Hungry Student Problem

Optymalizacja algorytmem Tabu Search - Przeszukiwania z Zabronieniami

Zespół:
Zofia Trzaskalik
Kamil Barczyński
Piotr Gołyźniak
Automatyka i Robotyka, III rok

Prowadzący:
dr inż. Piotr Kadłuczka

Spis treści

1. Opis problemu

Istotą problemu jest znalezienie planu żywieniowego dla studenta optymalizującego koszt spożywanych posiłków i czas na nie poświęcony. Dodatkowo uwzględniane są indywidualne preferencje smakowe. Problem motywowany jest rzeczywistym zagadnieniem obejmującym tygodniowy jadłospis studenta Akademii Górniczo-Hutniczej i zagadnienie obrazować będziemy głównie w oparciu o tę instancję problemu.

Jak większość problemów rzeczywistych, także i ten zależy od wielu parametrów. Stworzony model zagadnienia daje swobodę w doborze:

- liczby dni, dla których będzie poszukiwane rozwiązanie (domyślnie tydzień roboczy, czyli 5 dni),
- branych pod uwagę restauracji - ich liczby, lokalizacji, dostępności i ceny poszczególnych zestawów oraz indywidualnej oceny klienta (domyślnie 10 reprezentatywnych restauracji w najbliższej okolicy AGH),
- branych pod uwagę zestawów żywieniowych - ich liczby, kaloryczności, i indywidualnej oceny klienta (domyślnie 10 reprezentatywne zestawów),
- miejsc (budynki) i godzin zajęć (z których wynikają czas trwania *slotów* czasowych przeznaczonych na posiłek w poszczególnych dniach),
- budżetu przeznaczonego na wyżywienie (domyślny budżet tygodniowy wynosi 120 [zł]),
- energii, z jaką zaczynamy dzień (uzyskanej ze śniadania zjedzonego w domu),

- maksymalnej wartości energetycznej, jaką można posiadać w organizmie w danej chwili,
- możliwości dopuszczenia spóźnień na zajęcia (*kwadrans akademicki*),
- stopnia przyzwolenia na monotonię jedzenia (liniowy spadek zadowolenia wraz z kolejnym spożyciem danego zestawu),

Modelowanie rzeczywistości opiera się o przyjmowanie racjonalnych uproszczeń. Stworzony model powstał w oparciu o następujące założenia:

- codziennie występują 4 bloki zajęć, rozdzielone *slotami* czasowymi,
- codziennie na spożywanie posiłków przeznaczone są 3 *sloty* czasowe (o stałych porach, lecz modyfikowalnych długościach),
- w każdym ze slotów czasowych spożywamy dokładnie jeden posiłek,
- nie występuje rozróżnienie pomiędzy rodzajami posiłków (w każdym slocie można zjeść posiłek powszechnie uznawany za śniadanie/ obiad/ kolację),
- liniowe spalanie energii uzyskanej z posiłku i skokowy jej wzrost w momencie jedzenia,
- do kosztów wyżywienia nie wliczamy śniadania i kolacji jedzonych w domu.

Rozwiązanie naszego problemu prezentowane jest w postaci macierzy, w której umieszczamy podjęte decyzje którą restaurację i jaki zestaw wybieramy danego dnia w danym slocie. Nasze zmienne decyzyjne to numer restauracji i numer zestawu na którą decydujemy się w danym momencie.

Tab. 1. Przykładowa macierz rozwiązania:

	dzień 1		*dzień 2*		*dzień 3*		*dzień 4*		*dzień 5*	
	nr rest.	*nr zest.*	*nr rest.*	*nr zest.*	*nr rest.*	*nr zest.*	*nr rest.*	*nr zest.*	*nr rest.*	*nr zest.*
slot 1	1	3	3	9	5	3	3	2	1	3
slot 2	4	4	9	4	2	5	2	3	9	5
slot 3	2	7	10	2	8	3	5	3	10	6

Z przykładowego rozwiązania, przedstawionego w tab. 1 możemy odczytać np. że w pierwszym dniu w trzecim slocie czasowym zjemy zestaw nr 7 w restauracji o numerze 2. Gdy przyjmiemy naszą rzeczywistą postać problemu (tj. mamy do dyspozycji 10 restauracji, 10 zestawów i okres czasu równy 5 dni), a nasza macierz rozwiązania nazwiemy *Rozw* to można nasze zmienne decyzyjne zapisać w następujący sposób:

nr restauracji: $Rozw_{i,j}$ dla: $i = 1,2,3 \quad j = 1,3,5,7,9$
nr zestawu: $Rozw_{i,j}$ dla: $i = 1,2,3 \quad j = 2,4,6,8,10$

Jak każdy model rzeczywisty nasz także posiada pewne ograniczenia:

- ograniczenia czasowe wynikające z ograniczonych długości *slotów* między zajęciami,
- skończony budżet (np. tygodniowego), co powoduje, że w algorytmie musimy odpowiednio planować nasze wydatki patrząc perspektywicznie na cały okres badań (np. tydzień),
- dolne i górne ograniczenia energetyczne, które wynikają z minimalnego poziomu energii człowieka potrzebnego do normalnego funkcjonowania (umownie przyjęte jako 0), oraz maksymalnego poziomu wynikającego z ograniczonej ilości jedzenia jaką możemy jednorazowo spożyć.

Funkcja celu uwzględnia czas potrzebny na dojście do restauracji (wraz z przejściem z restauracji do budynku, w którym odbywają się kolejne zajęcia - czasy te obliczane są w minutach na podstawie odległości pomiędzy budynkami wyznaczonej za pomocą metryki Manhattan, zwanej także metryką taksówkową), przygotowanie i konsumpcję posiłku, jego cenę i *zadowolenie* ze zjedzenia danego posiłku. Czas przygotowania i cena zależą od wybranego zestawu i restauracji, natomiast zadowolenie uwzględnia indywidualną ocenę restauracji i zestawu, energię uzyskaną po zjedzeniu danego posiłku oraz liniową karę za monotonię żywieniową. Wszystkie trzy elementy są przed sumowaniem mnożone przez stałe normalizujące, których wartość zależy od nacisku położonego przez użytkownika czas, cenę lub zadowolenie z jedzenia i konkretnej instancji problemu - danych liczbowych takich jak ceny w restauracjach czy rozmieszczenie zajęć w poszczególnych dniach.

$$f_c(x) = \sum_{d=1}^{D} \sum_{s=1}^{3} [\, c_1 \cdot (Z_k(2) + R_r(2k+3) + l_1 + l_2) + c_2 \cdot R_r(2k+2) + c_3 \cdot (Z'_k(3) \cdot R_r(3) \cdot Z_k(1)],$$

gdzie:

d - numer dnia,

D - liczba dni,

s - numer slotu,

$c_1,\ c_2,\ c_3$ - stałe normalizujące,

r - numer restauracji,

R - macierz restauracji,

k - numer zestawu,

Z - macierz zestawów,

l_1 - czas przejścia z budynku (w którym jesteśmy przed slotem) do wybranej restauracji,

l_2 - czas przejścia z wybranej restauracji do budynku po końcu slotu.

Dane, od których przede wszystkim zależy dana instancja problemu zawarliśmy w macierzach: R - restauracji oraz Z - zestawów. Każda (r-ta) restauracja zawiera:

- $R_r(1,2)$ - położenie r-tej restauracji,
- $R_r(3)$ - indywidualna ocena r-tej restauracji,
- $R_r(2k+2)$ - cena k-tego zestawu w r-tej restauracji,

- $R_r(2k+3)$ - czas przygotowania *k*-tego zestawu w *r*-tej restauracji.

Natomiast każdy (*k*-ty) zestaw charakteryzują:

- $Z_k(1)$ - wartość odżywcza (energetyczność) *k*-tego zestawu,
- $Z_k(2)$ - czas konsumpcji *k*-tego zestawu,
- $Z_k(3)$ - indywidualna ocena *k*-tego zestawu.

W naszych rozważaniach należało jeszcze w jakiś sposób zapobiec "monotonii" jedzenia posiłków. Ten problem rozwiązaliśmy poprzez wprowadzenie nieliniowej funkcji zmniejszającej ocenę danego zestawu (poprzez przemnożenie oceny przez współczynnik c_4, $c_4 \in (0,1)$, aby zapewnić spadek zadowolenia) w momencie, gdy dany zestaw został już zjedzony (występuje już w rozwiązaniu). Wraz z częstszym jedzeniem tego samego zestawu jego ocena zmniejsza się.

$$Z_k'(3) = \begin{cases} Z_k(3), & \text{jeśli dany zestaw nie był jeszcze jedzony} \\ Z_k(3) \cdot {c_4}^i, & \text{jeśli dany zestaw był jedzony} \end{cases}$$

Gdzie: *i* - liczba określająca ilukrotnie zjedliśmy już dany zestaw

2. Algorytm Tabu Search

Pomimo - wydawać by się mogło - niewielkich rozmiarów problemu otrzymujemy (dla reprezentatywnej, rzeczywistej instancji problemu) aż 10^{30} możliwych rozwiązań (ze złożonością wykładniczą), co wyklucza zastosowanie metody przeglądu zupełnego (oszacowanie czasu potrzebnego do znalezienia rozwiązania dla pojedynczej instancji problemu pokazuje, że czas obliczeń znacznie przekracza wiek Wszechświata). Z tego powodu do poszukiwania rozwiązania problemu posłużyliśmy się algorytmem przybliżonym Przeszukiwania z Zabronieniami - Tabu Search (TS).

Istotnymi elementami algorytmu TS są definicja ruchu oraz listy tabu (*TL* - Tabu List). Ruch zdefiniowaliśmy jako zmianę w jednym z dni w jednym ze slotów wybranych restauracji lub zestawu (możliwa zmiana jednocześnie restauracji i zestawu). Lista tabu zabrania wykonywania pewnych ruchów przez określoną liczbę iteracji algorytmu. Zdefiniowaliśmy ją jako macierz o wymiarach macierzy rozwiązania, do której po każdym ruchu wpisujemy zabronienie o zadanej długości (*TT*) w miejsce, w którym nastąpił ruch (restauracja, zestaw lub restauracja i zestaw w jednym ze slotów w jednym z dni). Należy oczywiście pamiętać by po wykonaniu ruchu zmniejszyć wartość zabronienia wpisaną po poprzednich ruchach.

Ze względu na możliwość pominięcia rozwiązania suboptymalnego przez jego zabronienie zastosowaliśmy także kryterium aspiracji (*CA*) - jeśli nowe rozwiązanie jest zabronione (jest na *TL*), ale jest lepsze niż najlepsze dotychczas znalezione rozwiązanie to pomimo zabronienia wykonujemy do niego ruch.

Dopuszczalność rozwiązania jest sprawdzana na podstawie ograniczeń przyjętych w modelu (budżet *tygodniowy* mniejszy od maksymalnego zadanego, czas zużyty w danym slocie nie dłuższy niż dostępny i energia mieszcząca się z zadanym przedziale $[0, E_{max}]$.

Rozwiązanie początkowe domyślnie jest losowane, chociaż aplikacja uwzględnia możliwość zapisania rozwiązania początkowego i powtórzenia kolejnego przebiegu algorytmu dla tego samego rozwiązania startowego. W przypadku wylosowania rozwiązania niedopuszczalnego następuje próba jego poprawy (**algorytm konstrukcyjny**), wykorzystująca informację o tym, które z ograniczeń zostało naruszone. Gdy nie jest spełnione ograniczenie związane z zapłaconą kwotą, najdroższy posiłek zastępowany jest losowym tańszym (możliwość zmiany zestawu lub restauracji). Niespełnienie ograniczenia czasowego w slocie powoduje wylosowanie innego posiłku (możliwość zmiany zestawu lub restauracji) o krótszym czasie całkowitym poświęconym na jego spożycie i nie większym koszcie. Natomiast naruszenie ograniczenia związanego z energią dla slotu skutkuje wylosowaniem innego posiłku (możliwość zmiany zestawu lub restauracji), pozwalającego spełnić ograniczenie, a ponadto nie droższego od poprzedniego oraz o nie dłuższym całkowitym czasie spożycia.

Algorytm działa zadaną liczbę iteracji, jednak gdy *utknie* w jakimś obszarze (nie nastąpiła poprawa najlepszej dotychczas znalezionej wartości funkcji celu) następuje losowanie nowego rozwiązania, od którego nastąpi dalsze przeszukiwanie przestrzeni rozwiązań (wraz z ewentualną poprawą dopuszczalności wylosowanego węzła). Gdy przez zadaną liczbę iteracji nie następuje poprawa dopuszczalności rozwiązania pozwalamy na przebieg algorytmu z uwzględnieniem rozwiązań niedopuszczalnych, choć na końcu jego działania następuje sprawdzenie dopuszczalności rozwiązania.

