Algorytm Taboo Search

Zastosowanie do poszukiwania optymalnej drogi wydobycia złóż
w kopalni diamentu

Matematyczne Metody Wspomagania Decyzji

Joanna Muniak
Olga Borgula
WEAIiIB, AiR, rok 3, sem. 5

1. Opis problemu

Rozważanym przez nas problemem jest wydobycie złóż diamentu o znanych lokalizacjach i szacowanych kosztach wydobycia. Najważniejszym ograniczeniem jest minimalny zysk, jaki musimy osiągnąć po zebraniu wybranych złóż.

Rozpatrywane przez nas rozmieszczenie złóż, trudność ich wydobycia ze względu na znajdujący się po drodze grunt oraz koszty eksploatacji maszyn wiertniczych, dodatkowych umocnień w tunelach czy wentylacji zostały dla uproszczenia ujęte w jednej macierzy kosztów. W razie potrzeby zawsze istnieje możliwość generowania wprowadzanej do algorytmu macierzy kosztów w oparciu o inne tabele określające właściwości złóż oraz realizację przejść pomiędzy nimi, jednak nie jest to istotnym problemem matematycznym, ani też nie stanowi ciekawej części z punktu widzenia realizacji algorytmu.

Podobnie spodziewane zyski, które w rzeczywistości mają związek z jakością i wielkością złoża, przedstawione zostały jako pojedynczy wektor zysków.

Funkcją celu jest suma kosztów wydobycia poszczególnych elementów:

$$\sum_{i=2}^{m} mac_{kosz}\left(x_{(i-1)}, x_i\right)$$

gdzie:

x – wektor rozwiązań.

Natomiast wektor rozwiązań zawiera kolejne wybrane złoża:

$$x=[x_{1,} x_{2,} \dots, x_m], x_i \in \{1, 2, \dots, n\}$$

gdzie:

n – ilość dostępnych złóż,

m – długość rozwiązania.

Przyjęte ograniczenia są następujące:

$$\sum_{i=2}^{m} mac_{zysk}\left(x_{(i-1)}, x_i\right) \geqslant zysk_{min}$$

gdzie:

$zysk_{min}$ – minimalny zysk, jaki chcemy uzyskać.

oraz:

$$\{x_{1,} x_m\} = 1$$

a więc zawsze rozpoczynamy i kończymy w tym samym miejscu – u wejścia do kopalni.

Istotne dla działania algorytmu jest to, iż nie ma on na celu wydobycia wszystkich złóż, a jedynie te, które są w stanie zrealizować ograniczenia przy najmniejszym koszcie. Istnieje możliwość przejścia między dwoma elementami wielokrotnie, jednak zysk z wydobycia danego elementu jest, naturalnie, dodawany do sumy tylko raz.

2. Działanie algorytmu

Algorytm może działać na podstawie wprowadzonej z pliku macierzy kosztów oraz zysków lub też generować je w sposób losowy na podstawie podanych przez użytkownika parametrów: wielkości problemu (ilości złóż) oraz maksymalnych zysków i kosztów pojedynczego złoża. Pozostałymi parametrami wprowadzanymi przez użytkownika niezależnie od sposobu pozyskiwania macierzy zysków i kosztów są:

- minimalny oczekiwany zysk – pozwala na realizację ograniczenia;
- ilość iteracji przeprowadzanych od ostatniej poprawy wyniku – stanowi kryterium stopu;
- sposób tworzenia pierwszego rozwiązania – w programie zaimplementowane zostały dwa warianty tworzenia rozwiązania początkowego;
- czas obowiązywania listy tabu – ilość iteracji, na jaką odnalezione w kolejności rozwiązanie jest zabronione w celu uniemożliwienia powrotu do niego.

Po pobraniu wszystkich potrzebnych danych algorytm generuje rozwiązanie początkowe. Pierwszy sposób zakłada poszukiwanie kolejnych złóż, do których koszt przejścia z aktualnego złoża jest najmniejszy do momentu spełnienia ograniczeń. W przypadku, gdy droga się zapętli (zysk przestanie się zwiększać), program przejdzie do losowego złoża i będzie kontynuował poszukiwania po minimalnych kosztach. Ten wariant pozwala na niezwykle szybkie znalezienie rozwiązania optymalnego przez algorytm, gdyż rozwiązanie początkowe jest już bardzo wysokiej jakości.

Drugi sposób generowania rozwiązania początkowego opiera się na losowym dobraniu kolejnych złóż aż do momentu spełnienia ograniczenia. Ten sposób pozwala nam obserwować dłuższe działanie algorytmu optymalizującego rozwiązanie.

Dalej program przechodzi do realizacji poszukiwania z zabronieniami. Rozpoczyna od przeszukania sąsiedztwa aktualnego rozwiązania i ustalenia najlepszego sąsiada. W tym celu zostały zastosowane równolegle dwa rodzaje sąsiedztw:

Pierwszy sposób tworzenia sąsiedztwa zakłada zamianę pojedynczo w wektorze rozwiązań wszystkich możliwych par złóż prócz pierwszego i ostatniego (które zawsze są wejściem do kopalni – miejsce nr 1). W ten sposób nie zmienia się nigdy zysk końcowy, więc nie musimy brać go pod uwagę. Spośród wszystkich możliwych 'zamian par' algorytm wybiera i zapamiętuje najlepszą, która nie widnieje na liście tabu.

Drugi sposób tworzenia sąsiedztwa zakłada pojedynczą wymianę każdego kolejnego złoża w rozwiązaniu na każde inne z dostępnych. W ten sposób uzyskujemy wszystkie możliwe wektory z jednym wymienionym złożem. W tym momencie istotne jest sprawdzenie ograniczeń. Jeśli minimalny zysk nie jest osiągnięty, do wektora rozwiązania dodawany jest na przedostatnim miejscu jeszcze jeden element, aż do skutku. Jeśli zaś istnieje możliwość skrócenia rozwiązania przy dalszym spełnieniu ograniczenia, element o najmniejszym zysku jest usuwany z wektora rozwiązań. Po przeglądzie wszystkich generowanych w ten sposób wariantów i porównaniu ich z listą tabu wybierany jest najlepszy dopuszczany przez zabronienia sąsiad dla tej metody.

