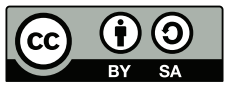
brmiversity: Umělá inteligence a teoretická informatika

Přednáška č. 12

Petr Baudiš ⟨pasky@ucw.cz⟩

brmlab 2011



brmlab hackerspace prague

Outline

Problém batohu

- Problém batohu — kapacita C_{max} a N věcí hodnoty $v(i)$ a ceny (velikosti) $c(i)$
- Maximalizujeme $\sum_i v(i)$, aby $\sum_i c(i) \leq C_{max}$
- Zakódování?

Problém batohu

- Problém batohu — kapacita C_{max} a N věcí hodnoty $v(i)$ a ceny (velikosti) $c(i)$
- Maximalizujeme $\sum_i v(i)$, aby $\sum_i c(i) \leq C_{max}$
- Zakódování? Kód je binární řetězec, pozice odpovídá věci
- Genetické operátory přímočaře
- Fitness?

Problém batohu

- Problém batohu — kapacita C_{max} a N věcí hodnoty $v(i)$ a ceny (velikosti) $c(i)$
- Maximalizujeme $\sum_i v(i)$, aby $\sum_i c(i) \leq C_{max}$
- Zakódování? Kód je binární řetězec, pozice odpovídá věci
- Genetické operátory přímočaře
- Fitness? Máme dva členy (zakódování dovoluje neplatná řešení); vážený součet, nebo změnit zakódování

Problém batohu

- Problém batohu — kapacita C_{max} a N věcí hodnoty $v(i)$ a ceny (velikosti) $c(i)$
- Maximalizujeme $\sum_i v(i)$, aby $\sum_i c(i) \leq C_{max}$
- Zakódování? Kód je binární řetězec, pozice odpovídá věci
- Genetické operátory přímočaře
- Fitness? Máme dva členy (zakódování dovoluje neplatná řešení); vážený součet, nebo změnit zakódování
- Zakódování' — 1: Vlož věc do batohu, *pokud* se to tam vejde

TSP

- Chceme objet n měst s minimálními náklady
- Fitness?

TSP

- Chceme objet n měst s minimálními náklady
- Fitness? Náklady za cestu
- Reprezentace?

TSP

- Chceme objet n měst s minimálními náklady
- Fitness? Náklady za cestu
- Reprezentace? Vrcholy? Hrany?

TSP

- Chceme objet n měst s minimálními náklady
- Fitness? Náklady za cestu
- Reprezentace? Vrcholy? Hrany?
- Cesta 1–2–4–3–8–5–9–6–7
- Reprezentace sousedností: (248397156)
  Cesta nemusí být přípustná; křížení nefunguje, schémata ano!
- Ordinální reprezentace: zásobník (123456789);
  kód (112141311)
  Křížení funguje — má smysl?
- Reprezentace cestou: (124385967)
  Křížení je třeba udělat custom; PMX, CX, OX, ER

Neuroevoluce

Klasická neuronová síť — učení vah nebo celé struktury

Učení vah

- Mám fitness, ale ne chybu v každém kroku; dobrá paralelizace
- Ovšem často (řádově) pomalejší než gradientní metody

Učení struktury

- Fitness: Postavím síť, několikrát naučím
- Přímé kódování: Matice sousednosti
- Gramatické kódování: Program na růst sítě (přidej neuron, rozděl neuron, přepoj synapsi, . . .)

Neuroevolution of Augmenting Topologies (NEAT)

- Každá hrana má informace o vrcholech, váze a *rodné číslo*
- Křížení: Mergují se váhy pouze hran se stejným číslem
- Fitness je relativní vzhledem k *druhu* (podobnost vektorů rodných čísel) — ochrana nových topologií

Strojové učení

Systém *rozhodovacích pravidel*

Michiganský model

- Jedinec je pravidlo $\{flags\} \rightarrow action$ s váhou (úspěšností)
- Expertní systém je celá populace
- Evoluce zvolna na části populace
- Problém reaktivnosti — vzájemná návaznost pravidel; bucket brigade algorithm

Pittsburský model

- Jedinec je celá sada pravidel
- Komplikovanější fitness i genetické operátory
- Konceptuálně jednodušší, spolehlivější?

Otázky?

To je o evolučních algoritmech vše!

Outline

Rekapitulace — Turingův stroj

- **Turingův stroj:** Pětice $(Q, \Gamma, b, \Sigma, q_0, F, \delta)$
- Nekonečná páska *(data)* rozdělená na buňky, v každé jedno písmeno z "abecedy"; můžeme mít i více pásek!
- Hlava stroje *(pozice na pásce)* se hýbe v jednom kroku o buňku doleva nebo doprava
- Stav stroje *(pozice v programu)* z množiny stavů
- Přechodová funkce *(program)* podle stavu a písmena pod hlavou přepne stroj do nového stavu, zapíše nové písmeno a posune hlavu

Rekapitulace — Třídy složitosti

- **Třídy složitosti:** Skupina algoritmů se stejnou řádovou složitostí v závislosti na délce vstupu (P, NP, LSPACE, PSPACE, EXPTIME, ...)
- P: Všechny algoritmy, které běží v *polynomiálním čase* na "obyčejném" Turingově stroji (DTS)
- NP: Algoritmy, které běží v polynomiálním čase na nedeterministickém Turingově stroji (NTS)
- PSPACE: Všechny algoritmy, které sežerou *polynomiálně mnoho pásky*

PSPACE-úplnost

- True Quantified Boolean Formula (TQBF) — SAT s posloupností kvantifikátorů
- Hra na hlavní města
- Poziční hry (hex, schody v Go, . . .)
- “Stromové” hry: Mohou trvat dlouho, EXPTIME!

