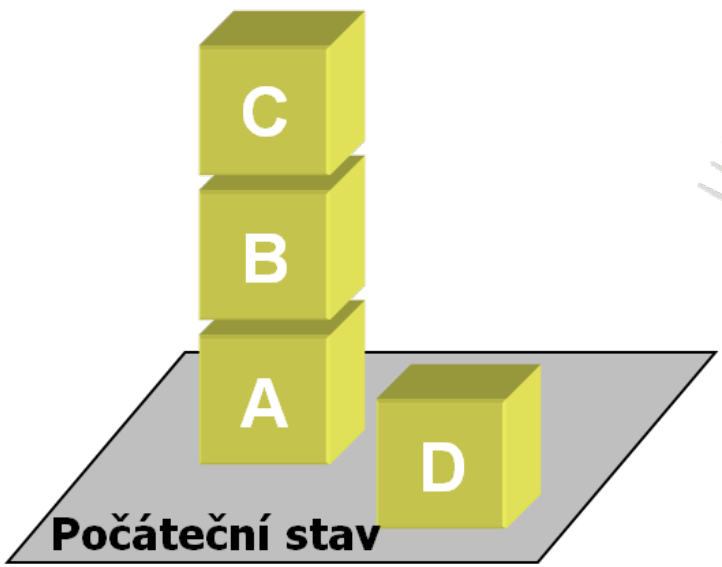
Plánování: příklad

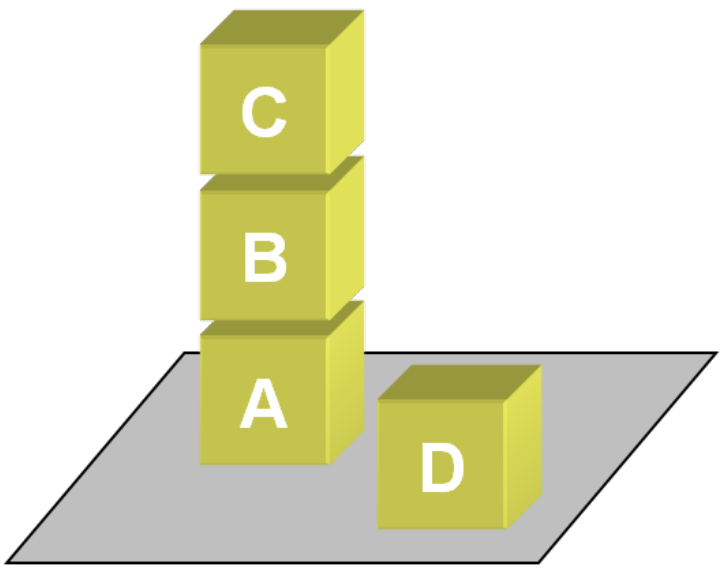




Plán:

zvedni(C)
polož_na(C,stůl)
zvedni(B)
polož_na(B,D)
zvedni(C)
polož_na(C,B)





Plánování v kostce

Vstup:

- počáteční (současný) stav světa
- popis akcí schopných měnit stav světa
- požadovaný stav světa

Výstup:

- seznam akcí (plán)

Příklady problémů a velikost

- řízení výtahu: $6.92 \cdot 10^{19}$ dosažitelných stavů
- automatizace skleníku: $1.68 \cdot 10^{21}$ dosažitelných stavů
- logistika dopravy: $6.31 \cdot 10^{218}$ dosažitelných stavů

Plánování v kostce: klasické plánování

- Stavové proměnné

  $x \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\},\ \mathtt{y} \in \{\mathtt{a}, \mathtt{b}\},\ \mathtt{z} \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$
- Počáteční stav

  $\{x \to \mathtt{a}, \mathtt{y} \to \mathtt{a}, \mathtt{z} \to \mathtt{a}\}$
- Cílový stav

  $\{x \to \mathtt{c}, \mathtt{z} \to \mathtt{b}\}$
- Akce a.k.a. operátory

  $a_1 : \quad x \to \mathtt{a},\ \mathtt{z} \to \mathtt{c} \quad \overset{4}{\to} \quad \mathtt{z} := \mathtt{b}$ *(akce s cenou 4)*

  $a_2 : \quad x \to \mathtt{a},\ \mathtt{y} \to \mathtt{a} \quad \overset{3}{\to} \quad \mathtt{y} := \mathtt{b},\ \mathtt{z} := \mathtt{c}$

  ...

Problém

- nalézt posloupnost akcí, které transformují počáteční stav na stav, který je konzistentní s cílovým stavem
- účelová funkce: součet ceny akcí

Plánování v kostce:
STRIPS plánování (varianta klasického plánování)

- Všechny proměnné mají doménu $\{\mathbf{T}, \mathbf{F}\}$
- V podmínce akce a v cílovém stavu pouze $v \to \mathbf{T}$
- Notace

  $a: \quad x \to \mathbf{T},\ y \to \mathbf{T} \quad \overset{5}{\to} \quad w := \mathbf{F},\ y := \mathbf{F},\ z := \mathbf{T}$

  zapsaná jako

  $\text{precond}(a) = \{x, y\},\ \text{effects}^+(a) = \{z\},\ \text{effects}^-(a) = \{w, y\},$
  $\text{cost}(a) = 5$

- Příklad: zvedneme ležící kostku `B` nahoru

  $\text{precond}(\texttt{zvedni-B}) = \{\texttt{lezi-B}\},$

  $\text{effects}^+(\texttt{zvedni-B}) = \{\texttt{nahore-B}\},$

  $\text{effects}^-(\texttt{zvedni-B}) = \{\texttt{lezi-B}\}$

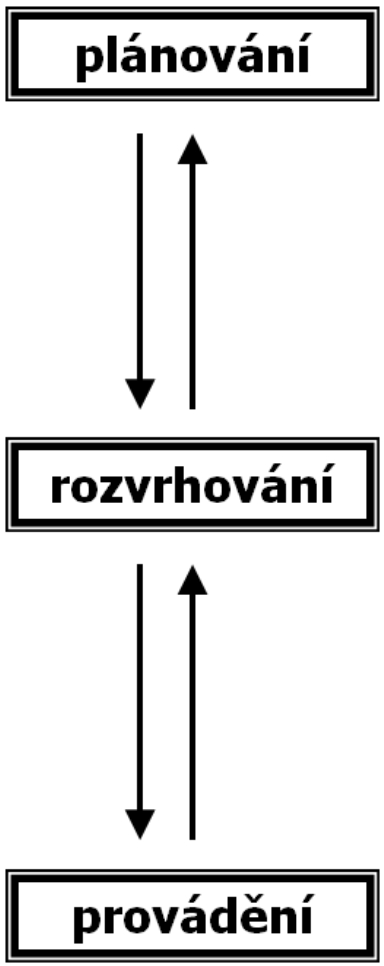
Plánování a rozvrhování

Plánování (planning)

- rozhodování, jaké akce jsou potřeba pro dosažení daných cílů
- složitost: často horší než NP-úplné
- rozhodnutelnost?

Rozvrhování (scheduling)

- rozhodování, jak zpracovat dané akce použitím omezených zdrojů a času
- složitost: typicky NP-úplné



Plánování: rozhodnutelnost

Problémy

1. PlanSAT: Existuje plán, který řeší zadaný plánovací problém?
2. BoundedPlanSAT: Existuje plán délky k nebo kratší, který řeší zadaný plánovací problém?

Oba problémy rozhodnutelné pro klasické plánování

$\Leftarrow$ počet stavů je konečný

Přidání funkčních symbolů do jazyka

- počet stavů je nekonečný
- PlanSAT částečně rozhodnutelný
  - existuje algoritmus, který skončí pro řešitelné problémy
  - pro neřešitelné problémy nemusí skončit
- BoundedPlanSAT zůstává rozhodnutelný

Plánování: obsah

Klasické plánování

- Konceptuální model
- Reprezentace problému
- Plánovací algoritmy
  - plánování se stavovým prostorem
  - plánování s prostorem plánů

Formalizace: konceptuální model

Plánování se zabývá volbou a organizací akcí, které mění stav systému

Systém Σ modelující stavy a přechody:

- množina stavů S (rekurzivně spočetná)
- množina akcí A (rekurzivně spočetná)
  - plánovač kontroluje akce!
  - no-op (prázdná akce)
- množina událostí E (rekurzivě spočetná)
  - událost mimo kontrolu plánovače!
  - neutrální událost ε
- přechodová funkce $\gamma : S \times A \times E \to P(S)$
  - někdy se akce a události aplikují odděleně $\gamma : S \times (A \cup E) \to P(S)$

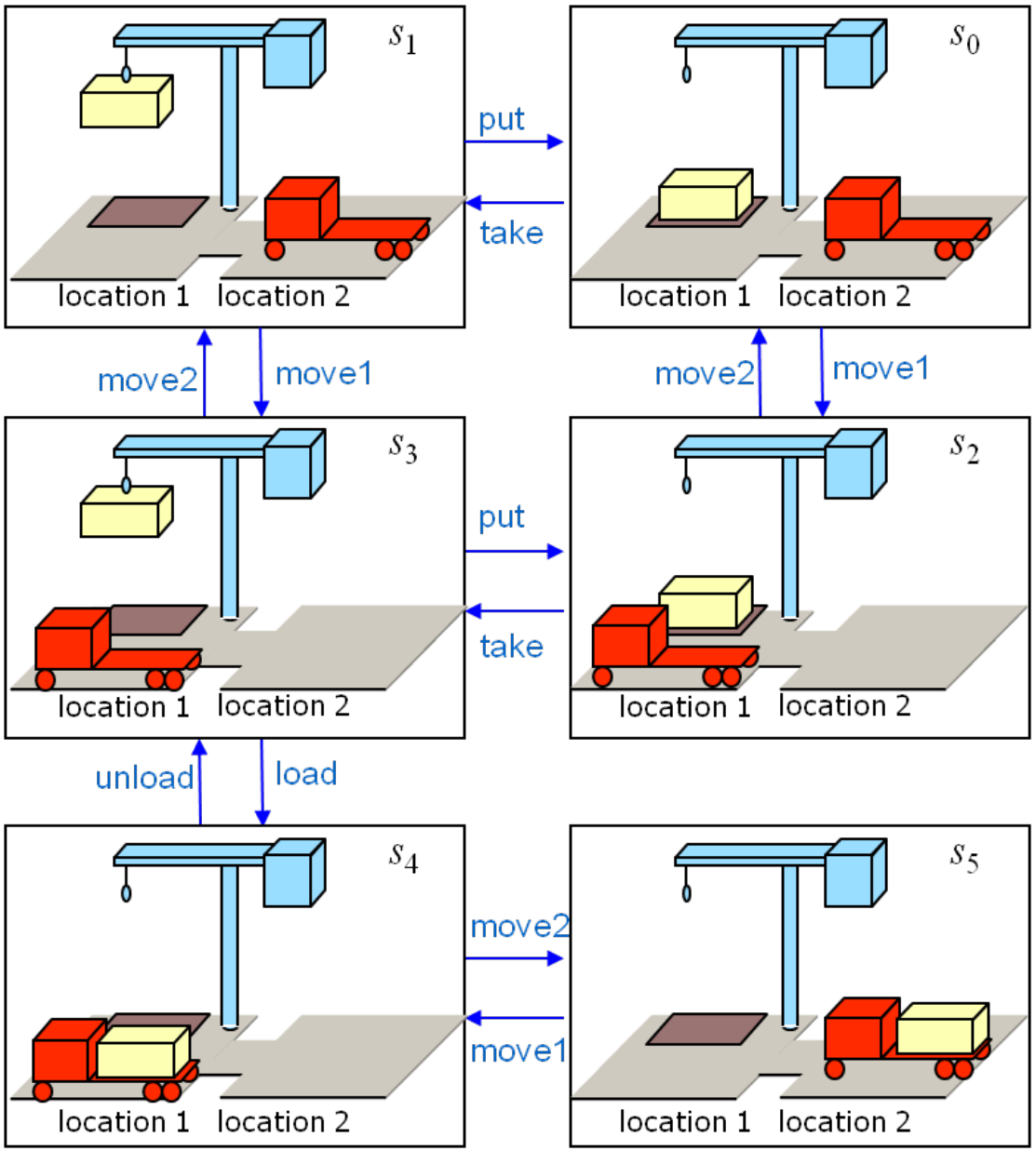
Cíle plánování

Cílem plánování je zjistit, jaké akce a na které stavy se mají aplikovat, abychom za dané situace dosáhli požadovaných cílů.

Co je cílem plánování?

- cílový stav nebo množina cílových stavů
- splnění dané podmínky nad posloupností stavů, přes které systém prochází
  - např. stavy, kterým se vyhnout, nebo stavy, které se musí navštívit
- optimalizace dané objektivní funkce nad posloupností stavů
  - např. maximum nebo součet ohodnocení stavů

Příklad



$\Sigma = (S, A, E, \gamma)$

- $S = \{s_0, \ldots, s_5\}$
- $E = \{\}$ resp. $\{\varepsilon\}$
- $A = \{$move1, move2, put, take, load, unload$\}$
- γ: obrázek

počáteční stav: s_0

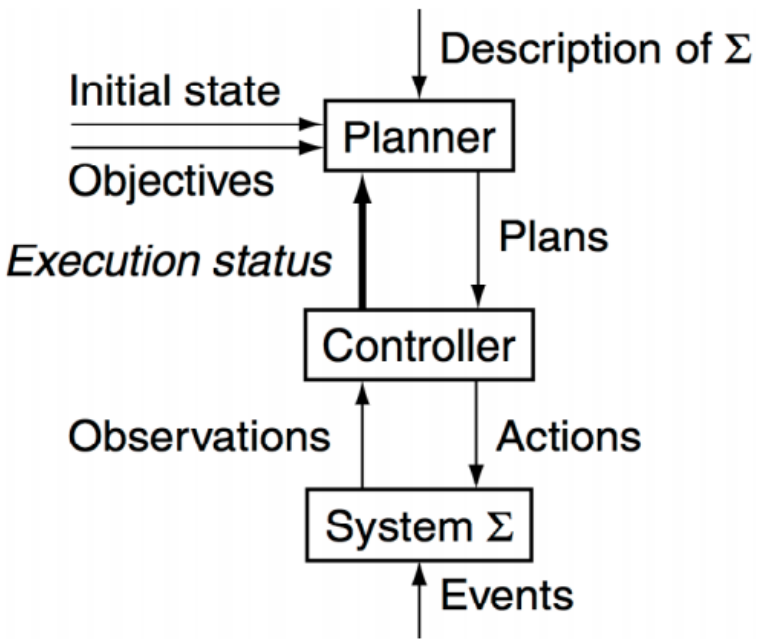
cíl: s_5

Jak to funguje?

Plánovač generuje plány

Řadič se stará o jejich realizaci

- pro daný stav určí akci k provedení
- pozorování převádějí reálný stav na modelovaný stav



Dynamické plánování

umožňuje přeplánování na základě aktuálního stavu provádění

Zjednodušení modelu

- Systém je konečný
- Systém je „plně pozorovatelný"
  - máme úplné informace o stavu systému
- Systém je deterministický
  - $\forall s \in S \, \forall u \in (A \cup E) : |\gamma(s, u)| \leq 1$
- Systém je statický
  - množina událostí je prázdná, tj. $E = \emptyset$
- Cíle jsou omezené
  - cílem je dosažení některého stavu z množiny cílových stavů
- Plány jsou sekvenční
  - plánem je úplně uspořádaná posloupnost akcí
- Čas je implicitní
  - akce i události jsou instantní (okamžité, tj. nemají žádné trvání)
- Plánujeme offline
  - stav systému se nemění v průběhu plánování

Klasické plánování (STRIPS plánování)

Pracujeme s deterministickým, statickým, konečným a plně pozorovatelným stavovým modelem s omezenými cíli a implicitním časem $\Sigma = (S, A, \gamma)$.

Plánovací problém $P = (\Sigma, s_0, g)$:

- s_0 je počáteční stav
- g charakterizuje cílové stavy

Řešením plánovacího problému P je

- posloupnost akcí $\langle a_1, a_2, \ldots, a_k \rangle$

odpovídající

- posloupnosti stavů $\langle s_0, s_1, \ldots, s_k \rangle$

takové, že

1. $s_i = \gamma(s_{i-1}, a_i)$,
2. s_k splňuje g

Zjednodušení?

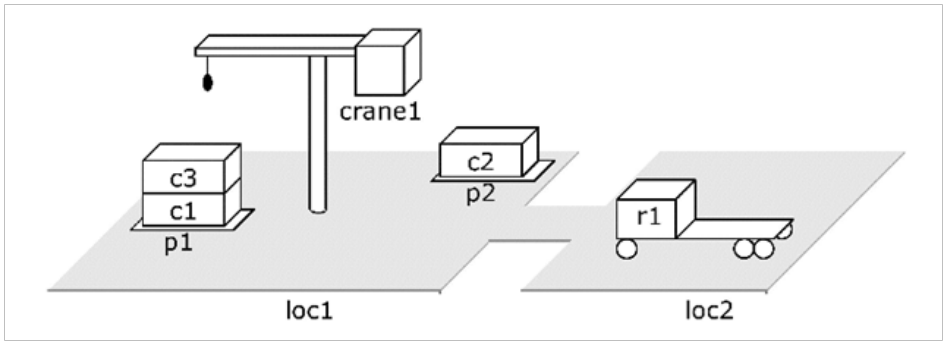
Plánování ve zjednodušeném modelu je „pouhé" hledání cesty v grafu.

Je to opravdu tak jednoduché?

- 5 míst, 3 hromady na každém místě, 100 kontejnerů, 3 roboti

$\Rightarrow$ 10^{277} stavů,

tj. 10^{190} krát více než jsou největší odhady počtu částic ve vesmíru



Co tedy potřebujeme?

- Jak tedy reprezentovat stavy a akce tak, aby nebylo třeba vyjmenovat množiny A a S?
  - nemůžeme přímo pracovat s 10^{277} stavy ...
- Jak efektivně hledat řešení plánovacího problému?
  - Jak najít cestu v grafu s 10^{277} uzly?

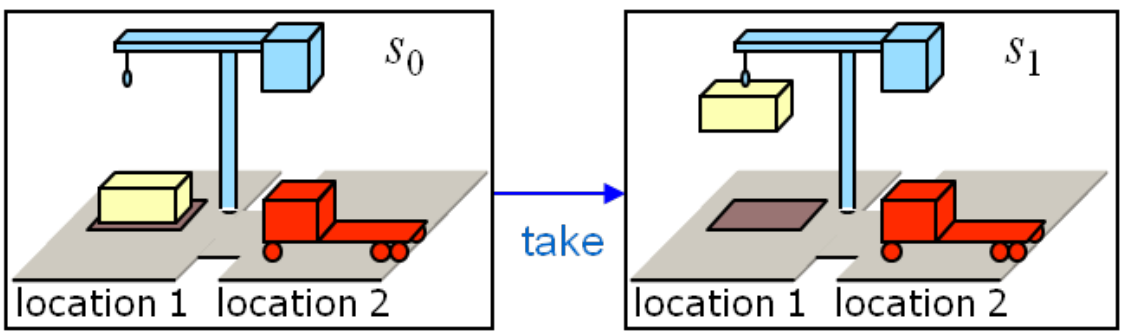
Klasické plánování: množinová reprezentace

Stav systému je popsán množinou výroků

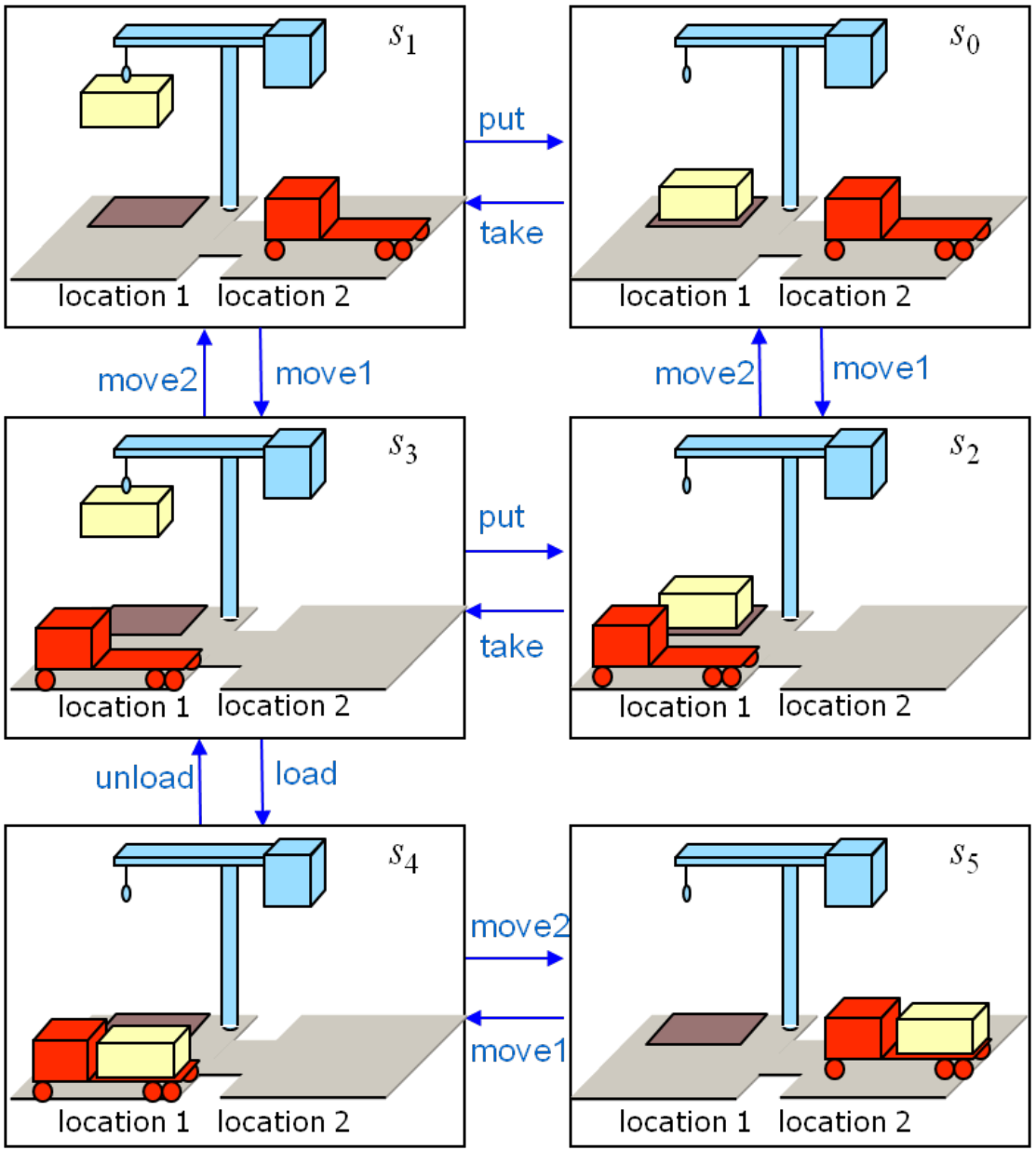
- např. {onground, at2}

Každá akce je syntaktický výraz specifikující:

- jaké výroky musí patřit do stavu, aby na něj byla akce aplikovatelná
  - např. take: {onground}
- jaké výroky akce přidá a jaké výroky smaže, aby vytvořila nový stav
  - např. take: $\{\text{onground}\}^-$, $\{\text{holding}\}^+$



Množinová reprezentace: příklad



$L = \{\text{onground, onrobot, holding, at1, at2}\}$

$s_0 = \{\text{onground, at2}\}$

$g = \{\text{onrobot}\}$

load = ({holding, at1},
{holding},
{onrobot})

⟨take, move1, load, move2⟩ je plán, ale není minimální (není nutné move2, v g není podmínka na location)

Pozor na velikost množinové reprezentace

Klasická reprezentace

Klasická reprezentace zobecňuje množinovou reprezentaci

směrem k predikátové logice:

- stavy jsou množiny logických atomů (místo výroků u množ.rep.), které jsou v dané interpretaci buď pravda nebo nepravda
- akce jsou reprezentovány plánovacími operátory, které mění pravdivostní hodnotu těchto atomů

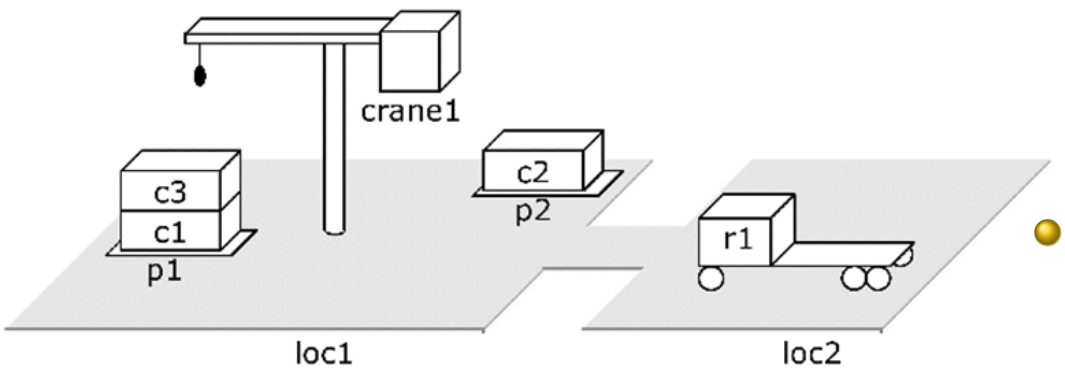
Přesněji:

- L (jazyk) je konečná množina predikátových symbolů a konstant (nemáme funkce!)
- atom je predikátový symbol s argumenty
  - např. on(c3, c1)
- můžeme používat proměnné
  - např. on(x,y)

Klasická reprezentace: reprezentace stavů

Stav je množina instanciovaných atomů (bez proměnných).

Opět jich je konečně mnoho!



{attached(p1,loc1), in(c1,p1), in(c3,p1), top(c3,p1), on(c3,c1), on(c1,pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), empty(crane1), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)}.

- Pravdivostní hodnota některých atomů se mění
  - flexibilní atomy (fluent)
  - např. at(r1,loc2)
- Některé atomy nemění svoji pravdivostní hodnotu s různými stavy
  - neměnné atomy (rigid)
  - např. adjacent(loc1,loc2)

Předpoklad uzavřeného světa (closed world assuption):

atom, který není ve stavu explicitně uveden, neplatí!

Klasická reprezentace: plánovací operátory

Operátor o je trojice: (name(o), precond(o), effects(o))

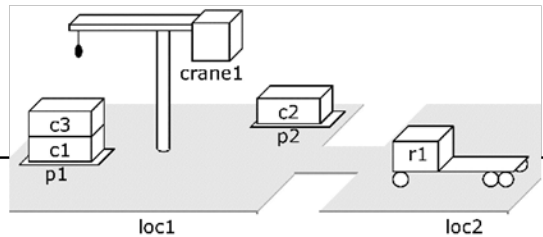
- name(o): jméno operátoru ve tvaru $n(x_1, \ldots, x_k)$
  - n: symbol operátoru (jednoznačný pro každý operátor)
  - $x_1, \ldots, x_k$: symboly proměnných (parametry operátoru)
    - musí obsahovat všechny symboly proměnných v operátoru!
- precond(o): předpoklady
  - literály, které musí být splnitelné, aby šlo operátor použít
- effects(o): důsledky
  - literály, které se stanou pravdivými aplikací operátoru
    - nesmí být neměnné atomy!

take(k, l, c, d, p)
;; crane k at location l takes c off of d in pile p
precond: belong(k, l), attached(p, l), empty(k), top(c, p), on(c, d)
effects: holding(k, c), $\neg$ empty(k), $\neg$ in(c, p), $\neg$ top(c, p), $\neg$ on(c, d), top(d, p)

Klasická reprezentace: akce

Akce jsou plně instanciované operátory

– za proměnné jsou dosazeny konstanty



take(k, l, c, d, p)

;; crane k at location l takes c off of d in pile p

precond: belong(k, l), attached(p, l), empty(k), top(c, p), on(c, d)

effects: holding(k, c), $\neg$ empty(k), $\neg$ in(c, p), $\neg$ top(c, p), $\neg$ on(c, d), top(d, p)

operator

take(crane1,loc1,c3,c1,p1)

;; crane crane1 at location loc1 takes c3 off c1 in pile p1

precond: belong(crane1,loc1), attached(p1,loc1),
empty(crane1), top(c3,p1), on(c3,c1)

effects: holding(crane1,c3), ¬empty(crane1), ¬in(c3,p1),
¬top(c3,p1), ¬on(c3,c1), top(c1,p1)

action

Klasická reprezentace: aplikace akce

Notace:

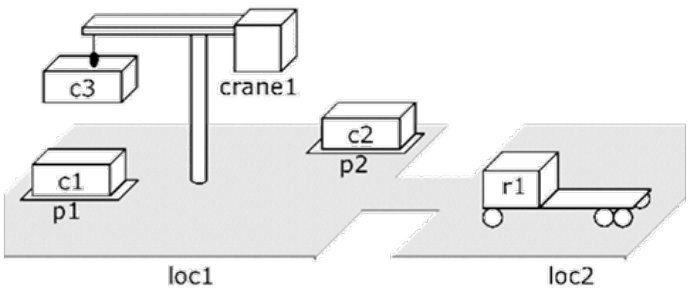
- $S^+ = \{\text{pozitivní atomy v } S\}$
- $S^- = \{\text{atomy, jejichž negace je v } S\}$

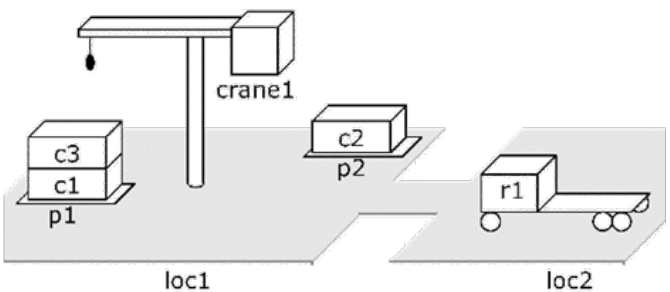
Akce a je použitelná na stav s právě když

$$\text{precond}^+(a) \subseteq s \quad \wedge \quad \text{precond}^-(a) \cap s = \emptyset$$

Výsledkem aplikace akce a na s je $\gamma(s, a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$

take(crane1,loc1,c3,c1,p1)
;; crane crane1 at location loc1 takes c3 off c1 in pile p1
precond: belong(crane1,loc1), attached(p1,loc1), empty(crane1), top(c3,p1), on(c3,c1)
effects: holding(crane1,c3), ¬empty(crane1), ¬in(c3,p1), ¬top(c3,p1), ¬on(c3,c1), top(c1,p1)





Klasická reprezentace: plánovací doména

Nechť L je jazyk a O je množina operátorů.

Plánovací doména Σ nad jazykem L a s operátory O je trojice (S, A, γ):

- stavy $S \subseteq P(\{\text{všechny instanciované atomy nad } L\})$
- akce $A = \{\text{všechny instanciované operátory z } O \text{ nad } L\}$
  - akce a je použitelná na stav s, pokud
    $\text{precond}^+(a) \subseteq s \quad \wedge \quad \text{precond}^-(a) \cap s = \emptyset$
- přechodová funkce γ:
  - $\gamma(s, a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$, je-li a použitelná na s
  - S je uzavřená vzhledem ke γ
    - pokud $s \in S$,
      potom pro každou akci a aplikovatelnou na s platí $\gamma(s, a) \in S$

Klasická reprezentace: plánovací problém

Plánovací problém P je trojice (Σ, s_0, g):

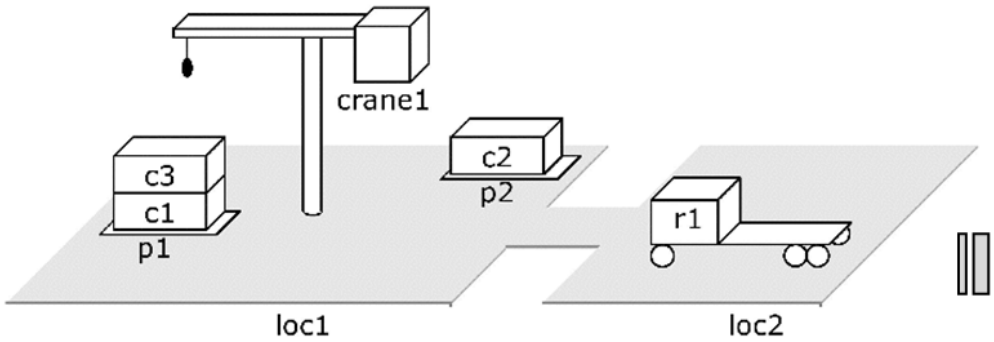
- $\Sigma = (S, A, \gamma)$ je plánovací doména
- s_0 je počáteční stav, $s_0 \in S$
- $g \subseteq L$ je množina instanciovaných literálů
  - stav s splňuje g právě tehdy, když $g^+ \subseteq s \ \wedge \ g^- \cap s = \emptyset$
  - $S_g = \{s \in S \mid s \text{ splňuje } g\}$ je množina cílových stavů

Zápis plánovacího problému je trojice (O, s_0, g)

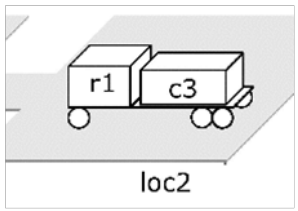
Plán π je posloupnost akcí $\langle a_1, a_2, \ldots, a_k \rangle$

Plán π je řešením P, právě když $\gamma(s_0, \pi)$ splňuje g

Klasická reprezentace: ukázka plánu



náš cíl

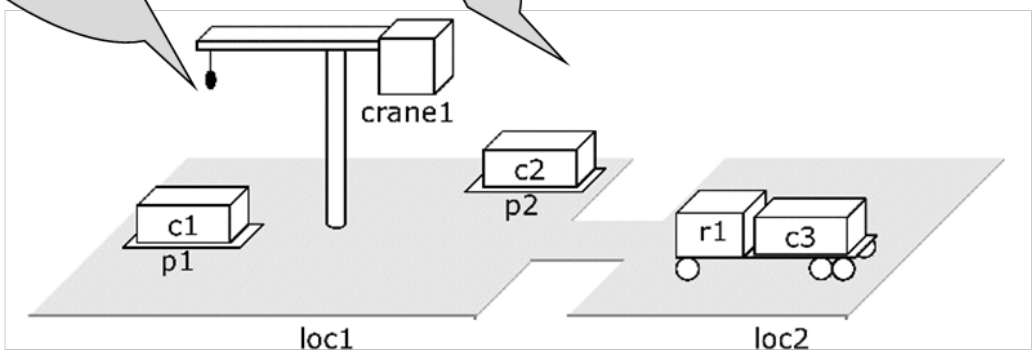


g = {loaded(r1,c3), at(r1,loc2)}

s_1 = {attached(p1,loc1), in(c1,p1), in(c3,p1), top(c3,p1), on(c3,c1), on(c1,pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), empty(crane1), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)}.

⟨move(r1,loc2,loc1),
take(crane1,loc1,c3,c1,p1),
load(crane1,loc1,c3,r1),
move(r1,loc1,loc2)⟩

⟨take(crane1,loc1,c3,c1,p1),
move(r1,loc2,loc1),
load(crane1,loc1,c3,r1),
move(r1,loc1,loc2)⟩



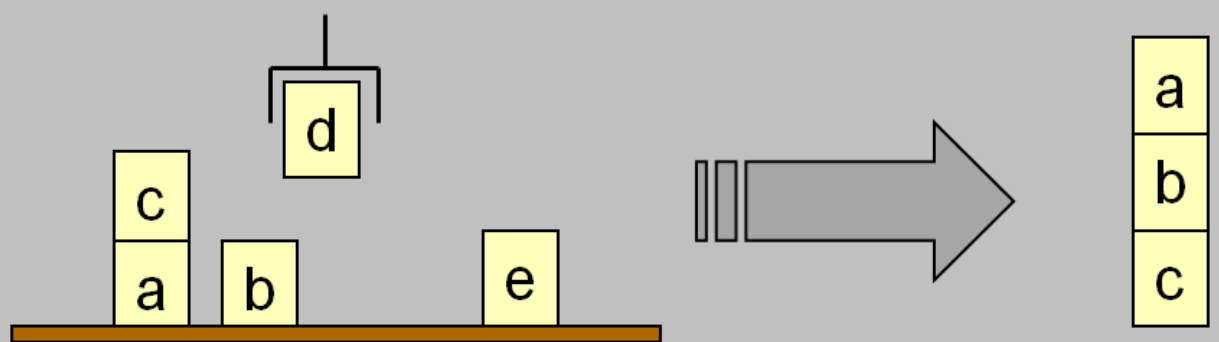
Cvičení

Navrhněte množinovou a klasickou reprezentaci pro svět kostek.

Svět kostek (the blocks world)

- nekonečně velký stůl, konečný počet kostek
- poloha kostky na stole nás nezajímá
- kostka může ležet buď na stole nebo na jiné kostce
- při plánování chceme přesouvat kostky tak, že v dané chvíli můžeme držet maximálně jednu kostku

Příklad:



Plánování se stavovým prostorem

8. ledna 2021

Zdroj: *Roman Barták, přednáška Plánování a rozvrhování, Matematicko-fyzikální fakulta, Karlova univerzita v Praze, 2014.*
http://kti.ms.mff.cuni.cz/~bartak/planovani

Plánování se stavy

Prohledávaný prostor odpovídá stavovému prostoru plánovacího problému

- uzly odpovídají stavům
- hrany odpovídají stavovým přechodům pomocí akcí
- cílem je najít cestu mezi počátečním stavem a některým koncovým stavem

Typy prohledávání

- dopředné (forward search)
- zpětné (backward search)
  - liftovaná verze
  - STRIPS
- problémově závislé (svět kostek)

Poznámka: algoritmy budeme uvádět pro klasickou reprezentaci

Dopředné plánování

Začínáme v počátečním stavu a jdeme k některému cílovému stavu

Je potřeba umět:

- rozhodnout, zda je daný stav cílový nebo ne
- najít množinu akcí aplikovatelných na daný stav
- vypočítat stav, do kterého se dostaneme aplikací akce

Algoritmus dopředného plánování

vstup: (O, s_0, g) Pozn. používáme spojení plánů, např. $\pi_1.\pi_2, a.\pi, \pi.a$

$s = s_0$;

$\pi =$ prázdný plán;

loop if s splňuje g **then return** π;

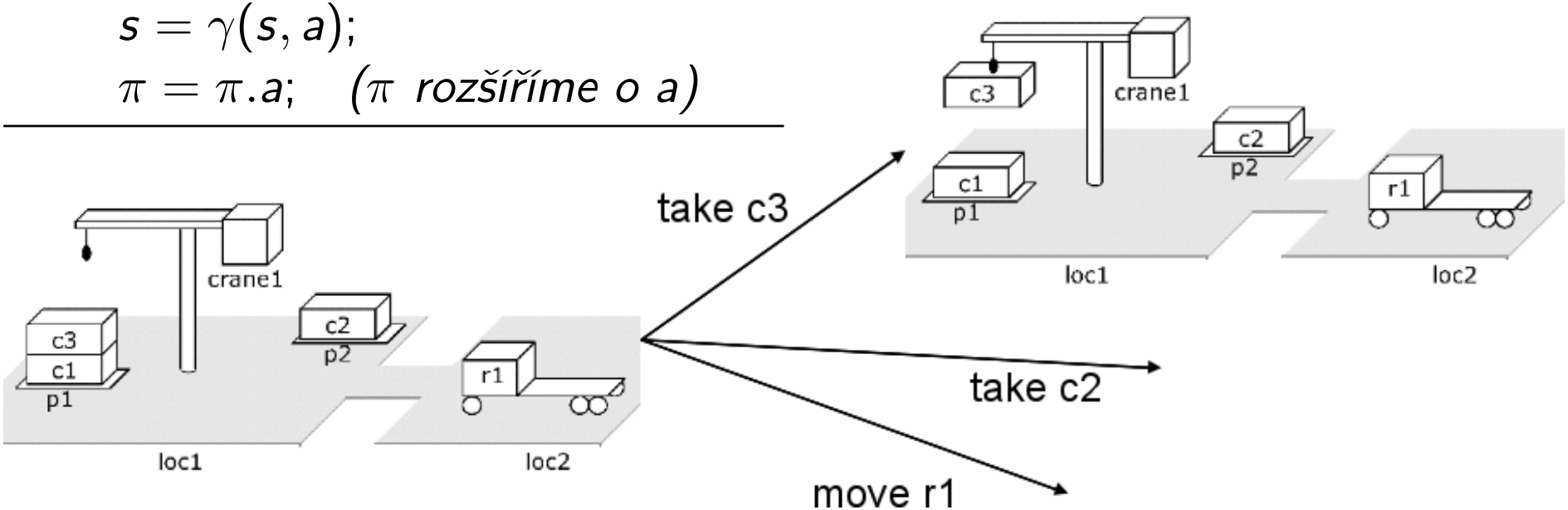
$E = \{a \mid a$ je instance operátoru v O bez proměnných a precond(a) je pravdivé v $s\}$;

if $E = \emptyset$ **then return failure**;

nedeterministicky vyber akci $a \in E$;

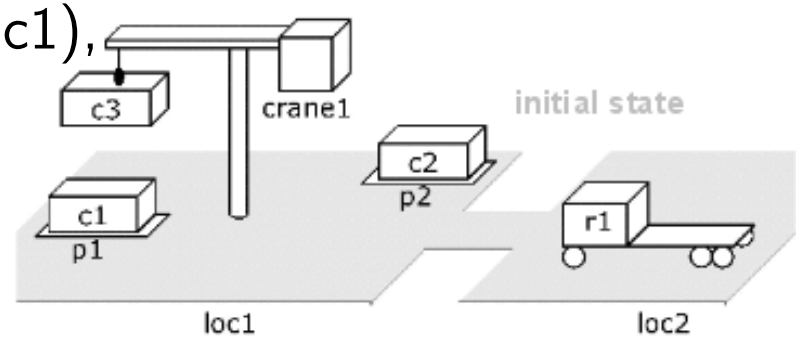
$s = \gamma(s, a)$;

$\pi = \pi.a$; *(π rozšíříme o a)*



Dopředné plánování: příklad

{belong(crane1, loc1), adjacent(loc2,loc1),
holding(crane1,c3), unloaded(r1),
at(r1,loc2), occupied(loc2), ...}
a occupied(loc1) nepatří do tohoto stavu



move(r1, loc2, loc1)

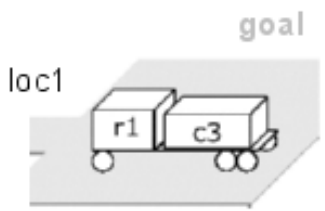
move(r, l, m)
;; robot r moves from location l to location m
precond: adjacent(l, m), at(r, l), ¬occupied(m)
effects: at(r, m), occupied(m), ¬occupied(l), ¬at(r, l)

{belong(crane1, loc1),
adjacent(loc2,loc1), holding(crane1,c3), unloaded(r1),
at(r1,loc1), occupied(loc1), ...}

load(k, l, c, r)
;; crane k at location l loads container c onto robot r
precond: belong(k, l), holding(k, c), at(r, l), unloaded(r)
effects: empty(k), ¬holding(k, c), loaded(r, c), ¬unloaded(r)

load(crane1, loc1, c3, r1)

{belong(crane1, loc1), adjacent(loc2,loc1),
empty(crane1), loaded(r1,c3),
at(r1,loc1), occupied(loc1), ...}



Cíl = {at(r1, loc1), loaded(r1, c3)} splněn

Dopředné plánování: vlastnosti

Procedura dopředného plánování je korektní

- pokud vrátí nějaký plán, potom je řešením
- stačí si uvědomit, že $s = \gamma(s_0, \pi)$

Procedura dopředného plánování je úplná

- pokud existuje plán,

  potom ho alespoň jedna z nedeterministických větví najde
- indukcí podle délky plánu
- v každém kroku má algoritmus šanci zvolit správnou akci

  (pokud v přechozích krocích volil akce z plánu)

Deterministické implementace

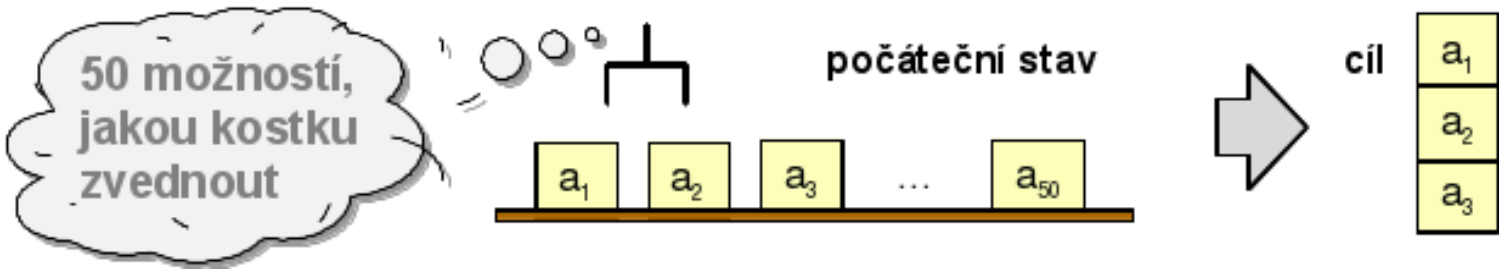
Algoritmus dopředného prohledávání můžeme implementovat deterministicky: korektnost, úplnost, paměť?

- prohledávání do šířky
  - korektní, úplné, ale paměťově náročné
- prohledávání do hloubky
  - korektní, úplnost lze zajistit kontrolou cyklů (stav se na cestě neopakuje)
- A* algoritmus
  - nejčastěji používaný přístup
  - je třeba vhodná heuristika, která směruje k nalezení kvalitního řešení
  - v daném uzlu stromu odhadujeme celkovou cenu jako $f(n) = g(n) + h(n)$ a jdeme větví s minimálním odhadem
    - $g(n)$ aktuální cena uzlu (např. dosavadní délka cesty)
    - $h(n)$ odhad ceny pro výpočet úplného řešení (heuristika)
      pro optimum nutná přípustná heuristika: cena nikdy nenadhodnocena
      př. vzdálenost přímé cesty z aktuálního bodu do cíle
  - podrobně viz např. PB016 Umělá inteligence I

Větvení

V čem je problém dopředného prohledávání?

Vysoký větvící faktor: počet možností k výběru



- to vadí u determinismu, který může ztrácet čas zkoušením irelevantních akcí

Řešení

- heuristika doporučující výběr akce
- ořezání prohledávacího prostoru
  - Např. pokud plány π_1 a π_2 dosáhly stejného stavu, potom víme, že také plány $\pi_1\pi_3$ a $\pi_2\pi_3$ dosáhnou stejného stavu. Delší z plánů π_1 a π_2 tedy nemusíme rozvíjet.
  - Je potřeba pamatovat si všechny navštívené stavy ☹.

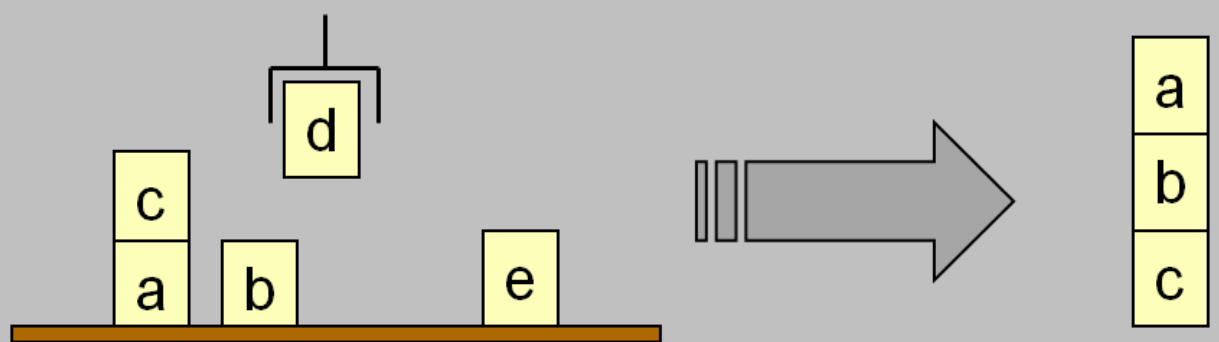
Cvičení

Navrhněte množinovou a klasickou reprezentaci pro svět kostek.

Svět kostek (the blocks world)

- nekonečně velký stůl, konečný počet kostek
- poloha kostky na stole nás nezajímá
- kostka může ležet buď na stole nebo na jiné kostce
- při plánování chceme přesouvat kostky tak, že v dané chvíli můžeme držet maximálně jednu kostku

Příklad:



Svět kostek: klasická reprezentace

Konstanty

- bloky: a,b,c,d,e

Predikáty

- ontable(x)
  kostka x je na stole
- on(x,y)
  kostka x je na kostce y
- clear(x)
  kostka x na sobě nic nemá
- holding(x)
  robotova ruka drží kostku x
- handempty
  robotova ruka nic nedrží

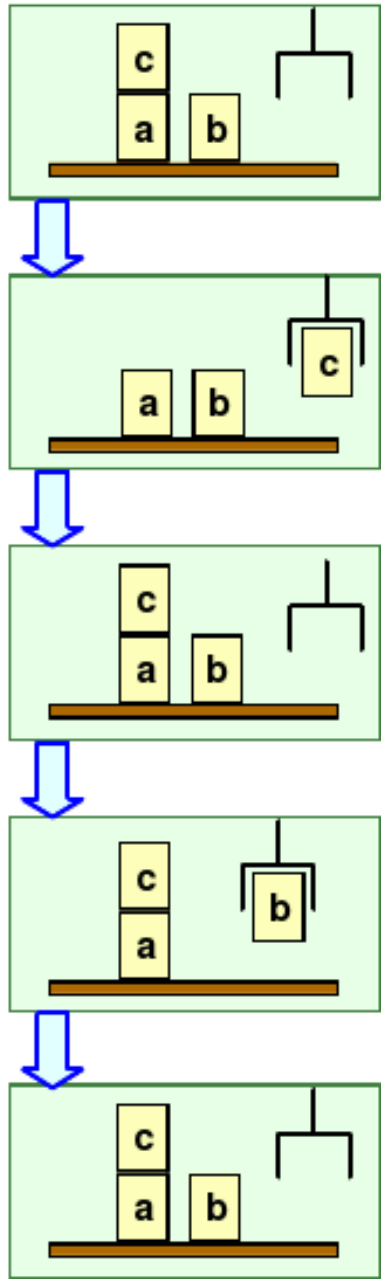
Operátory

unstack(x,y)
Precond: on(x,y), clear(x), handempty
Effects: ¬on(x,y), ¬clear(x), clear(y), ¬handempty, holding(x),

stack(x,y)
Precond: holding(x), clear(y)
Effects: ¬holding(x), ¬clear(y), on(x,y), clear(x), handempty

pickup(x)
Precond: ontable(x), clear(x), handempty
Effects: ¬ontable(x), ¬clear(x), ¬handempty, holding(x)

putdown(x)
Precond: holding(x)
Effects: ¬holding(x), ontable(x), clear(x), handempty



Svět kostek: množinová reprezentace

Výroky

pro 5 kostek máme
36 výroků

- ontable-a (5x)
  kostka a je na stole
- on-c-a (20x)
  kostka c je na kostce a
- clear-c (5x)
  kostka c na sobě nic nemá
- holding-d (5x)
  robotova ruka drží kostku d
- handempty (1x)
  robotova ruka nic nedrží

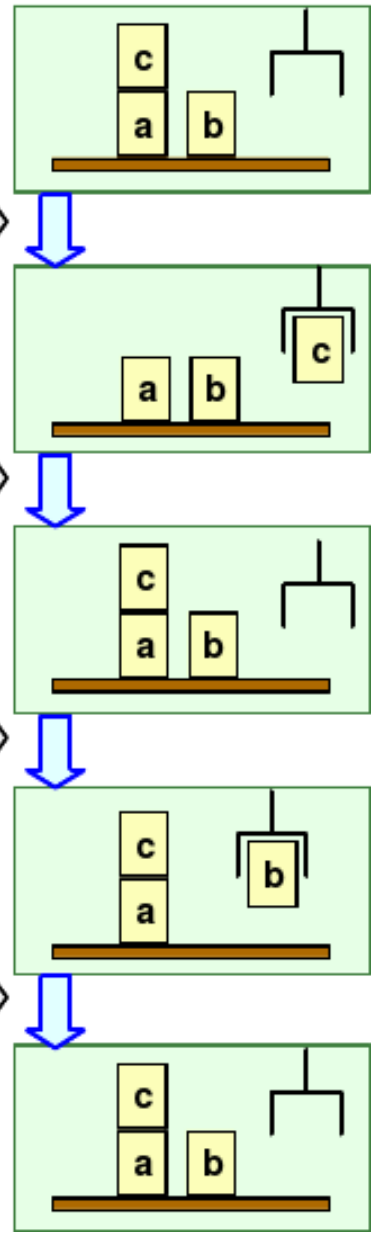
Akce: pro 5 kostek máme 50 akcí

unstack-c-a
Pre: on-c-a, clear-c, handempty
Del: on-c-a, clear-c, handempty
Add: holding-c, clear-a

stack-c-a
Pre: holding-c, clear-a
Del: holding-c, clear-a
Add: on-c-a, clear-c, handempty

pickup-b
Pre: ontable-b, clear-b, handempty
Del: ontable-b, clear-b, handempty
Add: holding-b

putdown-b
Pre: holding-b
Del: holding-b
Add: ontable-b, clear-b, handempty



Cvičení: výměna hodnot proměnných

Plánovací problém:

- s_0 = {value(v1,3), value(v2,5), value(v3,0)}
- g = {value(v1,5), value(v2,3)}
- assign(v, w, x, y)

  precond: value(v, x), value(w, y)

  effects: $\neg$ value(v, x), value(v, y)

Otázky:

- Kolik iterací poběží algoritmus dopředného prohledávání minimálně?
- Kolik existuje minimálních plánů a které to jsou?
- Můžeme provést assign(v1,v1,3,3)?
- Můžeme provést assign(v1,v2,3,3)?
- Jakým způsobem je prohledáván stavový prostor při prohledávání do šířky?

