

AKADEMIA GÓRNICZO-HUTNICZA

im. St. Staszica w Krakowie

WEAiE, Katedra Automatyki

Przedmiot: **Laboratorium badań operacyjnych**

Temat projektu: „**Poszukiwanie optymalnej drogi łączącej dwa przystanki, za pomocą algorytmu Tabu Search**”

Wykonali: Paweł Konieczny, Robert Szmaciński

1. Zapoznanie z problemem

Program ma za zadanie zoptymalizować drogę łączącą dwa przystanki. Danymi wejściowymi są numery dwóch przystanków, pomiędzy którymi będzie szukana optymalna trasa, oraz obraz w formacie bmp, będący mapą przystanków. Pierwsze zadanie programu jest zeskanowanie mapy i wprowadzenie do programu struktury połączeń między przystankami. Daną wyjściową programu jest również obraz w formacie bmp z zaznaczoną optymalną trasą łączącą dwa wybrane przystanki. Za optymalną trasę, w zależności od ustawień programu uważa się minimalną sumę odległości pomiędzy przystankami trasy, lub minimalną ilość przystanków trasy.

2. Metody rozwiązania

a) Szukanie rozwiązania startowego:

Z każdego krańcowego przystanku losowo wybieramy sąsiada. Od wybranego sąsiada losujemy kolejnego. W ten sposób otrzymujemy 2 listy o jednym końcu w przystanku początkowym bądź końcowym. Powiększamy obydwie listy do momentu ich przecięcia się. W momencie przecięcia 2 list odrzucamy „ślepe końce” i otrzymujemy rozwiązanie startowe. W momencie zapętlenia jednej z list, bądź dotarcia do ślepej uliczki odrzucamy zapętloną część listy.

Pseudokod wyżej opisanego algorytmu:

```
while(lista1 nie przecina się z lista2)
{
    losowy sąsiad=losuj sąsiada końca listy1;
    dołącz losowy sąsiad do listy1;
    losowy sąsiad=losuj sąsiada końca listy2;
    dołącz losowy sąsiad do listy2;
    if(zapętłona lista1)
        porzuć zapętloną część;
    if(zapętłona lista2)
        porzuć zapętloną część;
}
utnij ślepe końce listy1;
utnij ślepe końce listy2;
```

b) Szukanie rozwiązania należącego do sąsiedztwa rozwiązania aktualnego.

Rozwiązanie aktualne jest listą o końcach w przystanku początkowym i końcowym.

Losujemy dwa przystanki z środka listy i układamy 2 listy. Pierwsza lista od przystanku początkowego do bliższego wylosowanego. Druga lista od dalszego wylosowanego przystanku do końcowego. Część listy pomiędzy dwoma wylosowanymi przystankami zapominamy. Następnie w analogiczny sposób do szukania rozwiązania startowego powiększamy listy o losowo wybranych sąsiadów. Jeżeli wystąpi zapętlenie nakładamy karę na rozwiązanie tzn: zwiększamy maksymalnie funkcję celu i kończymy algorytm.

Pseudokod wyżej opisanego algorytmu:

```
losowy przystanek1= losuj przystanek z aktualnej listy;
losowy przystanek2= losuj przystanek z aktualnej listy;
przystanek1=min(losowy przystanek1, losowy przystanek2);
przystanek2=max(losowy przystanek1, losowy przystanek2);
lista1=(od przystanek początkowy do przystanek1);
lista2=(od przystanek2 do przystanek końcowy);
```

```

while(lista1 nie przecina się z lista2)
{
    losowy sąsiad=losuj sąsiada końca listy1;
    dołącz losowy sąsiad do listy1;
    losowy sąsiad=losuj sąsiada końca listy2;
    dołącz losowy sąsiad do listy2;
    if(zapętłona lista1)
    {
        funkcja celu+=kara;
        koniec;
    }
    if(zapętłona lista2)
    {
        funkcja celu+=kara;
        koniec;
    }
}
utnij ślepe końce listy1;
utnij ślepe końce listy2;

```

c) Algorytm popraw:

W programie wykorzystano algorytm tabu search. Polega on na znajdowaniu rozwiązania z sąsiedztwa rozwiązania aktualnego, które nie zawiera się w liście rozwiązań zabronionych. Jeżeli znalezione rozwiązanie jest lepsze od rozwiązania optymalnego to rozwiązanie optymalne równa się znalezionemu rozwiązaniu. Znalezione rozwiązanie dodajemy do listy zabronionych i rozwiązanie aktualne równa się znalezionemu rozwiązaniu. Opisaną wyżej operację powtarzamy określoną ilość razy lub jeżeli funkcja celu z rozwiązania minimalnego będzie mniejsza niż określony wcześniej próg.

