

Dokumentacja  
Problem Komiwożera  
na NetBeans Platform

Bartek Bułat  
Konrad Malawski  
Michał Nowak

08.06.2010

## Spis treści

<b>1</b>	<b>Problem komiwojażera</b>	<b>3</b>
1.1	Problem komiwojażera jako NP-zupełny . . . . .	3
1.2	Podejście algorytmiczne, a heurystyczne . . . . .	3
<b>2</b>	<b>Rozwiązanie problemu komiwojażera przy użyciu algorytmów ewolucyjnych</b>	<b>4</b>
2.1	Chromosomy . . . . .	4
2.2	Losowanie populacji początkowej . . . . .	4
2.3	Kryterium oceny osobników . . . . .	4
2.4	Metody selekcji . . . . .	4
2.5	Proces krzyżowania . . . . .	5
2.6	Mutowanie . . . . .	5
2.7	Warunek stopu . . . . .	5
<b>3</b>	<b>Parametry programu</b>	<b>6</b>
<b>4</b>	<b>Wykorzystane technologie/frameworki</b>	<b>8</b>
4.1	NetBeans Platform . . . . .	8
4.2	Prefuse . . . . .	8
4.3	GraphML . . . . .	9
4.4	JFreeChart . . . . .	9
<b>5</b>	<b>Kod źródłowy oraz licencja</b>	<b>9</b>
<b>6</b>	<b>Bibliografia</b>	<b>9</b>

# 1 Problem komiwojażera

Zagadnieniem naszego projektu był jeden z najstarszych problemów związanych z teorią grafów – problem komiwojażera (Traveling Salesman Problem). Tematem tego zagadnienia patrząc od strony teorii grafów jest znalezienie minimalnego cyklu Hamiltona w  $k$ -klicie, gdzie podane są wagi wszystkich krawędzi. Problem ten obrazowany jest z punktu widzenia komiwojażera (akwizytora) który chce odwiedzić wszystkie miasta, przy czym w każdym chce być dokładnie raz i nie chce dwa razy chodzić tą samą drogą. Zna on odległości pomiędzy każdymi dwoma miastami i wszystkie miasta są ze sobą połączone. Istnieje bardziej skomplikowana wersja tego problemu – wersja asymetryczna. Problem tamten różni się tym, że droga z pierwszego do miasta drugiego może mieć inną długość, niż droga powrotna. W naszym projekcie skupiliśmy się jednak na bardziej popularnej i standardowej wersji tego problemu grafowego.

## 1.1 Problem komiwojażera jako NP-zupełny

Zagadnienie związane z naszym projektem w teorii grafów, przy podejściu decyzyjnym uznane jest za należące do klasy NP-zupełnych. Należenie do klasy NP oznacza, że w czasie wielomianowym, możliwe jest sprawdzenie czy podane rozwiązanie jest poprawne. Ale samo rozwiązanie problemu nie jest możliwe. Definicja klasy NP-zupełnej jest bardziej skomplikowana. Dowód należenia problemu komiwojażera do tej klasy można znaleźć w [1] na stronie 1036. Przykładem problemu klasy NP jest problem znalezienia cyklu Hamiltona (w skończonym czasie jesteśmy w stanie sprawdzić, czy podana sekwencja wierzchołków tworzy cykl), problem komiwojażera rozszerza to zagadnienie (po otrzymaniu sekwencji węzłów możemy policzyć koszt przebycia trasy, ale nie mamy pewności, że jest ona najkrótszą z możliwych). Dlatego nasz problem jest „tak trudny jak dowolny problem w NP” [1] (strona 991).

## 1.2 Podejście algorytmiczne, a heurystyczne

Podejście algorytmiczne do problemu komiwojażera, gwarantuje rozwiązanie optymalne. Ilość przypadków do rozpatrzenia przy  $n$  wierzchołków wynosi  $n!$ ! Więc sprawdzanie wszystkich możliwych rozwiązań może być skuteczne tylko dla bardzo małych ilości węzłów. Przy większej ilości węzłów czas oczekiwania na wynik ostateczny zmierza do nieskończoności, a dodanie każdego kolejnego węzła (numer  $n+1$ ) tylko zwiększa ten czas (ilość przypadków do rozważenia rośnie  $n+1$  – krotnie). Podejściem całkiem odmiennym jest heurystyczne spojrzenie na ten problem. Trzeba być jednak świadomym wad tego rozwiązania. To podejście nie daje nam gwarancji znalezienia najlepszego rozwiązania, ale daje gwarancje obliczeń w skończonym czasie i zakończenie ich w jakimś minimum lokalnym (być może ekstremum globalnym). Lecz podejście drugie ze względu na gwarancję otrzymania wyniku zostało wykorzystane w tym projekcie. W celu implementacji tego podejścia wykorzystane są algorytmy genetyczne.

