

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 obrázky
└── src
    ├── Main.java         hlavní třída, 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 všemi uzly.

Pro řešení 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 pomocí ACO jen přibližně, ale kladem jsou prakticky použitelné casy.

Implementace

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

Rozhodl jsem implementovat grafické uživatelské rozhraní, abych znázornit zadání vstupního grafu a zobrazení nalezené cesty. Pro implementace grafického uživatelského rozhraní byla zvolena knihovna JavaFX 8. Výrobce knihovny JavaFX je Oracle, proto od verze Java 8, JavaFX je součástí 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:

- celý algoritmus je rozdělen do třech částí, abych vyvolávat každou novou iterace pomocí

události od grafického uživatelského rozhraní

- modifikované metody jsou implementovány pomocí smyček v základním ant-cycle algoritmu

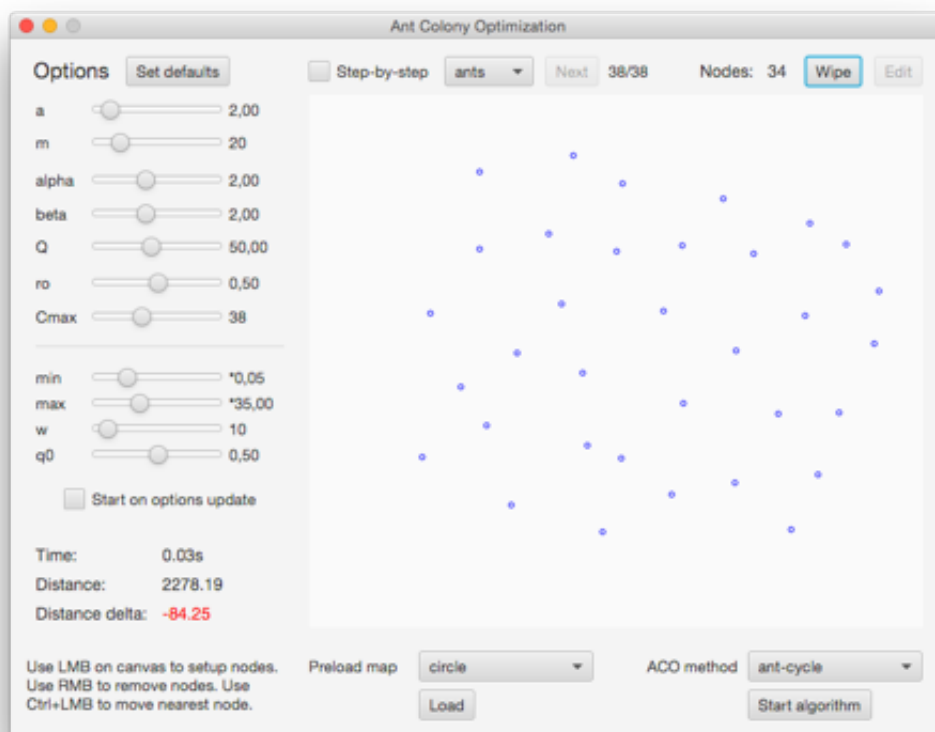
Překlad

```
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 navíc na systémech: Windows 7, Windows 10.

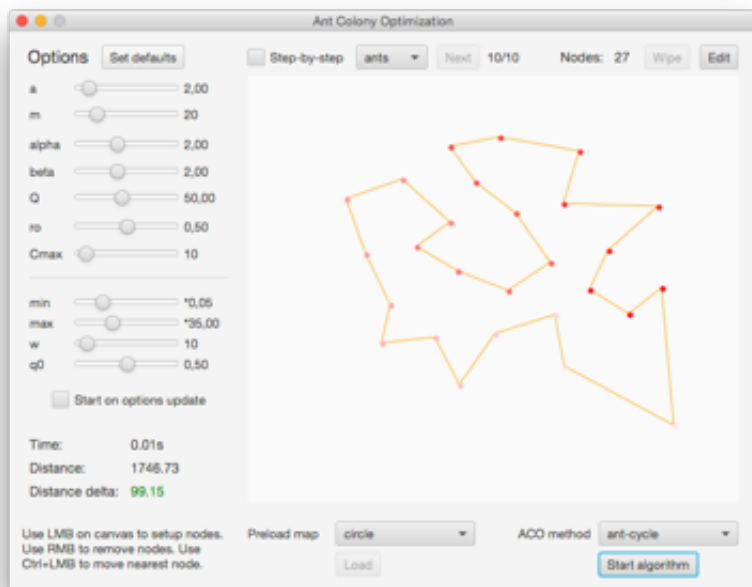
Návod na použití

Takhle vypadá uživatelské rozhraní vytvořeného programu:



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

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



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

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

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

Poslední parametr je užitečný při porovnání různých konfigurací algoritmu.

