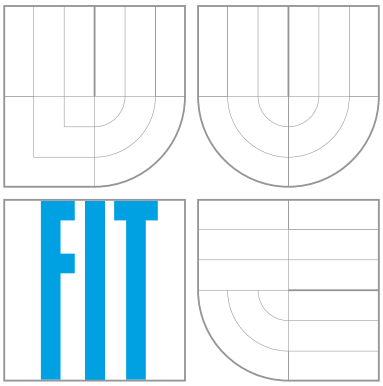


EVO Projekt

Vojtěch Šalbaba, xsalba00

1 Zadání

Cílem projektu je řešení problému obchodního cestujícího (Traveling Salesman Problem - TSP) pomocí techniky ACO.

Projekt implementuje metaheuristický algoritmus AntSystem (AS) v programovacím jazyce Ruby a porovnává dosažené výsledky s algoritmem RandomSearch (RS). Porovnává i optimalizační techniky v rámci AS a jejich vliv na dosažený výsledek.

2 Ant Colony Optimization

> [1] Ant colony optimization is a metaheuristic in which a colony of artificial ants cooperate in finding good solutions to difficult discrete optimization problems. Cooperation is a key design component of ACO algorithms: The choice is to allocate the computational resources to a set of relatively simple agents (artificial ants) that communicate indirectly by stigmergy, that is, by indirect communication mediated by the environment. Good solutions are an emergent property of the agents' cooperative interaction.

ACO je metaheuristika (heuristika pro řízení heuristik), kterou lze využít pro řešení jak statických, tak dynamických problémů. Výpočet probíhá průchodem umělých mravenců váženě náhodnou chůzí po řešení reprezentovaném úplným grafem. Agenti spolu komunikují nepřímo přes sdílené prostředí a sami jsou velmi primitivní. Synergií práce mnoha mravenců se kolonie blíží optimálnímu řešení. ACO má mnoho podvariant, dobrým přehledem je [1]. V tomto projektu jsme implementovali variantu AntSystem.

3 AntSystem

AntSystem je jedním z prvních uveřejněných algoritmů. Jedná se již o překonanou variantu. Lepšími alternativami jsou Elitist AS, Rank-Based AS nebo Ant Colony System. Všechny tyto alternativy z AS vycházejí, často jsou definovány pouze drobnou změnou vůči AS. AS slouží i jako jakýsi základ, vůči kterému se měří výkon alternativ. Kvůli této základnosti jsme zvolili pro implementaci právě AS.

AS bychom mohli zapsat následujícím pseudokódem:

```
procedure AntSystem
  inicializuj;
  while (not ukoncujici podminka) do
    sestav reseni;
    optimalizuj reseni;
    aktualizuj nejlepsi reseni;
    aktualizuj feromony;
  end-while
end-procedure
```

Konstrukce řešení

Ve fázi konstrukce řešení zadaný počet mravenců m (obecně asynchronně a nezávisle) řeší zadaný problém a každý z nich nalezne jedno (téměř jistě ne optimální) řešení procházením konstrukčního grafu G. Sada heuristických informací η_{ij} o kvalitě cesty z i do j se používá pro zlepšení kvality řešení (např. pohyb do blízkého města je v problému obchodního cestujícího obecně dobré, tah vedoucí k sebrání figur je obecně dobrý). Mravenec k vybere cestu z i do j s pravděpodobností

$$p_{ij} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{l \in S_i^k}(\tau_{il})^{\alpha}(\eta_{il})^{\beta}} \tag{1}$$

kde S_i^k je okolí vrcholu i pro mravence k. Poměr parametrů α a β ovlivňuje poměr vlivu obecných znalostí η a předchozích řešení τ na rozhodování mravenců. Fáze konstrukce řešení končí, jakmile všichni mravenci sestaví řešení.

Optimalizace nalezených řešení

Nalezená řešení se pokusíme ještě před jejich zanesením do feromonové funkce optimalizovat. Tato optimalizace je závislá na řešeném problému. Z cesty můžeme například odstranit kružnice nebo se ji pokusit vylepšit různými variantami Local Search.

