Týden 12
Diskrétní kombinatorické úlohy

1

Diskrétní kombinatorické úlohy

lze řešit prohledáváním **stavového prostoru** reprezentovaného grafem (strom, **acyklický** graf).

Prohledávání = rozvinutí stavového prostoru do tzv. prohledávacího stromu s (**větším**) počtem stavů:
- počáteční stav,
- přípustné mezistavy (částečná řešení) a nepřípustné mezistavy
- přípustné koncové stavy (řešení) a nepřípustné koncové stavy.

Prohledávání definuje pořadí stavů.
Prohledávání nejdříve do hloubky (DFS)
Prohledávání nejdříve do šířky (BFS)

2

Příklady úloh

- úlohy z teorie grafů
- návrh obvodů VLSI (členění, rozmisťování, propojování)
- generování testů VLSI
- robotika, plánování trajektorie
- celočíselné a 0/1 lineární programování
- dopravní problémy (problém obchodního cestujícího TSP)
- hádanky a hlavolamy (8 a více dam, 9 kamenů...)
- síťová analýza
- časové rozvrhování (v ekonomice, OS, multiprocesorech) – *„scheduling“*

Vesměs NP-úplné úlohy, není lehké nalézt dostatečně efektivní metody řešení.

3

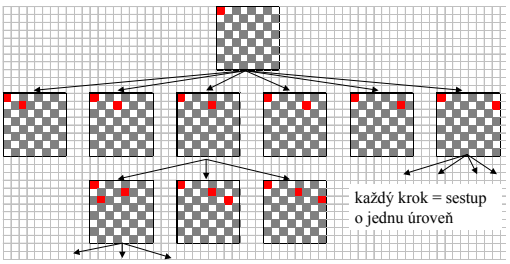
Typy úloh

1. Všechna řešení mají stejnou cenu, dosažitelná stejným počtem kroků. Hledáme první řešení.
2. Řešení (množina S) mají různou cenu, hledáme řešení s minimální cenou (cenová funkce f). Hloubka prohledávání je omezená a známá. *(Diskrétní optimalizační problémy,* DOP*)*
3. Hloubka prohledávání není známá ani omezená. Hledáme řešení v minimální hloubce prohledávacího stromu.

4

Typ 1: problém 8 královen

S pokračujícím prohledáváním stromu je na šachovnici přidáváno víc královen



5

Typ 2: Problém 0/1-lineárního programování

Vstup: int A [m][n], sloupcové vektory int b [m], c [n];
Problém: Najít sloupcový binární vektor int x [n] takový, že

$$Ax \geq b \text{ a } c^T x = \min$$

$$A = \begin{bmatrix} 5 & 0 & -2 & 3 \\ 3 & -2 & -3 & 1 \\ -2 & 2 & 0 & -1 \\ 4 & -1 & 3 & -2 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ -4 \\ -1 \\ 0 \end{bmatrix}, \quad c^T = \begin{bmatrix} -1 & 2 & 1 & -2 \end{bmatrix}$$

Např. $x = [0\ 0\ 1\ 1]^T$ vyhovuje, je to optimum?

Zde

$$S = \{x \in B^n;\ Ax \geq b\} \qquad f(x) = c^T x$$

6

Typ 3: Permutační hlavolam

Vstup: Mřížka $a \times b$, $a \geq b \geq 3$, s počáteční konfigurací ab - 1 kamenů očíslovaných 1, 2,…, ab - 1. Jedno políčko je prázdné.
Problém: Určit nejkratší sekvenci tahů, které povedou od počáteční konfigurace k uspořádání všech kamenů po řádcích 1, …, ab - 1.

S = množina všech sekvencí tahů, které vedou od počáteční ke konečné konfiguraci
$f(x)$ = počet potřebných tahů v sekvenci.

3	4	1
7	6	2
8		5

→

1	2	3
4	5	6
7	8	

Prohledávání grafů do hloubky a návraty zpět

Vyčerpávající prohledávání do hloubky: jsou vyhodnocována všechna přípustná řešení a je vybráno řešení s minimální cenou. Výpočetně neúnosné s výjimkou nejmenších problémů.

Metoda větví a mezí (*Branch and Bound search*, B&B) :
Částečné řešení odmítni a vrať se (*backtracking*)
- pokud není přípustné (nemůže vést k přípustnému řešení)
- nebo pokud je přípustné, ale nemůže vést k řešení s cenou nižší než je dosud nejlepší známá cena (zjistíme přesně nebo odhadem).

