Umělá inteligence

UI (AI) - součást informatiky s průniky mimo obor

Stručná historie UI

- 1943-56 začátky (modelování neuronů a sítí na počítači)
- 1952-69 velká očekávání (GPS, Lisp, microworlds)
- 1966-74 vystřízlivění
- 1969-79 znalostní systémy (zpracování nejistoty, plánování)
- 1980-88 UI se stává průmyslem (5. generace, star wars)
- od 1986 - znovuobjevení neuronových sítí
- současnost - mobilní agenti

? Co je to UI ?

? Etické , politické, sociální otázky ?

Letošní cena nadace Vize 2000 - Joseph Weizenbaum

Řešení problémů hledáním

UI v širším pohledu - viz předmět 33ZUI

Implementační nástroje UI - viz předmět 36JUI

Zde se věnujeme jen části navazující na grafovou tématiku.

"Problem Solving"

Problém (specifikace problému)

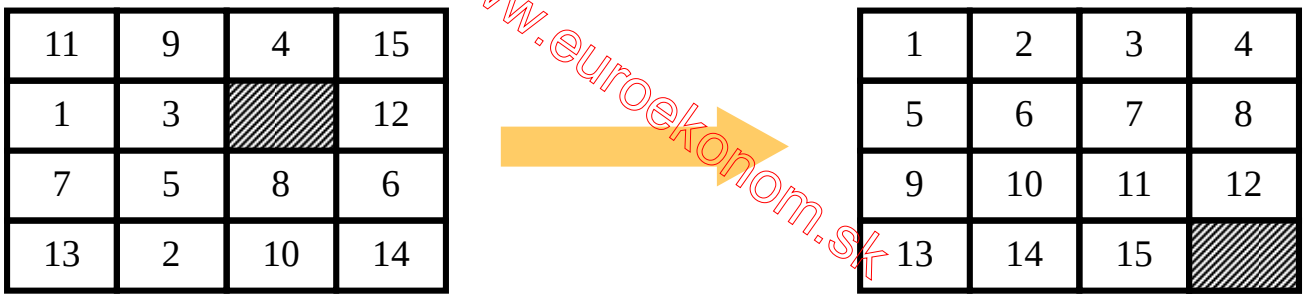
souhrn informací, podle nichž je možno rozhodovat se, co dělat

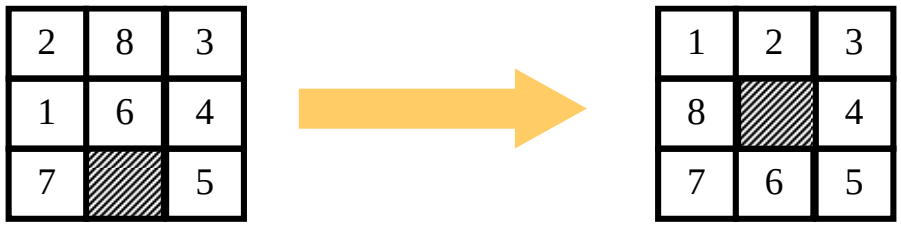
Počáteční stav	**operátor/akce**	**stavový prostor**
cílový stav	**cesta řešení**	**cena cesty**
cena hledání	**graf problému**	

8 nebo 15 - problém

Zadání: Tabulka/krabička 3×3 nebo 4×4 s kameny číslovanými 1 až 8 (nebo 15) a jedním volným místem.

Úkol: posouváním seřadit kameny.





Neinformované hledání

? Co nás zajímá na zvolené strategii hledání ?

- **úplnost** - zaručuje nalezení řešení (pokud existuje)
- **výpočetní** (časová) složitost
- **paměťová** složitost
- **optimálnost** - nalezne se nejlepší řešení?

Neinformované (slepé) vs. informované (heuristické) hledání

Běžné strategie neinformovaného hledání:

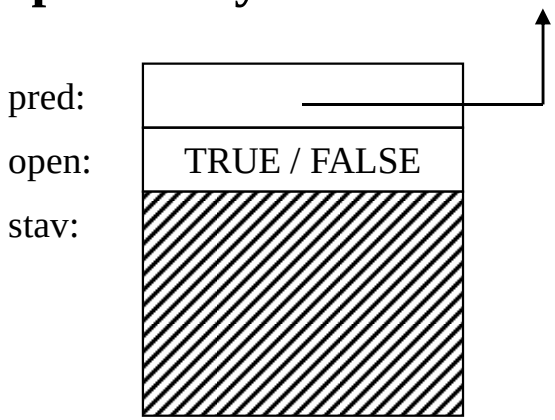
- prohledávání do šířky
- prohledávání uspořádaným výběrem (uniform cost)
- prohledávání do hloubky
- obousměrné prohledávání