Aktualizace nejlepšího řešení

Řešení nalezená v současné iteraci se porovnají s dosavadním celkově nejlepším řešením. Pokud je některé řešení lepší než celkově nejlepší řešení, použijeme ho jako nové celkově nejlepší.

Aktualizace feromonů

Fáze aktualizace feromonů proběhne ve dvou krocích, vyprchávání a ukládání. Nejdříve ze všech cest vyprchají feromony dle rovnice:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} \qquad \forall (i,j) \in H \tag{2}$$

Uvědomme si, že v AS probíhá vyprchávání až po konstrukci celé cesty. Hodnoty ρ se tedy volí vyšší, typicky 0.5. Po vyprchání feromonů z cest dojde k ukládání nových. Feromonovou funkci pro každou hranu navýšíme o feromonovou hodnotu rovnou součtu kvality cest všech mravenců, jejichž řešení hranu obsahuje.

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{3}$$

$\Delta\tau_{ij}^{k}$ označuje kvalitu cesty ij pro mravence k. Výpočet $\Delta\tau_{ij}^{k}$ volíme dle povahy problému. Kvalitu cesty volíme z intervalu $\langle 0, 1\rangle$. [1] [4]

3.1 Parametry algoritmu AS

Mezi nastavitelné parametry běhu AS patří

- počet mravenců v jedné iteraci m;
- počet iterací ;
- váhy přisuzované následování ostatních mravenců a použití metriky (α a β);
- optimalizační funkce se volí dle problému
- míra vyprchávání uložených feromonů ρ.

4 Problém obchodního cestujícího

> [3] Problém obchodního cestujícího (anglicky Travelling Salesman Problem – TSP) je obtížný diskrétní optimalizační problém, matematicky vyjadřující a zobecňující úlohu nalezení nejkratší možné cesty procházející všemi zadanými body na mapě.

Z matematického pohledu se jedná o úlohu nalezení nejkratší Hamiltonovské kružnice v grafu. Délku nalezené kružnice popisujeme cenovou funkcí. Jedná se tedy o minimalizační úlohu.

Mnoho instancí TSP bylo shromážděno v knihovně TSPLib [2]. Z této knihovny jsme vybrali problémy *berlin52* (52 vrcholový graf) a *bier127* (127 vrcholový graf). Jedná se o symetrické problémy (cesta z A do B je stejná jako z B do A) s již nalezenými optimálními řešeními. Jako cena cesty z A do B se počítá euklidovská vzdálenost mezi těmito body. Optimální řešení *berlin52* je 7542, optimálním řešením *bier127* je 118282.

5 Převod problému na instanci ACO

Převedení problému TSP na úplný orientovaný graf jsme provedli velmi přímočaře. Každé město je reprezentováno jedním vrcholem. Vzdálenosti mezi městy jsou reprezentovány ohodnocenými hranami. Cenová funkce je definována jako součet ohodnocení všech hran na dané Hamiltonovské kružnici.

6 Algoritmus RandomSearch

Algoritmus RandomSearch je nejjednodušším způsobem prohledávání stavového prostoru. Pouze se generují náhodná řešení a uchovává se nejlepší dosažené. Používá se pro rychlé nalezení slibného startovacího místa pro pokročilejší heuristiky, protože jeho výpočetní cena je zanedbatelná.

Algoritmus, který nedosáhne lepších výsledků než RandomSearch považujeme za nepoužitelný.

V pseudokódu zapíšeme RS takto:

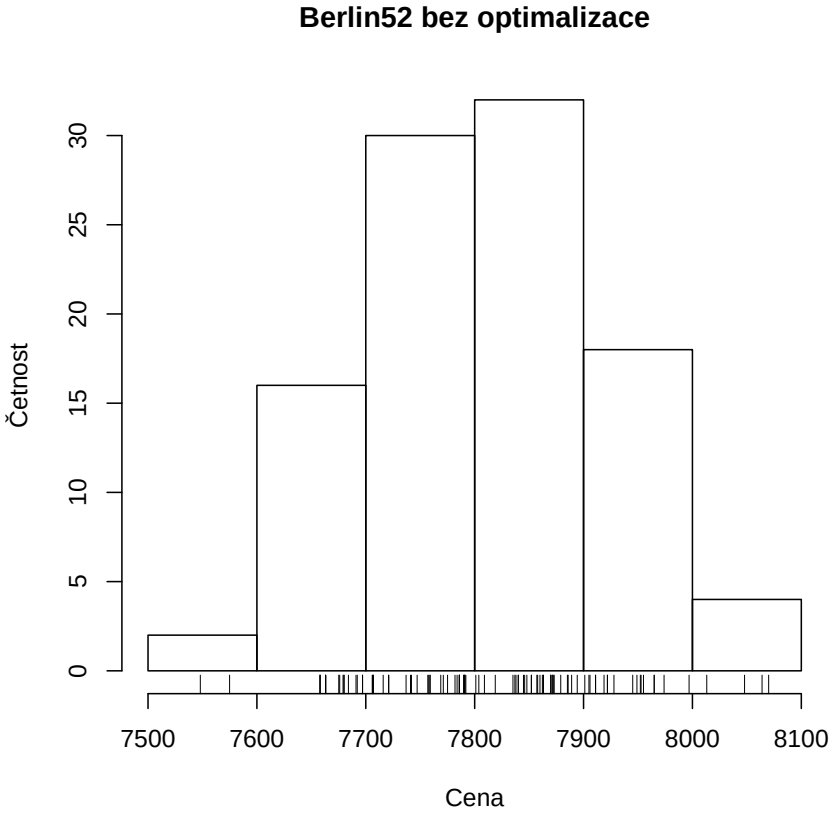
```
procedure RandomSearch
  Nejlepsi = nahodne reseni;
  while (not ukoncujici podminka) do
    kandidat = nahodne reseni;
    if Cena(kandidat) < Cena(Nejlepsi) then
      Nejlepsi = kandidat
    end
  end-while
end-procedure
```

7 Dosažené výsledky

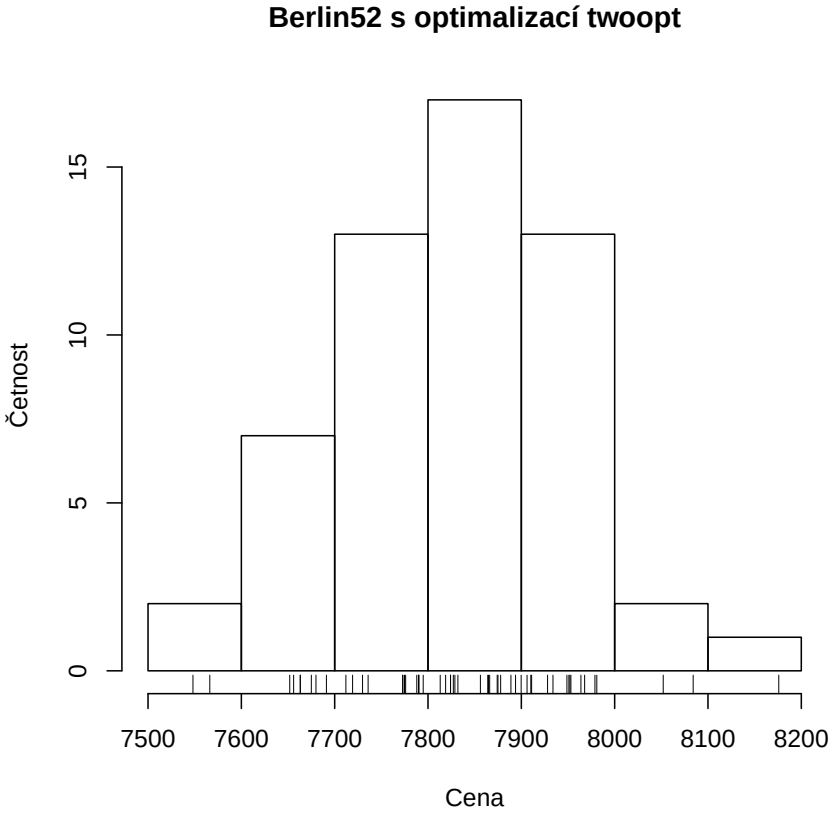
Algoritmus jsme inicializovali s 30ti mravenci a vyprcháváním feromonů nastaveným na 0,6. Učení jsme zastavili po 50ti iteracích. Dle očekávání algoritmus na hlavu poráží algoritmus RandomSearch. Zajimavým zjištěním je, že přidání optimalizace 2-opt nemělo pozorovatelný vliv na kvalitu výsledků. Jednotlivá řešení byla optimalizována, ale dle našeho názoru silný důraz na "exploitation"AS algoritmu způsobil, že vylepšení draze spočítaná optimalizační funkcí byla kolonií samovolně objevena téměř ihned během další iterace.

