# Zpráva k 4. domácímu úkolu z předmětu MI-PAA

Jan Sokol
sokolja2@fit.cvut.cz

21. prosince 2018

## Abstrakt

Zvolte si heuristiku, kterou budete řešit problém vážené splnitelnosti booleovské formule (simulované ochlazování, simulovaná evoluce, tabu prohledávání).
Tuto heuristiku použijte pro řešení problému batohu. Můžete použít dostupné instance problému, anebo si vygenerujte své instance pomocí generátoru. Používejte instance s větším počtem věcí (>30).
Hlavním cílem domácí práce je seznámit se s danou heuristikou, zejména se způsobem, jakým se nastavují její parametry (rozvrh ochlazování, selekční tlak, tabu lhůta...) a modifikace (zjištění počáteční teploty, mechanismus slekce, tabu atributy...). Není-li Vám cokoli jasné, prosíme ptejte se na cvičeních.
Problém batohu není příliš obtížný, většinou budete mít k dispozici globální maxima (exaktní řešení) z předchozích prací, například z dynamického programování. Závěrečná úloha je co do nastavení a požadovaného výkonu heuristiky podstatně náročnější a může vyžadovat zcela jiné nastavení parametrů.

## 1 Výběr jazyka

Pro svou implementaci problému batohu jsem si vybral jazyk Python. Ačkoli to je jazyk interpretovaný a nečekal jsem závratné rychlosti výpočtů, mojím výběrem byl pro to, že jsem jazyk znal a pro jakýkolik koncept je pro mne nejrychlejší.

## 2 Testovací Hardware

Všechny testy byly prováděny na cloudové linuxové instanci v AWS, běžící na Red Hat Enterprise Linux 7. Velikost instance byla: 2 Core CPU / 8 GB RAM, v názvosloví AWS **m4.large**.

## 3 Měření výpočetního času

Výpočet běhu funkce je řešen tak, že je spočten strojový čas před během funkce, a také po něm. Tyto časy jsou od sebe odečteny a je vrácen čas v ms.

```
def timing(f):
    def wrap(*args):
        time1 = time.time()
        ret = f(*args)
        time2 = time.time()
        measured_time.append(
          {'type': f.__name__,
           'time': (time2-time1)*1000.0})
        return ret
    return wrap
```

### 3.1 Popis algoritmu

Zde uvádím úryvek kódu řídící evoluci. Celý funkční kód je k dispozici v repozitáři.

Nejdříve vytvoříme počáteční populaci. Potom v n generacích běžíme nasledovné:

- vytřídíme nejlepší jedince dle fitness funkce necháme je soupeřit v turnaji,
- dle nastaveného elitismu převezmeme x jedinců z minulého kola do tohoto,
- vyplníme novou generaci novými jedinci, kteří jsou:
  - kříženci dvou náhodných minulých jedinců,
  - nebo jeden náhodný jedinec.
- mutujeme náhodné bity těchto nových jedinců
- ověříme, zda jedinec je validní (náklad batohu je menší, než je jeho kapacita).

```
# Create initial population
population = create_population(self.population_size,
                                      size)

# Run n generations
for generation in range(0, self.generations):
  sorted_population = sort_population(population)

  # Selection
  new_population = self.tournament(population,
                       self.tournament_count,
                       self.tournament_pool_size )

  # Elitsm
  del sorted_population[self.elitism_count:]

  new_population.extend(sorted_population)
  sorted_population = sort_population(new_population)

  # Fill population with new children
  while len(new_population) != self.population_size:
      child = []
      # Crossover
      if odds_are(self.xover_probability):
          in1 = self.random_individual(new_population)
          in2 = self.random_individual(new_population)

          child = self.crossover_single(in1, in2)
      else:
          # Just pick random individual
          child = deepcopy(random_individual(population)
```