## 2 Rozwiązanie problemu komiwojażera przy użyciu algorytmów ewolucyjnych

Mamy zamiar teraz przedstawić nasz sposób rozwiązania tego NP-zupełnego problemu wykorzystując algorytmy decyzyjne. W kolejnych punktach opisze struktury oraz mechanizmy stworzone i wykorzystane przez nas na potrzeby projektu. Zawierają się w nich: struktura chromosomu, losowanie populacji początkowej, kryterium oceny osobników, metody selekcji (używane przez nas: metoda ruletki i metoda turniejowa), proces krzyżowania (krzyżowanie OX oraz heurystyczne), a także wykorzystany przez nas sposób mutowania osobników.

### 2.1 Chromosomy

Spośród trzech możliwych reprezentacji (ścieżkowej, przyległościowej i porządkowej) chromosomu możliwych do wykorzystania w naszym projekcie wybraлиśmy reprezentację ścieżkową. Reprezentacja chromosomu zawiera u nas ciąg wartości odpowiadających kolejnym węzłom (miastom) cyklu. Reprezentacja ta jest najbliższa i najbardziej przejrzysta dla ludzi. Lecz jej wadą jest trudność w implementacji krzyżowania. Chromosomy można tworzyć z wykorzystaniem algorytmu zachłannego. Wersja bez algorytmu zachłannego przypisuje kolejnym wartości losowe numery węzłów w grafie. Wersja zachłanna różni się tym, że trasa jest już w procesie tworzenia optymalizowana. Wybierany jest pierwszy węzeł, a następnie wybierany jest kolejny położony najbliżej i tak zapełniany jest cały chromosom.

### 2.2 Losowanie populacji początkowej

Odbywa się na podstawie wprowadzonej zmiennej (liczba osobników populacji), generowana jest wybrana ilość chromosomów w sposób opisany powyżej (możliwa opcja zachłanna).

### 2.3 Kryterium oceny osobników

Kryterium w przypadku naszego projektu jest po prostu długość ścieżki (cyklu), dążymy do jak najmniejszej wartości tego parametru, a więc kryterium przystosowania w naszym przypadku jest odwrotnie proporcjonalne do wartości długości cyklu.

### 2.4 Metody selekcji

W naszym projekcie w celu wprowadzenie zróżnicowania i możliwości porównania skuteczności postanowiliśmy wdrożyć dwie metody selekcji: poprzez ruletkę oraz metodę turniejową.

**Metoda ruletki** — polega na wielokrotnym (dokładnie tyle ile mamy osobników) losowaniu osobników z pierwotnej populacji. Osobniki te przepisane zostaną do nowej populacji. Losowanie odbywa się poprzez stworzenie „koła fortuny” w którym osobniki lepsze (o większym kryterium przystosowania), zajmują większy obszar. Dzięki temu zabiegowi mają one większe prawdopodobieństwo na przeniesienie do kolejnego pokolenia. Wadą tego rozwiązania jest, stosunkowo szybkie zbieganie się do ewentualnego minimum lokalnego.

**Metoda turniejowa** — druga wybrana przez nas metoda, ma całkiem inne parametry, niż ruletka. Jest ona bardziej odporna na zbieganie się do minimum lokalnych. Jest tak dlatego, że przy każdym z  $n$  losowań, wybierana jest podgrupa osobników z których do kolejnego pokolenia przechodzi tylko najlepiej dopasowany. Ta zwiększona losowość chroni przed zbyt szybkim zbieganiem się. Zastosowany u nas sposób jest uproszczoną metodą turniejową. Przeprowadzany jest turniej wśród wszystkich osobników, w którym wygrywa lepsza połowa (sortujemy rosnąco wszystkie osobniki i wybieramy połowę lepiej przystosowaną), którą krzyżujemy z losowym osobnikiem z gorszej połowy, dzięki czemu dłużej zachowana jest różnorodność. Powstaje w ten sposób nowe pokolenie które przechodzi do następnego cyklu.