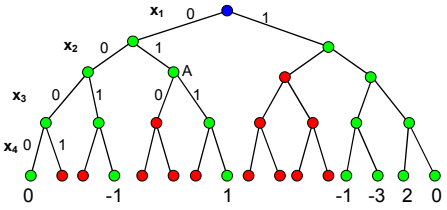
→ Ušetří se procházení mnoha podstromy.

Je-li koncový stav nepřípustný, vrať se; je-li přípustný (řešení):
- je-li třeba, aktualizuj současnou nejlepší cenu (rozhlaš ostatním)
- ukonči prohledávání, pokud je dosaženo známé dolní meze ceny
- pokračuj v prohledávání, když dolní mez ceny není známa.

Příklad: problém 8 královen - pokrač.

- **návrat:** když na řádku i+1 neexistuje žádná možná pozice pro královnu, vezmi královnu ze současné pozice na řádku i a zkus jinou dosud neprozkoumanou pozici na řádku i
- účinnější prodloužení částečných řešení: umísti královnu do toho řádku, na kterém je nejmenší počet možných pozic. V případě více řádků vezmi jeden náhodně. → počet návratů se dramaticky sníží (pro problém s 29 královnami z **1,532 210** na **313**), mnohem kratší řešení.
- **paralelizace:**
  - každému procesoru je dána jedna z 8 pozic na prvním řádku pro další průzkum (statická alokace)
  - generovaná práce je dynamicky přerozdělována mezi procesory (vyvažování zátěže, *load balancing*)

Příklad: 0/1-lineární programování - pokrač.



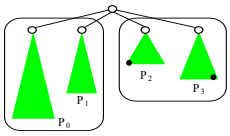
Částečné řešení A není třeba rozvíjet, protože cena $c^T x = [-1\ 2\ 1\ -2]\ [0\ \ 1\ \ *\ \ *\ ]^T$ může být nejméně 0

Proč paralelní DFS? Jaká je režie?

- Heuristické prohledávací algoritmy mohou najít suboptimální řešení pro specifické (NP-úplné) problémy v polynomiálním čase.
- Několik řešení se konstruuje nebo zlepšuje současně. Tím lze někdy dosáhnout superlineární zrychlení nebo i vyšší kvalitu suboptimálního řešení (prozkoumáním několika podprostorů)
- Mnoho DOPs vyžaduje řešení v reálném čase → paralelní zpracování může být jediná cesta jak získat výkonnost pro RT

1. Režie spojená s komunikací
2. Režie rozdělení práce
3. Soupeření o sdílené datové struktury
4. Prostoje kvůli nestejné zátěži
5. režie s detekcí globálního ukončení

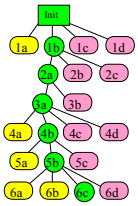
Anomálie paralelního DFS



Paralelní prohledávání může vykazovat anomální chování:
- na 2 procesorech může trvat stejně dlouho jako na 4 procesorech
- na 2 procesorech může trvat méně než 50% času sekvenčního DFS
- obecně, přidání procesoru může zrychlit prohledávání více než úměrně
- přidání procesoru může také zpomalit paralelní DFS

→ *je třeba zátěž vyvažovat dynamicky*. (LB, load balancing)

Organizace dat na zásobníku u DFS

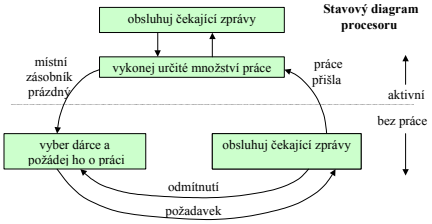


Dno zásobníku

Pro eliminaci krátkých cyklů se uchovávají rodičovské stavy nebo i již prohledané stavy jednu nebo několik úrovní zpět

Dynamické vyvažování zátěže (MP)

- Na počátku, CPU P_0 přiděluje každému CPU disjunktní část stromu
- Každý P_i, je-li aktivní, provádí DFS ve své části použitím svého zásobníku
- Když se P_i zastaví (má zásobník prázdný), žádá neprozkoumané alternativy ze zásobníku jiných CPU. P_j = dárce, P_i = příjemce.