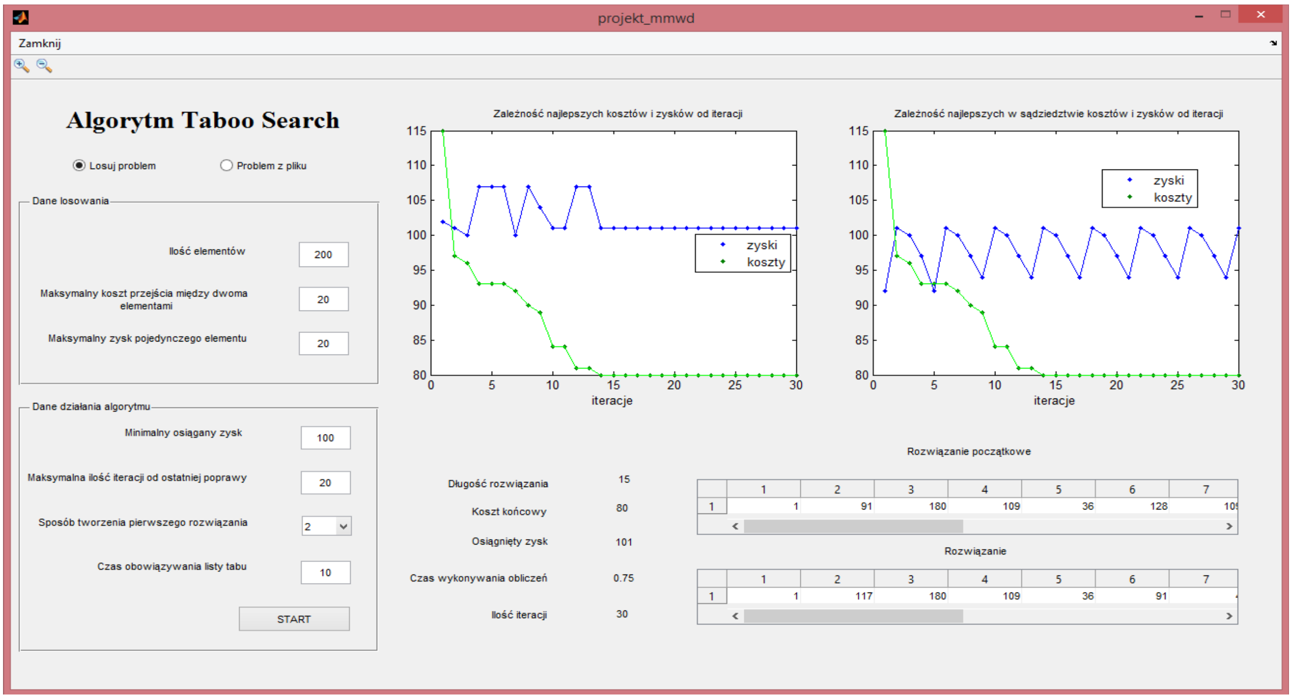
Następnie następuje porównanie osiąganego przez sąsiadów wygenerowanych dwoma metodami kosztu i wybranie lepszego. Wybrany najlepszy sąsiad staje się aktualnym rozwiązaniem. Jeśli jest ono lepsze od najlepszego dotychczas rozwiązania, zapamiętujemy je jako najlepsze.

Aktualne rozwiązanie dopisujemy do listy tabu, w której będzie się znajdować na określoną wcześniej ilość iteracji. Po zapisaniu wszystkich istotnych zmiennych przechodzimy do kolejnej iteracji.

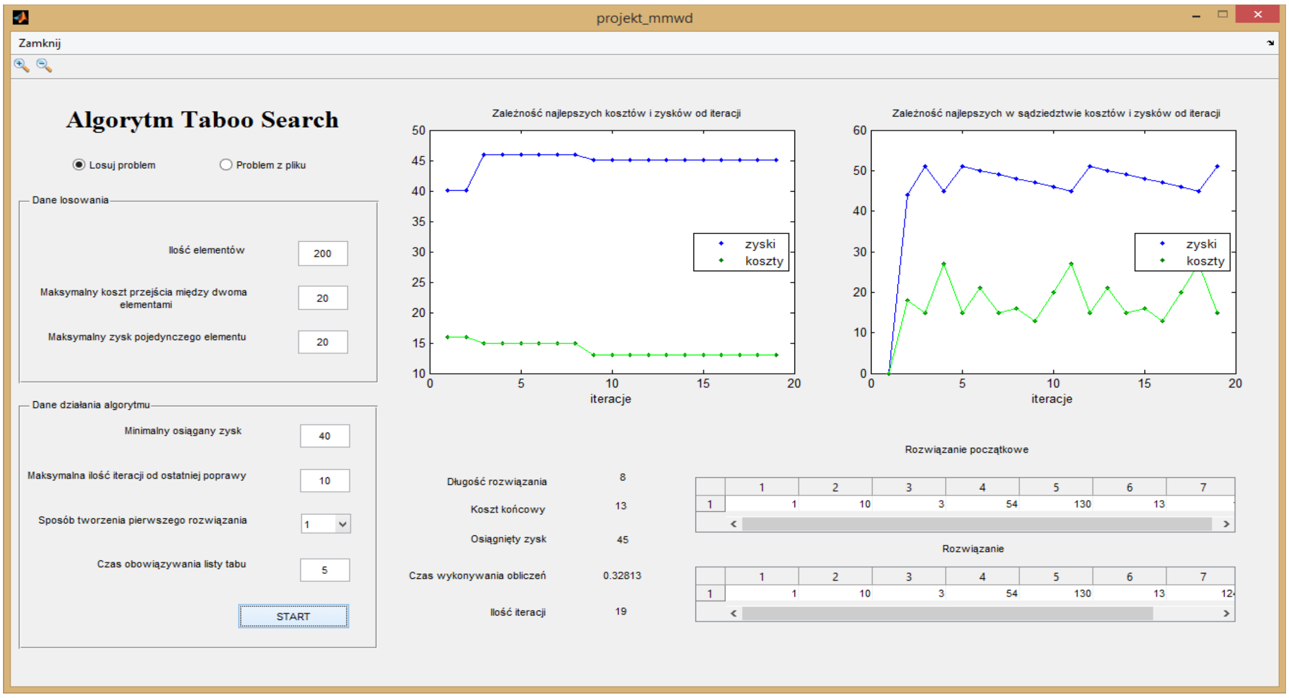
Program kończy pracę po spełnieniu warunku stopu – braku poprawy najlepszego rozwiązania przez zadaną ilość iteracji.