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

Abych vrátil 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 uzly pomocí tlačítka **Wipe**. V dolní části okna umístěn přepínač **Preload map**. Uživatel může zvolit jednu z dostupných map:

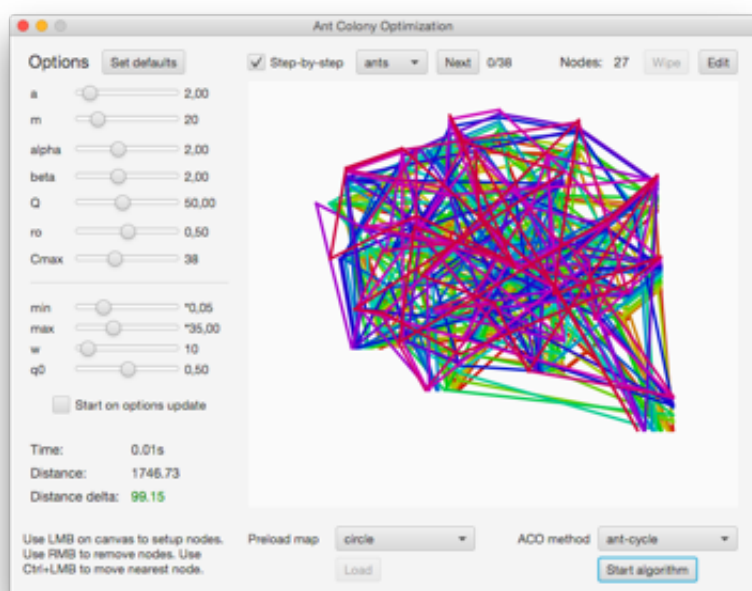
- kružnice (*circle*)
- náhodné umístění 20ti uzlu (*random20*)
- náhodné umístění 100 uzlu (*random100*)
- úsměv (*smile*)
- rozmístění velkých měst ČR (*czech*)

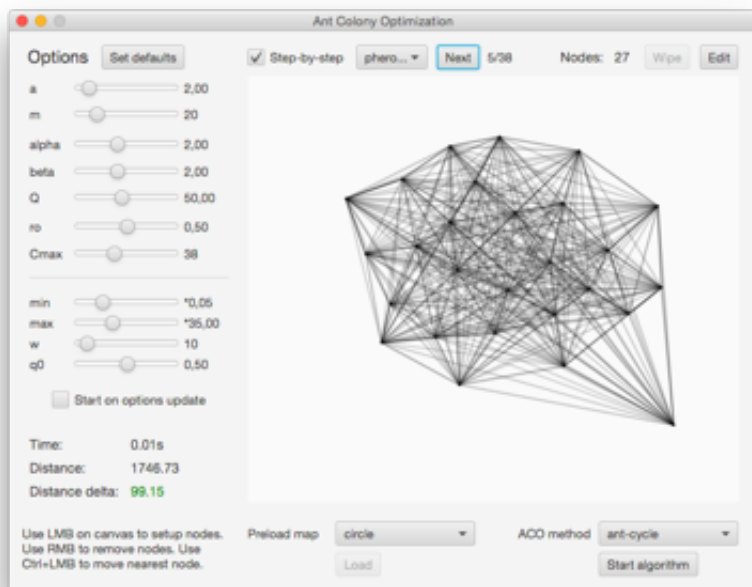
Tlačítko **Load** smaže aktuální graf a nahradí ho přednastaveným způsobem. Velmi užitečným způsobem naplňování grafu je náhodné umístění uzlu.