Zpětné plánování

Začínáme s cílem (pozor to není stav, ale reprezentace množiny stavů!) a jdeme přes podcíle k počátečnímu stavu

Je potřeba umět:

- zjistit, zda daný stav odpovídá cíli
- pro daný cíl najít relevantní akce
- vypočítat podcíl umožňující aplikovat danou akci

Zpětné plánování: relevantní akce

Příklad:

- cíl: {on(a,b), on(b,c)}
- akce stack(a,b) je relevantní

  její „zpětnou aplikací"
  dostaneme nový cíl:

  {holding(a), clear(b), on(b,c)}

stack(x,y)
Precond: holding(x), clear(y)
Effects: ~holding(x), ~clear(y),
on(x,y), clear(x), handempty

Akce a je relevantní pro cíl g právě když:

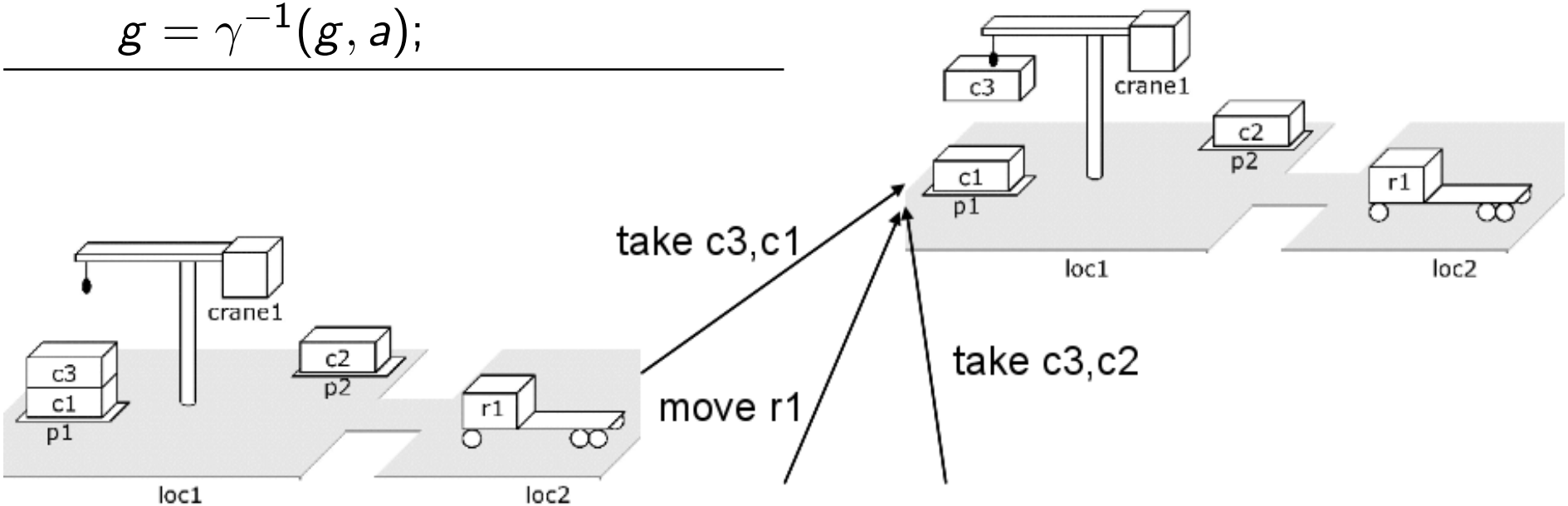
- akce přispívá do g: $g \cap \text{effects}(a) \neq \emptyset$
- efekty akce nejsou v konfliktu s g:
  - $g^- \cap \text{effects}^+ = \emptyset$
  - $g^+ \cap \text{effects}^- = \emptyset$

Regresní (zpětná) množina cíle g pro (relevantní) akci a:

$$\gamma^{-1}(g, a) = (g - \text{effects}(a)) \cup \text{precond}(a)$$

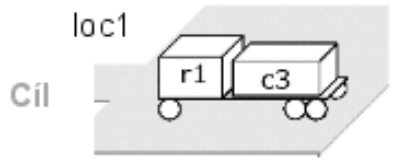
Algoritmus zpětného plánování

vstup: (O, s_0, g)
$\pi =$ prázdný plán;
loop if s_0 splňuje g **then return** π;
    $A = \{a \mid a$ je instance operátoru v O bez proměnných
        a $\gamma^{-1}(g, a)$ je definováno$\}$;
    if $A = \emptyset$ **then return failure**;
    nedeterministicky vyber akci $a \in A$;
    $\pi = a.\pi$; *(a je poslední akcí vzniklého plánu π)*
    $g = \gamma^{-1}(g, a)$;



Zpětné prohledávání: příklad

Cíl = {at(r1, loc1), loaded(r1,c3)}



load(crane1, loc1, c3, r1)

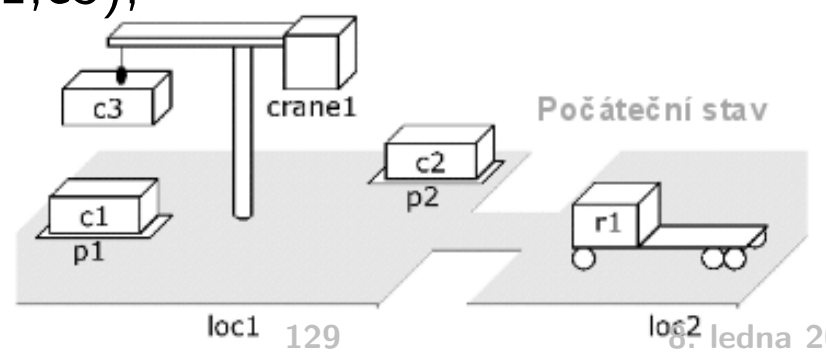
load(k, l, c, r)
;; crane k at location l loads container c onto robot r
precond: belong(k, l), holding(k, c), at(r, l), unloaded(r)
effects: empty(k), $\neg$ holding(k, c), loaded(r, c), $\neg$ unloaded(r)

{at(r1, loc1), belong(crane1, loc1),
holding(crane1, c3), unloaded(r1)}

move(r1, loc2, loc1)

move(r, l, m)
;; robot r moves from location l to location m
precond: adjacent(l, m), at(r, l), $\neg$ occupied(m)
effects: at(r, m), occupied(m), $\neg$ occupied(l), $\neg$ at(r, l)

{belong(crane1, loc1), holding(crane1,c3),
unloaded(r1), adjacent(loc2, loc1),
at(r1, loc2), $\neg$ occupied(loc1)}



Zpětné prohledávání: vlastnosti

Procedura zpětného prohledávání je korektní a úplná

Můžeme ji implementovat deterministicky

- pro úplnost potřebujeme detekci cyklů
  - mějme $(g_1, \ldots, g_k)$ posloupnost cílů:
    pokud $\exists i < k \; g_i \subseteq g_k$, pak můžeme prohledávání této cesty ukončit

Větvení

- může být menší než u prohledávání dopředu (zacílení)
- pořád ale zbytečně velké
  - Chceme-li, aby byl robot v pozici loc51, do které existují cesty z loc1, ..., loc50, dostaneme 50 možných podcílů. Nám je ale pro splnění cíle jedno, odkud robot přijel!
  - Částečné instanciování akcí (místo úplného) dále zmenší velikost větvení, tzv liftování (lifting).

Zpětné plánování: liftovaná verze

---

vstup: (O, s_0, g)
$\pi =$ prázdný plán;
loop if s_0 splňuje g **then return** π;
$A = \{(o, \theta) \mid o$ vznikne přejmenováním proměnných operátoru v O,
substituce θ je MGU atomu v g a atomu v effects(o),
a $\gamma^{-1}(\theta(g), \theta(o))$ je definováno$\}$;
if $A = \emptyset$ **then return failure**;
nedeterministicky vyber pár $(o, \theta) \in A$;
$\pi =$ spojení $\theta(o)$ a $\theta(\pi)$;
$g = \gamma^{-1}(\theta(g), \theta(o))$;

Poznámky:

- o je kopie operátoru s přejmenovanými (=novými) proměnnými
- MGU = největší společný unifikátor, tj. nejobecnější substituce atomů
- použití volných proměnných zmenšuje větvení,
  ale komplikuje detekci cyklů

Zpětné plánování v liftované verzi: příklad

- Cíl: at(robot,loc1)
- Operátor: move(r, l, m)
  - precond: adjacent(l, m), at(r, l), $\neg$occupied(m)
  - effects: at(r, m), occupied(m), $\neg$occupied(l), $\neg$at(r, l)
- Akce: move(robot, l, loc1)
  - l= ? (příliš mnoho voleb zvětšuje faktor větvení)

- Vstup (O, s_0, g) = ({move(r, l, m)}, s_0, {at(robot,loc1)})
- nechť s_0 nesplňuje cíl
- Prvek A: (o, θ) = (move(r_1, l_1, m_1),{$r_1 \rightarrow$ robot, $m_1 \rightarrow$ loc1})
  θ je MGU atomu v cíli at(robot,loc1) a atomu at(r_1, l_1) v effects(o)
- Plán $\pi = \langle$move(robot,l_1,loc1)$\rangle$
- $g = \gamma^{-1}(\theta(g), \theta(o)) =$
  {adjacent(l_1,loc1), at(robot,l_1), $\neg$occupied(loc1)}

STRIPS

Dosud probrané plánovací algoritmy sdílejí stejný problém – jak zlepšit efektivitu redukcí prohledávacího prostoru

STRIPS algoritmus redukuje prohledávaný prostor zpětného plánování, a to takto:

- z podcílů řeší vždy jen část odpovídající předpokladům poslední přidané akce
  - tj. místo $\gamma^{-1}(s, a)$ se jako nový cíl použije precond(a)
  - vede k neúplnosti algoritmu
- pokud aktuální stav splňuje všechny předpoklady operátoru, daný operátor se použije a tento závazek nebude rušen při backtrackingu

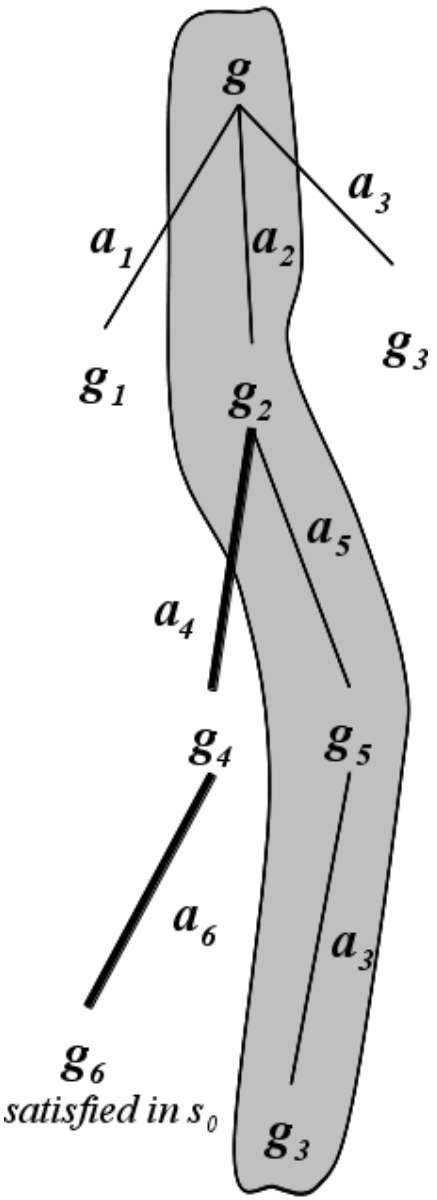
STRIPS algoritmus

Originální algoritmus STRIPS je liftovanou verzí níže popsaného algoritmu

---

function Ground-STRIPS(O, s, g)
$\pi =$ prázdný plán;
loop if s splňuje g **then return** π;
  $A = \{a \mid a$ je instance operátoru v O bez proměnných
    a a je relevantní pro $g\}$;
  if $A = \emptyset$ **then return failure**;
  nedeterministicky vyber akci $a \in A$;
  $\pi' =$ Ground-STRIPS($O, s,$ precond(a));
  if $\pi' =$ **failure then return failure**;
  (pokud jsme zde, pak π' dosáhlo precond(a) z s)
  $s = \gamma(s, \pi')$;
* *(s nyní splňuje precond(a))*
  $s = \gamma(s, a)$;
  $\pi = \pi.\pi'.a$;

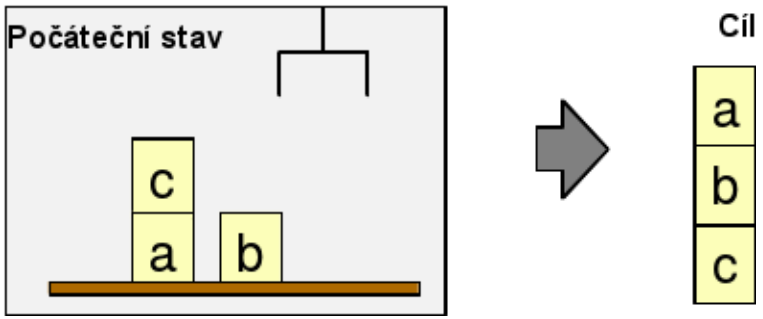
$g_2 = (g - (effects)(a_2)) \cup$ precond(a_2)
* $\pi' = \langle a_6, a_4 \rangle$ je plán pro cíl precond(a_2)
  $s = \gamma(\gamma(s_0, a_6), a_4)$ je stav splňující precond(a_2)



Sussmanova anomálie

Pravděpodobně nejznámější problém, který STRIPS neumí efektivně řešit (najde pouze redundantní plány)

Svět kostek

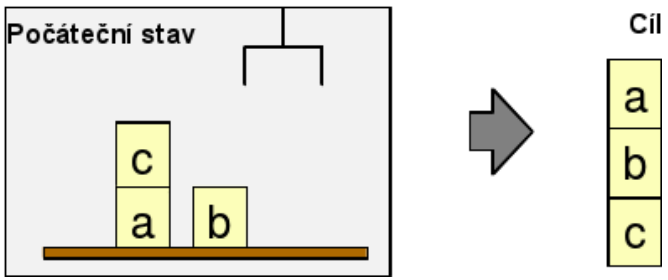


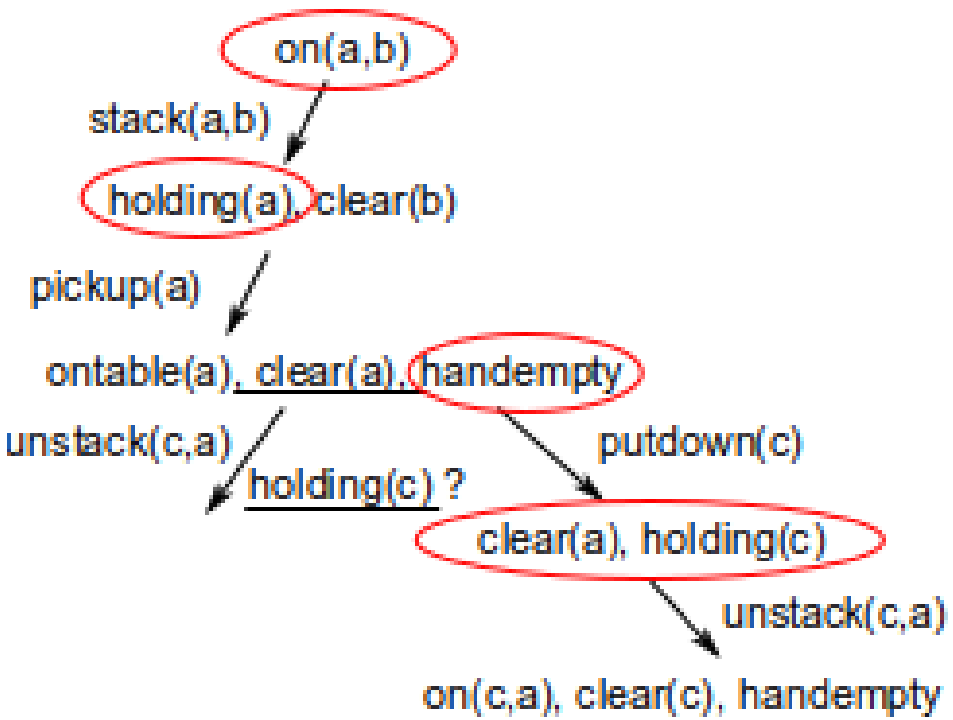
Plán vytvořený STRIPSem může vypadat takto:

- unstack(c,a), putdown(c), pickup(a), stack(a,b)

nyní jsme splnili cíl on(a,b)

Podrobněji: plán vytvořený STRIPSem pro on(a,b)





unstack(*x*,*y*)
Precond: on(x,y), clear(x), handempty
Effects: ¬on(x,y), ¬clear(x), clear(y),
¬handempty, holding(x),

stack(*x*,*y*)
Precond: holding(x), clear(y)
Effects: ¬holding(x), ¬clear(y),
on(x,y), clear(x), handempty

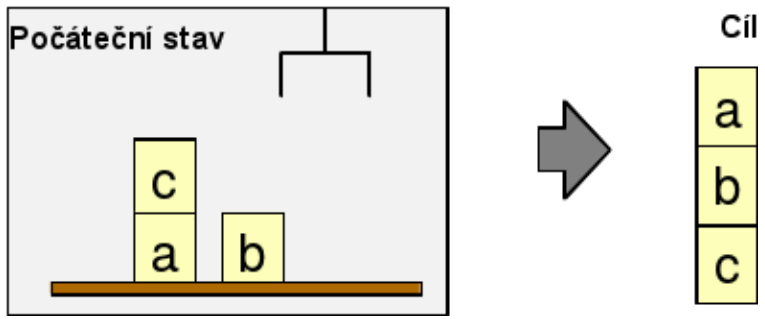
pickup(*x*)
Precond: ontable(x), clear(x), handempty
Effects: ¬ontable(x), ¬clear(x),
¬handempty, holding(x)

putdown(*x*)
Precond: holding(x)
Effects: ¬holding(x), ontable(x),
clear(x), handempty

Dokončení: Sussmanova anomálie

Pravděpodobně nejznámější problém, který STRIPS neumí efektivně řešit (najde pouze redundantní plány)

Svět kostek



Plán vytvořený STRIPSem může vypadat takto:

- unstack(c,a), putdown(c), pickup(a), stack(a,b)

  nyní jsme splnili cíl on(a,b)
- unstack(a,b), putdown(a), pickup(b), stack(b,c)

  nyní jsme splnili cíl on(b,c), ale musíme se vrátit k on(a,b)
- pickup(a), stack(a,b)

  podtržené akce se mohou vyřadit

tj. postupně řešíme cíle on(a,b) a on(b,c), při tom si ale on(a,b) pokazíme

Jak na svět kostek?

Řešení Sussmanovy anomálie

- prolínání plánů
  - plánování v prostoru plánů
- použití doménově závislé informace
  - Kdy má problém ve světě kostek řešení?
    - v cíli nesmí být kostky, které nejsou v počátečním stavu
    - kostka nesmí najednou stát na dvou jiných kostkách
    - ...
  - Jak to řešení najdeme?
    Celkem snadno a rychle!
    - dámě všechny kostky (samostatně) na stůl
    - postavíme požadované věže

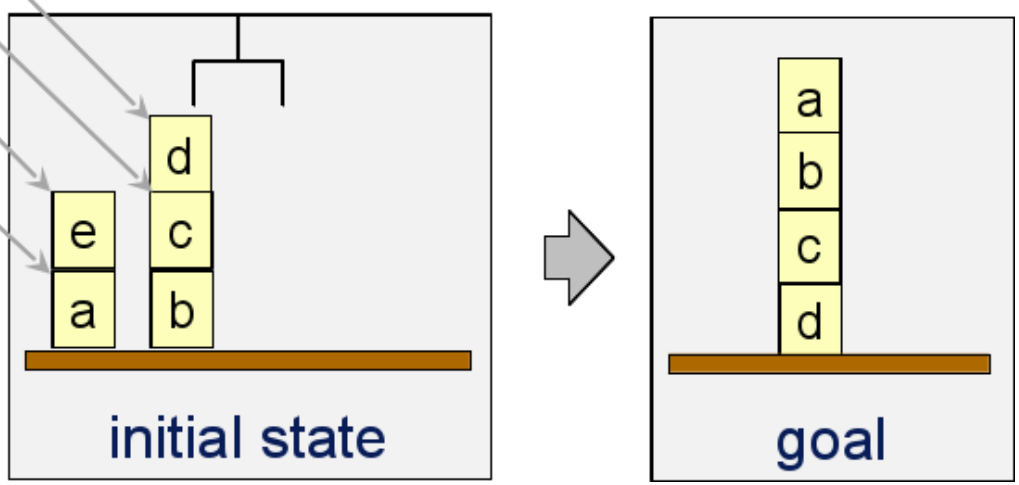
    S dalšími znalostmi to můžeme udělat ještě lépe!

Doménové znalosti (pro svět kostek)

Kdy musíme pohnout kostkou x?

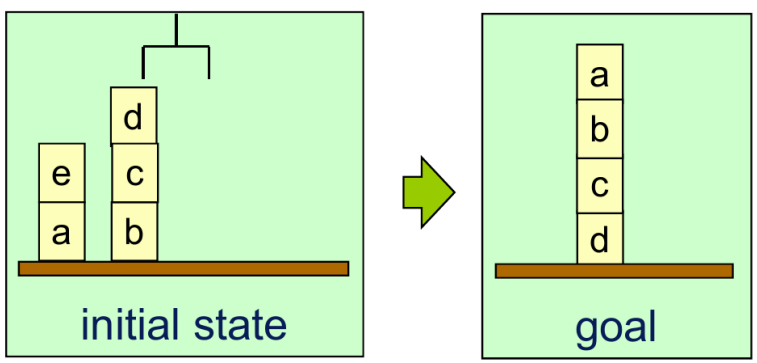
Pokud platí některé z následujících:

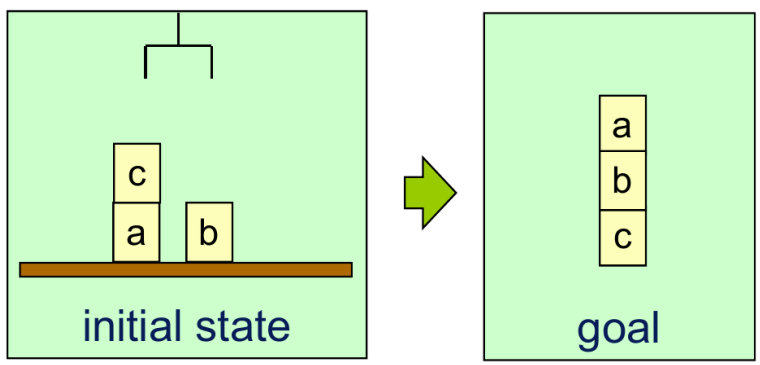
- s obsahuje ontable(x) a g obsahuje on(x,y)
- s obsahuje on(x,y) a g obsahuje ontable(x)
- s obsahuje on(x,y) a g obsahuje on(x,z) pro nějaké y≠z
- s obsahuje on(x,y) a y se musí pohnout



Doménově specifický algoritmus

procedure Block-stacking
loop
  if existuje volný blok x tak, že
    x se musí přesunout **and**
    x lze přesunout na místo, ze kterého ho už nemusíme přesouvat
  then přesuň x na toto místo
  else if existuje volný blok x tak, že
    x se musí přesunout
  then přesuň x na stůl
  else if cíl splněn
  then return plán
  else return failure





Vlastnosti algoritmu

- Korektní, úplný, garance ukončení
- Časová složitost $O(n^3)$
  - lze modifikovat s výslednou složitostí $O(n)$
- Často nalezne optimální (nejkratší) řešení
  - delka plánu pro svět kostek je NP-úplná

Plánování v prostoru plánů

8. ledna 2021

Zdroj: *Roman Barták, přednáška Umělá inteligence II, Matematicko-fyzikální fakulta, Karlova univerzita v Praze, 2014.*
http://kti.ms.mff.cuni.cz/~bartak/ui2

Plánování v prostoru plánů: základní myšlenka

Princip podobný zpětnému plánování ve stavovém prostoru:

- začínáme z „prázdného plánu"
  obsahujícího popis počátečního stavu a cíle
- přidáváme další akce, které plní dosud nesplněné (otevřené) cíle
- případně přidáváme vazby mezi již přítomnými akcemi

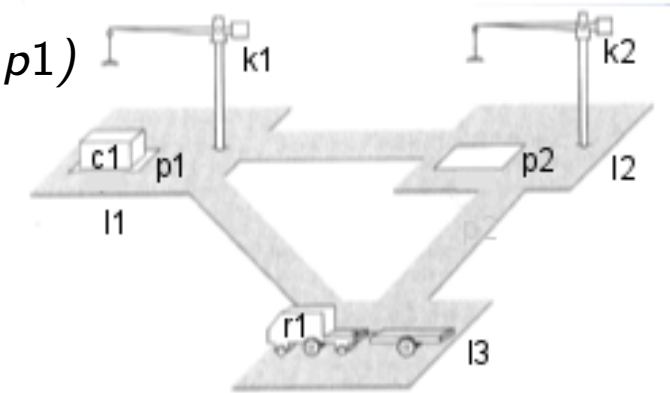
Na plánování se můžeme dívat jako na opravování kazů v částečném plánu

- přecházíme od jednoho částečného plánu k dalšímu,
  dokud nenajdeme úplný plán

Plánování v prostoru plánů: příklad

Předpokládejme, že v částečném plánu zatím máme akce

- take($k1, c1, p1, l1$) *(jeřáb $k1$ na $l1$ vezme $c1$ z $p1$)*
- load($k1, c1, r1, l1$) *($k1$ na $l1$ naloží $c1$ na $r1$)*



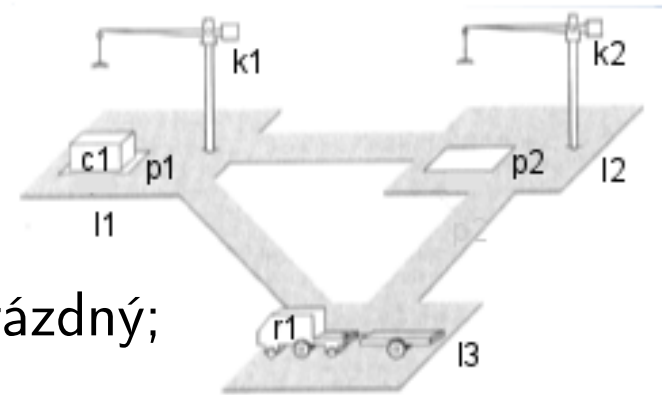
Možné úpravy plánu:

1. přidání akce
   - aby šlo použít akci „load", musí být robot $r1$ na místě $l1$
   - přesuň robota $r1$ na místo l1 ... move($r1, l, l1$)
2. svázání proměnných
   - akce move se týka správného robota na správném místě
3. přidání podmínky uspořádání
   - přesunutí robota se musí uskutečnit před akcí load
   - na pořadí vzhledem k akci „take" ale nezáleží
4. přidání kauzální (příčinné) vazby
   - nová akce byla přidána, aby se robot dostal tam, kam má
   - kauzální vazba mezi „move" a „load" nám zjistí, že mezi těmito akcemi robota někdo zase neodvolá

Plánování v prostoru plánů: počáteční plán

Počáteční stav i cíl zakódujeme jako speciální akce, které jsou v prvotním částečném plánu:

- akce a_0 reprezentuje počáteční stav tak, že nemá žádné předpoklady a počáteční stav je zakódován jako efekt;
  tato akce je před všemi ostatními akcemi
  - př. effects(a_0): in(c1,p1), at(r1,l3)
- akce a_∞ reprezentuje cíl, který je zakódován jako předpoklad, efekt akce je prázdný;
  tato akce je za všemi ostatními akcemi
  - př. precond(a_∞): in(c1,p2)



Plánování bude založeno na odstraňování kazů (flaws) v částečném plánu

- budeme přecházet od jednoho částečného plánu k dalšímu,
  dokud nenajdeme řešící plán

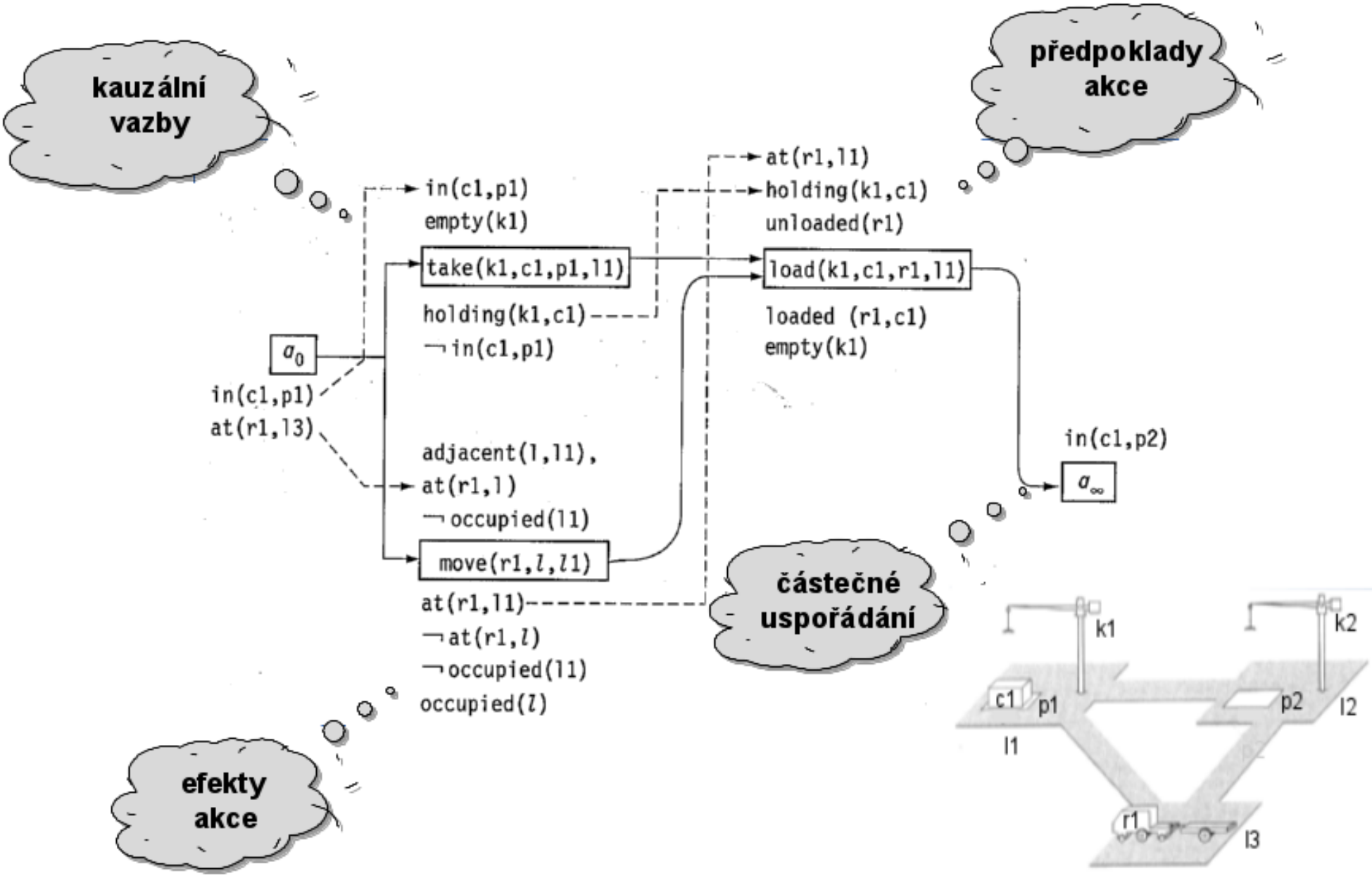
Uzly jsou plány

Uzly prohledávaného prostoru jsou tvořeny částečnými plány

Částečný plán Π je čtveřice $(A, <, B, L)$, kde

- A je množina částečně instanciovaných plánovacích operátorů $\{a_1, \ldots, a_k\}$
- $<$ je částečné uspořádání na A $(a_i < a_j)$
- B je množina vazeb tvaru $x = y, x \neq y$ nebo $x \in D_i$
- L je množina kauzálních vztahů tvaru $(a_i \xrightarrow{p} a_j)$
  - a_i, a_j jsou akce uspořádané $a_i < a_j$
  - p je výraz, který je efektem a_i a předpokladem a_j
  - v B jsou vazby svazující příslušné proměnné v p

Částečný plán: příklad



Otevřený cíl

Otevřený cíl (open goal) je kazem plánu

Jedná se o předpoklad p nějakého operátoru b, o kterém zatím nebylo rozhodnuto, jak ho splnit (neexistuje kauzální vazba $a_i \xrightarrow{p} b$)

Odstranění otevřeného cíle p akce b:

- najdi operátor a (buď již přítomný v plánu nebo nový), který lze použít na splnění p (a má p mezi efekty a může být před b)
- svaž proměnné
- vytvoř kauzální vazbu

Příklad:

- předpoklad p: at(r1,l1)
- operátor a: move(r,l,m)
- svázání proměnných: r/r1, l1/m
- kauzální vazba: move(r1,l,l1) $<$ load(k1,c1,r1,l1)

Hrozba

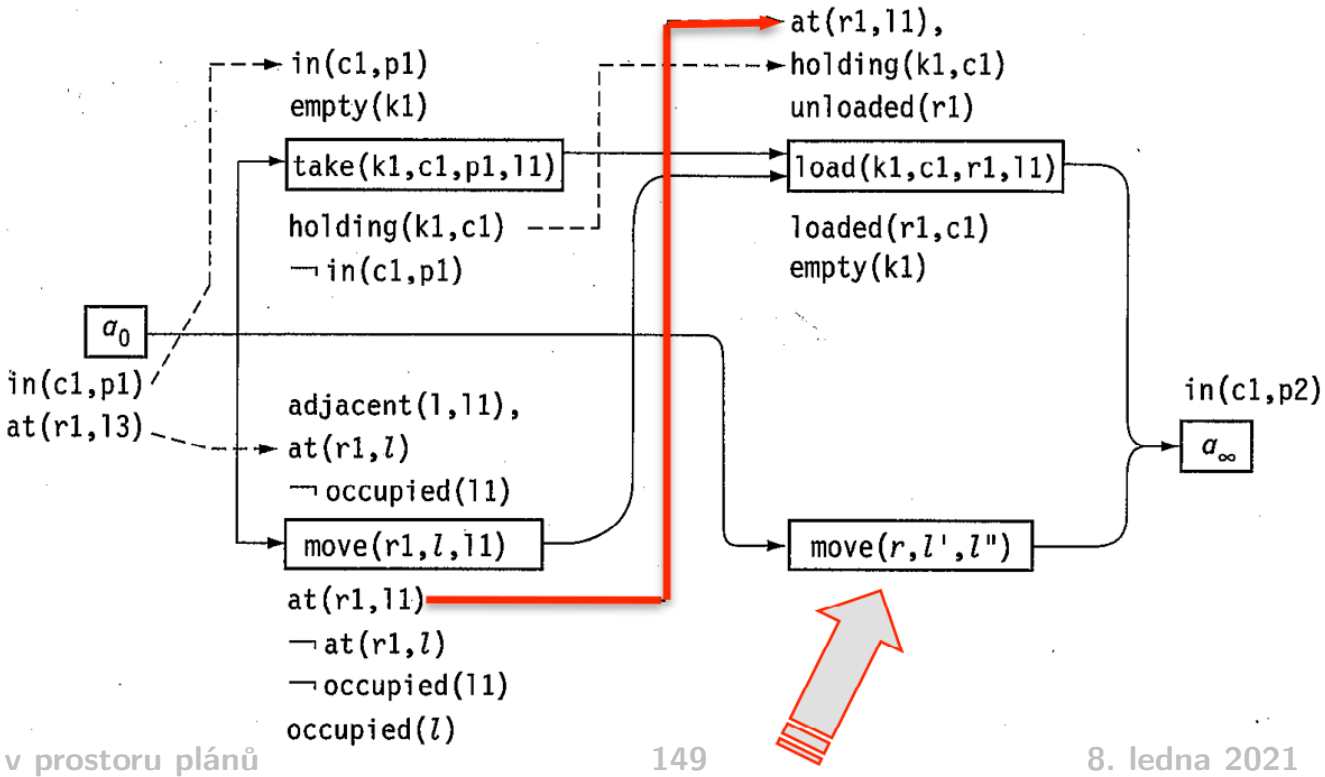
Hrozba (threat) je dalším kazem plánu

Jedná se o akci, která může porušit kauzální vazbu

- přesněji, je-li $a_i \xrightarrow{p} a_j$ kauzální vazba a akce b ma efekt unifikovatelný s negací p a může se nacházet mezi a_i a a_j, potom je b hrozbou (může porušit platnost kauzální vazby)
  - pozn. x je unifikovatelné s y? Existuje substituce, která ztotožní x s y

Odstranění hrozby lze udělat třemi způsoby:

- uspořádání b před a_i
- uspořádáním b za a_j
- navázáním proměnných v b tak, že neruší platnost p



Řešící plán

Částečný plán $\Pi = (A, <, B, L)$ je řešícím plánem

pro problém $P = (\Sigma, s_0, g)$, pokud:

- částečné uspořádání $<$ a vazby B jsou globálně konzistentní
  - v částečném uspořádání nejsou cykly
  - mohu proměnné přiřadit hodnotu z příslušné domény tak, že najdu hodnoty ostatních proměnných splňující B
- libovolná lineárně uspořádaná posloupnost plně instanciovaných akcí A splňující $<$ a vazby B vede z s_0 do stavu splňujícího g

Definice nám bohužel přímo nedává výpočtovou proceduru,

jak ověřit, zda je plán řešící!

Řešící plán – konstruktivní pohled

Jak efektivně ověřit, zda je daný plán řešící?

Tvrzení: Částečný plán $\Pi = (A, <, B, L)$ je řešící, pokud:

- nemá žádné kazy, tj. otevřené cíle a hrozby
- částečné uspořádání $<$ a vazby B jsou globálně konzistentní

Důkaz indukcí podle délky plánu

- $\{a_0, a_1, a_\infty\}$ je řešící plán
- pro větší množiny akcí vyber libovolnou z možných prvních akcí a spoj ji s akcí a_0

Algoritmus plánování v prostoru plánů (PSP)

---

function PSP(π):
 kazy = otevřené-cíle(π) $\cup$ hrozby(π);
 if *kazy* $= \emptyset$ **then return**(π)
 vyber libovolný kaz $\phi \in$ *kazy*
 zjemnění = odstraň-kaz(ϕ, π); *(vrací množinu možností odstraňujících ϕ)*
 if *zjemnění* $= \emptyset$ **then return(failure)**;
 nedeterministicky vyber $\rho \in$ *zjemnění*;
 π' = použij-zjemnění(ρ, π);
 return(PSP(π'));

Determinismus:

- volba kazu je deterministická
  - musí se odstranit všechny kazy
- volba zjemnění je nedeterministická
  - v případě neúspěchu se zkouší další alternativa

Podrobnosti k algoritmu PSP

- Otevřené cíle lze efektivně zjistit udržováním agendy předpokladů akcí
  - přidání kauzální vazby pro p vyřadí p z agendy
- Všechny hrozby lze najít prozkoumáním všech trojic akcí ($O(n^3)$)
  nebo inkrementálně: $a_i \xrightarrow{p} a_j$, efekt $\neg p$?
  - po přidání akce se zjistí, komu je hrozbou ($O(n^2)$),
  - a po přidání kauzální vazby se ověří její hrozby ($O(n)$)
- Pro odstranění otevřených cílů a hrozeb
  se používají pouze konzistentní zjemnění plánu
  - konzistence uspořádání buď detekcí cyklů nebo
    lépe udržováním tranzitivního uzávěru
  - konzistence vztahů B
    - pokud není negace, lze rychle
      (například pomocí hranové konzistence – základní technika z oblasti programování omezujících podmínek pro zajištění konzistence)
    - je-li přítomna negace, jedná se o NP-úplný problém

Vlastnosti PSP

Algoritmus PSP je korekní a úplný

- korektnost
  - pokud program skončí, pak v něm není žádný kaz a plán je konzistentní
- úplnost
  - pokud existuje plán, má algoritmus PSP vždy možnost vybrat ty správné akce

Pozor na deterministickou implementaci!

- prostor plánů není konečný!
- úplná deterministická procedura musí garantovat prohledání všech plánů dané délky
  - např. iterativní prohlubování (iterative deepening)
    prohledávání do hloubky s omezenou hloubkou prohledávání
    prohledávaná hloubka se může postupně zvětšovat

Algoritmus PoP

PoP je konkrétní (a populární) instance algoritmu PSP

function PoP(π,*agenda*) *(kde* $\pi = (A, <, B, L)$*)*
 if *agenda* $= \emptyset$ **then return**(π);
 vyber libovolný pár (a_j, p) a smaž ho z *agenda*
 relevant = pokryj-předpoklad(p, π);
 if *relevant* $= \emptyset$ **then return(failure)**;
 nedeterministicky vyber akci $a_i \in$ *relevant*;
 $L = L \cup \{\langle a_i \xrightarrow{p} a_j \rangle\}$;
 aktualizuj B se svázanými omezeními této kauzální vazby;
 if a_i je nová akce v A **then**
  aktualizuj A s a_i;
  aktualizuj $<$ s $(a_i < a_j)$, $(a_0 < a_i < a_\infty)$
  aktualizuj *agenda* se všemi předpoklady a_i;
 for každou hrozbu v $\langle a_i \xrightarrow{p} a_j \rangle$ nebo kvůli a_i **do**
  zjemnění = množina možností odstraňující tuto hrozbu;
  if *zjemnění* $= \emptyset$ **then return(failure)**;
  nedeterministicky vyber $\rho \in$ *zjemnění*;
  přidej ρ do $<$ nebo do B;
 return(PoP(π,*agenda*));

- agenda je množina dvojic (a, p), kde p je otevřený předpoklad akce a
- nejprve hledá akci a_i pro pokrytí nějakého p z agendy
- ve druhé fázi řeší všechny hrozby, které vznikly přidáním akce a_i, resp. kauzální vazby s a_i

Příklad

Počáteční stav:

- At(Home), Sells(OBI, Drill), Sells(Tesco, Milk), Sells(Tesco, Banana)
- tj. akce Start

  precond: none

  effects: At(Home), Sells(OBI, Drill), Sells(Tesco, Milk), Sells(Tesco, Banana)

Cíl:

- Have(Drill), Have(Milk), Have(Banana), At(Home)
- tj. akce Finish

  precond: Have(Drill), Have(Milk), Have(Banana), At(Home)

  effects: none

Máme k dispozici následující operátory:

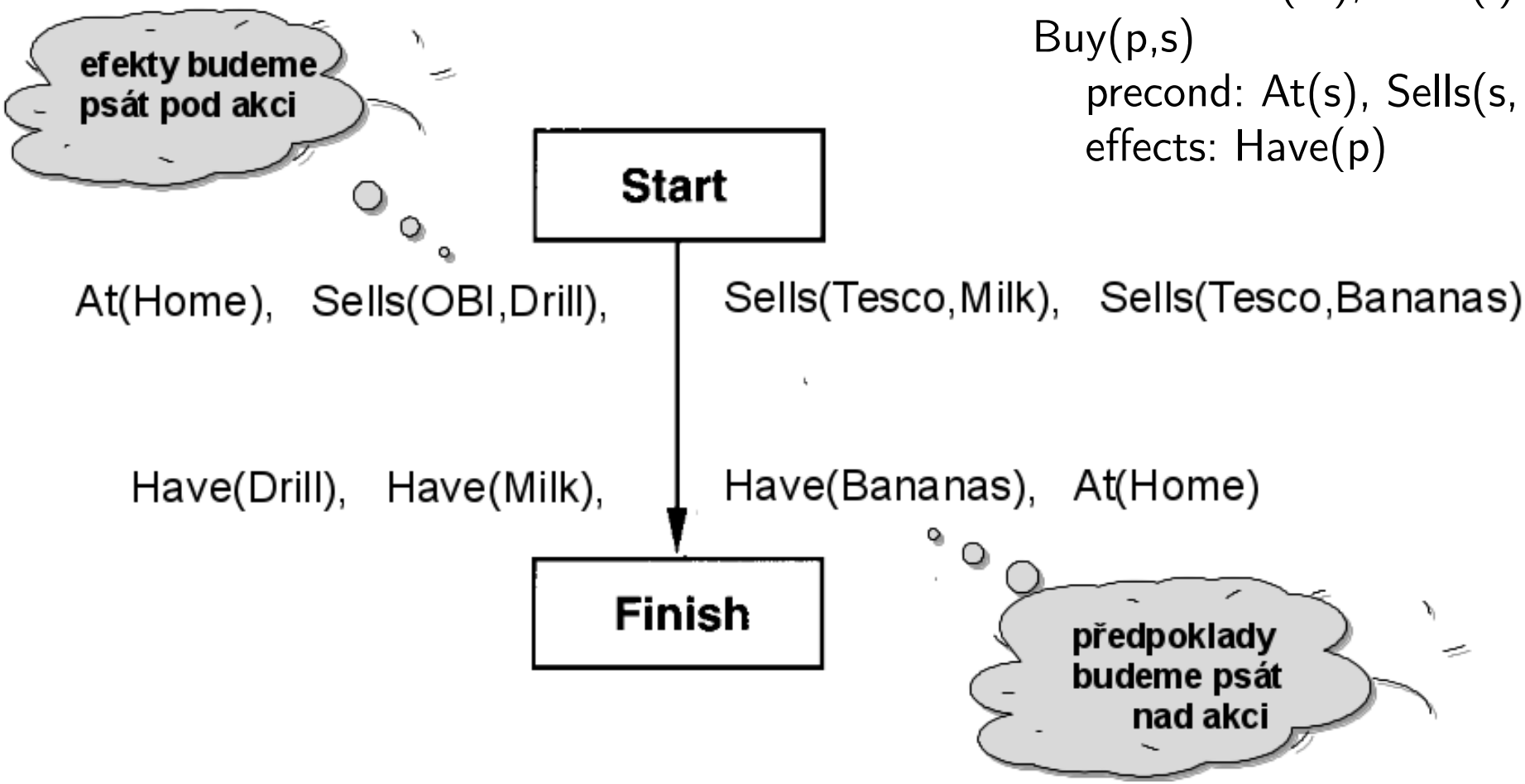
Go(l,m) *(jdi z místa l do místa m)*
  precond: At(l)
  effects: At(m), ¬ At(l)

Buy(p,s) *(na místě s kup p)*
  precond: At(s), Sells(s, p)
  effects: Have(p)

Příklad

- Počáteční plán

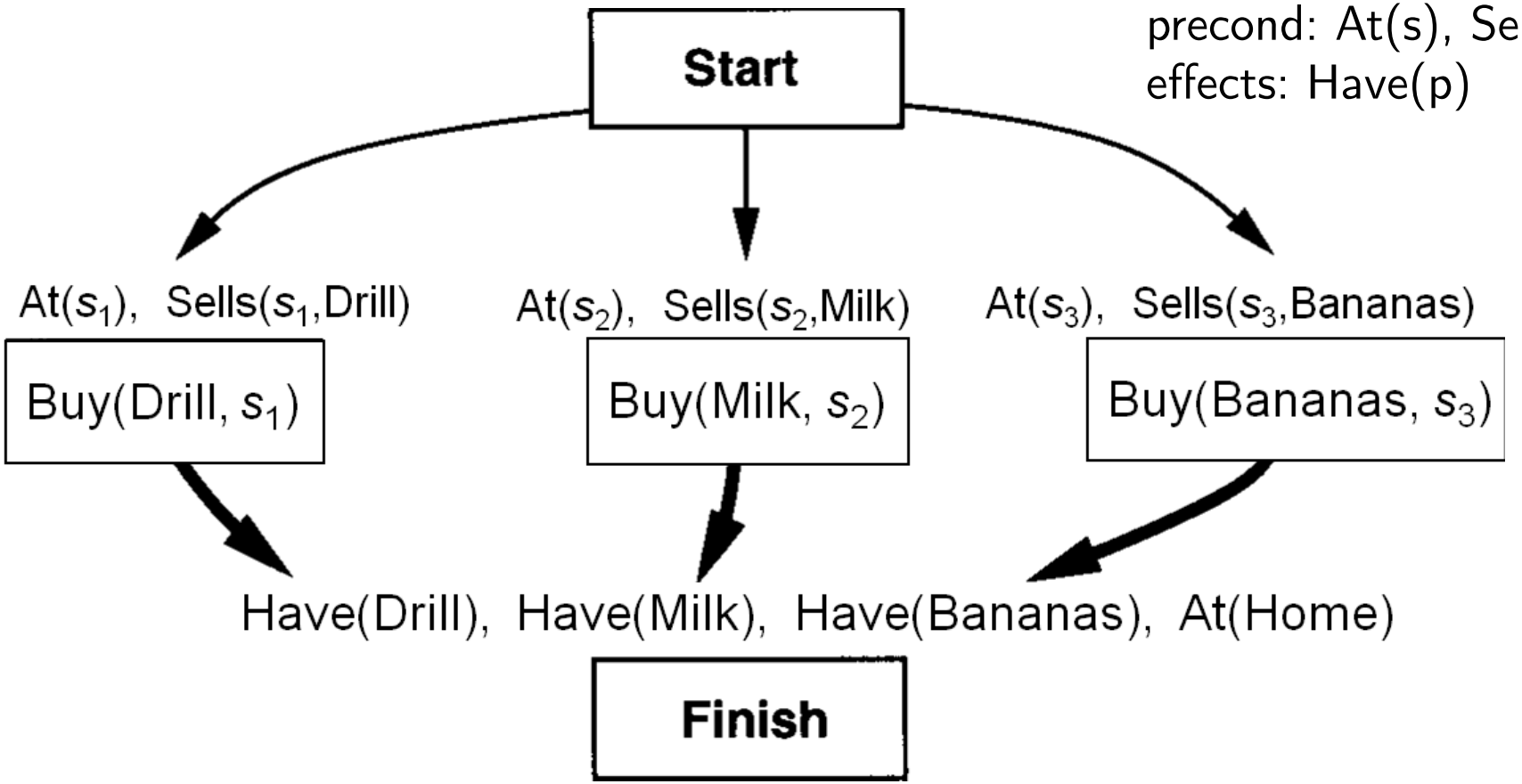
Operátory:
Go(l,m)
precond: At(l)
effects: At(m), ¬ At(l)
Buy(p,s)
precond: At(s), Sells(s, p)
effects: Have(p)



Příklad

- Jediný způsob, jak splnit otevřené cíle Have, je akcemi Buy (které netvoří žádné hrozby)

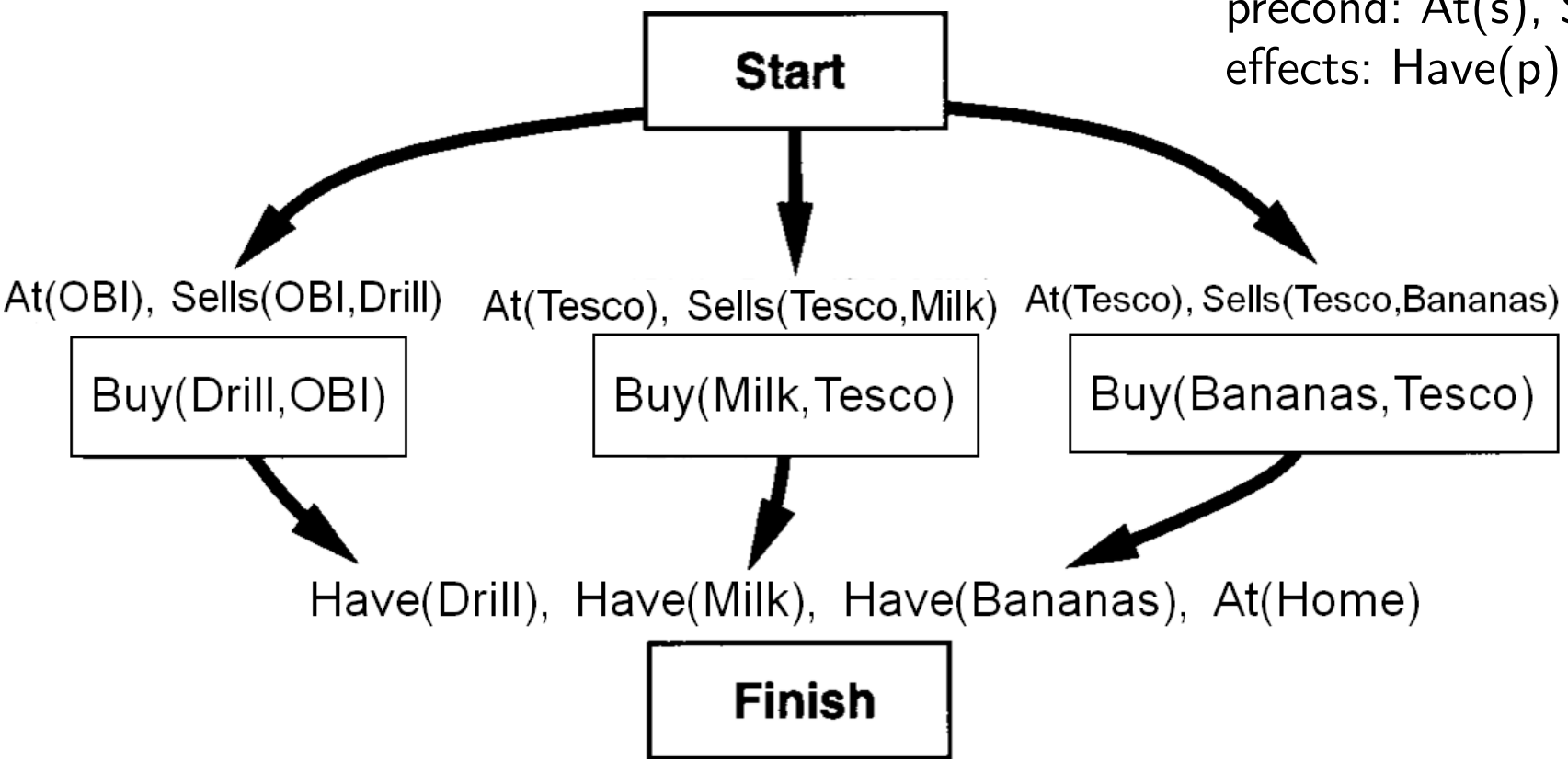
Operátory:
Go(l,m)
precond: At(l)
effects: At(m), ¬ At(l)
Buy(p,s)
precond: At(s), Sells(s, p)
effects: Have(p)



Příklad

- Jediný způsob, jak splnit předpoklady Sells je dosazení příslušných konstant

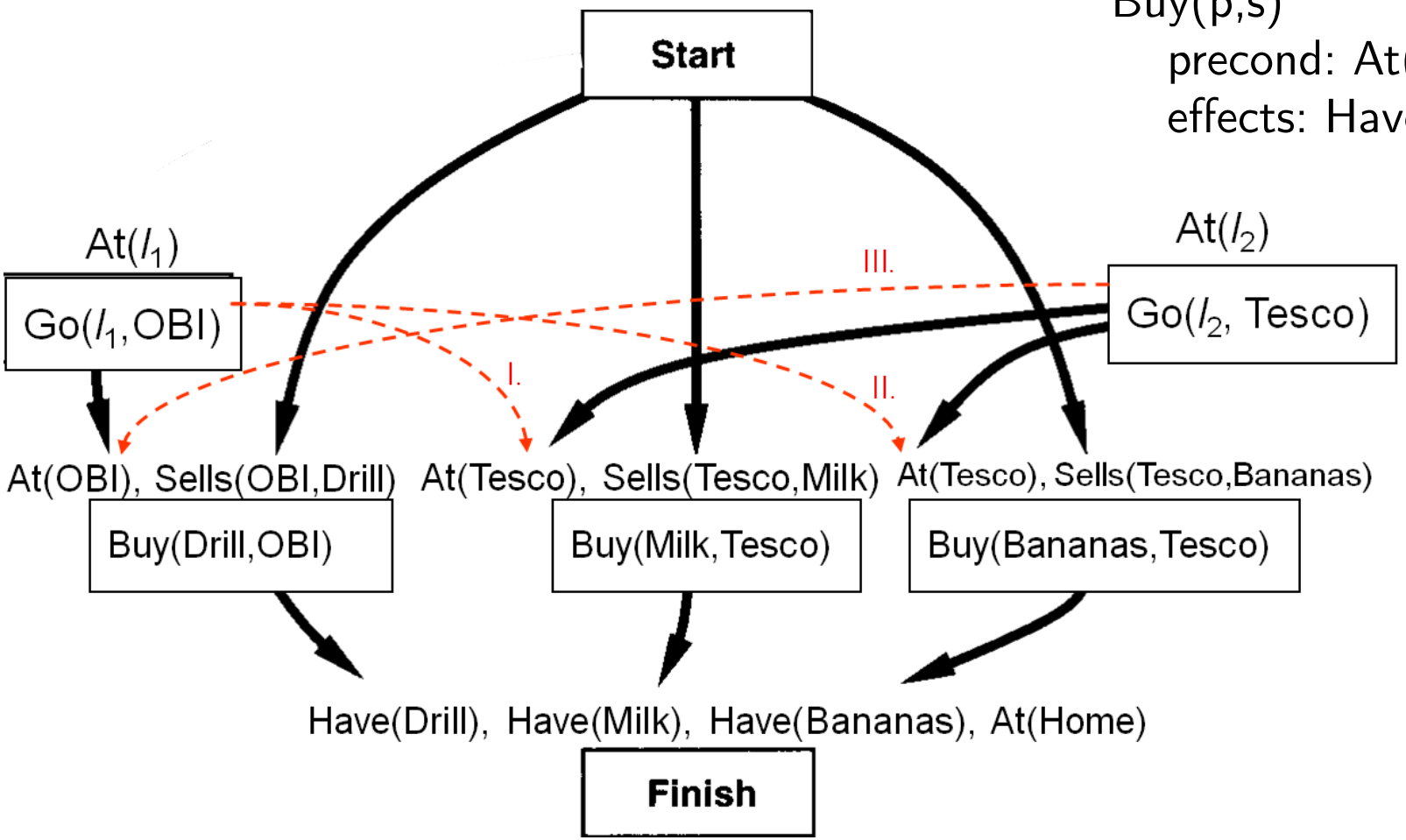
Operátory:
Go(l,m)
precond: At(l)
effects: At(m), ¬ At(l)
Buy(p,s)
precond: At(s), Sells(s, p)
effects: Have(p)



Příklad

- Jediný způsob, jak splnit otevřené cíle At, je přidání akcí Go
  - přibyly nové hrozby (I., II., III.)!