Dynamické vyvažování zátěže

- Každý procesor uchovává data v zásobníku
- Algoritmus hledání dárce
  - cyklické žádosti, každý procesor udržuje lokální čítač mod P, index potenciálního dárce; každá žádost čítač inkrementuje
  - globální cyklické žádosti (sdílený čítač, udržuje jej 1 proces)
  - náhodné výzvy
- Je-li ve frontě k žádostí o práci a žádaný má (a bude mít) nadprahové množství práce, rozdělí zásobník postupným půlením na k+1 částí, část si nechá, další odešle.
  Algoritmus pro dělení zásobníku:
  - půlení u dna (velké podstromy)
  - půlení všech neprohledaných stavů
- Algoritmus pro ukončení paralelního DFS.
  Netriviální, zejména při dynamickém vyrovnávání zátěže.

Simulované žíhání (Simulated annealing, SA)

- = iterativní zlepšovací heuristika užívající posloupnost malých perturbací (vychýlení)
- schopnost stoupat do kopce (*hill-climbing*) zvyšující cenu umožňuje opustit horší lokálně optimální řešení
- vychýlení, které zlepšuje řešení je vždy akceptováno
- vychýlení, které zhorší stávající řešení o ΔE je akceptováno *s* pravděpodobností $e^{-\Delta E/T}$, kde T je řídící parametr analogický teplotě při žíhání fyzikálních systémů
- T je snižována po skocích: $T_{n+1} = \alpha T_n, \quad 0.9 \le \alpha \le 0.99$
- na každé teplotě je proveden předem definovaný počet vychýlení

Na počátku je T vysoká a většina vychýlení směrem do kopce je akceptována. Jak postupuje žíhání, T je snižována a vychylování do kopce bude akceptováno se stále menší pravděpodobností.

Sekvenční algoritmus simulovaného žíhání

```
SEQ
   SEQ {current_solution := initial_solution;
         current_cost := evaluate(current_solution); T:= initial_temp}
   WHILE (T > T_final)
      SEQ
         SEQ i:=1 FOR iteration (T)
            SEQ
               new_solution := move (current_solution)
               new_cost := evaluate (new_solution)
               ΔE := new_cost - current_cost
               if
                (ΔE ≤ 0 OR exp(- ΔE /T) > random ( ))
                   SEQ
                      current_solution := new_solution
                      current_cost := new_cost
         T := next_temp (T);
```


Paralelní simulované žíhání

Pracuje se paralelně na několika verzích s různými generátory náhodných čísel.

Po určité době se zjistí kdo má nejlepší řešení a s tím pokračují všichni dál.

Nebo se z několika nejlepších řešení vytvoří nová generace genetickým algoritmem
(paralelní genetické simulované žíhání).

Genetické algoritmy (GA)

- = stochastická metoda globálního prohledávání a optimalizace
- pracuje s populací potenciálních řešení
- založena na principech přírodního vývoje (přežití jedinců nejvíce fit)
- jedinci reprezentováni *řetězci, chromozomy*
- abeceda řetězců = geny (binární, int, real)
- vhodnost (fitness) i-tého jedince v populaci x je určována použitím cenové funkce $c(x_i)$
- fitness je základem pro výběr dvojic jedinců kteří se budou účastnit reprodukce - jsou vybíráni z populace s pravděpodobností úměrnou jejich relativní fitness
- fitness se odvodí z cenové funkce: fitness $f = a*c(x_i)+ b$

  max cost = 30 $\quad 0 = 30a + b$

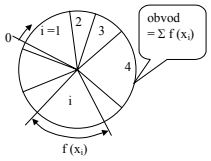
  min cost = 20 $\quad 100 = 20a + b$

  $\rightarrow f = 300 - 10 * c(x)$, rel.fitness $= f(x_i) / \Sigma f(x_i)$ [%]

19

Genetické algoritmy (GA) - pokrač.

Výběr ruletovým kolem:



1. Generuj náhodné číslo v intervalu $< 0, \Sigma f(x_i) >$
2. Najdi je na obvodu, urči i
2. Vyber chromosom (i)
3. Vyber chromosom (j)
4. Vytvoř nový chromosom rekombinací i a j

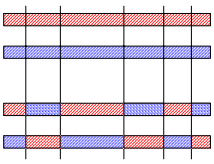
Rekombinace: dva potomci vzniknou
- jedním překřížením
- vícebodovým překřížením

Mutace - aplikována náhodně s nízkou pravděpodobností (0.001 - 0.01), změna jednoho bitu, výměna 2 hodnot atp.

