Problem komiwojażera:  
opis rozwiązania przy pomocy symulowanego wyżarzania dla grafu spójnego skierowanego.

Autorzy:  
Tomasz Kogut  
Mateusz Mielnicki

Konsultacja merytoryczna:  
dr inż. Sebastian Kozłowski

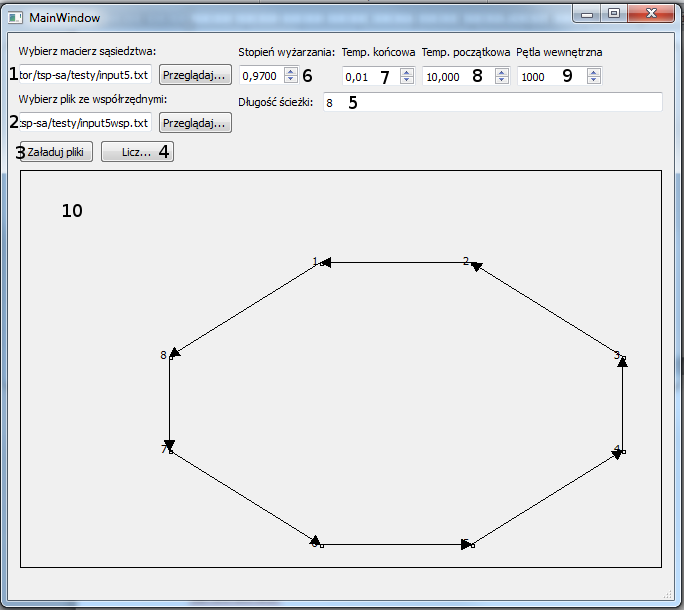
**Streszczenie**

**Poniższy dokument jest opisem realizacji projektu z przedmiotu „Grafy i sieci”. Tematem projektu było zaimplementowanie programu na komputer osobisty PC, który wczyta graf miast z pliku i rozwiąże problem komiwojażera używając symulowanego wyżarzania. W dokumencie czytelnik znajdzie szczegóły implementacyjne oraz opis zastosowania języka c++ oraz biblioteki QT do rozwiązania zadania. Przeprowadzamy również analizę działania programu w zależności od liczby miast i połączeń w grafie.**

**26 maja 2011**

# **Opis graficznego interfejsu użytkownika**

## Elementy interfejsu



Ilustracja 1: Zrzut ekranu z programu

Ilustracja [1] prezentuje zrzut ekranu z wyniku działania programu dla 8 miast wraz z nałożonymi numerami, które są referencją dla dostępnych opcji.

1. Lokalizacja pliku z macierzą sąsiedztwa.
2. Plik ze współrzędnymi miast (opcjonalny).
3. Przycisk ładujący dane wejściowe.
4. Przycisk uruchamiający działanie algorytmu.
5. Pole wynikowe prezentujące wynik działania algorytmu – długość znalezionej ścieżki.
6. Poziom wyżarzania używany przy obliczeniach.
7. Temperatura końcowa, której osiągnięcie zatrzymuje algorytm.
8. Temperatura początkowa na poziomie której algorytm zaczyna działanie.
9. Liczba iteracji dla stałego poziomu temperatury.
10. Płótno obrazujące wprowadzone współrzędne miast. Po i w trakcie działania algorytmu, obrazuje również ścieżki między miastami wraz z zaznaczonym kierunkiem.

## Wykonanie programu

Aby wykorzystać program do wykonania obliczeń należy wykonać następującą sekwencje czynności:

* Wskazać lokalizację pliku tekstowego zawierającego macierz sąsiedztwa.
* (Opcjonalnie) Wskazać plik zawierający współrzędne miast. Jeżeli plik nie zostanie wskazany, współrzędne zostaną wylosowane.
* Ustalić żądane parametry algorytmu: stopień wyżarzania, temperaturę początkową, temperaturę końcową i pętlę wewnętrzną.
* Kliknąć w przycisk „Załaduj pliki”, co powinno nanieść miasta na płótno wraz z numerami.
* Kliknąć w przycisk licz co uruchomi algorytm.

