MI-PAA
Úkol 4 – zpráva

Tomáš Přeučil

25. prosince 2018

1 Zadání úlohy

Zadáním úlohy byl výběr a naprogramování pokročilé heuristiky pro řešení problémů, jejichž konfigurační proměnnou je bitový vektor a testování na problému batohu. Na výběr byly tři možnosti: simulované ochlazování, genetický algoritmus a tabu prohledávání.

Zvolena byla varianta s genetickým algoritmem.

2 Použité prostředky

2.1 Programovací jazyky a software

Úloha byla řešena v jazyce Python ve verzi 3.6 pod operačním systémem OS X 10.11.6. Program byl spouštěn z Bashe a tudíž pro spuštění nebylo využito žádné IDE. Pro měření času byla využita knihovna time a funkce *time.process_time()*. Jedná se o novinku od verze 3.3, která pracuje velmi podobně jako doporučovaná knihovna *timeit*.

2.2 Konfigurace testovacího stroje

Testování bylo provedeno na MacBooku Pro 13", Early 2011, modelové číslo MC700LL/A. Stroj obsahuje CPU Intel Core i5 2415M (2,3 GHz) a 16 GB RAM. Jediný další rozdíl oproti výchozí konfiguraci je vyměněný disk (za SSD), což však v tomto případě nehraje roli.

3 Rozbor algoritmu

Genetický algoritmus funguje tak, že postupně šlechtí co nejlepší jedince – v tomto případě bitové vektory.

Na začátku je nutné vytvořit počáteční populaci. To je možné udělat buď náhodně, nebo jedince „seedovat" do té oblasti stavového prostoru, kde by se mohlo nacházet řešení – u problému batohu například vkládat primárně cenné věci.

Tato populace je dále šlechtěna. Jednak se dva jedinci mohou s určitou pravděpodobností křížit, ale také mohou mutovat. Po jednom cyklu šlechtění tedy máme generaci obsahující původní rodiče, mutanty a potomky. V tuto chvíli je nutné nechat přežít pouze nejschopnější jedince.

Docílit toho je poměrně snadné – pole jedinců je seřazeno dle jejich „fitness“, což je atribut značící jejich vhodnost pro další přežití (u problému batohu například suma cen všech vložených věcí), a pole zkrátit na délku danou uživatelem.

Po té, co je vyšlechtěno x populací, je z té poslední vybrán nejlepší jedinec, který je řešením.

4 Rámcový popis postupu řešení

Algoritmus byl implementován přesně dle postupu uvedeného v předchozí kapitole. Zde uvedu pouze určité implementační detaily.

Pro uložení dat bylo použito pole objektů *chromosome*. Toto pole má velikost danou velikostí populace. Každý objekt v sobě ukládá svou fitness, kapacitu batohu, velikost batohu a pole objektů *item*. Objekt *item* v sobě ukládá present bit (zda-li je přítomen v batohu, to simuluje klasickou konfigurační proměnnou – bitový vektor), svoji váhu a cenu.

Objekt *chromosome* dále implementuje metody inicializace, mutace, křížení (druhým parametrem je jiný chromosom) a výpočtu fitness. Tím je dosaženo maximálního zapouzdření a v případě použití heuristiky pro jiný problém stačí vyměnit fitness funkci a inicializační část konstruktoru. Návrh objektu *chromosome* funguje s libovolným objektem mající atribut *present*.

Výše zmíněná inicializační část je sama o sobě velmi zajímavá. Pokud je konstruktor volán s příznakem *init = 1*, je provedena inicializace bitového vektoru. Pro problém batohu byla nejdříve vyzkoušena varianta s náhodnou inicializací, která však byla výrazně pomalejší a nedávala moc přesné výsledky. Z tohoto důvodu je bitový vektor inicializován následovně:

- Vypočti průměrnou cenu věci v batohu
- Má daná věc nadprůměrnou cenu a je její hmotnost menší než 80 % kapacity?
  - Pokud ano, s 66 % pravděpodobností nastav proměnnou tmp na 1.
  - S 33 % pravděpodobností nastav proměnnou tmp na 0
- Hoď kostkou.
  - Pokud je výsledek sudý, vlož objekt do batohu.
  - Pokud je výsledek lichý a proměnná tmp je rovna 1, vlož objekt do batohu
  - Jinak objekt do batohu nevkládej.