3. Interfejs graficzny programu

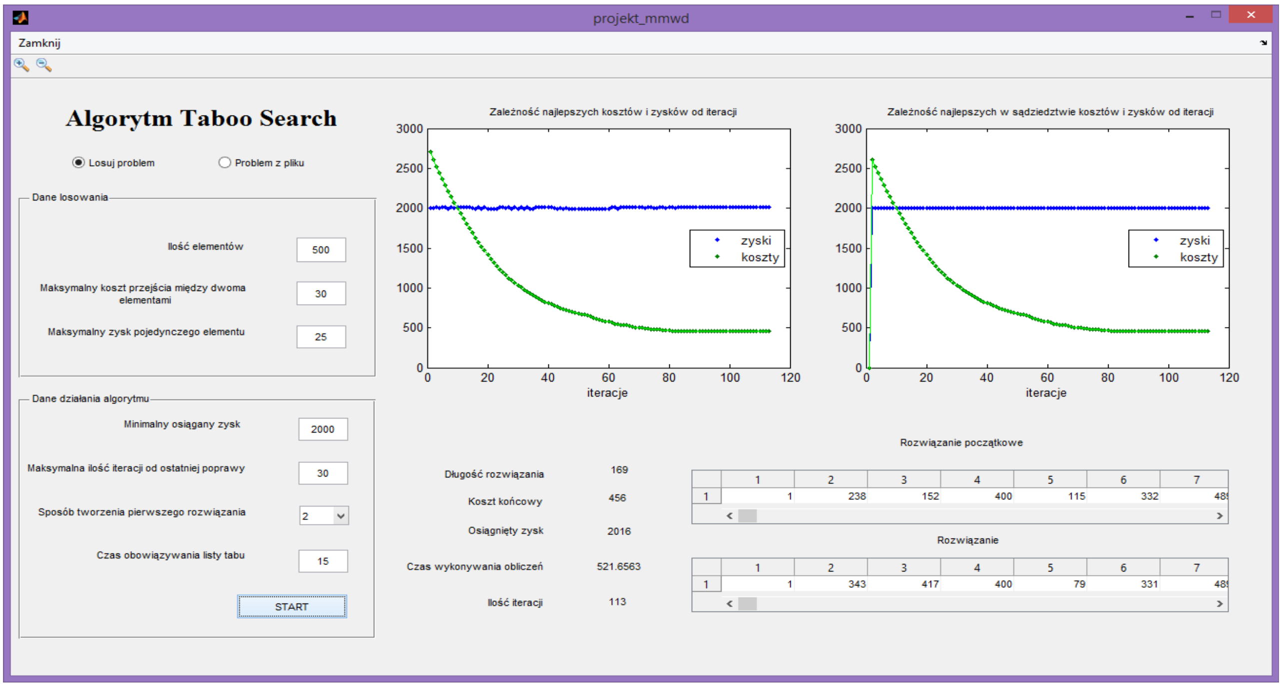
Aby każdy użytkownik mógł bez przeszkód korzystać z programu, opracowany został interfejs graficzny(GUI) w środowisku MATLAB. Pozwala on zarówno na podanie wartości początkowych dla algorytmu, jak i na obserwację wyników. Poniżej przedstawiony został przykładowy interfejs programu z rozwiązaniami trzech różnych problemów.



Rysunek 1: Przykładowy interfejs wraz z rozwiązaniem problemu



Rysunek 2: Przykładowy interfejs wraz z rozwiązaniem problemu 2



Rysunek 3: Przykładowy interfejs wraz z rozwiązaniem problemu 3

Pierwszym istotnym elementem interfejsu są przyciski służące do wyboru sposobu tworzenia problemu: może być wczytany z pliku lub wylosowany. Gdy zaznaczona zostaje opcja "losowanie", widoczny staje się panel o nazwie "Dane losowania". Służy on do wpisania danych potrzebnych do wygenerowania problemu. Użytkownik jest proszony o podanie ilości elementów- w naszym przypadku diamentów, maksymalnego kosztu przejścia między dwoma elementami oraz maksymalnego zysku z wydobycia danego diamentu. Dzięki temu program wylosuje macierz kosztów i wektor zysków, które następnie zapisywane są do pliku arkusza kalkulacyjnego.

Drugi panel "Dane działania algorytmu" pozwala na wpisanie ograniczenia: minimalnego zysku, a także maksymalnej ilości iteracji od ostatniej poprawy będącej warunkiem stopu i czasu obowiązywania listy tabu. Ponadto można wybrać sposób utworzenia pierwszego rozwiązania.

Gdy algorytm zakończy swoje działanie, wyświetlony zostaje wynik: u dołu okna pojawia się wektor rozwiązania początkowego i końcowego, jego długość, czyli ilość zebranych diamentów, wartość funkcji celu(koszt końcowy), osiągnięty zysk, czas wykonywania obliczeń oraz ilość przeprowadzonych iteracji. Pozwala to na weryfikację dopuszczalności rozwiązania. Co więcej, dzięki wykresom użytkownik może śledzić, jak zmieniały się zyski i koszty w trakcie wykonywania programu.

4. Przykład problemu o znanym rozwiązaniu optymalnym

Aby zweryfikować poprawność algorytmu, oprócz dokonywania obliczeń przez napisany program, rozwiązano ręcznie problem o niewielkim rozmiarze. Poniżej przedstawione zostały efekty pracy.

Tabela 1. Macierz kosztów badanego problemu

	1	**2**	**3**	**4**	**5**
1	∞	7	8	2	1
2	4	∞	1	5	2
3	5	2	∞	3	4
4	8	4	1	∞	1
5	3	5	7	2	∞

