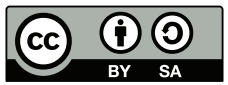
brmiversity: Umělá inteligence a teoretická informatika

Přednáška č. 3

Petr Baudiš ⟨pasky@ucw.cz⟩

brmlab 2011



brmlab hackerspace prague

Outline

Autonomní adaptivní agent (umělá bytost)

(Velmi inspirováno slajdy Cyrila Broma.)

- Virtuální bytost, ve virtuálním nebo skutečném prostředí, autonomní, s vlastním tělem

- Artificial mind is a piece of code that decides "what to do next"
- The problem of deciding what to do next is called the action selection problem
- To decide what to do next, the creature must perceive its environment
- An action causes a change in the environment, and it usually has a feedback on the creature

Architektura agenta

Non-free artwork, viz slajdy Cyrila Broma k Umělým bytostem.

- Reaktivní "plánování": "Turingův stroj" — na základě stavu a podnětu akce + nový stav; vlastně se neplánuje
- If-then pravidla: Když se stane X, udělej Y, sada uspořádaných pravidel s prioritou
- Konečný automat: viz minule
- Neuronové sítě atd.

Architektura agenta

- POSH: Parallel-rooted, Ordered Slip-stack Hierarchical: Hierarchická if-then pravidla; akce, kompetence, "pudy"
- Belief Desire Intention: Možná špatné beliefs, množina desires, některé z nich vybrané jako intentions
- Soar: State, Operator And Result (Input; Proposing – Decision – Applying; Output)

Otázky?

Příště: Pohyb agentů — navigace v prostoru.

Outline

Umělá evoluce

- Příroda: Živí jedinci jsou (částečně) popsáni svojí DNA, ta se mění s časem podle působení přírodního výběru
- Programy: Jedinci odpovídající problémům jsou popsáni nějak kódovaným řetězcem, ten se mění s časem podle působení výběru (selekce) na základě fitness
- Kódování: Řetězec hodnot
- Fitness: Ohodnocení výkonu jedince
- Změna jedince: Operace na kódování — mutace a křížení
- Velká populace mnoha jedinců!

Genetické operátory

- **Selekce** — ohodnotíme množinu jedinců WTF
- **Mutace** — $\forall i$ s (velmi nízkou) pravděpodobností p změníme hodnotu prvku i jedince
- **Křížení** — vyrobíme nového jedince rekombinací řetězců dvou jedinců

