Těžké problémy

a boj s nimi

- Převody problémů,
- těžkost a úplnost,
- způsoby nakládání s těžkými (nebo nepříjemnými) problémy.

Těžkost problémů

- Kdy řekneme, že problém A je těžší než problém B?

Těžkost problémů

- Kdy řekneme, že problém A je těžší než problém B?
- Můžeme poměřovat složitosti problémů, to je ale nepraktické,

Těžkost problémů

- Kdy řekneme, že problém A je těžší než problém B?
- Můžeme poměřovat složitosti problémů, to je ale nepraktické,
- protože o mnoha problémech toho mnoho nevíme.

Těžkost problémů

- Kdy řekneme, že problém A je těžší než problém B?
- Můžeme poměřovat složitosti problémů, to je ale nepraktické,
- protože o mnoha problémech toho mnoho nevíme.
- Jiné nápady?

Těžkost problémů

- Kdy řekneme, že problém A je těžší než problém B?
- Můžeme poměřovat složitosti problémů, to je ale nepraktické,
- protože o mnoha problémech toho mnoho nevíme.
- Jiné nápady?
- Těžší problém se pozná tak, že z řešení nějaké jeho instance jsme schopni zjistit (efektivně) řešení lehčího problému.

Smysl a význam definice

- Problém třídění je těžký vůči problému nalezení k-tého nejmenšího prvku.

Smysl a význam definice

- Problém třídění je těžký vůči problému nalezení k-tého nejmenšího prvku.
- Problém nejdelší cesty v grafu je těžký vůči problému nalezení Hamiltonské cesty (cesty procházející všemi vrcholy).

Smysl a význam definice

- Problém třídění je těžký vůči problému nalezení k-tého nejmenšího prvku.
- Problém nejdelší cesty v grafu je těžký vůči problému nalezení Hamiltonské cesty (cesty procházející všemi vrcholy).
- Proč zatím nemáme definici?

Smysl a význam definice

- Problém třídění je těžký vůči problému nalezení k-tého nejmenšího prvku.
- Problém nejdelší cesty v grafu je těžký vůči problému nalezení Hamiltonské cesty (cesty procházející všemi vrcholy).
- Proč zatím nemáme definici?
- Protože není jasné, co je to **efektivně.**

Smysl a význam definice

- Problém třídění je těžký vůči problému nalezení k-tého nejmenšího prvku.
- Problém nejdelší cesty v grafu je těžký vůči problému nalezení Hamiltonské cesty (cesty procházející všemi vrcholy).
- Proč zatím nemáme definici?
- Protože není jasné, co je to **efektivně.**
- Je hledání nejkratší cesty těžké vůči testování souvislosti grafu?

Smysl a význam definice

- Problém třídění je těžký vůči problému nalezení k-tého nejmenšího prvku.
- Problém nejdelší cesty v grafu je těžký vůči problému nalezení Hamiltonské cesty (cesty procházející všemi vrcholy).
- Proč zatím nemáme definici?
- Protože není jasné, co je to **efektivně.**
- Je hledání nejkratší cesty těžké vůči testování souvislosti grafu?
- Obvykle povolujeme tzv. polynomiální transformace, tedy smíme se zeptat na polynomiálně mnoho instancí těžkého problému a z jejich řešení máme být schopni v polynomiálním čase zjistit řešení problému původního.

Definice

Formálně tedy:

Definition

Existuje-li k, d takové, že pro každou instanci problému B velikosti n jsme schopni z řešení nejvýše kn^d instancí problému A jsme schopni v čase nejvýše kn^d zjistit řešení zadané instance problému B, řekneme, že problém B je těžký vůči problému A při polynomiálních transformacích.

Třídy problémů I

- Problémy má smysl zařazovat do tříd podle složitosti.

Třídy problémů I

- Problémy má smysl zařazovat do tříd podle složitosti.
- Třída P je třídou problémů, pro které jsme schopni v polynomiálním čase najít řešení (pokud existuje). Omezíme se však na rozhodovací problémy, kdy máme odpovědět, zda řešení existuje nebo ne.