Wersja algorytmu, którą zaimplementowaliśmy (uwzględniająca cechy naszego problemu) może zostać zapisana w postaci poniższej listy kroków (poniżej zamieszczamy wyjaśnienie przyjętych oznaczeń):

1) Określ parametry algortymu: $i = 0$, $j = 0$, LIM, LIM_{POPR}, TT, N. Wyzeruj TL. Wylosuj (lub wczytaj) rozwiązanie początkowe x_{pocz}.
2) Sprawdź dopuszczalność rozwiązania. Jeśli rozwiązanie nie jest dopuszczalne uruchom algorytm konstrukcyjny.
3) Przyjmij rozwiązanie za aktualne i oblicz dla niego wartość funkcji celu:
   $x_{wezel} := x_{min} := x_{pocz}$
   $f_{wezel} := f_{min} := f_c(x_{pocz})$
4) Dla $i = 1 \div LIM$ Wyznacz sąsiadów aktualnie przeszukiwanego rozwiązania, sprawdź ich dopuszczalność i dla dopuszczalnych wyznacz wartość funkcji celu, poszukując x_{new} oraz $x_{new-tabu}$:
   $x_{new} := arg\ min\ \{f_c(x) : x \in N(x_{wezel}) \setminus x_{wezel},\ x \notin TL\}$
   $x_{new-tabu} := arg\ min\ \{f_c(x) : x \in N(x_{wezel}) \setminus x_{wezel},\ x \in TL\}$

a) Wykonaj ruch:
   $x_{wezel} := x_{new}$

Jeśli $f_c(x_{wezel}) < f_{min}$ to

$x_{min} := x_{wezel}$

$f_{min} := f_c(x_{wezel})$

$j := 0$

b) Sprawdź kryterium aspiracji:

Jeśli $f_c(x_{new-tabu}) < f_{min}$ to

$x_{min} := x_{wezel} := x_{new-tabu}$

$f_{min} := f_c(x_{new-tabu})$

$j := 0$

c) Dokonaj korekty listy tabu:

Zmniejsz niezerowe wartości na *TL* o 1 oraz

$TL(x^i_{wezel}) := TT$

d) Jeśli $j \geq LIM_{POPR}$ to

Wylosuj nowe rozwiązanie $x_{wezel} := rand()$

Przyjęte oznaczenia parametrów algorytmu:

i - numer iteracji,

j - liczba iteracji bez poprawy,

LIM - limit iteracji,

LIM_{POPR} - limit iteracji bez poprawy najlepszej dotychczas znalezionej wartości funkcji celu $f_c(x)$,

N - sąsiedztwo rozwiązania (zależne od liczby sąsiadów),

TT (Tabu Tenure) - długość listy tabu (czas obowiązywania zabronienia),

x_{pocz} - rozwiązanie początkowe,

x_{wezel} - aktualnie przeszukiwane rozwiązanie,

f_{wezel} - wartość funkcji celu aktualnie przeszukiwanego rozwiązania,

x_{min} - rozwiązanie z najlepszą dotychczas znalezioną wartością funkcji celu,

f_{min} - najlepsza dotychczas znaleziona wartość funkcji celu,

x_{new} - rozwiązanie dopuszczalne z sąsiedztwa nie znajdujące się na *TL* z najlepszą wartością funkcji celu,

$x_{new-tabu}$ - rozwiązanie dopuszczalne z sąsiedztwa znajdujące się na *TL* z najlepszą wartością funkcji celu.

3. Aplikacja

Algorytm został zaimplementowany w środowisku MATLAB. Jego poszczególne elementy stanowią kilkanaście *m-plików*, odpowiedzialnych m.in. za obliczanie funkcji celu, poprawianie rozwiązania niedopuszczalnego czy wizualizację danych. Numeryczne wyniki działania przebiegu algorytmu są zapisywanie w plikach z rozszerzeniem *.mat*, natomiast najważniejsze parametry i wyniki prezentowane są także graficznie.

Dane wejściowe

Do jednego z *m-plików* wczytujemy dane wejściowe (macierze) w formacie **.txt* oraz zmieniamy pozostałe parametry (liczbowe). Sposób przedstawienia danych wejściowych jest zawsze jednoznaczny i ściśle określony. Poniżej przedstawiamy opis kolejnych danych wejściowych (macierzy) wraz z opisem:

Macierz restauracji - wiersze to kolejne restauracje, dwie pierwsze kolumny to położenie (x,y) restauracji, trzecia to ocena restauracji (1-10), nastepne kolumny parami to zestawy (cena i czas wykonania). W rzeczywistości nie we wszystkich restauracjach dostępne są wszystkie zdefiniowane przez nas zestawy. Sytuację taką modelujemy podaniem dla niedostępnego w danej restauracji zestawu ceny 1000 zł i czasu przygotowania 100 min. Dane te znacząco pogarszają wartość funkcji celu, jednak w przypadku trudności z wylosowaniem rozwiązania dopuszczalnego pozwalają na przetwarzanie rozwiązań niedopuszczalnych w trakcie działania algorytmu.

Przykładowy wejściowy plik macierzy restauraci:

```
526,  217,   7,    10,    5,    9,   2,    9,   2, 1000,  100,    3,   1,     9,   3, 1000, 100;
584,  277,   4,     9,    3,   11,   4,   11,   4, 1000,  100,    4,   2,    10,   2, 1000, 100;
616,  175,   3,     7,    8,   15,   5, 1000, 100,    9,    7,    3,   2,     8,   5,    2,   1;
604,   47,   8,    15,    5, 1000, 100, 1000, 100, 1000,  100, 1000, 100,  1000, 100,    3,   1;
```

Macierz zestawów - wiersze określają kolejno: kcal, czas konsumpcji (w min) i ocena(wartość od 1-10), natomiast kolumny to kolejne zestawy.

Przykładowy wejściowy plik macierzy zestawów:

```
405,  800,  600,  820,  200,  650,  100;
 10,   13,   15,   10,    4,   10,    3;
  8,    7,    6,    9,    4,    5,    2;
```

Macierz (długości) slotów czasowych - wiersze to sloty (slot 1 od godz. 10:00, slot 2 od godz. 13:00, slot 3 od godz. 16:30), a kolumny to kolejne dni. Wartości podawane są w minutach.

Przykładowy wejściowy plik z długością slotów czasowych:

```
30,    20,    25,    15,    30;
45,    60,    90,    50,    60;
15,    20,    15,    35,    15;
```

Macierz z rozmieszczeniem zajęć w ciągu tygodnia - wiersze to kolejno nr budynku: przed slot 1 (przed 10:00), po slot 1 (przed 13:00), po slot 2 (po 13:00), po slot 3 (po 16:30), a kolumny to kolejne dni.

Przykładowy wejściowy plik z rozmieszczeniem zajęć (w ciągu tygodnia):

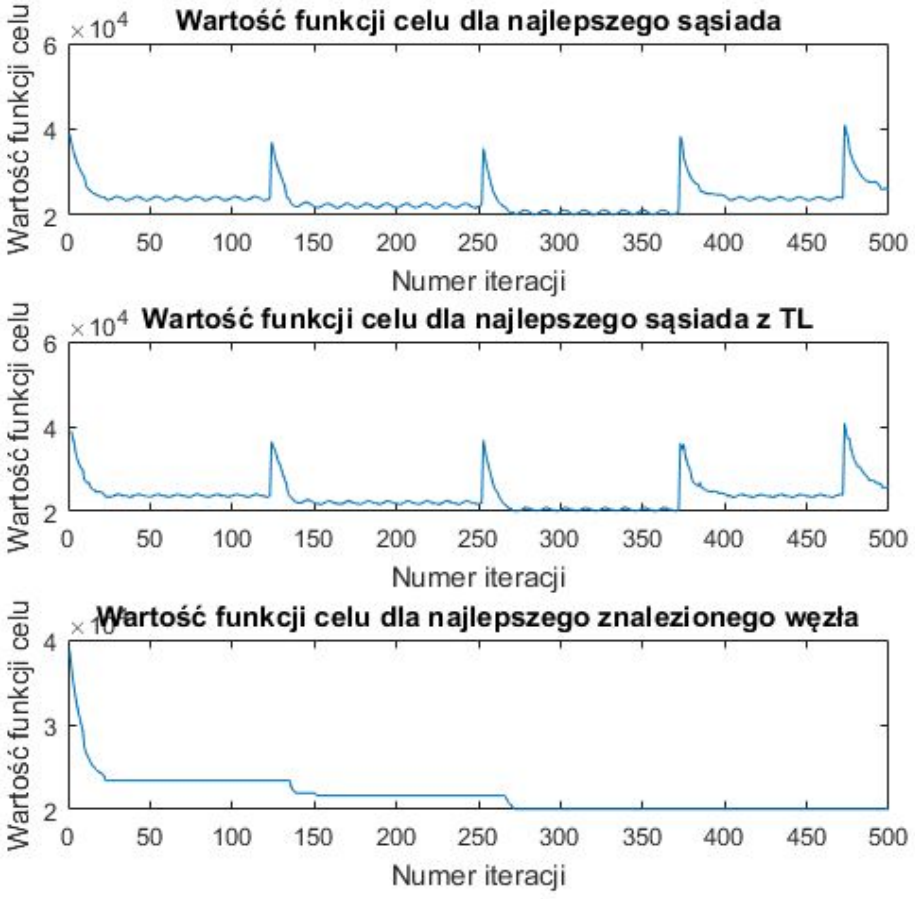
```
1,    2,    1,    5,    4;
1,    2,    3,    1,    3;
2,    5,    1,    1,    2;
3,    1,    4,    1,    1;
```

Pozostałe dane liczbowe są możliwe do zmiany w *m-pliku* w opisanych odpowiednio miejscach.

Prezentacja przebiegu algorytmu i rozwiązania

Po zakończeniu algorytmu otrzymane wyniki i przebieg algorytmu prezentowaliśmy za pomocą wielu graficznych oraz liczbowych reprezentacji.

Prezentacja zmiany funkcji celu podczas przebiegu algorytmu:

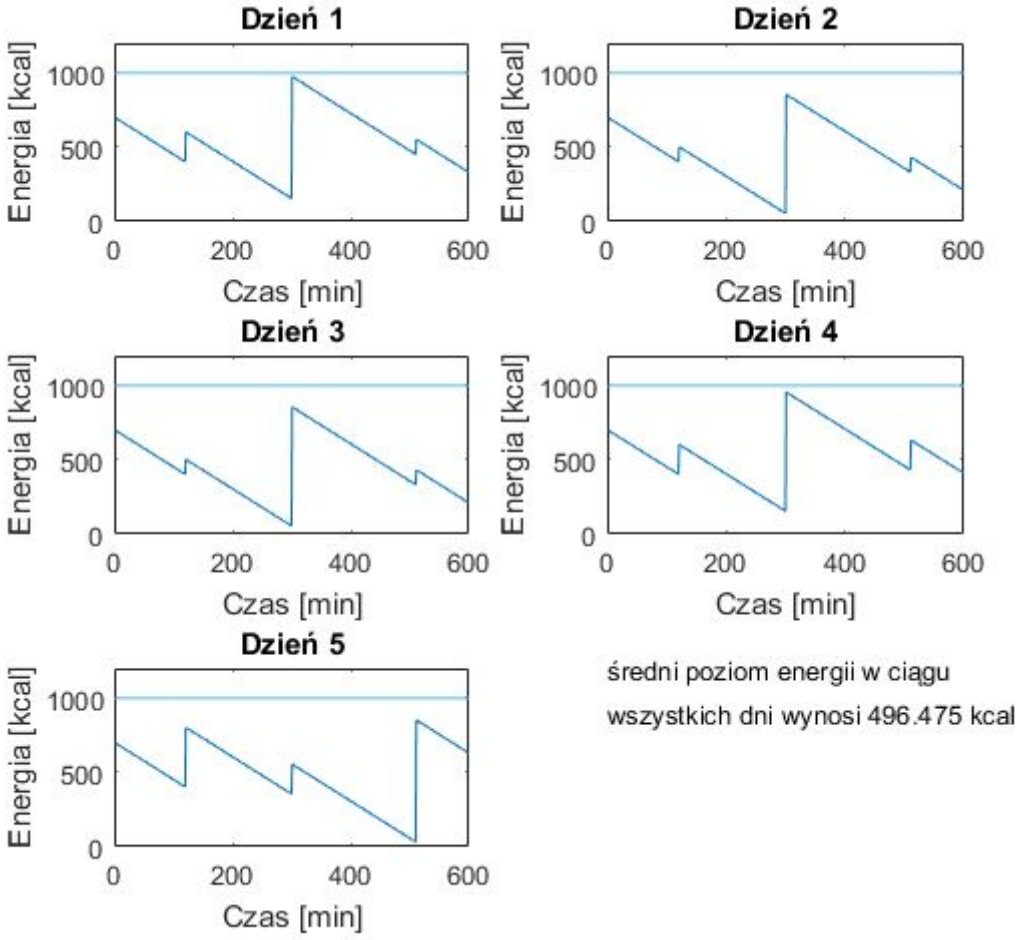


Rys. 1. Zmiana wartości funkcji celu dla przykładowego przebiegu algorytmu.