Genetický algoritmus

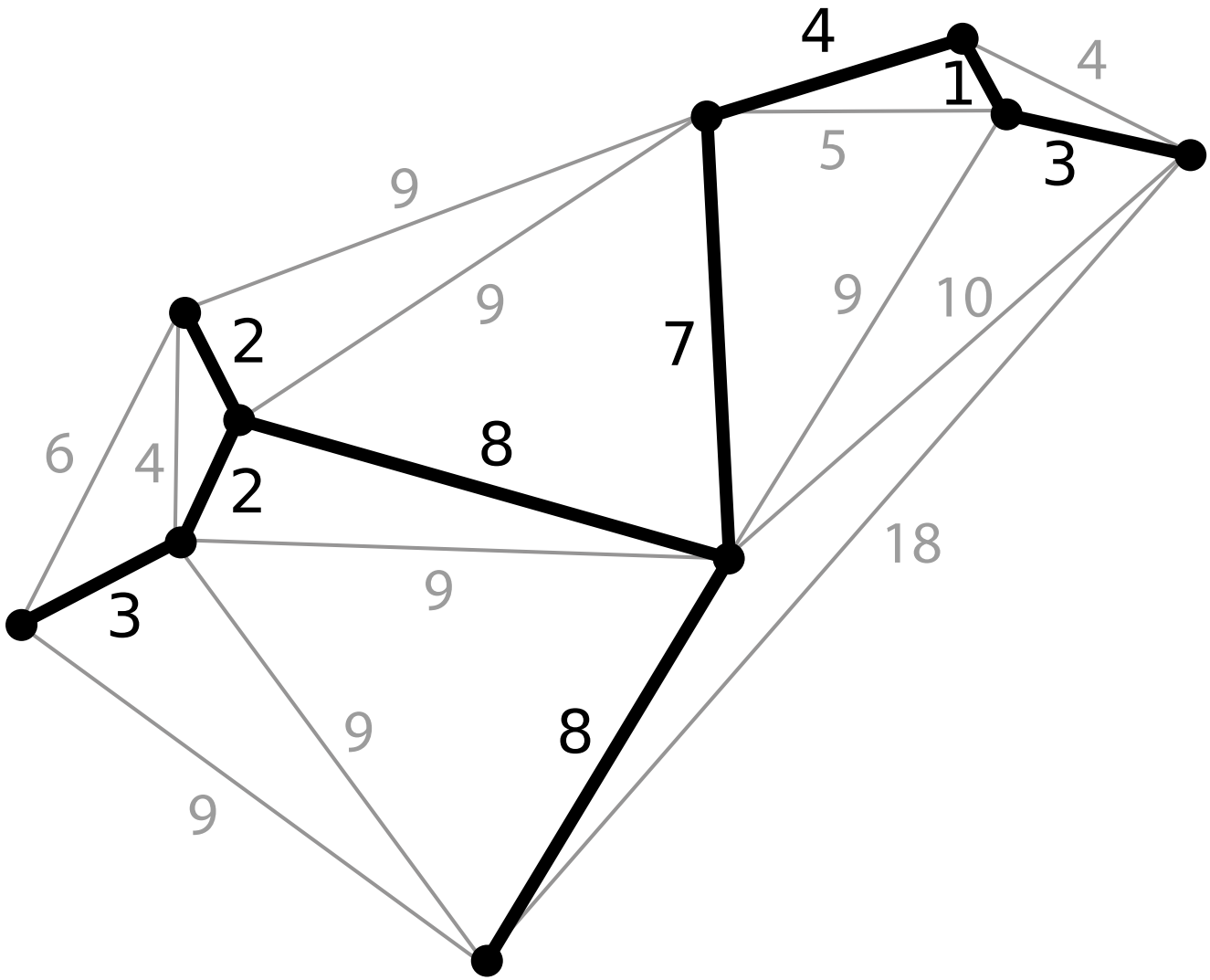
- Máme populaci předchozí generace, vyrábíme novou generaci (dokud nenajdeme optimum).
- Nová generace (dokud není plná):
  - Vyber dva jedince z minulé populace (selekce)
  - Vyrob dva nové jedince (křížení)
  - Zmutuj jedince
  - Dvojici vlož do nové generace
- Elitismus a spoustu dalších vylepšení

Otázky?

Příště: Reprezentační schémata, evoluční a genetické programování.