V levé části okna jsou umístěny různé "slidery", který umožňují nastavit parametry algoritmu. Po navedení myši na každý parametr zobrazí se nápověda s významem tohoto parametru. Nejdůležitějšími parametry jsou **m** (počet mravenců) a **Cmax** (počet iterací). 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 menší než jedna, což má zajímavý záporný vliv na výsledek. Vedle hodnot *min* a *max* jsou uvedeny symboly, tyhle hodnoty opravdu 100x menší, než napsáno. Interval pro *min* a *max* parametry byl zjištěn experimentálně.

Uživatel může nastavit hodnoty na stejné jako po spuštění programu pomocí tlačítka **Set defaults**. Označení (Checkbox) **Start on options update** je užitečné pro porovnání různých vstupních parametrů. V tomto režimu, po uvolnění "slideru", algoritmus spustí se automaticky.

Taky důležité i označení (checkbox) **Step-by-step** nahoře. Po spuštění algoritmu v tomto režimu, program umožňuje podívat na cesty mravenců a intenzitu feromonových stop na každé iteraci. Posun do další iterace provádí se pomocí tlačítka **Next**. Přepínač (ants/pheromone) přepíná zobrazení cest mravenců a feromonových stop.

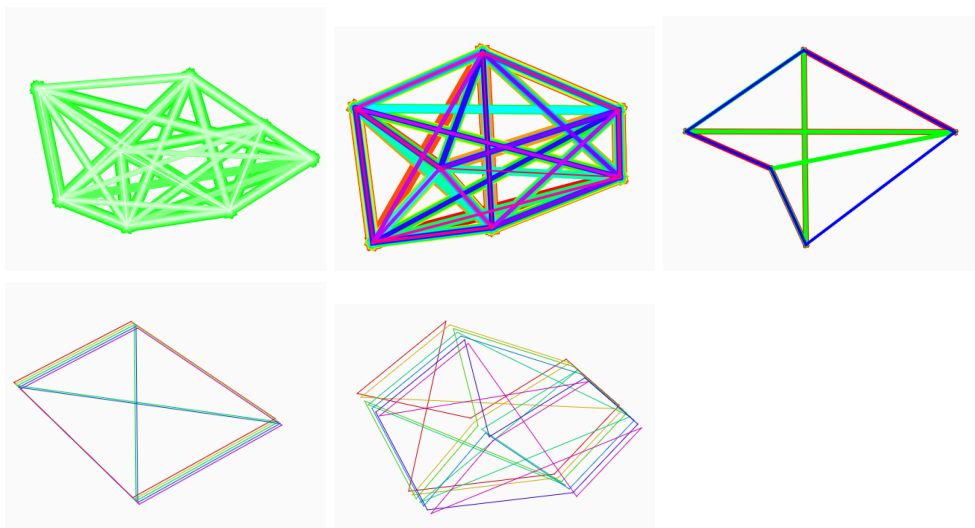




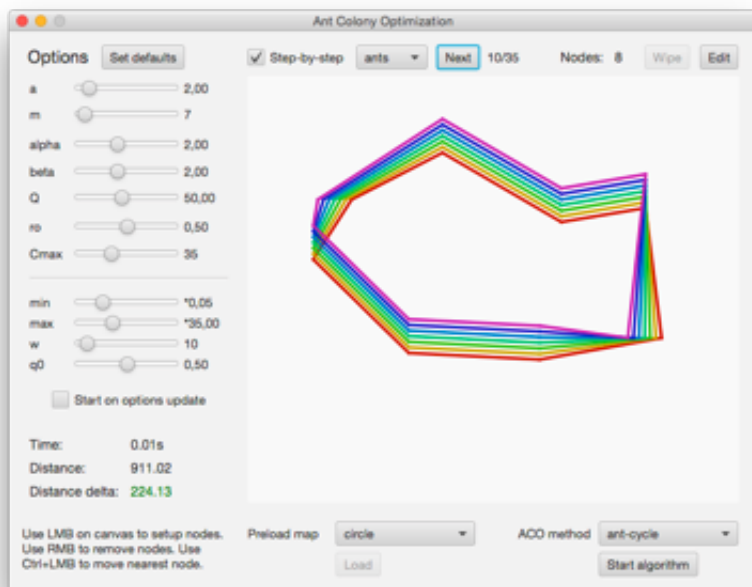
Jednotlivé barvy odpovídají jednotlivým mravencům. V případě zobrazování feromonových stop, tmavší černá barva označuje větší počet feromonu.