Třídy problémů I

- Problémy má smysl zařazovat do tříd podle složitosti.
- Třída P je třídou problémů, pro které jsme schopni v polynomiálním čase najít řešení (pokud existuje). Omezíme se však na rozhodovací problémy, kdy máme odpovědět, zda řešení existuje nebo ne.
- Ekvivalentně se třída P definuje jako třída jazyků rozpoznatelných v polynomiálním čase pomocí Turingových strojů.

Třídy problémů I

- Problémy má smysl zařazovat do tříd podle složitosti.
- Třída P je třídou problémů, pro které jsme schopni v polynomiálním čase najít řešení (pokud existuje). Omezíme se však na rozhodovací problémy, kdy máme odpovědět, zda řešení existuje nebo ne.
- Ekvivalentně se třída P definuje jako třída jazyků rozpoznatelných v polynomiálním čase pomocí Turingových strojů.
- Říkali jsme si, že TS má výpočetní sílu jakéhokoliv programovacího jazyka. Formálně bychom měli provést důkaz, kdy jeden elementární krok v programovacím jazyce odsimulujeme na TS. Důkaz udělá to, že přečteme veškerá data (zapamatujeme si ve stavu ta, která použijeme), rozhodneme se, co se má udělat a data zapíšeme.

Třídy problémů II

- Má-li algoritmus běžet v polynomiálním čase, použije (celkově) nejvýše polynomiální prostor. Pokud v každém kroku tento prostor přečteme (konstantně-krát), složitost výpočtu zůstává polynomiální.

Třídy problémů II

- Má-li algoritmus běžet v polynomiálním čase, použije (celkově) nejvýše polynomiální prostor. Pokud v každém kroku tento prostor přečteme (konstantně-krát), složitost výpočtu zůstává polynomiální.
- Tedy pro polynomiální algoritmus v programovacím jazyce máme polynomiální algoritmus na TS.

Třídy problémů II

- Má-li algoritmus běžet v polynomiálním čase, použije (celkově) nejvýše polynomiální prostor. Pokud v každém kroku tento prostor přečteme (konstantně-krát), složitost výpočtu zůstává polynomiální.
- Tedy pro polynomiální algoritmus v programovacím jazyce máme polynomiální algoritmus na TS.
- Viditelně každý jednotlivý problém třídy P je těžký vůči všem ostatním při polynomiálních transformacích. Proč?

Třídy problémů II

- Má-li algoritmus běžet v polynomiálním čase, použije (celkově) nejvýše polynomiální prostor. Pokud v každém kroku tento prostor přečteme (konstantně-krát), složitost výpočtu zůstává polynomiální.
- Tedy pro polynomiální algoritmus v programovacím jazyce máme polynomiální algoritmus na TS.
- Viditelně každý jednotlivý problém třídy P je těžký vůči všem ostatním při polynomiálních transformacích. Proč?
- Najdeme řešení v polynomiálním čase bez ohledu na těžký problém (neptáme se na žádnou jeho instanci).

Třídy problémů II

- Má-li algoritmus běžet v polynomiálním čase, použije (celkově) nejvýše polynomiální prostor. Pokud v každém kroku tento prostor přečteme (konstantně-krát), složitost výpočtu zůstává polynomiální.
- Tedy pro polynomiální algoritmus v programovacím jazyce máme polynomiální algoritmus na TS.
- Viditelně každý jednotlivý problém třídy P je těžký vůči všem ostatním při polynomiálních transformacích. Proč?
- Najdeme řešení v polynomiálním čase bez ohledu na těžký problém (neptáme se na žádnou jeho instanci).
- O problému těžkém vůči všem prvkům dané třídy řekneme, že je `těžký v dané třídě`. Označíme-li dotyčnou třídu A, nazýváme takový problém A-těžkým.

Třída NP

- Třída NP je třídou problémů, u kterých jsme v polynomiálním čase schopni ověřit kandidáta na řešení.

