Těžké problémy II

boj s nimi

- Aproximační algoritmy,
- online algoritmy,
- pravděpodobnostní algoritmy,
- heuristiky.

Minule bylo

- Definice těžkosti a úplnosti,
- třídy P a NP,
- důkazy NP-úplnosti problémů KACHL a SAT (Cookova věta).

Další NP-těžké problémy

- k-barevnost grafu (pro $k \geq 3$),

Další NP-těžké problémy

- k-barevnost grafu (pro $k \geq 3$),
- knapsack (problém batohu),

Další NP-těžké problémy

- k-barevnost grafu (pro $k \geq 3$),
- knapsack (problém batohu),
- bin-packing (problém ládování odpadu do košů),

Další NP-těžké problémy

- k-barevnost grafu (pro $k \geq 3$),
- knapsack (problém batohu),
- bin-packing (problém ládování odpadu do košů),
- vertex-cover (problém nalezení co nejmenší množiny vrcholů, ve které má každá hrana aspoň jeden konec).

Mýty a polopravdy

- Polopravda: Mezi třídami P a NP není žádná vlastní třída.

Mýty a polopravdy

- Polopravda: Mezi třídami P a NP není žádná vlastní třída.
- Pravda: Mezi P a NP není žádná vlastní třída, právě když $P = NP$. Jinak lze mezi třídy P a NP nastrkat nekonečně mnoho tříd (věta o hierarchii).

Mýty a polopravdy

- Polopravda: Mezi třídami P a NP není žádná vlastní třída.
- Pravda: Mezi P a NP není žádná vlastní třída, právě když $P = NP$. Jinak lze mezi třídy P a NP nastrkat nekonečně mnoho tříd (věta o hierarchii).
- Mýtus: Třída NP obsahuje všechny algoritmy.

Mýty a polopravdy

- Polopravda: Mezi třídami P a NP není žádná vlastní třída.
- Pravda: Mezi P a NP není žádná vlastní třída, právě když P = NP. Jinak lze mezi třídy P a NP nastrkat nekonečně mnoho tříd (věta o hierarchii).
- Mýtus: Třída NP obsahuje všechny algoritmy.
- Pravda: Existuje dokonce nekonečně dlouhý řetězec tříd (tedy tříd, kdy následující třída obsahuje třídu předchozí), které všechny obsahují třídu NP. Neví se, jak je to s polynomiální hierarchií (máme hierarchii TS počítajících v polynomiálním čase, kdy první má jako oraculum NP-těžký problém, každý následující má oraculum řešící problém pro stroj těžký pro předchozí stroj). Není známo, zda polynomiální hierarchie kolabuje, nebo ne.

Teď už pravda

- Pravda: Vyčíslitelnostní hierarchie nekolabuje.

Teď už pravda

- Pravda: Vyčíslitelnostní hierarchie nekolabuje.
- Důsledek tvrzení: Halting-problem není algoritmicky rozhodnutelný.

Teď už pravda

- Pravda: Vyčíslitelnostní hierarchie nekolabuje.
- Důsledek tvrzení: Halting-problem není algoritmicky rozhodnutelný.
- My se tedy pohybujeme ve velmi malé složitostní třídě.

Co s těžkými (příliš těžkými) problémy?

Teoretičtí informatici se nikdy nevzdávají, živé nás nikdo nedostane...

1. možnost: P- a NP-hypotéza (ukázat, že P=NP).

Co s těžkými (příliš těžkými) problémy?

Teoretičtí informatici se nikdy nevzdávají, živé nás nikdo nedostane...

1. možnost: P- a NP-hypotéza (ukázat, že P=NP).
2. možnost: Chceme nějaké řešení, ne optimální (aproximační algoritmy, aproximační schémata),

Co s těžkými (příliš těžkými) problémy?

Teoretičtí informatici se nikdy nevzdávají, živé nás nikdo nedostane...

1. možnost: P- a NP-hypotéza (ukázat, že P=NP).
2. možnost: Chceme nějaké řešení, ne optimální (aproximační algoritmy, aproximační schémata),
3. možnost: Chceme optimum aspoň pro některé instance v rozumném čase (pravděpodobnostní algoritmy Las Vegas),

Co s těžkými (příliš těžkými) problémy?

Teoretičtí informatici se nikdy nevzdávají, živé nás nikdo nedostane...