Operátory:
Go(l,m)
precond: At(l)
effects: At(m), ¬ At(l)
Buy(p,s)
precond: At(s), Sells(s, p)
effects: Have(p)



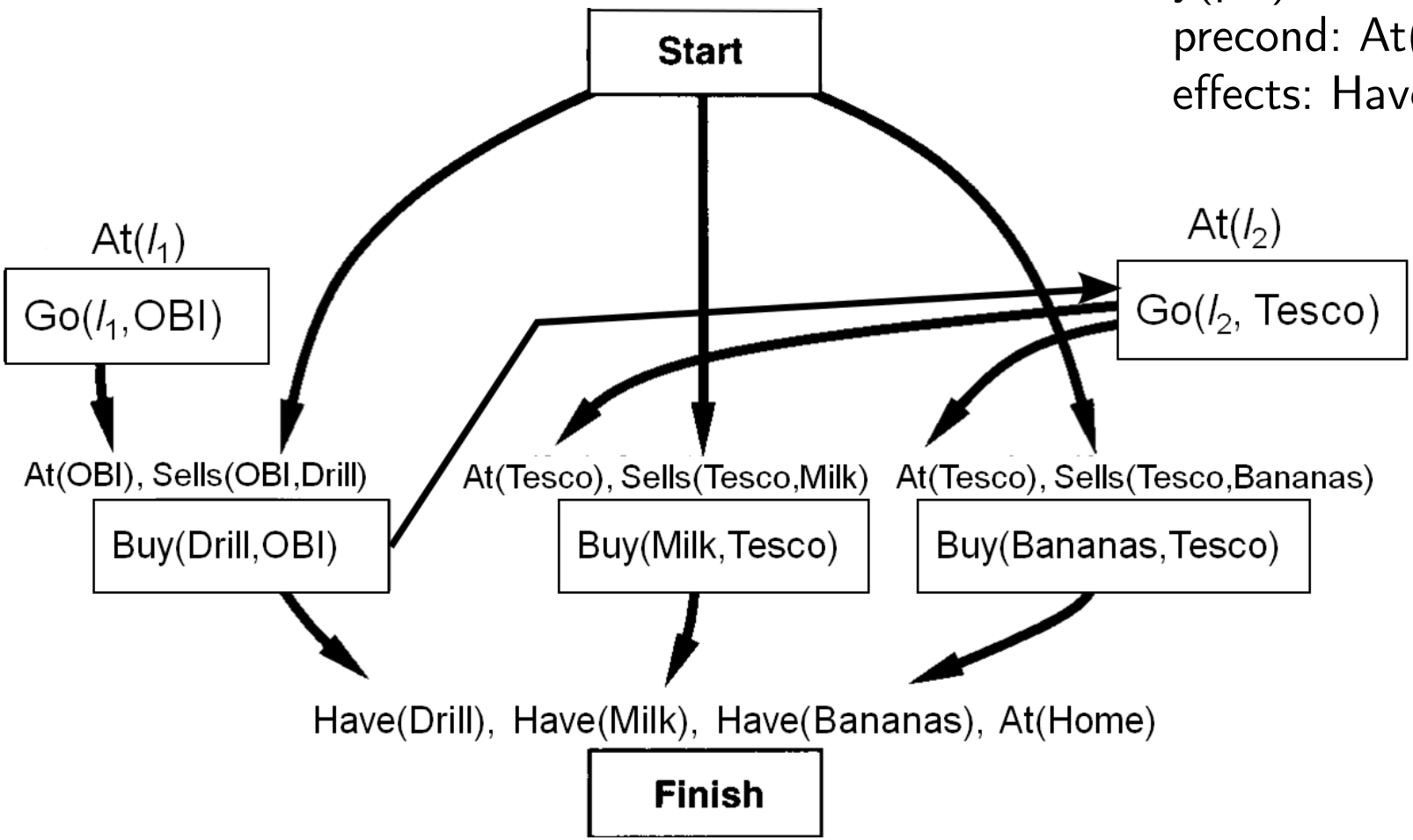
Příklad

- Třetí hrozbu můžeme odstranit uspořádáním akce Buy(Drill) před Go(Tesco)
  - to fakticky řeší všechny hrozby

Operátory:

Go(l,m)
  precond: At(l)
  effects: At(m), ¬ At(l)

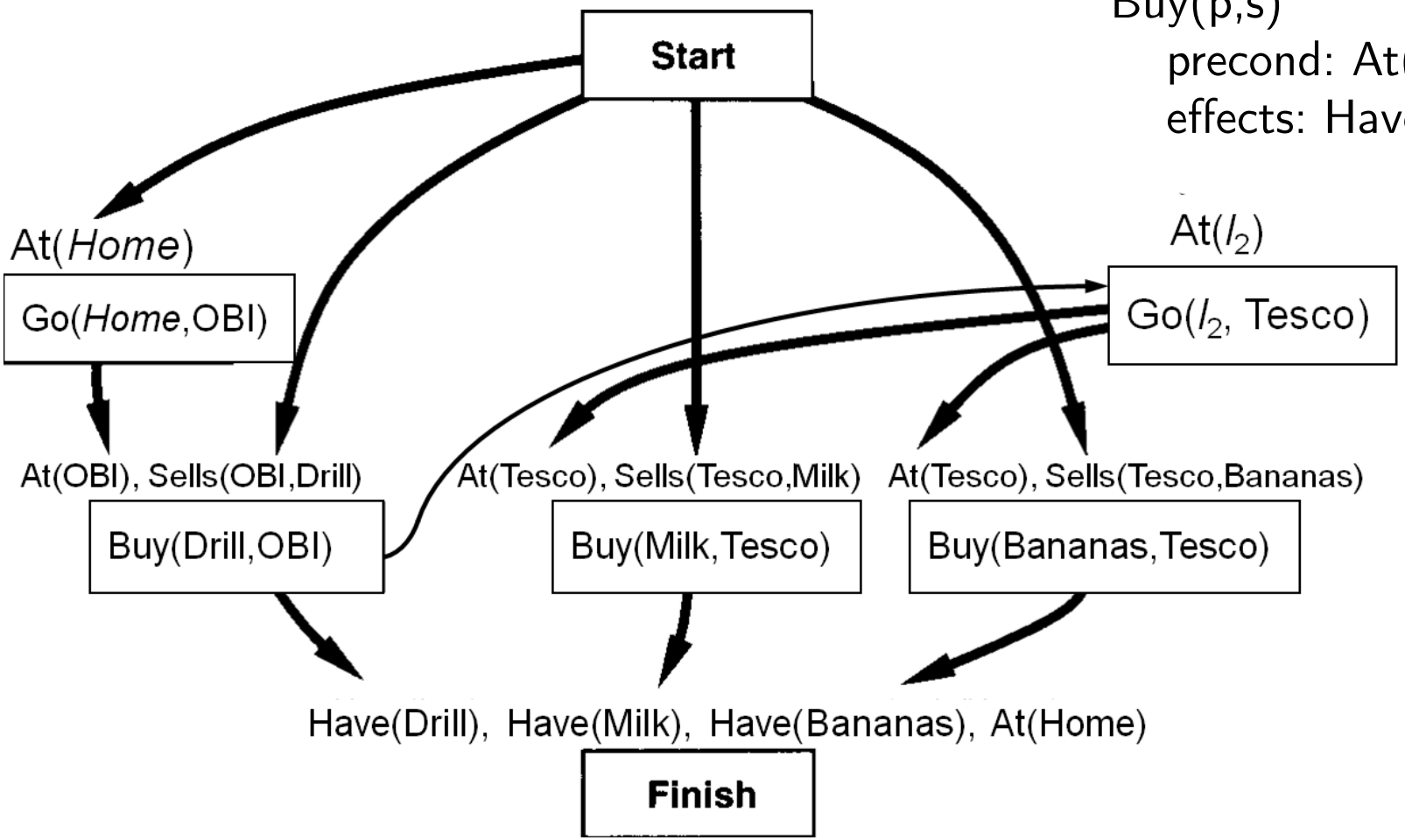
Buy(p,s)
  precond: At(s), Sells(s, p)
  effects: Have(p)



Příklad

- Otevřený cíl At(l_1) můžeme splnit přiřazením l_1=Home z akce Start

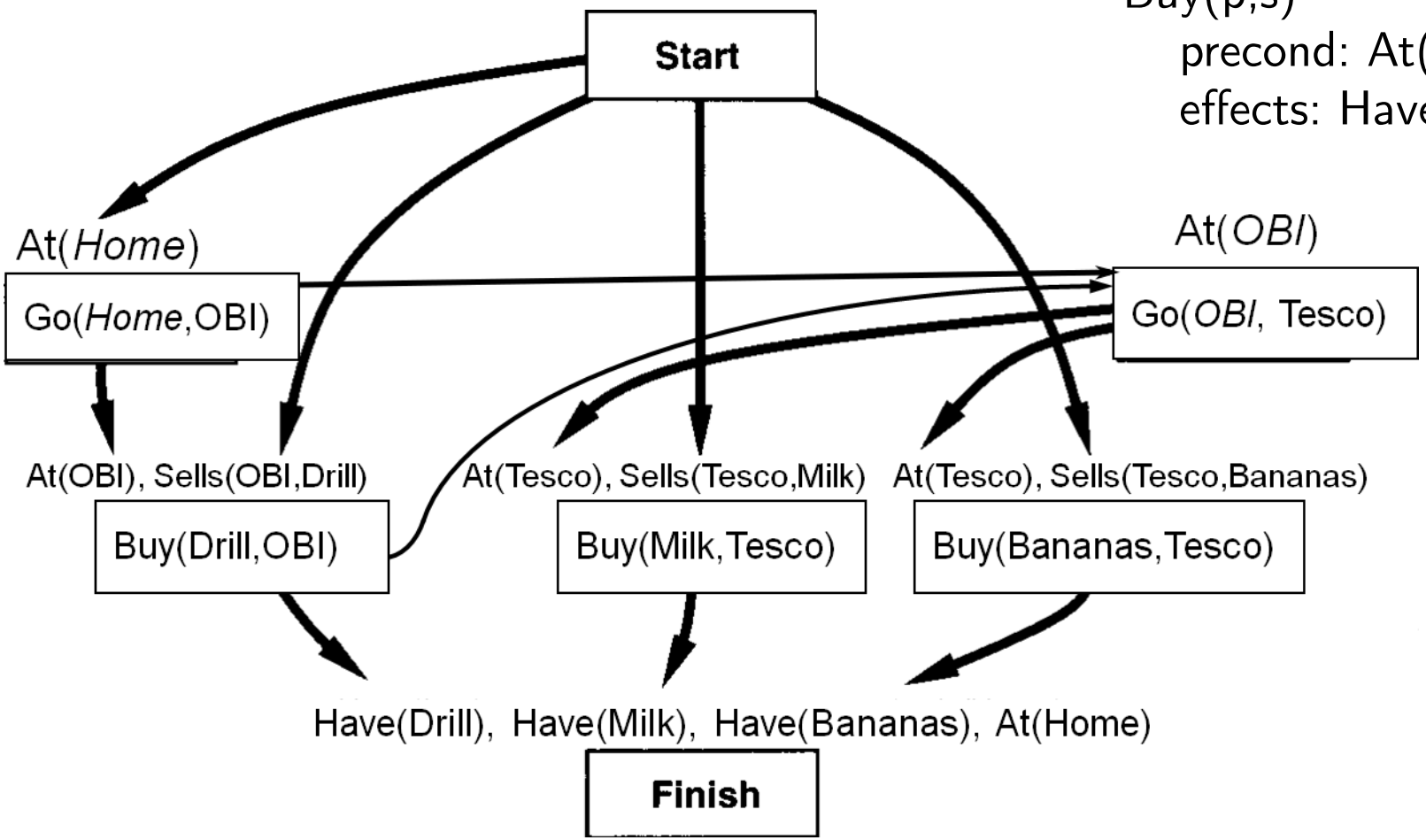
Operátory:
Go(l,m)
precond: At(l)
effects: At(m), ¬ At(l)
Buy(p,s)
precond: At(s), Sells(s, p)
effects: Have(p)



Příklad

- Otevřený cíl At(l_2) můžeme splnit přiřazením l_2=OBI z akce Go(Home, OBI)

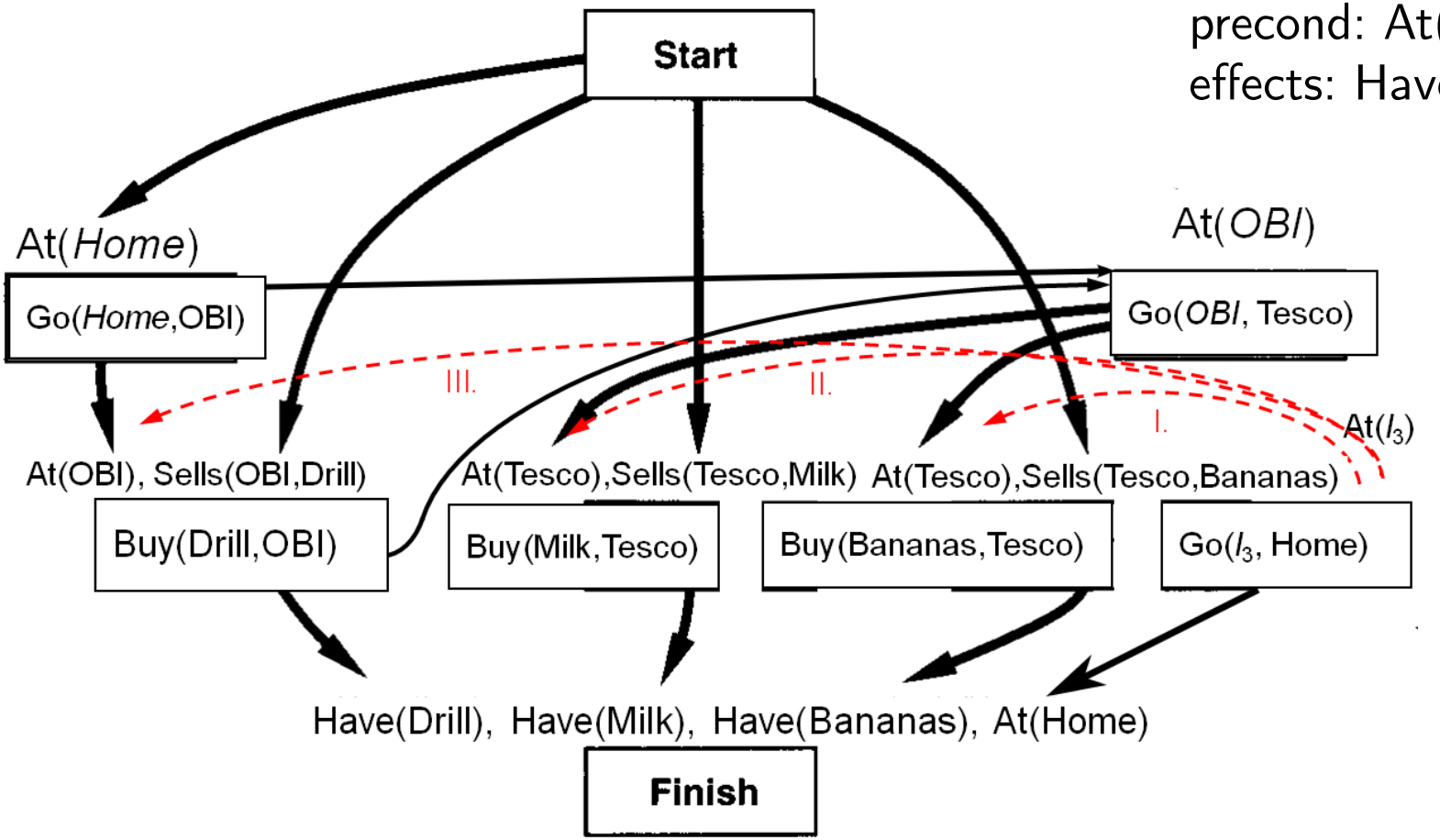
Operátory:
Go(l,m)
precond: At(l)
effects: At(m), ¬ At(l)
Buy(p,s)
precond: At(s), Sells(s, p)
effects: Have(p)



Příklad

- Otevřený cíl At(Home) z Finish splníme akcí Go(l_3, Home)
  - vzniknou nové hrozby (I., II., III.)

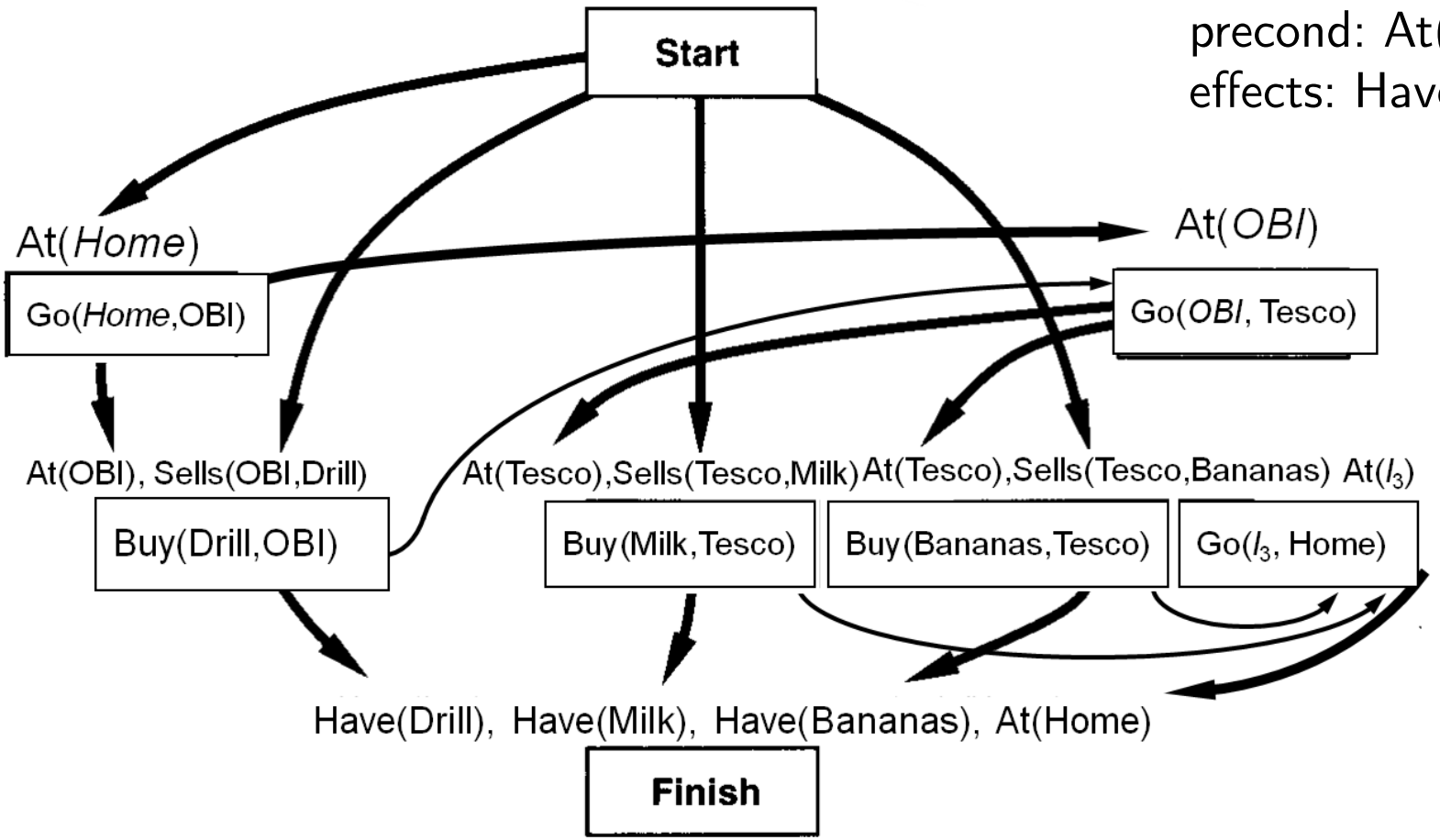
Operátory:
Go(l,m)
precond: At(l)
effects: At(m), ¬ At(l)
Buy(p,s)
precond: At(s), Sells(s, p)
effects: Have(p)



Příklad

- Hrozby na At(Tesco) (I., II.) odstraníme uspořádáním akce Go(Home) za obě akce Buy
  - to vyřeší i poslední hrozbu

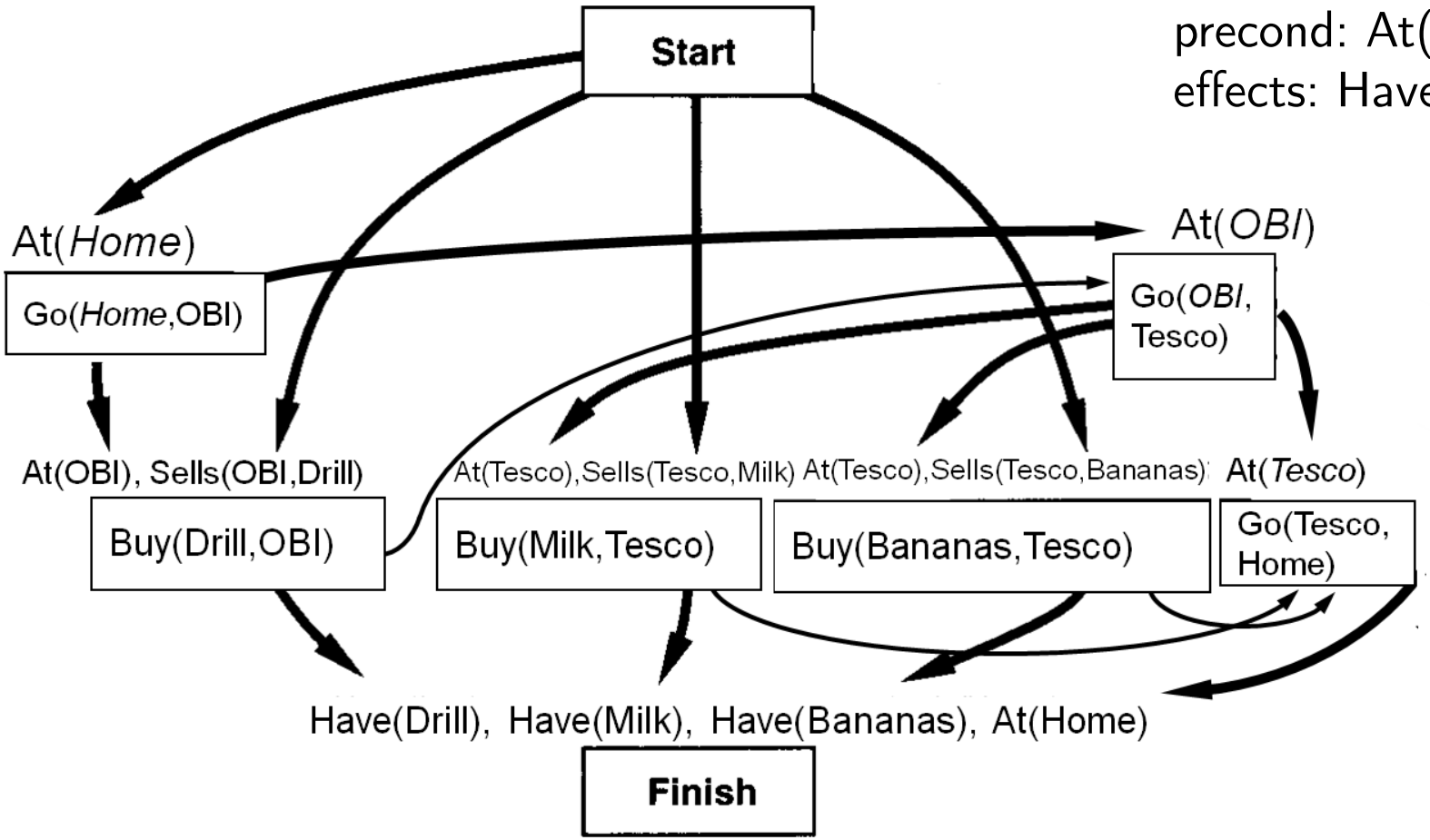
Operátory:
Go(l,m)
precond: At(l)
effects: At(m), ¬ At(l)
Buy(p,s)
precond: At(s), Sells(s, p)
effects: Have(p)



Příklad

- Otevřený cíl At(l_3) splníme přiřazením l_3=Tesco z akce Go(OBI, Tesco)

Operátory:
Go(l,m)
precond: At(l)
effects: At(m), ¬ At(l)
Buy(p,s)
precond: At(s), Sells(s, p)
effects: Have(p)



Srovnání technik

	Plánování se stavy	Plánování s plány
prohled. prostor	konečný	nekonečný
uzly	jednoduché	komplikovanější
	(stavy světa)	(částečné plány)
stavy světa	explicitní	nejsou
částečný plán	uspořádání a volba	volba akcí a jejich
	akcí se dělá najednou	pořadí oddělené
struktura plánu	lineární bez vazeb	kauzální vazby

- Díky doménově specifickým heuristikám je dnes plánování se stavy výrazně rychlejší.
- Ale, plánování v prostoru plánů:
  - vytváří flexibilnější plány díky částečnému uspořádání
  - umožňuje další rozšíření, např. přidání času a zdrojů

Cvičení: výměna hodnot proměnných

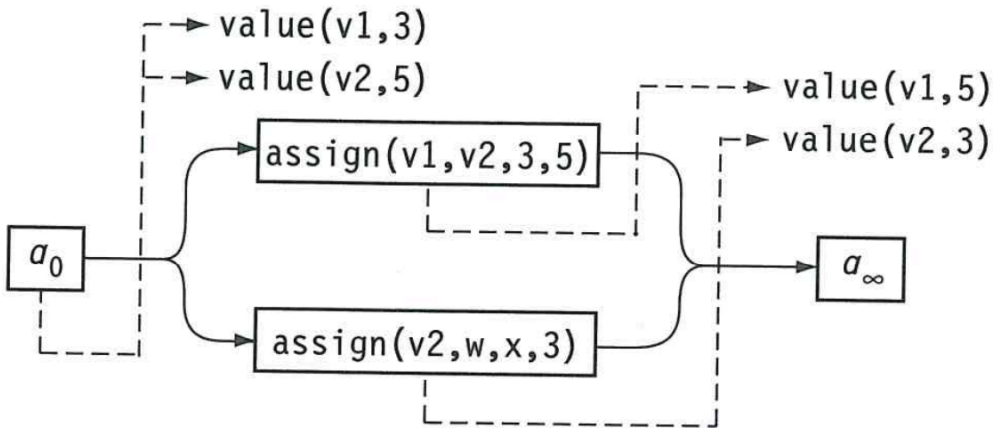
Plánovací problém:

- s_0 = {value(v1,3), value(v2,5), value(v3,0)}
- g = {value(v1,5), value(v2,3)}
- assign(v, w, x, y)

  precond: value(v, x), value(w, y)

  effects: ¬ value(v, x), value(v, y)

Částečný plán:



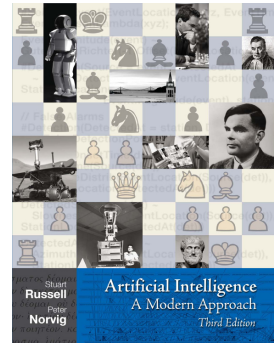
Otázky:

- Které jsou zde hrozby?
- Kolik přímých následníků má tento částečný plán?
- Provádějte operace algoritmem PSP počínaje uvedeným plánem. Který běh nalezne řešení s nejmenším možným počtem akcí?

Neurčitost: Bayesovské sítě

8. ledna 2021

Zdroj: *Roman Barták, přednáška přednáška Umělá inteligence II, Matematicko-fyzikální fakulta, Karlova univerzita v Praze, 2015.*
`http://kti.ms.mff.cuni.cz/~bartak/ui2`



Neurčitost a umělá inteligence: krátké historické shrnutí

- Pro modelování neurčitosti použijeme teorii pravděpodobnosti (podobně jako ve fyzice či ekonomii), ale bylo tomu tak vždy?
- První expertní systémy (kolem roku 1970) používali čistě logický přístup bez práce s neurčitostí, což se ukázalo nepraktické
- Další generace expertních systému zkusila pravděpodobnostní techniky, ale ty měly problém se škálovatelností (efektivní odvozování na Bayesovských sítích ještě nebylo známé)
- Proto byly zkoušeny alternativní přístupy (1975-1988) pro modelování neurčitosti, ignorance a vágnosti
  - pravidlově orientované systémy, Dempster-Shaferova teorie, fuzzy logika

Opakování: Podmíněná pravděpodobnost

Podmíněná pravděpodobnost

$$P(x|y) = P(x \wedge y)/P(y)$$

Příklad: Jaká je pravděpodobnost, že na dvou kostkách hodíme pár, pokud víme, že na kostce 1 padla 5?

- $P(dvojice|Hod1 = 5) = \frac{1/36}{1/6} = 1/6$

Podmíněnou pravděpodobnost často zapisujeme v podobě součinového pravidla:

$$P(x \wedge y) = P(x|y) \times P(y) \qquad \mathbf{P}(X, Y) = \mathbf{P}(X|Y) \times \mathbf{P}(Y)$$

- **P** znamená, že výsledek je vektor čísel
- $\mathbf{P}(X|Y)$ dává hodnoty $P(X = x_i|Y = y_j)$ pro každý možný pár i, j
- $P(x, y)$ odpovídá $P(X = x \wedge Y = y)$
- $\mathbf{P}(X, Y)$ označuje pravděpodobnosti pro všechny kombinace hodnot X, Y

(Úplná) nezávislost: náhodné proměnné jsou na sobě nezávislé

$$\mathbf{P}(X|Y) = \mathbf{P}(X) \text{ nebo } \mathbf{P}(Y|X) = \mathbf{P}(Y) \text{ nebo } \mathbf{P}(X, Y) = \mathbf{P}(X)\,\mathbf{P}(Y)$$

Opakování: Úplná sdružená distribuce (joint probability distribution), marginalizace

Pravděpodobnosti elementárních jevů můžeme popsat tabulkou, tzv. úplnou sdruženou distribucí $\mathbf{P}(Toothache, Cavity, Catch)$

	toothache		$\neg$*toothache*	
	catch	$\neg$*catch*	*catch*	$\neg$*catch*
cavity	0.108	0.012	0.072	0.008
$\neg$*cavity*	0.016	0.064	0.144	0.576

Chceme-li znát pravděpodobnost nějakého tvrzení, sečteme pravděpodobnosti všech „světů", kde tvrzení platí (marginalizace)

$$\mathbf{P}(\mathbf{Y}) = \sum_{z \in \mathbf{Z}} \mathbf{P}(\mathbf{Y}, z)$$

- pozn. $\mathbf{Y}$ značí množinu proměnných vs. Y značí jednu proměnnou
- př. $P(tootchache = true) = 0.108 + 0.012 + 0.016 + 0.064 = 0.2$

Opakování: Diagnostická vs. kauzální vazba

- Odhalení zdrojů dle symptomů, tj. diagnostická vazba $P(nemoc|symptomy)$, tj. $P(pricina|nasledek)$
- Z analýzy předchozích nemocí, spíš však máme k dispozici
  - pravděpodobnost nemoci $P(nemoc)$
  - pravděpodobnost symptomu $P(symptom)$
  - kauzální vztah nemocí a symptomu $P(symptomy|nemoc)$, tj. $P(nasledek|pricina)$
- Jak využít tyto znalosti ke zjištění $P(nemoc|symptomy)$?

Opakování: Bayesova věta

Víme, že platí

$$P(a \wedge b) = P(a|b)\,P(b) = P(b|a)\,P(a)$$

Můžeme odvodit Bayesovu větu v obecné podobě:

$$\mathbf{P}(Y|X) = \frac{\mathbf{P}(X|Y)\,\mathbf{P}(Y)}{\mathbf{P}(X)} = \alpha\,\mathbf{P}(X|Y)\,\mathbf{P}(Y)$$

- α je normalizační konstanta, zajišťující, že položky v $\mathbf{P}(Y|X)$ jsou sumarizovány na 1
  - př. $\alpha\,\langle 0.2, 0.3\rangle = \langle 0.4, 0.6\rangle$
- $1/\mathbf{P}(X)$ vlastně dočasně zanedbáme
- podobně u podmíněné pravděpodobnosti

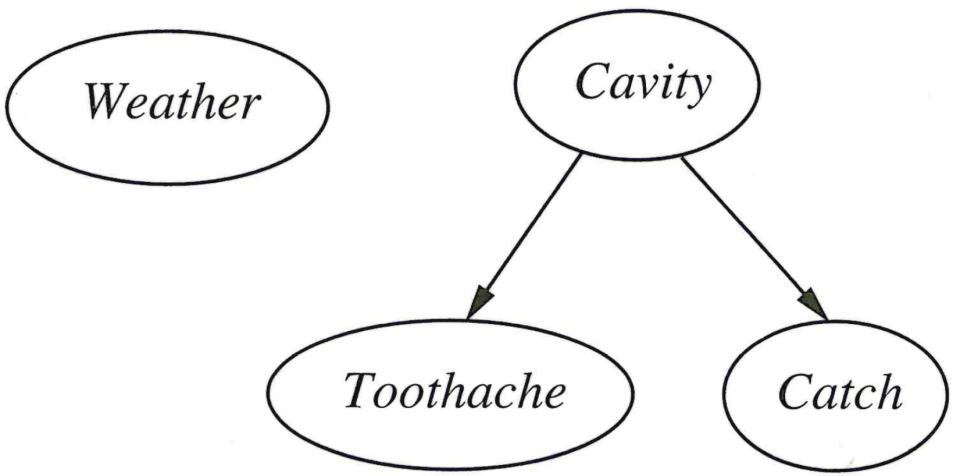
$$\mathbf{P}(Y|X) = \frac{\mathbf{P}(Y,X)}{\mathbf{P}(X)} = \alpha\,\mathbf{P}(Y,X)$$

Obsah přednášky

- Bayesovské sítě
  - efektivní způsob reprezentace podmíněných pravděpodobností a nezávislostí
- Sémantika sítě
  - vztah k úplně sdružené distribuci
- Konstrukce sítě
- Odvozování v Bayesovských sítích
  - exaktní metody
    - enumerace, eliminace proměnných
  - aproximační metody
    - vzorkovací techniky (sampling techniques)

Bayesovská síť (BS)

- Zachycuje závislosti mezi náhodnými proměnnými
- Orientovaný acyklický graf (DAG)
  - uzel odpovídá náhodné proměnné
  - předchůdci uzlu v grafu se nazývají rodiče
  - každý uzel má přiřazenu tabulku podmíněné pravděpodobnostní distribuce $\mathbf{P}(X|Parents(X))$



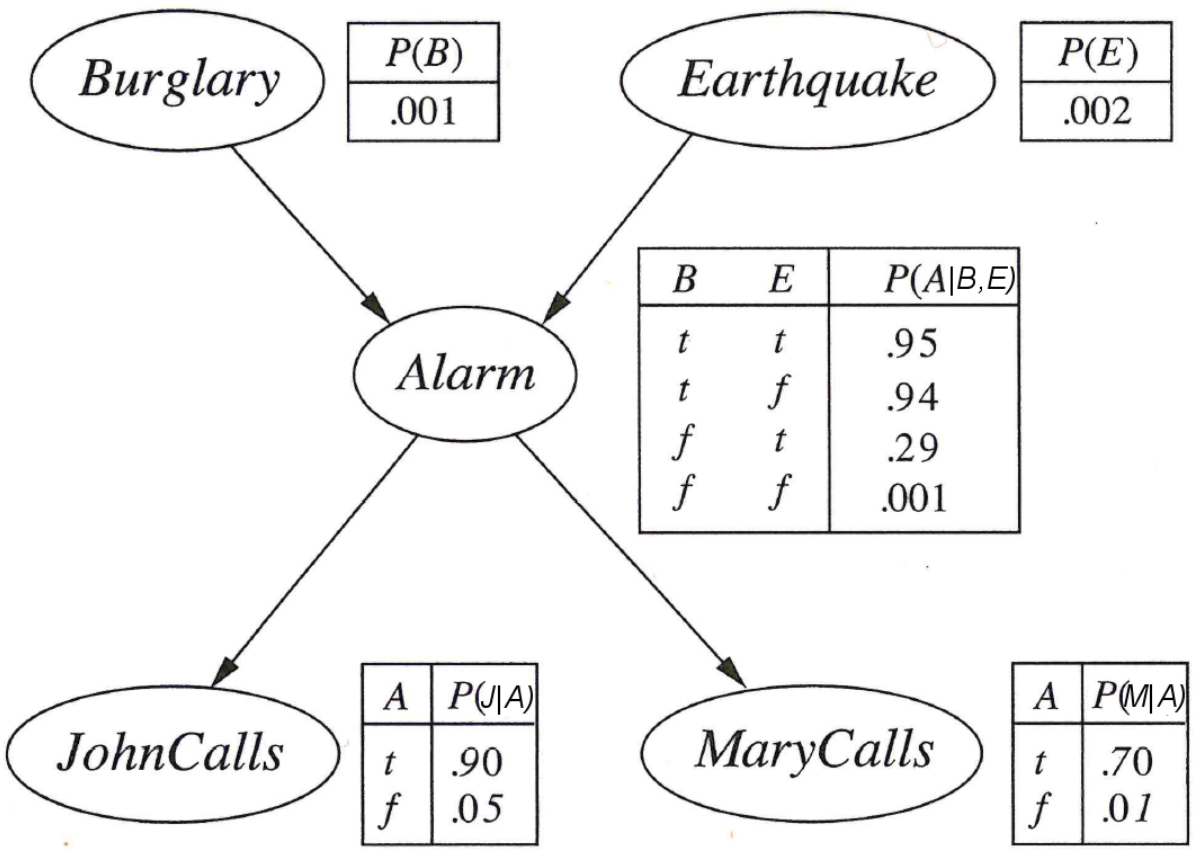
- Jiné názvy
  - belief network, probabilistic network,
    causal network (speciální případ BS), knowledge map

Bayesovská síť: modelová situace

- Máme v domě zabudovaný alarm, který se spustí při vloupání ale někdy také při zemětřesení
- Sousedi Mary a John slíbili, že vždy, když alarm uslyší, tak nám zavolají
  - John volá skoro vždy, když slyší alarm, ale někdy si ho splete s telefonním zvoněním
  - Mary poslouchá hlasitou hudbu a někdy alarm přeslechne
- Zajímá nás pravděpodobnost vloupání, pokud John i Mary volají
- Další předpoklady
  - sousedi přímo nevidí vloupání ani necítí zemětřesení
  - sousedi se nedomlouvají (volají nezávisle na sobě)

Bayesovská síť: příklad

- Náhodné boolovské proměnné reprezentují možné události
  - některé události (zvonění telefonu, přelet letadla, vadu alarmu, ...) ignorujeme
- Pravděpodobnostní tabulky reprezentují vztah podmíněné pravděpodobnosti
  - stačí reprezentovat hodnoty true



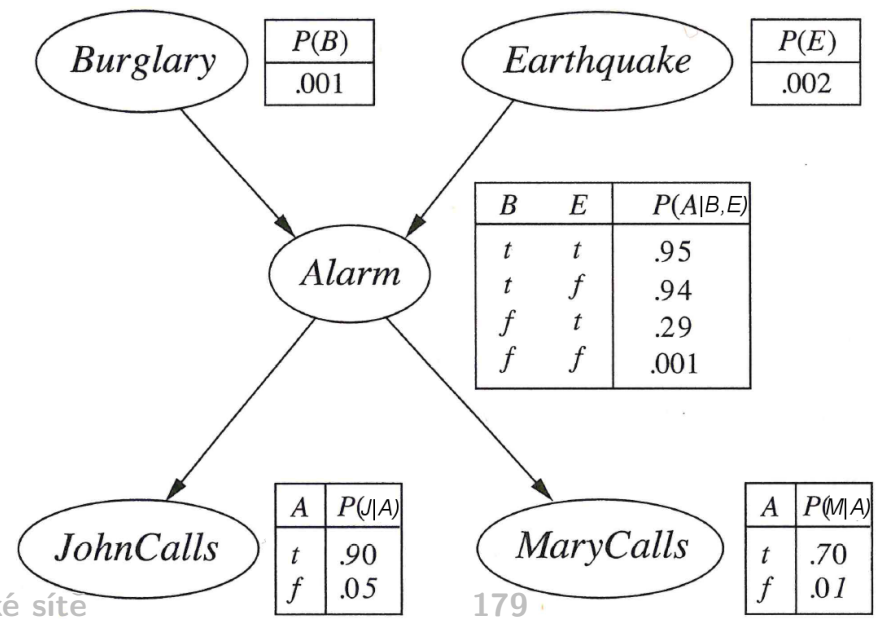
Sémantika sítě

- Bayesovská síť kompaktním způsobem reprezentuje úplnou sdruženou distribuci

$$P(x_1, \ldots, x_n) = \prod_i P(x_i | parents(X_i))$$

$$\begin{aligned} \text{př. } P(j, m, a, \neg b, \neg e) &= P(j|a)\, P(m|a)\, P(a|\neg b \wedge \neg e)\, P(\neg b)\, P(\neg e) \\ &= 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 = 0.000628 \end{aligned}$$

- Zpětně lze ukázat, že tabulky $\mathbf{P}(X|Parents(X))$ jsou podmíněné pravděpodobnosti podle výše uvedené sdružené distribuce
- Protože úplnou sdruženou distribuci lze použít pro odpověď na libovolnou otázku v dané doméně, lze stejnou dopověď získat z Bayesovské sítě (marginalizací)



Konstrukce sítě

Jak konstruovat Baysovské sítě?

Rozepíšeme $P(x_1, \ldots, x_n)$ pomocí tzv. řetězcového pravidla

- $n = 4 : \mathbf{P}(X_4, X_3, X_2, X_1) = \mathbf{P}(X_4|X_3, X_2, X_1)\,\mathbf{P}(X_3|X_2, X_1)\,\mathbf{P}(X_2|X_1)\,\mathbf{P}(X_1)$

$$P(x_1, \ldots, x_n) = \prod_i P(x_i|x_{i-1}, \ldots x_1)$$

Dále dostaneme

$$\mathbf{P}(X_i|X_{i-1}, \ldots X_1) = \mathbf{P}(X_i|Parents(X_i))$$

za předpokladu $Parents(X_i) \subseteq \{X_{i-1}, \ldots, X_1\}$, což platí, pokud je očíslování uzlů konzistentní s uspořádáním uzlů v síti

- pozn. $\mathbf{P}(X_4|X_3, X_2, X_1) = \mathbf{P}(X_4|X_3, X_2)$ pokud X_4 nezávisí na X_1
- tj. Bayesovská síť je korektní, pokud je každý uzel nezávislý na ostatních předchůdcích v uspořádání

Tj. sdružená distribuce

$$P(x_1, \ldots, x_n) = \prod_i P(x_i|parents(X_i))$$

odpovídá přidání $\mathbf{P}(X_i|Parents(X_i))$ pro každý uzel sítě X_i.