Třída NP

- Třída NP je třídou problémů, u kterých jsme v polynomiálním čase schopni ověřit kandidáta na řešení.
- Formálně: Třída NP je třídou jazyků rozpoznatelných na nedeterministickém Turingově stroji v polynomiálním čase.

Třída NP

- Třída NP je třídou problémů, u kterých jsme v polynomiálním čase schopni ověřit kandidáta na řešení.
- Formálně: Třída NP je třídou jazyků rozpoznatelných na nedeterministickém Turingově stroji v polynomiálním čase.
- Poznámka: Omezení se na rozhodovací problémy není újmou na obecnosti, jelikož můžeme řešení zkoušet po částech hádat tak, že část zafixujeme a zeptáme se na řešitelnost zbytku.

Třída NP

- Třída NP je třídou problémů, u kterých jsme v polynomiálním čase schopni ověřit kandidáta na řešení.
- Formálně: Třída NP je třídou jazyků rozpoznatelných na nedeterministickém Turingově stroji v polynomiálním čase.
- Poznámka: Omezení se na rozhodovací problémy není újmou na obecnosti, jelikož můžeme řešení zkoušet po částech hádat tak, že část zafixujeme a zeptáme se na řešitelnost zbytku.
- Otázka: Existuje ve třídě NP nějaký těžký problém (při polynomiální transformaci)?

Třída NP

- Třída NP je třídou problémů, u kterých jsme v polynomiálním čase schopni ověřit kandidáta na řešení.
- Formálně: Třída NP je třídou jazyků rozpoznatelných na nedeterministickém Turingově stroji v polynomiálním čase.
- Poznámka: Omezení se na rozhodovací problémy není újmou na obecnosti, jelikož můžeme řešení zkoušet po částech hádat tak, že část zafixujeme a zeptáme se na řešitelnost zbytku.
- Otázka: Existuje ve třídě NP nějaký těžký problém (při polynomiální transformaci)?
- Odpověď: Těch je. Doplňující otázka: Umíme to dokázat?

Třída NP

- Třída NP je třídou problémů, u kterých jsme v polynomiálním čase schopni ověřit kandidáta na řešení.
- Formálně: Třída NP je třídou jazyků rozpoznatelných na nedeterministickém Turingově stroji v polynomiálním čase.
- Poznámka: Omezení se na rozhodovací problémy není újmou na obecnosti, jelikož můžeme řešení zkoušet po částech hádat tak, že část zafixujeme a zeptáme se na řešitelnost zbytku.
- Otázka: Existuje ve třídě NP nějaký těžký problém (při polynomiální transformaci)?
- Odpověď: Těch je. Doplňující otázka: Umíme to dokázat?
- Vychloubačná (ale pravdivá odpověď): Ano.

Třída NP

- Třída NP je třídou problémů, u kterých jsme v polynomiálním čase schopni ověřit kandidáta na řešení.
- Formálně: Třída NP je třídou jazyků rozpoznatelných na nedeterministickém Turingově stroji v polynomiálním čase.
- Poznámka: Omezení se na rozhodovací problémy není újmou na obecnosti, jelikož můžeme řešení zkoušet po částech hádat tak, že část zafixujeme a zeptáme se na řešitelnost zbytku.
- Otázka: Existuje ve třídě NP nějaký těžký problém (při polynomiální transformaci)?
- Odpověď: Těch je. Doplňující otázka: Umíme to dokázat?
- Vychloubačná (ale pravdivá odpověď): Ano.
- Nápady (pro jaký problém těžkost ukázat)?

Cíl: Cookova věta

- Jako první dokázal NP-těžkost Steve Cook pro SAT.

Cíl: Cookova věta

- Jako první dokázal NP-těžkost Steve Cook pro SAT.
- Dnes (na MFF) používáme problém KACHL (kachlování koupelny).

Cíl: Cookova věta

- Jako první dokázal NP-těžkost Steve Cook pro SAT.
- Dnes (na MFF) používáme problém KACHL (kachlování koupelny).
- Instance: Zeď koupelny s předbarveným okrajem, popis dostupných typů kachlů (kachlů každého typu je celkově dost).