1. možnost: P- a NP-hypotéza (ukázat, že P=NP).
2. možnost: Chceme nějaké řešení, ne optimální (aproximační algoritmy, aproximační schémata),
3. možnost: Chceme optimum aspoň pro některé instance v rozumném čase (pravděpodobnostní algoritmy Las Vegas),
4. možnost: Chceme aspoň nějaký pokus o řešení, ale rychle (pravděpodobnostní algoritmy Monte Carlo),

Co s těžkými (příliš těžkými) problémy?

Teoretičtí informatici se nikdy nevzdávají, živé nás nikdo nedostane...

1. možnost: P- a NP-hypotéza (ukázat, že P=NP).
2. možnost: Chceme nějaké řešení, ne optimální (aproximační algoritmy, aproximační schémata),
3. možnost: Chceme optimum aspoň pro některé instance v rozumném čase (pravděpodobnostní algoritmy Las Vegas),
4. možnost: Chceme aspoň nějaký pokus o řešení, ale rychle (pravděpodobnostní algoritmy Monte Carlo),
5. možnost: Chceme aspoň pokus o řešení nebo optimum v některých případech rozumně rychle (heuristiky).

Útok na P- a NP-hypotézu

- Není příliš pravděpodobný,

Útok na P- a NP-hypotézu

- Není příliš pravděpodobný,
- obecně se ale pro různé problémy dají najít různé obskurní algoritmy.

Útok na P- a NP-hypotézu

- Není příliš pravděpodobný,
- obecně se ale pro různé problémy dají najít různé obskurní algoritmy.
- Příklad (vzdálený): Quicksort má složitost v nejhorším případě $\Omega(n^2)$, najdeme-li medián v lineárním čase, máme složitost $\Theta(n \log n)$.

Útok na P- a NP-hypotézu

- Není příliš pravděpodobný,
- obecně se ale pro různé problémy dají najít různé obskurní algoritmy.
- Příklad (vzdálený): Quicksort má složitost v nejhorším případě $\Omega(n^2)$, najdeme-li medián v lineárním čase, máme složitost $\Theta(n \log n)$.
- Nalezení polynomiálního algoritmu pro jakýkoliv NP-těžký problém řeší P- a NP-hypotézu.

Aproximační algoritmy

- Algoritmus nazveme k-aproximační, pokud pro každou instanci maximalizačního problému s optimálním řešením s hodnotou m nalezne řešení s hodnotou alespoň m/k, pro minimalizační problém chceme hodnotu nejvýš km.

Aproximační algoritmy

- Algoritmus nazveme k-aproximační, pokud pro každou instanci maximalizačního problému s optimálním řešením s hodnotou m nalezne řešení s hodnotou alespoň m/k, pro minimalizační problém chceme hodnotu nejvýš km.
- Aproximační algoritmy typicky bývají snadné (protože jinak by je nikdo nebyl schopný analyzovat).

Aproximační algoritmy

- Algoritmus nazveme k-aproximační, pokud pro každou instanci maximalizačního problému s optimálním řešením s hodnotou m nalezne řešení s hodnotou alespoň m/k, pro minimalizační problém chceme hodnotu nejvýš km.
- Aproximační algoritmy typicky bývají snadné (protože jinak by je nikdo nebyl schopný analyzovat).
- Příklady: Bin-packing (hladově), vertex-cover (z maximálního párování).

Aproximační schémata

- Aproximační algoritmus s parametrem α nazveme polynomiálním aproximačním schématem, pokud pro každý parametr α tvoří α-aproximační algoritmus a pohlížíme-li na α jako na konstantu, algoritmus je polynomiální.

Aproximační schémata

- Aproximační algoritmus s parametrem α nazveme polynomiálním aproximačním schématem, pokud pro každý parametr α tvoří α-aproximační algoritmus a pohlížíme-li na α jako na konstantu, algoritmus je polynomiální.
- Pokud je schéma polynomiální vůči n i $1 - \alpha$, resp. $1 - \frac{1}{\alpha}$, nazveme ho úplným polynomiálním aproximačním schématem.

Aproximační schémata

- Aproximační algoritmus s parametrem α nazveme polynomiálním aproximačním schématem, pokud pro každý parametr α tvoří α-aproximační algoritmus a pohlížíme-li na α jako na konstantu, algoritmus je polynomiální.
- Pokud je schéma polynomiální vůči n i $1 - \alpha$, resp. $1 - \frac{1}{\alpha}$, nazveme ho úplným polynomiálním aproximačním schématem.
- Pro Knapsack existuje ÚPAS a využívá se vhodného zaokrouhlení (viz algoritmus pro celočíselný knapsack z prvního ročníku).

Pravděpodobnostní algoritmy

- Využívají náhodné veličiny (náhodu je těžké získat).