W trakcie liczenia płótno będzie na bieżąco aktualizowane siecią dróg, które były rozważane w trakcie działania algorytmu.

W momencie gdy algorytm się zakończy, sieć dróg ustabilizuje się na najlepszej znalezionej trasie i pojawi się liczba oznaczająca długość trasy.

# **Szczegóły implementacyjne**

## Architektura programu

Lista klas i struktur:

* Graficzne:
  + MainWindow – reprezentuje widoczny dla użytkownika interfejs. Steruje uruchomieniem algorytmu oraz prezentuje wynik
  + Draw – reprezentuje płótno, które obrazuje rozmieszczenie miast oraz obliczoną ścieżkę.
* Funkcjonalne
  + InputReader – reprezentuje obiekt odpowiedzialny za wczytywanie danych wejściowych i dostarczanie ich obiektowej reprezentacji do pozostałych obiektów.
  + TSPSolver – reprezentuje obiekt obliczeniowy, który zawiera algorytmy potrzebne do przeprowadzenia wszystkich obliczeń.
* Pomocnicze
  + CityPosition – klasa reprezentująca położenie miasta we współrzędnych kartezjańskich
  + RoadPart – struktura reprezentująca połączenie między dwoma miastami. Jest to uporządkowana para identyfikatorów miast odpowiadająca wierszom w macierzy sąsiedztwa.
  + ComputeThread – obiekt reprezentujący wykonanie algorytmu w oddzielnym wątku. Służy do uruchamiania algorytmu oraz odbierania i przekazywania danych do okna graficznego.

## Opis algorytmów obliczających trasę.

### Dane wstępne

Za całość obliczeń odpowiada klasa TSPSolver, która zawiera metody dokonujące kolejnych kroków obliczeń. Do działania potrzebuje następujących informacji:

* Macierzy sąsiedztwa.
* Liczby miast.
* Początkową temperaturę.
* Końcową temperaturę.
* Stopień wyżarzania.
* Liczbę powtórzeń wewnętrznej pętli

### Generacja trasy początkowej

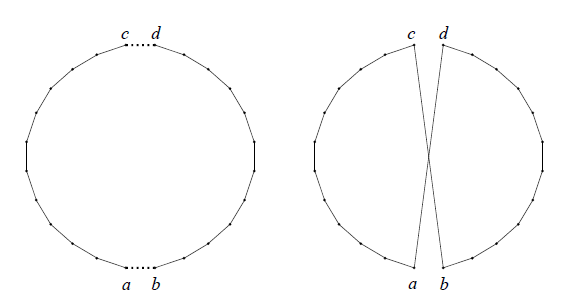
Po podaniu danych wstępnych TSPSolver generuje trasę początkową:

* Start następuje poprzez wylosowanie jednego z wierzchołków.
* Wszystkie miasta zostają oznaczone w pomocniczej tablicy jako nieodwiedzone.
* Metodą zachłanną przechodzimy do kolejnego sąsiada czyli wybierane jest miasto, które znajduje się najbliżej.
* Miasto które opuszczamy oznaczane jest w tablicy jako odwiedzone.
* Jeżeli wszyscy sąsiedzi byli już odwiedzeni to poprzez wyszukiwanie w głąb szukamy jeszcze nieodwiedzone miasta i powtarzamy algorytm.
* Algorytm działa dopóki licznik nieodwiedzonych miast nie spadnie do 1.
* Jeżeli licznik odwiedzonych miast spadnie do 1 to oznacza, że wszystkie miasta zostały odwiedzone.
* Szukamy drogi powrotnej do punktu wylosowanego na początku, od którego startowaliśmy poprzez wyszukiwanie w głąb.

Jeżeli nie jesteśmy w stanie znaleźć wierzchołka, który nie został jeszcze odwiedzony poprzez przeszukiwanie w głąb, a licznik jest > 1 to znaczy, że podany graf nie jest możliwy do obejścia i algorytm się kończy.

