Projekt předmětu SFC

```
Autor: Birger Mark, Bc. (xbirge00)
Varianta: 94, Demonstrace činnosti ACO, Java aplikace
```

Obsah balíku:

```
./
├── aco.jar                         spustitelný .jar soubor
├── report.pdf                          technická zprava
├── report_with_images.pdf              technická zprava z obrazky (12 stran)
└── src
    ├── Main.java                   hlavni třida, inicializuje okno
    ├── MainViewController.java     ACO, události uživatelského rozhraní
    ├── Makefile
    ├── layout.fxml                 popis uživatelského rozhraní
    └── META-INF
      └── MANIFEST.MF
```

Řešeným problémem je **úloha obchodního cestujícího**. Na vstupu máme úplný graf. Na výstupu musíme vrátit nejkratší cestu, procházející vsemi uzly.

Pro řešeni problému obchodního cestujícího využil jsem algoritmy **ACO (Ant Colony Optimization)**. Implementoval jsem základní algoritmus *ant cycle*, a všechny ho modifikace, probraný v rámci přednášek: *ant density*, *ant quantity*, *elitist strategy*, *max-min ant system*, *rank-based ant system*, *ant colony system*. Ovšem, da se řešit problém obchodního cestujícího pomoci ACO jen přibližně, ale kladem jsou prakticky použitelný casy.

Implementace

Programovací jazyk projektu je Java, **verze 8**.

Rozhodl jsem implementovat grafické uživatelské rozhraní, abych znázornit zadaní vstupního grafu a zobrazeni nalezené cesty. Pro implementace grafického uživatelského rozhraní byla zvolena knihovna JavaFX 8. Výrobcem knihovny JavaFX je Orcale, proto od verze Java 8, JavaFX je součásti distribuce JRE/JDK.

Jako vývojové prostředí využíval jsem *JetBrains IDEA*, pro návrh uživatelského rozhraní jsem využil *JavaFX SceneBuilder*, který umožňuje interaktivně rozmístit prvky uživatelského rozhraní a exportovat .fxml soubor, který se da využít už uvnitř aplikace.

Během implementace došlo ke drobným změnám ant cycle algoritmu:

- cely algoritmus je rozdělen do třech části, abych vyvolávat každou novou iterace pomoci události od

grafického uživatelského rozhraní
- modifikovaný metody jsou implementovaný pomoci smyček v základním ant-cycle algoritmu

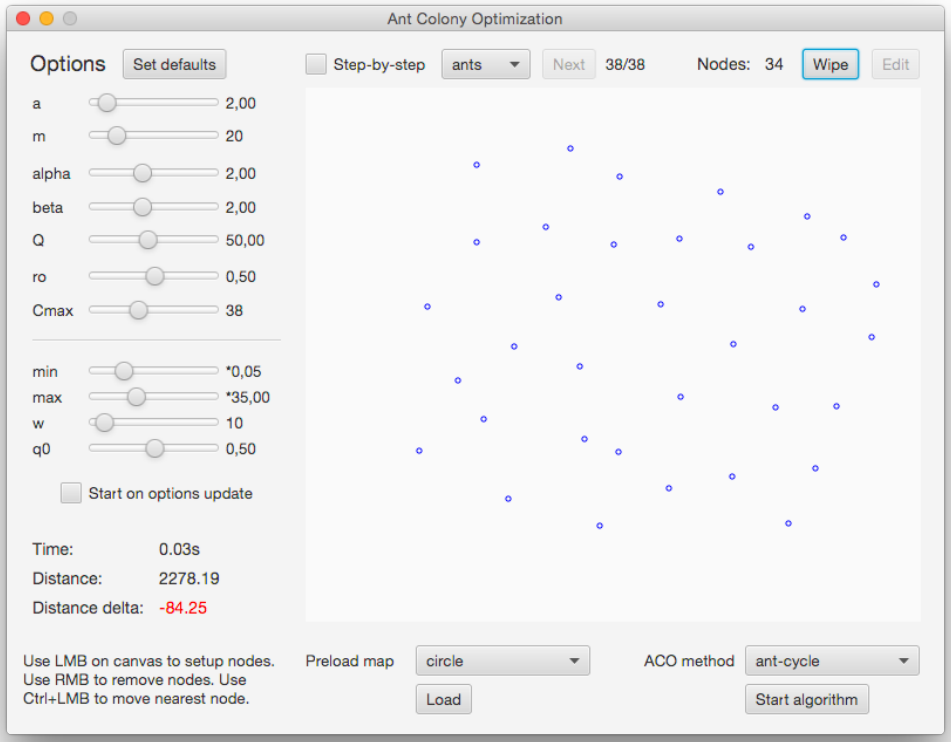
Překlad

```
cd ./src
make
make run
make clean
```

Překlad vyžaduje nainstalovaný JDK8 (`javac` , `jar` a `java`). Manuální překlad ověřen na systémech: Ubuntu 14.04, MacOSX 10.10. Spustitelný .jar soubor ověřen navic na systemech: Windows 7, Windows 10.