Optimalizační funkce měla ale výrazný vliv na dobu výpočtu, protože 2-opt má exponenciální složitost oproti lineární složitosti AS.

Ve výsledku se nám podařilo velmi přiblížit teoretickému optimu obou instancí. Výsledky jsou shrnuty v tabulkách 1 a 2. V sloupci `n` je počet experimentů. V sloupci `optimum` je poměr dosaženého výsledku k optimu instance.



Obrázek 1: Histogram výsledků algoritmu ACO pro instanci Berlin52. Optimální řešení je 7542. Pod histogramem jsou zaznačeny konkrétní výsledky. Jeden sloupec má šířku 100



Obrázek 2: Histogram výsledků algoritmu ACO pro instanci Berlin52. Optimalizováno twoopt metodou. Optimální řešení je 7542. Pod histogramem jsou zaznačeny konkrétní výsledky. Jeden sloupec má šířku 100

Variable	**n**	**Min**	**Max**	$\bar{x}$	$\tilde{x}$	**s**	**IQR**	optimum
Bez optimalizace	102	7548	8070	7814.971	7814	106.197	145.500	1.001
Two-opt	55	7548	8176	7833.964	7829	124.300	137.500	1.001
Random	157	22111	25584	24205.191	24310	620.933	836.000	2.932

Tabulka 1: Statistické parametry výsledků běhu algoritmů na instanci berlin52

Variable	**n**	**Min**	**Max**	$\bar{x}$	$\tilde{x}$	**s**	**IQR**	optimum
Bez optimalizace	30	126918	133800	130344.933	130403	1653.825	2552.500	1.073
Two-opt	20	128115	133791	130649.050	130613	1663.650	2030.500	1.083
Random	50	527701	574007	559287.720	560073	8520.502	11215.250	4.461

Tabulka 2: Statistické parametry výsledků běhu algoritmů na instanci bier127

Průběh učení algoritmu je na obrázcích 3 a 4. Za pozornost stojí prudké vylepšení během prvních 10ti iterací. V grafu vidíme na ose x číslo iterace a na ose y průměrnou cenu řešení v dané iteraci.

Rozložení dosažených výsledků vidíme na obrázcích 1 a 2. Vidíme že algoritmus poměrně konzistentně dosahuje výsledku blízko optimu.

8 Programová dokumentace

Program je rozdělen do několika jednoduchých skriptů dle účelu obsažených funkcí. Vstupním bodem je soubor as-clever_algorithms.rb. V souboru problem.rb jsou definovany TSP instance *berlin52* a *bier127* . V souboru optimizations.rb je definovan algoritmus optimalizace instance TSP. Soubor export.rb je vyhrazen pro metody pro vypis nalezenych reseni.

Proces učení se spustí následovně:

```
$ ruby as-clever_algorithms.rb [optimalizace [problem]]
```

kde `optimalizace` může nabývat hodnot `noopt twoopt` a `problem` hodnot `berlin52` `bier127`. Výchozí hodnoty pro `optimalizace` je `noopt` a výchozí hodnotou `problem` je `berlin52`.