Vizualizace

V případě zobrazení cest mravenců, narazil jsem na problém zobrazení velkého počtu dat na malém plátně. Mel jsem různé varianty zobrazovaní:

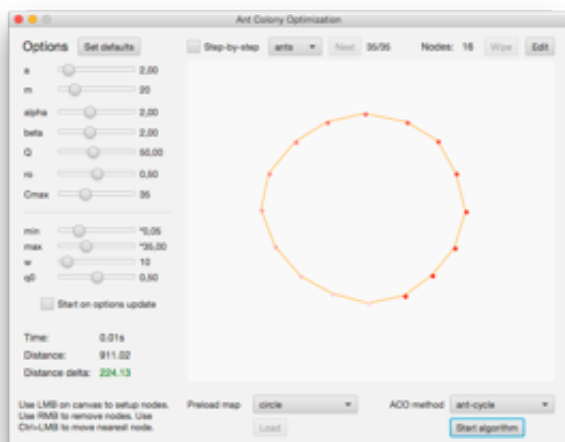


Na konci jsem přišel k speciálnímu algoritmu, který obarvuje jednotlivé cesty a inteligentně posouvá cesty pro znázornění.

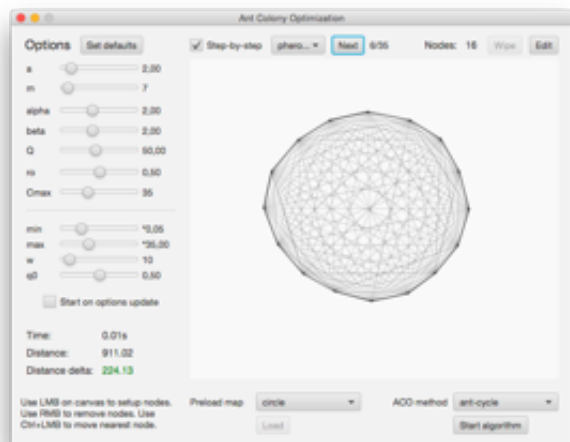


Ověřování funkčnosti

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



Všichni metody našli tuto nejkratší cestu, feromonové stopy taky mají větší hodnoty na okrajových hranách:

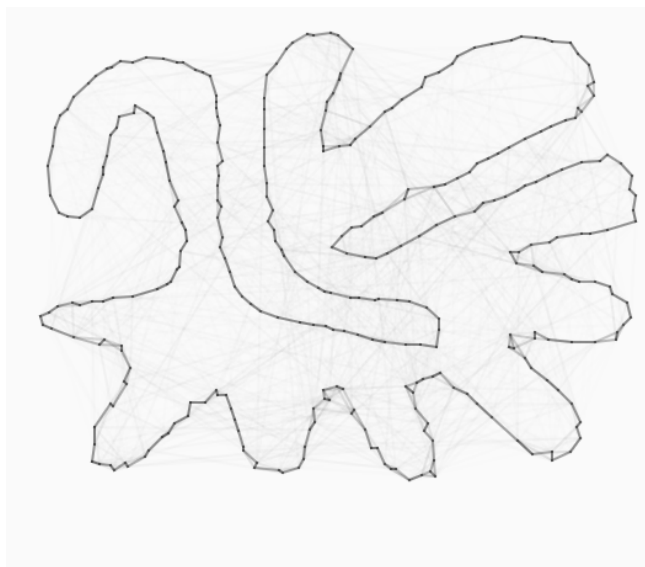


Výsledky

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

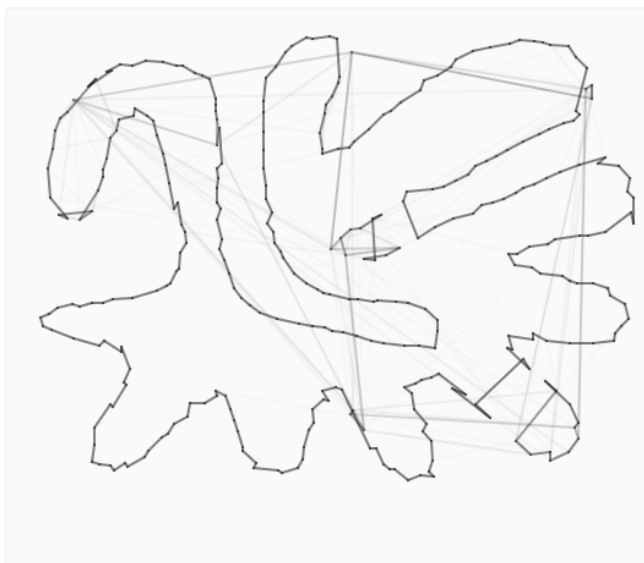
Ant Cycle

Referenční výsledek, střední "čistota" grafu.