Na wykresach z rys. 1 wyświetlamy wartości funkcji celu w każdej z wykonanych iteracji dla: najlepszego sąsiada, najlepszego sąsiada znajdującego się na liście tabu i dla najlepszego znalezionego węzła. Na podstawie tych wykresów można odczytywać interesujące nas zachowania algorytmu, m. in. jak szybko osiągane jest rozwiązanie

suboptymalne, po jakim czasie widać wyraźny brak poprawy i algorytm *podejmuje decyzję* o wylosowaniu kolejnego węzła startowego albo jak głęboko (a właściwie jak wysoko, gdyż realizujemy zadanie minimalizacji i odejście od lokalnego minimum następuje poprzez *wspinanie się* w kierunku gorszych wartości funkcji celu) zostaje przeszukane sąsiedztwo danego punktu.

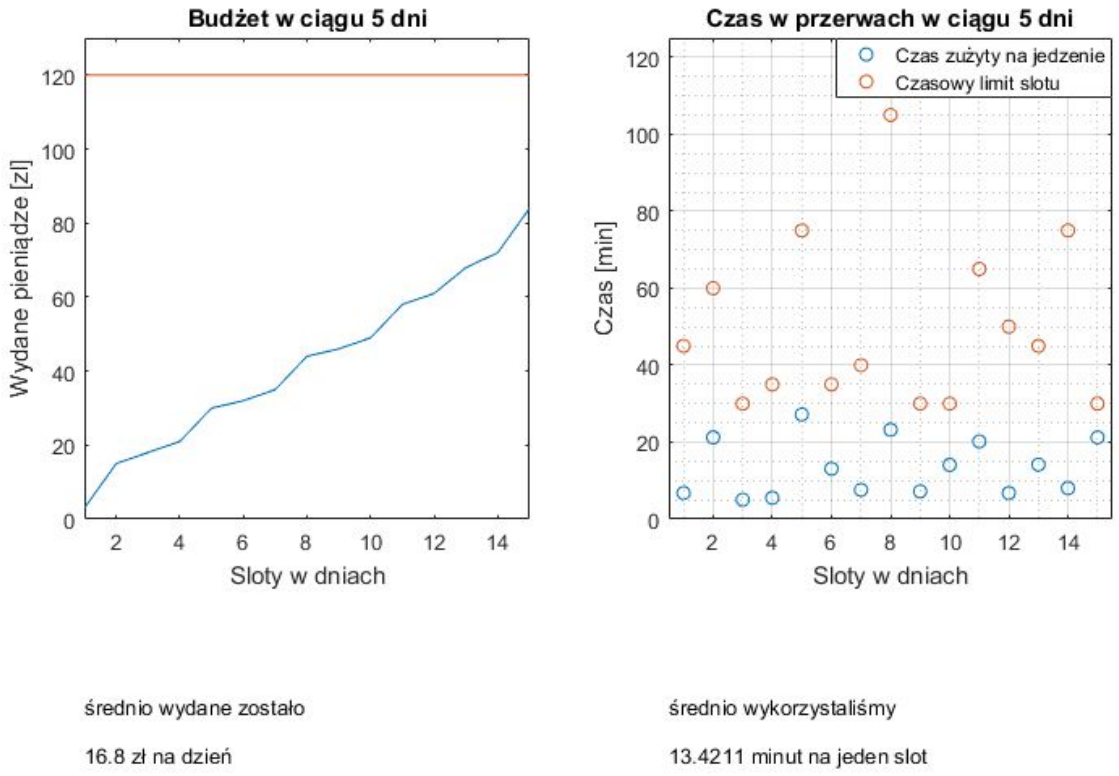
Przebieg poziomu energii każdego dnia tygodnia dla otrzymanego rozwiązania:



Rys. 2. Przykładowy przebieg poziomu energii dla każdego z dni.

Rys. 2 przedstawia wykresy energii w danych dniach - jasnoniebieską linią jest oznaczony maksymalny poziom energii, który możemy uzyskać. Przypomnijmy, że minimalny poziom energii w organizmie, jaki musimy zagwarantować został umownie oznaczony przez 0 Wyświetlamy także średni poziom energii w ciągu wszystkich dni.

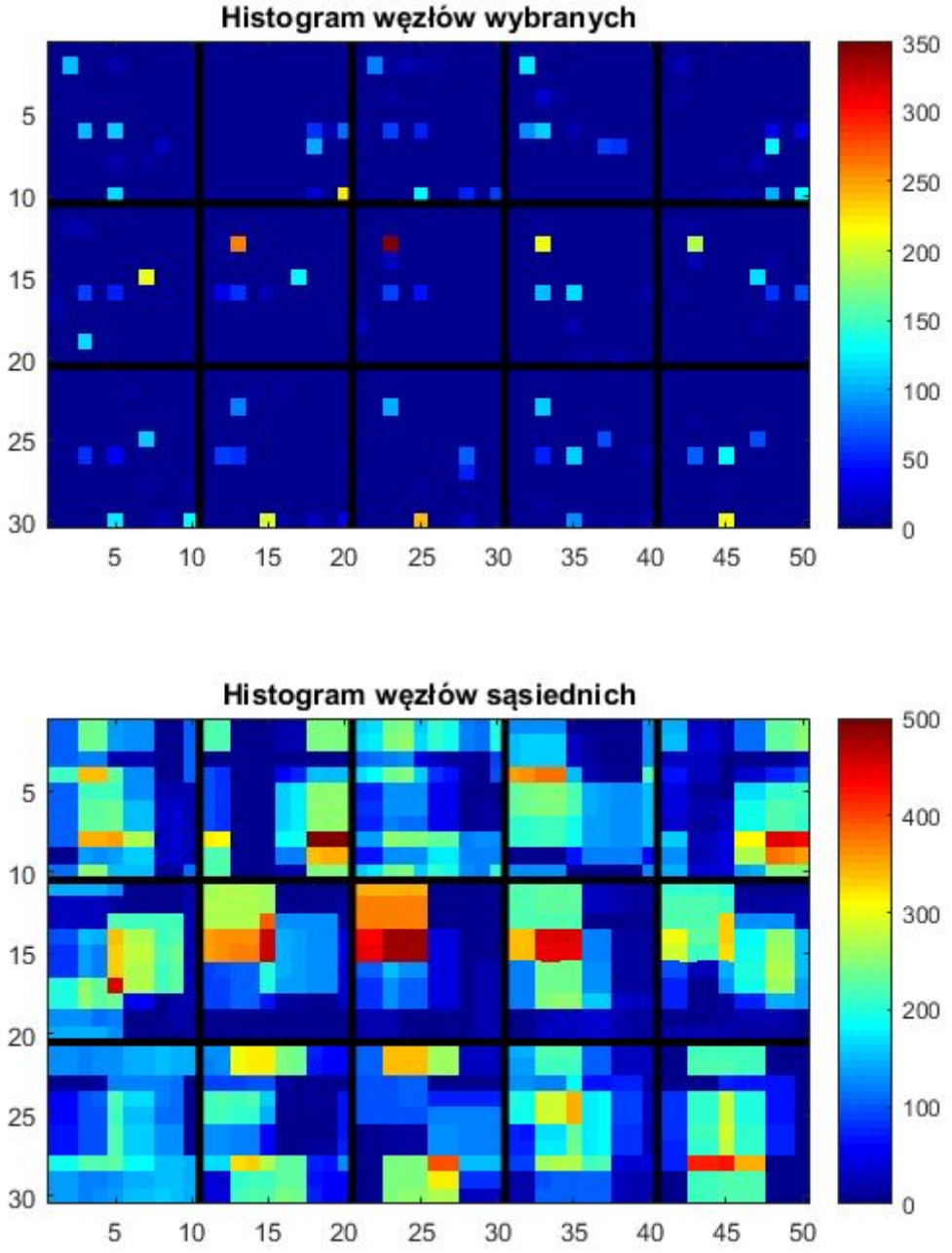
Graficzne przedstawienie zmiany w budżecie i wykorzystanego czasu podczas slotów czasowych dla otrzymanego rozwiązania:



Rys. 3. Zmiany wydanych pieniędzy w ciągu tygodnia i czasu wykorzystanego w poszczególnych slotach.

Na rys. 3. po lewej widzimy ile pieniędzy z budżetu wykorzystaliśmy w każdym ze slotów (czerwona linia to wyznaczony budżet). Poniżej znajduje się także informacja o tym ile złotych wydaliśmy średnio w ciągu każdego dnia. Po prawej widzimy długości każdego ze slotów czasowych (czerwone kropki) oraz ile z tego dostępnego czasu wykorzystaliśmy (niebieskie kropki), a poniżej średni czas w ciągu tygodnia jaki poświęcamy na zjedzenie.

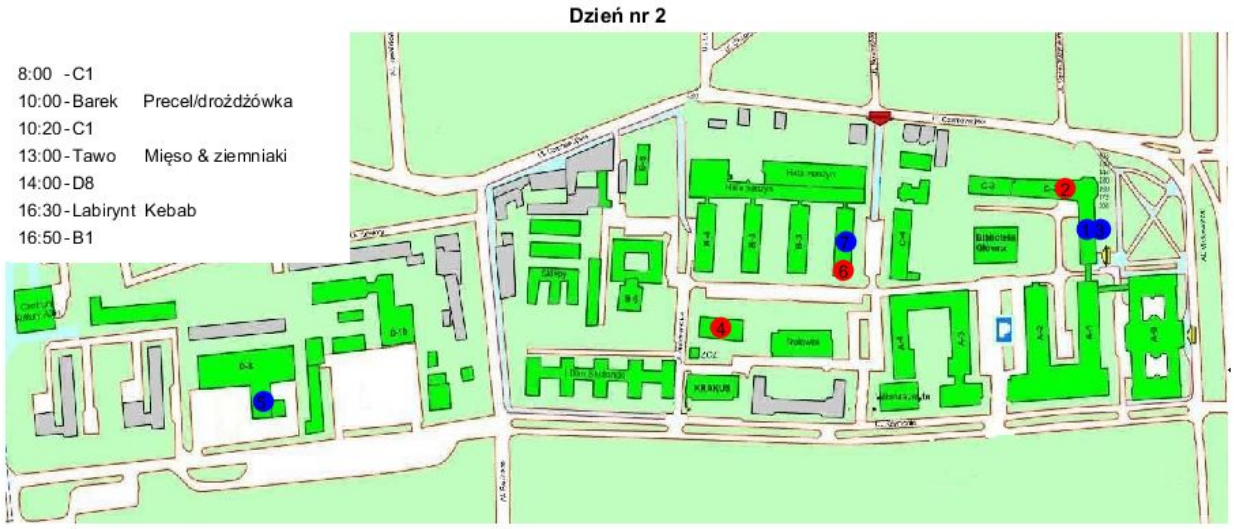
Histogramy odwiedzonych węzłów podczas przebiegu algorytmu:



Rys. 4. Przykładowy histogram węzłów odwiedzonych i przebadanych sąsiadów.

Po przebiegu algorytmu możemy uzyskać informację o tym które węzły zostały wybrane i ile razy (na górze) oraz, którzy z sąsiadów i ile razy zostali zbadani (na dole). Z tego wykresu można szczególnie zauważyć, jak wielkość sąsiedztwa ma wpływ na "szerokość" przeszukiwań (porównanie w rozdziale Testy). Można także zobaczyć, że przeszukiwane przez nas sąsiedztwo jest kwadratem (o ustalonej "długości" boku), a także, że dane sąsiedztwo nie ma ograniczeń w postaci wykraczania poza indeks, to znaczy w przypadku, gdy sąsiedztwo wykracza poza najmniejszy lub największy indeks to "przechodzi" ono odpowiednio na największy i najmniejszy indeks.

Mapa przemieszczania na dany dzień:



Rys. 5.Graficzna prezentacja rozwiązania dla przykładowego dnia.

Na podstawie otrzymanego rozwiązania mamy możliwość reprezentacji otrzymanych wyników liczbowych w formie graficznej: na każdy dzień otrzymujemy jedną mapę z zaznaczonymi miejscami zajęć (na niebiesko) i miejscami restauracji, do których się wybierzemy (na czerwono). Kropki zawierają też numerację zgodną z kolejnością odwiedzonych miejsc w ciągu dnia. Mapa jest zaopatrzona również w legendę, na której przedstawione są godziny (ramy czasowe slotów) oraz nazwy budynków i restauracji w których w danej przerwie czasowej będziemy. W każdej z restauracji mamy także określoną nazwę zestawu, który zjemy.