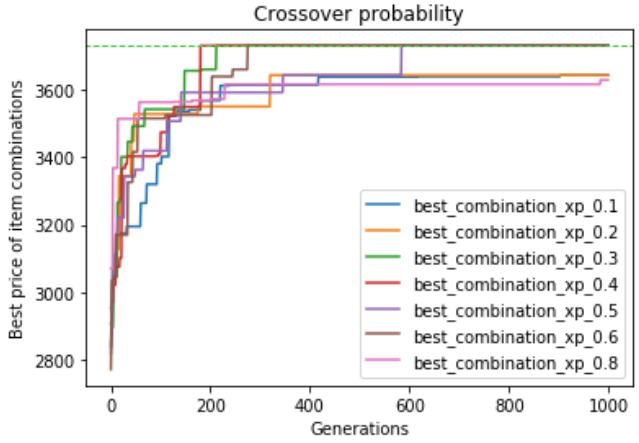
```
# Mutation
child = self.mutator_random_inverse(child,
                    self.mutation_probability)

# Check if mutated/crossed individual is valid
if self.constraint_fn(child):
    new_population.append(child)
```

## 3.2 Experimenty s nastavením parametrů

Různě jsem nastavoval parametry genetického algoritmu - pravděpodobnost křížení, pravděpodobnost mutace, elitismus, velikost turnaje, počet turnajů a počet generací. Z grafů jsem poté usoudil, které hodnoty jednotlivých parametrů jsou nejlepší.

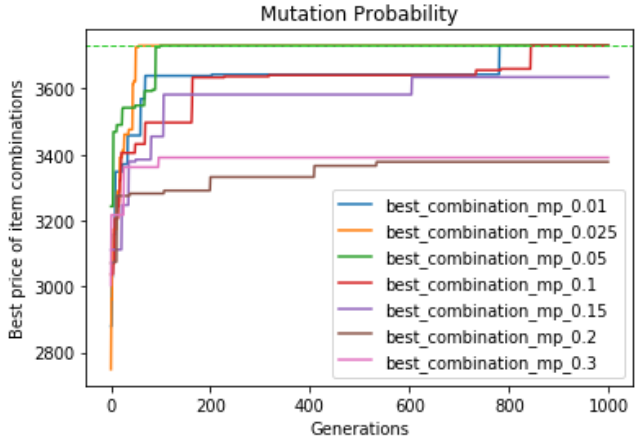
## 3.3 Pravděpodobnost křížení

Osobně hodnotím 0.3 až 0.4 jako nejlepší hodnotu pravděpodobnosti křížení. Při pravděpodobnosti 0.7 a více ani nebylo možné dosáhnout optimální hodnoty.




## 3.4 Pravděpodobnost mutace

Na grafu velmi dobře vidíme, že tato hodnota by měla být nízká. Osobně hodnotím 0.025 až 0.1 jako nejlepší hodnotu pravděpodobnosti mutace. Při pravděpodobnosti 0.7 a více ani nebylo možné dosáhnout optimální hodnoty.

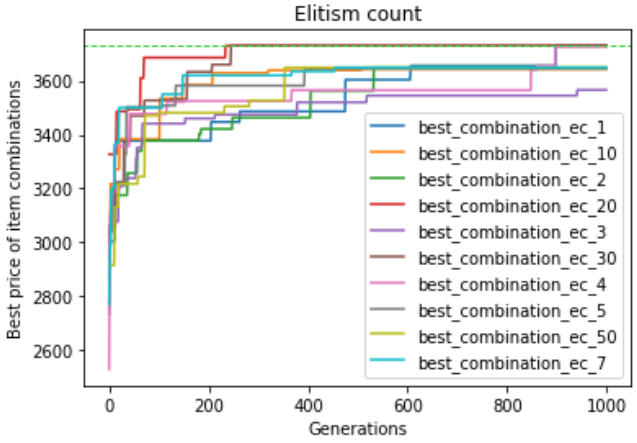



## 3.5 Elitismus

Vysoký elitismus zvýhodňuje kvalitnější jedince a znevýhodňuje slabší jedince, kteří ale v sobě mohou nést lepší informaci.

Pokud toto nastavíme na vysokou hodnotu, velmi často uvízneme v lokálním maximu/minimu.

Na grafu vidíme, že elitismus se zdá být nejlepší při hodnotě 5.