Konstrukce sítě: algoritmus

Uzly: rozhodněte, jaké náhodné proměnné jsou potřeba a uspořádejte je

- funguje libovolné uspořádání, ale pro různá uspořádání dostaneme různě kompaktní sítě
- doporučené uspořádání je takové, kdy příčiny předcházejí efekty

Hrany: bereme proměnné X_i v daném pořadí od 1 do n

- v množině $\{X_1, \ldots, X_{i-1}\}$ vybereme nejmenší množinu rodičů X_i tak, že platí $\mathbf{P}(X_i|Parents(X_i)) = \mathbf{P}(X_i|X_{i-1}, \ldots X_1)$
  - intuitivně: rodiči X_i jsou pouze ty uzly, které *přímo* ovlivňují X_i
  - M ovlivněno B a E ale nepřímo, tj. věříme, že $\mathbf{P}(M|J, A, E, B) = \mathbf{P}(M|A)$
- z rodičů vedeme hranu do X_i
- vypočteme podmíněné pravděpodobnostní tabulky $\mathbf{P}(X_i|Parents(X_i))$

Vlastnosti

- síť je z principu konstrukce acyklická
- síť neobsahuje redundantní informaci a tudíž je vždy konzistentní (splňuje axiomy pravděpodobnosti)

Konstrukce sítě: poznámky

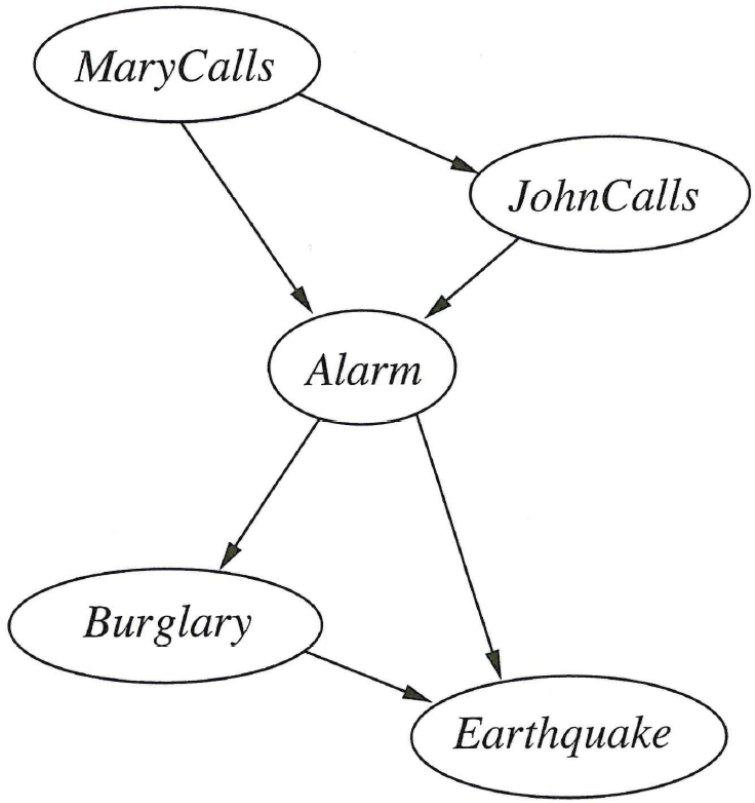
Bayesovská síť může být mnohem kompaktnější než úplná sdružená distribuce, pokud je síť řídká (je lokálně strukturovaná)

- náhodná proměnná často závisí jen na omezeném počtu jiných proměnných
- nechť je takových proměnných k a celkem máme n proměnných, pak potřebujeme prostor
  - $n.2^k$ pro Bayesovskou síť
  - 2^n pro uplnou sdruženou distribuci
- můžeme ignorovat „slabé vazby", čímž budeme mít menší přesnost reprezentace, ale reprezentace bude kompaknější
  - př. nebudeme brát v úvahu, zda Mary nebo John volají kvůli zemětřesení
- samozřejmě kompaktnost sítě hodně závisí na vhodném uspořádání proměnných

Konstrukce sítě: příklad

Nechť jsme zvolili pořadí MarryCalls, JohnCalls, Alarm, Burglary, Earthquake

- MarryCalls nemá rodiče
- pokud volá Marry, je zřejmě aktivní alarm, což ovlivňuje Johnovo zavolání
- Alarm asi zní, pokud volá Marry nebo John
- pokud známe stav Alarmu, tak vloupání nezávisí na tom, zda volá Marry nebo John
  - $P(Burglary|Alarm, JohnCalls, MarryCalls) = P(Burglary|Alarm)$
- Alarm je svým způsobem detektor zemětřesení, ale pokud došlo k vloupání, tak pravděpodobně nebylo zemětřesení



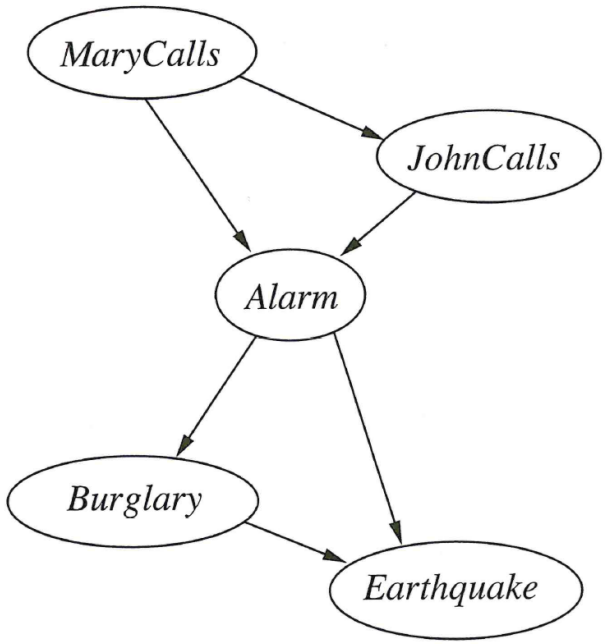
Konstrukce sítě: uspořádání proměnných

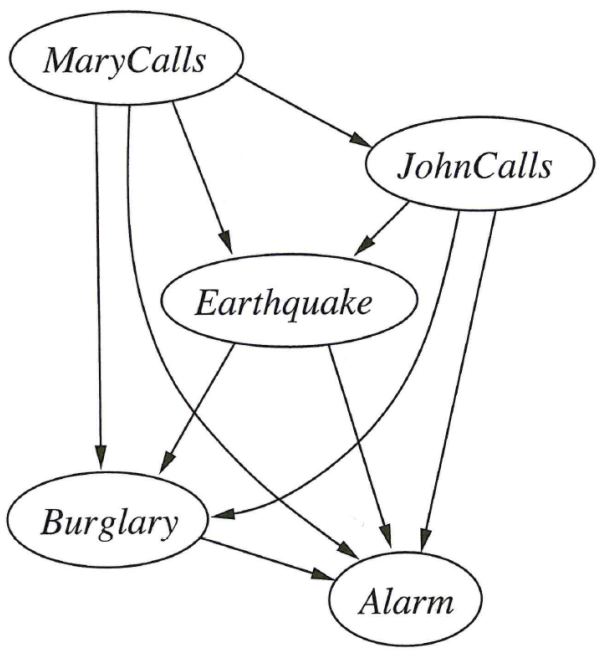
Máme sice jen dvě hrany navíc oproti původnímu návrhu, ale problém je vyplnění tabulek podmíněných závislostí

- stejný problém jako u kauzální vs. diagnostické vazby
- diagnostickou vazbu $P(pricina|nasledek)$ často neznáme
- je lepší držet se kauzální vazby (příčina před následkem) $P(nasledek|pricina)$
  - dává menší sítě a je snažší vyplnit tabulky podmíněných závislostí

Při „špatném" uspořádání proměnných nemusíme nic uspořit vzhledem k úplné sdružené distribuci

- MarryCalls, JohnCalls, Eartquake, Burglary, Alarm





Odvozování v Bayesovských sítí pomocí enumerace

- Připomeňme, k čemu mají Bayesovské sítě sloužit:
  zjistit pravděpodobnostní distribuce náhodných proměnných X
  v dotazu za předpokladu znalosti hodnot $\mathbf{e}$ proměnných z pozorování
  (ostatní proměnné jsou skryté).

$$\mathbf{P}(X|\mathbf{e}) = \alpha\,\mathbf{P}(X,\mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X,\mathbf{e},\mathbf{y})$$

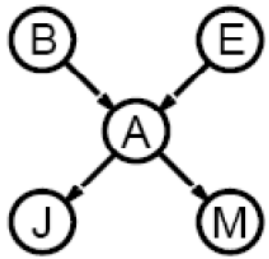
- Hodnotu $P(X,e,y)$ zjistíme z Bayesovské sítě
  $P(x_1,\ldots,x_n) = \prod_i P(x_i|parents(X_i))$

- Můžeme ještě vhodně přesunout
  některé členy $P(x_i|parents(X_i))$ před součty
  - viz příklad na další straně

Odvozování enumerací: příklad

- Nechť máme dotaz, zda došlo k vloupání, pokud Marry i John volají

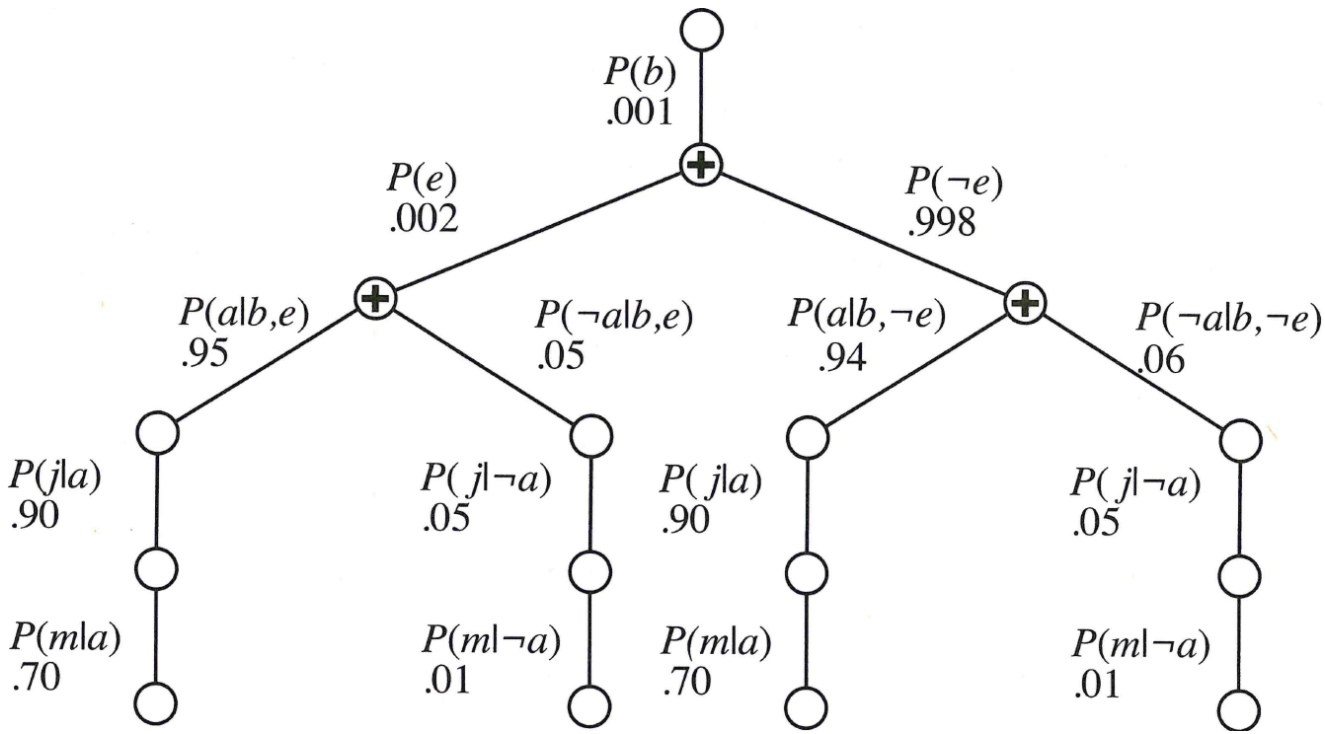
$$P(b|j,m) = \alpha \sum_e \sum_a P(b)\,P(e)\,P(a|b,e)\,P(j|a)\,P(m|a)$$
$$= \alpha P(b) \sum_e P(e) \sum_a P(a|b,e)\,P(j|a)\,P(m|a)$$



  - Earthquake a Alarm jsou skryté proměnné

  Strukturu výpočtu můžeme zachytit stromovou strukturou
  - je to hodně podobné řešení CSP a SAT

- Všimněme si, že některé části výpočtu se opakují!
- Evaluace se provádí shora dolů násobením hodnot po každé cestě a sečítáním hodnot v „+" uzlech



Odvozování enumerací: algoritmus

---

function enumerace($X, \mathbf{e}, bn$) **returns** distribuci pro X
(X: *náhodná proměnná*
$\mathbf{e}$: *hodnoty proměnných z pozorování pro* $\boldsymbol{E}$
bn: BS s proměnnými $\{X\} \cup \boldsymbol{E} \cup \boldsymbol{Y}$, *kde* $\boldsymbol{Y}$ *jsou skryté proměnné)*
$\mathbf{Q}(X) =$ distribuce pro X, iniciálně prázdná;
forall hodnoty x_i proměnné X **do**
$\mathbf{Q}(x_i) =$ enumerace-hodnot($bn.VARS, \mathbf{e}_{x_i}$), kde $\mathbf{e}_{x_i}$ je $\mathbf{e}$ rozšířeno o $X = x_i$;
return normalizace($\mathbf{Q}(X)$)

---

function enumerace-hodnot($vars, \mathbf{e}$) **returns** reálné číslo
if $vars = \emptyset$ **then return** 1.0;
Y = first($vars$);
if Y má hodnotu y v $\mathbf{e}$
then return $P(y|parents(Y)) \times$ enumerace-hodnot(rest($vars$), $\mathbf{e}$)
else return $\sum_y P(y|parents(Y)) \times$ enumerace-hodnot(rest($vars$), $\mathbf{e}_y$)
kde $\mathbf{e}_y$ je $\mathbf{e}$ rozšířeno o $Y = y$;

Eliminace proměnných

- Enumerační metoda zbytečně opakuje některé výpočty
- Stačí si výsledek zapamatovat a následně použít
  - $\mathbf{f}_i$ spočítáme jednou a uschováme pro budoucí použití
    dynamické programování

$$\begin{aligned}\mathbf{P}(B|j,m) &= \alpha\,\mathbf{P}(B)\sum_e P(e)\sum_a \mathbf{P}(a|B,e)\,P(j|a)\,P(m|a)\\ &= \alpha\,\mathbf{f}_1(B)\sum_e \mathbf{f}_2(E)\sum_a \mathbf{f}_3(A,B,E)\,\mathbf{f}_4(A)\,\mathbf{f}_5(A)\end{aligned}$$

- Činitelé $\mathbf{f}_i$ jsou matice (tabulky) pro dané proměnné
- Vyhodnocení provedeme zprava doleva (viz příklad dále)
  - násobení činitelů je násobení po prvcích (ne násobení matic)
  - vysčítáním činitelů eliminujeme příslušnou proměnnou
  - na závěr provedeme normalizaci

tj. bottom-up přístup pro předchozí graf u odvozování enumerací

Eliminace proměnných: operace s tabulkami

- Máme-li dvě tabulky, potom jejich součin je tabulka nad sjednocením proměnných z obou tabulek

$$\mathbf{f}(A_1, \ldots, A_j, B_1, \ldots, B_k, C_1, \ldots, C_l) = \mathbf{f}_1(A_1, \ldots, A_j, B_1, \ldots, B_k) \times \mathbf{f}_2(B_1, \ldots, B_k, C_1, \ldots, C_l)$$

A	B	f_1(A,B)	B	C	f_2(B,C)	A	B	C	f_3(A,B,C)
T	T	0.3	T	T	0.2	T	T	T	0.06 = 0.3*0.2
T	F	0.7	T	F	0.8	T	T	F	0.24 = 0.3*0.8
F	T	0.9	F	T	0.6	T	F	T	0.42 = 0.7*0.6
F	F	0.1	F	F	0.4	T	F	F	0.28 = 0.7*0.4
						F	T	T	0.18 = 0.9*0.2
						F	T	F	0.72 = 0.9*0.8
						F	F	T	0.06 = 0.1*0.6
						F	F	F	0.04 = 0.1*0.4

- Při vysčítání dojde k eliminaci proměnné

$$\mathbf{f}(B, C) = \sum_a \mathbf{f}_3(A, B, C) = \mathbf{f}_3(a, B, C) + \mathbf{f}_3(\neg a, B, C)$$

$$\begin{bmatrix} 0.06 & 0.24 \\ 0.42 & 0.28 \end{bmatrix} + \begin{bmatrix} 0.18 & 0.72 \\ 0.06 & 0.04 \end{bmatrix} = \begin{bmatrix} 0.24 & 0.96 \\ 0.48 & 0.32 \end{bmatrix}$$

Cvičení: eliminace proměnných

Spočtěte hodnodu $\mathbf{P}(B|j,m)$

Máme:

$$\mathbf{P}(B|j,m) = \alpha\, \mathbf{f}_1(B) \times \sum_e \mathbf{f}_2(E) \times \sum_a \mathbf{f}_3(A,B,E) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A)$$

Odvodíme (eliminujeme A):

$$\mathbf{f}_6(B,E) = (\mathbf{f}_3(a,B,E) \times \mathbf{f}_4(a) \times \mathbf{f}_5(a)) + (\mathbf{f}_3(\neg a,B,E) \times \mathbf{f}_4(\neg a) \times \mathbf{f}_5(\neg a))$$

Tedy:

$$\mathbf{P}(B|j,m) = \alpha\, \mathbf{f}_1(B) \times \sum_e \mathbf{f}_2(E) \times \mathbf{f}_6(B,E)$$

Dále (eliminujeme E):

$$\mathbf{f}_7(B) = \sum_e \mathbf{f}_2(E) \times \mathbf{f}_6(B,E) = \mathbf{f}_2(e) \times \mathbf{f}_6(B,e) + \mathbf{f}_2(\neg e) \times \mathbf{f}_6(B,\neg e)$$

Tedy:

$$\mathbf{P}(B|j,m) = \alpha\, \mathbf{f}_1(B) \times \mathbf{f}_7(B)$$

Zkuste: $\mathbf{P}(B|j,m) = \alpha\, \mathbf{f}_1(B) \times \sum_a \mathbf{f}_4(A) \times \mathbf{f}_5(A) \times \sum_e \mathbf{f}_2(E) \times \mathbf{f}_3(A,B,E)$

Eliminace proměnných: algoritmus

---

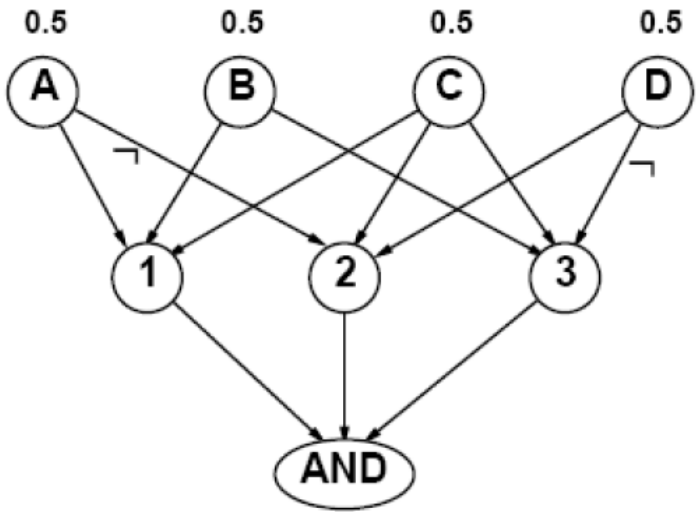
function eliminace(X, **e**, bn) **returns** distribuci pro X
  (*X: náhodná proměnná*
  e**: hodnoty proměnných z pozorování pro **E
  bn: BS s úplnou sdruženou distribucí $\boldsymbol{P}(X_1, \ldots, X_n)$)
  tabulky $=$ [];
  forall *var* $\in$ order($bn.VARS$) **do**
    tabulky $=$ [vytvoř-tabulku(*var*, **e**)|*tabulky*];
    if *var* je skrytá proměnná **then** *tabulky* $=$ marginalizace(*var*, *tabulky*);
  return normalizace(součin-po-prvcích(*tabulky*))

- Algoritmus funguje pro libovolné uspořádání proměnných (order)
- Složitost je dána velikostí největšího činitele (tabulky) v průběhu výpočtu
- Vhodné je proto pro eliminaci vybrat proměnnou, jejíž eliminací vznikne nejmenší tabulka

Složitost problému

- Eliminace proměnných urychluje odvozování, ale jak moc?
- Pokud je Bayesovská síť poly-strom (mezi každými dvěma vrcholy vede maximálně jedna neorientovaná cesta), potom je časová a prostorová složitost odvozování lineární vzhledem k velikosti sítě (tj. velikosti tabulek)
- Pro více-propojené sítě je to horší
  - 3SAT lze redukovat na odvození v Bayesovské síti, takže odvození je NP-těžké

1. A v B v C
2. C v D v ¬A
3. B v C v ¬D



  - odvozování v Bayesovské síti je ekvivalentní zjištění počtu řešení SAT-formule, je tedy striktně těžší než NP-úplné problémy

Vzorkovací metody

- Exaktní odvozování je výpočetně náročné, můžeme ale použít aproximační techniky založené na metodě Monte Carlo
- Monte Carlo algoritmy slouží pro odhad hodnot, které je těžké spočítat exaktně
  - vygeneruje se množství vzorků
    - vzorek = ohodnocení náhodných proměnných
  - hledaná hodnota se zjistí statisticky
  - více vzorků = větší přesnost

- Pro Bayesovské sítě ukážeme přístupy
  - příme vzorkování
  - vzorkování se zamítáním
  - vzorkování s vážením věrohodností
- Další zajímavé metody (viz Russel & Norvig)
  - vzorkování s Markovovskými řetězci

Přímé vzorkování

- Vzorkem pro nás bude ohodnocení náhodných proměnných
- Vzorek je potřeba generovat tak, aby „odpovídal" tabulkám v Bayesovské síti
  - uzly (proměnné) bereme v topologickém uspořádání
  - ohodnocení rodičů nám dá pravděpodobnostní distribuci hodnot aktuální náhodné proměnné
  - náhodně vybereme hodnotu podle této distribuce
- Nechť N je počet vzorků a $N(x_1, \ldots, x_n)$ vyjadřuje počet výskytů jevu $x_1, \ldots, x_n$, potom $P(x_1, \ldots, x_n) = \lim_{N\to\infty}(N(x_1, \ldots, x_n)/N)$

---

function vzorky(bn) **returns** vzorek získaný dle bn
(bn: BS specifikující úplnou sdruženou distribuci $\mathbf{P}(X_1, \ldots, X_n)$)
forall proměnnou X_i z $X_1, \ldots, X_n$ **do**
x_i = náhodný vzorek z $\mathbf{P}(X_i|parents(X_i))$;
return x

Přímé vzorkování: příklad

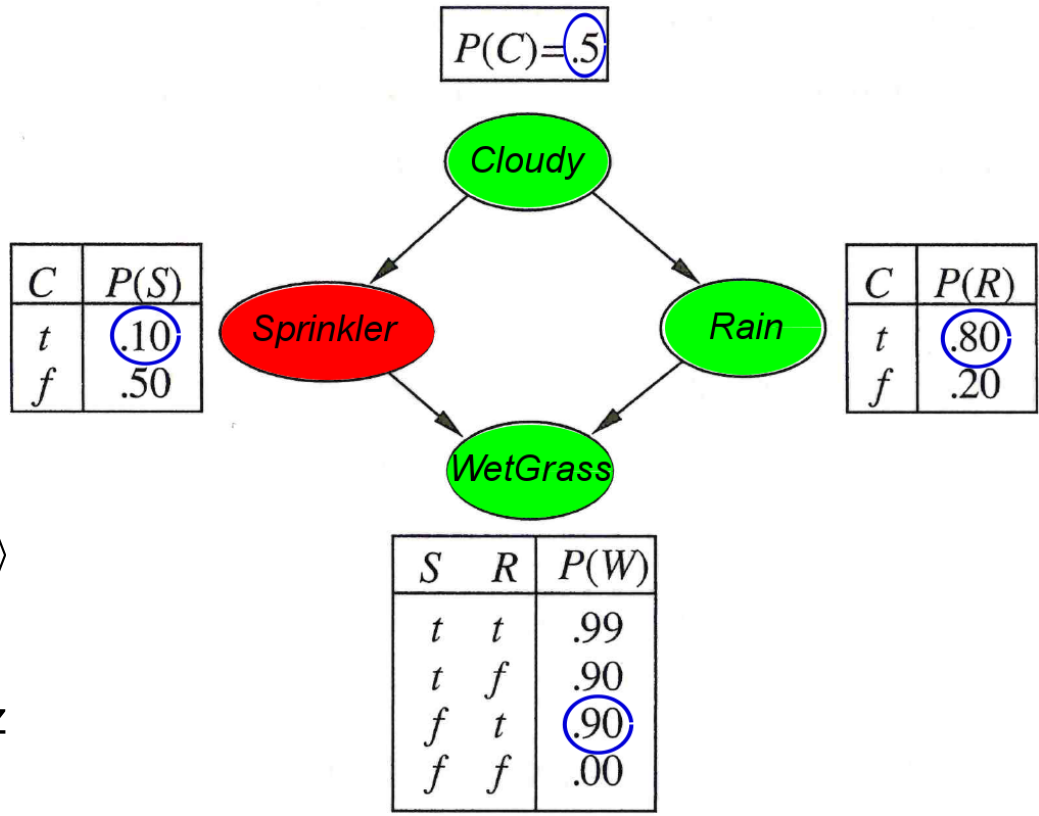
- Vybereme hodnotu pro *Cloudy* z distribuce $\mathbf{P}(Cloudy) = \langle 0.5, 0.5 \rangle$

  nechť je true
- Vybereme hodnotu pro *Sprinkler* z distribuce $\mathbf{P}(Sprinkler|Cloudy = true) = \langle 0.1, 0.9 \rangle$

  nechť je false
- Vybereme hodnotu pro *Rain* z distribuce $\mathbf{P}(Rain|Cloudy = true) = \langle 0.8, 0.2 \rangle$

  nechť je true
- Vybereme hodnotu pro *WetGrass* z distribuce $\mathbf{P}(WetGrass|Sprinkler = false, Rain = true) = \langle 0.9, 0.1 \rangle$

  nechť je true



Získali jsme vzorek $Cloudy = true, Sprinkler = false, Rain = true, WetGrass = true$

Pravděpodobnost jeho získání je zřejmě $0.5 * (1 - 0.1) * 0.8 * 0.9 = 0.324$

Vzorkování se zamítáním

- Nás ale zajímá $\mathbf{P}(X|e)$!
- Ze vzorků, které vygenerujeme, vezmeme jen ty, které jsou kompatibilní s e (ostatní zamítneme) $\mathbf{P}(X|\mathbf{e}) \approx \alpha\, \mathbf{N}(X, \mathbf{e}) = \mathbf{N}(X, \mathbf{e})/N(\mathbf{e})$

---

function vzorky-zamítání($X, \mathbf{e}, bn, N$) **returns** odhad pro $\mathbf{P}(X|\mathbf{e})$
(X: náhodná proměnná
$\mathbf{e}$: hodnoty proměnných z pozorování pro $\boldsymbol{E}$
bn: BS s úplnou sdruženou distribucí $\boldsymbol{P}(X_1, \ldots, X_n)$
N: celkový počet vzorků, který máme generovat)
forall hodnotu x v X **do** $\mathbf{N}[x] = 0$; *(vektor počtu hodnot inicializujeme na 0)*
for $j = 1$ **to** N **do**
  $\mathbf{x}$ = vzorky(bn);
  if $\mathbf{x}$ je konzistentní s $\mathbf{e}$ **then**
    $\mathbf{N}[x] = \mathbf{N}[x] + 1$, kde x je hodnota X v $\mathbf{x}$;
return normalizace($\mathbf{N}$)

- Nechť v našem příkladu vygenerujeme 100 vzorků, z toho u 27 platí $Sprinkler = true$ a z nich u 8 je $Rain = true$ a u 19 je $Rain = false$. Potom $\mathbf{P}(Rain|Sprinkler = true) \approx$ normalizace($\langle 8, 19\rangle$) $= \langle 0.296, 0.704\rangle$
- Hlavní nevýhoda metody je zamítání příliš mnoha vzorků!

Vážení věrohodností

Místo zamítání vzorků je efektivnější: generovat pouze vzorky vyhovující pozorování **e**

- Zafixujeme hodnoty z pozorování **e** a vzorkujeme pouze ostatní proměnné
- Pravděpodobnost získání vzorku je (mezi rodiči máme i pozorování)

$$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_i P(z_i | parents(Z_i))$$

- To ale není to, co potřebujeme! Ještě nám chybí

$$w(\mathbf{z}, \mathbf{e}) = \prod_j P(e_j | parents(E_j)) \qquad \text{(mezi rodiči máme i } Z_i\text{)}$$

- Tedy

$$S_{WS}(\mathbf{z}, \mathbf{e}) w(\mathbf{z}, \mathbf{e}) = \prod_i P(z_i | parents(Z_i)) \prod_j P(e_j | parents(E_j)) = P(\mathbf{z}, \mathbf{e})$$

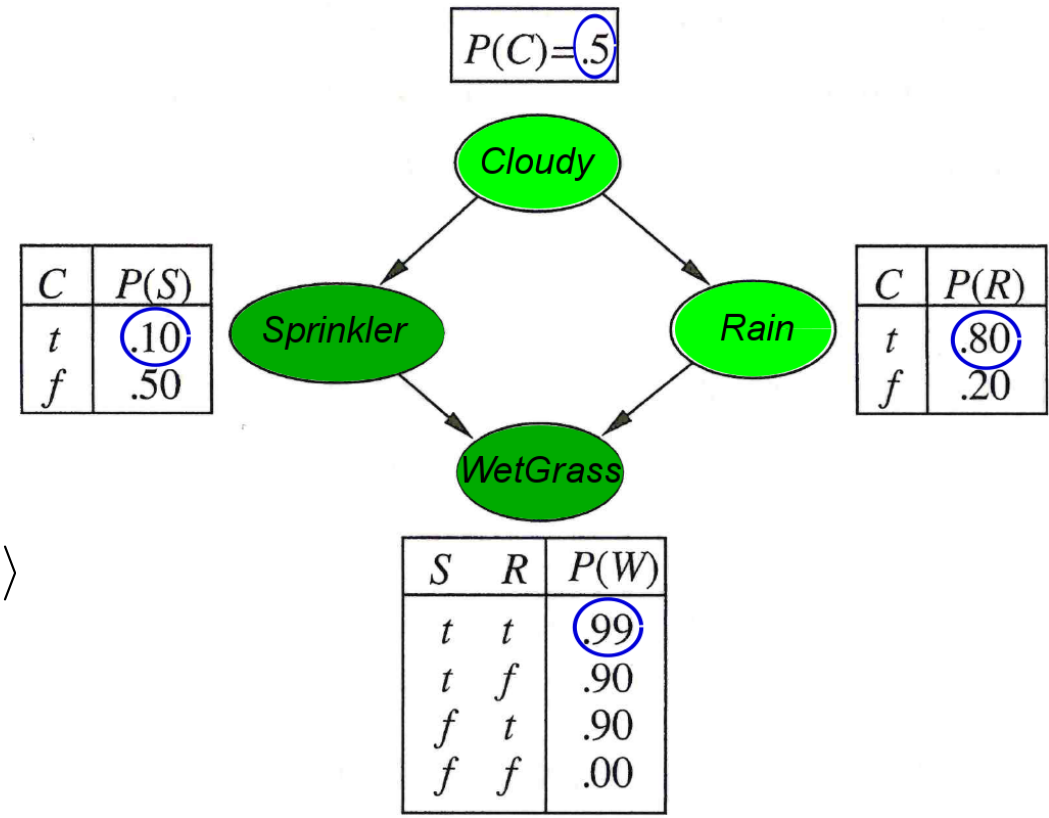
- Každý vzorek tedy doplníme o příslušnou váhu

$$\mathbf{P}(X|\mathbf{e}) \approx \alpha \sum_{\mathbf{y}} \mathbf{N}(X, \mathbf{y}, \mathbf{e})\, w(X, \mathbf{y}, \mathbf{e})$$

Vážení věrohodností: příklad

Nechť zpracováváme dotaz $\mathbf{P}(Rain|Sprinkler = true, WetGrass = true)$

- Počáteční váha vzorku $w = 1.0$
- Vybereme hodnotu pro *Cloudy* z distribuce $\mathbf{P}(Cloudy) = \langle 0.5, 0.5 \rangle$ nechť je true
- Hodnotu $Sprinkler = true$ známe, ale upravíme váhu $w = w \times P(Sprinkler = true|Cloudy = true) = 0.1$
- Vybereme hodnotu pro *Rain* z distribuce $\mathbf{P}(Rain|Cloudy = true) = \langle 0.8, 0.2 \rangle$ nechť je true
- Hodnotu $WetGrass = true$ známe, ale upravíme váhu $w = w \times P(WetGrass = true|Sprinkler = true, Rain = true) = 0.099$



Získali jsme vzorek $Cloudy = true, Sprinkler = true, Rain = true, WetGrass = true$, který má váhu 0.099

Vážení věrohodností: algoritmus

---

function vážení-věrohodností($X, \mathbf{e}, bn, N$) **returns** odhad pro $\mathbf{P}(X|\mathbf{e})$
  (X: náhodná proměnná
  $\mathbf{e}$: hodnoty proměnných z pozorování pro $\boldsymbol{E}$
  bn: BS s úplnou sdruženou distribucí $\boldsymbol{P}(X_1, \ldots, X_n)$
  N: celkový počet vzorků, který máme generovat)
  forall hodnotu x v X **do** $\mathbf{W}[x] = 0$; *(vektor vah inicializujeme na 0)*
  for $j = 1$ **to** N **do**
    $\mathbf{x}, w =$ vážený-vzorek($bn, \mathbf{e}$);
    $\mathbf{W}[x] = \mathbf{W}[x] + w$, kde x je hodnota X v $\mathbf{x}$;
  return normalizace($\mathbf{W}$)

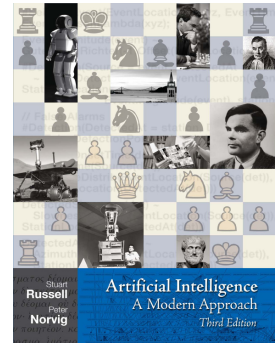
---

function vážený-vzorek($bn, \mathbf{e}$) **returns** vzorek a váhu
  forall X_i v $X_1, \ldots, X_n$ **do**
    if X_i má v $\mathbf{e}$ hodnotu x_i
    then $w = w \times P(X_i = x_i | parents(X_i))$
    else $x_i =$ náhodný vzorek z $\mathbf{P}(X_i | parents(X_i))$;
  returns $\mathbf{x}, w$

Čas a neurčitost

8. ledna 2021

Zdroje: *Roman Barták, přednáška přednáška Umělá inteligence II, Matematicko-fyzikální fakulta, Karlova univerzita v Praze, 2015.*
http://kti.ms.mff.cuni.cz/~bartak/ui2



Pravděpodobnostní uvažování o čase

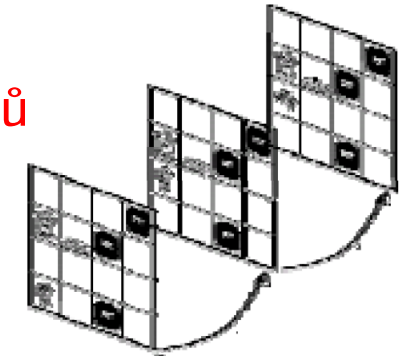
Agent pracující v částečně pozorovatelném prostředí udržuje na základě senzorického modelu domnělý stav a na základě přechodového modelu odhaduje, jak se svět může vyvíjet

- domnělý stav reprezentuje možné skutečné stavy světa buď výčtově nebo logickou formulí (přes vlastnosti stavu)
- pomocí teorie pravděpodobnosti umíme kvantifikovat, které z domnělých stavů jsou pravděpodobnější
- teď přidáme pravděpodobnost k přechodům mezi stavy

Pravděpodobnostní uvažování o čase: obsah přednášky

- reprezentace přechodů s pravděpodobností
- typy řešených úloh (otázek)
- základní odvozovací algoritmy

Čas a neurčitost



- Dynamiku světa budeme modelovat sérií časových řezů
- t je označení časového řezu
  - uvažujeme tedy diskrétní čas se stejnými časovými kroky
- Každý řez/stav bude popsán (stejnou) množinou náhodných proměnných, které se dělí na
  - nepozorovatelné náhodné proměnné $\mathbf{X}_t$
  - pozorované náhodné proměnné $\mathbf{E}_t$
    (*observable evidence variables*)
    - konkrétní pozorovanou množinu hodnot označíme $\mathbf{e}_t$
- Notace
  - $X_{t1:t2}$ označujeme množinu proměnných od X_{t1} po X_{t2}

Modelová situace

- Uvažujme agenta pracujícího na tajné podzemní základně, 
  ze které nikdy nevychází
- Zajímá ho, zda venku prší
  - náhodná nepozorovatelná proměnná $\mathbf{R}_t$
- Jediné pozorování, které má k dispozici, říká,
  zda ráno ředitel přišel s deštníkem nebo bez deštníku
  - náhodná pozorovaná proměnná $\mathbf{U}_t$

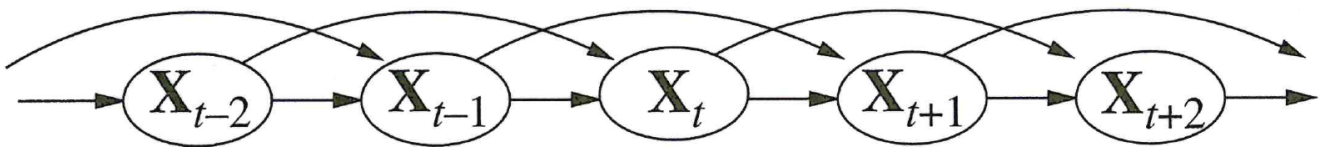
Přechodový model

Přechodový model popisuje, jak je stav ovlivněn předchozími stavy
Přesněji popisuje pravděpodobnostní distribuci $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1})$

1. problém: s rostoucím t neomezeně roste množina $\mathbf{X}_{0:t-1}$
   - použijeme Markovský předpoklad: současný stav závisí pouze na pevně daném konečném počtu předchozích stavů – hovoříme potom o obecných Markovských řetězcích/procesech
   - např. současný stav závisí pouze na předchozím stavu – Markovský proces prvního řádu $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$

Markovský proces prvního řádu

Markovský proces druhého řádu



2. problém: pořád máme nekonečně mnoho různých přechodů
   - použijeme předpoklad stacionárního procesu, tj. stav se vždy mění podle stejných pevně daných pravidel
   - distribuce $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$ je stejná pro všechny časy t

Senzorický model

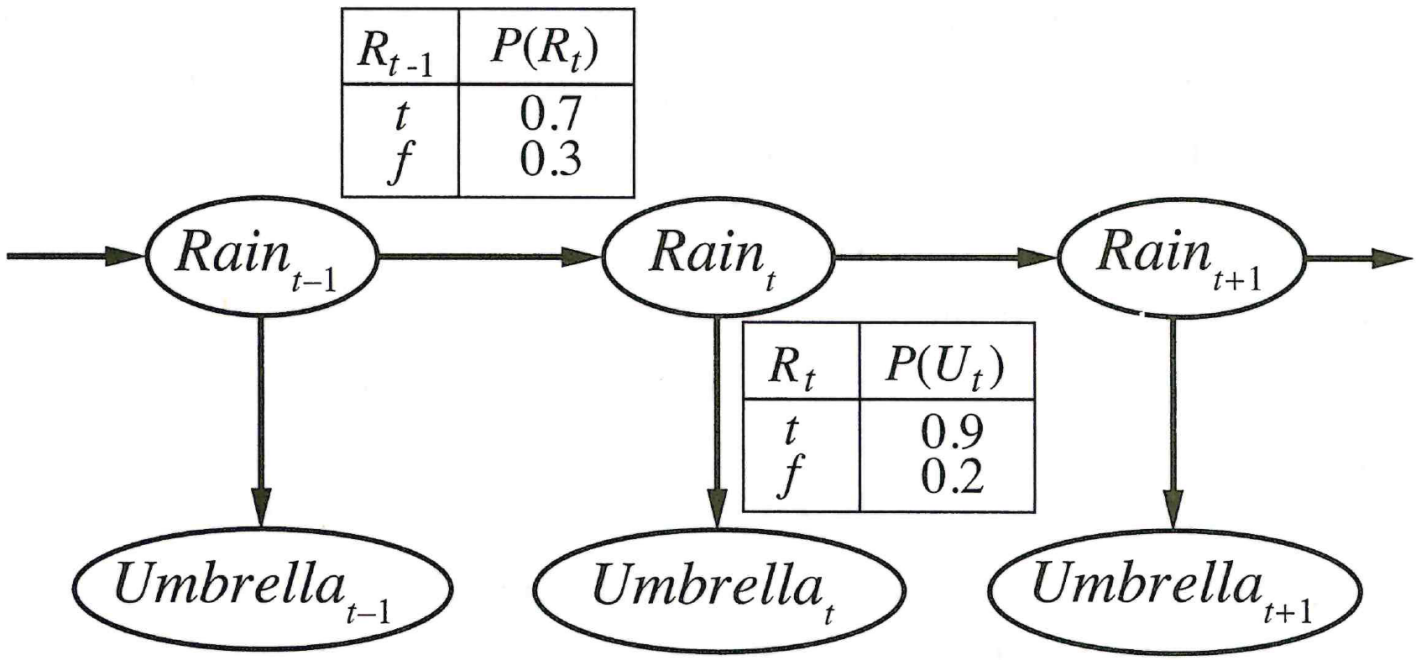
Senzorický model popisuje, na čem závisí pozorované náhodné proměnné $\mathbf{E}_t$

- Ty mohou záviset i na proměnných z předchozích stavů, ale uděláme Markovský senzorický předpoklad – pozorované proměnné závisí pouze na nepozorovatelných proměnných $\mathbf{X}_t$ ze stejného stavu

$$\mathbf{P}(\mathbf{E}_t|\mathbf{X}_{0:t}, E_{1:t-1}) = \mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$$

Bayesovské sítě pro přechodový a senzorický model

- Přechodový a senzorický model můžeme popsat Bayesovskou sítí
- Kromě tabulek $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$ a $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$ musíme ještě zadat, jak to vše začalo: $\mathbf{P}(\mathbf{X}_0)$



- Z vlastností Bayesovských sítí víme

$$\mathbf{P}(\mathbf{X}_{0:t}, \mathbf{E}_{1:t}) = \mathbf{P}(\mathbf{X}_0) \prod_{i=1}^{t} \mathbf{P}(\mathbf{X}_i|\mathbf{X}_{i-1})\, \mathbf{P}(\mathbf{E}_i|\mathbf{X}_i)$$

Zpřesnění modelu

Markovský proces prvního řádu předpokládá, že stavové proměnné obsahují veškerou informaci pro charakteristiku pravděpodobnostní distribuce dalšího stavu

Co když je tento předpoklad nepřesný?

- můžeme zvýšit řád Markovského procesu
- můžeme rozšířit množinu stavových proměnných
  - např. přidáme proměnnou *Season* nebo sadu proměnných $Temperature_t, Humidity_t, Pressure_t$
  - první případ (větší řád) lze vždy převést na druhý případ (více proměnných)

Řešené úlohy

- Odhad stavu (filtrace): cílem je zjištění pravděpodobnosti aktuálního stavu na základě dosavadních pozorování:

$$\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$$

- Predikce: cílem je zjištění pravděpodobnosti budoucího stavu na základě dosavadních pozorování:

$$\mathbf{P}(\mathbf{X}_{t+k}|\mathbf{e}_{1:t}) \text{ pro } k > 0$$

- Vyhlazování: cílem je zjištění pravděpodobnosti minulého stavu na základě dosavadních pozorování:

$$\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t}) \text{ pro } k, \text{ kde } 0 \leq k < t$$

- Nejpravděpodobnější průchod: na základě posloupnosti pozorování chceme zjistit nejpravděpodobnější posloupnost stavů, která tato pozorování generuje:

$$\text{argmax}_{\mathbf{x}_{1:t}} P(\mathbf{x}_{1:t}|\mathbf{e}_{1:t})$$

Odhad stavu (filtrace)

- Úkolem je zjistit pravděpodobnost aktuálního stavu na základě dosavadních pozorování: $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$
- Dobrý filtrační algoritmus odhaduje aktuální stav z odhadu předchozího stavu a aktuálního pozorování (rekurzivní odhad) $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, \mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t}))$

Jak najdeme funkci f?

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}, \mathbf{e}_{t+1}) =$$

$$= \alpha\, \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}, \mathbf{e}_{1:t})\, \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$$

z Bayesovy věty $\mathbf{P}(Y|X) = \alpha\, \mathbf{P}(X|Y)\, \mathbf{P}(Y)$,
kde $Y = \mathbf{X}_{t+1}$ a $X = \mathbf{e}_{t+1}$

$$= \alpha\, \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})\, \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$$ z Mark. senzorického předpokladu,
kde $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$ je jednokroková predikce následujícího stavu

$$= \alpha\, \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, \mathbf{e}_{1:t})\, P(\mathbf{x}_t|\mathbf{e}_{1:t})$$

z marginalizace $\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y}, \mathbf{z}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y}|\mathbf{z})\, P(\mathbf{z})$ a podmíněné pst.
kde $\mathbf{Y} = \mathbf{X}_{t+1}$ a $\mathbf{z} = \mathbf{x_t}$

$$= \alpha\, \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)\, P(\mathbf{x}_t|\mathbf{e}_{1:t})$$ z Mark.procesu 1.řádu

Filtrace (dokončení)

Máme tedy:

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = \alpha\, \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)\, P(\mathbf{x}_t|\mathbf{e}_{1:t})$$

- α normalizuje pravděpodobnosti, aby byl součet 1
- $\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})$: známe ze senzorického modelu
- $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)$: známe z přechodového modelu
- $P(\mathbf{x}_t|\mathbf{e}_{1:t})$: z rekurze

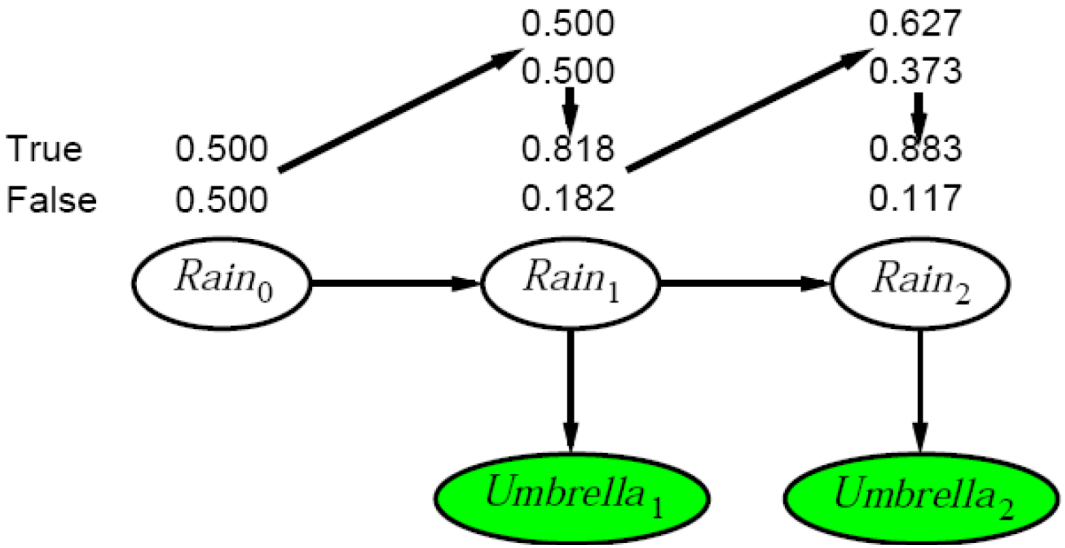
Tedy získali jsme rekurzivní formulaci vhodnou pro dopřednou propagaci zprávy:

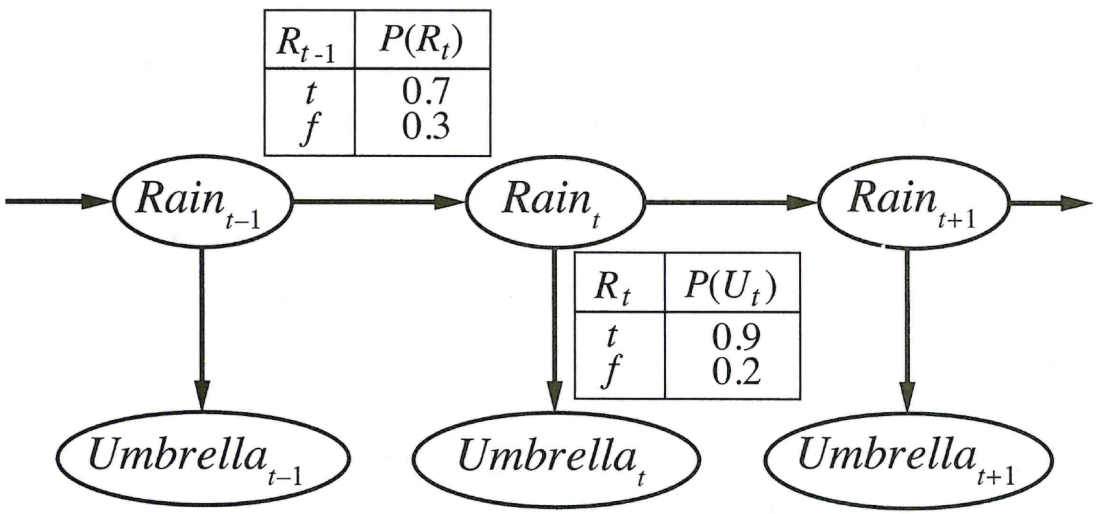
$$\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t}) = \mathbf{f}_{1:t}$$
$$\mathbf{f}_{1:t+1} = \alpha\, \mathrm{forward}(\mathbf{f}_{1:t}, e_{t+1})$$
$$\mathbf{f}_{1:0} = \mathbf{P}(\mathbf{X}_0)$$

Filtrace: příklad

$\mathbf{P}(\mathbf{R}_{t+1}|\mathbf{u}_{1:t+1}) =$ *Umbrella*$_1$=true, *Umbrella*$_2$=true

$$\alpha\,\mathbf{P}(\mathbf{u}_{t+1}|\mathbf{R}_{t+1})\,\mathbf{P}(\mathbf{R}_{t+1}|\mathbf{u}_{1:t}) = \alpha\,\mathbf{P}(\mathbf{u}_{t+1}|\mathbf{R}_{t+1})\sum_{\mathbf{r}_t}\mathbf{P}(\mathbf{R}_{t+1}|\mathbf{r}_t)\,P(\mathbf{r}_t|\mathbf{u}_{1:t})$$





$\mathbf{P}(\mathbf{R}_0) = \langle 0.5, 0.5\rangle$

$\mathbf{P}(\mathbf{R}_1)$

$= \sum_{\mathbf{r}_0}\mathbf{P}(\mathbf{R}_1|\mathbf{r}_0)P(\mathbf{r}_0)$

$= \langle 0.7,0.3\rangle\times 0.5 + \langle 0.3,0.7\rangle\times 0.5$

$= \langle 0.5, 0.5\rangle$

$\mathbf{P}(\mathbf{R}_1|\mathbf{u}_1)$

$= \alpha\mathbf{P}(\mathbf{u}_1|\mathbf{R}_1)\,\mathbf{P}(\mathbf{R}_1)$

$= \alpha\,\langle 0.9, 0.2\rangle\langle 0.5, 0.5\rangle$

$\approx \langle 0.818, 0.182\rangle$

$\mathbf{P}(\mathbf{R}_2|\mathbf{u}_1)$

$= \sum_{\mathbf{r}_1}\mathbf{P}(\mathbf{R}_2|\mathbf{r}_1)P(\mathbf{r}_1|\mathbf{u}_1)$

$= \langle 0.7, 0.3\rangle \times 0.818$

$+ \langle 0.3, 0.7\rangle \times 0.182$

$\approx \langle 0.627, 0.372\rangle$

$\mathbf{P}(\mathbf{R}_2|\mathbf{u}_1, \mathbf{u}_2) = \mathbf{P}(\mathbf{R}_2|\mathbf{u}_{1:2})$

$= \alpha\mathbf{P}(\mathbf{u}_2|\mathbf{R}_2)\,\mathbf{P}(\mathbf{R}_2|\mathbf{u}_1)$

$= \alpha\,\langle 0.9, 0.2\rangle\langle 0.627, 0.372\rangle$

$\approx \langle 0.883, 0.117\rangle$

Predikce

- Úkolem je zjistit pravděpodobnost budoucího stavu na základě dosavadních pozorování:

$$\mathbf{P}(\mathbf{X}_{t+k}|\mathbf{e}_{1:t}) \text{ pro } k > 0$$

- Jedná se v podstatě o filtraci bez přidávání dalších pozorování

$$\mathbf{P}(\mathbf{X}_{t+k+1}|\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_{t+k}} \mathbf{P}(\mathbf{X}_{t+k+1}|\mathbf{x}_{t+k})\, P(\mathbf{x}_{t+k}|\mathbf{e}_{1:t})$$

(rozšíření jednokrokové predikce z odvození filtrace)

- Po určité době (mixing time) konverguje předpovězená distribuce ke stacionární distribuci a nadále zůstane stejná
  - příklad: pravděpodobnost deště konverguje k $\langle 0.5, 0.5 \rangle$

Vyhlazování

- Úkolem je zjistit pravděpodobnost minulého stavu na základě dosavadních pozorování: $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$, kde $0 \leq k < t$
- Opět použijeme rekurzivní předávání zpráv, tentokrát ve dvou směrech

$$\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t}) = \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) =$$
$$= \alpha\, \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\, \mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k, \mathbf{e}_{1:k}) \quad \text{z Bayesovy věty}$$
$$= \alpha\, \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\, \mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) \quad \text{z podmíněné nezávislosti}$$
$$= \alpha \mathbf{f}_{1:k} \times \mathbf{b}_{k+1:t}$$

(tj. spočítáme rekurzí přes čas: dopředně od 1 do k, zpětně od t ke $k+1$)

$$\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) = \sum_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k, \mathbf{x}_{k+1})\, \mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k)$$

z marginalizace $\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y}, \mathbf{z}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y}|\mathbf{z})\, P(\mathbf{z})$ a podmíněné pst.

kde $\mathbf{Y} = \mathbf{e}_{k+1:t}$ a $\mathbf{z} = \mathbf{x}_{k+1}$

$$= \sum_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{x}_{k+1})\, \mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \quad \text{z podmíněné nezávislosti}$$
$$= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}, \mathbf{e}_{k+2:t}|\mathbf{x}_{k+1})\, \mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k)$$
$$= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}|\mathbf{x}_{k+1}) P(\mathbf{e}_{k+2:t}|\mathbf{x}_{k+1})\, \mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \quad \text{z podmíněné nezávislosti}$$

Vyhlazování (dokončení)

Máme tedy:

$$\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) = \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}|\mathbf{x}_{k+1}) P(\mathbf{e}_{k+2:t}|\mathbf{x}_{k+1})\, \mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k)$$

- $P(\mathbf{e}_{k+1}|\mathbf{x}_{k+1})$: známe ze senzorického modelu
- $P(\mathbf{e}_{k+2:t}|\mathbf{x}_{k+1})$: zpětný chod
- $\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k)$: známe z přechodového modelu

Můžeme tedy použít techniku zpětné propagace zprávy:

$$\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) = \mathbf{b}_{k+1:t}$$
$$\mathbf{b}_{k+1:t} = \text{backward}(\mathbf{b}_{k+2:t}, e_{k+1})$$
$$\mathbf{b}_{t+1:t} = \mathbf{P}(\mathbf{e}_{t+1:t}|\mathbf{X}_t) = \mathbf{P}(\ |\mathbf{X}_t)\mathbf{1} = \mathbf{1}$$

vzhledem k tomu, že $\mathbf{e}_{t+1:t}$ je prázdná posloupnost

Vyhlazování: příklad

$\mathbf{P}(\mathbf{R}_k|\mathbf{u}_{1:t+1}) = \alpha\,\mathbf{P}(\mathbf{R}_k|\mathbf{u}_{1:k})\,\mathbf{P}(\mathbf{u}_{k+1:t}|\mathbf{R}_k)$, např. $\mathbf{P}(R_1|u_{1:2}) = \alpha\mathbf{P}(R_1|u_1)\times\mathbf{P}(u_2|R_1)$

$\mathbf{P}(\mathbf{u}_{k+1:t}|\mathbf{R}_k) = \sum_{\mathbf{r}_{k+1}} \mathbf{P}(u_{k+1}|\mathbf{r}_{k+1})\mathbf{P}(\mathbf{u}_{k+2:t}|\mathbf{r}_{k+1})\,\mathbf{P}(\mathbf{r}_{k+1}|\mathbf{R}_k)$

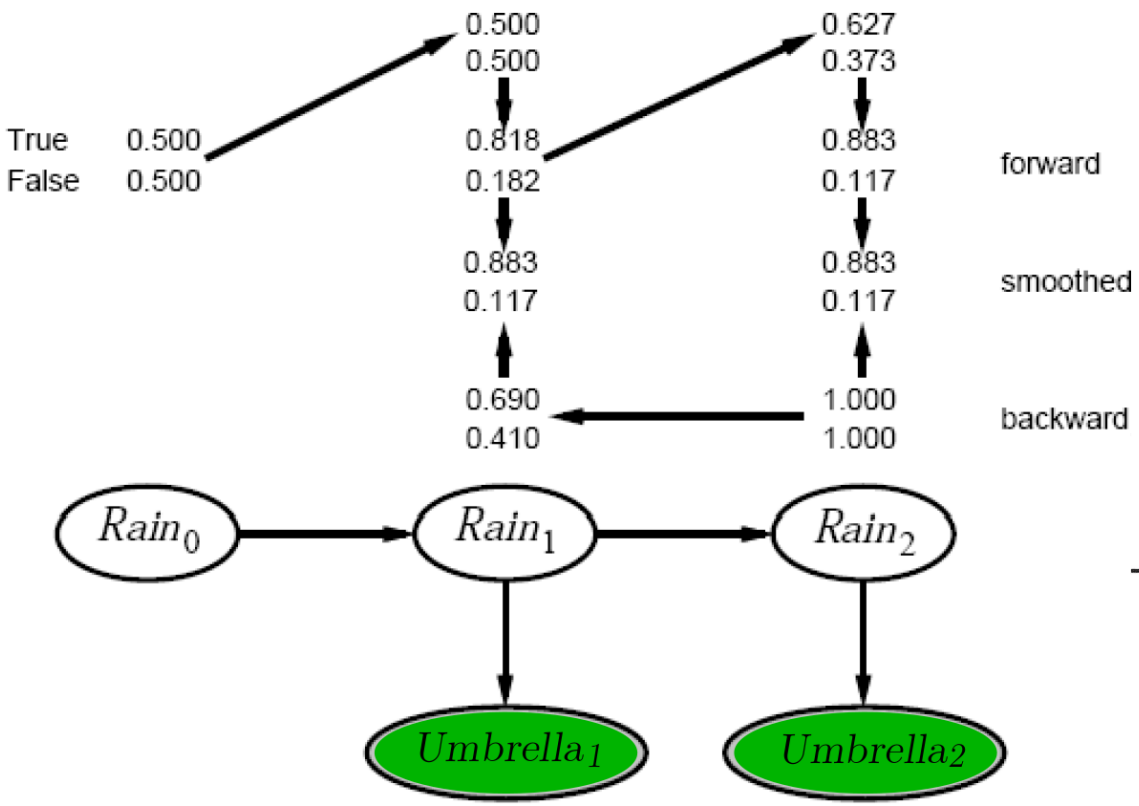
$\mathbf{P}(\ |\mathbf{R}_2) = \mathbf{1}$ *Umbrella*$_1$=true, *Umbrella*$_2$=true

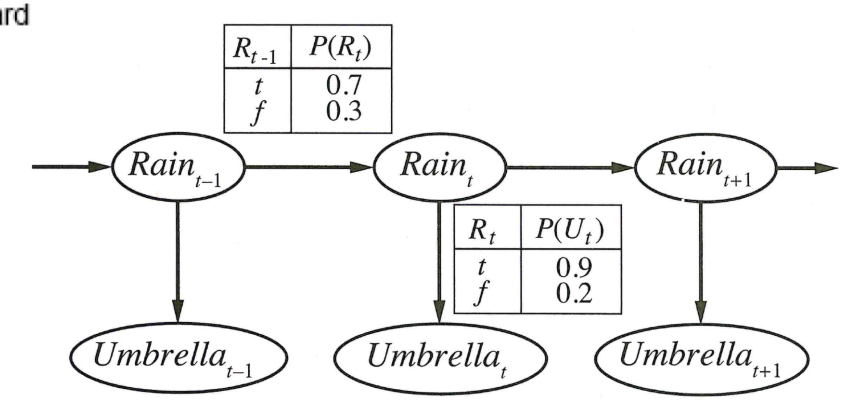
$\mathbf{P}(\mathbf{u}_2|\mathbf{R}_1) = \sum_{r_2} \mathbf{P}(\mathbf{u}_2|\mathbf{r}_2)\,\mathbf{P}(\ |\mathbf{r}_2)\,\mathbf{P}(\mathbf{r}_2|\mathbf{R}_1) =$

$= 0.9 \times 1 \times \langle 0.7, 0.3\rangle + 0.2 \times 1 \times \langle 0.3, 0.7\rangle = \langle 0.69, 0.41\rangle$

$\mathbf{P}(\mathbf{R}_1|\mathbf{u}_1) \times \mathbf{P}(\mathbf{u}_2|\mathbf{R}_1) = \langle 0.818, 0.182\rangle \times \langle 0.69, 0.41\rangle \approx \langle 0.883, 0.117\rangle = \mathbf{P}(\mathbf{R}_1|\mathbf{u}_1, \mathbf{u}_2)$

tj. vyhlazený odhad (0.883)
je vyšší než filtrovaný odhad
(0.818 z $\mathbf{P}(\mathbf{R}_1|\mathbf{u}_1)$)
vzhledem k dešti v den 2

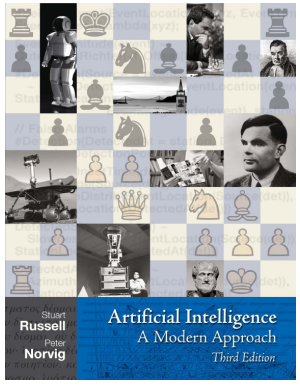




Užitek a rozhodování

8. ledna 2021

Zdroje: *Roman Barták, přednáška přednáška Umělá inteligence II, Matematicko-fyzikální fakulta, Karlova univerzita v Praze, 2015.*
http://kti.ms.mff.cuni.cz/~bartak/ui2



Úvod

- Cílem je tvorba racionálních agentů maximalizujících očekávanou míru užitku
- Teorie pravděpodobnosti říká, čemu máme věřit na základě pozorování
- Teorie užitku (utility theory) popisuje, co chceme a jak máme ohodnotit rozhodnutí
- Teorie rozhodování (decision theory) spojuje obě teorie dohromady a popisuje, jak bychom měli vybrat akci s největším očekávaným užitkem
  - podíváme se na statický případ (jedno rozhodnutí)
  - i na posloupnost rozhodnutí

Teorie užitku

- (Agentovy) preference lze zaznamenat funkcí užitku $U(s)$, která mapuje stavy s na reálné číslo
- Očekávaný užitek (expected utility) potom spočteme jako průměrný užitek přes všechny možné stavy vážené pravděpodobností výsledků

$$EU(a|\mathbf{e}) = \sum_s P(Result(a) = s|a, \mathbf{e})U(s)$$

  - $\mathbf{e}$ pozorování, a akce
  - $Result(a)$ náhodná proměnná, jejíž hodnoty jsou možné výsledné stavy
- Racionální agent potom volí akci maximalizující očekávaný užitek *MEU*

$$akce = \text{argmax}_a EU(a|\mathbf{e})$$

- *MEU* formalizuje racionalitu, ale jak budeme celý postup operačně realizovat? Tj. budeme se zabývat:
  - jak volit akci při jednom rozhodnutí?
  - jak volit postupně akce při posloupnosti rozhodnutí?