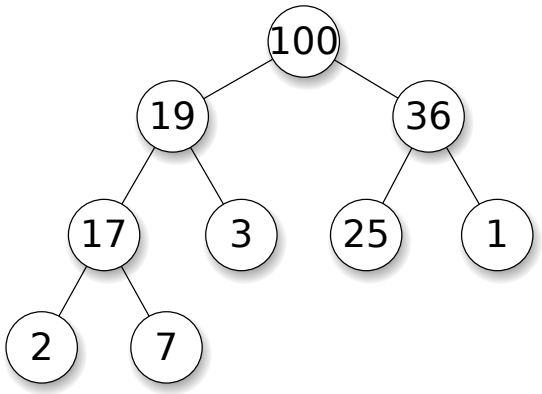
Outline

Minimální kostra grafu



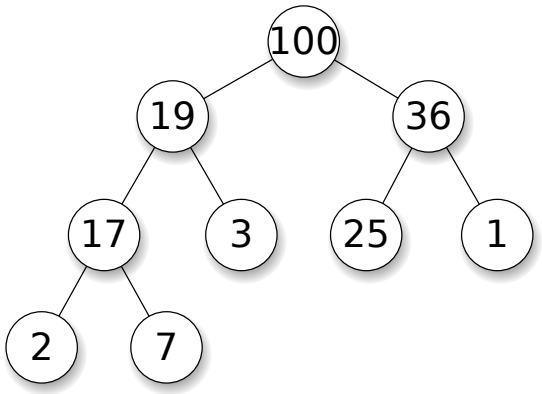
Outline

Halda



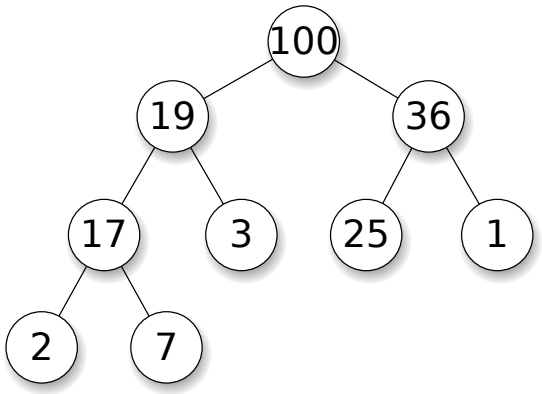
- Chceme ukládat data tak, abychom vždy mohli snadno vytáhnout minimum
- Prioritní fronta, např. na minimální kostru grafu :-)

Halda



- Chceme ukládat data tak, abychom vždy mohli snadno vytáhnout minimum
- Prioritní fronta, např. na minimální kostru grafu :-)
- . . . cože?

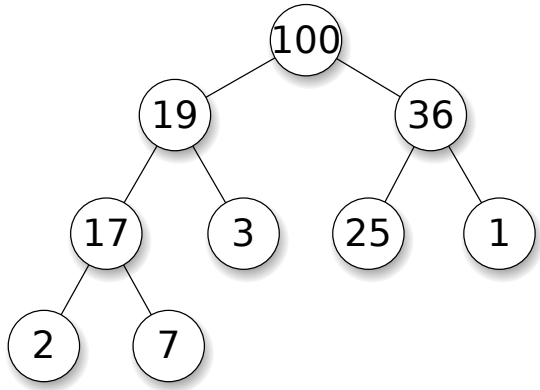
Halda



- Chceme ukládat data tak, abychom vždy mohli snadno vytáhnout minimum
- Prioritní fronta, např. na minimální kostru grafu :-)
- . . . cože?
- Speciálně uspořádaný strom: synové jsou vždy větší než otec
- Tedy v kořeni je nejmenší prvek
- Zejména operace INSERT, DELETEMIN

brmlab hackerspace prague

Regulární halda



- Pěkně "zarovnaný" strom, každý otec kromě těch u dna má dva syny (má hloubku $\log n$)
- Mezi syny není žádné pořadí
- INSERT: Přidáme na "konec" haldy (na dně), posouváme nahoru, dokud je porušena podmínka
- DELETEMIN: Prohodíme kořen a poslední prvek, umažeme poslední prvek, prvek z kořene posouváme dolů
- Složitosti $O(\log n)$
- Heap sort!

brmlab hackerspace prague

Levicové (leftist) haldy

- Místo probublávání budeme mít jako základní operaci MERGE dvou hald.
- Línější strukturální podmínka; levý syn má vždy delší cestu k nejbližšímu listu.
- MERGE prohazuje uzly podle uspořádání, syny podle cesty k listu, rekurze do *pravého* syna.

Otázky?

Příště: Složitější haldy. Hashovací algoritmy.

Outline

Rekapitulace: P a NP

- P: Třída problémů řešitelných v polynomiálním čase $O(n^k)$ na DTS (Turingově stroji)
- NP: Třída problémů řešitelných v polynomiálním čase $O(n^k)$ na NTS (nedeterministickém Turingově stroji)
- NTS: Nepřecházíme do jednoznačného dalšího stavu programu, ale do několika možných
- (V praxi musíme NTS simulovat na DTS a vykonávat postupně všechny možné větve výpočtu — exponenciální časová složitost $O(2^n)$)
- (Konsekvence NP: *Certifikát* NP problému (popis větvení programu) lze ověřit v polynomiálním čase na DTS.)
- Může být nějaký program ještě pomalejší než NP?