Pseudokod wyżej opisanego algorytmu:

```

while(ilość iteracji < zadana ilość iteracji lub funkcja celu (rozwiązanie optymalne)<zadany koszt)
{
    do
    {
        nowe rozwianie=losuj z sąsiedztwa(rozwiazanie aktualne);
    }
    while(nowe rozwiazanie nie zawiera się w lista tabu)
    if(nowe rozwianie < optymalne rozwianie)
        optymalne rozwianie=nowe rozwianie;
    dołącz do listy tabu nowe rozwianie;
    usuń ostatnie rozwianie z listy tabu;
    if(długość listy tabu > zadana długość listy tabu )
        usuń ostatnie rozwianie z listy tabu;
    rozwianie aktualne= nowe rozwianie;
}
wyprowadź optymalne rozwianie;

```

d) Kryterium aspiracji

Program umożliwia uwzględnianie w pojedynczej iteracji algorytmu tabu search kryterium aspiracji. Polega ono na odliczanie ilości iteracji bez zmiany rozwiązania optymalnego. Jeżeli odliczona wartość jest większa od zadanej to znaczy, że algorytm wpadł w minimum lokalne przestrzeni rozwiązań. Podejmujemy wtedy następujące działania: wybieramy minimalne rozwiązanie znajdujące się w liście tabu nie będące rozwiązaniem aktualnym ani optymalnym i kopiujemy je do rozwiązania aktualnego następnie kontynuujemy

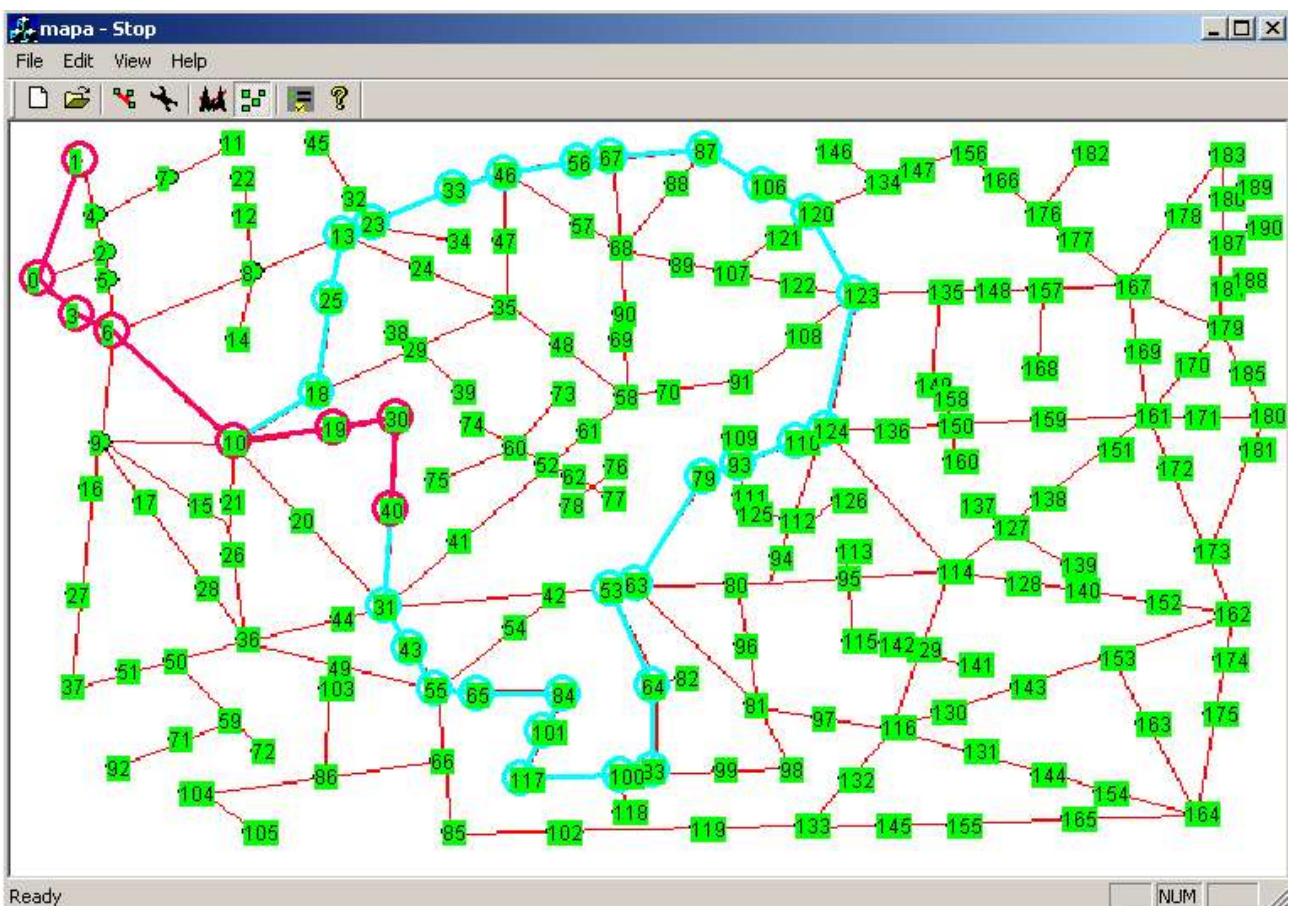
algorytm tabu search.