Tabela 2. Wektor zysków badanego problemu

1	5
2	1
3	3
4	8
5	2

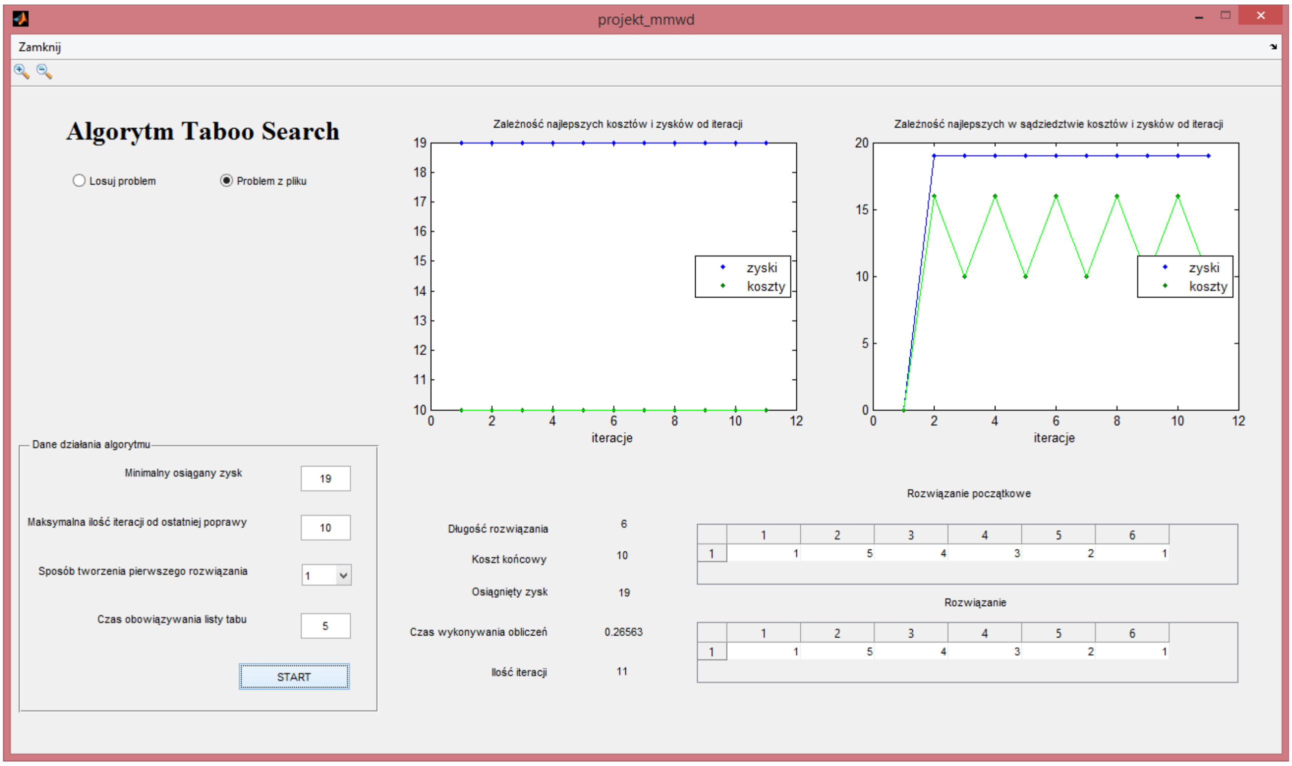
Generowanie rozwiązania początkowego:

Na podstawie macierzy kosztów (jak w sposobie 1 w algorytmie):

wiersz 1: 1 → 5 ($k_1 = 1$)
wiersz 5: 5 → 4 ($k_2 = 2$)
wiersz 4: 4 → 3 ($k_3 = 1$)
wiersz 3: 3 → 2 ($k_4 = 2$)
wiersz 2: 2 → 1 ($k_5 = 4$)

Rozwiązanie dla ograniczenia 19: 1 → 5 → 4 → 3 → 2 → 1 (koszt = 10, zysk = 19)

Rozwiązanie wyznaczone przez program:



Rysunek 4: Interfejs programu z wyznaczonym rozwiązaniem omawianego problemu

Jak widać, rozwiązanie wyznaczone przez program jest zgodne z obliczonym ręcznie.

5. Biblioteka zadań testowych

Działanie wykonanego programu opiera się na losowym generowaniu problemów o zadanych parametrach. Dzięki zapisywaniu danych do arkusza kalkulacyjnego możliwe było stworzenie biblioteki zadań testowych, która znajduje się w załączeniu do niniejszego projektu. Zawiera ona 10 przykładowych zadań o różnych parametrach- wygenerowane macierze kosztów i wektory zysków oraz wyniki działania algorytmu taboo search dla każdego z nich.

6. Statystyka testów

Statystyka testów została wykonana w celu obserwacji rozwiązań i porównania wpływu zmiany zadanych parametrów na działanie algorytmu.

a) Wpływ sposobu generowania rozwiązania początkowego na rozwiązanie końcowe

Wykorzystano 4 problemy następujących parametrach:

- Dane losowania:
  - Maksymalny koszt przejścia: 200
  - Maksymalny zysk dla elementu: 150
- Dane algorytmu:
  - Minimalny zysk całkowity: 1000
  - Maksymalna ilość iteracji od ostatniej poprawy: 15
  - Czas obowiązywania listy tabu: 10

Tabela 1: Porównanie rozwiązania końcowego dla dwóch sposobów wyznaczania rozwiązania początkowego

Rozmiar problemu	**50**		**100**		**200**		**300**
Sposób	**1**	**2**	**1**	**2**	**1**	**2**	**1**
Długość rozwiązania	37	22	11	13	22	17	19
Koszt końcowy	209	773	104	339	178	450	108
Osiągnięty zysk	1024	1000	1073	1103	1063	1000	1000
Czas obliczeń	0,2496	0,1404	0,1092	0,1092	0,4212	0,2496	0,39
Ilość iteracji	15	23	18	21	15	23	15

Jak widać w powyższej tabeli, sposób wyznaczenia rozwiązania początkowego ma wpływ przede wszystkim na ilość iteracji. W każdym rozważanym przypadku, dla sposobu 1 jest ona mniejsza, niż dla drugiego/ Co więcej, przy maksymalnej ilości iteracji od ostatniej poprawy równej 15, wynik uzyskany I metodą w większości problemów nie zostaje przekształcony przez algorytm, gdyż uzyskana ilość iteracji równa się ilości iteracji od ostatniej poprawy.

Można również zauważyć, że końcowa wartość funkcji celu jest większa przy zastosowaniu drugiego sposobu kreowania rozwiązania początkowego. Jest to zrozumiałe, gdyż ta metoda, w odróżnieniu od pierwszej, jest w większym stopniu losowa i nie uwzględnia kosztów.

b) Wpływ czasu obowiązywania listy tabu na rozwiązanie

Pierwszy problem wykorzystany do testu:

- Dane losowania:
  - Ilość elementów: 200
  - Maksymalny koszt przejścia: 200
  - Maksymalny zysk dla elementu: 150
- Dane algorytmu:
  - Minimalny zysk całkowity: 1000
  - Maksymalna ilość iteracji od ostatniej poprawy: 15

Dla każdego wywołania algorytmu wykorzystano to samo rozwiązanie początkowe- wygenerowane przed wykonaniem testu za pomocą sposobu 2. Dzięki temu zminimalizowano czynnik losowy i zwiększono efektywność statystyki.