Pravděpodobnostní algoritmy

- Využívají náhodné veličiny (náhodu je těžké získat).
- Lze rozdělit do dvou rodin.

Pravděpodobnostní algoritmy

- Využívají náhodné veličiny (náhodu je těžké získat).
- Lze rozdělit do dvou rodin.
- Jedny jsou nepřesné, druhé nemusejí vždy končit včas.

Pravděpodobnostní algoritmy

- Využívají náhodné veličiny (náhodu je těžké získat).
- Lze rozdělit do dvou rodin.
- Jedny jsou nepřesné, druhé nemusejí vždy končit včas.
- Monte Carlo (algoritmy rychlé, ale nepřesné):

Pravděpodobnostní algoritmy

- Využívají náhodné veličiny (náhodu je těžké získat).
- Lze rozdělit do dvou rodin.
- Jedny jsou nepřesné, druhé nemusejí vždy končit včas.
- Monte Carlo (algoritmy rychlé, ale nepřesné):
- Výpočet plochy (objemu) komplikovaného útvaru: Útvar vepíšeme do hyperkrychle správné dimenze, generujeme prvky z hyperkrychle vybrané z uniformního rozdělení a zjišťujeme, kolikrát vygenerovaný bod padl do útvaru a kolikrát ne. Využijeme zákon velkých čísel a objem určíme jako podíl počtu bodů padlých dovnitř ku počtu všech vygenerovaných bodů.

Monte Carlo

- MAX-CUT (maximální řez v grafu): Rozděl vrcholy na dvě hromádky náhodně nezávisle uniformně. Každá hrana vede napříč s pravděpodobností 1/2, tedy v průměrném případě je algoritmus 2-aproximační.

Monte Carlo

- MAX-CUT (maximální řez v grafu): Rozděl vrcholy na dvě hromádky náhodně nezávisle uniformně. Každá hrana vede napříč s pravděpodobností $1/2$, tedy v průměrném případě je algoritmus 2-aproximační.
- E3-MAX-SAT (klauzule velikosti právě 3, chceme co nejvíce klauzulí splnit). Generujeme náhodná ohodnocení, až najdeme ohodnocení s aspoň $7/8$ klauzulí splněných, zastavíme. Tento algoritmus je $8/7$-aproximační!

Monte Carlo

- MAX-CUT (maximální řez v grafu): Rozděl vrcholy na dvě hromádky náhodně nezávisle uniformně. Každá hrana vede napříč s pravděpodobností $1/2$, tedy v průměrném případě je algoritmus 2-aproximační.
- E3-MAX-SAT (klauzule velikosti právě 3, chceme co nejvíce klauzulí splnit). Generujeme náhodná ohodnocení, až najdeme ohodnocení s aspoň $7/8$ klauzulí splněných, zastavíme. Tento algoritmus je $8/7$-aproximační!
- Tento algoritmus je již pomezní s metodou Las Vegas.

Algoritmy Las Vegas

- Tyto algoritmy sice odpoví vždy správně, ale ne vždy rychle.

Algoritmy Las Vegas

- Tyto algoritmy sice odpoví vždy správně, ale ne vždy rychle.
- Příklad: Testování prvočíselnosti pomocí Malé Fermatovy věty (čísla složená typicky odhalíme rychle, může se nám ale dařit generovat samá kvadratická rezidua). Opakujeme, dokud nevyzkoušíme všechna čísla.

Algoritmy Las Vegas

- Tyto algoritmy sice odpoví vždy správně, ale ne vždy rychle.
- Příklad: Testování prvočíselnosti pomocí Malé Fermatovy věty (čísla složená typicky odhalíme rychle, může se nám ale dařit generovat samá kvadratická rezidua). Opakujeme, dokud nevyzkoušíme všechna čísla.
- Dámy (věže) a podobné figurky na šachovnici:

Algoritmy Las Vegas

- Tyto algoritmy sice odpoví vždy správně, ale ne vždy rychle.
- Příklad: Testování prvočíselnosti pomocí Malé Fermatovy věty (čísla složená typicky odhalíme rychle, může se nám ale dařit generovat samá kvadratická rezidua). Opakujeme, dokud nevyzkoušíme všechna čísla.
- Dámy (věže) a podobné figurky na šachovnici:
- Zkoušíme prvky rozmístit náhodně. Pokud si dáme pozor, abychom se nezacyklili, v nejhorším případě vyzkoušíme všechny možnosti (správně bude až ta poslední).