Užitek a preference

- Cíl: hledáme funkci užitku popisující preference
- Agentovy preference se často vyjadřují relativním porovnáním
  - $A > B$: agent preferuje A před B
  - $A < B$: agent preferuje B před A
  - $A \sim B$: agent mezi A a B nemá žádnou preferenci (nerozlišuje A a B)
- Co je A nebo B?
  - mohou to být stavy světa, ale
    pro neurčité výstupy se používají loterie
  - loterie popisuje možné výstupy $S_1, \ldots, S_n$,
    které se vyskytují s danými pravděpodobnostmi $p_1, \ldots, p_n$
    - $[p_1, S_1; \ldots, p_n, S_n]$
- Příklad loterie (nabídka jídla v letadle): Chcete kuře nebo těstoviny?
  - [0.8, dobré kuře; 0.2 připečené kuře]
  - [0.7, dobré těstoviny; 0.3, rozvařené těstoviny]

Funkce užitku

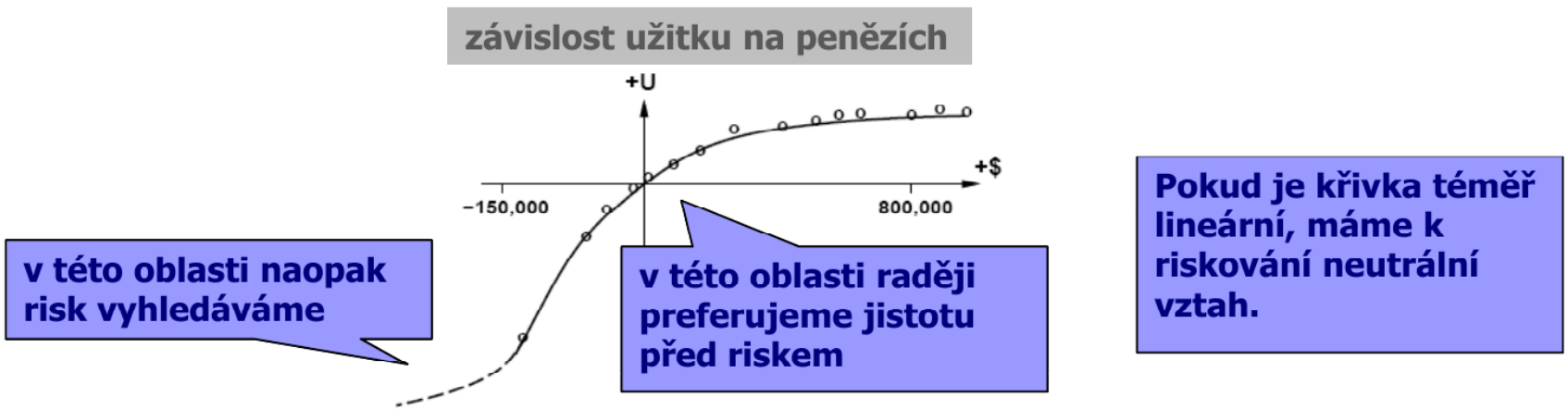
- Existuje funkce užitku vracející pro danou loterii reálné číslo tak, že
  - $U(A) < U(B) \Leftrightarrow A < B$
  - $U(A) = U(B) \Leftrightarrow A \sim B$
- Očekáváný užitek loterie lze spočítat:
  - $U([p_1, S_1; \ldots, p_n, S_n]) = \sum_i p_i U(S_i)$
- Racionální agent ani nemusí svoji funkci užitku znát, ale pozorováním jeho preferencí ji lze zrekonstruovat
- Jak zjistit funkci užitku konkrétního agenta (preference elicitation)?
  - budeme hledat normalizovanou funkci užitku
  - uvažujme nejlepší možný stav S_{max} a dejme mu užitek 1: $U(S_{\text{max}}) = 1$
  - podobně pro nejhorší možný stav S_{min} a dejme užitek 0: $U(S_{\text{min}}) = 0$
  - nyní se pro libovolný stav S ptejme agenta na porovnání S a standardní loterie $[p, S_{\text{max}}; 1 - p, S_{\text{min}}]$
  - podle výsledku upravíme pravděpodobnost p a ptáme se znova, dokud agent vztah A a standardní loterie nepovažuje za nerozlišitelný
  - získané p je užitkem S: $U(S) = p$

Peníze jako užitek?

- V běžném životě používáme peníze pro ohodnocení různého zboží a služeb
  - agent zpravidla preferuje více peněz před méně penězi, je-li vše ostatní stejné
- Proč nejsou peníze přímo mírou užitku?
- Uvažujme, že jsme vyhráli 1 mil USD a můžeme si ho buď nechat nebo přimeme sázku na hod mincí – padne-li orel, dostaneme 2,5 mil. USD, jinak nic. Co zvolíte?
  - očekávaný peněžní zisk při sázce je 1,25 mil USD
  - většina lidí ale volí jistotu 1 mil. USD, je to snad iracionální?

Užitek z peněz

- Volba v předchozí hře závisí nejen na hře samé, ale i na současném majetku hráče!
- Nechť S_n je stav označující majetek n USD
- Potom můžeme očekávaný užitek (EU) akcí popsat takto
  - $\mathrm{EU}(\mathrm{Accept}) = \frac{1}{2}\,\mathrm{U}(S_k) + \frac{1}{2}\,\mathrm{U}(S_{k+2\,500\,000})$
  - $\mathrm{EU}(\mathrm{Decline}) = \mathrm{U}(S_{k+1\,000\,000})$
- Nechť $\mathrm{U}(S_k)=5$, $\mathrm{U}(S_{k+1\,000\,000})=8$, $\mathrm{U}(S_{k+2\,500\,000})=9$
  - tj. $\mathrm{EU}(\mathrm{Accept}) = 7$, $\mathrm{EU}(\mathrm{Decline}) = 8$
- Potom je rozhodnutí odmítnout sázku zcela racionální!



Více atributů

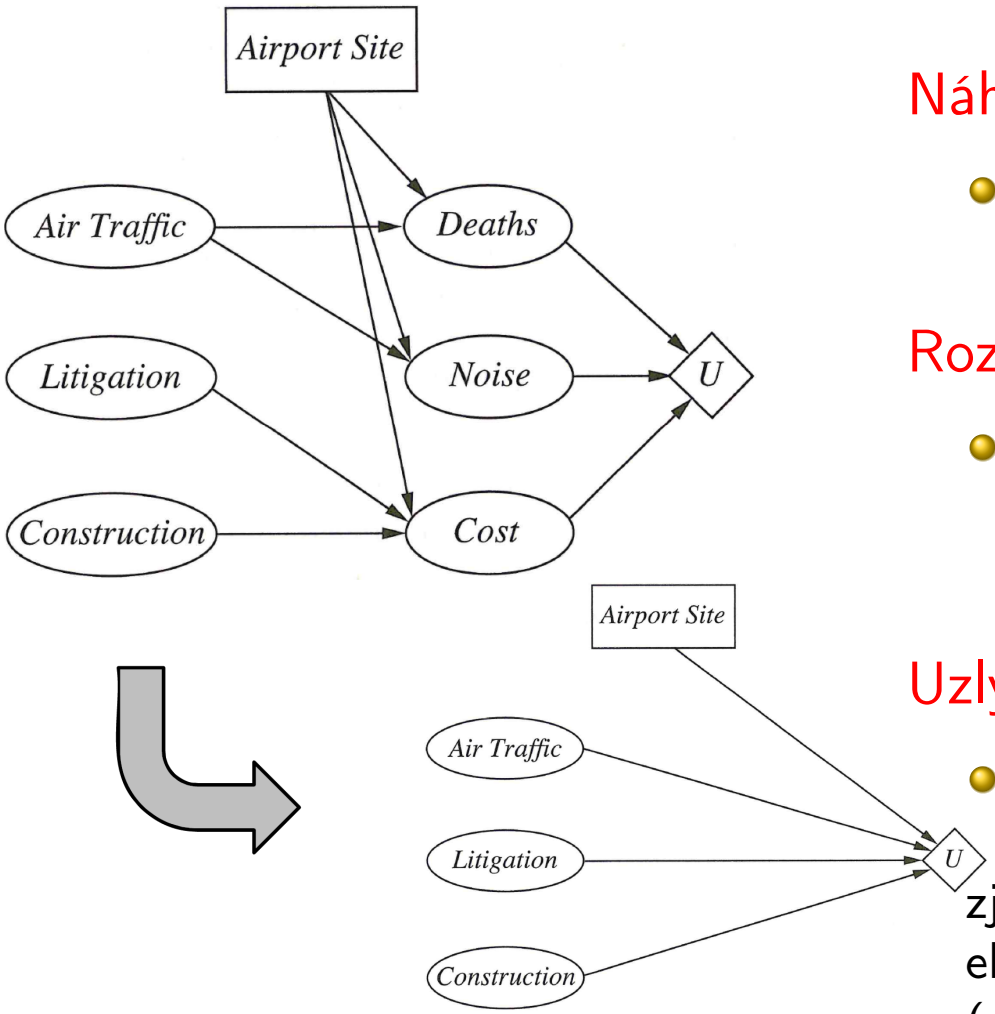
- V praxi se často vyskytuje více atributů užitku, např. cena, nebezpečnost, užitečnost – víceatributová funkce užitku
- Budeme uvažovat, že každý atribut má definované preferované pořadí hodnot, vyšší hodnoty odpovídají lepšímu řešení
- Jak definovat preference pro více atributů dohromady?
  - přímo bez kombinace hodnot atributů – dominance
    - pokud je ve všech atributech řešení A horší než řešení B, potom B je lepší i celkově – striktní dominance
    - lze použít i pro nejisté hodnoty atributů
      – stačí, když každá možná hodnota všech atributů A je horší, než každá možná hodnota odpovídajících atributů B
    - striktní dominance není moc obvyklá, ale může alespoň odfiltrovat špatná řešení
  - kombinací hodnot atributů do jedné hodnoty
    - pokud jsou atributy (preferenčně) nezávislé, lze použít aditivní oceňovací funkci

      $U(x_1, \ldots, x_n) = \sum_i U_i(x_i)$

      může se jednat např. o vážený součet atributů

Rozhodovací sítě

- Jak obecně popsat mechanismus rozhodování?
- Rozhodovací síť (influenční diagram) popisuje vztahy mezi vstupy (současný stav), rozhodnutími (budoucí stav) a užitkem (budoucího stavu)



Náhodné uzly (ovál)

- reprezentují náhodné proměnné stejně jako v Bayesovských sítích

Rozhodovací uzly (obdélníky)

- popisují rozhodnutí (výběr akce), které může agent udělat (zatím uvažujeme jednoho agenta)

Uzly užitku (kosočtverce)

- popisují funkci užitku

zjednodušená rozhodovací síť eliminující budoucí stavy (nelze pozorovat hodnoty)

Rozhodovací sítě: vyhodnocovací algoritmus

Akce jsou vybrány na základě vyzkoušení všech možných hodnot rozhodovacího uzlu

---

1. nastavíme hodnoty pozorovaných proměnných
2. pro každou možnou hodnotu rozhodovacího uzlu
   2.1 nastavíme rozhodovací uzel na danou hodnotu
   2.2 použitím pravděpodobnostní inference vypočteme pravděpodobnosti rodičů uzlu užitku
   2.3 vypočteme užitek pomocí funkce užitku
3. vybereme akci s největším užitkem

Pro případy, kde je více uzlů užitku používáme při výběru akce techniky pro víceatributové funkce užitku

Přímé rozšíření algoritmu pro Bayesovské sítě

- při rozhodování agenta vybíráme akci s největším užitkem

Zajímavější problém až při vykonávání posloupnosti rozhodnutí

Rozhodovací sítě: konstrukce a příklad

Vytvoření kauzálního modelu

- určení symptomů, nemocí, léčby a vztahů mezi nimi (dom.experti, literatura)

Zjednodušení kvalitativního modelu

- pokud nám jde o léčbu, můžeme odstranit uzly, které s nimi nesouvisí
- některé uzly lze spojit (léčba *Treatment* a její načasování *Timing*)

Přiřazení pravděpodobností – z kauzálních vazeb, diagnostické vazby z BS:

- vyplnění pravděpodobnostních tabulek iako v Bavesovské síti (dle učebnic. databází pacientů,rozhodnutí experta)

Navržení funkce užitku

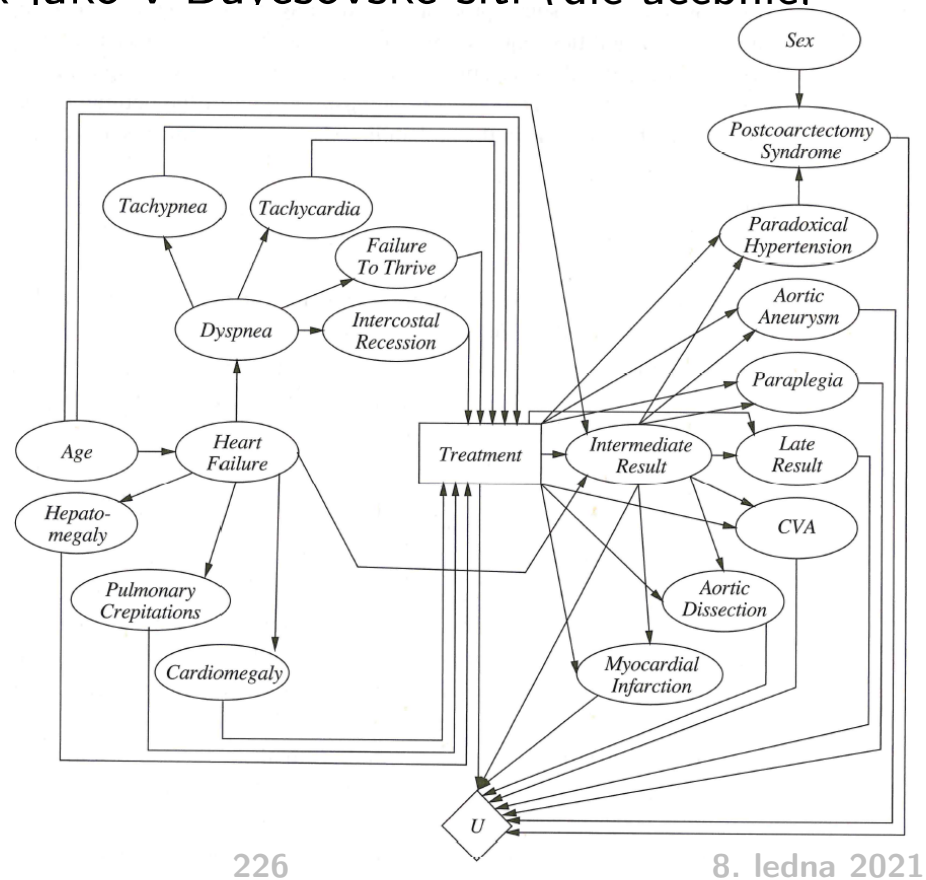
- můžeme enumerovat možné výstupy (záleží nejen na lékaři ale i na pacientovi, který může mít jiné preference)

Verifikace a doladění modelu

- výsledky se porovnají se zlatým standardem, např. skupinou lékařů

Analýza citlivosti

- kontrola, zda výsledky systému nejsou citlivé na drobné změny vstupu (malé změny vstupu vedoucí k velkým rozdílům výstupu indikují problém v modelu)



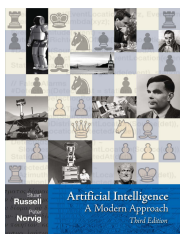
Shrnutí

- Teorie pravděpodobnosti nám umožní kvantifikovat, jak máme danému tvrzení věřit na základě pozorování
  - Bayesovské sítě a odvozování pomocí nich
    - staticky v daném čase
  - Pravděpodobnostní uvažování o čase
    - reprezentace přechodů s pravděpodobností: Markovský proces
    - zjištění pravděpodobnosti stavu vzhledem k probíhajícímu času
- Teorie užitku pro ohodnocení rozhodnutí
- Teorie rozhodování
  - pro jedno rozhodnutí (statický případ)
  - příště: pro posloupnost rozhodnutí

Sekvenční rozhodovací problémy

8. ledna 2021

Zdroje: *Roman Barták, přednáška přednáška Umělá inteligence II, Matematicko-fyzikální fakulta, Karlova univerzita v Praze, 2015.* `http://kti.ms.mff.cuni.cz/~bartak/ui2`



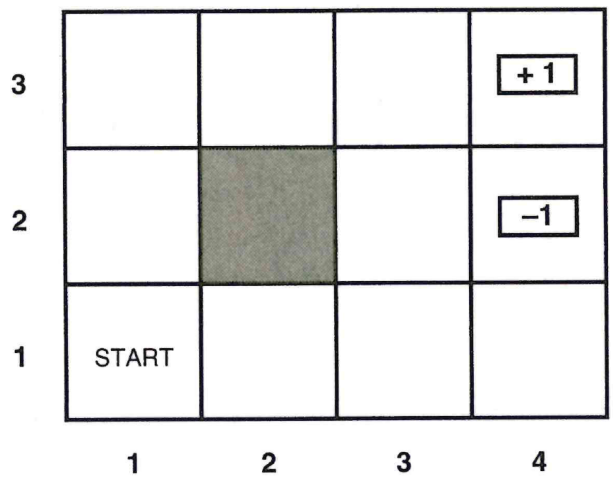
Úvod

- Cílem je tvorba racionálních agentů maximalizujících očekávanou míru užitku
- Teorie pravděpodobnosti říká, čemu máme věřit na základě pozorování
- Teorie užitku (utility theory) popisuje, co chceme a jak máme ohodnotit rozhodnutí
- Teorie rozhodování (decision theory) spojuje obě teorie dohromady a popisuje, jak bychom měli vybrat akci s největším očekávaným užitkem
  - minule: statický případ (jedno rozhodnutí)
  - dnes: i na posloupnost rozhodnutí

Sekvenční rozhodovací problémy

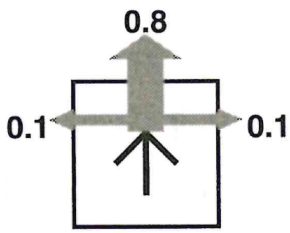
Doposud jedno rozhodnutí → nyní posloupnost rozhodnutí

- užitek bude záviset na posloupnosti rozhodnutí



- Uvažujme agenta pohybujícího se v prostředí o rozměrech 3×4
- Prostředí je plně pozorovatelné (agent vždy ví, kde se nachází)
- Agent se chce dostat do stavu +1 a vyhnout se stavu -1

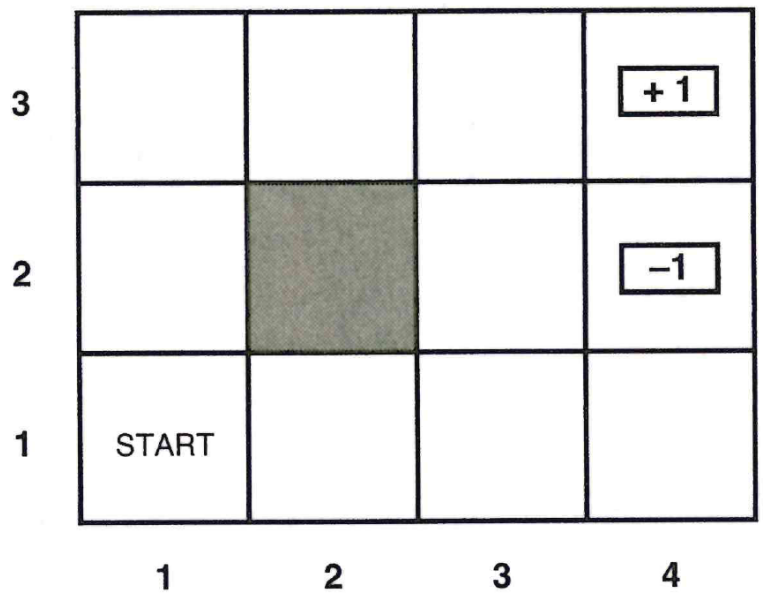
- Agent může provést akce Up, Down, Left, Right
  - množinu akcí pro stav s značíme $A(s)$
  - výsledek akce je ale nejistý
    - s pravděpodobností 0.8 půjde správným směrem
    - s pravděpodobností 0.1 půjde kolmo k požadovanému směru (resp. stojí na místě, pokud narazí na zeď)



  - pravděpodobnost [Up, Up, Right, Right, Right]: 0.8^5
    celkem ze START do cíle (stav +1):
    $0.8^5 + 0.1^4 * 0.8 = 0.32776$ (+ cesta dokola [Right, Right, Up, Up, Right])

Příklad: pravděpodobnost dosažení stavu

Na která pole a s jakou pravděpodobností můžete dojít z pole (1,1) posloupností [Up, Up, Right, Right, Right]?



Nápověda: nakreslete si strom se stavy dosaženými po každém kroku s pravděpodobnostmi přechodu do daného stavu.

Dále: Pravděpodobnost dosažení listu je součin pravděpodobností po cestě (pravděpodobnosti jsou Markovské). Pokud se stejný stav vyskytuje ve více listech, pravděpodobnosti sečteme (cesty jsou disjunktní).

Markovský rozhodovací proces (MDP)

- Pro popis přechodů (aplikace akce) použijeme pravděpodobnostní distribuci $P(s'|s,a)$ – pravděpodobnost přechodu z s do s' akcí a
  - opět uvažujeme Markovský předpoklad (pravděpodobnost přechodu nezávisí na předchozích navštívených stavech)
- Užitek tentokrát závisí na prošlých stavech
  - každý stav má přiřazeno ocenění (reward) $R(s)$
    - např. +1 (cíl), -1 (nežádoucí stav), -0.04 (ostatní stavy)
  - funkce užitku bude (zatím) součet ocenění navštívených stavů
    - funkce užitku je zde podobná jako při hledání nejkratší cesty do cíle (plus máme stochastický charakter akcí)
- Markovský rozhodovací proces (Markov Decision Process MDP)
  - sekvenční rozhodovací problém v plně pozorovatelném stochastickém prostředí s Markovským přechodovým modelem a aditivní funkcí užitku
    - tvořen množinou stavů s počátečním stavem s_0
    - množinou akcí v každém stavu $A(s)$
    - přechodovým modelem $P(s'|s,a)$
    - oceněním $R(s)$

Řešení MDP

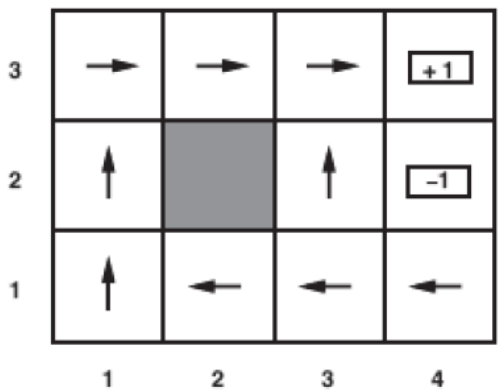
- Ve stochastickém prostředí nemůžeme pracovat s pevným pořadím akcí (plánem)
  - agent se může vydat jinam, než bylo naplánováno
- Protože předem nevíme, kam akce agenta zavede potřebujeme v každém stavu vědět, co dělat (kam jít)
- Řešením MDP je strategie (policy) $\pi(S)$, což je funkce určující pro každý stav doporučenou akci
  - hledáme strategii s největším očekávaným užitkem = optimální strategie

Optimální strategie

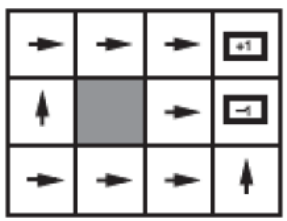
Optimální strategie maximalizuje očekávaný užitek

- záleží samozřejmě na ocenění stavů

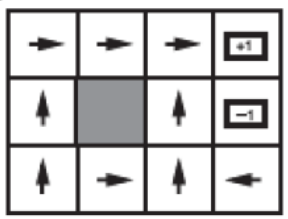
Ocenění stavů je -0.04
ocenění je malé, agent se snaží dostat k východu, ale bezpečně



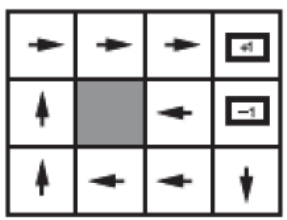
Ocenění stavů je záporné
agent se snaží co nejdříve dostat ke každému východu, i za cenu rizika



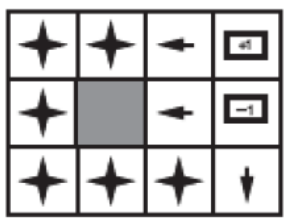
$R(s) < -1.6284$



$-0.4278 < R(s) < -0.0850$



$-0.0221 < R(s) < 0$



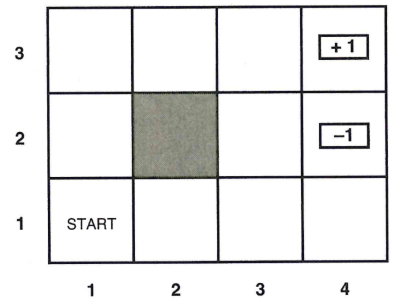
$R(s) > 0$

Ocenění stavů je blízké nule
agent se snaží dostat k východu, ale minimalizuje každé riziko

Ocenění stavů je kladné
agent se snaží zůstat v kladných stavech (užívá se život)

Užitek v čase: horizont

Jak obecně definovat funkci užitku pro posloupnost stavů?



- je to podobné jako pro víceatributové funkce užitku $U([s_0, s_1, \ldots, s_n])$ (stavy jsou atributy), ale
- jaký máme horizont?

Konečný horizont

- máme daný pevný termín N a po něm už na ničem nezáleží
  $U([s_0, s_1, \ldots, s_{N+k}]) = U([s_0, s_1, \ldots, s_N])$
- optimální strategie záleží na termínu (není stacionární)
  - pro $N = 3$ volí ze stavu (3,1) akci Up
  - pro $N = 100$ volí ze stavu (3,1) bezpečnou akci Left

Nekonečný horizont

- to neznamená nutně nekonečné posloupnosti stavů, pouze zde není žádný deadline
- není důvod se ve stejném stavu chovat v různé časy různě
- optimální strategie je stacionární

Užitek v čase: definice funkce

- Funkce užitku se chová jako víceatributová funkce užitku
  $U([s_0, s_1, \ldots, s_n])$
- Pro získání jednoduchého vztahu pro výpočet funkce užitku uvažujme stacionární preference
  - preference posloupnosti stavů $[s_0, s_1, s_2, \ldots]$ a $[s_0, s'_1, s'_2, \ldots]$ stejné jako preference $[s_1, s_2, \ldots]$ k $[s'_1, s'_2, \ldots]$
  - naše preference „zítra a dnes" jsou stejné
- V případě stacionárních preferencí jsou dva způsoby, jak rozumně definovat funkci užitku
  - aditivní funkce užitku
    $U([s_0, s_1, \ldots, s_n]) = R(s_0) + R(s_1) + R(s_2) + \cdots$
  - kumulovaná (discounted) funkce užitku
    $U([s_0, s_1, \ldots, s_n]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$
    faktor slevy $\gamma \in (0, 1)$
    - preference mezi aktuálním a budoucím oceněním
      – γ blízko 0: budoucnost není důležitá,
      – γ blízko 1: budoucnost je stejně důležitá jako současnost,
      tj. obdržíme aditivní funkci užitku

Užitek v čase: vlastnosti

Budeme používat kumulovanou funkci užitku

- pro světy bez cílového stavu a nekonečný horizont by aditivní funkce užitku byla problémová
  - pro nekonečné posloupnosti stavů dává $+\infty$ nebo $-\infty$
- užitek v kumulované funkci užitku je konečný (uvažujeme omezené ocenění s maximem $R_{\max}$)
  $U([s_0, s_1, \ldots, s_n]) = \sum_{t=0,\ldots+\infty} \gamma^t R(s_t) \leq \sum_{t=0,\ldots+\infty} \gamma^t R_{\max} = R_{\max}/(1-\gamma)$
  - plyne ze součtu nekonečné geometrické posloupnosti
- pokud má prostředí cílový stav, do kterého se agent garantovaně může dostat, nebudeme potřebovat pracovat s nekonečnými posloupnostmi
  - řádná (proper) strategie – garantuje dosažení cílového stavu
    - můžeme používat aditivní funkci užitku
    - strategie, které nejsou řádné (improper), mají nekonečný celkový užitek
- u nekonečné posloupnosti lze porovnat průměrné ocenění
  - lepší je zůstat ve stavu s oceněním 0.1 než ve stavu s oceněním 0.01
  - analýza tohoto chování však náročná

Optimální strategie: vlastnosti I.

- Hledáme očekávaný užitek (střední hodnotu E) strategie π

  pro počáteční stav s :

$$U^{\pi}(s) = E\left[\sum_{t=0,\ldots,+\infty} \gamma^t R(S_t)\right]$$

  - S_t je náhodná proměnná popisující stav v čase t
  - připomenutí: používáme kumulovanou funkci užitku

- Optimální strategie pro počáteční stav s:

$$\pi_s^{\star} = \text{argmax}_{\pi} U^{\pi}(s)$$

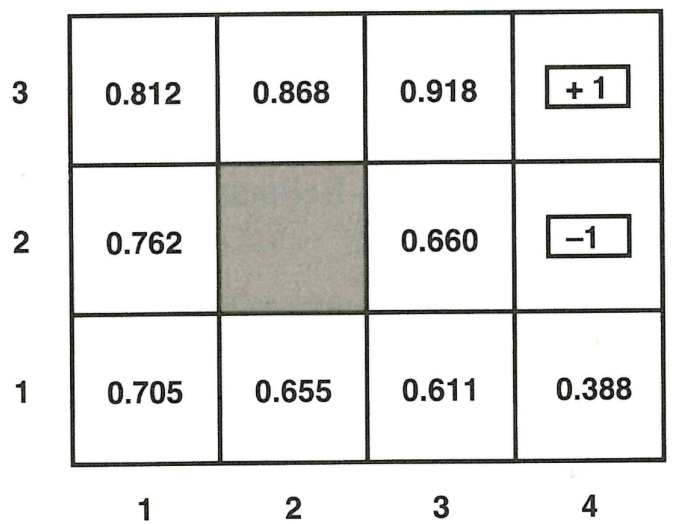
  - tj. strategie, která maximalizuje očekávaný užitek pro poč.stav s

- Záleží ale na počátečním stavu?
  - když se strategie $\pi_a^{\star}$ a $\pi_b^{\star}$ sejdou v nějakém stavu c, není důvod, aby pokračovaly různě, tj. můžeme používat pouze $\pi^{\star}$

Optimální strategie: vlastnosti II.

- Můžeme definovat užitek stavu $U(s) = U^{\pi^\star}(s)$
  - připomeňme, že $U^{\pi^\star}(s)$ je očekávaný užitek z optimální strategie
  - $R(s)$ je „krátkodobé" ocenění vs.

    $U(s)$ je „dlouhodobé" ocenění



Opakování: očekávaný užitek

- Očekávaný užitek (expected utility) spočteme jako průměrný užitek přes všechny možné stavy vážené pravděpodobností výsledků

$$EU(a|\mathbf{e}) = \sum_s P(Result(a) = s|a, \mathbf{e})U(s)$$

  - $\mathbf{e}$ pozorování, a akce
  - $Result(a)$ náhodná proměnná, jejíž hodnoty jsou možné výsledné stavy

- Racionální agent potom volí akci maximalizující očekávaný užitek *MEU*

$$akce = \text{argmax}_a EU(a|\mathbf{e})$$

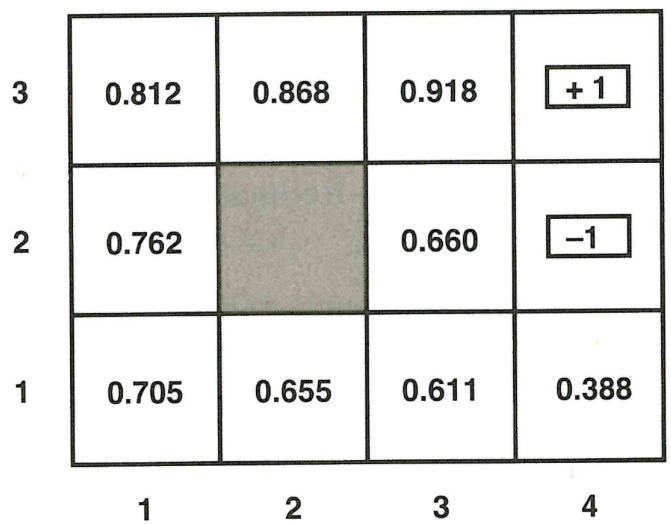
Optimální strategie: vlastnosti II. (pokračování)

- Můžeme definovat užitek stavu $U(s) = U^{\pi^\star}(s)$
  - připomeňme, že $U^{\pi^\star}(s)$ je očekávaný užitek z optimální strategie
  - $R(s)$ je „krátkodobé" ocenění vs.

    $U(s)$ je „dlouhodobé" ocenění
  - akce budeme vybírat na základě principu maximalizace očekávaného užitku *v následujícím stavu*

    $$\pi^\star(s) = \text{argmax}_{a \in A(s)} \sum_{s'} P(s'|s,a) U(s')$$

    připomeňme, že $P(s'|s,a)$ je pravděpodobnost přechodu z s do s' akcí a
  - pozor, to nemusí být akce vedoucí do stavu s největším $U(s)$!



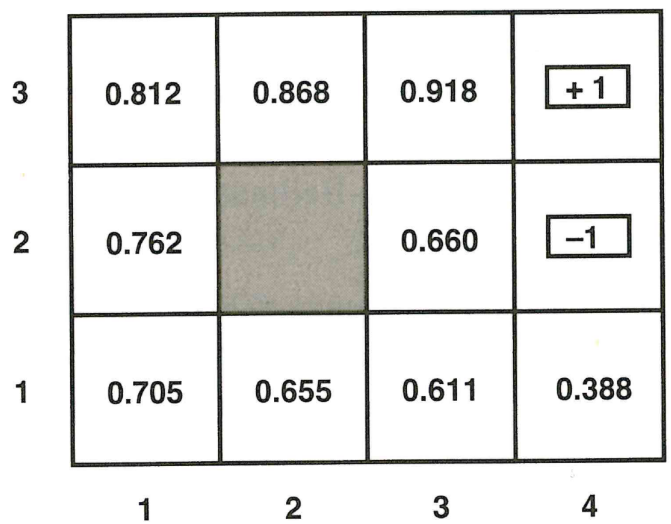
Bellmanova rovnice

- Navrhnout algoritmus pro řešení MDP?
  - spočítáme užitek každého stavu
  - použijeme ho pro výběr optimální akce v každém stavu
- Užitek stavu přímo závisí na užitku okolních stavů (pokud agent volí optimální akci)

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)\, U(s')$$

- Tomuto vztahu se říká Bellmanova rovnice

$$\begin{aligned} U((1,1)) = {} & -0.04 + \\ & \gamma \max[\\ & \quad 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), && (Up) \\ & \quad 0.9U(1,1) + 0.1U(1,2), && (Left) \\ & \quad 0.9U(1,1) + 0.1U(2,1), && (Down) \\ & \quad 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)] && (Right) \end{aligned}$$



Po doplnění čísel z obrázku dostaneme Up jako nejlepší akci

Iterace hodnot (užitku)

- Na základě Bellmanovy rovnice

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)\, U(s')$$

  můžeme navrhnout algoritmus pro řešení MDP
  - pro n stavů máme n rovnic s n proměnnými
  - proměnná pro stav s = užitek stavu s
- Problém je, že máme soustavu nelineárních rovnic
  - pozn. max není lineární operátor
- Použijeme iterativní postup
  1. začneme s libovolnými vstupními hodnotami $U(s)$ (např. nulovými)
  2. $U(s)$ upravíme na základě Bellmanových rovnic (Bellmanův update)

$$U_{i+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)\, U_i(s')$$

Iterace hodnot (užitku): algoritmus

- Tedy použijeme už popsaný iterativní postup
  1. začneme s libovolnými vstupními hodnotami $U(s)$ (např. nulovými)
  2. $U(s)$ upravíme na základě Bellmanových rovnic (Bellmanův update)

$$U_{i+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)\, U_i(s')$$

---

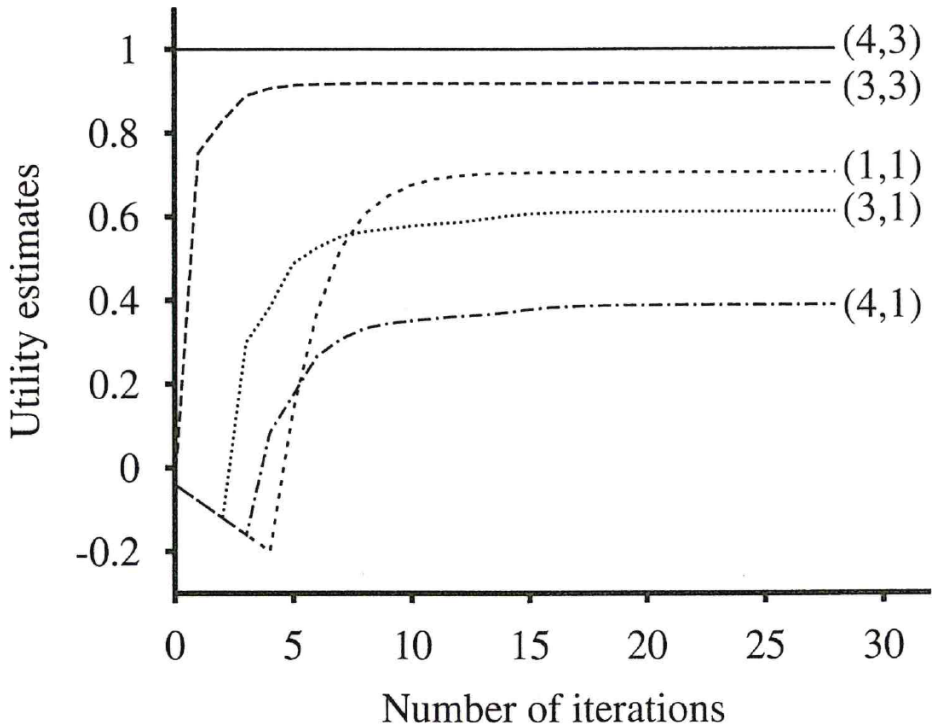
function iterace-hodnot(mdp, ϵ) **returns** funkci užitku
  (mdp: MDP a jeho stavy S, akce $A(s)$, přechodový model $P(s'|s,a)$
    ocenění $R(s)$, faktor slevy γ
  ϵ: maximální dovolená chyba v užitku libovolného stavu
  U, U': vektory užitku pro stavy v S, iniciálně 0
  δ: maximální změna užitku libovolného stavu v iteraci)
  repeat $U = U'$; $\delta = 0$;
    for each stav $s \in S$ **do**
      $U'[s] = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)\, U[s']$;
      if $|U'[s] - U[s]| > \delta$ **then** $\delta = |U'[s] - U[s]|$;
  until $\delta < \epsilon(1-\gamma)/\gamma$;
  return U

Iterace hodnot: konvergence a příklad

3	0.812	0.868	0.918	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Graf vývoje hodnot užitku po jednotlivých iteracích

- při ocenění $R(s)$: 1/-1/-0.04
- pro dané stavy (4,1), (3,1), (1,1), ...
- grafy (4,1), (3,1), (1,1) na začátku klesají, protože nevíme, že jdeme do cíle



Jak můžeme garantovat konvergenci iterace hodnot ke skutečné hodnotě užitku stavu?

Iterace hodnot: konvergence

- Bellmanův update splňuje vlastnost kontrakce
  ($|f(x) - f'(x)| < c|x - x'|$, kde $0 \leq c < 1$)
  - má jediný pevný bod
  - po každé iteraci se přiblížíme k pevnému bodu
- Vzdálenost měříme jako maximální vzdálenost odpovídajících prvků vektoru
  $|U - U'| := \max_s |U(s) - U'(s)|$
- Nechť BU je Bellmanův update vektoru U a U_i vektor užitku v iteraci i, potom
  $U_{i+1} \leftarrow BU_i$
  lze ukázat: $|BU_i - BU'_i| \leq \gamma|U_i - U'_i|$ (Bellm.update je kontrakcí o faktor γ)
  tj. $|BU_i - U| \leq \gamma|U_i - U|$, kde U je hledaný užitek stavu (pevný bod)
- Z kontrakce lze odvodit
  - počet kroků pro přiblížení se k U na vzdálenost ϵ
  - ukončovací podmínku pro dosažení vzdálenosti ϵ od U
    $|U_{i+1} - U_i| \leq \epsilon(1 - \gamma)/\gamma$

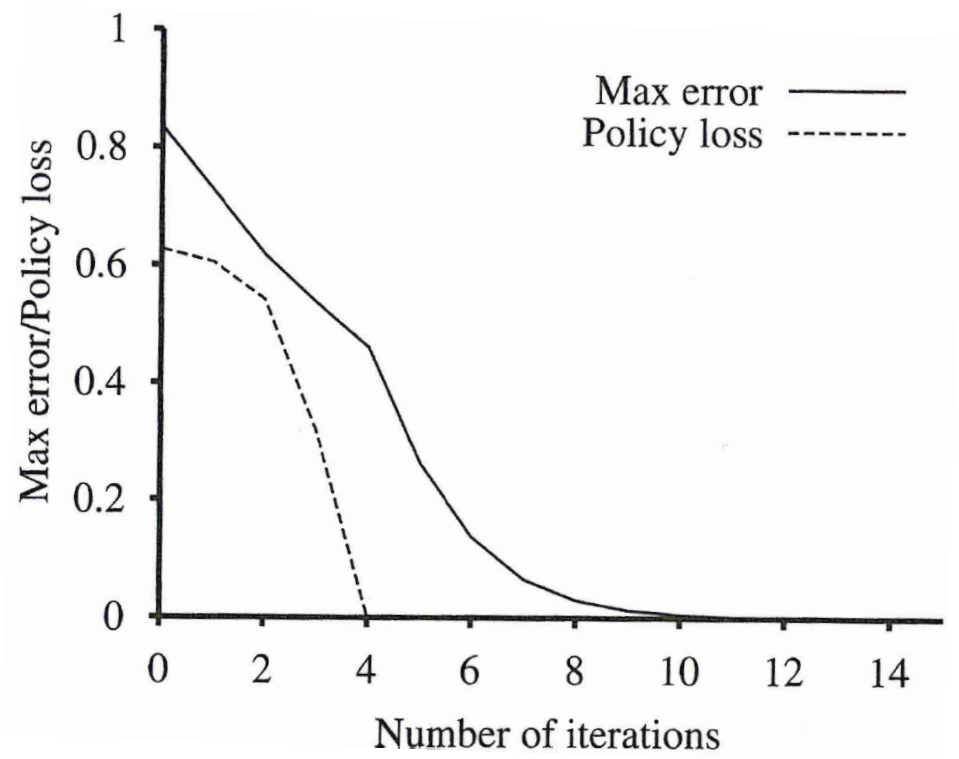
Iterace hodnot: ztráta strategie

Agenta fakticky zajímá, jak dobrá je strategie π_i dle aktuální hodnoty U_i vzhledem k optimální strategii $\pi^\star$

- $U^{\pi_i}(s)$ užitek získaný prováděním π_i ze stavu s
- definujeme ztrátu strategie π_i jako $|U^{\pi_i} - U|$
- pokud $|U_i - U| < \epsilon$, potom $|U^{\pi_i} - U| < \epsilon\,\gamma/(1-\gamma)$

V praxi strategie často konverguje k optimální strategii rychleji než funkce užitku a pevného bodu dosáhne dříve

Viz náš příklad s $\gamma = 0.9$:



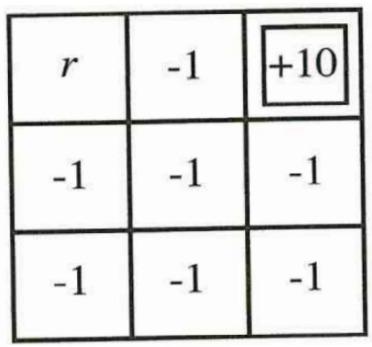
Max error $|U_i - U|$

Iterace hodnot: cvičení

Uvažujte 3×3 svět. Přechodový model je stejný jako v původním 3×4 světě: 80% času jde agent směrem, který si vybere, a zbytek času se pohybuje v pravých úhlech směrem k předpokládanému směru.

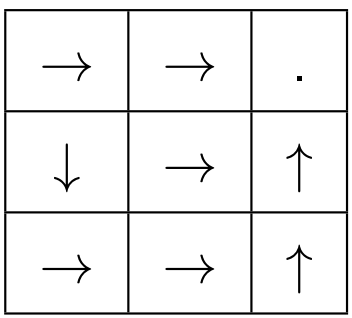
Realizujte iteraci hodnot pro každou uvedenou hodnotu r. Používejte faktor slevy 0.99 a vypočítejte strategii pro každé r. Vysvětlete intuitivně, proč jednotlivé hodnoty r vedou k dané strategii.

- $r = -100$
- $r = -3$
- $r = 0$
- $r = +3$

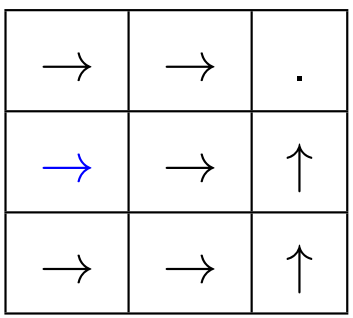


Výsledné strategie

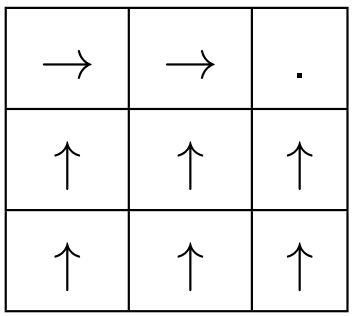
$r = -100$



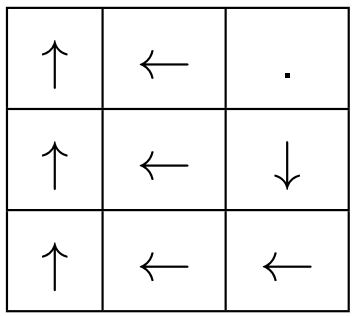
$r = -3$



$r = 0$



$r = 3$



Iterace strategie: princip

Optimální strategii můžeme získat, i když je funkce užitku ještě nepřesná

- pokud jedna akce jasně lepší než ostatní, pak užitek stavu nemusí být přesný

To můžeme využít u jiného algoritmu řešení MDP, tzv.

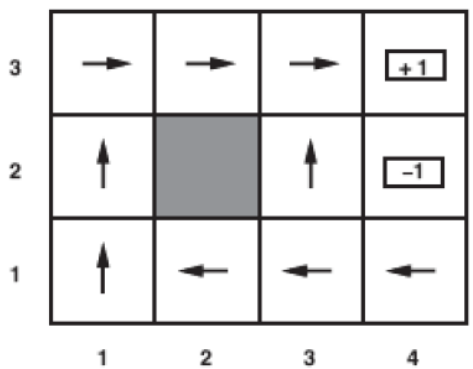
iterace strategie, který je založený na opakování následujících kroků

1. evaluace strategie – výpočet U^{π_i} pro strategii π_i
   - protože známe akce, řešíme zjednodušené (lineární) Bellmanovy rovnice

$$U^{\pi_i}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s))\, U^{\pi_i}(s')$$

zjednodušeno z $U_{i+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)\, U_i(s')$, protože použijeme přímo stav $\pi(s)$

Příklad:



- Uvažujme ocenění stavů -0.04 a strategii na obrázku
- Pak máme $\pi_i(1,1) = Up, \pi_i(1,2) = Up, \ldots$

  a zjednodušené rovnice

$$U_i(1,1) = -0.04 + 0.8U_i(1,2) + 0.1U_i(1,1) + 0.1U_i(2,1)$$

- $U_i(1,2) = -0.04 + 0.8U_i(1,3) + 0.2U_i(1,2)$
- ...