## 2.5 Proces krzyżowania

W procesie tworzenia grafu w naszym projekcie możliwy jest wybór typu krzyżowania. Dostępne jest krzyżowanie OX (popularne) oraz krzyżowanie heurystyczne (w tym przypadku na decyzje mają wpływ obecne warunki zadania). Poszczególne metody przedstawiają się następująco:

**Metoda OX** polega na przeniesieniu pewnej losowej części z jednego chromosomu-rodzica i uzupełnieniu wartościami z drugiego, przy zachowaniu kolejności ich występowania i poczynając od tylnej części chromosomu-dziecka. Ogólnie metoda ta polega na wymianie pewnej środkowej części genotypu i uzupełnieniu wartościami drugiego niezmienionego chromosomu, w ramach możliwości. W trakcie prac nad projektem udało nam się zauważyć, że metoda OX nie gwarantuje dobrej zbieżności, co skłoniło nas do implementacji innej metody.

**Metoda heurystyczna** w naszym projekcie działa następująco. Dane są dwa chromosomy - rodzice z których będą wybierane lepsze warianty. Kolejnym krokiem jest losowanie dowolnego węzła. Kolejnym krokiem jest sprawdzenie w którym chromosomie – rodzicu kolejne miasto leży najbliżej, a następnie wybranie i przepisanie go. Jeżeli nie jest to możliwe następuje wybór możliwości z drugiego rodzica. Jeżeli obydwa węzły zostały już wykorzystane w potomku, następuje przepisanie kolejnego najbliższego wariantu. W momencie wywoływania tego krzyżowania jest on wywoływane dla każdego z chromosomów pełniących rolę z rodzica, z parametrem w postaci drugiego z tych osobników. Zachowanie takie ma na celu stworzenie dwójki potomnych dzieci.

## 2.6 Mutowanie

Proces ten w naszym projekcie oparliśmy na prostym wylosowaniu dwóch różnych węzłów na drodze komiwojażera i zamienieniu ich miejscami. Procesowi mutacji podlega co cykl ilość osobników równa:  $(\text{ilość pokoleń bez poprawy})/(\text{max ilość pokoleń bez poprawy}) \cdot 0.5$  (oba parametry można ustawić w momencie tworzenia grafu).

## 2.7 Warunek stopu

W celu zakończenia naszego algorytmu stworzyliśmy dwa warunki stopu, zależne od dwóch zmiennych wprowadzanych przez użytkownika. Ilość pokoleń bez poprawy – jeżeli przez zadaną liczbę pokoleń najlepszy osobnik w nie będzie ulegał zmianą, algorytm zatrzyma się, ponieważ poprawienie wynik jest już

prawdopodobnie niemożliwe. Drugim warunkiem stopu jest maksymalna ilość pokoleń, po przekroczeniu zadanej wartości algorytm zatrzyma się, ze względu na wystarczająco długi czas jego trwania, lub z powodu takich ustawień wprowadzonych przez użytkownika. Oczywiście można też wymusić zatrzymanie algorytmu poprzez dostępne w interfejsie użytkownika opcje.

### 3 Parametry programu

The screenshot shows the 'Generator Grafów' application window. The title bar is green with a close button. The left sidebar has a 'Steps' section with '1. Wybierz parametry symulacji'. The main content area is titled 'Wybierz parametry symulacji' and contains several sections: 'Parametry grafu' with a text input for 'Ilość wierzchołków' and a checkbox 'Wybierz gotowy graf' with a dropdown menu showing '18-miast.xml'; 'Warunki zakończenia algorytmu' with text inputs for 'Ilość pokoleń bez poprawy' (200) and 'Maksymalna ilość pokoleń' (3000); 'Parametry algorytmu' with text inputs for 'Rozmiar populacji' (50), dropdowns for 'Typ krzyżowania' (Heurystyczne) and 'Typ selekcji' (Ruletka), and a checkbox for 'Generowanie alg. zachłannym'; and 'Metoda wizualizacji' with a dropdown for 'Wierzchołków' (Punkty). At the bottom are buttons for '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