### Generacja sąsiedztwa.

W projekcie do generowania sąsiedztwa, czyli tras alternatywnych, został użyty algorytm 2-opt. Kolejne jego kroki przedstawiają się następująco:

* Losowane są 2 krawędzie nie sąsiadujące ze sobą (wyznaczają one zatem 4 wierzchołki)
* Następuje zamiana krawędzi zgodnie z algorytmem 2-opt co przedstawia poniższy rysunek [2]   
  

**Ilustracja 2: prezentacja działania algorytmu 2-opt**

* Nowo stworzona ścieżka jest sprawdzana pod kątem sąsiadujących duplikatów i jeśli takie istnieją zostają usunięte

Zasadą działania tego algorytmu jest tworzenie ścieżek różniących się nieznacznie od oryginalnej, jednak umożliwiających dojście do lepszych rozwiązań.

### Symulowanie wyżarzanie.

Jest to główny algorytm projektu. Jego działanie polega na poszukiwaniu rozwiązań w sąsiedztwie aktualnego rozwiązania. Dodatkowo wprowadzone jest pojęcie kroku w dół (czyli podążanie algorytmu za lepszym rozwiązaniem), który wykonywany jest zawsze w przypadku jego wystąpienia, oraz pojęcie kroku w górę (czyli przyjęcie gorszego rozwiązania), które jednak może wystąpić tylko z określonym przez algorytm prawdopodobieństwem, określanym przez aktualną temperaturę. Działanie algorytmu można opisać następującymi krokami:

1. Pobranie temperatury początkowej, końcowej, stopnia wyżarzania, oraz ilości powtórzeń wewnętrznej pętli
2. Wygenerowanie trasy początkowej
3. Dopóki aktualna temperatura jest większa od temperatury końcowej
   1. Powtórz poniższe kroki zadaną ilość razy (powtórzenia pętli wewnętrznej)
      1. Stwórz sąsiedztwo aktualnego rozwiązania korzystając z metody 2-opt
      2. Określ różnicę odległości między starą a nową trasą i oznacz przez
      3. Jeśli nowa trasa jest krótsza lub tej samej długości co stara, to ustal za aktualną nową trasę
      4. Jeśli nowa trasa jest krótsza od najkrótszej, zapisz ją jako nową najkrótszą
      5. Jeśli nowa trasa jest dłuższa od aktualnej to wybierz losową liczbę r w przedziale [0,1] a następnie w przypadku gdy ustaw jako aktualną nową trasę
      6. Koniec pętli z kroku 3.1.
   2. Oblicz nową temperaturę ze wzoru: nowa temperatura = stara temperatura \* stopień wyżarzania
   3. Koniec pętli z kroku 3.
4. Zwrócenie najlepszego uzyskanego wyniku

Algorytm ten używa temperatury do określenie prawdopodobieństwa wybrania gorszej trasy. Celem działania tego mechanizmu jest niwelowanie lokalnych minimów, a co za tym idzie możliwość znalezienia minimum globalnego. Wraz z obniżaniem temperatury algorytm dąży do uzyskania najmniejszego wyniku z badanego aktualnie minimum.

# Opis danych wejściowych i wyjściowych

## Generacja danych testowych

Ponieważ aby przetestować działanie algorytmu należy wygenerować szereg grafów, które będą:

* spełniać warunek możliwości odwiedzenia wszystkich miast,
* będą różnych, często znacznych rozmiarów,

cały proces należy zautomatyzować. Do programu dołączany jest skrypt napisany w języku Python. Skrypt *„generate\_matrix.py”* przyjmuje dwa parametry, pierwszy mówiący o liczbie miast a drugi o liczbie połączeń między miastami.

Aby zapewnić warunek, że wszystkie miasta można obejść i wrócić do punktu wyjścia, skrypt łączy w pierwszym kroku w cykl wszystkie miasta. Następnie pozostała żądana liczba połączeń jest generowana losowo.

Skrypt drukuje losową macierz sąsiedztwa na standardowe wyjście w formacie zgodnym z tym, który program rozwiązujący problem komiwojażera przyjmuje na wejście.

Liczba miast nie ma narzuconego ograniczenia wynikającego ze specyfiki programu. Jedynym ograniczeniem jest moc obliczeniowa komputera.

Liczba połączeń między miastami zależy od liczby miast „n”, która należy do przedziału: <n,n(n-1)>. Dolne ograniczenie wynika z potrzeby zapewnienia cyklu. Górne ograniczenie daje graf pełny, gdzie każde miasto ma połączenie z innym.