Iterace strategie: princip (pokračování)

Optimální strategii můžeme získat, i když je funkce užitku ještě nepřesná

- pokud jedna akce jasně lepší než ostatní, pak užitek stavu nemusí být přesný

To můžeme využít u jiného algoritmu řešení MDP, tzv.

iterace strategie, který je založený na opakování následujících kroků

1. evaluace strategie – výpočet U^{π_i} pro strategii π_i
   - protože známe akce, řešíme zjednodušené (lineární) Bellmanovy rovnice

     $$U^{\pi_i}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s))\, U^{\pi_i}(s')$$

     zjednodušeno z $U_{i+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)\, U_i(s')$, protože použijeme přímo stav $\pi(s)$
   - pro malý počet stavů lze použít přesný výpočet metodami lineární algebry s časovou složitostí $O(n^3)$
   - pro větší počet stavů použijeme několik kroků zjednodušené iterace hodnot (zjednodušení $\sim$ strategie je zafixována):

     $$U_{j+1}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s))\, U_j(s')$$

2. zlepšení strategie

   $$\pi_{i+1}(s) = \text{argmax}_{a \in A(s)} \sum_{s'} P(s'|s,a) U^{\pi_i}(s')$$

Iterace strategie: algoritmus (se zjednodušenou iterací hodnot)

function iterace-strategie(mdp) **returns** strategii

(mdp: MDP a jeho stavy S, akce $A(s)$, přech. model $P(s'|s,a)$
U: vektor užitku pro stavy v S, iniciálně 0
π: vektor strategie pro stavy v S, iniciálně náhodný)

repeat

$U =$ evaluace-strategie(π, U, mdp);

$zmena =$ false;

for each stav $s \in S$ **do**

(existuje lepší akce pro s než aktuální $\pi[s]$?)

if $\max_{a \in A(s)} \sum_{s'} P(s'|s,a)\, U[s'] > \sum_{s'} P(s'|s,\pi[s])\, U[s']$ **then**

$\pi[s] = \text{argmax}_{a \in A(s)} \sum_{s'} P(s'|s,a)\, U[s']$;

$zmena = true$;

until not($zmena$);

return π

Strategií konečně mnoho, při každém kroku strategii zlepšena $\Rightarrow$ alg. musí skončit

Často mnohem efektivnější než iterace hodnot

Práce s neurčitostí: shrnutí

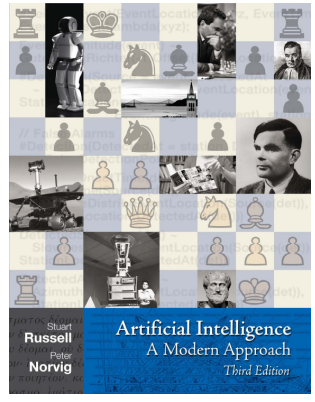
- Teorie pravděpodobnosti nám umožní kvantifikovat, jak máme danému tvrzení věřit na základě pozorování
  - Bayesovské sítě a odvozování pomocí nich
    - staticky v daném čase
  - Pravděpodobnostní uvažování o čase
    - reprezentace přechodů s pravděpodobností: Markovský proces
    - zjištění pravděpodobnosti stavu vzhledem k probíhajícímu času
- Teorie užitku pro ohodnocení rozhodnutí
- Teorie rozhodování
  - pro jedno rozhodnutí (statický případ)
  - pro posloupnost rozhodnutí

Robotika

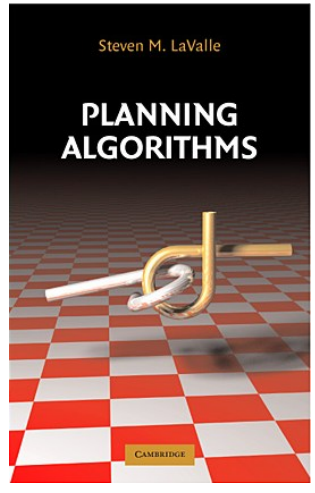
8. ledna 2021

Zdroje:

Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, third edition. Prentice Hall, 2010.
`http://aima.cs.berkeley.edu/`



Steven M. LaValle, Planning algorithms, Cambridge University Press, 2006.
`http://planning.cs.uiuc.edu/`



Robotika: obsah

Robot

Robot: fyzický agent, který vykonává úlohy manipulací s fyzickým světem

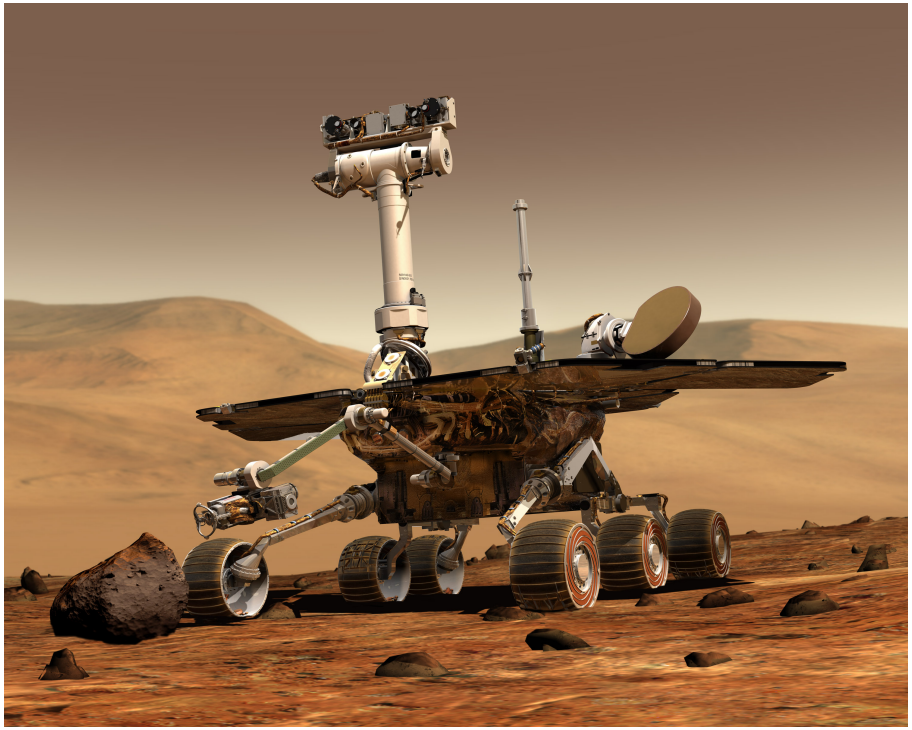
- pro realizaci úloh robot vybaven efektory
  - např. nohy, kola, klouby, chapadla
  - účelem efektoru je uplatnění fyzických sil na prostředí
- dále robot vybaven senzory, které umožňují vnímání
  - např. kamery a lasery pro pozorování prostředí, gyroskopy a akcelerometry pro měření pohybu robota

Typy robotů

- manipulátor: fyzicky připevněné k pracovnímu místu
  - robotická ruka
- mobilní robot: pohybují se v prostředí pomocí kol, nohou apod.
  - bezpilotní pozemní vozidlo (unmanned ground vehicle UGV) pro autonomní jízdu
  - planetární vozidlo (planetary rover)
  - bezpilotní letadlo (unmanned air vehicle UAV)
  - bezpilotní ponorka (autonomous underwater vehicle AUV)
- mobilní manipulátor: kombinace mobility s manipulací – humanoidní roboti

Mobilní manipulátor: příklady

NASA Mars Exploration Rover
(vesmír)



- 2003: odlet na Mars dvojice robotů, přistání 2004
- výzkum skal, půdy, dřívější vodní aktivity (robotické paže)

DLR Rollin' Justin
(profesionální a domácí služby)

- představen 2008 firmou DLR, poloviční humanoid, posun na kolech
- 2012: rychlejší paže (ruce se 4 prsty) pro interakci se všemi typy objektů

Mobilní manipulátor: příklady – armáda

Detekce a zneškodňování těžkých výbušných zbraní a improvizovaných výbušných zařízení

ECA Group TSR202

- představen 2009 (ECA Group)
- 70 kg kapacita

Iguana E

- představen 2015 (ECA Group)
- lehký modulární robot (20 kg zdvih)
- obtížně dosažitelné prostory

Mobilní manipulátor: příklady – průmysl

Automatizace průmyslové výroby

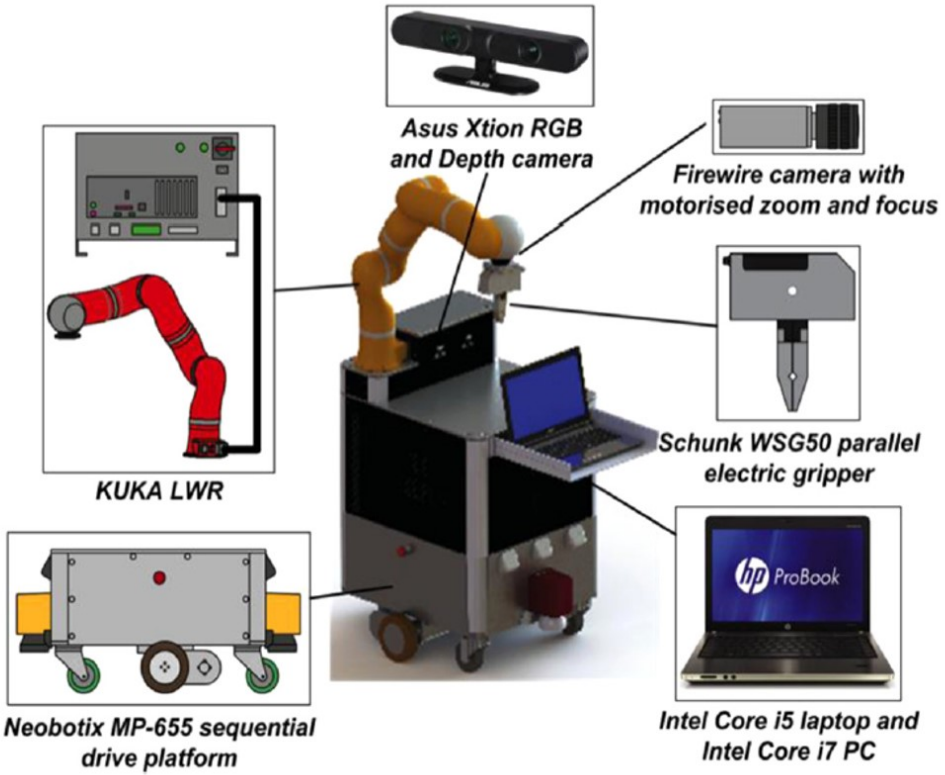
Neobotix MM-800

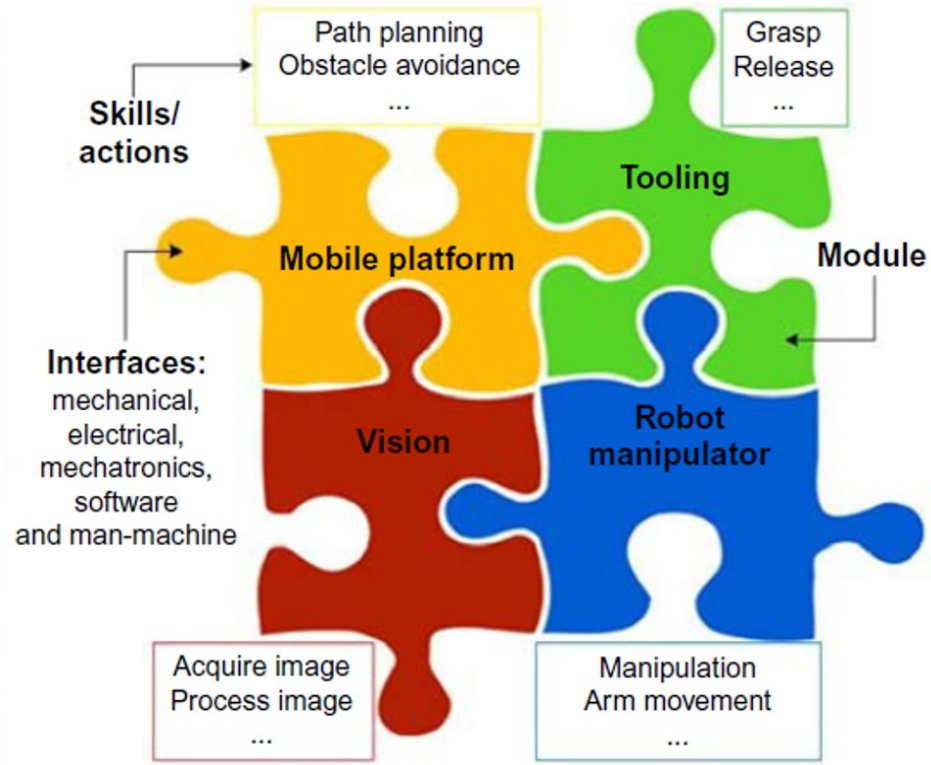
- 2010:
  použití v továrně Audi AG

KMR iiwa

- představen 2016 (KUKA AG)

Konfigurace mobilního manipulátoru: příklad





- Mobilní platforma: pohyb, navigace, plánování cesty (překážky)
- Manipulace robota: robotická paže pro montážní úkony
- Pořízení a zpracování obrazu
- Práce s nástrojem: 2 „prsty”

Sociální robot – příklad z FI

Robot Pepper: `https://nlp.fi.muni.cz/trac/pepper/`

`https://nlp.fi.muni.cz/trac/pepper/chrome/site/pepper/2019-02-04-m2u00747-meet-and-can.ogg`

Robotický hardware: senzory

Senzory představují rozhraní mezi robotem a prostředím

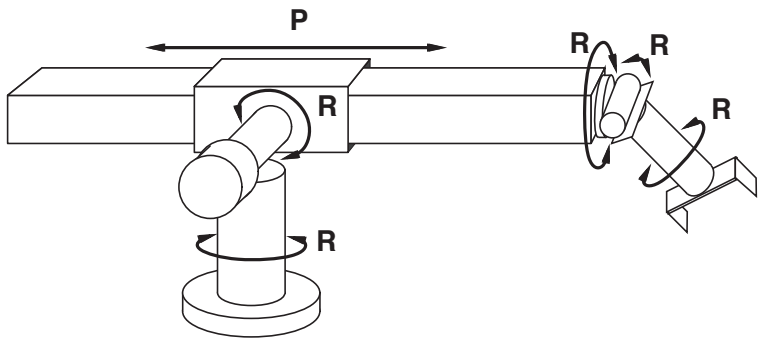
- pasivní senzory slouží k pozorování prostředí, kdy zachycují signály generovanými dalšími zdroji v prostředí
- aktivní senzory vysílají energii do prostředí (např. sonar)
  založeny na tom, že je energie reflektována senzoru zpět
  - poskytují více informací než pasivní senzory
  - náročnější na energii než pasivní senzory
  - hrozí nebezpečí interfence při použití více aktivních senzorů zároveň

Senzory dělíme v závislosti na tom, zda určeny pro vnímání

- prostředí, tj. dálkoměry pro měření vzdálenosti od blízkých objektů
- robotova umístění, tj. lokační senzory
  - časté řešení lokalizačního problému: Global Positioning System (GPS)
- robotovy vnitřní konfigurace, např.
  - pro měření přesné konfigurace robotických kloubů, motorů
  - gyroskopy – pro udržení referenčního směru
  - snímače točivého momentu – pro práci s křehkými objekty

Robotický hardware: efektory

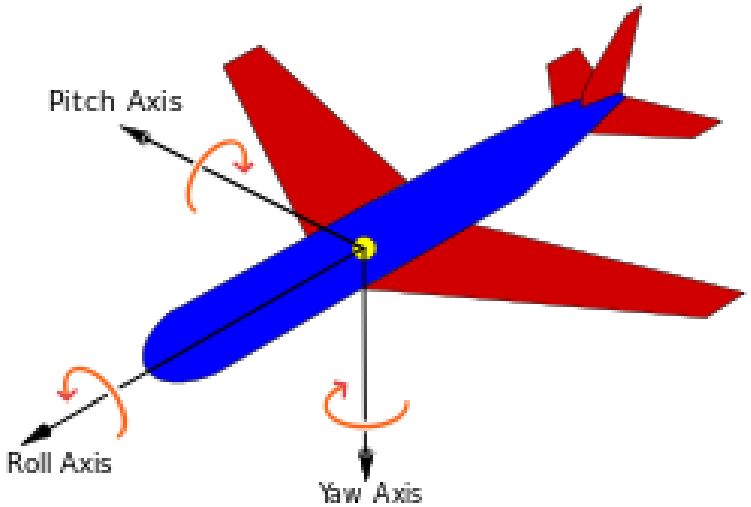
- Efektory umožňují robotovi pohyb a změnu tvaru těla
- Stupeň volnosti (degree of freedom DOF)
  - umožňuje pochopit návrh efektoru
  - jeden DOF pro každý nezávislý směr, ve kterém se robot nebo jeden z jeho efektorů může pohybovat



robotická paže s 6 DOFs
klouby pro rotační pohyb (R)
kloub pro posun (P)

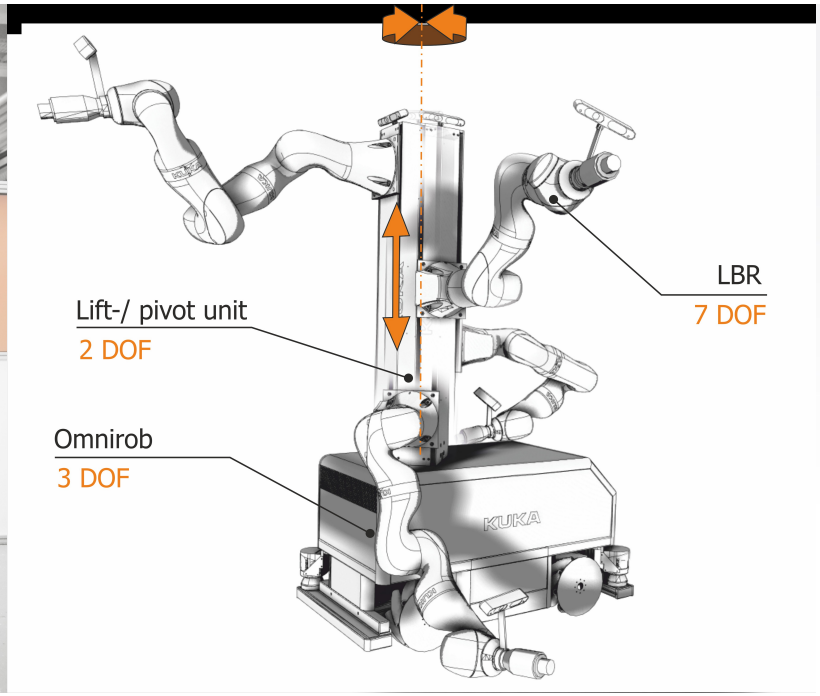
Efektory, kinematický a dynamický stav

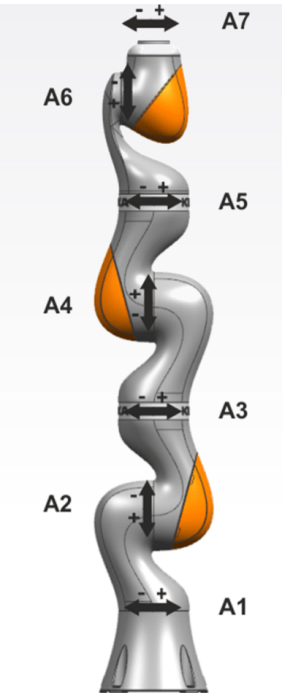
- Kinematický stav: určen DOFs
  - u AUV určen 6 DOFs: (x, y, z) + úhlová orientace *yaw, roll, pitch*



- Dynamický stav: zahrnuje DOF kinematického stavu + dimenze pro rychlost změny kinematické dimenze (např. 6+6)
- Počet DOFs nemusí být stejný jako počet ovládaných prvků (mobilní roboti)
  - př. auto: pohyb dopředu/dozadu, otáčení, tj. 2 DOFs
    kinematická konfigurace třídimenzionální – na otevřeném plochém povrchu pohyb do libovolného (x, y) v libovolné orientaci θ
    3 skutečné DOFs avšak jen 2 regulovatelné DOFs (non-holonomic)
  - pokud by byl počet ovládaných a skutečných DOFs stejný, robot by se snáze ovládal (např. auto by šlo posunout do strany)

Příklad: VALERI OmniRob





Robotická paže s 12 DOFs:

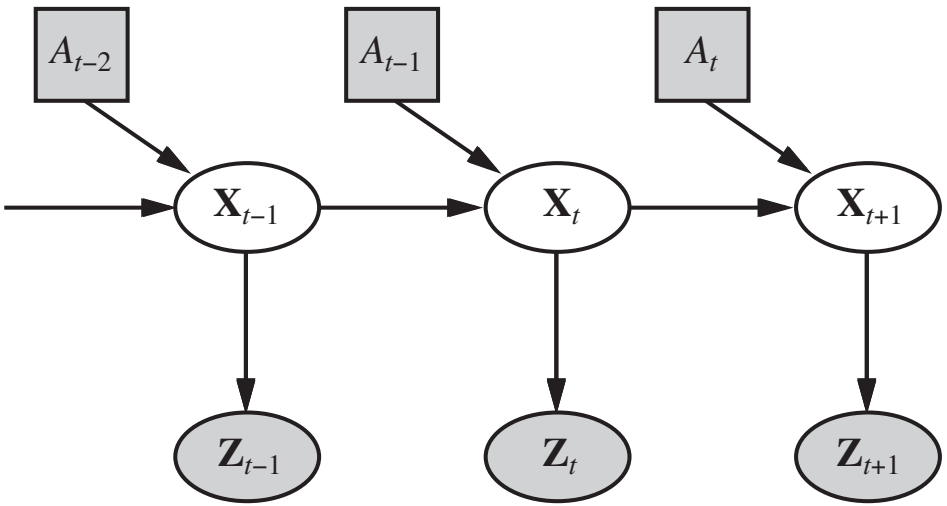
- 3 DOFs (2 translační, 1 rotační) pro základnu s koly pro přesun robota v libovolném směru
- 2 DOFs pro sloupec (1 translační – nahoru a dolu, 1 rotační)
- 7 DOFs pro robotickou paži (1 rotační pro každý spoj – viz vpravo)

Vnímání robota

Vnímání:

- proces, kdy robot mapuje měření senzorů na vnitřní reprezentaci prostředí
- obtížné, protože senzory jsou nepřesné (šum), prostředí je částečně pozorovatelné, nepředvídatelné a často dynamické
- tj. robot řeší problémy filtrace/odhadu stavu (viz přednáška Čas a neurčitost: $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$), kdy je cílem zjištění pravděpodobnosti aktuálního stavu na základě dosavadních pozorování

Robotovo vnímání můžeme chápat jako temporální odvozování z posloupnosti akcí a pozorování



- $\mathbf{X}_t$ stav prostředí (včetně robota) v čase t
- $\mathbf{Z}_t$ pozorování zjištěné v čase t
- $\mathbf{A}_t$ akce realizovaná **po** zjištění pozorování

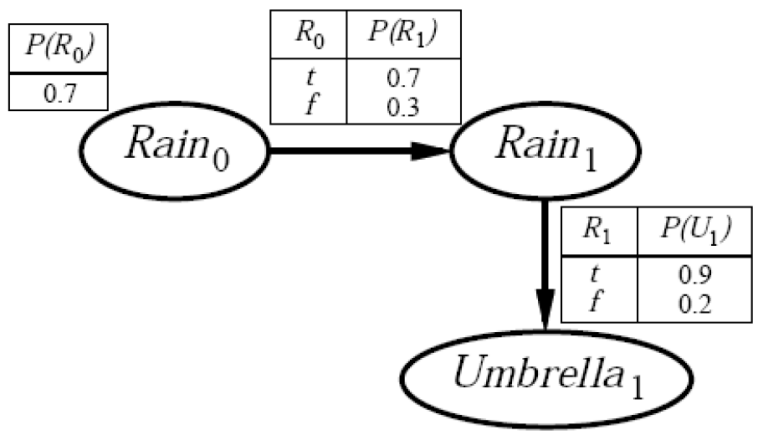
Dynamické Bayesovské sítě

Připomenutí:

- Bayesovské sítě – DAG s tabulkami pravděpodobnostní distribuce $P(X|Parents(X))$ pro uzly reprezentující náhodné proměnné
- příklad s agentem a pozorováním počasí

Dynamická Bayesovská síť (DBN) reprezentuje temporální pravděpodobnostní model

- skládá se z opakujících se vrstev proměnných
  - stačí tedy popsat první vrstvu
  - apriorní distribuci $\mathbf{P}(\mathbf{X}_0)$
  - přechodový model $\mathbf{P}(\mathbf{X}_1|\mathbf{X}_0)$
  - senzorický model $\mathbf{P}(\mathbf{E}_1|\mathbf{X}_1)$



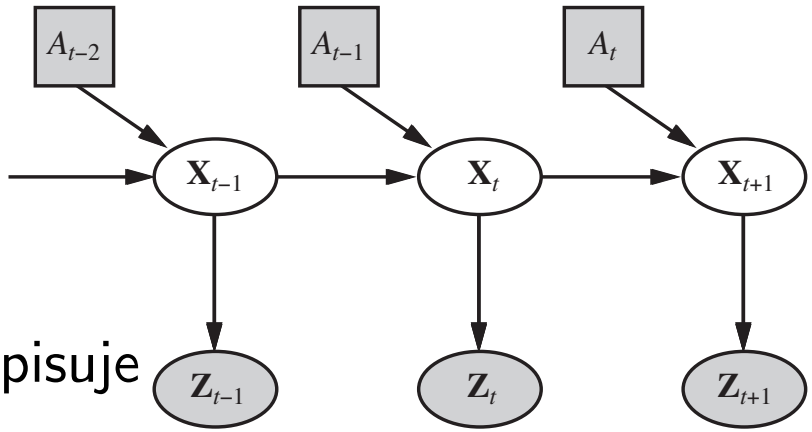
- dle Markovkých předpokladů má každá proměnná rodiče buď ve stejné nebo předchozí vrstvě

Filtrace: $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = \alpha\, \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)\, P(\mathbf{x}_t|\mathbf{e}_{1:t})$

Odhad stavu (filtrace)

Dynamická Bayesovská síť reprezentuje
přechodový model (model pohybu)
a senzorický model
v částečně pozorovaném prostředí

- přechodový model $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, a_t)$
- senzorický model $\mathbf{P}(\mathbf{z}_{t+1}|\mathbf{X}_{t+1})$ popisuje na čem závisí nové pozorování $\mathbf{z}_{t+1}$



Zajímá nás pravděpodobnost nového stavu $\mathbf{X}_{t+1}$ na základě dosavadních pozorování (pomocí senzorů) $z_{1:t+1}$ a akcí $a_{1:t}$, tj. $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{z}_{1:t+1}, a_{1:t})$

- př. $\mathbf{X}_{t+1}$ může určovat umístění fotbalového míče relativně k robotovi
- na rozdíl od dříve uvažované filtrace musíme kromě pozorování zahrnout i akce a také spojité místo diskrétních náhodných proměnných
- tj. odvození je stejné jako u filtrace, ale místo součtu používáme integraci

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{z}_{1:t+1}, a_{1:t}) = \alpha\, \mathbf{P}(\mathbf{z}_{t+1}|\mathbf{X}_{t+1}) \int \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, a_t)\, P(\mathbf{x}_t|\mathbf{z}_{1:t}, a_{1:t-1})\, \mathrm{d}\mathbf{x}_t$$

- máme tak rekurzivní filtrační rovnici, kdy se pravděpodobnost $\mathbf{X}$ v čase $t+1$ spočítá na základě odhadu o časovou jednotku dříve

Typy vnímání

Lokalizace: problém zjišťování polohy objektů včetně robota

- můžeme znát mapu, a pak hledáme pozici na mapě (úloha mapování)
- mapa nemusí být k dispozici,
  pak je realizována současně lokalizace a mapování

Vnímání teploty, pachů, akustických signálů apod.

Lze vyhodnotit pomocí

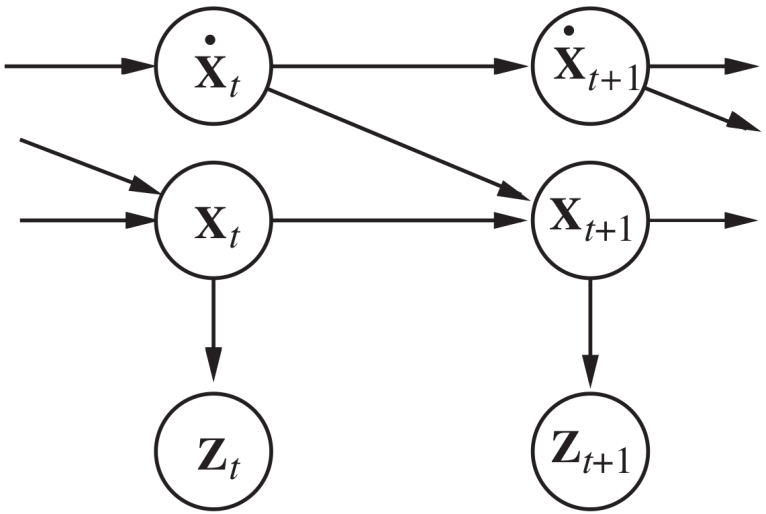
- dynamických Bayesovských sítí, např. Monte Carlo lokalizace
  - potřebujeme znát podmíněné pravděpodobnostní distribuce
    charakterizující evoluci stavu proměnných v čase a
    senzorické modely, které popisují vztahy k pozorovaným proměnným
  - použitelné pro diskrétní proměnné
- Kalmanových filtrů
  - vhodné pro spojité proměnné
- strojového učení

Více např. Norvig & Russel nebo Barták, UI II

Příklad DBN: robot a jeho pozice + rychlost

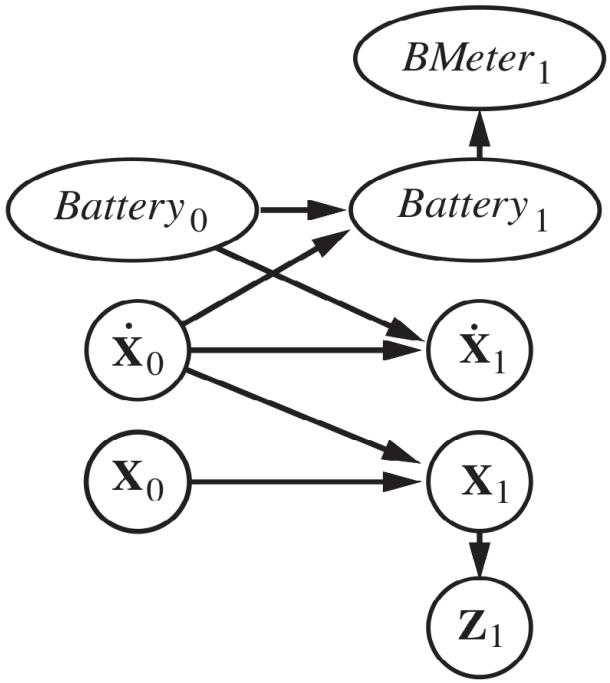
Popis chování robota

- poloha robota $\mathbf{X}_t = (X_t, Y_t)$
- rychlost robota $\dot{\mathbf{X}}_t = (\dot{X}_t, \dot{Y}_t)$
- pozorování polohy pomocí GPS $\mathbf{Z}_t$



Přidáme baterii

- stav baterie $\mathbf{Battery}_t$
  - další poloha záleží na rychlosti a původní poloze
  - další rychlost záleží na předchozí rychlosti a stavu baterie
- senzor $\mathbf{BMeter}_t$
  - pozorování stavu baterie



Rozvrhování robotů v továrně

Job shop problém s transportem a zpracováním roboty

Stroje

- rozmístěné v továrně dle matice vzdáleností
- jeden stroj může zpracovávat nejvýše jednu úlohu

Úlohy: výroba výrobků

- nepřekrývající se operace
- určeno pořadí operacích na stroji
- odlišný stroj pro každou operaci

Roboti: identiční

- převáží vyráběné výrobky
- zpracovávají některé operace se stroji

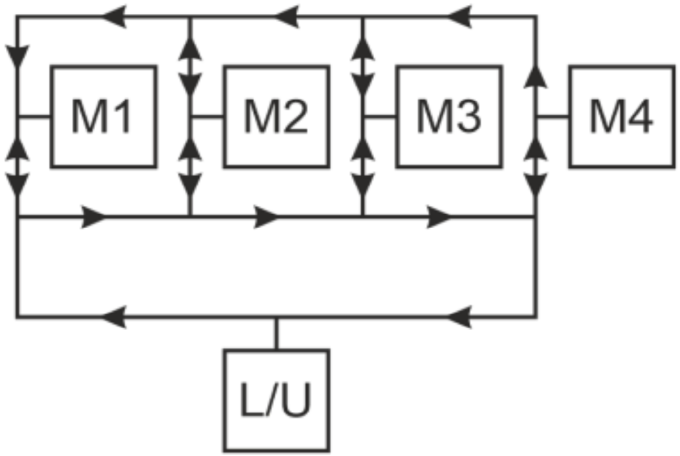
Jednoduchý příklad

Machine	L/U	M1	M2	M3	M4
L/U	0	6	8	10	12
M1	12	0	6	8	10
M2	10	6	0	6	8
M3	8	8	6	0	6
M4	6	10	8	6	0

	Op.1	Op.2	Op.3
Job 1	M1(8)	**M2(16)**	M4(12)
Job 2	**M1(20)**	M3(10)	M2(18)
Job 3	M3(12)	**M4(8)**	M1(15)
Job 4	M4(14)	M2(18)	–
Job 5	**M3(10)**	M1(15)	–

Cíl

- přiřazení startovního času a robota pro operace a transport
- minimalizovat čas dokončení výroby, tzv. makespan C_{max}

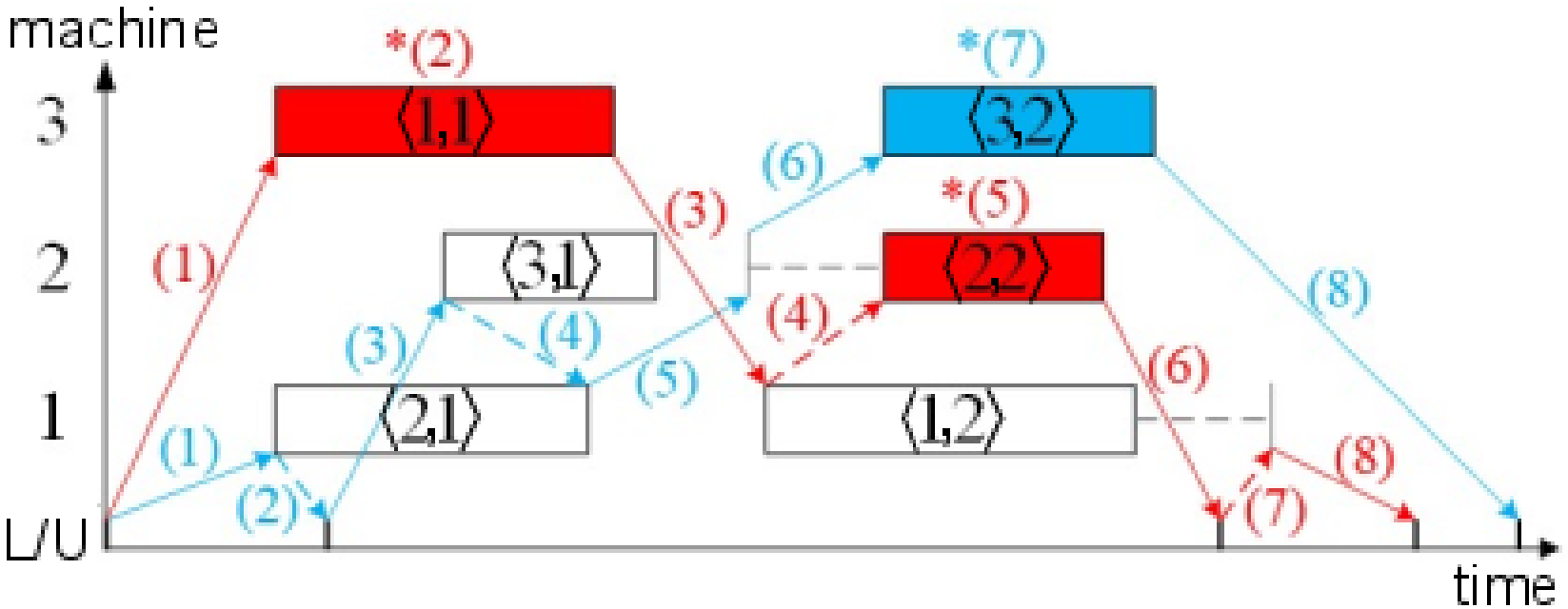


Genetický algoritmus kombinovaný s tabu prohledáváním

$t \leftarrow 1$
initialize P_t *(P_t population of individuals/solutions)*
tabu search P_t; evaluate P_t
while $t \leq$ *maxGenerations* **do**
  if partial reinitialization **then**
    initialize I_t to become a part of P_t
    tabu search I_t; evaluate I_t
  create C_t by crossover of individuals from P_t
  tabu search C_t; evaluate C_t
  replace the worst inidividuals from P_t by C_t
  create M_t by mutation of individuals from P_t
  tabu search M_t; evaluate M_t
  replace the chosen inidividuals in P_t by M_t
  decrease population size of P_t by eliminationg the worst individuals
  update and keep the best individuals in P_t
  $t \leftarrow t + 1$

Řešení (jedinci/chromozomy), funkce vhodnosti (fitness)

Chromozom e	e_1	e_2	e_3	e_4	e_5	e_6
Posloupnost ⟨j,o⟩	⟨2,1⟩	⟨1,1⟩	⟨3,1⟩	⟨1,2⟩	⟨2,2⟩	⟨3,2⟩
Transportní robot	1	2	1	2	1	1
Zpracovávající robot	–	2	–	–	2	1
Stroj	1	3	2	1	2	3



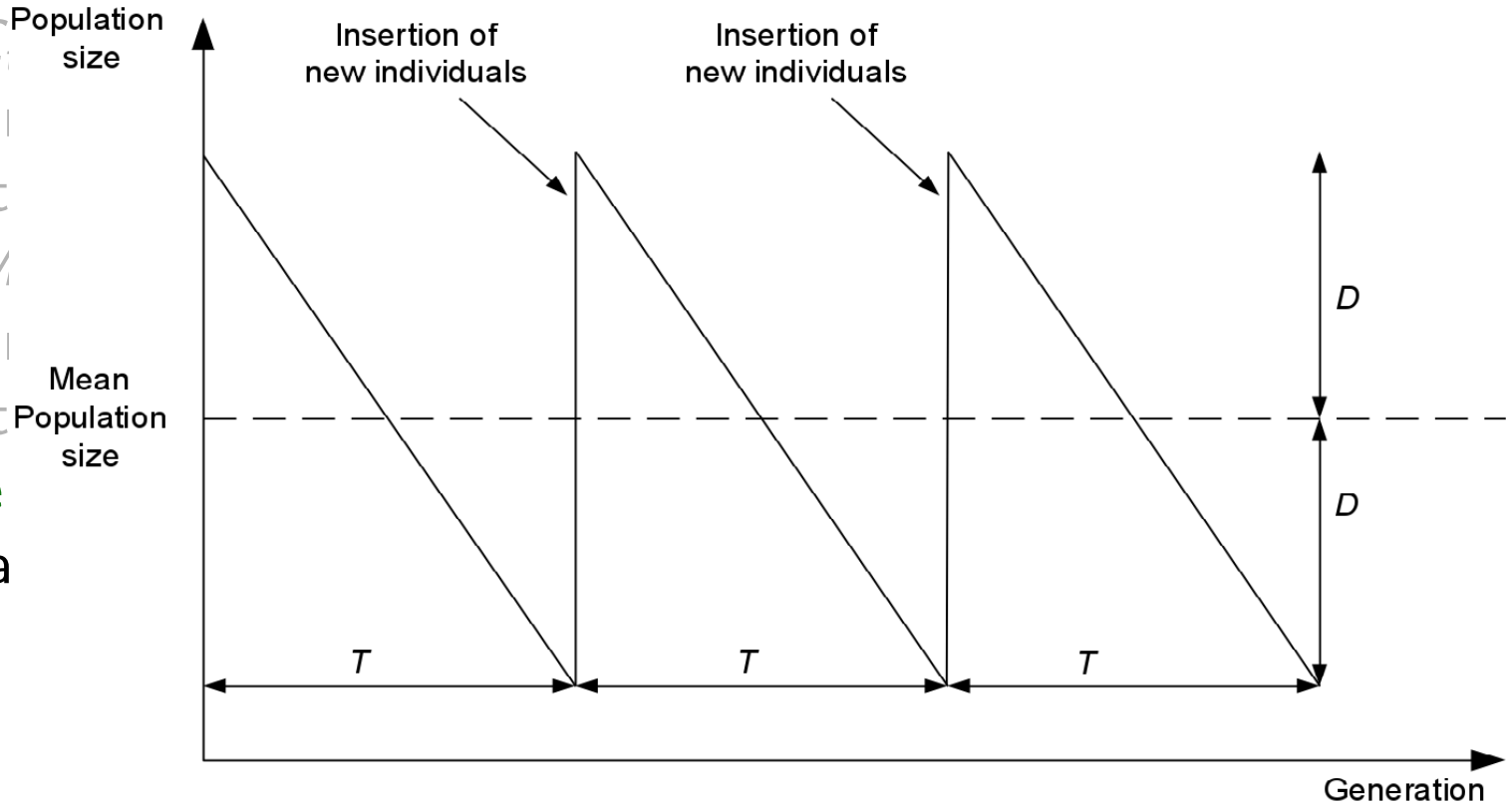
Tabu prohledávání na robotech

$t \leftarrow 1$
initialize P_t
tabu search P_t; evaluate P_t
while $t \leq$ *maxGenerations* **do**
    ...
    update and keep the best individuals in P_t
    $t \leftarrow t + 1$

tabu search P_t:
  forall individuals e in P_t
    repeat
      random change of transporting robot in e
      random change of processing robot in e
      assigned robots stored in tabu list not to be selected repeatedly
      if new better individual e' **then** $e = e'$

Reinicializace

$t \leftarrow 1$
initialize P_t
tabu search P_t; evaluate P_t
while $t \leq$ *maxGenerations* **do**
    if partial reinitialization **then**
        initialize I_t to become a part of P_t
        tabu search I_t; evaluate I_t
    create C_t
    tabu sea
    replace t
    create M
    tabu sea
    replace t
    decrease
    update a
    $t \leftarrow t +$

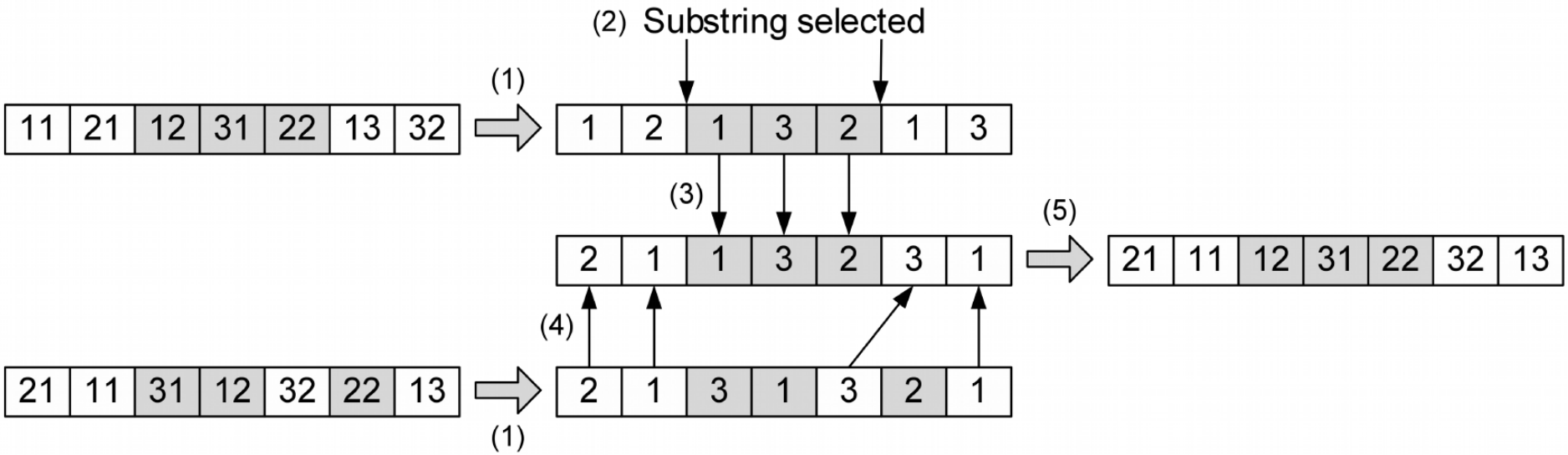


Křížení & mutace pro posloupnosti operací

$t \leftarrow 1$
initialize P_t
tabu search P_t; evaluate P_t
while $t \leq$ *maxGenerations* **do**
    update and keep the best individuals in P_t
    if partial reinitialization **then**
        initialize I_t to become a part of P_t
        tabu search I_t; evaluate I_t
    create C_t by crossover of individuals from P_t
    tabu search C_t; evaluate C_t
    replace the worst inidividuals from P_t by C_t
    create M_t by mutation of individuals from P_t
    tabu search M_t; evaluate M_t
    replace the chosen inidividuals in P_t by M_t
    decrease population size of P_t by eliminationg the worst individuals
    $t \leftarrow t + 1$

Posloupnost operací: operátor křížení & mutace

Křížení



Pravděpodobnost křížení a mutace

Mutace

- náhodná operace ⟨j,o⟩ vložena náhodně mezi předchůdce ⟨j,o-1⟩ a následníka ⟨j,o+1⟩

Různé metody řešení

Heuristické přístupy

- Genetický algoritmus s tabu prohledáváním [1]
- Adaptivní prohledávání s velkým okolím [2]
  - sada heuristik pro výběr souseda
  - výpočet váhy pro heuristiky dle úspěšnosti
  - výběr heuristiky dle jejich váhy

Exaktní přístupy

- Celočíselné programování [1]
- Programování s omezujícími podmínkami [3]

---

1. *Dang, Nguyen, Rudová, Scheduling of mobile robots for transportation and manufacturing tasks. Journal of Heuristics, 25(2):175–213, 2019.*
2. *Dang, Rudová, Nguyen, Adaptive large neighborhood search for scheduling of mobile robots. In Genetic and Evolutionary Computation Conference (GECCO), pages 224-232, 2019.*
3. *Murín and Rudová, Scheduling of Mobile Robots using Constraint Programming. In 25th International Conference on Principles and Practice of Constraint Programming (CP), pages 456–471. Springer LNCS 11802, 2019.*

Plánování pohybu robota: konfigurační prostor

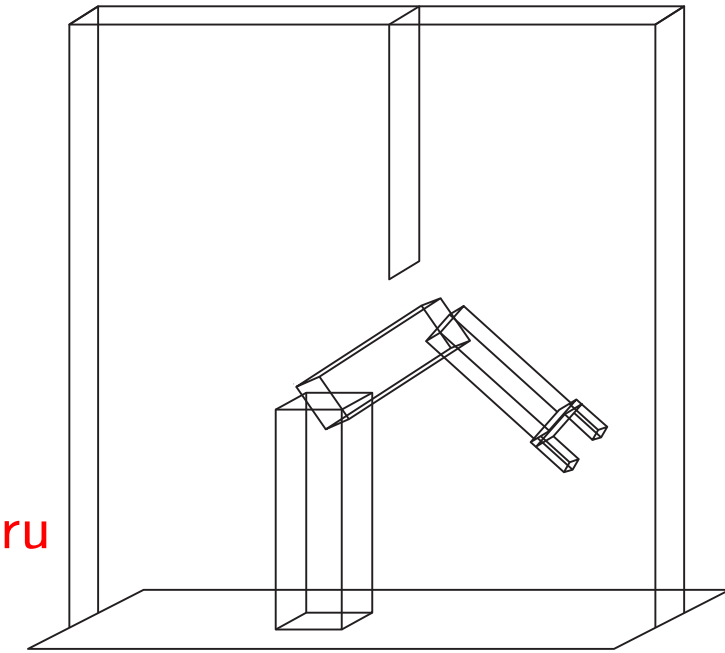
- Základem je rozhodování, jakým způsobem přesunout efektory
- Pohyb z bodu do bodu: přesun robota (nebo jeho efektoru) do cíle
- Koordinovaný pohyb (compliant motion)
  - robot se přesunuje v kontaktu s předmětem
  - např. šroubování žárovky, přesun krabice na stůl
- Konfigurační prostor (configuration space, $\mathcal{C}$-space)
  - používá se pro reprezentaci plánovacího problému
  - je to prostor stavů robota definovaný
    jeho umístěním, orientací a úhly kloubů
  - vhodnější reprezentace než původní 3D prostor (viz dále robotická paže)
    tzv. pracovní prostor

Reprezentace pracovního prostoru

Robotická paže se dvěma klouby

- pohyb kloubů mění (x, y) souřadnice kolena a chapadla (souřadnice z se nemění)
- jak popsat konfiguraci paže?
- (x_e, y_e) umístění kolena

  (x_c, y_c) umístění chapadla



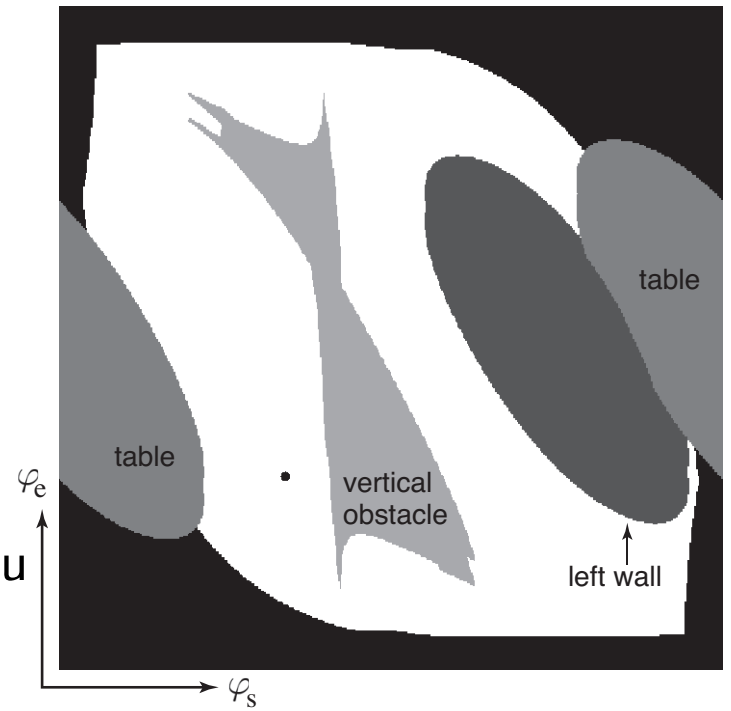
Souřadnice určují reprezentaci pracovního prostoru (workspace representation), tj.

- reálného prostoru, kdy jsou souřadnice robota zadány stejným souřadným systémem jako objekty, kterými manipulujeme
- reprezentace vhodná pro kontrolu kolizí zejména robota a objektů zadaných jednoduchými polygonálními modely
- problémy:
  - všechny souřadnice pracovního prostoru nejsou dosažitelné
  - i díky omezením daným vazbami
    - např. fixní vzdálenost (x_k, y_k) a (x_c, y_c)
  - generování cest, které dodržují tato omezení je velmi náročné!
    - plyne ze spojitého prostoru a nelinearity omezení

Reprezentace konfiguračního prostoru

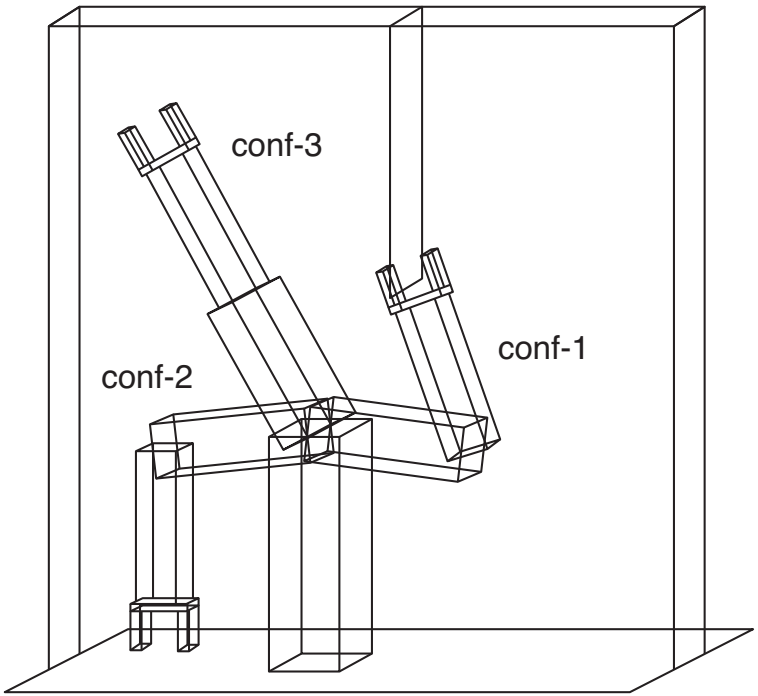
Reprezentace pomocí robotových kloubů

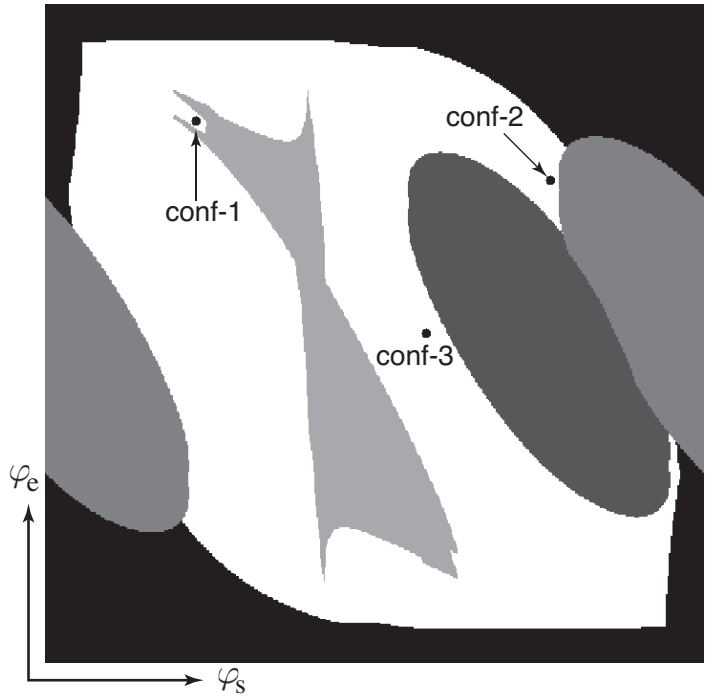
- stav reprezentován úhly φ_s a φ_e
  - φ_s úhel ramena, φ_e úhel kolena
- tj. určuje konfigurační prostor
- pokud neuvažujeme překážky, každá kombinace hodnot φ_s a φ_e je možná
- plánování cesty:
  - spojíme aktuální pozici a cíl rovnou čarou a robot se po ní pohybuje konstatní rychlostí, dokud nedorazí do cíle



- problém:
  - úloha robota většinou zadána v pracovním a ne v konfig. prostoru
- transformace souřadnic z konfiguračního prostoru do pracovního prostoru, tzv. kinematika je jednoduchá
  - lineární transformace pro otočné klouby
- inverzní kinematika (opačná transformace) obtížná (zejména při více DOFs)
  - zřídka jedno řešení, např. pozice chapadla (viz obrázek dříve) umožňují dvě konfigurace (koleno pod ramenem)

Překážky





Pracovní prostor (vlevo)

- překážky mají jednoduchý geometrický tvar

Konfigurační prostor (vpravo)

- díky překážkám je prostor členěn na obsazený (tmavé odstíny) a volný (bílá)

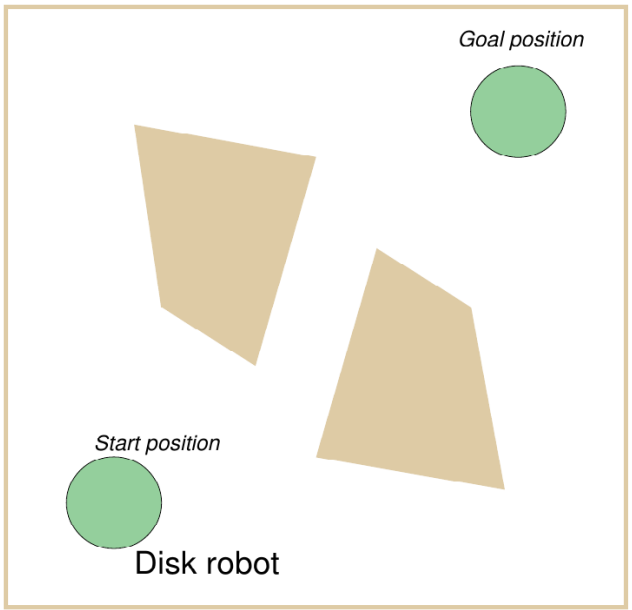
Konstrukce volného prostoru náročná, místo toho:

- plánovač generuje konfiguraci a testuje, zda je ve volném prostoru (aplikací kinematiky robota a kontrolou kolize v souřadnicích pracovního prostoru)

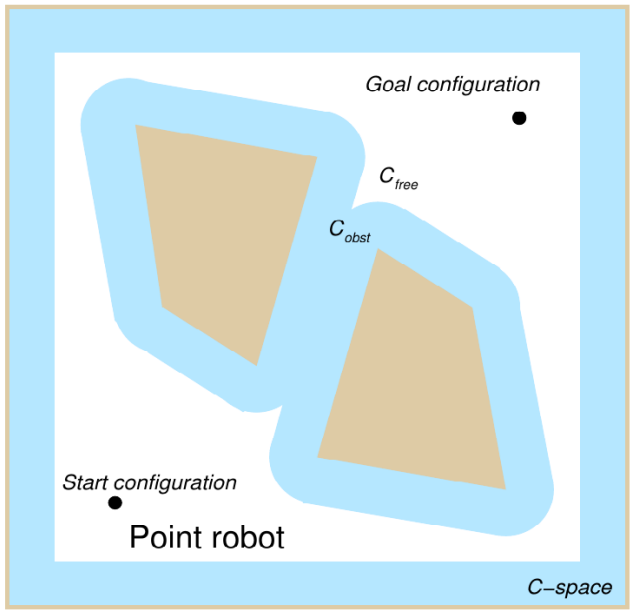
Konfigurační prostor s překážkami

Jak získat konfigurační prostor s překážkami?