Dále je prováděn typický genetický algoritmus. Po mnoha experimentech se jako nejlepší hodnoty parametrů pro instance s 30-40 objekty ukázaly jako nejvhodnější parametry tyto:

- Pravděpodobnost mutace – 7 %
- Pravděpodobnost křížení – 70 %

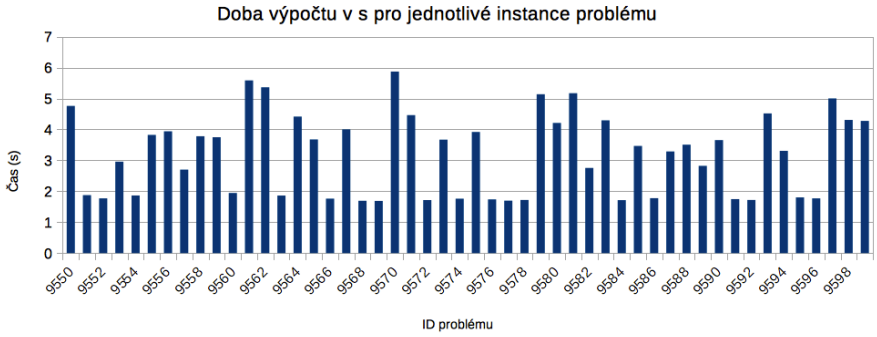
Maximální počet generací bez zlepšení byl nastaven na 20. (Což na grafu 2 není pro přehlednost zohledněno.)

Následný výstup byl zpracován pomocí awk a LibreOffice.

5 Naměřené výsledky

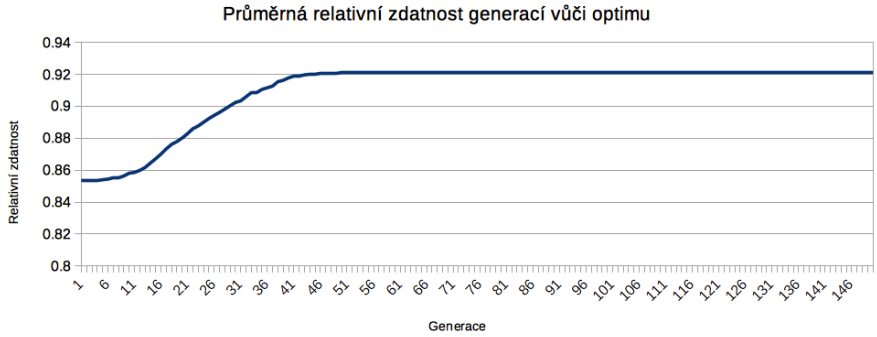
Veškerá měření byla prováděna na dodané sadě problémů o 40 věcech.

Na grafu 1 je znázorněn čas výpočtu pro jednotlivé instance problému. Průměrná doba výpočtu byla 3,2 s, což je výrazně více, než při dynamickém programování. Nejedná se však o nic nečekaného.



Obrázek 1: Časy všech výpočtů

Graf 2 znázorňuje průměrnou relativní fitness v průběhu generací. Zde byly naměřeny zdatnosti ve všech generacích u všech problémů a všechny instance byly pro danou generaci zprůměrovány. Referenčním maximem byla hodnota uvedená jako řešení v dodaném referenčním souboru. Důvodem pro vysokou počáteční fitness je způsob inicializace popsaný výše.

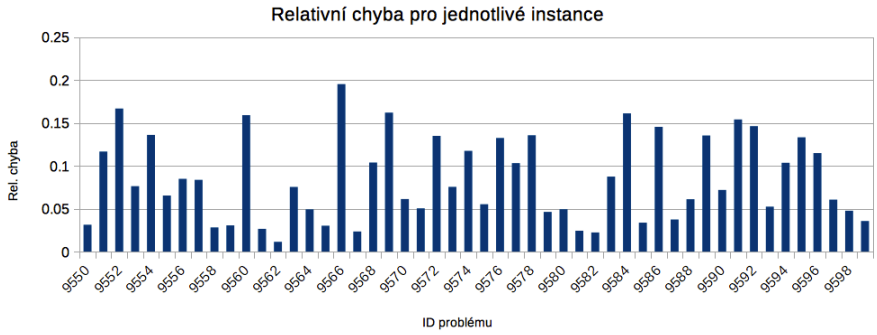


Obrázek 2: Průměrná zdatnost dle generace

Poslední měřenou veličinou byla relativní chyba. Na grafu 3 je znázorněna pro každou jednotlivou instanci problému. Průměr je 8,48 %.

Nejlepší vyzkoušená nastavení parametrů jsou uvedena v předchozí kapitole. Co se počtu generací týče, cokoli pod 40 znamenalo velmi výrazné snížení výsledné fitness. Vzhledem k maximálně 20 průchodům bez zlepšení byl zvolen limit 150 generací.

Nastavení maximální velikosti populace pod 250 značně zkreslilo výsledky. Nakonec byla zvolena hodnota 400.



Obrázek 3: Relativní chyba

6 Závěr

Byla provedena implementace a analýza genetického algoritmu, který řeší problém batohu v průměrném čase 3,2 s a průměrnou relativní chybou 8,5 %.

Algoritmus se ukázal velmi pomalý oproti například dynamickému programování, na stranu druhou je mnohem univerzálnější a řeší libovolný problém, jehož konfigurační proměnnou je bitový vektor.

Zároveň bylo pomocí objektového návrhu dosaženo vysoké míry zapouzdření a snadné přenositelnosti.

V příloze se nachází naměřené výsledky ve formátu .ods, grafy v plném rozlišení a zdrojové kódy.