Dodatkowo interesującymi nas informacjami były liczba zadziałań zabronień (tzn. ile razy algorytm wykonał ruch w kierunku sąsiada innego niż gdyby usunąć mechanizm zabronień) oraz liczba zadziałań kryterium aspiracji, czyli liczba ruchów wykonanych pomimo ich zabronienia, ze względu na poprawę najlepszej dotychczas znalezionej wartości funkcji celu.

4. Testowanie

Aby skutecznie sprawdzić działanie naszego algorytmu, napisaliśmy osobny program *testowanie_testy.m*, który pozwolił nam na dowolną liczbę wywołań programu głównego *main.m*, a następnie uzyskanie z tak otrzymanych wyników interesujących nas wartości. Dzięki temu programowi mogliśmy zmieniać interesujące nas parametry algorytmu i sprawdzać, jaki miały wpływ na wyniki końcowe. Aby otrzymać wiarygodne wartości średnie, a także odchylenia standardowe wywoływaliśmy zawsze nasz program główny *main.m* 100 razy dla zadanych parametrów. Zamieszczona graficzna prezentacja rozwiązań odnosi się do przypadku, który dał najlepszą znalezioną wartość funkcji celu.

Wartości rzeczywiste

Na początku przeprowadziliśmy testy dla danych wejściowych rzeczywistych. Metodą prób i błędów dobraliśmy pewne parametry algorytmu, następnie wyniki tych, a także następnych następnych testów przedstawiamy za pomocą tabeli.

Tab. 2 Podstawowe parametry algorytmu - dane rzeczywiste

liczba restauracji	10
liczba zestawów	10
liczba dni	5
ogólna liczba wywołań algorytmu	500
Tabu Tenure	5
wielkość sąsiedztwa	5
(c_1 c_2 c_3)	(100 200 0,005)

Tab. 3 Wyniki przeprowadzonego testu - dane rzeczywiste

min. osiągnięta wartość fc	**20063,57**	**max. osiągnięta wartość fc**	**25121,17**
średnia wartość fc	**22445,16**	**odchylenie wartości fc**	**916,52**
śr. liczba zadziałań kryterium aspiracji	7,98	σ z liczby zadziałań kryterium aspiracji	3,87
śr. liczba zadziałań zabronienia	363,03	σ z liczby zadziałań zabronienia	16,52
śr. wartości energii w ciągu tygodnia	504,00	σ z wartości energii w ciągu tygodnia	20,99
śr. suma pieniędzy wydanych w slocie	17,10	σ z sumy pieniędzy wydanych w slocie	0,92
śr. czas wykorzystany w slocie	14,48	σ z czasu wykorzystanego w slocie	0,69

Wpływ liczby restauracji i zestawów

Następnie sprawdziliśmy nasz algorytm pod względem uniwersalności ze względu na liczbę restauracji i zestawów. Dla przykładu prezentujemy poniżej wyniki przeprowadzone dla **4 restauracji i 7 zestawów**. Pozostałe parametry nie zostały zmienione w stosunku do przyjętych podstawowych.

Tab. 4 Wyniki przeprowadzonego testu - 4 restauracje i 7 zestawów

min. osiągnięta wartość fc	**21927,00**	**max. osiągnięta wartość fc**	**24005,40**
średnia wartość fc	**22462,55**	**odchylenie wartości fc**	**535,92**
śr. liczba zadziałań kryterium aspiracji	2,77	σ z liczby zadziałań kryterium aspiracji	1,61
śr. liczba zadziałań zabronienia	329,17	σ z liczby zadziałań zabronienia	17,48
śr. wartości energii w ciągu tygodnia	471,81	σ z wartości energii w ciągu tygodnia	10,35
śr. suma pieniędzy wydanych w slocie	13,36	σ z sumy pieniędzy wydanych w slocie	0,28
śr. czas wykorzystany w slocie	14,22	σ z czasu wykorzystanego w slocie	0,27

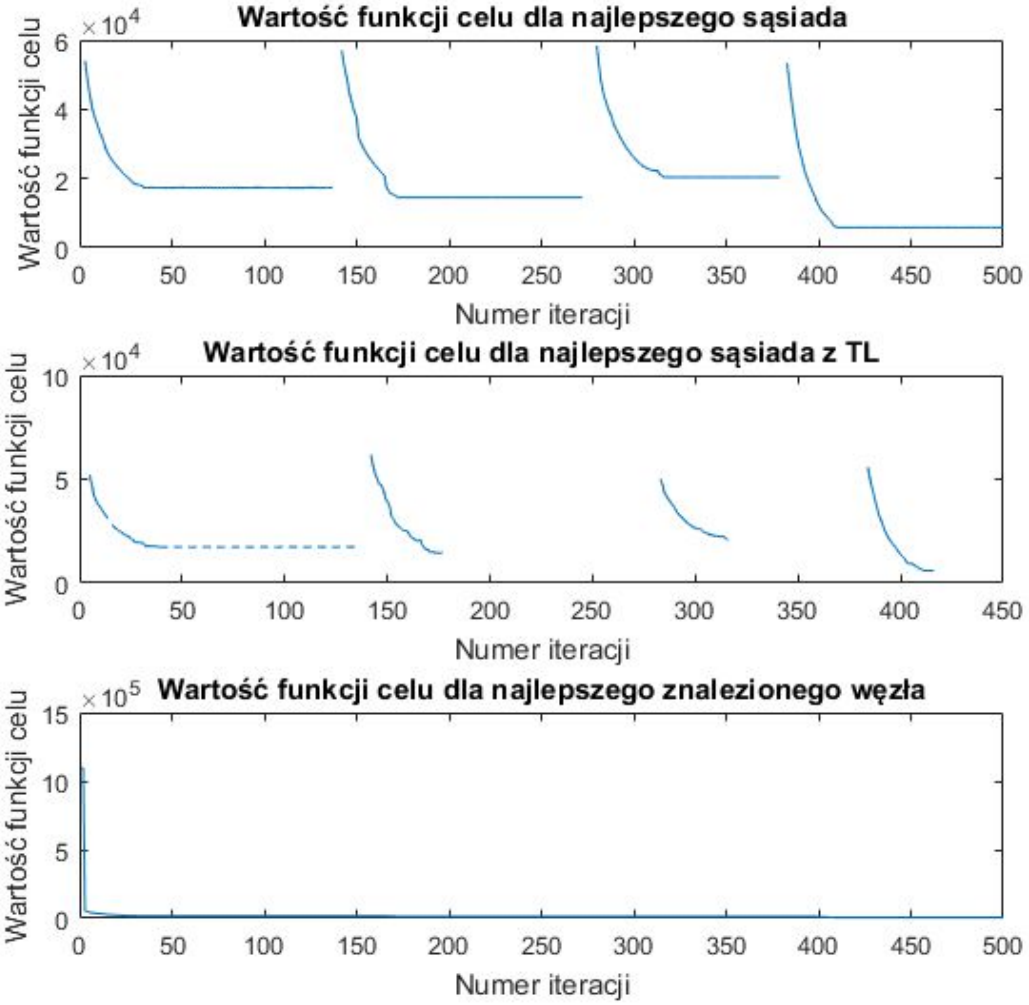
Wpływ liczby dni

Kolejnym elementem naszego zainteresowania była liczba dni. Postanowiliśmy sprawdzić jak nasz algorytm poradzi sobie z większą liczbą dni (niż podstawowa równa długości tygodnia roboczego: 5). Przedstawiamy wyniki naszego testu dla **10 dni**:

Tab. 5 Wyniki przeprowadzonego testu - 10 dni

min. osiągnięta wartość fc	**5819,48**	**max. osiągnięta wartość fc**	**15920,28**
średnia wartość fc	**11652,37**	**odchylenie wartości fc**	**2237,98**
śr. liczba zadziałań kryterium aspiracji	3,59	σ z liczby zadziałań kryterium aspiracji	2,07
śr. liczba zadziałań zabronienia	76,21	σ z liczby zadziałań zabronienia	37,74
śr. wartości energii w ciągu tygodnia	541,30	σ z wartości energii w ciągu tygodnia	18,25
śr. suma pieniędzy wydanych w slocie	21,92	σ z sumy pieniędzy wydanych w slocie	0,71
śr. czas wykorzystany w slocie	19,44	σ z czasu wykorzystanego w slocie	0,45

Podczas tego testu zaobserwowaliśmy dla najlepszego z otrzymanych wyników następujące przebiegi funkcji celu:



Rys. 6. Zmiana wartości funkcji celu.

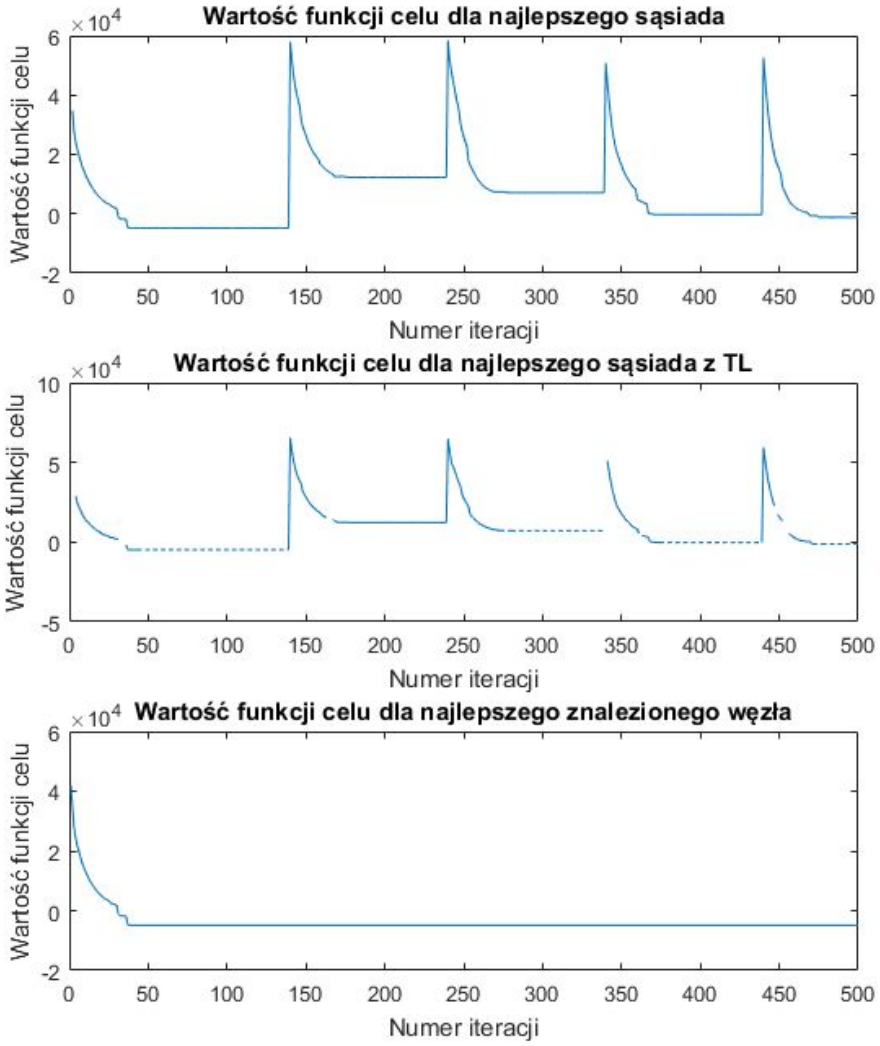
Wpływ dostępności zestawów

Na wykresie funkcji celu z rys. 6 pojawia się dużo “przerwań” co sugeruje, że jest bardzo dużo niedopuszczalnych rozwiązań dla takiego zestawu danych. Podając inne dane wejściowe: takie, w których był większy wybór zestawów w restauracjach (**każdy zestaw był dostępny w każdej restauracji**). To pozwoliło nam uzyskać następujące wyniki również dla **10 dni**:

Tab. 6 Wyniki przeprowadzonego testu - 10 dni z dostępnymi wszystkimi zestawami

min. osiągnięta wartość fc	**-4841,20**	**max. osiągnięta wartość fc**	**11681,53**
średnia wartość fc	**1950,17**	**odchylenie wartości fc**	**3045,04**
śr. liczba zadziałań kryterium aspiracji	3,10	σ z liczby zadziałań kryterium aspiracji	2,29
śr. liczba zadziałań zabronienia	83,27	σ z liczby zadziałań zabronienia	50,03
śr. wartości energii w ciągu tygodnia	542,11	σ z wartości energii w ciągu tygodnia	20,11
śr. suma pieniędzy wydanych w slocie	21,25	σ z sumy pieniędzy wydanych w slocie	0,81
śr. czas wykorzystany w slocie	20,36	σ z czasu wykorzystanego w slocie	0,53

W tym przypadku przebieg funkcji celu dla najlepszego z rozwiązań wyglądał następująco:



Rys. 7. Zmiana wartości funkcji celu.