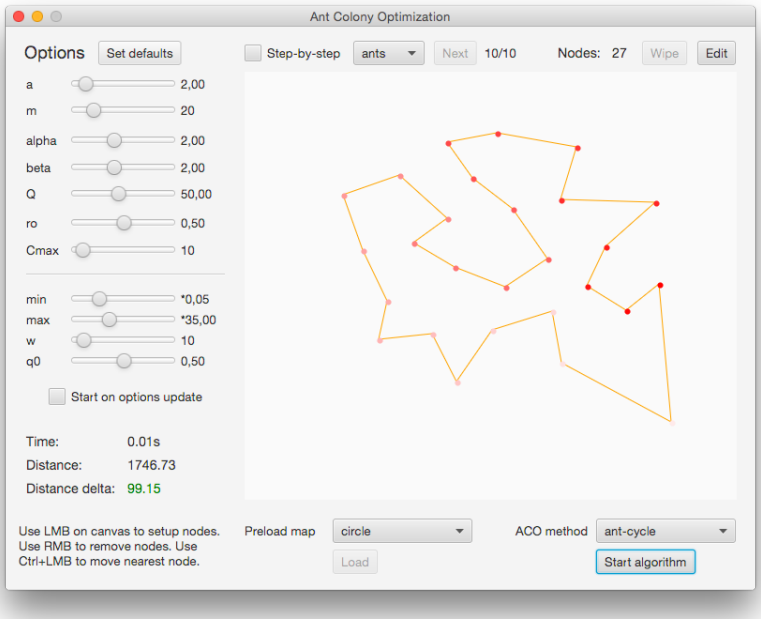
Návod na použity

Takhle vypadá uživatelské rozhraní vytvořeného programu:



Uprostřed okna umístěno plátno, které má bílé pozadí. Pomoci **levého tlačítka** myšy uživatel může přidávat uzly (modré body v režimu úpravy grafu). Pomoci **pravého tlačítka** odstraňuje se nejbližší uzel. Pomoci **levého tlačítka** mysy a současné stisknuté klávesy `Ctrl` posune se nejbližší uzel na aktuální pozici.

Abych zpracovávat graf, je potřeba přidat alespoň dva body. Pote uživatel může stisknout tlačítko `Start algorithm` v dolním pravém uhlu. Pak program zobrazí vysledek.



Na platně se zobrazí cesta oranžovou barvou. Kazdy uzel označí se růžovou barvou různé intenzity. Intenzita růžové barvy označuje nejlepší cestu mravence (od světlo-růžové až k červené). Tohle potřeba, abych zvýraznit nalezenou cestu v případě velkého (100+) poctu uzlu.

Po spuštěni algoritmu v levé dolní části okna obnoví se 3 parametry:

- čas, který trval vypočet *(Time)*
- délka nalezené cesty *(Distance)*
- rozdíl mezi, předchozí délkou nalezené cesty *(Distance delta)*

Poslední parametr je užitečný při porovnaní různých konfigurace algoritmu.

Uživatel může zvolit potřebnou modifikace algoritmu v přepínače ACO method. Dostupné metody jsou: *ant-cycle, ant-density, ant-quantity, elitist-strategy, max-min-as, rank-based-as, acs*.

Abych vrátit se do režimu opravy grafu je potřeba stisknout tlačítko `Edit` v pravém horním uhlu. Když uživatel se nachází v režimu editace grafu, může smazat všechny uzlu pomoci tlačítka `Wipe`. V dolní části okna umístěn přepínač `Preload map`. Uživatel může zvolit jednu z dostupných map:

- kružnice *(circle)*
- náhodné umístěni 20ti uzlu *(random20)*
- náhodné umístěni 100 uzlu *(random100)*
- úsměv *(smile)*
- rozmístěni velkých mest CR *(czech)*

Tlačítko `Load` smaže aktuální graf a nahradí ho přednastaveným způsobem. Velmi užitečným způsobem naplňovaní grafu je náhodné umístěni uzlu.