Rekapitulace: P a NP

- P: Třída problémů řešitelných v polynomiálním čase $O(n^k)$ na DTS (Turingově stroji)
- NP: Třída problémů řešitelných v polynomiálním čase $O(n^k)$ na NTS (nedeterministickém Turingově stroji)
- NTS: Nepřecházíme do jednoznačného dalšího stavu programu, ale do několika možných
- (V praxi musíme NTS simulovat na DTS a vykonávat postupně všechny možné větve výpočtu — exponenciální časová složitost $O(2^n)$)
- (Konsekvence NP: *Certifikát* NP problému (popis větvení programu) lze ověřit v polynomiálním čase na DTS.)
- Může být nějaký program ještě pomalejší než NP?
- Může! EXPTIME (Go)

brmlab hackerspace prague

Úplné problémy

- Problém je ve třídě NP: Dá se v polynomiálním čase spočítat na NTS.
  (Popíšeme, jak certifikát ověřit v p. čase.)
  V NP jsou i všechny *lehčí* problémy!
- Problém je NP-těžký: Každý problém z NP lze převést na ten náš
  (Napíšeme program, který to dělá.)
- Problém je NP-úplný: Problém je NP a NP-těžký

Úplné problémy

- Problém je ve třídě NP: Dá se v polynomiálním čase spočítat na NTS.
  (Popíšeme, jak certifikát ověřit v p. čase.)
  V NP jsou i všechny *lehčí* problémy!
- Problém je NP-těžký: Každý problém z NP lze v polynomiálním čase převést na ten náš
  (Napíšeme program, který to dělá.)
- Problém je NP-úplný: Problém je NP a NP-těžký

Úplné problémy

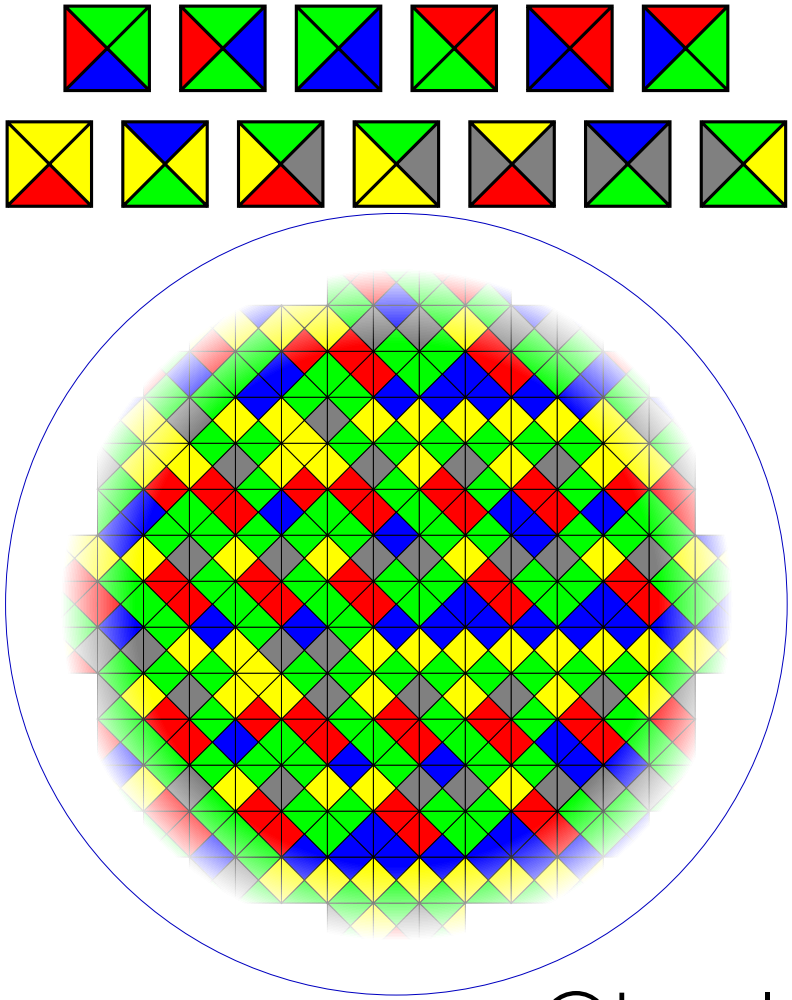
- Problém je ve třídě NP: Dá se v polynomiálním čase spočítat na NTS.
  (Popíšeme, jak certifikát ověřit v p. čase.)
  V NP jsou i všechny *lehčí* problémy!
- Problém je NP-těžký: Každý problém z NP lze v polynomiálním čase převést na ten náš
  (Napíšeme program, který to dělá.)
- Problém je NP-úplný: Problém je NP a NP-těžký
- Je to zábava, lépe pochopíme vlastnosti třídy NP
- P=NP dokážeme na jednom NP-úplném algoritmu, můžeme pak v P řešit všechny
- Některé NP problémy umíme heuristicky rychle řešit "skoro dokonale" (3SAT), aplikujeme na jiné problémy

brmlab hackerspace prague

Kachlíkování

- Máme sadu barev a sadu kachlíků s různě obarevnými hranami. K sobě patří kachlíky se stejně barevnými hranami
- Obdélníkové pole (“koupelna”), vstupní hrana s obarvenými “bity”, výstupní hrana, kterou musíme obarvit
- Program jsou barvy a kachlíky!