Rozdíl - graf problému je implicitní a (potenciálně) nekonečný

Důsledek:

- nejsou FRESH uzly, jen OPEN a CLOSED
- musíme uchovávat stav problému
- struktura grafu je zadána operačně (generováním následníků)

Data ukládaná pro každý uzel



Prohledávání do šířky

BFS (Γ, s, t) (s,t jsou stavy problému)

```
1   ps:=CreateNode(s)
2   ps^.open := TRUE; ps^.pred := NIL
3   InitQueue;  Enqueue(ps);  Found := FALSE
4   while not (EmptyQueue or Found) do
5       pu := QueueFirst
6       if pu^.stav = t then Found := TRUE
7           else for všechna v∈Γ(u) do
8                   if v je nový stav
9                      then pv := CreateNode(v); pv^.open := TRUE
10                            pv^.pred := pu;  Enqueue(pv)
11      Dequeue; pu^.open := FALSE
12  return Found
```

! problém !

? Složitost ?

Prohledávání uspořádaným výběrem (uniform cost)

Předpokládá **ocenění cest** k uzlům **f(n)** - hodnotící funkce

```
1   S:=∅; InitQueue(Q)
2   Enqueue(Q,s); Found:=FALSE
3   while not (Empty(Q) or Found) do
4       u:= ExtractMin(Q); S:=S ∪{u}
5       if u=t then Found:=TRUE
6          else for každé v∈Γ(u) do
7                    if v je nový uzel
8                       then zařaď v do Q s hodnotou f(v)
9                       else přepočti f(v) a možná zařaď
10      Dequeue(Q)
11  return Found
```

Dikstrův algoritmus: f(n) = d(s,u)

? Nalezne se optimální řešení ?

<u>Podmínka:</u> hodnoty **f(n)** neklesají podél cesty řešení.
Splněno automaticky, když se cena uzlu počítá jako součet cen jednotlivých hran (akcí) a když jsou tyto ceny nezáporné.

Prohledávání do hloubky

?Jak prohledávat do hloubky nekonečný stavový prostor?

Řešení:

- omezení hloubky hledání (? úplnost ?)
- **iterativní prohlubování**

Obousměrné prohledávání (do šířky)

Podmínka: explicitně zadaný cílový stav

Srovnání neinformovaných metod

	BFS	Unif. Cost	DFS	Lim. Depth	Iter. Deep.	Bi-direct
čas	b^d	b^d	b^m	b^k	b^d	$b^{d/2}$
paměť	b^d	b^d	b.m	b . k	b.d	$b^{d/2}$
optimal	ANO	ANO	NE	NE	ANO	ANO
úplnost	ANO	ANO	NE	ANO pro k>=d	ANO	ANO

d = délka cesty řešení, m = hloubka hledání, k = omezení hloubky

Heuristické hledání

Předpokládejme, že jsme schopni odhadovat célku (zbývající) cesty k cíli - **heuristická funkce h(u).**

Označení:

g(u) ... délka cesty od počátku do u, d(s,u)

h(u) ... délka cesty od u do cíle, d(u,t)

f(u) ... hodnotící funkce uzlu (kombinuje g a h)

ĝ'(u), h'(u) , f'(u) ... aproximace g, h, f

Použijeme uspořádané hledání s hodnotící funkcí f(u):

f '(u) = h'(u) ... Best/First

f '(u) = h'(u) + g'(u) ... A* (A-star)

f '(u) = α . h'(u) + (1- α) . g'(u)

Přípustnost heuristického hledání

Přípustný algoritmus hledání - najde optimální řešení (min. cestu), pokud existuje.

Lemma: Nechť $h'(u) \leq h(u)$ pro všechny uzly. Potom až do skončení A* existuje na minimální cestě P(s,t) otevřený uzel u' tak, že

$$f'(u) \leq f(s)$$

D: $P=\langle s=u_0, u_1, u_2, \ldots v, \ldots, u_n=t\rangle$ je min. cesta, v otevřený

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

přitom $g'(v) = g(v)$ - předchůdci CLOSED, $h'(v) \leq h(v)$ tedy

$f'(v) = h'(v) + g'(v) \leq h(v) + g(v) = f(v) = f(s)$

V: Nechť $h'(u) \leq h(u)$ pro všechny uzly a délky všech hran jsou větší než jisté $\delta > 0$. Pak je algoritmus A* přípustný.

$h'(u)$ je **konzistentní** ... pro každé u,v platí

$$h'(u) - h'(v) \leq d(u,v)$$

V: Nechť $h'(u)$ je konzistentní a u byl uzavřen během A*. Potom je $g'(u) = g(u)$.