20

Genetické algoritmy - pokrač.

```
SEQ  -- sekvenční algoritmus
   t = 0
   initialize P(t)  -- random gen.
   evaluate P(t)
   WHILE (NOT finished)
      SEQ
         t = t +1
         select P(t) from P(t-1)
         reproduce pairs in P(t)
```



Vícebodové překřížení
Multipoint crossover (m=5)

21

Genetické algoritmy - pokrač.

Inicializace GA:
počáteční populace je generována náhodně

Vložení nových jedinců do staré generace:
nemají-li nová a stará generace stejnou velikost, potřebujeme <u>strategii nahrazování</u> (*replacement strategy)*:
- jedinci nejméně fit jsou nahrazováni deterministicky
- nejstarší jedinci jsou rušeni
- 1 - 2 jedinci nejvíc fit jsou vždy drženi v nové generaci (tzv. elitářská strategie (*elitist strategy)*

Zakončení GA:
- po daném počtu generací
- jakmile je $\Delta f(x) < \varepsilon$

22

Paralelní genetické algoritmy

1. Globální GA - jedna populace
   GA farmář: výběr dvojic ruletou a odesílání
   GA dělník: rekombinace, mutace, vyhodnocení ceny
2. Migrační GA - několik sub-populací, čas od času nejlepší jedinci migrují mezi sub-populacemi:

```
WHILE NOT finished
   SEQ
      … Selection
      … Reproduction
      … Evaluation
      PAR
         … Send emigrants
         … Receive immigrants
```

3. Difuzní GA, sousedské GA, buňkové GA

23

Difuzní genetické algoritmy (2DT)

Také známé jako sousedské GA, buňkové GA

```
-- každý uzel ( i, j ) na 2D-toru:
WHILE NOT finished
   SEQ
      … Vyhodnoť
      PAR
         … Pošli sebe sousedům
         … Přijmi sousedy
      … Vyber partnera
      … Reprodukuj
```

Použití: vyvíjející se hardware, http://www.cellmatrix.com

24

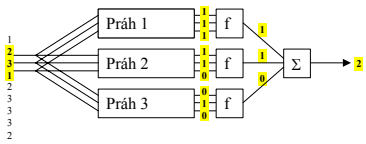
Příklad na GA: návrh zásobníkového filtru

- Aplikace: potlačení šumu v to 1D nebo 2D signálech
- zásobníkové filtry jsou adaptivní a nelineární
- užívají tzv. pozitivní bool. funkce pBf (které mohou být vyjádřeny v CNF bez negovaných proměnných, např. $f = x_1x_2 + x_2x_3 + x_3x_1$ (majorita)
- filtrační vlastnost pBf (vyhlazování obrazů) vychází z jejich monotonicity
  $x \geq y \rightarrow f(x) \geq f(y)$ $\quad [\,(x \geq y) \equiv \forall i\ (x_i = 1 \rightarrow y_i = 1)\,]$
- S, počet pBf of n = 9 proměnných:
  pro 3 × 3 filter 2D je : $2^{256} < S < 2^{511}$.
- Problém: najít pBf 9 proměnných s nejlepší filtrační schopností. Celkem je 512 součinů 9 proměnných = genů v chromozomu

$$\binom{9}{0}+\binom{9}{1}+\binom{9}{2}+\ldots+\binom{9}{9}=2^9=512$$

Zpracování 1D-signálu zásobníkovým filtrem

signál	1	2	3	1	2	3	3	3	2	0	1			
práh 1	1	1	1	1	1	1	1	1	1	0	1		f(1,1,1) =	1
práh 2	0	1	1	0	1	1	1	1	1	0	0		f(1,1,0) =	1
práh 3	0	0	1	0	0	1	1	1	0	0	0		f(0,1,0) =	0
													aritm.sum	2
výstup		2	2	2	2	3	3	3	2	1				

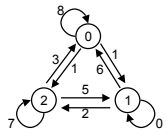


Podrobnosti návrhu zásobníkového filtru

Rozměr obrazu: 256 × 256
Okénko pro vyhodnocování fitness: nejdůležitější výřez 50 × 50 pixelů
Podrobnosti implementace GA:
- 100 chromosomů/ populaci
- 100 generací, rekombinace: jedno- a dvou-bodové překřížení
- cenová funkce vrací údaj, jak blízko je restaurovaný obraz původnímu obrázku bez šumu:
  střední absolutní chyba $\quad MAE = \frac{1}{n}\sum_{image}\left|I_0(i,j) - I_1(i,j)\right|$
- 1 vyhodnocení fitness = 0.3 s na SparcStation 10/41 (≈1 hodina/10k pixelů)
- paralelní implementace:
  globální GA + lokální gradientové hledání po bitech (sekvence mutací);
  nejlepší chromosom se hledá lokálně každých 10 generací (poprvé 30 gen.)
- COW: doba řešení pro P = 6 byla 15 min, pro P = 1 jedna hodina.