Czas obowiązywania listy tabu w tabeli oznaczono literą T.

Tabela 2: Porównanie rozwiązania dla różnych czasów obowiązywania listy tabu- Problem 1

T	Długość rozwiązania	Koszt końcowy	Osiągnięty zysk	Czas obliczeń	Ilość iteracji
5	20	679	1000	0,39	23
10	20	679	1000	0,4212	23
20	21	660	1001	0,3588	24
30	22	662	1001	0,3588	24
40	22	662	1001	0,4368	24

Drugi problem wykorzystany do testu:

- Dane losowania:
  - Ilość elementów: 1000
  - Maksymalny koszt przejścia: 100
  - Maksymalny zysk dla elementu: 150
- Dane algorytmu:
  - Minimalny zysk całkowity: 5000
  - Maksymalna ilość iteracji od ostatniej poprawy: 50

Tabela 3: Porównanie rozwiązania dla różnych czasów obowiązywania listy tabu- Problem 2

T	Długość rozwiązania	Koszt końcowy	Osiągnięty zysk	Czas obliczeń	Ilość iteracji
5	75	796	5059	59,7172	82
10	75	796	5098	60,0604	82
40	75	796	5059	59,7016	82

Na podstawie powyższych tabeli można stwierdzić, iż długość trwania listy tabu, czyli ilość iteracji, dla których dane zabronienie jest aktualne, ma wpływ na rozwiązanie szczególnie dla problemu o mniejszym rozmiarze. Widzimy, iż dla 200 elementów wraz ze zwiększeniem tego parametru zwiększa się długość rozwiązania, ilość iteracji oraz zysk przy zmniejszeniu się funkcji celu, co jest zjawiskiem korzystnym. Test dla problemu o rozmiarze 5 razy większym wykazał brak zmiany rozwiązania.

c) Wpływ maksymalnej ilości iteracji od ostatniej poprawy na rozwiązanie

Pierwszy problem wykorzystany do testu:

- Dane losowania:
  - Ilość elementów: 200
  - Maksymalny koszt przejścia: 200
  - Maksymalny zysk dla elementu: 150
- Dane algorytmu:
  - Minimalny zysk całkowity: 1000
  - Czas obowiązywania listy tabu: 10

Maksymalną ilość iteracji od ostatniej poprawy w tabeli oznaczono literą M.

Tabela 4: Porównanie rozwiązania dla różnych ilości iteracji od ostatniej poprawy

M	Długość rozwiązania	Koszt końcowy	Osiągnięty zysk	Czas obliczeń	Ilość iteracji
1	24	1698	0	0,2028	1
2	24	787	1000	0,312	11
3	24	787	1000	0,312	12
4	24	787	1000	0,3588	13
10	24	787	1000	0,468	19
20	24	787	1000	0,5772	29
30	24	787	1000	0,87361	39

Jak wskazuje powyższa tabela, wpływ maksymalnej ilości iteracji od ostatniej poprawy na rozwiązanie jest niewielki.

d) Badanie rozwiązania zadań dla różnych wartości ograniczenia

Główny ograniczeniem narzuconym na problemy jest minimalny osiągany zysk. Ten parametr jest edytowalny w interfejsie graficznym, dzięki czemu można zbadać jego wpływ na rozwiązanie.

Zestaw problemów wykorzystany do testu:

- Dane losowania:
  - Ilość elementów: 100
  - Maksymalny koszt przejścia: 100
  - Maksymalny zysk dla elementu: 100
- Dane algorytmu:
  - Maksymalna ilość iteracji od ostatniej poprawy: 20
  - Czas obowiązywania listy tabu: 10
  - Sposób tworzenia pierwszego rozwiązania: 2

Tabela 5: Porównanie rozwiązania dla różnych wartości ograniczenia- grupa problemów 1

Minimalny osiągany zysk	Długość rozwiązania	Koszt końcowy	Osiągnięty zysk	Czas obliczeń	Ilość iteracji
50	3	10	96	0,1248	21
100	5	59	101	0,1092	21
200	6	74	242	0,2652	22
300	12	216	302	0,1248	25
400	11	241	412	0,1872	23
500	12	120	543	0,1872	29
600	18	314	602	0,4056	31
700	12	207	703	0,2028	25
800	23	448	853	0,3588	29
900	23	363	934	0,4212	31
1000	20	230	1008	0,3432	27
2000	62	710	2005	3,3156	50
3000	85	1044	3011	8,9389	56
4000	137	1269	4004	28,0334	87
5000	286	2149	5012	457,3637	160

Powyższa tabela ukazuje porównanie rozwiązań dla problemów wygenerowanych na podstawie podanych wcześniej parametrów. Ze względu na dużą losowość, szczególnie dla małej wartości ograniczenia rozwiązanie ma porównywalną długość. Im większy osiągalny zysk, tym więcej elementów musi zawierać wynik, aby móc je uznać za dopuszczalne. Oczywiście powoduje to wzrost kosztu końcowego i złożoności czasowej. Dla minimalnego zysku równego 4000 algorytm nie znalazł dopuszczalnego rozwiązania. W rozważanym przypadku długość wyniku wynosi 137. Oznacza to, że algorytm powracał do odwiedzonych wcześniej miejsc. W przypadku ograniczenia wynoszącego 5000 algorytm znalazł dopuszczalne rozwiązanie, lecz czas trwania obliczeń i ilość iteracji jest zdecydowanie większa, niż dla poprzednich wartości. Co więcej, długość rozwiązania jest niemal 3 razy większa od ilości elementów, co spowodowało znaczny wzrost kosztów.

Drugi test został wykonany dla jednego problemu. Dane losowania zostały wygenerowane tylko raz, a następnie, dla sprawdzenia wpływu kolejnych wartości ograniczenia na rozwiązanie, dane wczytano z arkusza kalkulacyjnego.