Pseudokod wyżej opisanego algorytmu:

```
if(ilość iteracji bez zmiany rozwiązania optymalnego > zadana wartość)
{
    do
    {
        nowe rozwiązanie = min(lista tabu);
        usuń z listy tabu(nowe rozwiązanie);
    }
    while(nowe rozwiązanie != rozwiązanie optymalne i
        nowe rozwiązanie != rozwiązanie aktualne)
    dodaj do listy tabu(nowe rozwiązanie);
    rozwiązanie aktualne = nowe rozwiązanie;
}
```

3. Instrukcja obsługi programu

Interfejs programu wygląda następująco:



Programem steruje się z paska narzędzi.

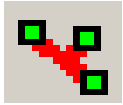
Opis paska narzędzi:



Wyczyszczenie mapy.



Otwarcie nowej mapy.



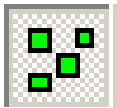
Inicjuje rozwiązanie startowe.



Algorytm popraw.



Wykres zmian funkcji celu podczas działania algorytmu popraw



Powrót do widoku mapy.



Wyświetl parametry.

Poniższe okno przedstawia dialog do zaminy i wglądu ustawień programu:

The screenshot shows a 'Dialog' window with a title bar containing a close button. The window is divided into two main sections. The left section, titled 'Algorytm TabuSearch', contains several input fields and a checkbox. The right section, titled 'Metric', contains two radio buttons. At the top right of the dialog are 'OK' and 'Cancel' buttons.

Field	Value	Description
Number of iteration	100	
Min Cost	-1	
Size of Tabu List	30	
Enable Criterium of aspiration	<input checked="" type="checkbox"/>	
Number of iteration without any increase, whereby actual solution is changing	10	
Metric	<input type="radio"/> Totality of distans <input checked="" type="radio"/> Stuops count	

Opis ustawień:

Number of iteration- liczba iteracji w algorytmie tabu search.

Min Cost- minimalna wartość funkcji celu rozwiązania optymalnego poniżej której algorytm kończy działanie zwracając rozwiązanie optymalne.

Size of tabu list- wielkość listy zabronień.