Sprawdziliśmy także dla jak poradzi sobie algorytm, gdy **wszystkie zestawy będą dostępne we wszystkich restauracjach** a ilość zestawów, restauracji i dni będzie inna niż podstawowa. Poniżej przedstawiamy przykładowe wyniki dla **4 restauracji, 7 zestawów** i **5 dni:**

Tab. 7 Wyniki przeprowadzonego testu - 5 dni, 4 restauracji i 7 zestawów z dostępnymi wszystkimi zestawami

min. osiągnięta wartość fc	**15809,67**	**max. osiągnięta wartość fc**	**20297,07**
średnia wartość fc	**17523,26**	**odchylenie wartości fc**	**1114,45**
śr. liczba zadziałań kryterium aspiracji	2,41	σ z liczby zadziałań kryterium aspiracji	1,93
śr. liczba zadziałań zabronienia	323,64	σ z liczby zadziałań zabronienia	16,87
śr. wartości energii w ciągu tygodnia	490,21	σ z wartości energii w ciągu tygodnia	18,12
śr. suma pieniędzy wydanych w slocie	14,01	σ z sumy pieniędzy wydanych w slocie	1,15
śr. czas wykorzystany w slocie	15,57	σ z czasu wykorzystanego w slocie	0,79

Wpływ liczby iteracji algorytmu

Zbadaliśmy również wpływ wpływ limitu iteracji w naszym algorytmie. W tym celu przeprowadziliśmy testy przy danych rzeczywistych dla następujących **limitów iteracji: 10, 250, 500** (przy czym standardowo limit iteracji wynosił 500):

Tab. 8 Wyniki przeprowadzonego testu - limit iteracji: 10

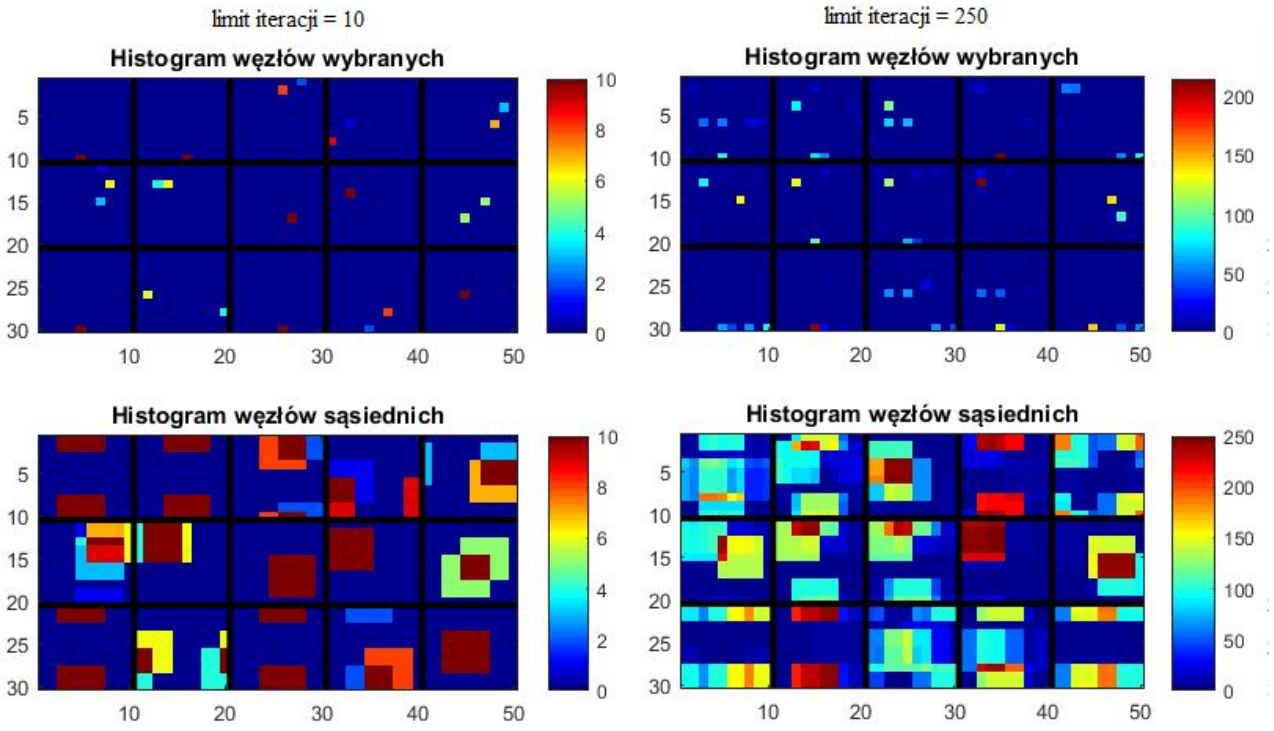
min. osiągnięta wartość fc	**23394,17**	**max. osiągnięta wartość fc**	**34544,93**
średnia wartość fc	**28107,43**	**odchylenie wartości fc**	**2071,39**
śr. liczba zadziałań kryterium aspiracji	0,92	σ z liczby zadziałań kryterium aspiracji	0,85
śr. liczba zadziałań zabronienia	0,92	σ z liczby zadziałań zabronienia	0,85

Tab. 9 Wyniki przeprowadzonego testu - limit iteracji: 250

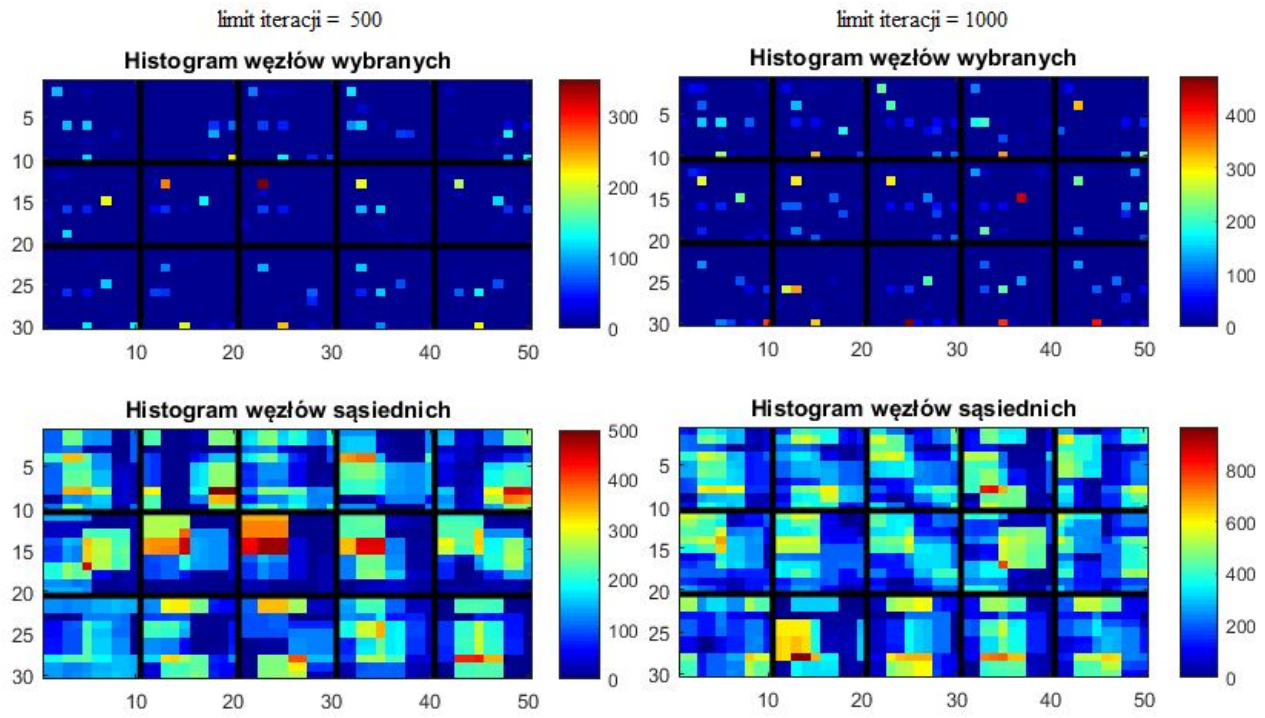
min. osiągnięta wartość fc	**19834,23**	**max. osiągnięta wartość fc**	**26890,73**
średnia wartość fc	**23171,21**	**odchylenie wartości fc**	**1185,98**
śr. liczba zadziałań kryterium aspiracji	6,57	σ z liczby zadziałań kryterium aspiracji	3,08
śr. liczba zadziałań zabronienia	175,29	σ z liczby zadziałań zabronienia	12,32

Tab. 10 Wyniki przeprowadzonego testu - limit iteracji: 1000

min. osiągnięta wartość fc	**20165,23**	**max. osiągnięta wartość fc**	**23226,50**
średnia wartość fc	**21932,96**	**odchylenie wartości fc**	**677,78**
śr. liczba zadziałań kryterium aspiracji	9,50	σ z liczby zadziałań kryterium aspiracji	4,53
śr. liczba zadziałań zabronienia	737,02	σ z liczby zadziałań zabronienia	23,24



Rys. 8. Histogram węzłów odwiedzonych i przebadanych sąsiadów.



Rys. 9. Histogram węzłów odwiedzonych i przebadanych sąsiadów.

Wpływ wielkości wartości TT

Zbadaliśmy także wpływ wartości TT (długości listy tabu) na rozwiązanie. Przeprowadziliśmy testy dla TT równego 0, 1, 2, … aż do momentu, kiedy wartość była tak duża, że algorytm nie mógł już znaleźć dopuszczalnego rozwiązania niebędącego na liście tabu. Poniżej przedstawiamy przykładowe wyniki dla **TT** równego **0, 2, 7 i 10**:

Tab. 11 Wyniki przeprowadzonego testu - TT = 0

min. osiągnięta wartość fc	**20063,57**	**max. osiągnięta wartość fc**	**24924,83**
średnia wartość fc	**22823,17**	**odchylenie wartości fc**	**984,68**
śr. liczba zadziałań kryterium aspiracji	0,00	σ z liczby zadziałań kryterium aspiracji	0,00
śr. liczba zadziałań zabronienia	0,00	σ z liczby zadziałań zabronienia	0,00

Tab. 12 Wyniki przeprowadzonego testu - TT = 2

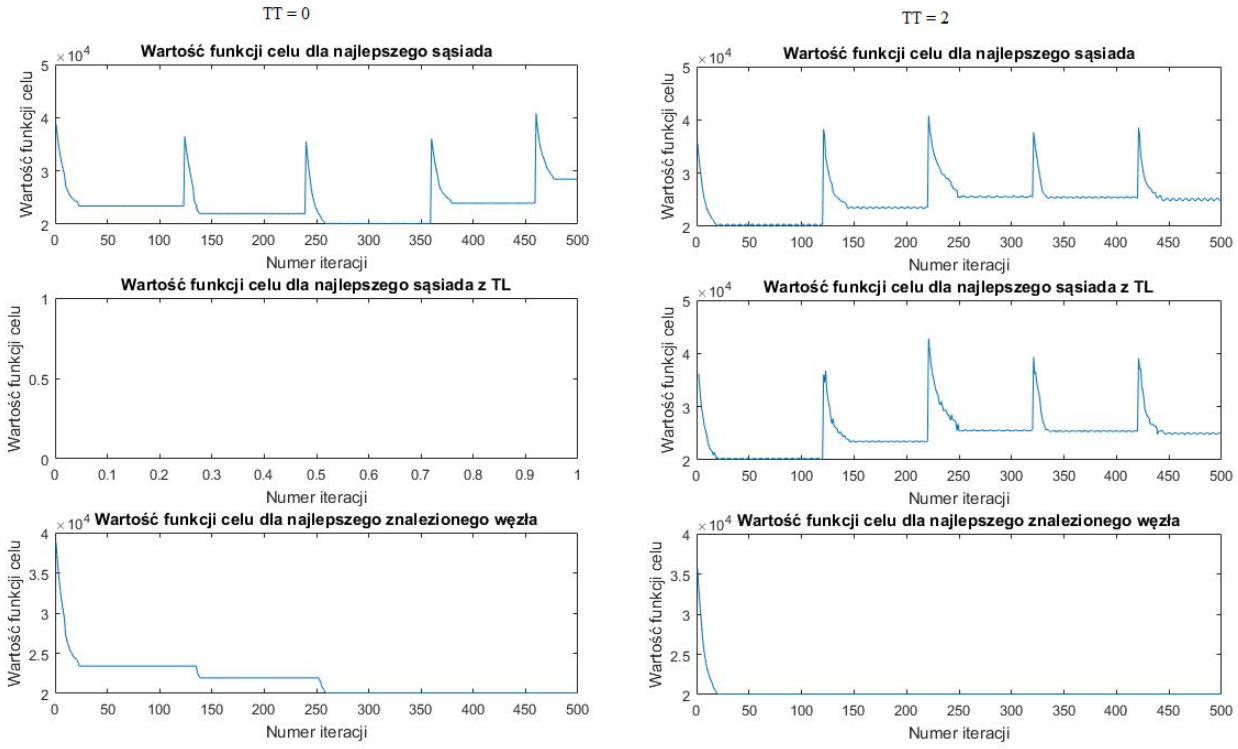
min. osiągnięta wartość fc	**20063,57**	**max. osiągnięta wartość fc**	**24924,83**
średnia wartość fc	**22632,53**	**odchylenie wartości fc**	**864,25**
śr. liczba zadziałań kryterium aspiracji	4,61	σ z liczby zadziałań kryterium aspiracji	2,59
śr. liczba zadziałań zabronienia	279,24	σ z liczby zadziałań zabronienia	18,80