- Dane losowania:
  - Ilość elementów: 50
  - Maksymalny koszt przejścia: 10
  - Maksymalny zysk dla elementu: 10
- Dane algorytmu:
  - Maksymalna ilość iteracji od ostatniej poprawy: 20
  - Czas obowiązywania listy tabu: 10
  - Sposób tworzenia pierwszego rozwiązania: 2

Tabela 6: Porównanie rozwiązania dla różnych wartości ograniczenia- problem 2

Minimalny osiągany zysk	Długość rozwiązania	Koszt końcowy	Osiągnięty zysk	Czas obliczeń	Ilość iteracji
10	3	4	10	0,2652	21
20	8	10	25	0,1716	22
30	7	13	35	0,0624	22
40	14	30	47	0,0936	25
50	9	11	52	0,078	24
60	12	26	63	0,078	25
70	18	50	72	0,1872	25
80	22	45	82	0,1872	29
90	20	43	98	0,156	27
100	29	68	108	0,3276	30
150	38	79	155	0,4992	30
200	61	114	203	1,4664	47
250	121	203	250	14,83	69
300	-	-	-	-	-

Powyższa tabela doskonale ukazuje zmianę rozwiązania w zależności od wielkości ograniczenia. Wraz ze zwiększaniem minimalnego koniecznego do osiągnięcia zysku, zgodnie z oczekiwaniami wzrasta długość rozwiązania, końcowy koszt i liczba iteracji.

Dla minimalnego zysku równego 300 nie istnieje rozwiązanie dopuszczalne problemu, gdyż suma zysków ze wszystkich elementów wynosi 281.

7. Przypadki "złośliwe"

a) Błędne dane- zbyt duże ograniczenie

Ustalając dane służące do wygenerowania programu, szczególnie ilość elementów i maksymalny zysk dla danego przedmiotu, należy zwrócić uwagę na odpowiedni dobór ograniczenia. Jeśli podamy zbyt duży oczekiwany minimalny zysk, program nie będzie działał właściwie, gdyż rozwiązanie dopuszczalne nie istnieje.

Opisana wyżej sytuacja zaistniała w punkcie 6d, dla rozważania problemu o 50 wierzchołkach i maksymalnym zysku dla elementu(podczas losowania) równemu 10. Największa możliwa wartość ograniczenia wynosi 500. Jednakże jego spełnienie jest możliwe tylko wówczas, gdy każdy wylosowany diament ma przypisany maksymalny zysk. Dla wygenerowanego problemu suma zysków dla wszystkich elementów wyniosła 281, więc dla ograniczenia o wartości 300 nie da się wyznaczyć rozwiązania.

8. Podsumowanie

Zaimplementowana przez nas aplikacja pozwala na obserwację działania algorytmu taboo search dla wielu różnych parametrów wejściowych i ograniczeń i parametrów programu. Rozważany przez nas model kopalni został zrealizowany w algorytmie w sposób bardzo uniwersalny, przez co możliwe jest zastosowanie programu do różnych problemów świata rzeczywistego przy odpowiednim przygotowaniu danych wejściowych. Prosty i przyjazny interfejs użytkownika w przejrzysty sposób ukazuje wyniki działania.

Można zaobserwować, że wynik działania programu zależny jest znacznie od sposobu generowania pierwszego rozwiązania. Związanie jest to z faktem, że algorytm nie przeszukuje dokładnie wszystkich możliwych rozwiązań, a jedynie pewien obszar, znacznie zależny od rozwiązania początkowego. Przy generowaniu losowym wynik jest znacznie gorszy, niż przy wykorzystaniu opisanego wcześniej sposobu pierwszego, jednak pozwala on na obserwację ciekawszego przebiegu pracy algorytmu, w związku z tym częściej używany był przez nas do testów. Należy zawsze pamiętać, że odnajdywane przez algorytm rozwiązanie rzadko jest prawdziwie optymalne, a jedynie pseudo-optymalne i jego jakość wysoce zależna jest od właściwej generacji rozwiązania pierwszego.