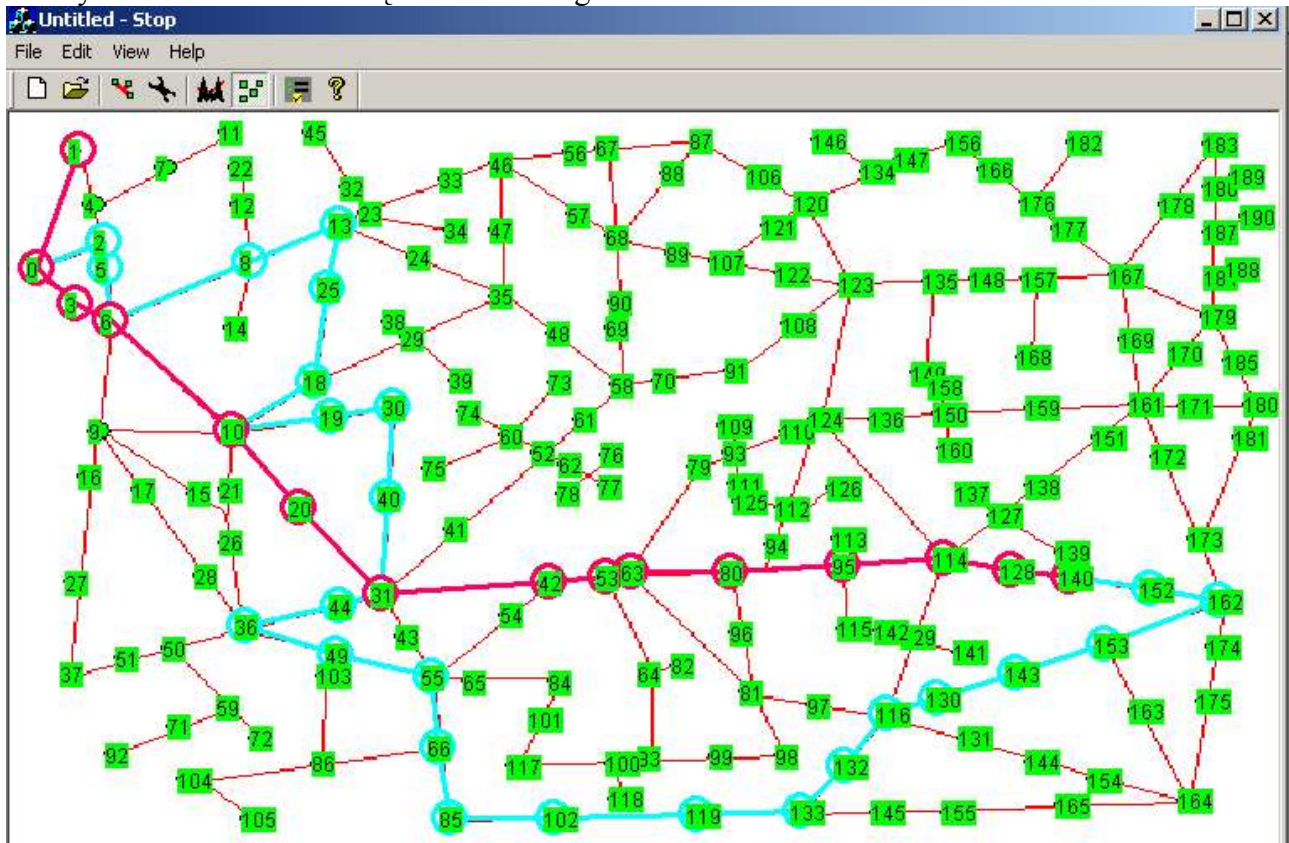
Enable criterium of aspiration- uwzględniaj/ nie uwzględniaj kryterium aspiracji.
 Metric-Totality of distans/Stuops count- wybranie metryki będącej sumą odległości pomiędzy przystankami rozwiązania/ ilość przysatnków rozwiązania.

4. Przykłady działania programu:

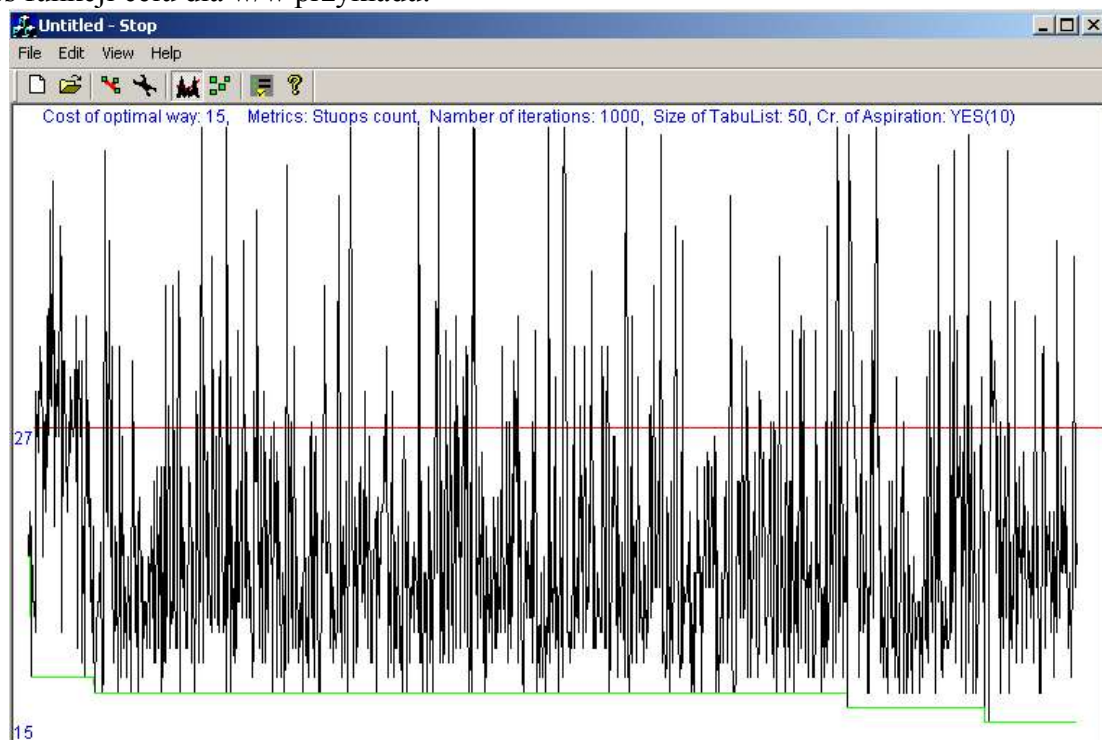
Przystanek początkowy 1 końcowy 140.