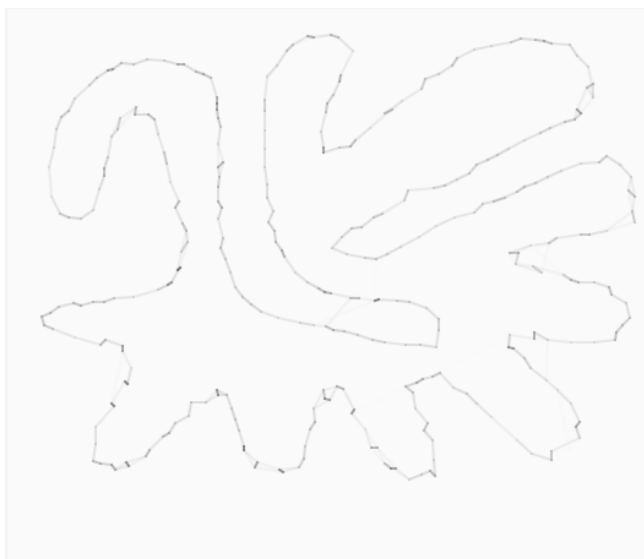
Ant-density

Vidíme, jak náhodné dlouhé cesty taky přidány do feromonových stop (není dělení Lk).



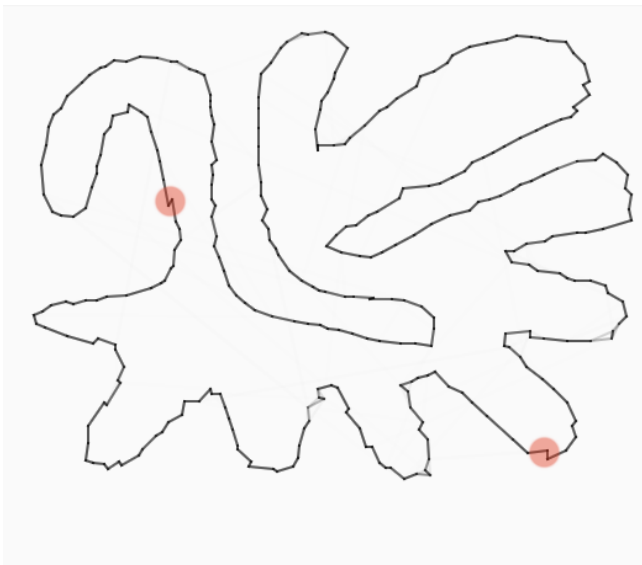
Ant-quantity

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



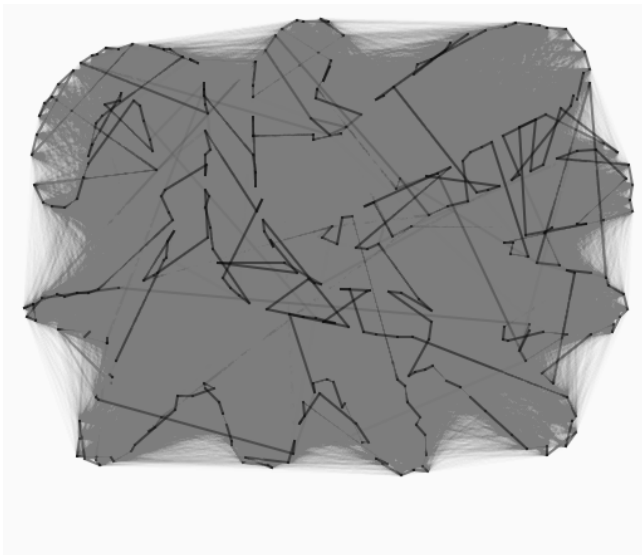
Elitist strategy

Velmi cistý graf, ovšem vidíme lokální extrémy na rozdíl od Ant Colony System.