Problém nejkratších cest mezi všemi páry uzlů

Obecná formulace (orientovaný graf, plně propojený)



Graf G(V,E)

E

	0	1	2
0	8	1	1
1	6	0	2
2	3	5	7

hrana

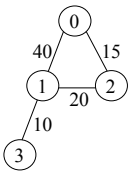
D

	0	1	2
0	4	1	1
1	5	0	2
2	3	4	6

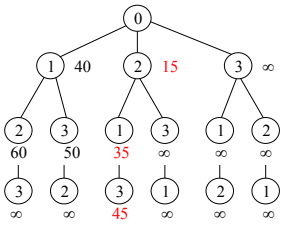
cesta

Nejkratší cesty z jednoho uzlu grafu

Neorientovaný graf

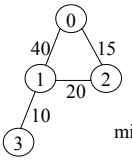


prohledávání do šířky, $O(n^3)$

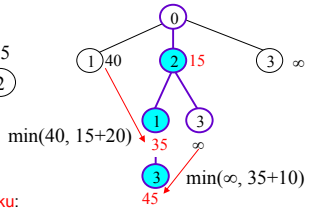


Nejkratší cesty z jednoho uzlu grafu (Dijkstrův sekvenční algoritmus)

Neorientovaný graf



Efektivní prohledávání, $O(n^2)$



Minimum v řádku:
udává určený uzel (known). Je-li vzdálenost dosud neurčených uzlů od src přes poslední určený uzel kratší, je třeba ji v dalším kroku aktualizovat.

Nejkratší cesty z jednoho uzlu grafu (src) (paralelně)

Deklarace a inicializace:

```
int n, # počet uzlů grafu; indexy uzlů: 0 (src), 1, 2, ..., n-1
    P, # počet procesorů, indexy 1, 2, ..., P
    lastKnown = 0; # index uzlu, jehož min. vzdálenost od src
                   # byla určena v předchozím cyklu
bool known[0:n-1]; # true, když min. vzdálenost uzlu od src byla
 určena, jinak false; inicializován na [true, false, false, ..., false]
int minx[0:P]; # indexy uzlů nejbližších k src nahlášené
               jednotlivými procesory
int E[0:n-1,0:n-1]; #matice hran, váha hrany mezi uzly
int D[0:n-1]; # hledaný vektor min. vzdáleností uzlů od src (0),
              inicializován na E[0,*]
```

31

Nejkratší cesty v grafu - SV program (SPMD)

```
process worker [id =1 to P] {
  npp = n/P; base = npp*(id-1);
  for [i = 1 to n-1] {# pro každý uzel kromě src
    for [j=base to base+npp-1] { # u svých neurčených uzlů
      if (Not known[j]) # aktualizuj min.vzdálenosti:
       D[j]= min(D[j],D[lastKnown]+E[lastKnown,j]);
      bariéra(id);
      ze svých zbývajících neurčených uzlů najdi stromovou redukcí
      index lokálního uzlu nejbližšího k src a ulož jej do minx[id];
      bariéra(id);
      index uzlu minx[id] s globálně nejmenší D[id] do minx[0];
      bariéra(id);
      if (id = 1) {# jen procesor 1 provede aktualizaci
        lastKnown=minx[0];
        known[lastKnown]= true;
      }
    }
```

Největší známé prvočíslo
Elektronický časopis HPCwire, 7.prosince 2001

- Nejnovější prvočíslo lze zapsat ve tvaru 2^13,466,917 -1, obsahuje 4,053,946 číslic a jeho ruční zápis by trval kolem 3 týdnů. (Na nalezení prvočísla s více než 10^7 číslic je vypsána odměna USD $100 000).
- bylo objeveno Michaelem Cameronem, 20-letým účastníkem projektu Great Internet Mersenne Prime Search (Gimps).
- 130,000 dobrovolníků z řad domácích uživatelů, studentů, škol, univerzit a podniků z celého světa přispělo do Gimps; nalezení nového prvočísla spotřebovalo 13,000 roků strojového času během dvou let nepřetržité práce.
- Mersennova prvočísla jsou důležitá pro teorii čísel a mohou napomoci při vývoji neporazitelných kódů a šifrování zpráv.

33