Rysunek 1: Generator grafów

Na załączonym rysunku widać okno które pozwala nam wygenerować graf w którym będziemy szukać minimalnego cyklu. Nasz projekt udostępnia użytkownikowi takie opcje jak: Ilość wierzchołków – w zależności od wartości tego parametru, generowany jest graf pełny o zadanej liczbie węzłów. Wybierz gotowy graf – na potrzeby projektu stworzone zostało kilkanaście grafów, na których można bardzo przejrzysto zaobserwować proces wyznaczania minimalnego cyklu. Niektóre grafy, mają też swoje odwzorowanie na mapach Polski i liczą cykle dla wybranych miast. Ilość pokoleń bez poprawy – parametr ten jest pierwszym warunkiem stopu, określa on, ile razy pod rząd może nie wystąpić zmiana u najlepszego osobnika populacji, jeżeli wartość ta zostanie osiągnięta

program zakończy działanie.

- **Maksymalna ilość pokoleń** — drugi warunek stopu, po osiągnięciu tej wartości podczas procesów krzyżowania program zostanie zatrzymany. Pokolenie o indeksie równym temu parametrowi będzie ostatnim potomstwem.
- **Rozmiar populacji** — określa ilość chromosomów zawierających się w każdej populacji, zwiększenie tej wartości powoduje zwiększenie szans na znalezienie ekstremum globalnego, ale jest to osiągane kosztem czasu (zwiększa się ilość obliczeń, a co za tym idzie ich czas). Zmniejszenie go, zwiększa szanse na to, że nasze obliczenia zatrzymają się na pewnym minimum lokalnym, ale przyspiesza działanie programu.
- **Typ krzyżowania** — przełącza pomiędzy zaimplementowanymi przez nas rodzajami krzyżowania. W programie dostępne są krzyżowania typu OX oraz krzyżowanie heurystyczne. Opis różnic poszczególnych krzyżowań zawiera się w części 2.d.
- **Typ selekcji** — podobnie jak parametr wyżej, ustawia jeden z dostępnych rodzajów selekcji. W projekcie zaimplementowane zostały selekcje metodą ruletki oraz uproszczoną metodą turniejową. Opisy tych metod znaleźć można w sekcji 2.e.
- **Gen. alg. zachłannym** — zaznaczenie tej opcji spowoduje wykorzystanie algorytmu zachłannego w procesie tworzenia chromosomów (w trakcie generowania populacji początkowej). Spowoduje to jeszcze szybsze zbieranie się algorytmu. Opis tego algorytmu znajduje się w podpunkcie 2.a.
- **Metoda wizualizacji wierzchołków** — do wyboru mamy tutaj dwie opcje. Wybór punktów polecany jest szczególnie przy większych grafach, a także dla zwiększenia przejrzystości grafu. Opcja z etykietami miast jest dużo ciekawsza, jeżeli obserwujemy działanie algorytmu na przykładzie grafu który odwzorowuje położenie geograficzne określonych miast i na ich przykładzie rozwiązuje problem komiwojażera.

## 4 Wykorzystane technologie/frameworki

Projekt został stworzony przy wykorzystaniu tylko i wyłącznie wolnego oprogramowania jak i korzysta z wolnych bibliotek oraz frameworków. Poniżej kilka słów o najważniejszych z nich, celem przybliżenia dlaczego zostały wybrane, i czy polecilibyśmy je do wykorzystania innym.

### 4.1 NetBeans Platform

Projekt został napisany w Javie, korzystając z NetBeans Platform (w wersji 6.8). Przed rozpoczęciem programowania naszej aplikacji wszyscy uczestniczyliśmy w szkoleniu *NetBeans Platform Certified Training* które organizował Konrad jako członek *Polish Java User Group*. Więcej informacji o samym szkoleniu można znaleźć pod: [netbeans.edu.pl](http://netbeans.edu.pl).