Gruba czerwona linia trasa rozwiązania końcowego.

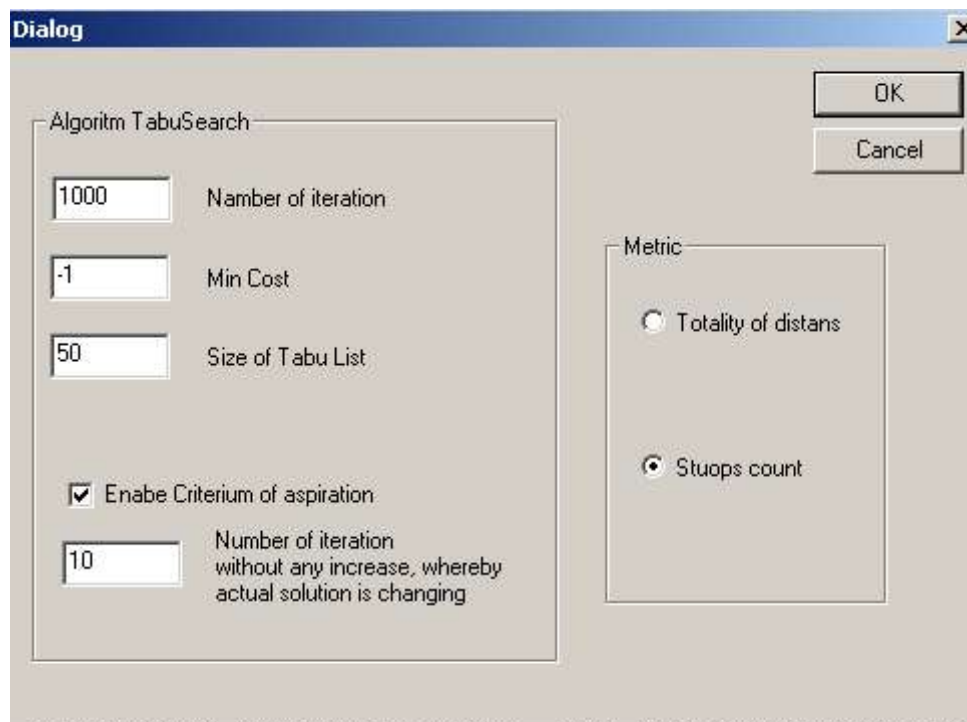
Seledynowa linia trasa rozwiązania startowego.



Wykres funkcji celu dla w/w przykładu:



Parametry algorytmu dla w/w przykładu:



The image shows a Windows-style dialog box titled "Dialog" with a close button (X) in the top right corner. The dialog is divided into two main sections. The left section, titled "Algorithm TabuSearch", contains several input fields and a checkbox. The right section, titled "Metric", contains two radio button options. At the top right of the dialog are "OK" and "Cancel" buttons.

Algorithm TabuSearch

Number of iteration: 1000

Min Cost: -1

Size of Tabu List: 50

☒ Enable Criterium of aspiration

Number of iteration without any increase, whereby actual solution is changing: 10

Metric

☐ Totality of distans

☒ Stuops count