V: Nechť

- A_1^* je A* s heuristickou funkcí $h_1'(u)$,
- A_2^* je A* s heuristickou funkcí $h_2'(u)$,
- $h_1'(u) > h_2'(u)$ pro všechny uzly mimo cílových,
- $h_1'(u)$ je konzistentní

Potom každý uzel expandovaný algoritmem A_1^* bude expandován také algoritmem A_2^*.

více informované hledání

Hledání s omezenou pamětí

Paměťová složitost A* ... počet uzlů roste s b^d, d = délka cesty řešení.

? Jak dosáhnout zlepšení ? Pokud

$$|h'(u) - h(u)| = O(\log h(u)),$$

pak roste počet uzlů pomaleji. Jak ale nalézt takové h'(u) ?

? Jak vyjít s danou velikostí paměti ?

Dvě varianty A*:

- **IDA* - iterative deepening A***
- **SMA* - simplified memory-bounded A***

IDA*

```
1  function IDA* (s, Γ);
2     f_limit := f(s); solution := nil;
3     while (solution = nil) and (f_limit < ∞)
4          do solution, f_limit := DFS(s,f_limit)
5      if solution = nil
6          then return failure
7          else return solution
```

```
1  function DFS (u, f_lim);
2      if f(u) > f_lim then return nil, f(u);
3      if goal(u) then return u, f_lim;
4      next_f := ∞;
5      for x in Γ(u) do
6          sol, fnew := DFS(x, f_lim);
7          if sol ≠ nil then return sol, f_lim;
8          next_f := min(next_f, fnew);
9      return nil, next_f
```

Vrátí řešení nebo nejbližší hodnotu f za limitem

SMA*

Problémy IDA*

- opakované hledání stejných cest
- žádné předávání informace mezi iteracemi

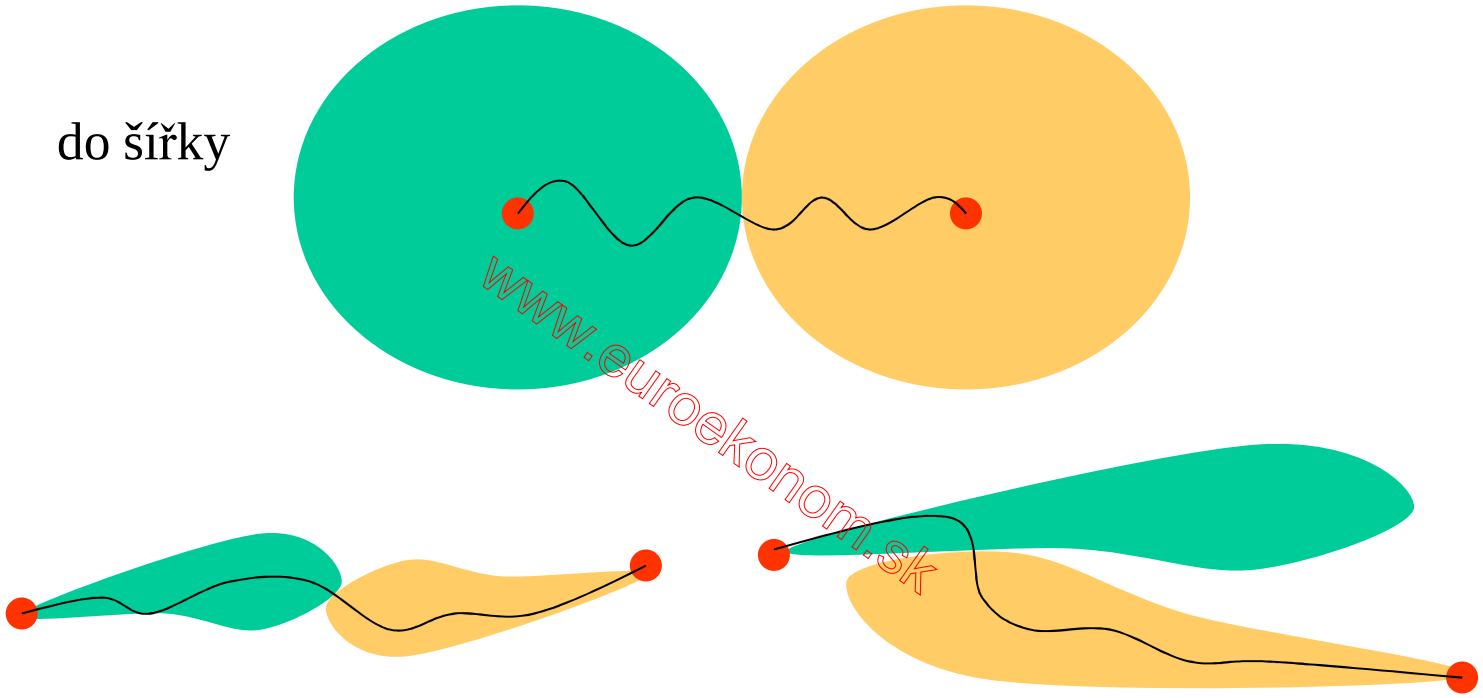
SMA* se snaží pamatovat, co lze

- využije všechnu dostupnou paměť
- brání opakování stavů
- za jistých podmínek je přípustný