Powodem wyboru NBP jako platformy do rozwoju naszej aplikacji były ogólne zalety frameworków, czyli przyspieszonego cyklu rozwoju samej aplikacji oraz konkretne zalety NetBeans Platform tj. budowa *modularnych aplikacji*. Realne aplikacje generalnie nie są już obecnie pisane nie-modularnie, powodem są oczywiste zalety płynące z luźnego wiązania aplikacji, czyli możliwość niezależnej pracy nad poszczególnymi modułami przez programistów - nie przeszkadzając sobie zmianami ew. udostępnianych metod.

Do realizacji luźnego wiązania między modułami - konkretniej między 'algoritmami' oraz 'elementami' rysującymi graf wynikowy został wykorzystany wzorzec Observer/Observable zaimplementowany przy wykorzystaniu Lookup'ów (jednej z zalet NetBeans Platform). Implementację tą Konrad przedstawił na dZone.com za prośbą Pana Wielenga (Technical Writer w Sun Microsystems) do publikacji tegoż tekstu tam: NetBeans Platform Lookups as Communication Method.

Bardzo ważną zaletą NetBeans Platform, odróżniającą ją na przykład od *Eclipse RCP* lub *Spring RCP* jest bazowanie na Swing - dzięki czemu nie występują problemy podczas integracji istniejących rozwiązań (prefuse, jfreechart etc) z aplikacją bazującą na NetBeans RCP - co w przypadku Eclipse RCP potencjalnie mogłoby sprawić duże problemy, jako że bazuje on na SWT (natywnych kontrolkach).

### 4.2 Prefuse

Prefuse jest dostępną pod <http://prefuse.org/> wolnoźródłową biblioteką implementującą różne wizualizacje oraz reprezentacje grafów. Niestety projekt jest od lat martwy i nigdy nie osiągnął wersji stabilnej. Dokumentacja również jest bardzo nie pełna, i pozostaje korzystanie z samego JavaDoc, co do naszych celów jednak jak najbardziej wystarczało.

Cykl rysowania grafu w prefuse idealnie wpasował się w nasz problem do rozwiązania. Deklaruje się pewne 'akcje', które następnie są wołane w pewnych odstępach czasu, dokładnie tak jest wołany nasz algorytm rozwiązywania problemu komiwojażera. Warto nadmienić iż listy akcji mogą być od siebie niezależne i dzięki temu rysowanie/kolorowanie grafu nie musi blokować liczenia algorytmu.

Inną zaletą jest wbudowana obsługa parsowania GraphML — była ona de facto głównym czynnikiem wyboru tej biblioteki.



### 4.3 GraphML

Program korzysta z plików XML zapisywanych w folderze data w celu przechowywania danych o wygenerowanych (oraz dostępnych predefiniowanych) grafach. Format tych plików jest ściśle określony przez specyfikację GraphML, dzięki czemu możliwe jest sprawdzenie naszych algorytmów na dowolnym grafie - dostarczonym do folderu data w postaci pliku zgodnego z GraphML.

### 4.4 JFreeChart

Do rysowania wykresów zbieżności algorytmu użyliśmy bardzo dobrej biblioteki jaką jest JFreeChart. Jest ona dostępna na licencji LGPL pod następującym adresem: <http://www.jfree.org/jfreechart/>. Biblioteka ta jest dość znana stąd pominę głębsze rozważania nad jej przydatnością — słowem: jest najlepszą (wolnoźródłową) znaną nam biblioteką służącą do wizualizacji wykresów.

## 5 Kod źródłowy oraz licencja

Kod źródłowy programu był rozwijany od początku do końca przy użyciu systemu kontroli wersji **Git**. Repozytorium znajduje się pod poniższym adresem:

<http://github.com/ktoso/TravelingSalesman-NBP>

Możliwe do wglądu są kolejne 'wydania' (tagi) jak i najświeższa wersja kodu. Program będzie w całości licencjonowany na warunkach **GPLv3** — odpowiednie adnotacje w kodzie źródłowym zostaną dodane już niebawem.

## 6 Bibliografia

- Cormen Thomas H., Leiserson Charles E., Rivest Ronald L., Stein Clifford — 'Wprowadzenie do algorytmów' 2007
- Zbigniew Michalewicz — 'Algorytmu genetyczne + programy ewolucyjne = programy ewolucyjne' 1996
- Tim Boudreau, Jaroslav Tulach, Geertjan Wielenga — 'Rich Client Programming: Plugging into the NetBeans Platform' 2007