9. Kod programu

Poniżej znajduje się kod głównej części programu. Implementacja interfejsu użytkownika została pominięta.

```
function [dlugosc1, rozwiazanie, koszt1, zysk8, czas, ilosc_it, rozw_pocz] =
tabusearch (il_elem, max_koszt_ele, max_zysk_ele, min_os_zysk, max_il_it, sp_tw,
czas_ob_tabu, axes1, axes2)


start_time = cputime; %wczytuję aktualny czas
nn=il_elem;% ilosc diamentow- wielkosc zloza
ma=max_zysk_ele; %max zysk z jednego elementu
%mb=10; %max wspolrzedna
mc=max_koszt_ele; %max koszt przejscia miedzy elementami
zysk_zadany = min_os_zysk; %to będziemy pobierać z GUI
sumazysku=0;
zadana_wartosc_iteracji=max_il_it;
wybierz_sposob=sp_tw;
max_tabu_shrt = czas_ob_tabu; %do pobrania z GUI!
ilosc_iteracji=1;

% Aśka zmienila -------------------

while sumazysku<zysk_zadany
    for i=1:nn
       zysk0(i)=randi(ma,1); %losujemy zysk, 1 liczba od 1 do ma
    sumazysku=sumazysku+zysk0(i);
   end
end
%wyznaczanie macierzy kosztow
for i=1:nn
    for j=1:nn
        koszt0(i,j)=randi(mc,1);
    end
    koszt0(i,i)=10^6;  %bardzo duza liczba, zeby uniemozliwic przejscie i-i%end
end

%Wczytywanie macierzy kosztów
mac_k = koszt0;
mac_k_orig = mac_k; %zachowuję macierz oryginalną
dim = size(mac_k,1); %wczytuję wymiar macierzy
dim2 = size(mac_k); %wektor obu wymiarów macierzy
%Wczytywanie wektora zysków
mac_z = zysk0;
mac_z_orig = mac_z;
tabu_index=1;
tabu_long_index=1;



%Wykluczenie przejścia a->a
for i=1:dim
    mac_k(i,i)=10e+06;
end

%Inicjalizacja parametrów
mac_k1 = mac_k;
droga = zeros (dim2); % inicjalizuję macierz wyjściową
```

```
koszt = 0;
zysk = 0;

min_k = []; %minimalny koszt cząstkowy
rozw = []; %wektor rozwiania
best_nbr_cost = 0;
best_nbr = []; %wektor najlepszego rozw w sąsiedztwie


%Inicjalizacja pierwszego rozwiązania
%Tu wprowadzić może kilka wariantów? Np. Losowy, po kolei i po kolejnym
%najmniejszym koszcie.

%Generowanie pierwszego rozwiązania po kolejnym min. koszcie
k=1; %zaczynamy od punktu 1 - wejście do kopalni
i=1;
rozw(i)=1;
i=i+1;
mac_z1=mac_z;
zysk=mac_z1(1);
mac_z1(1)=0;
licznik=0; %zysk_since_last_improvment

%wyszukujemy po kolei w pętli:
if wybierz_sposob==1
    while zysk < zysk_zadany
        min_k(i)=min(mac_k1(k,:)); %znajdujemy najtańsze przejście
        rozw(i) = find((mac_k1(k,:)==min_k(i)),1); %znajdujemy jego współrzędne
        koszt = koszt + min_k(i); %aktualizacja kosztu całkowitego
        k = rozw(i);%przechodzimy do wybranego punktu
        zysk1 = zysk + mac_z1(k);
        if zysk1==zysk
            licznik=licznik+1;
        end
        if licznik==5
            k=randi(nn,1);
            licznik=0;
        end
        mac_z1(k)=0; %zysk zero, gdy już tam byliśmy


        zysk=zysk1;
        mac_z1=mac_z;
        mack_1=mac_k;
        for q=1:length(rozw)
            mac_z1(rozw(q))=0;
        end

        %nie zabraniamy powrotu

        %droga(rozw(i-1),rozw(i)) = 1; %wpisujemy 1 w macierzy drogi
        i=i+1;

    end
elseif wybierz_sposob==2
    while zysk < zysk_zadany
          k=randi(nn,1);
          rozw(i)=k;
          koszt = koszt + mac_k1(rozw(i-1),k);
          zysk1 = zysk + mac_z1(k);
          if zysk1==zysk
              licznik=licznik+1;
```

```
        end
        if licznik==10
           k=randi(nn,1);
        else
           mac_z1(k)=0; %zysk zero, gdy już tam byliśmy
        end

        zysk=zysk1;
        i=i+1;
        mac_z1=mac_z;
        mac_k1=mac_k;
       for q=1:length(rozw)
           mac_z1(rozw(q))=0;
       end

    end
end

rozw(i) = 1; % po uzyskaniu zysku wracamy do wyjścia
%droga(rozw(i), rozw(i+1)) = 1; %wpisujemy drogę
rozw_pocz=rozw;
koszt = koszt + mac_k1(k,1);% uaktualniamy koszt o nasze wyjście
dlugosc = length(rozw); %ilość zebranych elementów
xlswrite('poczatkowe.xlsx',rozw_pocz);
xlswrite('koszty.xlsx',koszt);
for i=1:dlugosc
    tabu_shrt(tabu_index,i)=rozw(i);
end

tabu_shrt_term(tabu_index)=max_tabu_shrt;
tabu_index=tabu_index+1;
for i=1:dlugosc
    tabu_long(tabu_long_index,i)=rozw(i);
end
tabu_long(tabu_long_index)=1;
tabu_long_index=tabu_long_index+1;


%Teraz zaczynamy optymalizować nasze pierwsze rozwiązanie
crnt_rozw = rozw; %obecne rozwiązanie
best_rozw = rozw; %najlepsze rozwiązanie
best_koszt = koszt;
crnt_koszt = koszt; %ggaepsong!
crnt_zysk = zysk;
best_zysk = zysk;
crnt_dlugosc = dlugosc;
mac_k1=mac_k;
mac_z1=mac_z;
wektor_zyskow(ilosc_iteracji)=zysk;
wektor_kosztow(ilosc_iteracji) = koszt;
best_nbr_zysk=zysk;
best_nbr_cost=koszt;

iter_snc_last_imprv = 0; %iteracje od ostatniej poprawy

%%TU SIĘ ZACZYNA PĘTLA OD ITERACJI
while iter_snc_last_imprv<zadana_wartosc_iteracji
    iter_snc_last_imprv=iter_snc_last_imprv+1;
    ilosc_iteracji=ilosc_iteracji+1;
    %Tworzenie sasiedztwa
```

```
    best_nbr_cost = 10000000;
    nbr_rozw = crnt_rozw;

    %tworzenie sąsiedztwa poprzez zamianę par
    for i=2:length(nbr_rozw)-2
        for k=2:length(nbr_rozw)-2
            nbr_rozw = crnt_rozw;
            if k~=i
                temp_r = nbr_rozw(i);
                nbr_rozw(i)=nbr_rozw(k);
                nbr_rozw(k) = temp_r;
                %uaktualniamy macierze kosztów i zysków
                mac_z1=mac_z;
                mac_k1=mac_k;
                for q=1:length(nbr_rozw)
                    mac_z1(nbr_rozw(q))=0;
                end

                %sprawdzamy z listą tabu
                dl_t=length(tabu_shrt(i));
                if dl_t<crnt_dlugosc
                    dl_spr=dl_t;
                else
                    dl_spr=crnt_dlugosc;
                end
                yehet=0;
                for q=1:tabu_index-1
                    for w=1:dl_spr
                        if tabu_shrt(q,w)==nbr_rozw(w)
                            yehet=yehet+1;
                        end
                    end
                end
                if yehet==length(nbr_rozw)
                    yehet=0;
                else
                    crnt_nbr_cost = 0;
                    for j=1:length(nbr_rozw)-1
                        crnt_nbr_cost = crnt_nbr_cost +
mac_k1(nbr_rozw(j),nbr_rozw(j+1));
                    end
                    if crnt_nbr_cost < best_nbr_cost
                        best_nbr_cost = crnt_nbr_cost;