Základní myšlenka SMA *

- generuje se následník a není místo pro jeho uložení
- vypustí se uzel s nejhorší hodnotou f '(u)
- u jeho předchůdce se pamatuje délka nejlepší cesty ze zapomenutého podstromu
- zapomenutý uzel se re-generuje, jen když se ostatní uzly ukázaly ještě horší

Obousměrné heuristické hledání

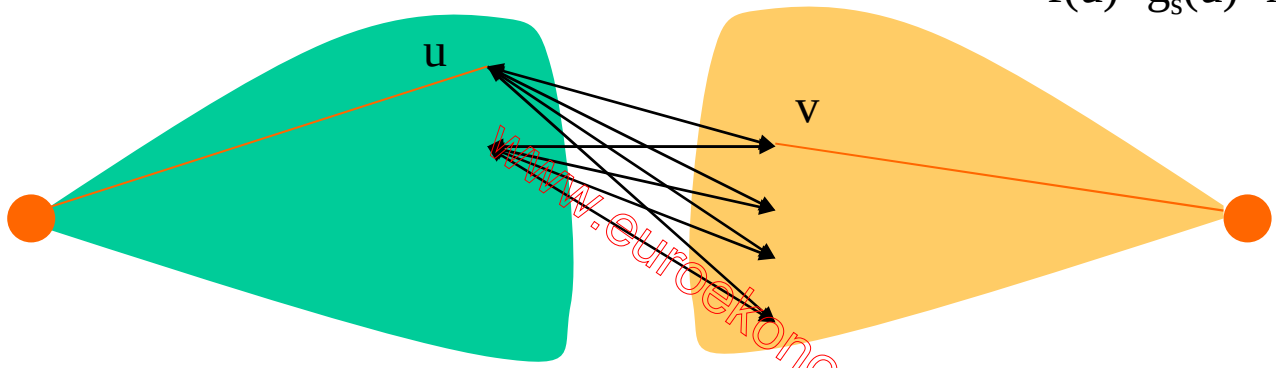


Kriteria: optimálnost cesty / počet generovaných uzlů

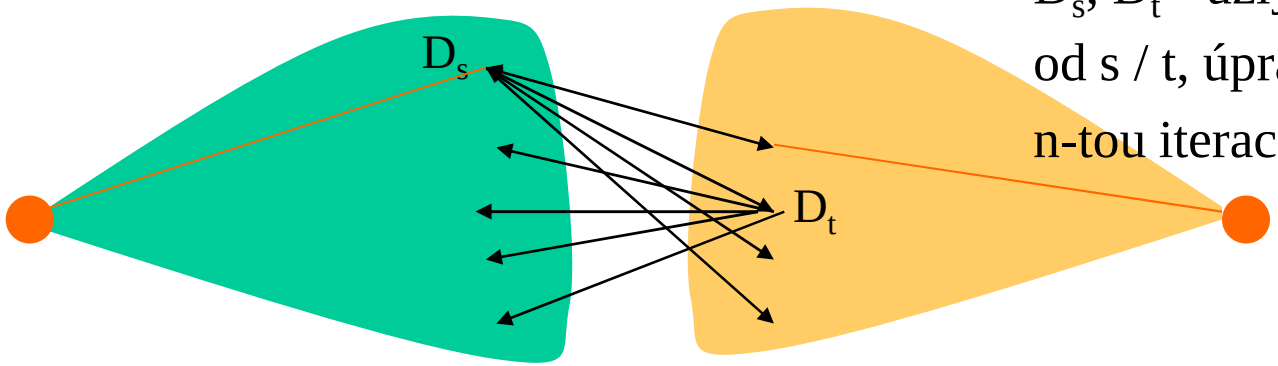
Zlepšení obousměrného heuristického hledání

směrování vln

$f(u)=g_s(u)+h(u,v)+g_t(v)$



přibližné směrování vln



D_s, D_t - uzly nejvzdálenější od s / t, úprava každou n-tou iteraci