Tab. 13 Wyniki przeprowadzonego testu - TT = 7

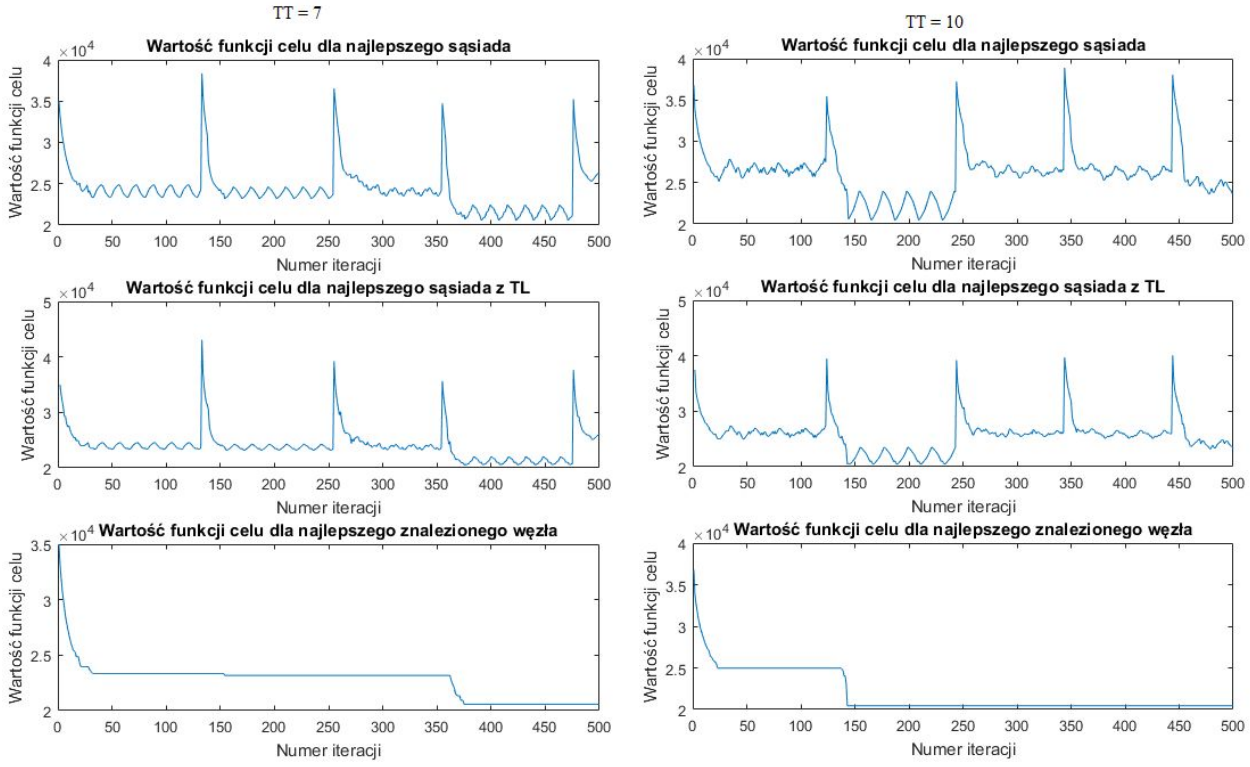
min. osiągnięta wartość fc	**20575,23**	**max. osiągnięta wartość fc**	**24585,27**
średnia wartość fc	**22396,19**	**odchylenie wartości fc**	**749,03**
śr. liczba zadziałań kryterium aspiracji	10,10	σ z liczby zadziałań kryterium aspiracji	4,44
śr. liczba zadziałań zabronienia	391,31	σ z liczby zadziałań zabronienia	14,46

Tab. 14 Wyniki przeprowadzonego testu - TT = 10

min. osiągnięta wartość fc	**20444,23**	**max. osiągnięta wartość fc**	**24410,10**
średnia wartość fc	**22332,07**	**odchylenie wartości fc**	**803,82**
śr. liczba zadziałań kryterium aspiracji	10,39	σ z liczby zadziałań kryterium aspiracji	4,69
śr. liczba zadziałań zabronienia	420,68	σ z liczby zadziałań zabronienia	11,79



Rys. 10. Zmiana wartości funkcji celu.



Rys. 11. Zmiana wartości funkcji celu.

Wpływ wielkości sąsiedztwa

Zbadaliśmy także przebieg algorytmu dla różnej wielkości sąsiedztwa, zarówno parzystego jak i nie. Poniżej przedstawiamy wyniki testów dla **sąsiedztwa** równego **2, 3, 7, 9** i **10:**

Tab. 15 Wyniki przeprowadzonego testu - sąsiedztwo 2

min. osiągnięta wartość fc	**27329,57**	**max. osiągnięta wartość fc**	**33400,00**
średnia wartość fc	**30840,67**	**odchylenie wartości fc**	**1277,19**
śr. liczba zadziałań kryterium aspiracji	7,93	σ z liczby zadziałań kryterium aspiracji	4,14
śr. liczba zadziałań zabronienia	336,06	σ z liczby zadziałań zabronienia	12,23

Tab. 16 Wyniki przeprowadzonego testu - sąsiedztwo 3

min. osiągnięta wartość fc	**24129,97**	**max. osiągnięta wartość fc**	**31262,62**
średnia wartość fc	**27803,62**	**odchylenie wartości fc**	**1464,20**
śr. liczba zadziałań kryterium aspiracji	9,30	σ z liczby zadziałań kryterium aspiracji	4,95
śr. liczba zadziałań zabronienia	374,38	σ z liczby zadziałań zabronienia	17,31

Tab. 17 Wyniki przeprowadzonego testu - sąsiedztwo 7

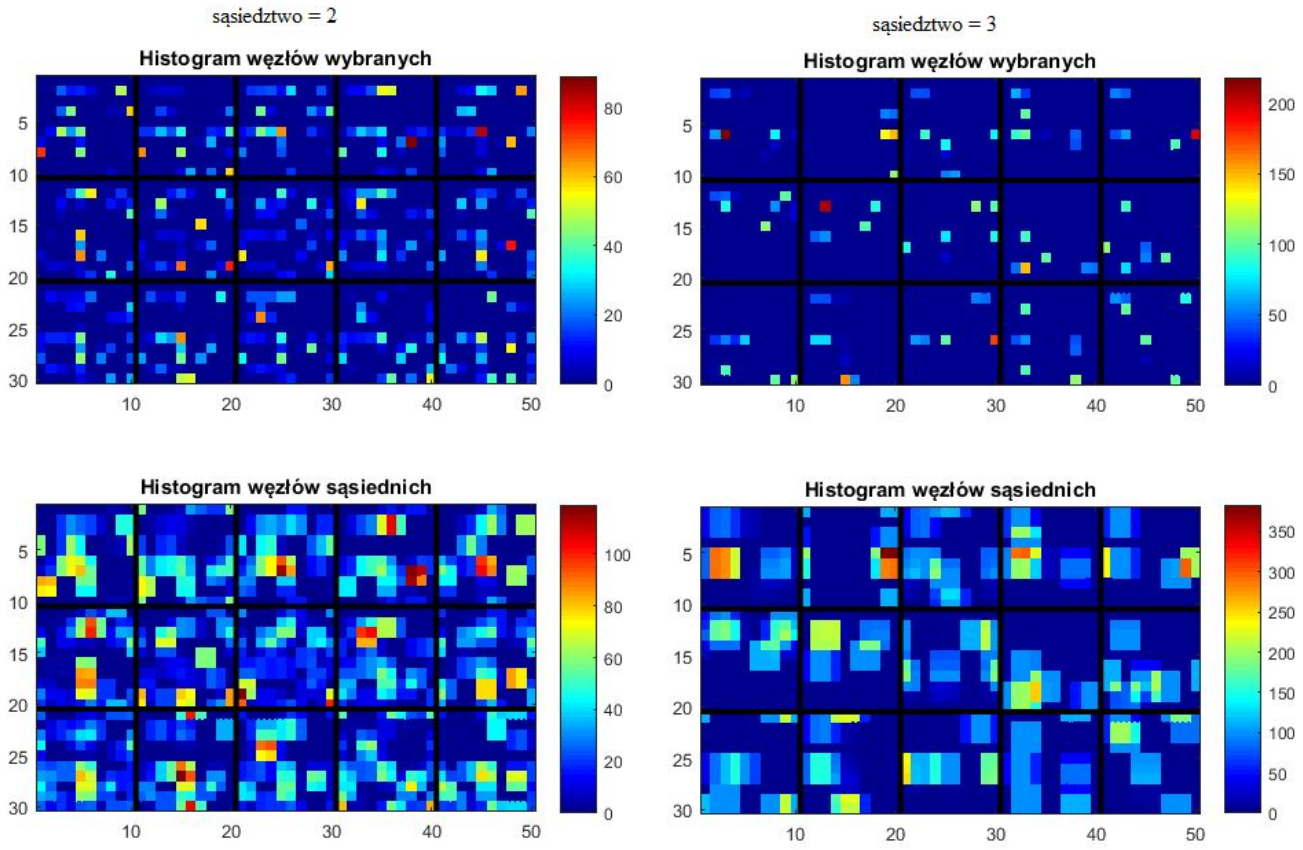
min. osiągnięta wartość fc	**19335,97**	**max. osiągnięta wartość fc**	**22671,13**
średnia wartość fc	**21095,68**	**odchylenie wartości fc**	**700,38**
śr. liczba zadziałań kryterium aspiracji	6,96	σ z liczby zadziałań kryterium aspiracji	3,42
śr. liczba zadziałań zabronienia	348,08	σ z liczby zadziałań zabronienia	18,24

Tab. 18 Wyniki przeprowadzonego testu - sąsiedztwo 9

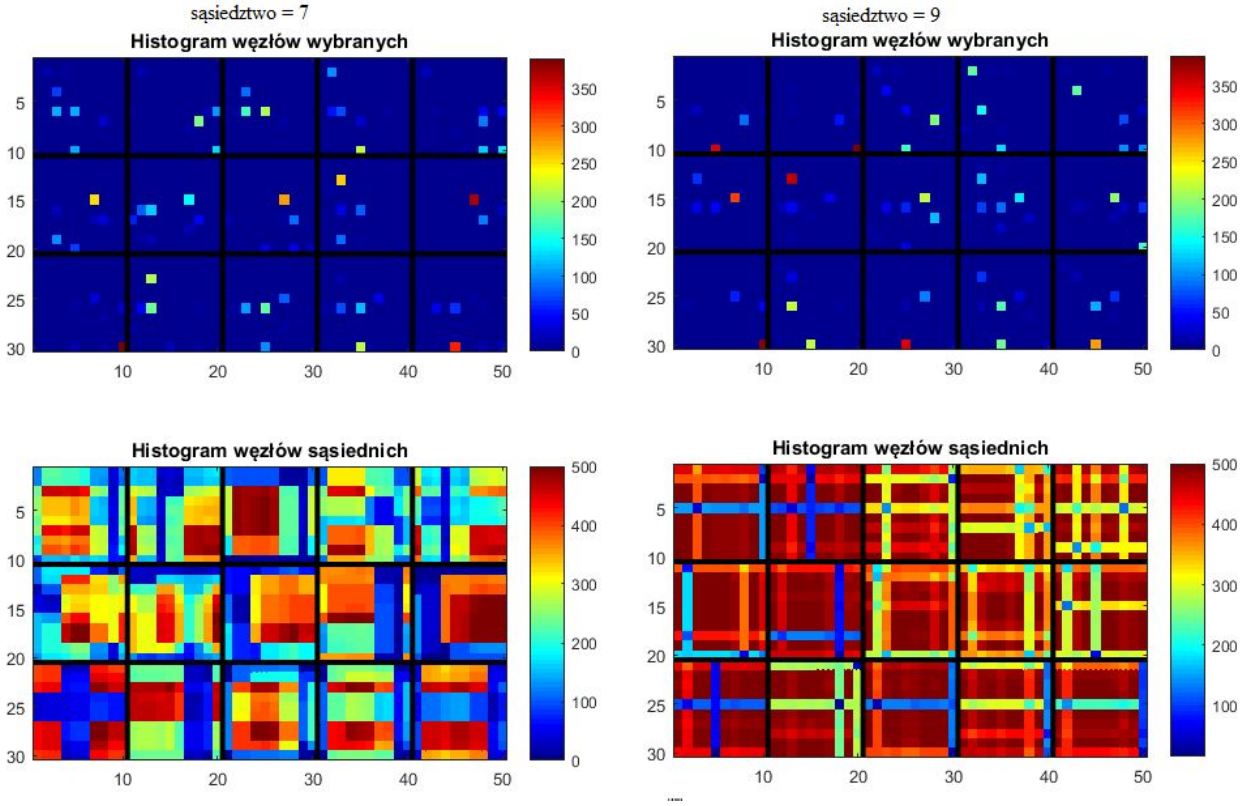
min. osiągnięta wartość fc	**18140,97**	**max. osiągnięta wartość fc**	**21345,90**
średnia wartość fc	**19206,45**	**odchylenie wartości fc**	**794,23**
śr. liczba zadziałań kryterium aspiracji	6,31	σ z liczby zadziałań kryterium aspiracji	3,23
śr. liczba zadziałań zabronienia	329,46	σ z liczby zadziałań zabronienia	18,02