Algoritmy Las Vegas

- Tyto algoritmy sice odpoví vždy správně, ale ne vždy rychle.
- Příklad: Testování prvočíselnosti pomocí Malé Fermatovy věty (čísla složená typicky odhalíme rychle, může se nám ale dařit generovat samá kvadratická rezidua). Opakujeme, dokud nevyzkoušíme všechna čísla.
- Dámy (věže) a podobné figurky na šachovnici:
- Zkoušíme prvky rozmístit náhodně. Pokud si dáme pozor, abychom se nezacyklili, v nejhorším případě vyzkoušíme všechny možnosti (správně bude až ta poslední).
- SAT, KACHL: Zkoušíme náhodná ohodnocení dokud nenajdeme správné.

Třídy napěchované mezi P a NP

- ZPP – Třída problémů řešitelných pravděpodobnostním algoritmem, který odpoví správně s pravděpodobností 1 a v průměrném případě běží v polynomiálním čase.

Třídy napěchované mezi P a NP

- ZPP – Třída problémů řešitelných pravděpodobnostním algoritmem, který odpoví správně s pravděpodobností 1 a v průměrném případě běží v polynomiálním čase.
- RP – TPŘPA, který vždy končí v polynomiálním čase a pokud má odpovědět "ne", odpoví "ne" s pravděpodobností 1, pokud má odpovědět "ano", odpoví správně s pravděpodobností aspoň $1/2$.

Třídy napěchované mezi P a NP

- ZPP – Třída problémů řešitelných pravděpodobnostním algoritmem, který odpoví správně s pravděpodobností 1 a v průměrném případě běží v polynomiálním čase.
- RP – TPŘPA, který vždy končí v polynomiálním čase a pokud má odpovědět "ne", odpoví "ne" s pravděpodobností 1, pokud má odpovědět "ano", odpoví správně s pravděpodobností aspoň $1/2$.
- Všimněte si, že místo $1/2$ lze použít jakoukoliv konstantu (ostře) mezi 0 a 1.

Třídy napěchované mezi P a NP

- ZPP – Třída problémů řešitelných pravděpodobnostním algoritmem, který odpoví správně s pravděpodobností 1 a v průměrném případě běží v polynomiálním čase.
- RP – TPŘPA, který vždy končí v polynomiálním čase a pokud má odpovědět "ne", odpoví "ne" s pravděpodobností 1, pokud má odpovědět "ano", odpoví správně s pravděpodobností aspoň $1/2$.
- Všimněte si, že místo $1/2$ lze použít jakoukoliv konstantu (ostře) mezi 0 a 1.
- co-RP – doplněk třídy RP.

Třídy napěchované mezi P a NP

- RP $\cap$ co-RP $=$ ZPP.

Třídy napěchované mezi P a NP

- RP $\cap$ co-RP $=$ ZPP.
- Důkaz: Střídavě pouštíme RP a co-RP algoritmus, v každém (po sobě jdoucím) páru kroků se dobereme správné odpovědi s pravděpodobností $1/2$ (pravděpodobnost správné odpovědi tak konverguje k 1) a v průměrném případě použijeme 2 páry běhů.

Třídy napěchované mezi P a NP

- RP $\cap$ co-RP $=$ ZPP.
- Důkaz: Střídavě pouštíme RP a co-RP algoritmus, v každém (po sobě jdoucím) páru kroků se dobereme správné odpovědi s pravděpodobností 1/2 (pravděpodobnost správné odpovědi tak konverguje k 1) a v průměrném případě použijeme 2 páry běhů.
- BPP – TPŘPA, který končí vždy v polynomiálním čase a odpoví správně s pravděpodobností alespoň 2/3.

Třídy napěchované mezi P a NP

- RP $\cap$ co-RP $=$ ZPP.
- Důkaz: Střídavě pouštíme RP a co-RP algoritmus, v každém (po sobě jdoucím) páru kroků se dobereme správné odpovědi s pravděpodobností 1/2 (pravděpodobnost správné odpovědi tak konverguje k 1) a v průměrném případě použijeme 2 páry běhů.
- BPP – TPŘPA, který končí vždy v polynomiálním čase a odpoví správně s pravděpodobností alespoň 2/3.
- BPP algoritmus lze navrhnout například pro testování prvočíselnosti podle Malé Fermatovy věty.

Třídy napěchované mezi P a NP

- RP $\cap$ co-RP $=$ ZPP.
- Důkaz: Střídavě pouštíme RP a co-RP algoritmus, v každém (po sobě jdoucím) páru kroků se dobereme správné odpovědi s pravděpodobností 1/2 (pravděpodobnost správné odpovědi tak konverguje k 1) a v průměrném případě použijeme 2 páry běhů.
- BPP – TPŘPA, který končí vždy v polynomiálním čase a odpoví správně s pravděpodobností alespoň 2/3.
- BPP algoritmus lze navrhnout například pro testování prvočíselnosti podle Malé Fermatovy věty.
- Problémy těchto tříd se považují za relativně snadné.