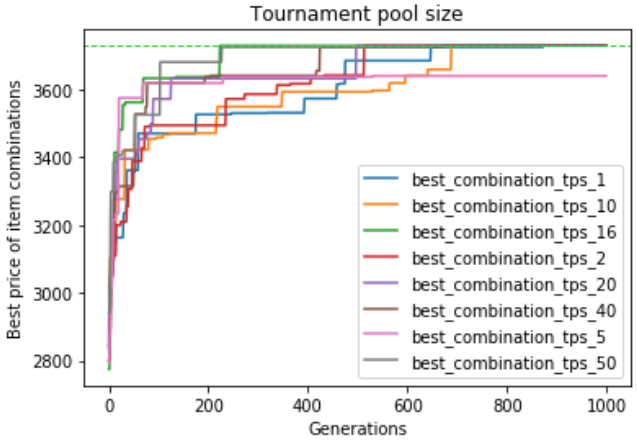

## 3.6 Velikost turnaje

Turnajem se vybírají jedinci ke křížení. Do každého turnaje vstupuje několik náhodných jedinců z celé populace.

Tento parametr se chová obdobně, jako elitimus, při velkých turnajích jsou znevýhodněni slabší jedinci.

Pokud toto nastavíme na vysokou hodnotu, velmi často uvízneme v lokálním maximu/minimu.

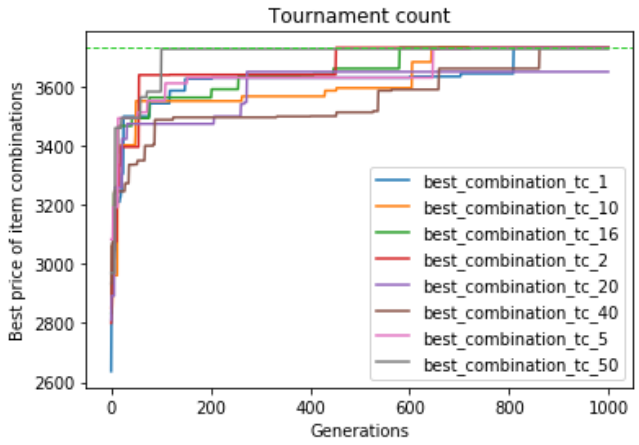
Na grafu vidíme, že velikost turnaje se zdá být nejlepší při hodnotě 2.




## 3.7 Počet turnajů

Turnajem se vybírají jedinci ke křížení. Z každého turnaje je vybírán vždy jeden, tudíž počet turnajů nepřímo určuje, kolik jedinců bude vybráno ke křížení do nové populace.

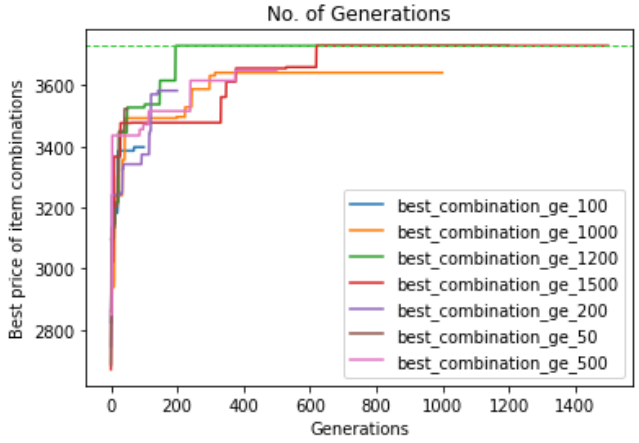
Na grafu vidíme, že počet turnajů se zdá být nejlepší při hodnotě 20 až 50.




### 3.8 Počet generací

Počtem generací jsem zjišťoval, kolik nejméně generací je potřeba, dokud nedojde ke konvergenci, pro instance s 30 předměty.

Jako moment pro ukončení evoluce jsem vybral 1000 generací, ikdyž jak vidíme níže na grafu, tato hodnota má poměrně velkou rezervu.




## 4 Shrnutí a závěr

Pokročilá iterativní metoda je nesmírně efektivní při velkých počtech předmětů v instanci.

Přesnost se pohybuje kolem 90-100% (tj, relativní chyba většinou menší, než 10%) a čas strávený výpočtem je několikařádově menší, než při použití jiných metod.

Implementace byla úspěšná a napsána pro jednoduchou výměnu batůžku na SAT.

Pro vytváření grafů bylo využito Python notebooku, který je přiložen v adresáři **report/**. Grafy jsou vykresleny pomocí knihovny **mathplotlib**.