- Texturování v počítačové grafice
- Proteiny místo kachliček — biologické výpočty



brmlab hackerspace prague

TS na kachlíkování

- Intuice: Najít vykachlíkování je těžké (Eternity). Ověřit předložené vykachlíkování (postup běhu programu, certifikát) je snadné.
- Problém je v NP (řešení ověříme rychle)

- Jak dokázat, že je problém NP-těžký?

TS na kachlíkování

- Intuice: Najít vykachlíkování je těžké (Eternity). Ověřit předložené vykachlíkování (postup běhu programu, certifikát) je snadné.
- Problém je v NP (řešení ověříme rychle)

- Jak dokázat, že je problém NP-těžký?
- (i) Převedeme na něj jiný NP-těžký
  (ii) Budeme s ním simulovat Turingův stroj

TS na kachlíkování

- Kachlíkujeme po řádcích, první řádek (vstupní hrana) je vstupní páska
- V každém řádku bude zakódovaný jeden krok Turingova stroje
- Hrany od vstupní k výstupní hraně: stav pásky před krokem, za krokem
  highlight buňky pod hlavou
  hrany mezi buňkami řádku: přechody hlavy

Non-free artwork, viz skripta Vladana Majerecha "Úvod do složitosti a NP-úplnosti".

brmlab hackerspace prague

SAT

- Satisfiability: Vstup je logický výraz v "konjunktivní normální formě":

$$(a \vee b \vee \neg c) \wedge (\neg \vee a \vee b \vee \neg c)$$

  Existuje splňující kombinace logických hodneg proměnných?
- Dá se převést kachlíkování na 3-SAT
- $x_{i,j,k}$ kachlíček k na souřadnicích i, j
- $\neg x_{i,j,k} \vee \neg x_{i,j,k'}$, $x_{i,j,k} \vee x_{i,j,k'}$
- $\neg x_{i,j,k} \vee \neg x_{i,j+1,k'}$ a analogicky pro i podle barev

3-SAT

3-Satisfiability: trojliterálové závorky

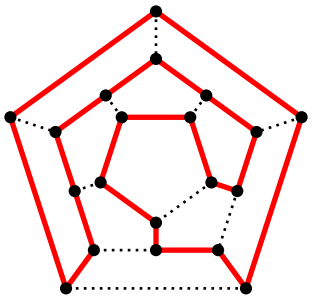
$$(a \vee b \vee \neg c) \wedge (\neg \vee a \vee b \vee \neg c)$$

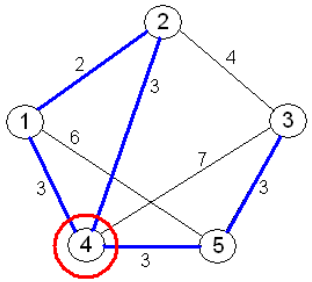
SAT: Obecný počet literálů. Převedeme ho na 3-SAT, pro delší závorky vyrobíme pomocné literály:

$$(a \vee \neg b \vee c \vee d)$$

$$(a \vee \neg b \vee x_1) \wedge (\neg x_1 \vee c \vee x_2) \wedge (\neg x_2 \vee d \vee 1)$$

Hamiltonovská kružnice





- Chceme navštívit všechny vrcholy grafu, abychom každý navštívili právě jednou.
- SAT umíme převést na HK

Non-free artwork, viz skripta Vladana Majerecha “Úvod do složitosti a NP-úplnosti”.

brmlab hackerspace prague

Otázky?

Příště: Metody tvorby algoritmů. (Problém batohu později.)

Děkuji vám

pasky@ucw.cz

Příště: Neuronové sítě, umělá inteligence (hry), složitost, vyčíslitelnost (halting problem).

brmlab hackerspace prague