Cíl: Cookova věta

- Jako první dokázal NP-těžkost Steve Cook pro SAT.
- Dnes (na MFF) používáme problém KACHL (kachlování koupelny).
- Instance: Zeď koupelny s předbarveným okrajem, popis dostupných typů kachlů (kachlů každého typu je celkově dost).
- Otázka: Lze zadanými kachly vykachlovat zadanou zeď? Kachle mají předepsanou orientaci.

Těžkost KACHLu

- Tvrzení: Každý problém třídy NP lze vyřešit v polynomiálním čase z řešení problému KACHL.

Těžkost KACHLu

- Tvrzení: Každý problém třídy NP lze vyřešit v polynomiálním čase z řešení problému KACHL.
- Důkaz (náznak): Simulujeme výpočet NTS. Předkreslený strop simuluje vstup, předpokládáme, že stroj před zastavením pásku smaže a hlavu nastaví na definované místo na pásce. Předkreslená podlaha simuluje konec výpočtu, jedna řada kachlů simuluje jeden krok TS, předkreslené sousední stěny hlídají, aby hlava nevypadla z pásky. Z vykachlování zjistíme průběh výpočtu NTS. Tudíž KACHL je těžký v NP (NP-těžký).

Těžkost SATu

- Stačí transformovat problém KACHL (protože složení polynomiálních transformací je polynomiální transformace). Zavedeme jednu proměnnou pro každý vzor kachlu a pro každé místo v kachlované stěně, tedy $\phi_{i,j,k}$ říká, zda je na pozici (i, j) kachl vzoru k a formulemi popíšeme kachlování: Na každém místě je aspoň jeden kachl, na žádném místě nejsou kachly dva a sousedi musejí být kompatibilní. Ze splňujícího ohodnocení zjistím vykachlování. Tudíž SAT je NP-těžký.

Těžkost SATu

- Stačí transformovat problém KACHL (protože složení polynomiálních transformací je polynomiální transformace). Zavedeme jednu proměnnou pro každý vzor kachlu a pro každé místo v kachlované stěně, tedy $\phi_{i,j,k}$ říká, zda je na pozici (i, j) kachl vzoru k a formulemi popíšeme kachlování: Na každém místě je aspoň jeden kachl, na žádném místě nejsou kachly dva a sousedi musejí být kompatibilní. Ze splňujícího ohodnocení zjistím vykachlování. Tudíž SAT je NP-těžký.
- Je-li problém P ve třídě A a současně je A-těžký, řekneme, že je A-úplný.

Těžkost SATu

- Stačí transformovat problém KACHL (protože složení polynomiálních transformací je polynomiální transformace). Zavedeme jednu proměnnou pro každý vzor kachlu a pro každé místo v kachlované stěně, tedy $\phi_{i,j,k}$ říká, zda je na pozici (i, j) kachl vzoru k a formulemi popíšeme kachlování: Na každém místě je aspoň jeden kachl, na žádném místě nejsou kachly dva a sousedi musejí být kompatibilní. Ze splňujícího ohodnocení zjistím vykachlování. Tudíž SAT je NP-těžký.
- Je-li problém P ve třídě A a současně je A-těžký, řekneme, že je A-úplný.
- SAT i KACHL jsou ve třídě NP, jsou tudíž NP-úplné.

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, aby každý vrchol byl ve vzdálenosti nejvýš 1 od této množiny).

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 váhu 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 váhu 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 váhu nejvýš km.
- Aproximační algoritmy typicky bývají snadné (protože jinak by je nikdo nebyl schopný analyzovat).
- Příklady: Bin-packing (hladově), vertex-cover (z maximálního párování vezmi oba konce).

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 α, 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 α, 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).

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).
- 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).
- 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).
- 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.

Heuristiky

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

Heuristiky

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

Heuristiky

- 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

- 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.
- Lokální prohledávání, tabu-search a simulované žíhání, genetické algoritmy.