Tab. 19 Wyniki przeprowadzonego testu - sąsiedztwo 10

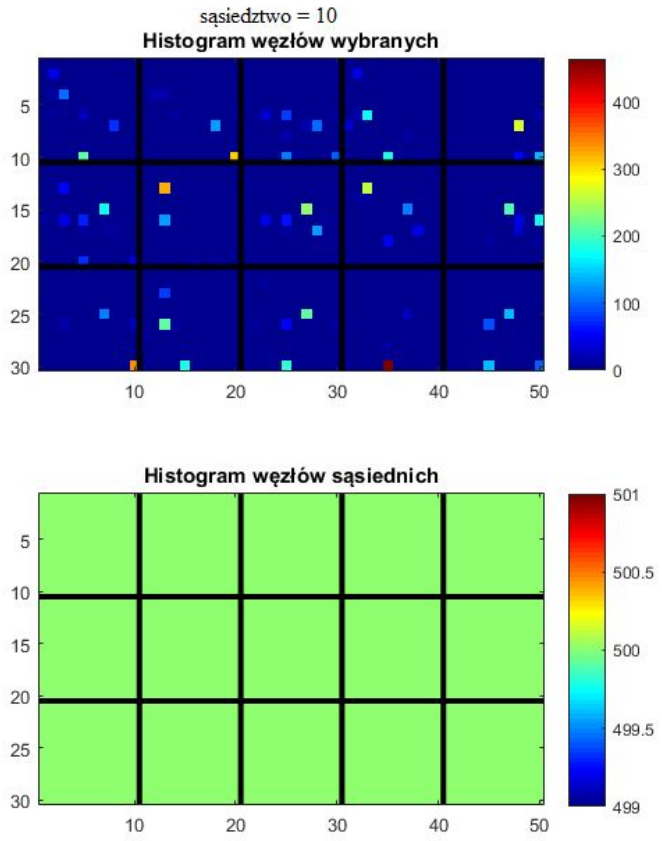
min. osiągnięta wartość fc	**18140,97**	**max. osiągnięta wartość fc**	**21418,23**
średnia wartość fc	**19078,61**	**odchylenie wartości fc**	**707,44**
śr. liczba zadziałań kryterium aspiracji	3,25	σ z liczby zadziałań kryterium aspiracji	2,31
śr. liczba zadziałań zabronienia	332,12	σ z liczby zadziałań zabronienia	16,78



Rys. 12. Histogram węzłów odwiedzonych i przebadanych sąsiadów.



Rys. 13. Histogram węzłów odwiedzonych i przebadanych sąsiadów.



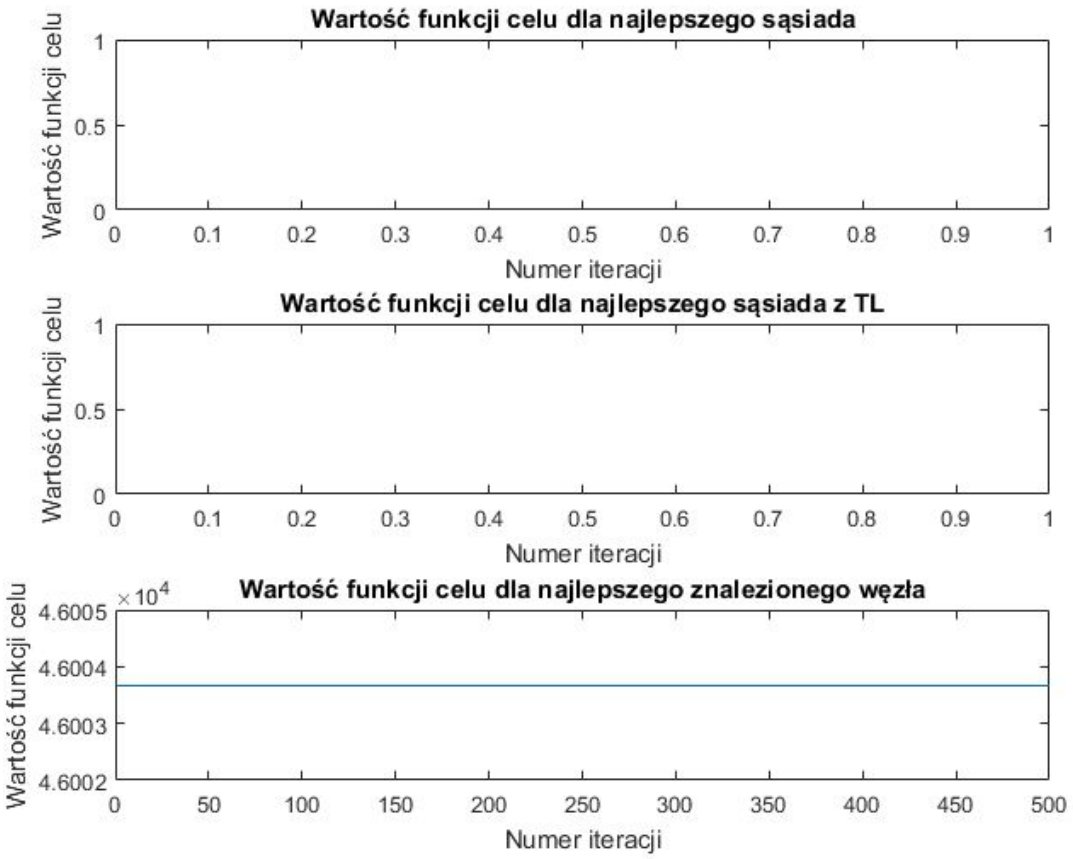
Rys. 14. Histogram węzłów odwiedzonych i przebadanych sąsiadów.

Przypadki złośliwe

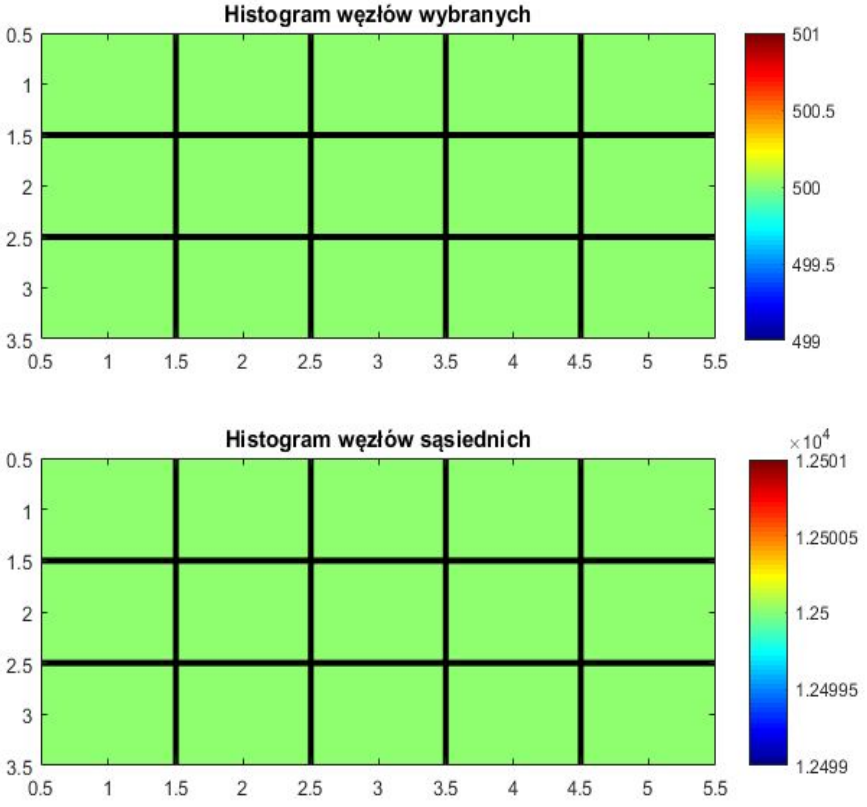
Sprawdziliśmy jak nasz algorytm poradzi sobie ze złośliwymi danymi, stosując w tym celu dane dla **1 restauracji, 1 zestawu** i **5 dni**, przy czym **cena** zestawu była dobrana tak, **by nie było spełnione ograniczenie budżetowe**, dodatkowe **utrudnienie - sąsiedztwo = 5:**

Tab. 20. Wyniki przeprowadzonego testu - 5 dni, 1 restauracja i 1 zestaw bez rozwiązania dopuszczalnego

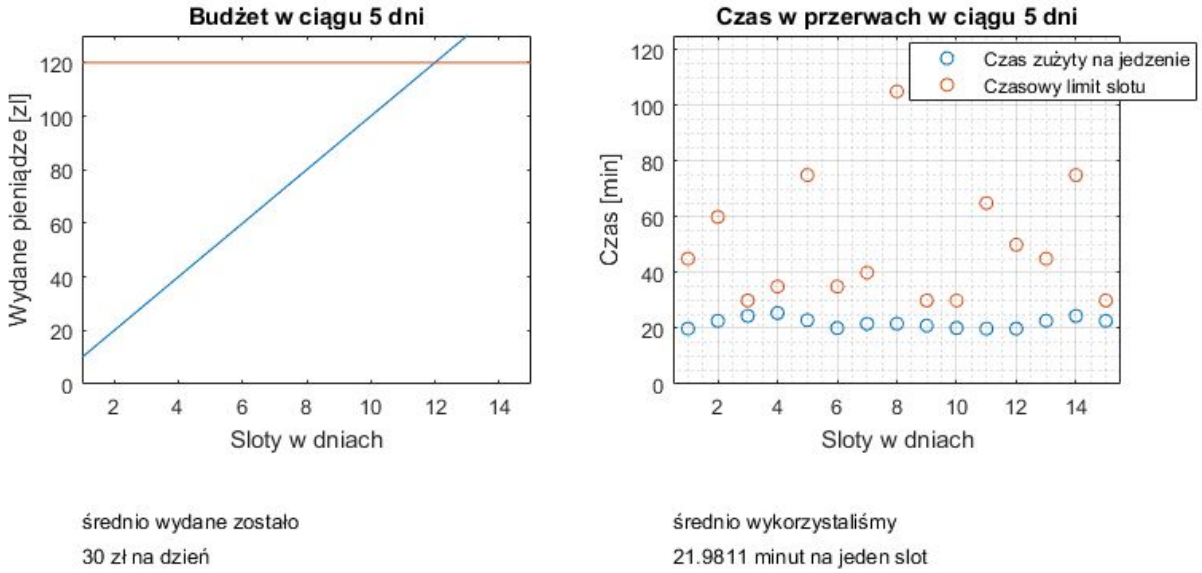
min. osiągnięta wartość fc	**46003,67**	**max. osiągnięta wartość fc**	**46003,67**
średnia wartość fc	**46003,67**	**odchylenie wartości fc**	**0,00**
śr. liczba zadziałań kryterium aspiracji	0,00	σ z liczby zadziałań kryterium aspiracji	0,00
śr. liczba zadziałań zabronienia	0,00	σ z liczby zadziałań zabronienia	0,00
śr. wartości energii w ciągu tygodnia	687,13	σ z wartości energii w ciągu tygodnia	0,00
śr. suma pieniędzy wydanych w slocie	30,00	σ z sumy pieniędzy wydanych w slocie	0,00
śr. czas wykorzystany w slocie	21,98	σ z czasu wykorzystanego w slocie	0,00



Rys. 15. Zmiana wartości funkcji celu.