Plánování pohybu diskového robota $\mathcal{A}$ s poloměrem ρ



geometrická reprezentace pracovního prostoru



konfigurační prostor s překážkami

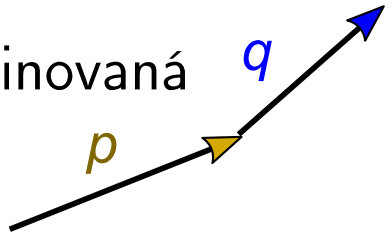
Konfigurační prostor získán zvětšením překážek diskem $\mathcal{A}$ s poloměrem ρ

- pomocí Minkowského sumy

Minkowského suma

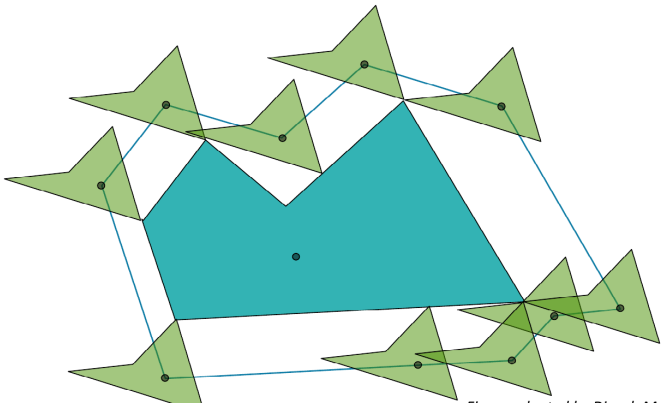
Minkowského suma dvou množin P a Q značená $P \oplus Q$ definovaná

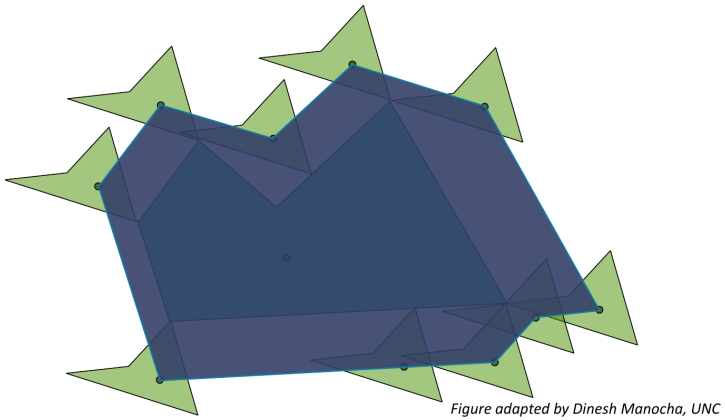
$$P \oplus Q = \{p + q \mid p \in P, q \in Q\}$$



Minkowského suma dvou konvexních polygonů P a Q s m a n vrcholy je polygon $P \oplus Q$ s $m + n$ vrcholy

- vrcholy $P \oplus Q$ jsou „součty" vrcholů P a Q





Použití

- složitost: časová $O(n + m)$, prostorová $O(n + m)$
- nekonvexní překážky
  - dekompozice na konvexní polygony, výpočet Minkowského sumy a spojení; složitost $O(n^2 m^2)$
- 3D prostor

Shrnutí: dnes a příště

- Robot, hardware
  - typy robotů
  - senzory (rozhraní robot-prostředí)
  - efektory (pro pohyb a změnu tvaru)
- Vnímání robota
  - odhad stavu (dynamické Bayesovské sítě)
  - lokalizace a mapování
- Rozvrhování robotů
  - rozvrhování transportu a zpracování roboty v továrně
  - genetický algoritmus kombinovaný s tabu prohledáváním
- Plánování pohybu robota
  - konfigurační prostor (reprezentace, překážky, Minkowského suma)
  - prohledávání v grafu (typy grafů, jump-point prohledávání)
  - přístupy k plánování pohybu (kombinatorické, pravděpodobnostní)
- Pohyb robota

(Diskrétní) plánování pohybu robota

- Plánování cesty (path planning)
  - problém nalezení cesty z jedné konfigurace do druhé v konfig. prostoru
  - na rozdíl od klasického hledání cesty v diskrétním prostoru se pohybujeme ve spojitém prostoru
- Redukce plánování cesty ve spojitém prostoru na diskrétní problémy prohledávání v grafu
  - kombinatorické metody
  - pravděpodobnostní metody
- Prohledávání váženého grafu reprezentující konfigurační prostor
  - Dijkstrův algoritmus, A* (známe)
  - Jump-point search

Spojitá reprezentace konfiguračního prostoru

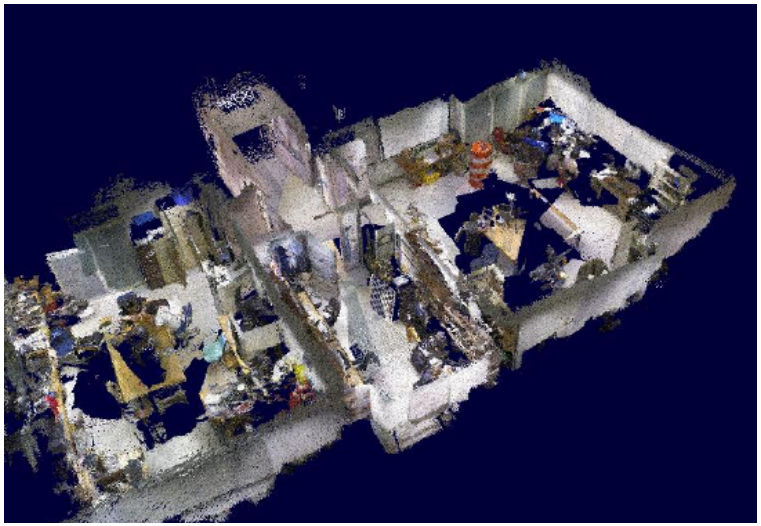
$\Downarrow$

Diskretizace

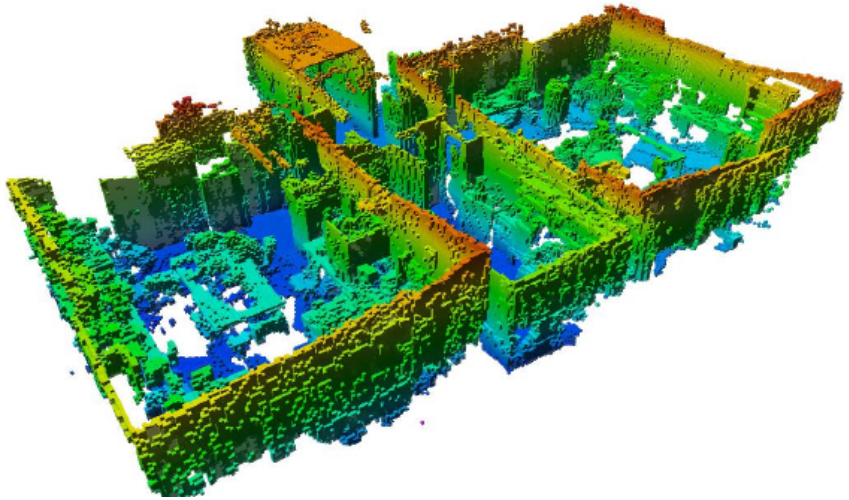
$\Downarrow$

Prohledávání grafu

Spojité plánování pohybu vs. diskrétní aproximace volného prostoru



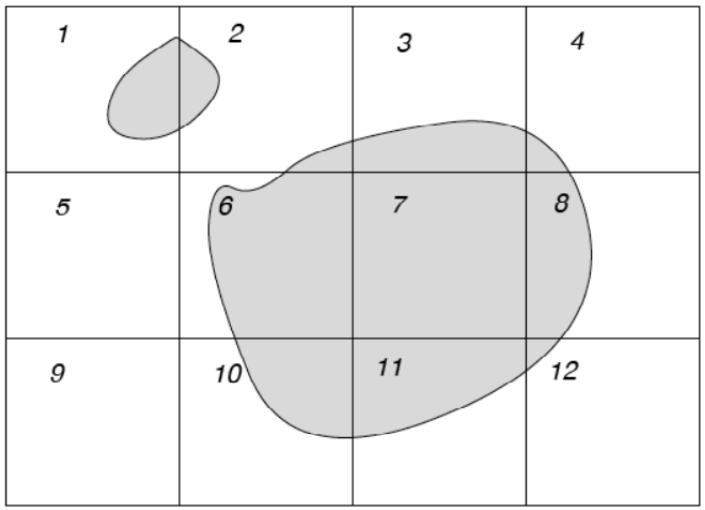
point cloud

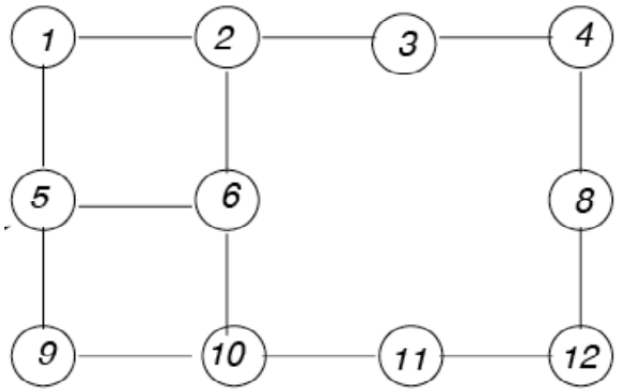


voxel map

Plánování na grafech

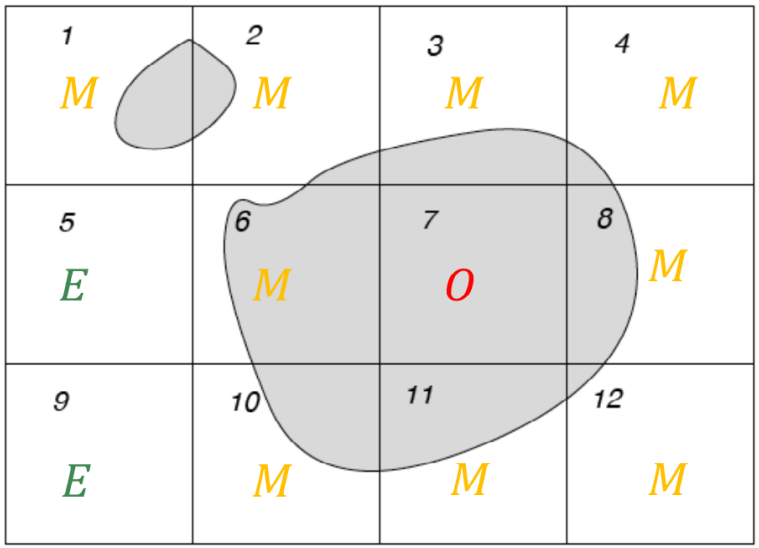
Nalezení nejkratší cesty mezi dvěma uzly grafu





Buňková dekompozice a cestovní mapy dávají diskrétní reprezentaci (ve formě váženého grafu) konfiguračního prostoru

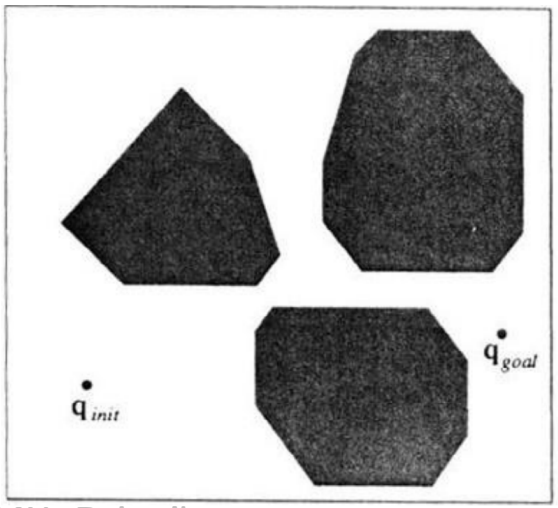
Částečně obsazené buňky

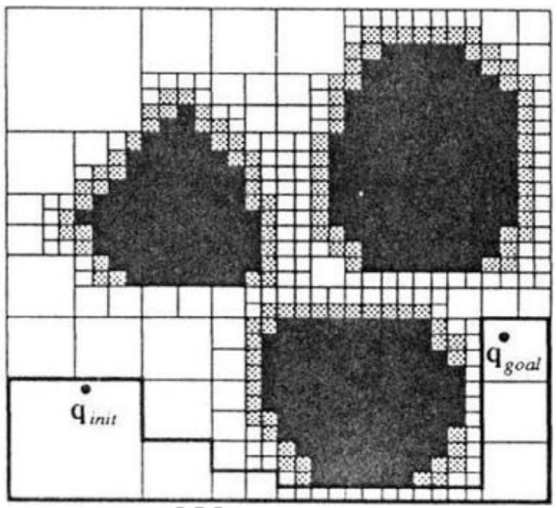


E prázdná *M* obsazená
M částečně obsazená

Zpracování částečně obsazených buněk:

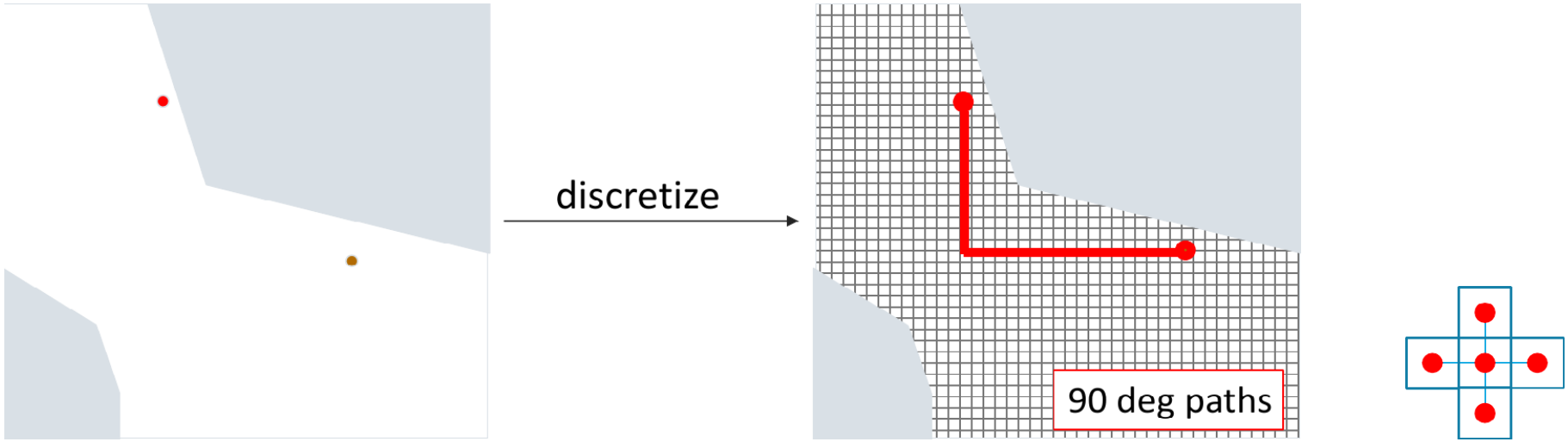
1. neprocházet tyto buňky
   - neúplné: nemusí existovat cesta
2. procházet tyto buňky
   - nekorektní: může vrátit cestu, která neexistuje
3. zvětšit rozlišení mřížky
   - drahé: zejména ve vyšší dimenzi
4. adaptivní diskretizace
   - ztráta uniformní velikosti mřížky



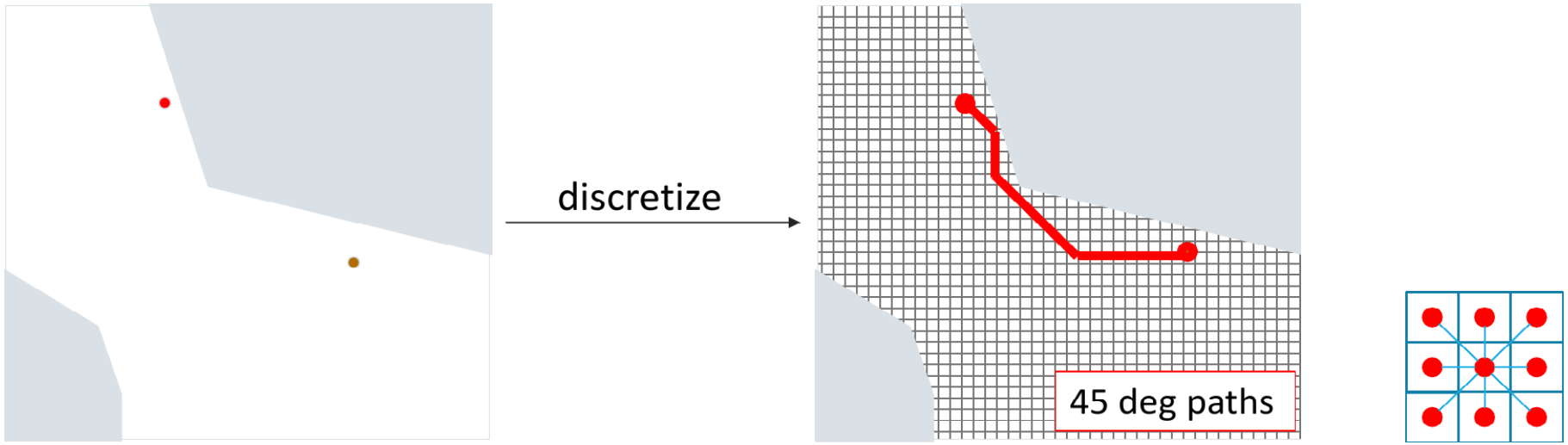


4-connected vs. 8-connected graf

4-connected graf: každý uzel spojený se 4 sousedy



8-connected graf: každý uzel spojený s 8 sousedy



Prohledávací algoritmy

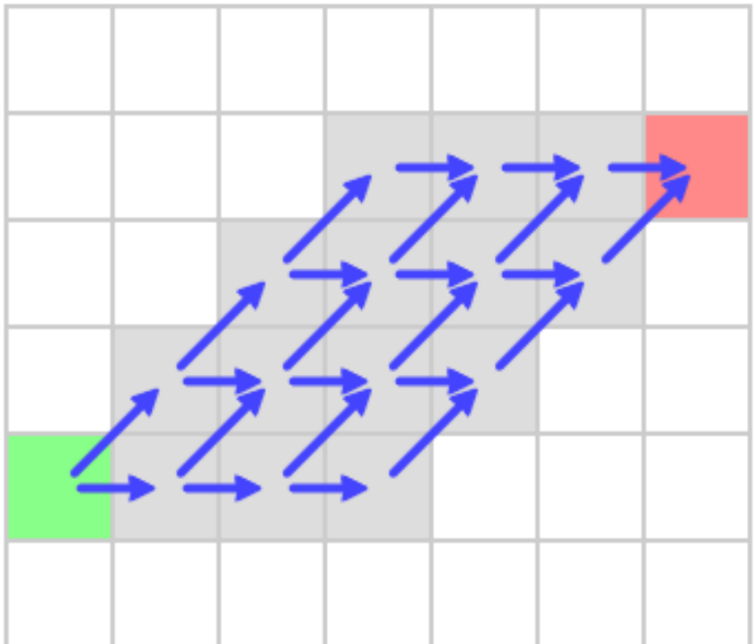
- Dijkstrův algoritmus
- A*: rozšíření Dijkstrova algoritmu
  - prohledává nejprve uzly v s lepší

$$f(v) = g(v) + h(v)$$

  - $f(v)$: cena cesty do v
  - $h(v)$: heuristický odhad ceny z v do cíle
  - přípustná heuristika: cena odhadu není výšší než reálná cena
- Jump-point search: zrychlení A* expanzí vybraných uzlů
  - pro efektivní prohledávání prostoru se čtvercovou mřížkou
  - podrobně viz `http://zerowidth.com/2013/05/05/jump-point-search-explained.html`

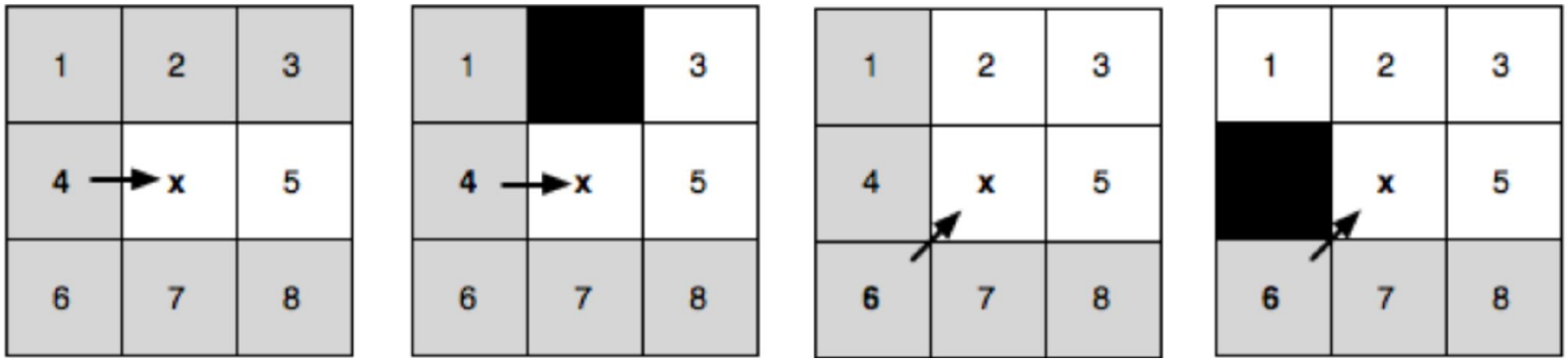
Jump-point search (JPS) jako vylepšení A*

- Existuje mnoho „symetrických" cest stejné ceny
- A* expanduje všechny sousedy aktuálního uzlu
  - pro přímé cesty: expanze příliš mnoha uzlů
- Cíl: zrychlit A* výběrem uzlů pro expanzi
- Předpoklady:
  - uniformní, 8-connected graf (ve 2D)
  - cena přesunu rovně: 1, diagonální přesun: $\sqrt{2}$

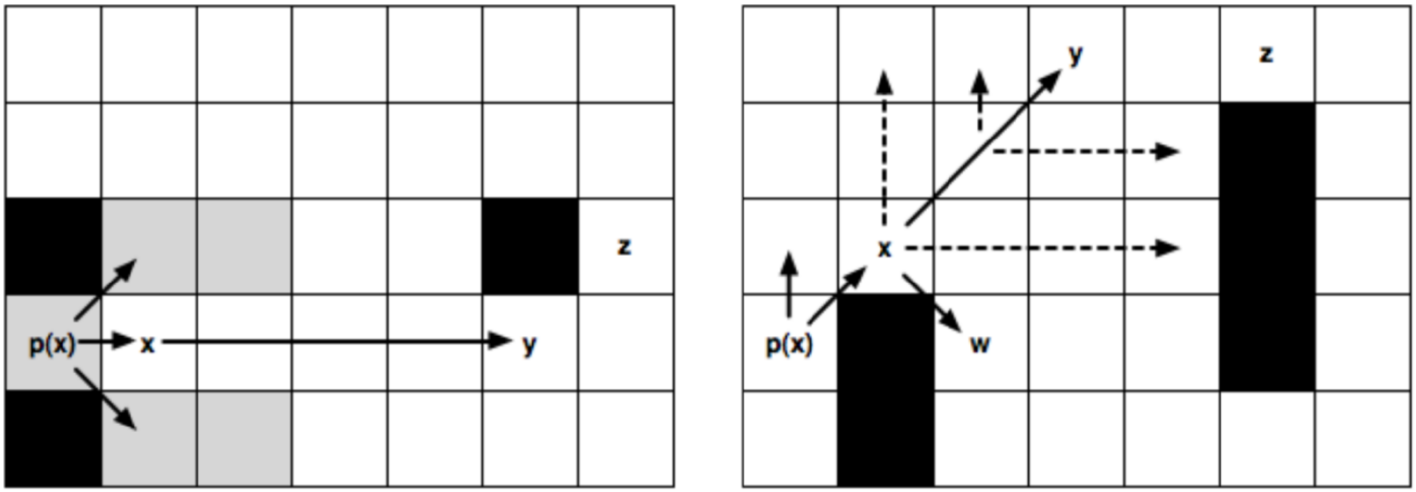


JPS: motivace a myšlenka

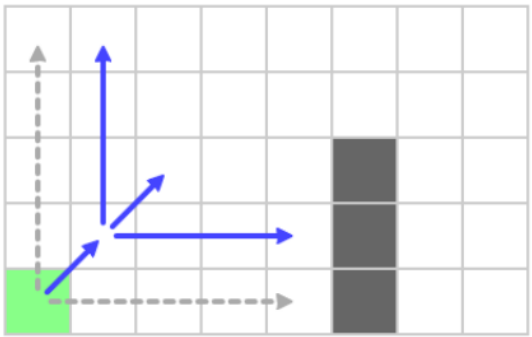
- Nechť x je aktuální uzel a p[x] jeho předchůdce
- Pak můžeme ignorovat všechny šedé uzly,
  protože mohou být optimálně dosaženy z p[x] bez cesty přes x

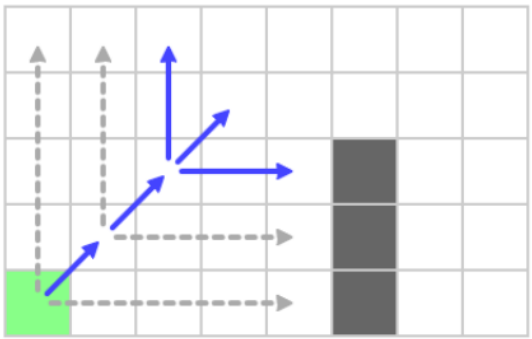


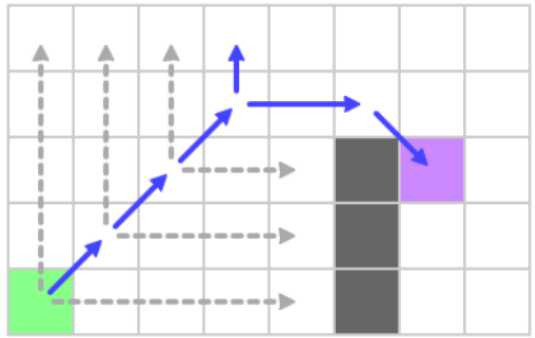
- Myšlenka:

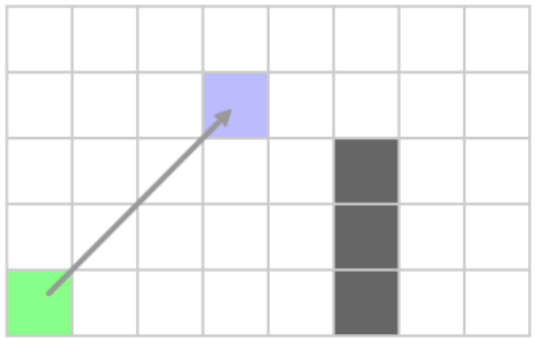


JPS: příklad









Vynucený uzel

- nelze přes něj pokračovat dál
- zastavíme "skákání", přidáme ho do množiny otevřených uzlů
- pokračujeme v další iteraci A*

Pokračování příkladu viz: `https://zerowidth.com/2013/a-visual-explanation-of-jump-point-search.html`

JPS: algoritmus (hledání následníků v A*)

function identify_successors(G, v, v_s, v_g) *(v aktuální uzel, v_s start, v_g cíl)*
  S = ∅; *(následníci)*
  N = prune(v, neighbors(v)); *(sousedé)*
  for each n ∈ N
    n = jump(G, v, direction(v, n), v_s, v_g);
    S = S ∪ {n};
  return S;

function jump(G, v, $\vec{d}$, v_s, v_g) *(v aktuální uzel, $\vec{d}$ směr, v_s start, v_g cíl)*
  n = step(v, $\vec{d}$);
  if n ∉ G **then return** ∅; *(překážka nebo venku z mapy)*
  if n = v_g **then return** n;
  if ∃n' ∈ neighbors(n) **such that** n' vynucen **then return** n;
  if $\vec{d}$ diagonální **then**
    for each i ∈ 1,2 **then** *(podívej se horizontálně/vertikálně)*
      if jump(n, $\vec{d}_i$, v_s, v_g) ≠ ∅ **then return** n;
  return jump(n, $\vec{d}$, v_s, v_g);

JPS: shrnutí

- Optimální algoritmus
- Bez preprocesingu
- Bez navýšení paměti
- Zrychlení A* prohledávání 10x nebo více

- Daniel Damir Harabor, Alban Grastien, Online graph pruning for pathfinding on grid maps. AAAI 2011.

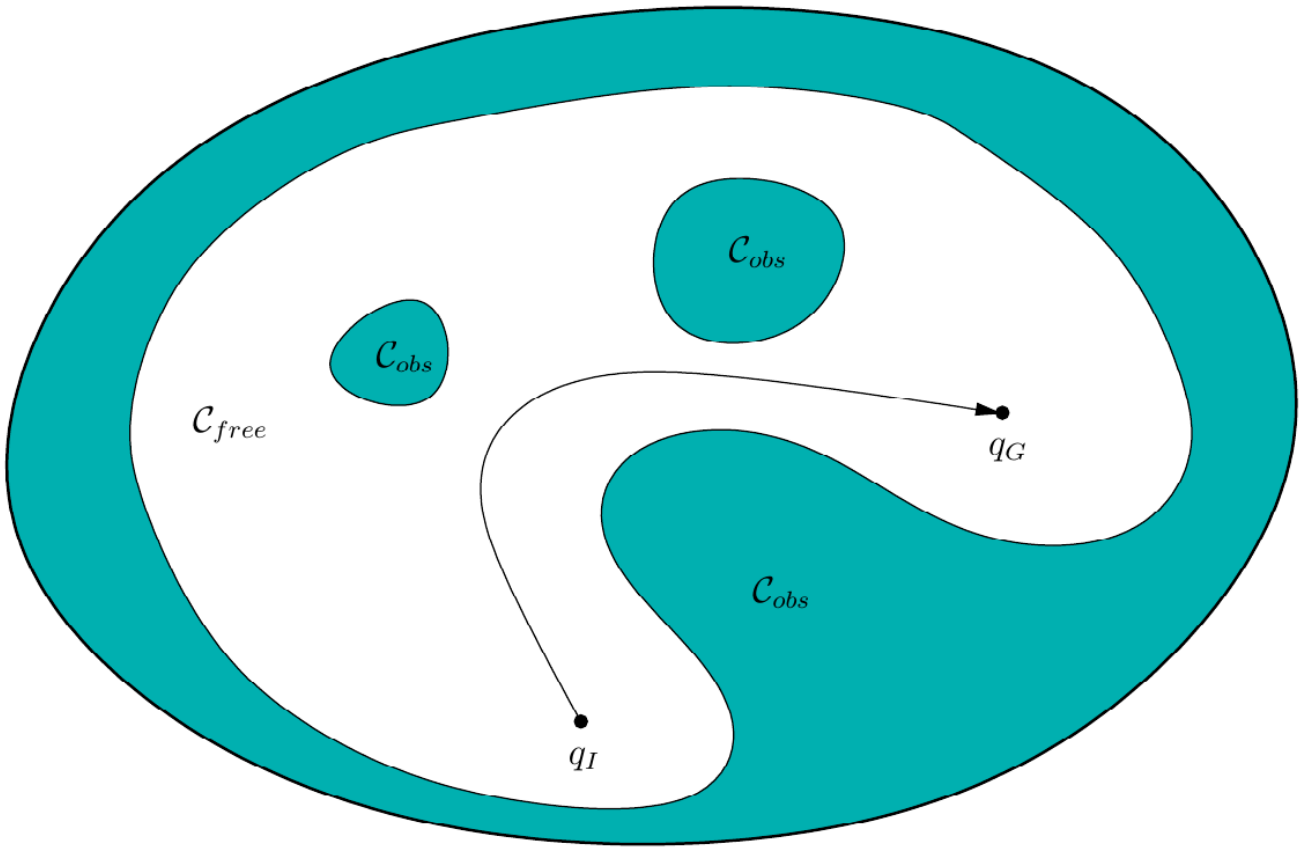
- Tutoriály o JPS

  `http://zerowidth.com/2013/05/05/jump-point-search-explained.html`

  `https://harablog.wordpress.com/2011/09/07/jump-point-search/`

Problém plánování pohybu (motion planning)

Základní problém plánování pohybu: nalezení spojité posloupnosti konfigurací, která převede robota z iniciální konfigurace q_I do cílové konfigurace q_G tak, že se žádná konfigurace po cestě nepřekrývá s překážkou.



C_{free} volný prostor

C_{obs} překážky

Plánování pohybu: postup řešení

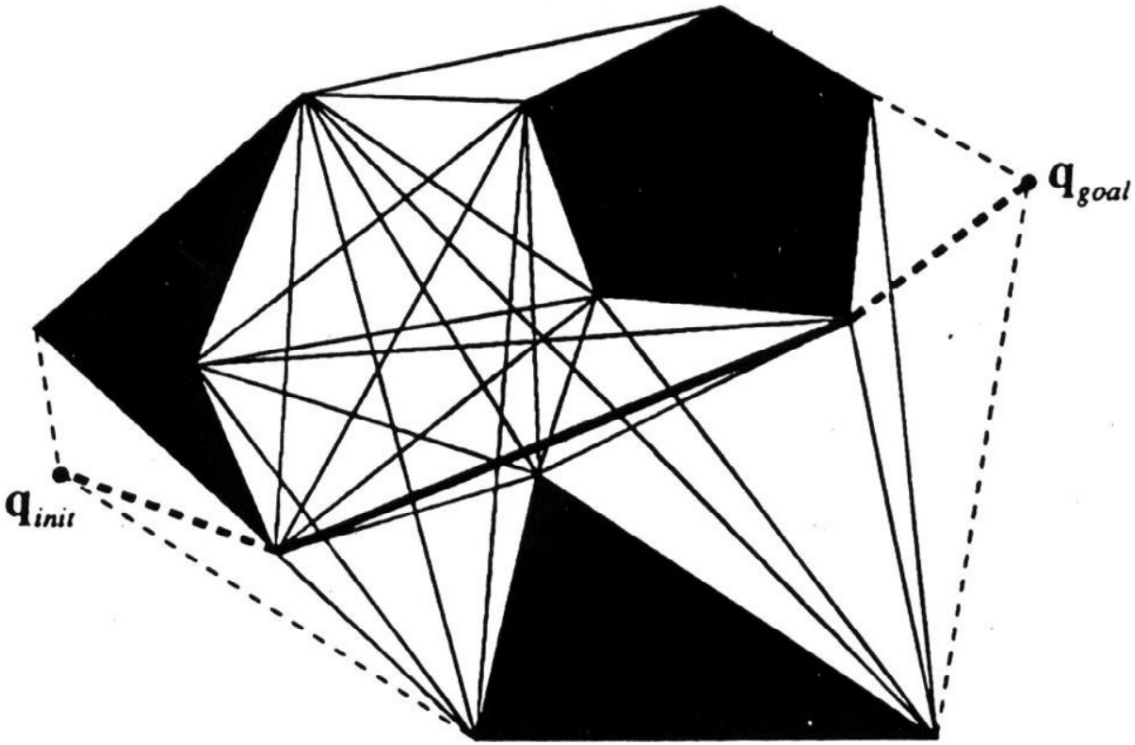
1. Předzpracování / učící fáze / konstrukce grafu
   - vypočítání grafu – často označován jako cestovní mapa (roadmap)
     - graf viditelnosti
     - Voroného diagram
     - buňková dekompozice
     - potenční funkce
2. Zpracování dotazu
   1. spojení počáteční a cílové konfigurace s grafem
   2. identifikace počáteční a cílové buňky
   3. prohledávání grafu

NEBO Inkrementální konstrukce grafu směrem k cíli

- rychle prozkoumávající náhodný strom (RRT)

Cestovní mapa jako graf viditelnosti (visibility graph)

1. Spojíme všechny páry vrcholů plus iniciální a cílové konfigurace
2. Eliminujeme hrany, které protínají překážky
3. Graf viditelnosti vždy zahrnuje nejkratší cestu konfiguračním prostorem



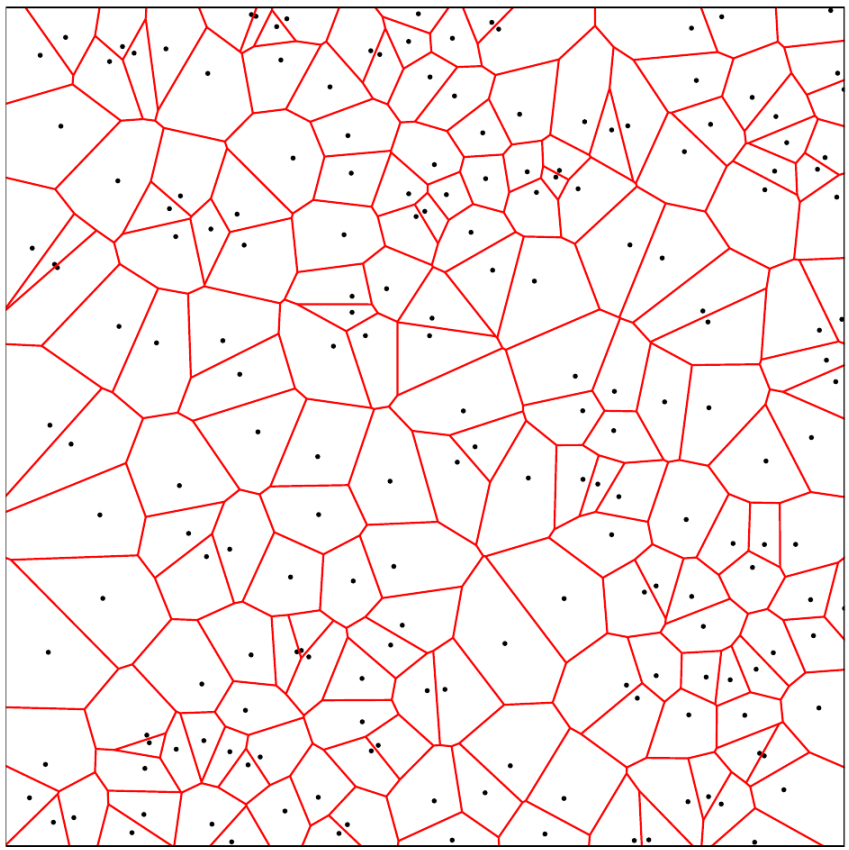
Konstrukce grafu viditelnosti

- popsaný (naivní) postup pro n vrcholů $O(n^3)$
- s pomocí rotačního stromu pro množinu segmentů $O(n^2)$

Cestovní mapa: Voroného diagram (VD)

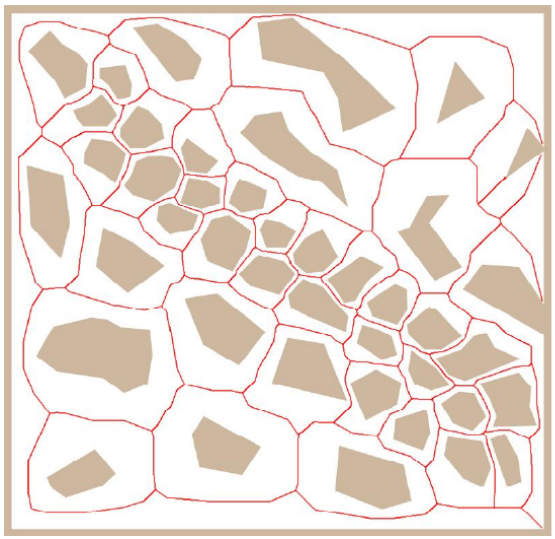
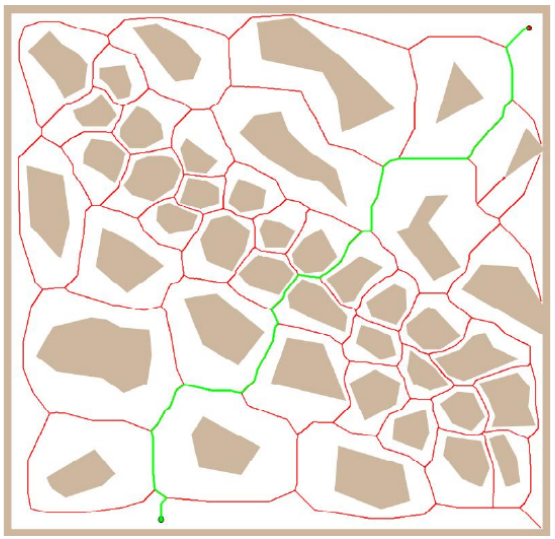
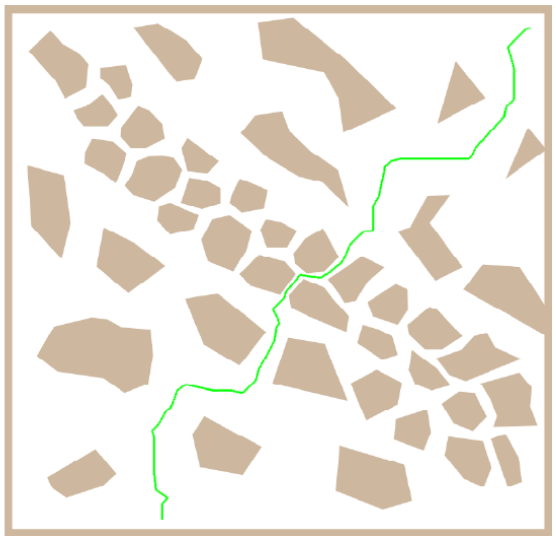
Voroného diagram

- dělí prostor $\mathbb{R}^2$ na oblasti podle zadaných objektů, např. bodů $x \in M$
- každý bod x má přiřazenu Voroného oblast $V(x)$
- pro každý bod $y \in V(x)$ je x nejbližší bod z M



Cestovní mapa jako Voroného diagram

- Cestovní mapa jako VD, který maximalizuje vzdálenost od překážek
  - body zobecněného VD s cestovní mapou jsou ekvidistantní od dvou bodů hranice C_{obs}

1. Robot změní konfiguraci na bod ve VD
   - toto lze realizovat pohybem po rovné čáře v konfiguračním prostoru
2. Robot se pohybuje ve VD, dokud nedosáhne bod, který je nejbližší cílové konfiguraci
3. Robot opustí VD a přesune se do cíle
   - opět rovnou čarou v konfiguračním prostoru

Graf viditelnosti vs. Voroného diagram

Graf viditelnosti

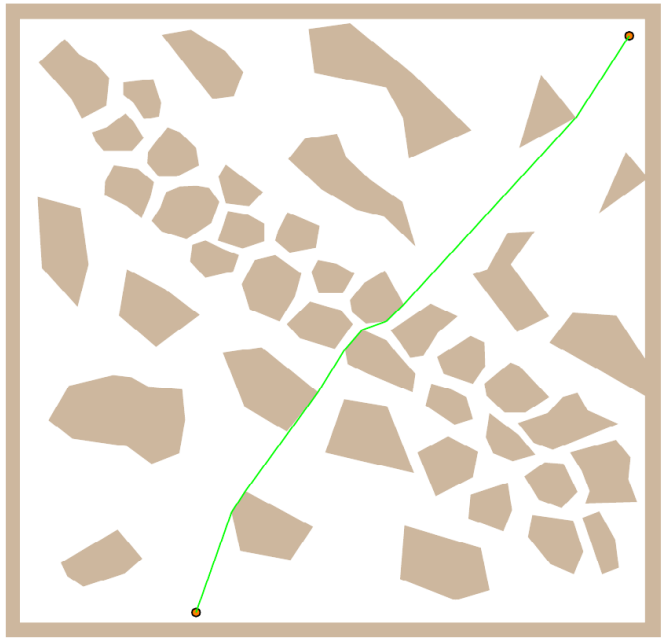
- nejkratší cesta, ale blízko k překážkám

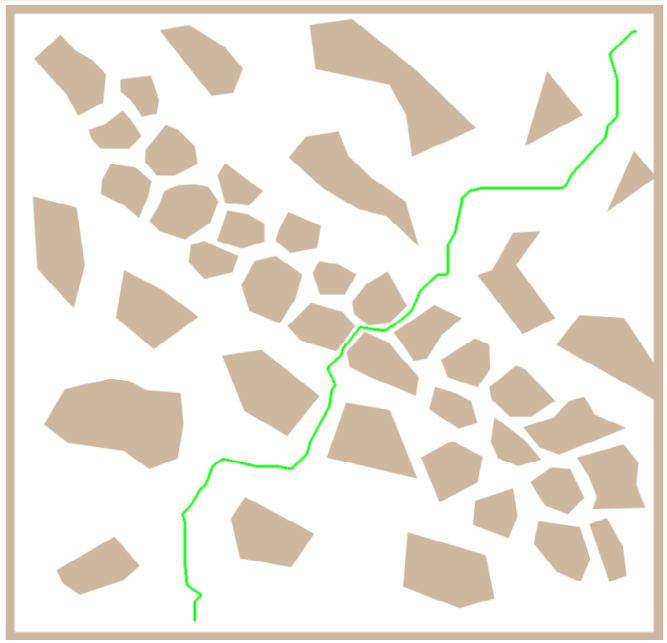
  nutno uvažovat bezpečnost cesty
- komplikované ve vyšší dimenzi

Voroného diagram

- maximalizuje vzdálenost od překážek, což poskytuje konzervativní cesty
- malé změny překážek mohou vést k velkým změnám VD
- komplikované ve vyšší dimenzi

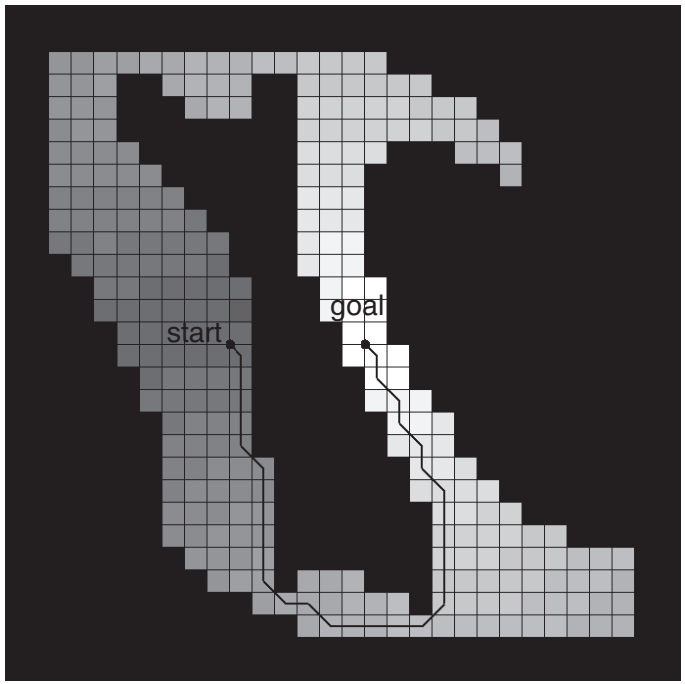
Pro vyšší dimenze se používají jiné typy cestovních map

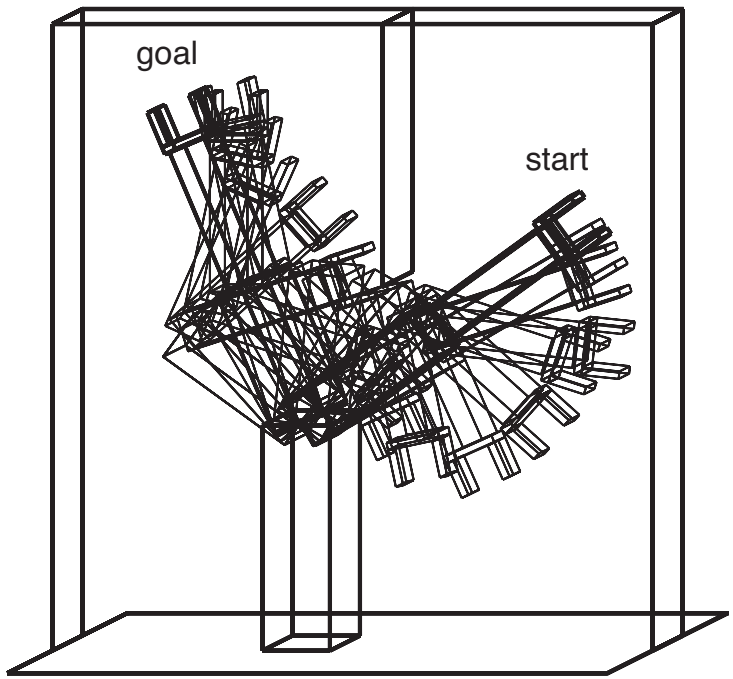




Buňková dekompozice (cell decomposition)

- Dekompozice volného prostoru na konečný počet spojitých regionů nazvaných buňky
- Předpoklad: problém plánování cesty v jedné buňce je jednoduchý
  - např. posun podél rovné čáry
- Základní buňková dekompozice: pravidelná mřížka
  - úroveň šedi buňky určuje hodnotu buňky = cenu nejkratší cesty do cíle
  - hodnoty spočítáme deterministickou verzí algoritmu iterace hodnot (viz přednášky Nejistota) nebo (upraveným) A* algoritmem

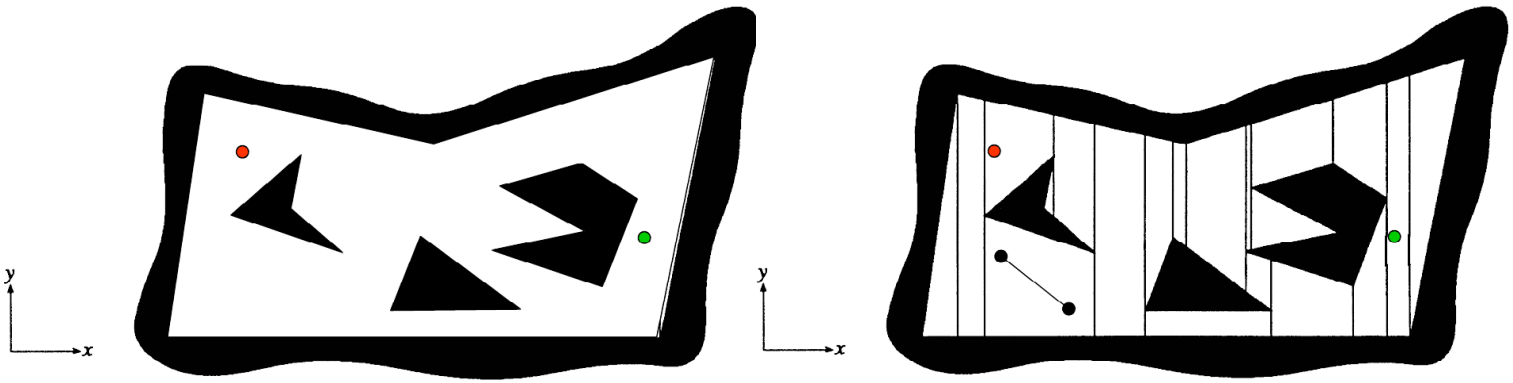




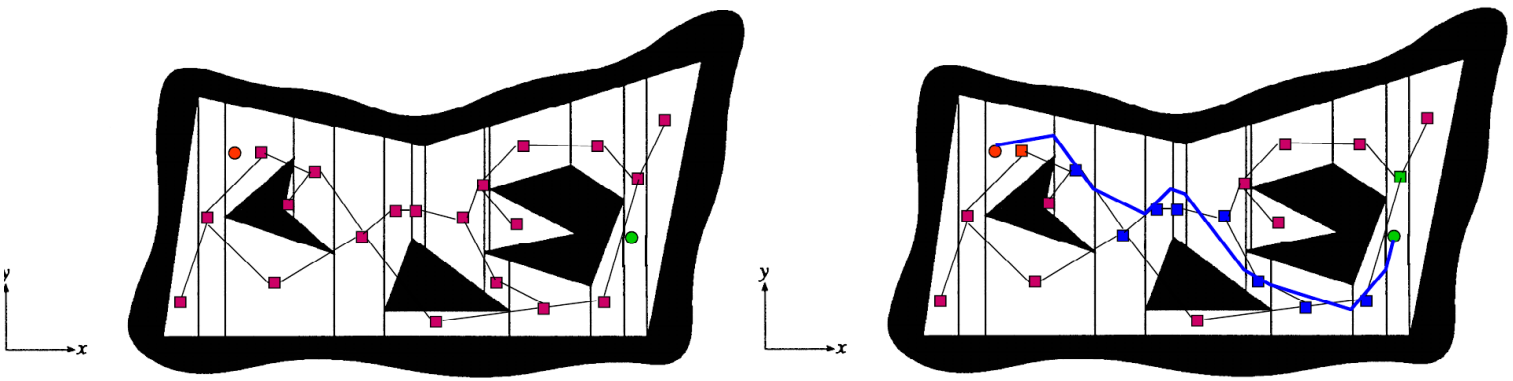
Vertikální dekompozice (vertical decomposition)

Exaktní buňková dekompozice (př. vertikální/lichoběžníková, triangulace):

- volný prostor C_{free} reprezentován kolekcí nepřekrývajících se buněk, jejichž spojení **reprezentuje přesně** C_{free}



1. Rozdělíme prostor vertikálně dle překážek



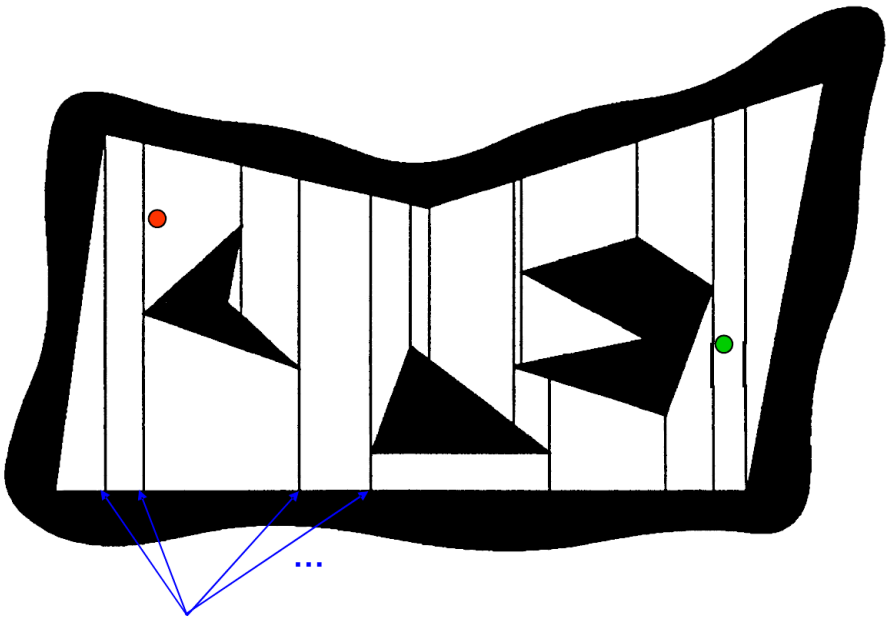
2. Centroid reprezentuje buňku
3. Spojíme hranami centroidy sousedních buněk $\Rightarrow$ graf
4. Hledáme cestu v grafu

Konstrukce vertikální dekompozice

Jak prostor vertikálně rozdělit dle překážek?