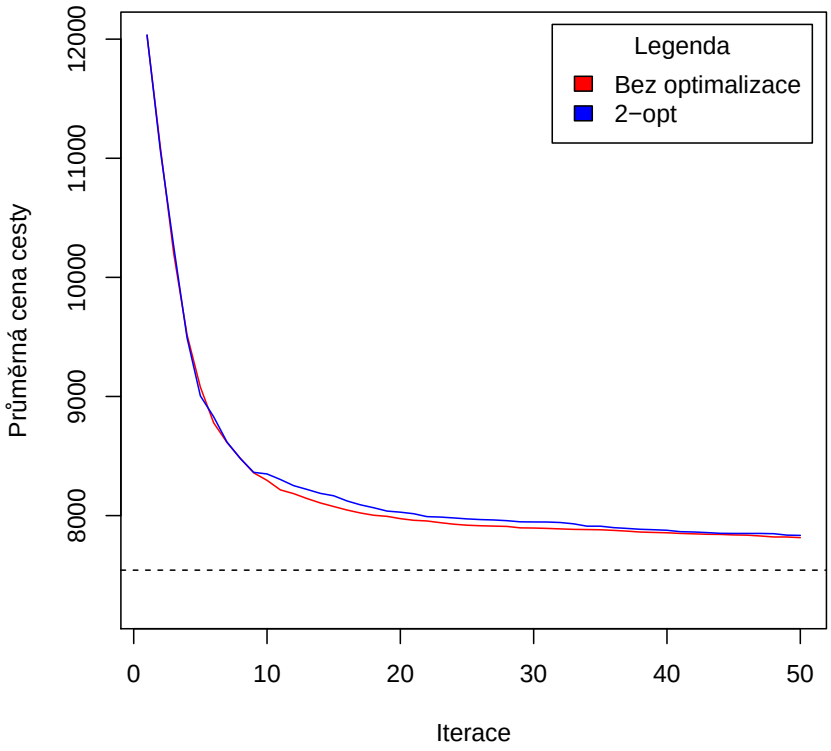
Reference

[1] Marco Dorigo, Thomas Stutzle, *Ant colony Optimization*. MIT Press, 2004.

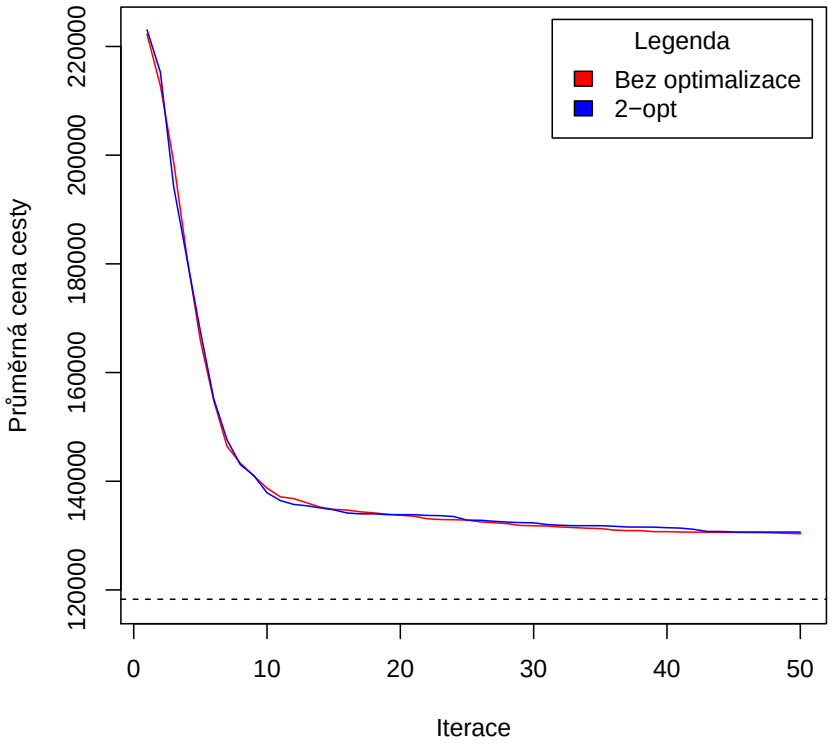
[2] TSPLib, Internetová stránka

[3] Wikipedia, Problém obchodního cestujícího

[4] Vojtěch Šalbaba *Samoučící algoritmus pro deskové hry*. Bakalářská práce, Přírodovědecká fakulta Univerzity Palackého, Katedra informatiky



Obrázek 3: Zprůměrovaný průběh učení ACO algoritmu na instanci Berlin52. Optimální řešení 7542 zaznačeno vodorovnou čarou



Obrázek 4: Zprůměrovaný průběh učení ACO algoritmu na instanci Bier127. Optimální řešení 118282 zaznačeno vodorovnou čarou