                        best_nbr = nbr_rozw;
                    end
                end
            end
        end
    end

    best_nbr_cost1 = best_nbr_cost;
    best_nbr1 = best_nbr;

    %pętla sprawdza wszystkie możliwe zamiany, a następnie najlepszą z nich
    %uzyskujemy jako best_nbr

    %kolejny sposób tworzenia sąsiedztwa poprzez dodanie nowego punktu

    for n = 1:dim
        nbr_rozw = crnt_rozw;
        nbr_zysk = crnt_zysk;
        nbr_dlugosc = length(nbr_rozw);
```

```
        if nbr_dlugosc>nn
            yehet=0;
        else
            a=0;
            if a==0
                for place=2:length(nbr_rozw)-1
                    mac_z1=mac_z;
                    for q=1:length(nbr_rozw)
                        mac_z1(nbr_rozw(q))=0;
                    end
                    nbr_rozw(place)=n;
                    nbr_zysk = nbr_zysk - mac_z1(place) + mac_z1(n);
                    mac_z1(place)=mac_z(place);
                    mac_z1(n)=0;
                    best_nbr_zysk = nbr_zysk;
                    %sprawdzamy, czy nie mamy zbyt małego zysku
                    if nbr_zysk < zysk_zadany
                        while nbr_zysk < zysk_zadany
                            if n==dim
                                z=-1;
                            else
                                z=1;
                            end
                            nbr_rozw(crnt_dlugosc)=n+z;
                            nbr_rozw(crnt_dlugosc+1)=1;
                            nbr_zysk=nbr_zysk+mac_z(n+z);
                            nbr_dlugosc = nbr_dlugosc+1;
                        end
                        %sprawdzamy, czy nie mamy zbyt dużego zysku
                    elseif nbr_zysk > zysk_zadany
                        najmniejszy_zysk = mac_z(nbr_rozw(2));
                        naj_punkt = 2;
                        for g = 3:length(nbr_rozw)-1
                            temp_zysk = mac_z(nbr_rozw(g));
                            if temp_zysk < najmniejszy_zysk
                                najmniejszy_zysk = temp_zysk;
                                naj_punkt = g;
                            end
                        end
                        %znaleźlismy najmniejszy zysk w rozwiązaniu
                        if (najmniejszy_zysk < nbr_zysk - zysk_zadany ||
najmniejszy_zysk~=0)

                            nbr_zysk = nbr_zysk - najmniejszy_zysk;
                            for h=naj_punkt:nbr_dlugosc-1
                                nbr_rozw(h) = nbr_rozw(h+1);
                            end
                        end
                    end


                    %obliczamy koszt tego, co nam wyszło
                    crnt_nbr_cost=0;
                    nbr_dlugosc=length(nbr_rozw);
                    for j=1:nbr_dlugosc-1
                        crnt_nbr_cost = crnt_nbr_cost +
mac_k1(nbr_rozw(j),nbr_rozw(j+1));
                    end
                     %sprawdzamy z listą tabu
                    dl_t=length(tabu_shrt(i));
                    if dl_t<crnt_dlugosc
                        dl_spr=dl_t;
                    else
```

```
                    dl_spr=crnt_dlugosc;
                end
                yehet=0;
                for q=1:tabu_index-1
                    for w=1:dl_spr
                        if tabu_shrt(q,w)==nbr_rozw(w)
                            yehet=yehet+1;
                        end
                    end
                end
                if yehet==length(nbr_rozw)
                    yehet=0;
                else
                    if crnt_nbr_cost < best_nbr_cost
                        best_nbr_cost = crnt_nbr_cost;
                        best_nbr = nbr_rozw;
                        best_nbr_zysk = nbr_zysk;
                    end
                end
            end
        end
    end
end

if best_nbr_cost1<best_nbr_cost
     best_nbr_cost=best_nbr_cost1;
     best_nbr=best_nbr1;
     %best_nbr_zysk = crnt_zysk;
end


mac_z1 = mac_z;
best_nbr_zysk = 0;
for q=1:length(best_nbr)
    best_nbr_zysk = best_nbr_zysk + mac_z1(best_nbr(q));
    mac_z1(best_nbr(q)) = 0;
end


if(best_nbr_zysk~=0||best_nbr_cost~=0) %Aśka zmienila!
    wektor_kosztow_nbr(ilosc_iteracji)=best_nbr_cost;
    wektor_zyskow_nbr(ilosc_iteracji)=best_nbr_zysk;
end


%teraz w teorii pod best_nbr mamy najlepsze sąsiedztwo i jego koszt pod
%best_nbr_cost
%jeśli koszt best_nbr jest mniejszy od obecnego best_koszt, przyjmiemy to
%jako nowe rozwiązanie



crnt_rozw=best_nbr;
crnt_koszt = best_nbr_cost;
crnt_zysk = nbr_zysk;
crnt_dlugosc = length(crnt_rozw);

if crnt_koszt<best_koszt
    best_rozw = crnt_rozw; %najlepsze rozwiązanie
    best_koszt = crnt_koszt;
    best_zysk=nbr_zysk;
    iter_snc_last_imprv=0;
```

```
    end

    %updateujemy tabulisty

    for i=1:crnt_dlugosc
        tabu_shrt(tabu_index,i) = crnt_rozw(i); %wpisujemy rozw.
    end
    tabu_shrt_term(tabu_index) = max_tabu_shrt; %kryterium aspiracji
    for i=1:tabu_index
        tabu_shrt_term(i)=tabu_shrt_term(i)-1;
        if tabu_shrt_term(i)==0
            tabu_shrt(i)=0;
        end
    end


     %wpisujemy rozw.


    tabu_index=tabu_index+1;
    end_time = cputime;
    time = end_time-start_time;
    wektor_zyskow(ilosc_iteracji)=best_zysk;
    wektor_kosztow(ilosc_iteracji)=best_koszt;

    end
%KONIEC PĘTLI ITERACJI

%dane do wyjścia funkcji
dlugosc1=length(best_rozw);
rozwiazanie=best_rozw;
koszt1=best_koszt;
zysk8=best_zysk;
czas=time;
ilosc_it=ilosc_iteracji;

%generowanie wykresów
kolejne_iteracje=1:ilosc_iteracji;
plot(axes1,kolejne_iteracje,wektor_zyskow,'.',kolejne_iteracje,wektor_kosztow,'.
',kolejne_iteracje,wektor_zyskow,'b',kolejne_iteracje,wektor_kosztow,'g');
%hold on;
xlabel(axes1,'iteracje')
legend(axes1,'zyski','koszty','Location','East')

plot(axes2,kolejne_iteracje,wektor_zyskow_nbr,'.',kolejne_iteracje,wektor_koszto
w_nbr,'.',kolejne_iteracje,wektor_zyskow_nbr,'b',kolejne_iteracje,wektor_kosztow
_nbr,'g');
%hold on;
xlabel(axes2,'iteracje')
legend(axes2,'zyski','koszty','Location','East')

xlswrite('macierz_kosztow.xlsx', koszt0);
xlswrite('wektor_zyskow.xlsx', zysk0);


end
```