Heuristiky I

- Pokoušíme se zaútočit na strukturu problému viditelnou člověku, ale ne stroji.

Heuristiky I

- Pokoušíme se zaútočit na strukturu problému viditelnou člověku, ale ne stroji.
- Nezřídka vznikají z pravděpodobnostních algoritmů tzv. derandomizací.

Heuristiky I

- Pokoušíme se zaútočit na strukturu problému viditelnou člověku, ale ne stroji.
- Nezřídka vznikají z pravděpodobnostních algoritmů tzv. derandomizací.
- Příklad: MAX-CUT lze derandomizovat hladově, tedy vrchol dáme do takové množiny, ve které z něj povede "napříč" větší počet hran.

Heuristiky II

- Lokální prohledávání se pokouší suboptimální řešení zlepšit lokálními modifikacemi (hledá tedy, kudy kandidát na řešení zlepšit).

Heuristiky II

- Lokální prohledávání se pokouší suboptimální řešení zlepšit lokálními modifikacemi (hledá tedy, kudy kandidát na řešení zlepšit).
- Tabu-search – brání zacyklení tím, že si znamená, kudy jsme prošli.

Heuristiky II

- Lokální prohledávání se pokouší suboptimální řešení zlepšit lokálními modifikacemi (hledá tedy, kudy kandidát na řešení zlepšit).
- Tabu-search – brání zacyklení tím, že si znamená, kudy jsme prošli.
- Simulované žíhání – zkouší (několik kroků) s nenulovou pravděpodobností i horší řešení (pokouší se vybřednout do jiné famílie řešení).

Heuristiky II

- Lokální prohledávání se pokouší suboptimální řešení zlepšit lokálními modifikacemi (hledá tedy, kudy kandidát na řešení zlepšit).
- Tabu-search – brání zacyklení tím, že si znamená, kudy jsme prošli.
- Simulované žíhání – zkouší (několik kroků) s nenulovou pravděpodobností i horší řešení (pokouší se vybřednout do jiné famílie řešení).
- Genetické algoritmy – pokoušíme se vyvinout evolučním způsobem algoritmus, který bude uspokojivě řešit problém. Používá se na problémy, o kterých nic nevíme.

Lineární programování I

- Masna vyrábí salám, do kterého přidává maso (100 Kč/kg), vodu (0,1 Kč/kg), mouku (5) a piliny (0,01). Chce dosáhnout co největšího zisku, ovšem kvůli předpisům EU smí pilin dát nejvýše tolik, co vody. Mouky nejvýše tolik, kolik masa a vody dohromady. Vody smí být nejvýše dvakrát tolik, kolik je masa a mouky. V jakém poměru má suroviny míchat?

Lineární programování I

- Masna vyrábí salám, do kterého přidává maso (100 Kč/kg), vodu (0,1 Kč/kg), mouku (5) a piliny (0,01). Chce dosáhnout co největšího zisku, ovšem kvůli předpisům EU smí pilin dát nejvýše tolik, co vody. Mouky nejvýše tolik, kolik masa a vody dohromady. Vody smí být nejvýše dvakrát tolik, kolik je masa a mouky. V jakém poměru má suroviny míchat?
- LP se zabývá problémy, které jsme schopni popsat jako soustavu lineárních nerovnic s konstantními koeficienty a jednou (taktéž lineární funkcí s konstantními koeficienty) účelovou funkcí (kterou buďto minimalizujeme, nebo maximalizujeme).

Lineární programování II

- O složitosti úlohy LP se dlouho nevědělo, jak je těžká.

Lineární programování II

- O složitosti úlohy LP se dlouho nevědělo, jak je těžká.
- Časem se ukázala elipsoidová metoda, která je schopná v polynomiálním čase (vůči délce zápisu všech koeficientů) najít řešení libovolně blízko optimu.

Lineární programování II

- O složitosti úlohy LP se dlouho nevědělo, jak je těžká.
- Časem se ukázala elipsoidová metoda, která je schopná v polynomiálním čase (vůči délce zápisu všech koeficientů) najít řešení libovolně blízko optimu.
- V praxi se však používá tzv. simplexová metoda, která polynomiální být nemusí (dokonce se může i zacyklit), typicky ale doběhne rychleji, než metoda elipsoidová.