Rys. 16. Histogram węzłów odwiedzonych i przebadanych sąsiadów.



Rys. 17. Zmiany wydanych pieniędzy w ciągu tygodnia i czasu wykorzystanego w poszczególnych slotach.

Program poinformował także o niedopuszczalności otrzymanego (jedynego istniejącego) rozwiązania:

Warning: Rozwiazanie niedopuszczalne
> In analiza_testow (line 55)

Aplikacja rozpoznaje także podstawowe błędy, mogące pojawić się podczas wprowadzania danych testowych, kończąc swoje wykonanie i zwracając powód zakończenia działania, np.:

Error: W macierzy restauracji jest więcej zestawów niż w macierzy zestawów

Dodatkowo sprawdziliśmy **czas wykonywania programu** dla danych rzeczywistych na podstawie 100 jego wykonań (i 500 iteracji algorytmu). Dodajmy, że czas ten uwzględnia m.in. wczytywanie danych z plików czy zliczanie zadziałań kryterium aspiracji (jednak bez najbardziej obciążających algorytm funkcji graficznych). Otrzymaliśmy średni czas wykonywania programu $t_{śr} = 3,5679\ s$ z odchyleniem standardowym $\sigma = 0,218\ s$. Najkrótszy czas wykonywania programu wynosił $t_{min} = 3,2787\ s$, natomiast najdłuższy: $t_{max} = 4,6856\ s$.

5. Wnioski

Naszym głównym przedmiotem badań było odpowiednie zaimplementowanie algorytmu przeszukiwania rozwiązań Tabu Search, dlatego mniejszą uwagę skupiliśmy na precyzyjnie rzeczywistym odzwierciedleniu “Problemu głodnego studenta”. W tym celu dokonaliśmy wymienionych w opisie problemu uproszczeń modelu, a w trakcie testów

skupiliśmy się na badaniu poprawnego działania samego algorytmu Tabu Search, zmieniając jego poszczególne kluczowe parametry.

Już w trakcie pisania kodu i pierwszych testów musieliśmy zmierzyć się z problemem wynikającym z pojawianiem się **rozwiązań niedopuszczalnych**. Ponieważ nasz algorytm rozpoczynał od tego, że losowaliśmy początkowe rozwiązanie w skończonej liczbie iteracji, to mogło się zdażyć tak, że nie udawało się wylosować początkowego rozwiązania dopuszczalnego. W takim wypadku wyświetlaliśmy warning, jednak nie przerywaliśmy działania programu. Doszliśmy bowiem do wniosku, że algorytm Tabu Search może przetwarzać rozwiązania niedopuszczalne, aby następnie przeszukując sąsiadów znalazł rozwiązanie dopuszczalne. Aby obronić się przed przypadkiem, że wylosowane rozwiązanie niedopuszczalne jest najlepsze i ostatecznie zostaje wybrane jako rozwiązanie suboptymalne na końcu działania algorytmu sprawdzamy czy jest ono dopuszczalne - niemal wszystkie (poza złośliwymi) sprawdzone przez nas przypadki skutkowały otrzymaniem jako suboptymalnego rozwiązania dopuszczalnego.

Wykorzystanie w głównej instancji problemu danych rzeczywistych (opracowanych na podstawie własnych obserwacji i mapy okolicy AGH) pozwala na intuicyjne sprawdzenie poprawności działania algorytmu - otrzymujemy zdroworozsądkowe wyniki, np. mając kolejne zajęcia w tym samym budynku zwykle jemy w pobliskiej restauracji. Dane dla pozostałych przypadków zostały opracowane na podstawiet danych rzeczywistych, jednak odbiegają już od realnego problemu.

Nasze testy rozpoczęliśmy od najprostszego sprawdzenia poprawności wykonywania algorytmu dla danych wejściowych rzeczywistych. Zdecydowaliśmy się wykonać taki test, aby w dalszej badań móc się odnosić do tych "podstawowych" wyników i sprawdzać wpływy zmian poszczególnych parametrów. Początkowe ustawienia dobraliśmy metodą prób i błędów, tak aby skutecznie rozwiązać nasz podstawowy "Problem głodnego studenta". Wyniki przedstawione w *tabeli 3*, pokazują że napisany przez nas program "korzysta" w pełni ze zmodyfikowanej przez nas wersji algorytmu Tabu Search, to jest zachodzi losowość, od której zależy znalezienie rozwiązania subopytmalnego, dochodzi do zabronień, a także działa kryterium aspiracji.

Kolejne testy polegały głównie na sprawdzeniu uniwersalności naszego algorytmu pod względem zmiany rozmiaru danych wejściowych, tj. zmiana liczby restauracji, zmiana liczby zestawów, zmiana liczby dni a także zmiana dostępności zestawów.

W pierwszej kolejności **zmniejszenie liczby restauracji i zestawów** jest jak najbardziej akceptowalne dla algorytmu i odpowiednio zmieniają się wartości funkcji celu. Warto zwrócić uwagę, że **odchylenie standardowe wartości funkcji celu zmniejszyło się** względem zagadnienia bazowego (porównaj *tabele 3* i *4*) co jest skutkiem zmniejszenia liczby dopuszczalnych rozwiązań.

Następnie chcieliśmy sprawdzić jak poradzi sobie nasz algorytm, gdy **zwiększymy liczbę dni**. W tym przypadku po raz pierwszy spotkaliśmy się ze "złośliwością" danych wejściowych, ponieważ nasz algorytm przez skończoną liczbę iteracji nie mógł znaleźć

rozwiązania dopuszczalnego. Nasz program dokończył obliczenia, jednak ich wyniki mogą być mało satysfakcjonujące (patrz *rys. 6*). Dla przedstawionego przypadku zostało szczęśliwie wylosowane bardzo dobre rozwiązanie i dalsze przeszukiwanie jego sąsiedztwa nie przyniosło już poprawy najlepszej znalezionej wartości funkcji celu (także w pozostałych 99 wywołaniach algorytmu).

Aby skuteczniej sprawdzić poprawność działania algorytmu dla **większej liczby dni** zmieniliśmy dane wejściowe i zwiększyliśmy liczbę dostępnych zestawów we wszystkich restauracjach (porównaj przebiegi funkcji celu na *rysunkach 6* i *7*). **Zwiększyliśmy** również **dostępność zestawów** dla przypadku 4 restauracje, 7 zestawów. Można zauważyć, że dzięki takim danym wejściowym algorytm ma dużo więcej możliwości rozpoczęcia poszukiwań rozwiązania i otrzymywane **wyniki silnie zależą od węzłów początkowych** (porównaj odchylenie standardowe wartości funkcji celu w *tabelach 4* i *7*).

Następna seria testów dotyczyła zbadania wpływu limitu liczby iteracji w algorytmie. Z tej części można wywnioskować, że **zbyt mała liczba iteracji nie pozwala na skuteczne przeszukiwanie przestrzeni rozwiązań**, w tym sąsiedztwa (*tabela 8*, *rysunek 8*). Kolejne **zwiększanie limitu liczby iteracji** algorytmu powoduje osiąganie **coraz mniej zróżnicowanych wyników końcowych** (porównaj odchylenia standardowe wartości funkcji celu w *tabelach 3*, *8*, *9* i *10*). Natomiast inna sprawa dotyczy samej **średniej wartości funkcji celu**: widać, że **początkowe zwiększenie** limitu iteracji **poprawiało** tą wartość, jednak **dla limitu 1000** nastąpiło **już pogorszenie** (porównaj średnie wartości funkcji celu w *tabelach 3*, *8*, *9* i *10*). Można zauważyć, że od pewnego momentu wyniki końcowe zależą już bardziej od losowania, które pojawia się kilkukrotnie w algorytmie. Wnioskujemy więc, że warto możliwie zwiększyć limit iteracji, jednak nie można z tym przesadzić, ponieważ od pewnego limitu nie przyczynia się to do poprawy rozwiązań, a co gorsze znacznie zwiększa czas działania programu.

Kolejne badania dotyczyły wpływu wielkości ***Tabu Tenure*** na pracę algorytmu. Z wyników tych badań można jasno zauważyć, że **zwiększenie** wielkości **TT** wpływa na **średnią liczbę zabronień i zadziałań kryterium aspiracji** (*tabele 11*, *12*, *13* *i* *14*). Ze względu na **specyfikę rzeczywistych danych wejściowych** po samych wynikach funkcji celu nie można wysnuć jednoznacznych wniosków. Dzieje się tak, ponieważ w przypadku gdy **TT = 0** i **zabronienia** zwyczajnie **nie działają** na dobrą jakość funkcji celu większy **wpływ** ma częste **losowanie** punktów startowych. Gdy pojawiają się **zabronienia** to **wyniki poprawiają** się tylko **nieznacznie** za sprawą występowania w danej instancji problemu zbyt **małej liczby rozwiązań dopuszczalnych**. Nie mniej jednak jednak na dowód działania listy tabu i długości TT można zobaczyć na *rys. 10* i *11*. Widać tam porównanie przebiegu wartości funkcji celu poszczególnych węzłów. **Zwiększenie liczby TT** powoduje powstawanie charakterystycznych **"zębów" na charakterystyce**, co może pozwala na **"głębsze" przeszukiwanie sąsiedztwa**. Podczas przeprowadzonych testów sprawdziliśmy także, że **zbytnie zwiększenie parametru TT** wpływa na **długość obliczeń** algorytmu,

a zbyt duża wartość może powodować problemy w znalezieniu kolejnych dopuszczalnych węzłów, nie będących na liście tabu.

Zbadaliśmy także wpływ **rozmiaru sąsiedztwa** na wyniki przeszukiwań. Ogólnie przyjęliśmy, że nasze sąsiedztwo zawsze jest kwadratem, zmieniamy tylko długość jego "boku". Wykonując te testy od razu zauważyliśmy, że rozmiar sąsiedztwa bardzo znacząco wpływa na długość wykonywania się algorytmu. Na histogramie *rys. 12* wyraźnie widać, że **zbyt mały rozmiar "boku" kwadratu** powoduje **słabe przeszukiwanie sąsiadów**, co przekłada się na bardzo **niekorzystne wyniki** średniej wartości funkcji celu (patrz *tabela 15*). Sukcesywne **zwiększanie sąsiedztwa** naturalnie powoduje **poprawę wyników** (**przeszukujemy dokładniej**). Można jednak zauważyć, że **od pewnego momentu kolejne powiększenie nie wpływają spektakularnie na polepszenie wyników** (przykładowo porównaj *tabelę 18* i *19*), **natomiast bardzo mocno zwiększają złożoność obliczeń** (patrz na histogram z *rys. 14* - dla każdego węzła przeglądamy wszystkich sąsiadów, co jest kosztowne obliczeniowo).

Aby upewnić się co do poprawności i uniwersalności działania naszego algorytmu postanowiliśmy także przetestować **przypadki** ogólnie uważane za **"złośliwe"**. W tym celu zmniejszyliśmy radykalnie liczbę restauracji i zestawów do 1, przy czym resztę parametrów nie zmienialiśmy. Przeprowadzone przez nas testy (patrz *tabela 20*), pokazały, że algorytm poprawnie reaguje na takie dane wejściowe (poprawna indeksacja), działają także wszystkie wizualizacje przebiegu algorytmu (patrz *rysunek 15* i *16*). Pokazaliśmy również co się dzieje, gdy cenę zestawu dobierzemy tak, że musi zostać przekroczony budżet (patrz *rysunek 17*), co potwierdza poprawne działanie algorytmu w przypadku, gdy nałożone ograniczenia tworzą **zbiór pusty możliwych rozwiązań dopuszczalnych**.

Przygotowany algorytm i jego implementacja mogą być także zastosowane do rozwiązywania instancji problemów o większych wymiarach (np. żywienie roczne mieszkańca jakiegoś miasta), a także problemów o zbliżonym modelu matematycznym. Oczywiście wersja aplikacji do rozwiązywania znacząco większych problemów powinna zostać dodatkowo zoptymalizowana pod kątem złożoności obliczeniowej i pamięciowej (przede wszystkim usunięte muszą zostać procedury graficzne i dodatkowe fragmenty kodu badające wnikliwie przebieg algorytmu). Należy także pamiętać, że dla każdej instancji problemu powinniśmy indywidualnie dobrać wartości poszczególnych parametrów algorytmu.