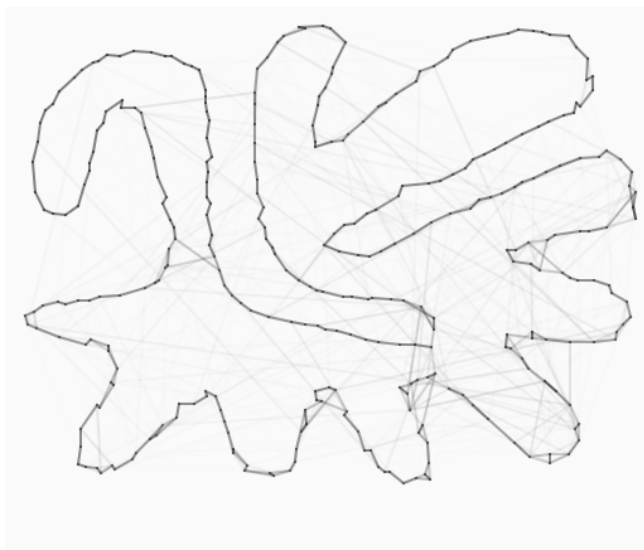
Max-min Ant System

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

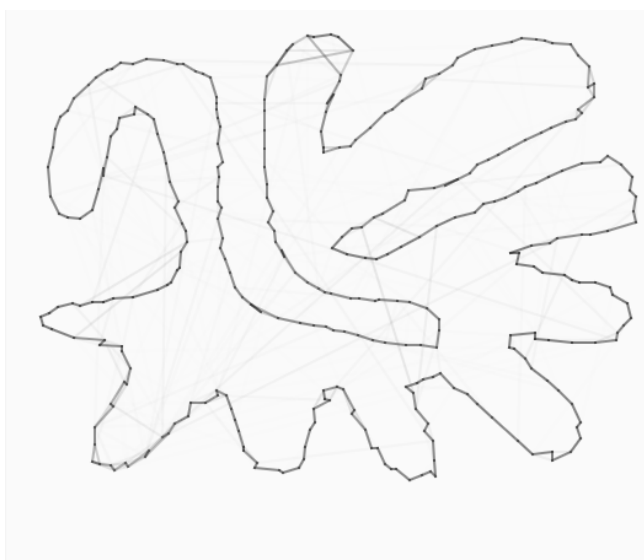


Rank-based Ant System

10 lepších ze 20. Graf není čistý, vidíme neoptimální cesty z počátečních iterací. Pravděpodobně tato metoda vyžaduje větší počet iterací.

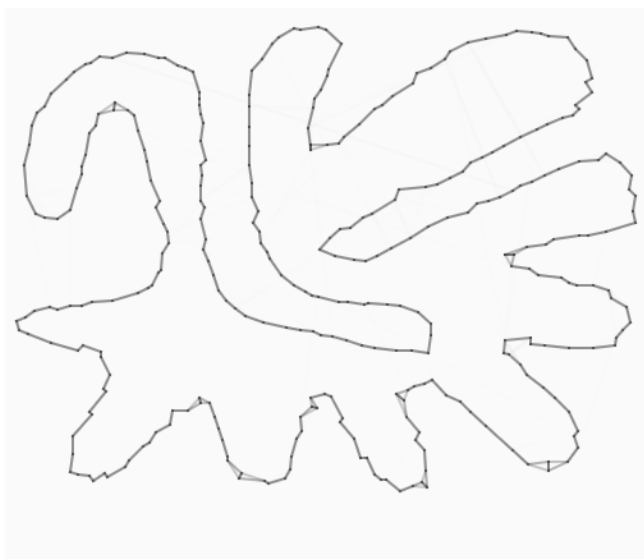


15 lepších ze 20. Výsledek je lepší při stejném počtu iteraci, ale už připomíná Ant Cycle.



Ant colony system

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



Výsledná aplikace:

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