Vyčíslitelnost a složitost

- Dnes: Trochu spojíme vyčíslitelnost a složitost
- Dnes: Budeme zkoumat, jak se třídy složitosti vzájemně proplétají

- Zjednodušení: Zadání i výsledek budeme kódovat n-ticí jedniček; zajímá nás nějaká $f : \mathbb{N} \to \mathbb{N}$
- f je **rekurzivní**, pokud existuje DTS M přepisující $1^n \to 1^{f(n)}$
- f je **vyčíslitelná** v čase $O(f)$, pokud M udělá nejvýše $cf(n)$ kroků
- f je **vyčíslitelná** v prostoru $O(f)$, pokud M použije nejvýše $cf(n)$ buněk

Konstruovatelné funkce

- Jaké druhy funkcí můžou definovat složitost algoritmu?
- g je **časově konstruovatelná**, pokud M zastaví právě po $g(n)$ krocích
- g je **prostorově konstruovatelná**, pokud M použije právě $g(n)$ buněk
- Lze dokázat: Pro superlineární funkce je už vyčíslitelnost a konstruovatelnost to samé!

Hierarchie tříd složitosti

$DTIME(T(n)), DSPACE(S(n)), NTIME(T(n)), NSPACE(S(n))$
Třída jazyků (predikátů, problémů) řešitelných v určité mezi na DTS resp. NTS

- $\forall n : F_1(n) \leq F_2(n) \Rightarrow CLASS(F_1(n)) \subseteq CLASS(F_2(n))$
- $DTIME(T(n)) \subseteq NTIME(T(n))$
- $DTIME(T(n)) \subseteq DSPACE(T(n))$,
  $NTIME(T(n)) \subseteq DSPACE(T(n))$
- $DSPACE(S(n)) \subseteq DTIME(c^{S(n)})$ $\quad S(n) \geq \log_2(n)$
- $NTIME(T(n)) \subseteq DTIME(c^{T(n)})$

- Časová a prostorová hierarchie jsou otevřené shora: Pro každou rekurzivní funkci T existuje $L \notin DTIME(T(n))$.

brmlab hackerspace prague

Věty o zrychlení a mezerách

- Věta o lineárním zrychlení DTS: Tím, že velmi rozšíříme množinu stavů, dokážeme lineárně zrychlovat práci stroje

Věty o zrychlení a mezerách

- Věta o lineárním zrychlení DTS: Tím, že velmi rozšíříme množinu stavů, dokážeme lineárně zrychlovat práci stroje
- Borodinova věta o mezerách: Pro každou rekurzivní funkci $g(n) \geq n$ existuje rekurzivní $T(n)$ (mez) taková, že $DTIME(T(n)) = DTIME(g(T(n)))$
  Tedy i pro velmi rychle rostoucí $g(n)$ existuje třída složitosti T, která ji pozře!

Věty o zrychlení a mezerách

- Věta o lineárním zrychlení DTS: Tím, že velmi rozšíříme množinu stavů, dokážeme lineárně zrychlovat práci stroje
- Borodinova věta o mezerách: Pro každou rekurzivní funkci $g(n) \geq n$ existuje rekurzivní $T(n)$ (mez) taková, že $DTIME(T(n)) = DTIME(g(T(n)))$
  Tedy i pro velmi rychle rostoucí $g(n)$ existuje třída složitosti T, která ji pozře!
- Blumova věta o zrychlení: Pro každou rekurzivní funkci g (zrychlovací funkce) existuje rekurzivní predikát f (úloha) takový, že pro každý DTS M_i existuje DTS M_j řešící f tak, že $g(T_j(n)) \leq T_i(n)$ pro *skoro každý* vstup.
  Řešení některých úloh tedy lze stále zrychlovat!

brmlab hackerspace prague

Věty o zrychlení a mezerách

- Věta o lineárním zrychlení DTS: Tím, že velmi rozšíříme množinu stavů, dokážeme lineárně zrychlovat práci stroje
- Borodinova věta o mezerách: Pro každou rekurzivní funkci $g(n) \geq n$ existuje rekurzivní $T(n)$ (mez) taková, že $DTIME(T(n)) = DTIME(g(T(n)))$
  Tedy i pro velmi rychle rostoucí $g(n)$ existuje třída složitosti T, která ji pozře!
- Blumova věta o zrychlení: Pro každou rekurzivní funkci g (zrychlovací funkce) existuje rekurzivní predikát f (úloha) takový, že pro každý DTS M_i existuje DTS M_j řešící f tak, že $g(T_j(n)) \leq T_i(n)$ pro *skoro každý* vstup.
  Řešení některých úloh tedy lze stále zrychlovat!