V levé části okna jsou umístěny různý "slidery", který umožňuj nastavit parametry algoritmu. Po navedeni myši na každý parametr zobrazí se nápověda s významem tohoto parametru. Nejdůležitějšími parametry jsou **m** (počet mravenců) a **Cmax** (počet iteraci). Tyhle hodnoty mají nejvyšší vliv na kvalitu výsledné cesty.

Pod čarou umístěny parametry pro rozšířené metody (*max-min*, *rank-based* a *ACS*). *Alpha* a *beta* pro experimentální účely mohou nabývat hodnot mene než jenda, což má zajímavý záporný vliv na výsledek. Vedle hodnot *min* a *max* jsou uveden symbol *, tyhle hodnoty opravdu 100x menší, než napsáno. Interval pro* min *a* max* parametry byl zjištěn experimentálně.

Uživatel může nastavit hodnoty na stejné jako po spuštěni programu pomoci tlačítka `Set defaults` . Označeni (Checkbox) `Start on options update` je užitečné pro porovnaní různých vstupních parametru. V tomto režimu, po uvolněni "slideru", algoritmus spustí se automaticky.

Taky důležité i označeni (checkbox) `Step-by-step` nahoře. Po spusteni algoritmu v tomto režimu, program umožňuje podívat na cesty mravenců a intenzitu feromonových stop na každé iterace. Posun do další iterace provádí se pomoci tlačítka `Next` . Přepínač (ants/pheromone) přepíná zobrazeni cest mravenců a feromonových stop.

Jednotlivé barvy odpovídají jednotlivým mravencům. V případě zobrazovaní feromonových stop, tmavší černa barva označuje vetší počet feromonu.

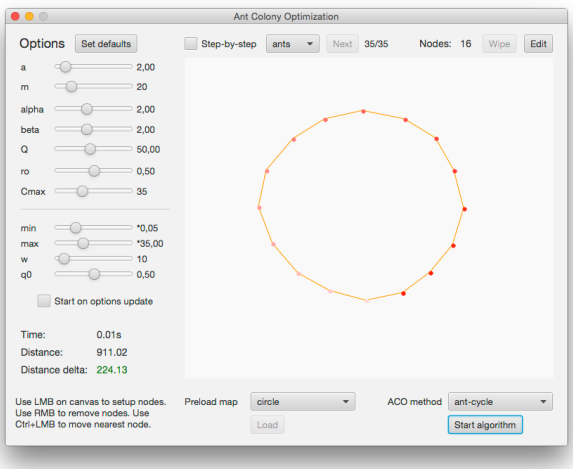
Vizualizace

V případě zobrazeni cest mravenců, narazil jsem na problém zobrazeni velkého poctu dat na malém platně. Mel jsem různý varianty zobrazovaní.

Na konci jsem přisel k speciálnímu algoritmu, který obarvuje jednotlivé cesty a inteligentně posouvá cesty pro znázorněni.

Ověřovaní funkčnosti

Jsem ověřil správnost nalezených cest pomoci grafu, body kterého tvoří kružnice:



Všichni metody nalezli tuto nejkratší cestu, feromonový stopy taky mají vetší hodnoty na okrajových hranách:

Výsledky

Navrhnul jsem cestu, kolem které jsem umístil uzlu grafu. Dole umístěny intenzity feromonových stop pro různý metody na 11te iterace pro stejný graf. Vyhodnotil jsem metody podle "**čistoty**" feromonových stop a zbytečných **lokálních extrému**.

Ant Cycle

Referenční výsledek, střední "čistota" grafu.

Ant-density

Vidíme, jak náhodné dlouhé cesty taky přidaný do feromonových stop (není děleni Lk).

Ant-quantity

Délka hrany ovlivňuje, a význačné posiluje velmi malý hrany. Ovsem, graf je čistější než u Ant Cycle.

Elitist strategy

Velmi cisty graf, ovsem vidíme lokální extrémy na rozdíl od Ant Colony System.

Max-min Ant System

Max-min nedokázal dobry výsledky a vyžaduje přesnou manuální konfiguraci.

Rank-based Ant System

10 lepších ze 20. Graf není cisty, vidíme neoptimální cesty z počátečních iteraci. Pravděpodobně tato metoda vyžaduje vetší počet iteraci.

15 lepších ze 20. Výsledek je lepší při stejném poctu iteraci, ale už připomíná Ant Cycle.

Ant colony system

"Cisty" graf, neobsahuje lokální extrémy na rozdíl od Elitist strategy. Nejlepší metoda.

Výsledná aplikace:

- ověřena
- znázorňuje klady a zápory různých metod ACO
- může byt využita pro demonstrace/vyuku metod ACO
- vizuálně demonstruje algoritmy na každé iteraci