Program testowano na maksymalnie 500 wierzchołkach, gdzie każde miasto miało połączenie z co drugim miastem przy 2GB pamięci.

## Wyjście

Program po wykonaniu obliczeń prezentuje graficzną reprezentacje znalezionej trasy odmalowaną na płótnie oraz jedną liczbę, sumę wag krawędzi.

# **Analiza działania programu**

Skuteczność działania programu można określić w różny sposób. Zależy to przede wszystkim od wymagań stawianych programowi, czy to czasowych czy też jakościowych.

## Czasowa analiza działania programu

Poniżej przedstawiam wykres [1] prezentujący czasy wykonania programu dla następujących parametrów:

- liczba wierzchołków: 200

- temperatura początkowa: 25

- temperatura końcowa: 0,001

**Wykres 1: prezentacja czasu wykonania w zależności od przyjętych argumentów**

Jak widać czas wykonania programu może się bardzo różnić w zależności od użytych parametrów. W najdłuższym zaprezentowanym przypadku program musi wykonać wewnętrzną pętlę aż 1 012 200 razy, zaś w przypadku najkrótszym liczba powtórzeń wyniesie zaledwie 10122. To liczba koniecznych powtórzeń wpływa w głównym stopniu na czas trwania algorytmu.

Testy czasowe zostały wykonane na maszynie z 4 rdzeniowym procesorem AMD Phenom II 955 taktowanym zegarem 3600MHz. Zadanie obliczeniowe naszego programu wykorzystuje jednak tylko 1 rdzeń.

## Jakościowa analiza działania programu

**Wykres 2: prezentacja uzyskanego wyniku w zależności od ustawień algorytmu**

Badano również skuteczność algorytmu dla danych wygenerowanych losowo, które były użyte do testów czasowych. Wyniki tych badań przedstawia powyższy wykres [2]. Można z nich wywnioskować, że zarówno samo stosowanie minimalnego wyżarzania (0,999) jak i stosowanie ogromnej liczby powtórzeń (1000) kosztem stopnia wyżarzania (0,8) nie przynosi dobrych rezultatów. Obie te wartości powinny być ze sobą zgrane. Przykładowo rozpoczynając próby przy stopniu wyżarzania 0,9 i liczbie powtórzeń 10, w celu poszukiwania lepszych rozwiązań należy zarówno zwiększyć stopień wyżarzania (np. do 0,95) ale także zwiększyć liczbę powtórzeń (np. do 30). Zaniechanie którejś z tych części może prowadzić do uzyskiwania słabszych wyników przez program.

Należy jednak pamiętać o losowym charakterze działania algorytmu co ma niewątpliwie duży wpływ na uzyskiwane przez niego wyniki.

# Załącznik

Zamieszczony poniżej plik zawiera pliki źródłowe stworzonego programu.



Spis treści.

[1. Opis graficznego interfejsu użytkownika 2](#_Toc294358116)

[1.1. Elementy interfejsu 2](#_Toc294358117)

[1.2. Wykonanie programu 3](#_Toc294358118)

[2. Szczegóły implementacyjne 4](#_Toc294358119)

[2.1. Architektura programu 4](#_Toc294358120)

[2.2. Opis algorytmów obliczających trasę. 4](#_Toc294358121)

[2.2.1. Dane wstępne 4](#_Toc294358122)

[2.2.2. Generacja trasy początkowej 4](#_Toc294358123)

[2.2.3. Generacja sąsiedztwa. 5](#_Toc294358124)

[2.2.4. Symulowanie wyżarzanie. 5](#_Toc294358125)

[3. Opis danych wejściowych i wyjściowych 7](#_Toc294358126)

[3.1. Generacja danych testowych 7](#_Toc294358127)

[3.2. Wyjście 7](#_Toc294358128)

[4. Analiza działania programu 8](#_Toc294358129)

[4.1. Czasowa analiza działania programu 8](#_Toc294358130)

[4.2. Jakościowa analiza działania programu 9](#_Toc294358131)

[5. Załącznik 10](#_Toc294358132)