- Poučení: V teorii složitosti záleží jen na řádových změnách (O-notace má smysl)

Savičova věta

- $S(n)$ prostorově konstruovatelná funkce, $S(n) \geq \log_2(n)$
- **Savičova věta:** $NSPACE(S(n)) \subseteq DSPACE(S^2(n))$
- Prostorové omezení je mnohem volnější než časové!
  $PSPACE = NPSPACE$

Idea důkazu

- $TEST(l_1, l_2, i)$ — ze stavu l_1 do l_2 za 2^i kroků?
- $TEST(l_1, l_2, i)$ odpoví hned, může-li, jinak volá
  $TEST(l_1, l_x, i-1) \wedge TEST(l_x, l_2, i-1) \quad \forall x$
- Spustím $TEST(l_0, l_p, kS(n))$ (stavů $2^{kS(n)}$)
- Každý $TEST$ potřebuje $3S(n)$ paměti pro ukládání mezistavu, hloubka rekurze je $kS(n)$, tedy celkem $3kS^2(n)$

Orákulové systémy

- **Orákulum:** Black box, který "umí něco řešit" a radí vám (můžete jej použít při výpočtu)
- V praxi neexistují! Užitečné při analýze algoritmů (např. kryptografické algoritmy)
- Notace: NP(C) je třída jazyků, které rozpozná NTS s orákulem řešícím C

- NP(P) = ?
- P(P) = ?
- NP(NP) = ?

Orákulové systémy

- **Orákulum:** Black box, který "umí něco řešit" a radí vám (můžete jej použít při výpočtu)
- V praxi neexistují! Užitečné při analýze algoritmů (např. kryptografické algoritmy)
- Notace: NP(C) je třída jazyků, které rozpozná NTS s orákulem řešícím C

- NP(P) = NP
- P(P) = ?
- NP(NP) = ?

Orákulové systémy

- **Orákulum:** Black box, který "umí něco řešit" a radí vám (můžete jej použít při výpočtu)
- V praxi neexistují! Užitečné při analýze algoritmů (např. kryptografické algoritmy)
- Notace: NP(C) je třída jazyků, které rozpozná NTS s orákulem řešícím C

- NP(P) = NP
- P(P) = P (zřetězíme výpočet)
- NP(NP) = ?

Orákulové systémy

- **Orákulum:** Black box, který "umí něco řešit" a radí vám (můžete jej použít při výpočtu)
- V praxi neexistují! Užitečné při analýze algoritmů (např. kryptografické algoritmy)
- Notace: NP(C) je třída jazyků, které rozpozná NTS s orákulem řešícím C

- NP(P) = NP
- P(P) = P (zřetězíme výpočet)
- NP(NP) = už nevíme

Orákulové systémy

- **Orákulum:** Black box, který "umí něco řešit" a radí vám (můžete jej použít při výpočtu)
- V praxi neexistují! Užitečné při analýze algoritmů (např. kryptografické algoritmy)
- Notace: NP(C) je třída jazyků, které rozpozná NTS s orákulem řešícím C

- NP(P) = NP
- P(P) = P (zřetězíme výpočet)
- NP(NP) = už nevíme
- Vždy platí $P(C) \subseteq NP(C) \subseteq PSPACE(C)$
- Polynomiální hierarchie: $\Sigma_0 = P$, $\Sigma_{i+1} = NP(\Sigma_i)$, $PH = \bigcup_{i=0}^{\infty} \Sigma_i$
- Platí $PH \subseteq PSPACE$ (indukcí podle i)

brmlab hackerspace prague

Otázky?

Příště: Pseudopolynomiální a aproximační algoritmy.

Outline

Algoritmicky nerozhodnutelné problémy

- Definice: Problém je algoritmicky rozhodnutelný, je-li jeho predikát rekurzivní. Jazyk, který není algoritmicky rozhodnutelný, je algoritmicky nerozhodnutelný.

- Již známe: Halting problem, Riceova věta
- Obecný *rozhodovací problém* pravdivosti logických výroků
- Je jazyk přijímaný daným TS prázdný, neprázdný, konečný?
- Je daná TS vítězný přičinlivý bobr? (busy beaver champion)
- Kolmogorovská složitost řetězce
- Postův korespondenční problém
- Řešení diofantických rovnic
- Problém smrtelné matice
- Kachlíkování nekonečné roviny

brmlab hackerspace prague

Otázky?

Příště: Gödelovy věty.

Děkuji vám

pasky@ucw.cz

Příště: Umělá inteligence a adaptivní agenti
(reprezentace znalostí a odvozování).
Neuronové sítě (hierarchické a modulární).
Složitost? Datové struktury (v externí paměti).

brmlab hackerspace prague