- plane sweep/sweep line algoritmus z výpočetní geometrie

  přímka/rovina prochází prostorem přes body,
  kde dochází ke kritické změně informace

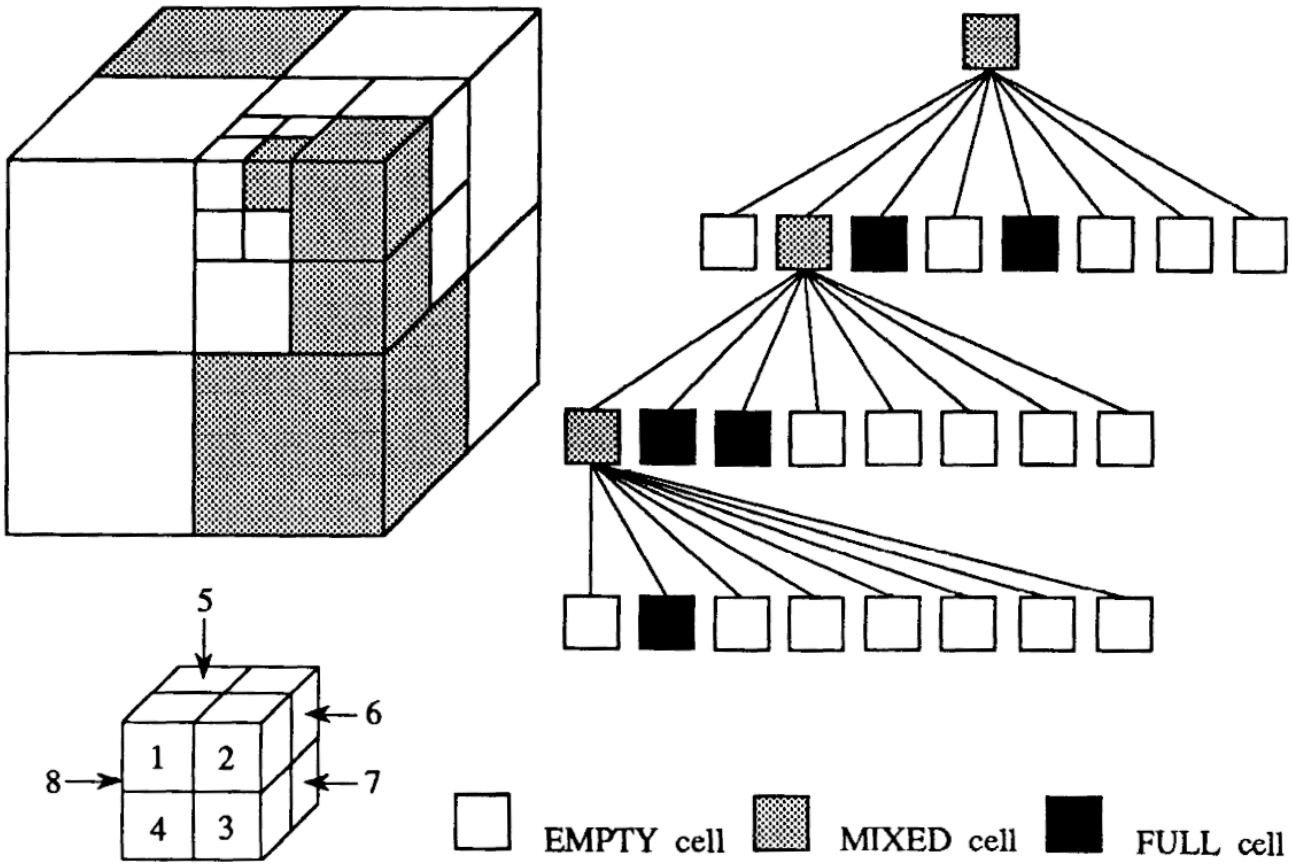


kritické změny

Oktalová dekompozice (octree decomposition)

Metoda přibližné buňkové dekompozice

- volný prostor C_{free} reprezentován kolekcí nepřekrývajících se buněk, jejichž spojení **je obsaženo v** C_{free}
- příklady: quadtree, 2^n-tree, oktalová (octree) dekompozice, pravidelná mřížka

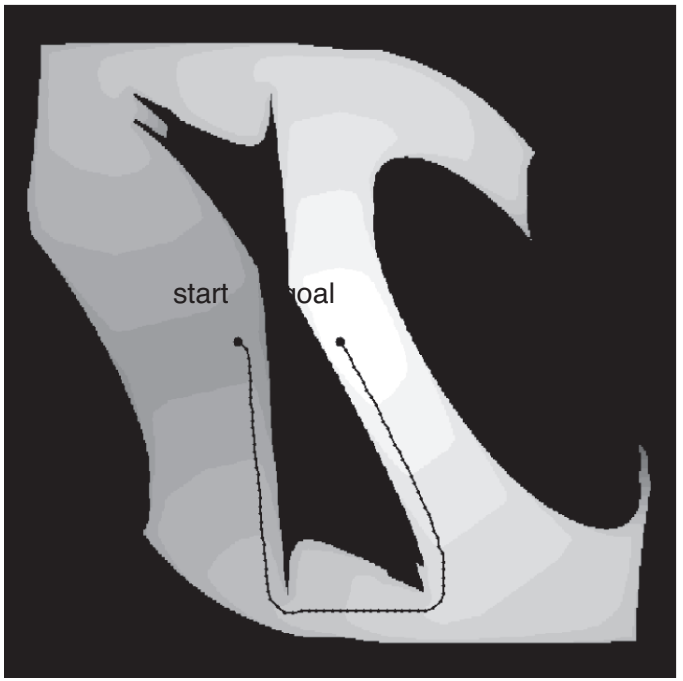


Buňková dekompozice: náčrt algoritmu

1. Spočítej buňkovou dekompozici až k danému rozlišení
2. Idenfikuj startovní a cílovou buňku
3. Prohledávej posloupnost prázdných a smíšených (mixed) buněk mezi startem a cílem
4. Pokud posloupnost neexistuje, vrať „cesta neexistuje"
5. Pokud existuje posloupnost prázdných buněk, vrať „řešení"
6. Pokud dosažena hranice rozlišení
   1. Dekomponuj dále smíšené buňky
   2. Vrať se do bodu 2.

Potenční funkce (potential field/navigation function)

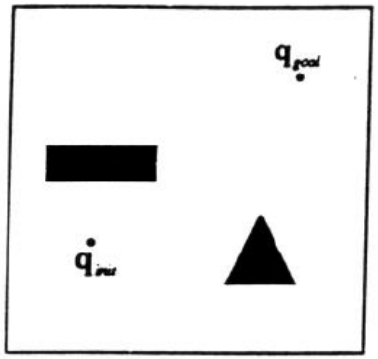
- Prvotní použití: cesta by neměla vést těsně vedle překážky
  - podobně jako neprojedeme autem, pokud máme přesně tolik místa jako je šířka auta
- Řešení: zavedeme dodatečnou cenovou funkci, tzv. potenční funkce
  - hodnota funkce narůstá ze vzdáleností od nejbližší překážky
  - potenční funkce je pak využita jako dodatečná cenová funkce při výpočtu nejkratší vzdálenosti

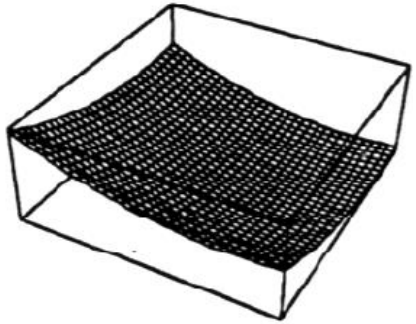


Potenční funkce

Mnoho dalších prací zobecňuje využití potenční funkce pro prohledávání celého prostoru

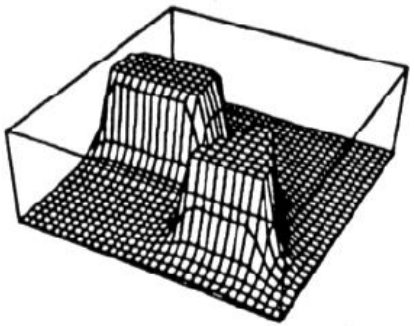
Latombe 1991

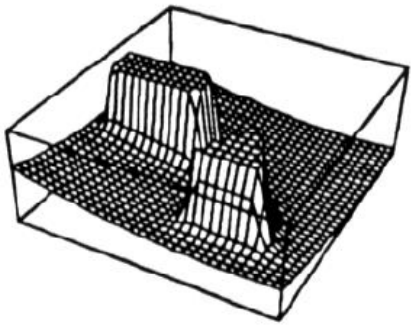




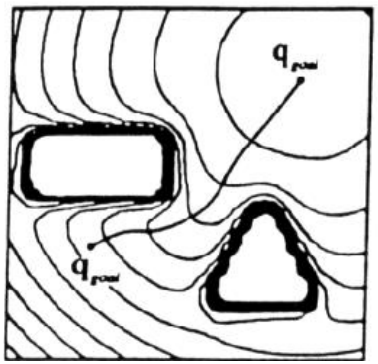
atraktivní potenční pole pro cíl

odpuzující potenční pole pro překážky

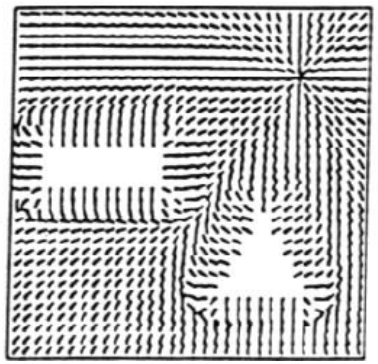




atraktivní + odpuzující potenční pole



equi-potenciál pro kontury

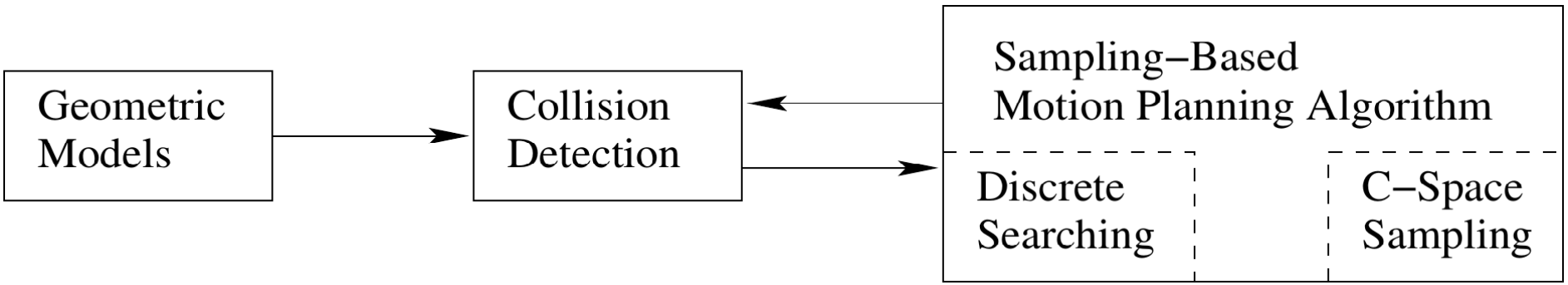


vynucená pole

Obsah: jak pokračujeme

Plánování pohybu založené na vzorkování (sampling-based)

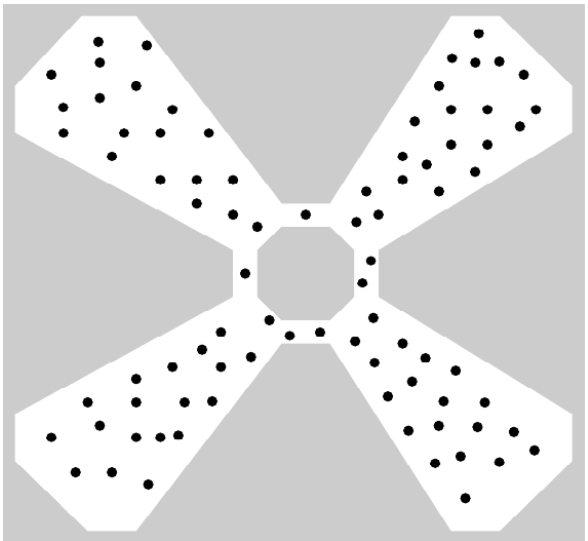
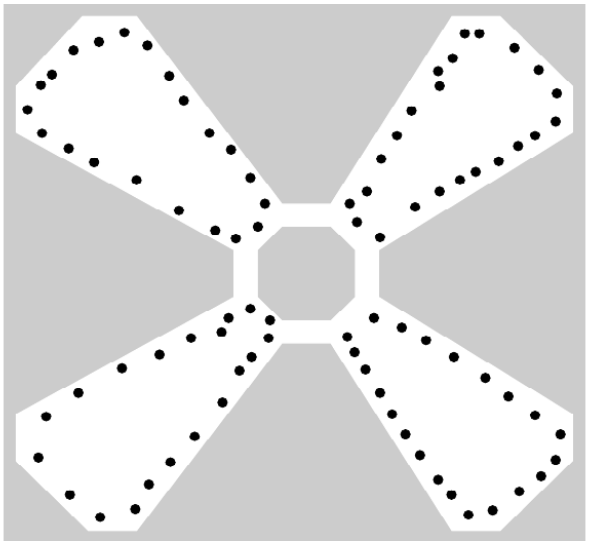
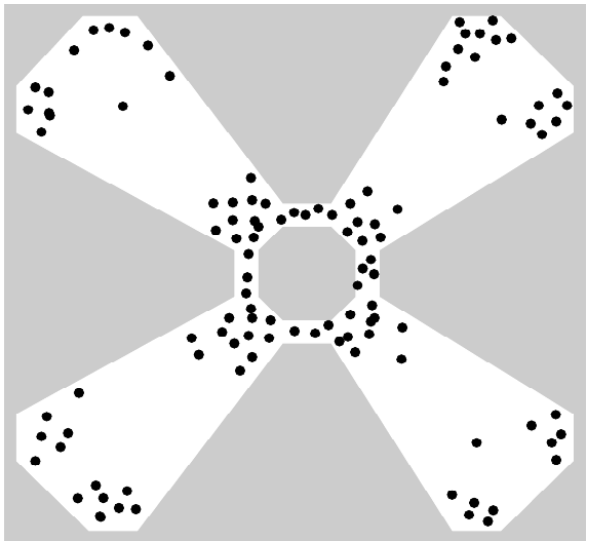
- Vyhýbá se explicitní reprezentaci překážek v konfiguračním prostoru
  - black-box funkce použita pro vyhodnocení, zda je konfigurace bez kolizí,
    založeno např. na geometrických modelech a testování kolizí modelů
- Vytvoří diskrétní reprezentaci C_{free}
- Konfigurace v C_{free} jsou náhodně vzorkovány a spojeny do pravděpodobnostní cestovní mapy (probabilistic roadmap)



- Není úplné jako doposud uvažované metody, ale jedná se o pravděpodobnostní úplnost
  - pravděpodobnostně úplný algoritmus: se vzrůstajícím počtem vzorků nalezene přípustné řešení (pokud existuje)

Náhodné vzorkování

- Mohou být použity různé vzorkovací strategie (distribuce)

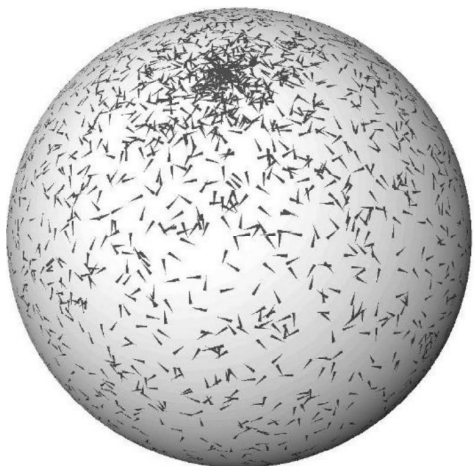




- Hlavní problém přístupů založených na náhodném vzorkování spočívá v zahrnutí úzkých pasáží
- Různé modifikace vzorkovacích strategií navrženy v poslední dekádě, studuje je teorie vzorkování

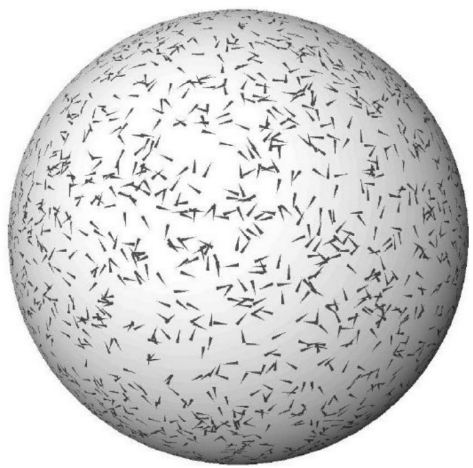
Náhodné vzorkování (dokončení)

- Řešení může být nalezeno s použitím pouze pár vzorků
- Vzorkovací strategie jsou důležité
  - blízké překážky
  - úzké pasáže
  - mřížkově založené
  - uniformní strategie musí být pečlivě zváženy

Kuffner, Effective sampling and distance metrics for 3D rigid body path planning. ICRA, 2004.



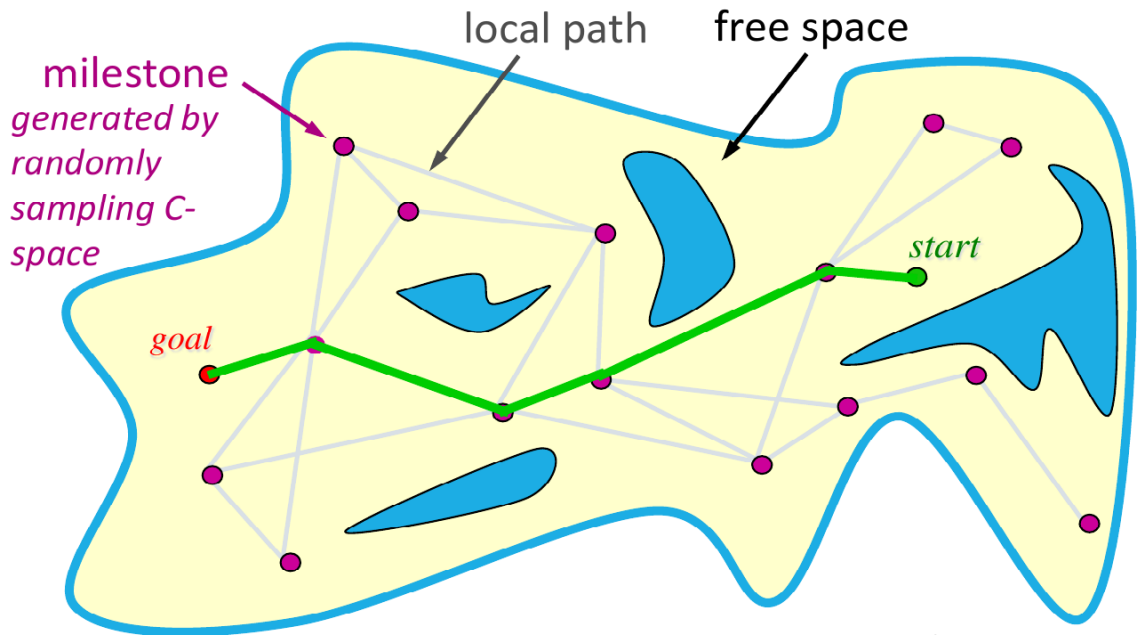
naivní vzorkování



uniformní vzorkování
s použitím Eulerových úhlů

Pravděpodobnostní cestovní mapa – probabilistic roadmap

- Diskrétní reprezentace spojitého konfiguračního prostoru generována náhodným vzorkováním konfigurací v C_{free}, které spojíme do grafu
  - vrcholy: přípustné konfigurace robota (milestones)
  - hrany: reprezentují platnou cestu (trajektorii) mezi konfiguracemi
- Nevyžaduje explicitní popis C_{obs}
- Cesta (trajektorie) ze startu do cíle: grafové prohledávací algoritmy



Jeden vs. více dotazů (single-query vs. multi-query)

Více dotazů na pravděpodobnostní cestovní mapu

- generování jedné cestovní mapy, která je používána pro plánovací dotazy vícekrát
- Reprezentativní technika Probabilistic RoadMap (PRM)

  Probabilistic roadmaps for path planning in high dimensional configuration spaces, Kavraki, Svestka, Latombe, Overmars. IEEE Transactions on Robotics and Automation, 12(4):566–590, 1996.

  . . . první plánovač, který ukázal, že je schopen řešit problém ve 4-5 dimenzích

Jeden dotaz na pravděpodobnostní cestovní mapu

- pro každý plánovací problém konstruována nová cestovní mapa charakteristická pro podprostor konfiguračního prostoru relevantní pro problém
  - rychle prozkoumávající náhodný strom (RRT) LaValle, 1998

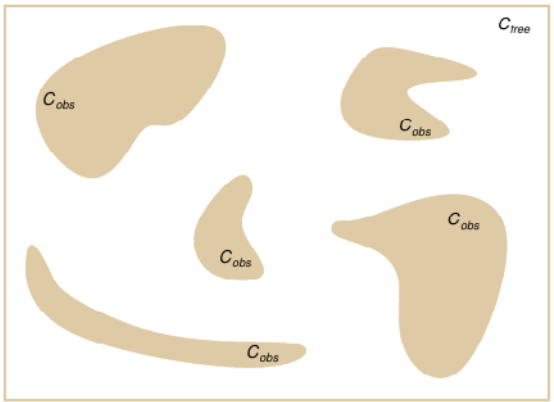
Strategie pro více dotazů

Vytvoření cestovní mapy (grafu), který reprezentuje prostředí

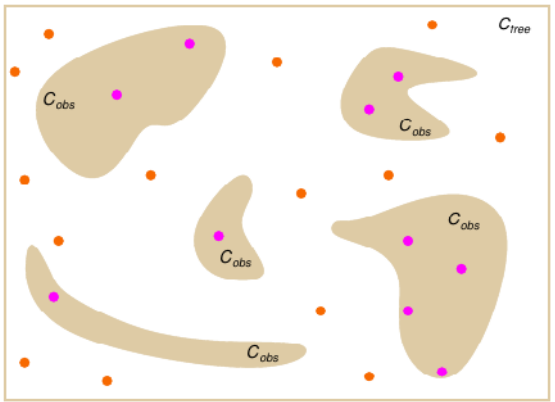
1. Učící fáze
   1. Vzorkování n bodů v C_{free}
   2. Spojení náhodných konfigurací s použitím lokálního plánovače (lokální plánovač hledá lokální cesty, např. přímé)
2. Dotazovací fáze
   1. Spojení počáteční a cílové konfigurace s PRM (např. lokálním plánovačem)
   2. Použití prohledávání grafu k nalezení cesty

Konstrukce PRM

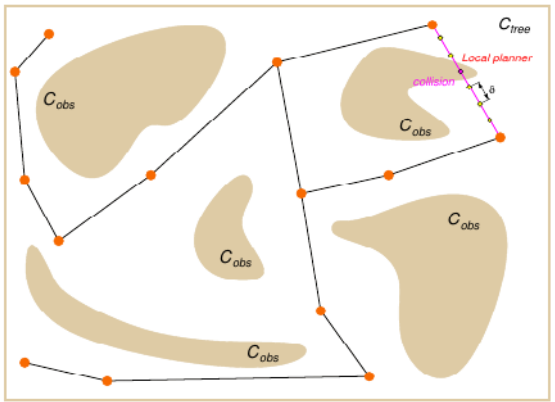
#1 Given problem domain



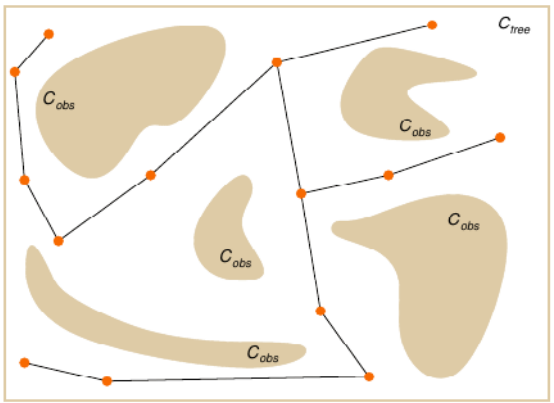
#2 Random configuration



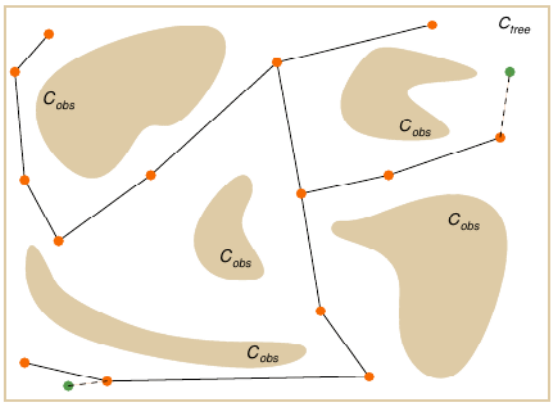
#3 Connecting samples



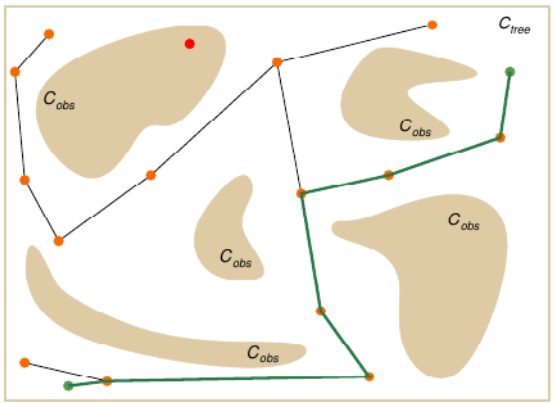
#4 Connected roadmap



#5 Query configurations



#6 Final found path



sPRM – zjednodušený (simplified) PRM

---

function sPRM(q_{init}, počet vzorků n, poloměr ρ)

$V = \{q_{init}\} \cup \{\text{SampleFree}_i\}_{i=1,\ldots,n-1}$; $E = \emptyset$;

foreach $v \in V$ **do**

$U = \text{Near}((V, E), v, \rho) \backslash \{v\}$; *(vrcholy v radiusu ρ od v)*

foreach $u \in U$ **do**

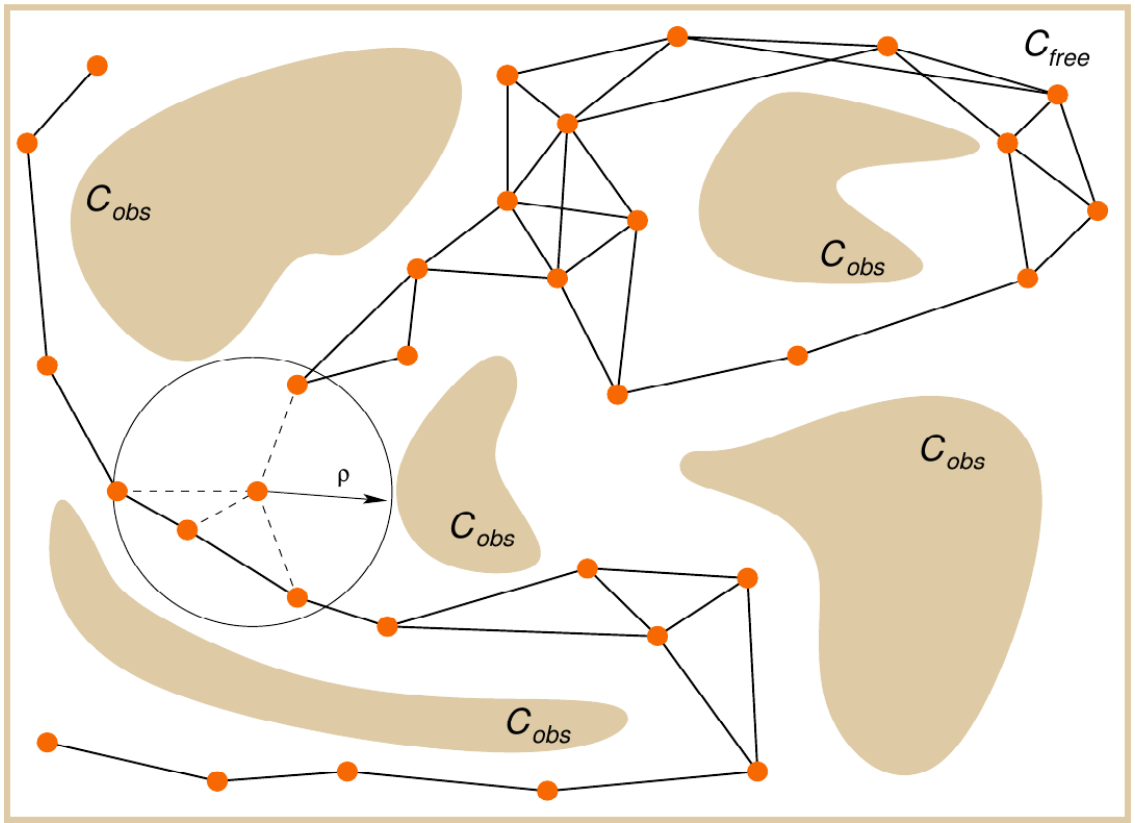
if CollisionFree(v, u) **then**

$E = E \cup \{(v, u), (u, v)\}$;

return $G = (V, E)$;

Praktická PRM

- Inkrementální konstrukce
- Spojení uzlů v radiusu ρ
- Lokální plánovač testuje kolize vzhledem k danému rozlišení δ
- Cesty mohou být hledány Dijkstrovým algoritmem



PRM vs. sPRM

function sPRM(q_{init}, počet vzorků n, poloměr ρ)
  $V = \{q_{init}\} \cup \{\text{SampleFree}_i\}_{i=1,\ldots,n-1}$; $E = \emptyset$;
  foreach $v \in V$ **do**
    $U = \text{Near}((V, E), v, \rho) \backslash \{v\})$; *(vrcholy v radiusu ρ od v)*
    foreach $u \in U$ **do**
      if CollisionFree(v, u) **then**
        $E = E \cup \{(v, u), (u, v)\}$;
return $G = (V, E)$;

---

function PRM(q_{init}, počet vzorků n, poloměr ρ)
  $V = \emptyset$; $E = \emptyset$;
  for $i = 0, \ldots, n$ **do**
    $q_{rand} = \text{SampleFree}$; *(vzorek z C_{free})*
    $U = \text{Near}((V, E), q_{rand}, \rho)$; *(vrcholy v radiusu ρ of q_{rand})*
    $V = V \cup \{q_{rand}\}$;
    foreach $u \in U$ se vzrůstající $|u - q_{rand}|$ **do**
      if q_{rand} a u nejsou ve stejné spojené komponentě (V, E) **then**
        if CollisionFree(q_{rand}, u) **then**
          $E = E \cup \{(q_{rand}, u), (u, q_{rand})\}$;
return $G = (V, E)$;

PRM vlastnosti

- Úplnost pro standardní PRM nediskutována
- Většinou studována sPRM, která je
  - pravděpodobnostně úplná a asymptoticky optimální
  - složitost konstrukce grafu (učící fáze) $O(n^2)$
  - složitost dotazu $O(n^2)$
  - prostorová složitost $O(n^2)$
- Často prakticky používány heuristiky pro funkci Near
  - k-nearest (k-nejbližších) sousedů od v
  - variable connection radius (proměnlivý radius) ρ jako funkce n
  - k-nearest sPRM, variable radius sPRM nejsou pravděpodobnostně úplné

Rychle prozkoumávající náhodný strom – Rapidly exploring random tree (RRT)

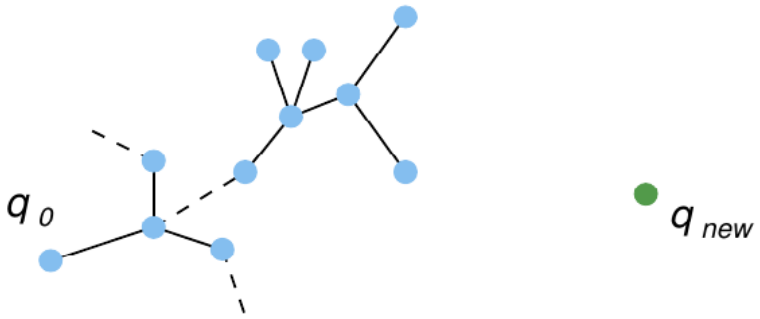
- Algoritmus pro jeden dotaz
- Motivací je hledání cesty založené na řízení
- Inkrementální konstrukce grafu (stromu) směrem k cílové oblasti
  - negarantuje přesnou cestu k cílové konfiguraci

Postup tvorby stromu

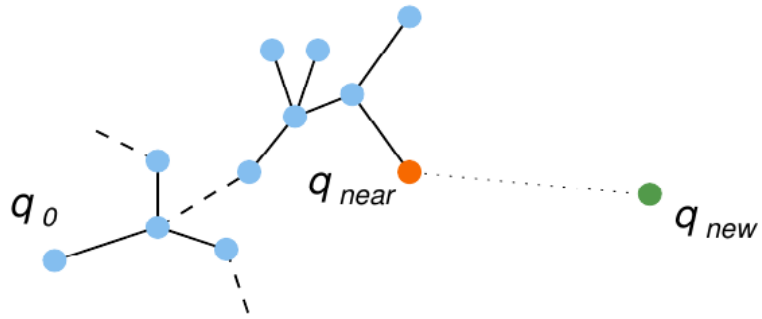
1. Začni s počáteční konfigurací q_{init}, která je kořenem konstruovaného stromu
2. Generuj náhodnou konfiguraci q_{new} v C_{free}
3. Nalezni nejbližší uzel q_{near} ke q_{new} ve stromě
   - např. použitím implementace k-dimensionálních stromů jako je ANN nebo FLANN knihovna
4. Rozšiř q_{near} směrem ke q_{new}
   - strom rozšiřujeme v malých krocích, ale často s přímou kontrolou výběru vrcholu tak, abychom posunuli robota na nejbližší pozici ke q_{new}
5. Pokud není strom v dostatečné vzdálenosti od cílové konfigurace, běž na krok 2.

Konstrukce RRT

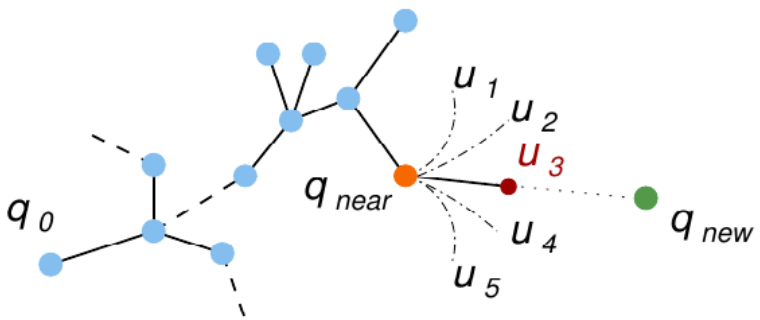
#1 new random configuration



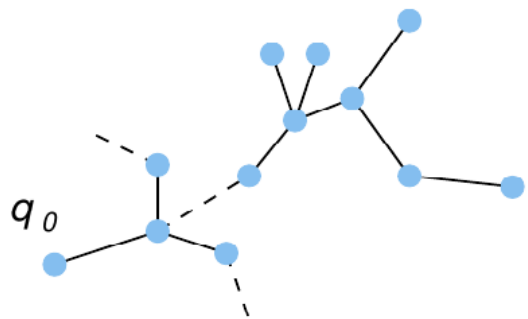
#2 the closest node



#3 possible actions from q_{near}



#4 extended tree



RRT algoritmus

---

function RRT(q_{init}, počet vzorků n)
returns cestovní mapa $G = (V, E)$

$V = \{q_{init}\}$;
$E = \emptyset$;
for $i = 1, \ldots, n$ **do**
  q_{new} = SampleFree; *(bezkolizní náhodná konfigurace)*
  q_{near} = Nearest($(V, E), q_{new}$); *(nejbližší vrchol ve V of q_{new})*
  $N_{q_{near}}$ = množina náhodně získaných vzorků blízko q_{near};
  u = bezkolizní vzorek z $N_{q_{near}}$ blízko q_{new};
  $V = V \cup \{u\}$;
  $E = E \cup \{(q_{near}, u)\}$;
return $G = (V, E)$;

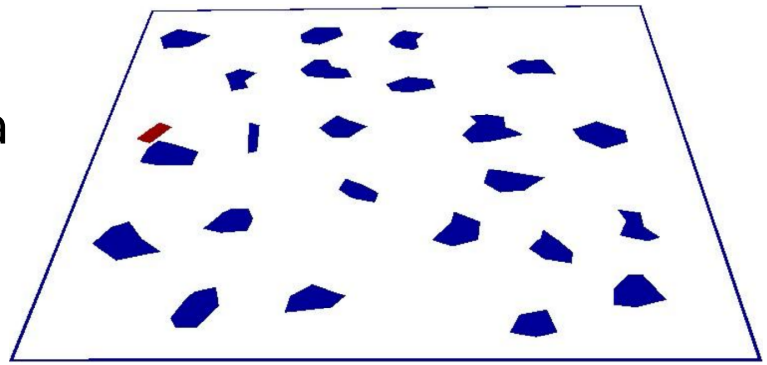
Vyvinut S. LaValle a spolupracovníky, 2001-2003
LaValle, Planning algorithms. Cambridge University Press, 2006.
http://planning.cs.uiuc.edu/

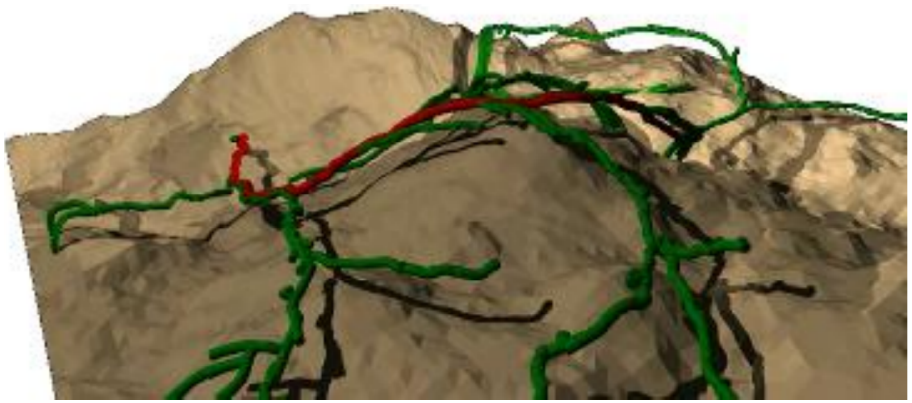
Vlastnosti RRT algoritmu

- Rychle prozkoumává prostor
  - q_{new} bude pravděpodobně generováno ve velké dosud neprozkoumané části
- Umožňuje uvažování kinodynamických/dynamických omezení (během expanze)
- Umí nalézt trajektorie nebo posloupnosti přímou kontrolou příkazů robotových ovladačů
- Test na detekci kolize obvykle jako black-box
  - např. RAPID, Bullet knihovny
- Podobně jako PRM má RRT problémy s úzkými pasážemi
- RRT nalezne proveditelnou cestu bez garance kvality
  - cesta může být relativně daleko od optimální cesty dané např. délkou cesty
  - navzdory tomu je úspěšně používán v mnoha praktických aplikacích
- Navrženo mnoho variant RRT

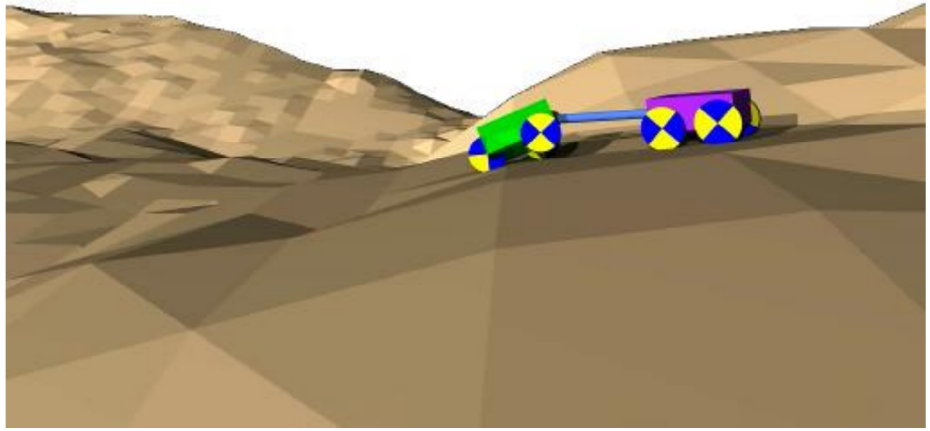
RRT: příklady

Plánování pro robota typu auta





Plánování na 3D povrchu



Plánování s dynamikou

V. Vonásek & P. Vaněk

Přístupy k plánování pohybu: shrnutí

Metody pro více dotazů

- Graf viditelnosti
- Voroného diagram
- Buňková dekompozice
  - exaktní metody: vertikální dekompozice
  - přibližné metody: pravidelná mřížka, oktalová dekompozice
- Potenční funkce
- Pravděpodobnostní mapy: algoritmy sPRM, PRM

Metody pro jeden dotaz

- Rychle prozkoumávající náhodný strom (RRT)

Úplnost plánovačů pohybu

- Plánovač pohybu je úplný,
  pokud nalezne bezkolizní cestu vždy, když existuje a jinak selže.
  - graf viditelnosti
  - Voroného diagram
  - přesná buňková dekompozice
  - potenční funkce

- Pravděpodobnostně úplný plánovač pohybu
  vrací cestu s vysokou pravděpodobností, pokud existuje,
  nemusí však skončit, pokud cesta neexistuje.
  - sPRM

- Rozlišením úplný (resolution complete) plánovač pohybu
  diskretizuje prostor a vrací cestu, pokud existuje v této reprezentaci.
  - přibližná buňková dekompozice

Porovnání plánovačů pohybu

	Přesná buňk. dekomp.	Přibližná buňk. dekomp.	G.viditel. VD	Pravděp. metody	Potenční funkce
praktické pro 2-3 dimenze	velmi pomalé paměť	pomalé, paměť m^n požadavek[1]	velmi pomalé	rychlé	rychlé
praktické nad 5 dimenzí	ne	snad ne nejasné	ne	ano	ano
optimální	ano	ano, po rozlišení	ano	pravděp. garance	ne
snadná implementace	ne	ano	ne	ano	ano
úplné	ano	ano, po rozlišení	ano	pravděp. úplné	ne[2]

[1] m buněk mřížky každá s n osami

[2] navigační funkce pro některé třídy problémů poskytují teoretické záruky

Pohyb: kinematický vs. dynamický stav

Zatím jsme pohyb pouze plánovali avšak nerealizovali

- Předpokládali jsme, že robot je schopen následovat navrženou cestu
- V reálu však není pravda, je třeba počítat se setrvačností robota
- Roboti spíš vyvíjejí síly než určují pozice
- Jak počítat tyto síly?

Popsali jsme dynamický stav (kromě kinematiky zahrnuje rychlost změny)

- přechodový model pro dynamický stav zahrnuje účinek sil na rychlost změny
- nutné modely vyjádřené diferenciálními rovnicemi, které provážou kvantitu (např. kinematický stav) s odpovídající změnou v čase (např. rychlost)
- pokud bychom uměli generovat takové plány, roboti by měli skvělý výkon
- dynamický stav má však více dimenzí než kinematický, plánovací algoritmy by byly použitelné pouze pro nejjednodušší roboty ☹

$\Rightarrow$ robotické systémy v praxi se spoléhají na jednodušší kinematické plánovače cest

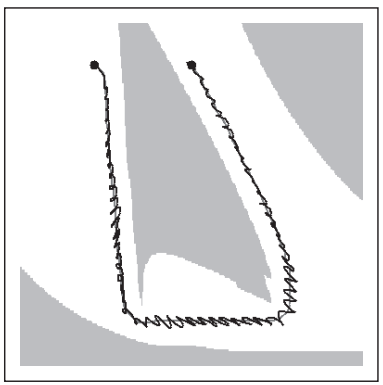
Regulátor

Regulátor kompenzuje limitace kinematických plánů, aby byla zachována cesta

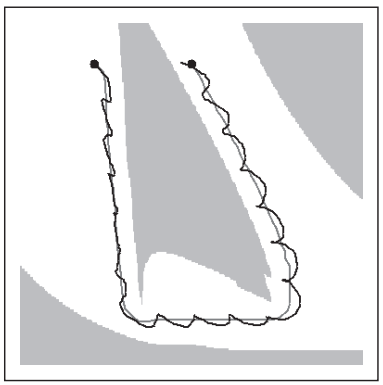
Referenční (reference) regulátor:

- snaží se robota udržet na naplánované cestě (referenční cestě)

Cesta robota, který se snaží sledovat kinematickou cestu:



$K_P = 1$



$K_P = 0.1$

- jakmile nastane odchylka (díky šumu nebo omezením sil, které může robot uplatnit), robot vyvine opačnou sílu – vede k vibracím, které jsou důsledkem setrvačnosti
- P regulátor (P proporční): řízení je proporční chybě manipulace robota
- řízení a_t generované P regulátorem má tvar $a_t = K_P(y(t) - x_t)$
  - $y(t)$ referenční cesta parametrizovaná časem
  - x_t je stav robota v čase t
  - K_P parametr zisku ovlivňující, jak silně změnit odchylku mezi x_t a $y(t)$
- pomůže nižší hodnota $K_P(0.1)$ zlepšit chování? Ne, pouze vibrace zpomalí

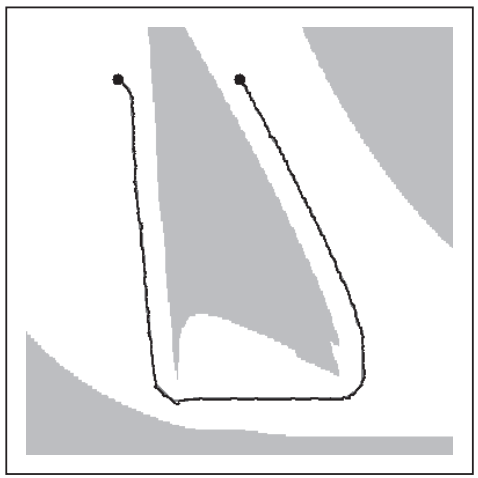
PD regulátor

P proporční člen, D derivační člen

- nejjednodušší regulátor, který zachovává striktní stabilitu v naší doméně

$$a_t = K_P(y(t) - x_t) + K_D \frac{\partial(y(t) - x_t)}{\partial t}$$

- druhý člen součtu je proporční první derivaci chyby $y(t) - x_t$ v čase t
- pokud se chyba $y(t) - x_t$ výrazně mění v čase
  $\Rightarrow$ derivace této chyby
  bude neutralizovat první člen
- pokud bude chyba přetrvávat
  a nebude se měnit:
  $\Rightarrow$ derivace zmizí a
  první člen bude dominovat



$K_P = 0.3, K_D = 0.8$

PID regulátor

I integrační člen, P proporční člen, D derivační člen

- umožňuje odstranit chybové chování PD regulátoru v některých případech
- chybové chování jako důsledek systematické vnější síly, která není součástí modelu
  - např. opotřebení robotické paže

$$a_t = K_P(y(t) - x_t) + K_I \int (y(t) - x_t)dt + K_D \frac{\partial(y(t) - x_t)}{\partial t}$$

- dlouhotrvající odchylky mezi referenčním a aktuálním stavem opraveny
  - regulátor tak řeší systematické chyby na úkor zvýšeného nebezpečí oscilací

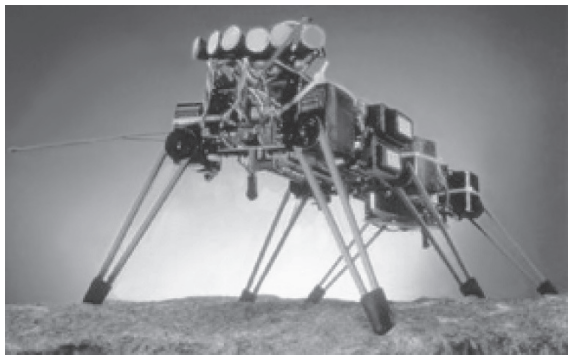
Další způsoby řízení pohybu: řízení pomocí potenční funkce

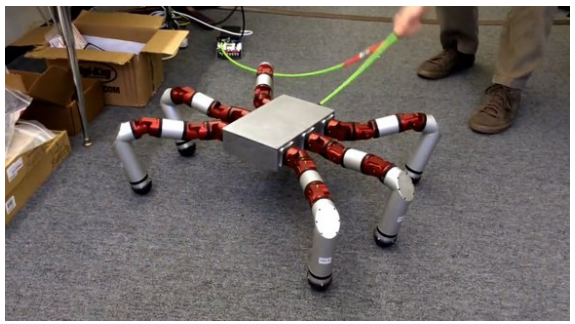
Řízení pomocí potenční funkce

- podobně jako při plánování pohybu

Řízení pomocí refenční cesty a potenční funkce nemusí:

- být postačující v komplexním a vzdáleném prostředí (Mars)
- mít dostatečnou přesnost
- př. hexapod kráčející na hrbolatém terénu
  - nemá postačující senzory pro získání modelu terénu vhodného pro plánování a
  - 12 DOFs (2 pro každou nohu) jsou příliš mnoho pro efektivní plánování

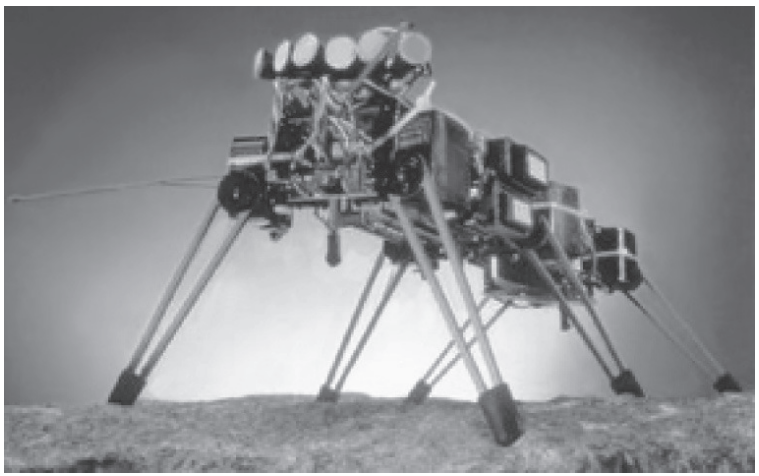


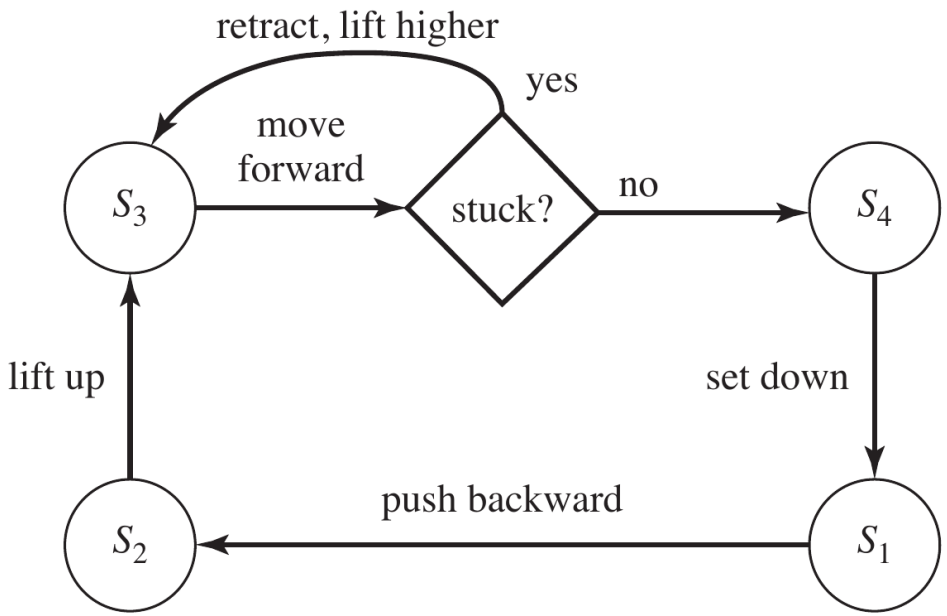


Další způsoby řízení pohybu: reaktivní řízení

Reaktivní řízení

- můžeme specifikovat regulátor bez explicitního modelu prostředí
- nejprve si určíme vzor posunu končetin
- a pak už reaktivně reagujeme na daný stav
  - když je noha blokována, dej ji zpět, zdvihni ji výše a zkus to znova
  - tj. definujeme konečně stavový automat





Další způsoby řízení pohybu: zpětnovazební učení

Zpětnovazební učení

- př. otočení helikoptéry

- postup
  - zaznamenání řízení lidského pilota a stavu proměnných helikoptéry
  - získáme prediktivní model pro simulaci pohybu

Robotika: shrnutí

- Robot a jeho hardware
  - typy robotů
  - senzory, efektory
- Vnímání robota
  - dynamické Bayesovské sítě a odhad stavu
  - lokalizace a mapování
- Plánování pohybu robota
  - konfigurační prostor
  - prohledávání v grafu
  - diskretizace spojitého problému
- Pohyb robota
  - regulátory

Zdroje, ze kterých průsvitky čerpají, a poděkování

V průsvitkách jsou použity obrázky a texty z uvedených zdrojů

- Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, third edition. Prentice Hall, 2010. `http://aima.cs.berkeley.edu/`
- El-Ghazali Talbi, Metaheuristics: From Design to Implementation. Wiley, 2009. `http://www.lifl.fr/~talbi/metaheuristic/`
- Roman Barták, Plánování a rozvrhování. Přednáška na MFF UK, `http://ktiml.mff.cuni.cz/~bartak/planovani`
- Roman Barták, Umělá inteligence II. Přednáška na MFF UK, `http://ktiml.mff.cuni.cz/~bartak/ui2`
- Steven M. LaValle, Planning algorithms, Cambridge University Press, 2006. `http://planning.cs.uiuc.edu/`
- Přednáška MEAM 620: Robotics, University of Pennsylvania, 2017. `https://alliance.seas.upenn.edu/~meam620/wiki/index.php?n=Main.Schedule`
- Jan Faigl, Robot Motion Planning, přednáška Planning and Games na FEL, ČVUT, 2016.