

Automatyka i Robotyka

Badania Operacyjne II

Optymalizacja harmonogramu zajęć

Za pomocą algorytmu Symulowanego Wyżarzania

Autorzy:

Marek Lechowicz

Kacper Motyka

1 . Model zagadnienia.

1) Krótki opis

Program ma za zadanie układanie optymalnego planu zajęć. Korzystając z listy kwalifikacji instruktorów, dostępności sal a także listy zajęć, które zostały wybrane przez poszczególnych klientów, algorytm ma znajdować najbardziej optymalny rozkład zajęć tak, aby nie nastąpiła żadna kolizja, ale także tak, żeby usatysfakcjonować największą liczbę klientów i generować największe zyski dla instytucji prowadzącej zajęcia. Naszą motywacją jest ułożenie harmonogramu zajęć grupowych na siłowni. Problem został zdefiniowany tak, aby model łatwo można było przenosić na układanie zajęć dla grup tanecznych, harmonogramów dla pracowników pracujących na różne zmiany i wielu innych zagadnień.

Optymalizacji podlega tygodniowa liczba zajęć, która musi zostać zorganizowana, żeby zarobki siłowni były jak największe. Model matematyczny został zdefiniowany tak, aby była to optymalizacja jednokryterialna (zastosowana została jedna funkcja celu).

2) Ograniczenia i uproszczenia

Istotne uwarunkowania i zależności:

• Nie może dojść do sytuacji, gdy dwa różne zajęcia odbywają się w tej samej sali w tym samym czasie.

• Nie można doprowadzić do sytuacji, gdzie nauczyciel prowadzi w jednym momencie różne zajęcia.

• Pomiędzy zajęciami muszą być przerwy dla uczniów i nauczycieli.

Zastosowane uproszczenia zachowujące istotę problemu:

• Sale traktuje jako równoważne, nie przewiduję sal „do Zumby”, “do crossfitu” itp.

• Warto pominąć także fakt minimalnej liczby godzin przydzielonych dla danych instruktorów. Zakładam, że mogą oni organizować zajęcia także w innych siłowniach a w naszej tylko dorabiać.

• Pomijam także fakt występowania świąt i innych przerw w ciągu roku i przepadania zajęć z tego powodu.

2) Model matematyczny

Struktury danych

• D – day – dni tygodnia D = {1, 2, …, 6}

• S – slot – okna czasowe na przeprowadzenie zajęć S = {1, 2, …, 6} gdzie 1 to czas od 16:00 do 17:00, 2 to czas od 17:00 do 18:00 itd.

• C – classroom – sala, w której prowadzone będą zajęcia C = {1, 2, … 4}

• T – type – rodzaj prowadzonych zajęć T = {1, 2, …, 𝑡𝑚𝑎𝑥} gdzie na przykład 1 – Zumba, 2 – Zumba Advanced, 3 – Crossfit itd.

• I – instructor – osoba prowadząca zajęcia I = {1, 2, …, 𝑖𝑚𝑎𝑥}

• N – numer – ID osoby, która za pomocą ankiety wyraziła chęć udziału w zajęciach N = {1, 2, …, 𝑛𝑚𝑎𝑥}

Postać rozwiązania

Rozwiązanie ma postać macierzy o wymiarach **d x s x c** (dni tygodnia x ilość slotów w każdym dniu x liczba klas). Elementami macierzy są wektory [i, t, l] przechowujące informacje o:

• i – instruktor prowadzący zajęcia, • t – typ zajęć, • l– lista uczestników.

Zmienne decyzyjne

• 𝑎𝑑, 𝑠, 𝑐, 𝑡, 𝑖 ∈ {0, 1} − 1 jeśli typ zajęć t jest nauczany w klasie c w oknie czasowym s w dniu d przez instruktora i, w przeciwnym wypadku 0

• 𝑥𝑑, 𝑠, 𝑐, 𝑡, 𝑖, 𝑛 ∈ {0, 1} - 1 jeśli użytkownik n bierze udział w zajęciach typu t prowadzonych przez instruktora i w klasie c w dniu d, w przedziale czasowym s, w przeciwnym wypadku 0

• 𝑦𝑖, 𝑑 ∈ {0, 1} - 1 jeśli prowadzący i musi dnia d przyjść do pracy, w przeciwnym wypadku 0

• 𝑧𝑛, 𝑑 ∈ {0, 1} − 1 gdy użytkownik n musi dnia d przyjść na trening, w przeciwnym wypadku 0

• q𝑑 ∈ {0, 1} − 1 gdy sala jest zajęta w dniu d, w przeciwnym wypadku 0

Zmienne pomocnicze

• 𝑆𝑒𝑙𝑒𝑐𝑡𝑒𝑑𝑛, 𝑡 = 1 jeśli użytkownik siłowni n wybrał dane zajęcia t, 0 w przeciwnym wypadku • 𝑄𝑢𝑎𝑙𝑖𝑓𝑖𝑒𝑑𝑖, 𝑡 = 1 jeśli instruktor i ma kwalifikacje, aby uczyć dany typ zajęć t

Istotne warunki i ograniczenia

1) Σ𝑡 Σ 𝑖 𝑎𝑑, 𝑠, 𝑐, 𝑡, 𝑖 ≤ 1, ∀d ∈ 𝐷, ∀s ∈ 𝑆, ∀c ∈ 𝐶

Dla każdego konkretnego dnia, przedziału czasowego oraz sali lekcyjnej mogą być prowadzone tylko jedne zajęcia (zapobieganie odbywania się w tym samym momencie dwóch rodzajów zajęć w jednej sali) a także tylko jeden prowadzący może być dopasowany do danych zajęć.

2) Σ 𝑡 Σ 𝑐 𝑎𝑑, 𝑠, 𝑐, 𝑡, 𝑖 ≤ 1, ∀d ∈ 𝐷, ∀s ∈ 𝑆, ∀i ∈ 𝐼

Każdy prowadzący może być przepisany w jednym przedziale czasowym do tylko jednych zajęć.

3) Σ 𝑑 Σ 𝑠 Σ 𝑐 Σ 𝑡 Σ 𝑖 𝑥𝑑, 𝑠, 𝑐, 𝑡, 𝑖, 𝑛 = Σ 𝑡 𝑆𝑒𝑙𝑒𝑐𝑡𝑒𝑑𝑛, 𝑡 , ∀n ∈ 𝑁

4) Σ 𝑑 Σ 𝑠 Σ 𝑐 Σ 𝑖 𝑥𝑑, 𝑠, 𝑐, 𝑡, 𝑖, 𝑛 = Σ 𝑡 𝑆𝑒𝑙𝑒𝑐𝑡𝑒𝑑𝑛, 𝑡 , ∀n ∈ 𝑁, ∀t ∈ 𝑇

**Sens powyższych wzorów:** każdy użytkownik siłowni musi mieć przydzielone tyle zajęć, ile wybrał oraz muszą to być dokładnie te typy zajęć, na które się zapisał

Dodatkowe warunki i ograniczenia

Aby model działał zadowalająco dla osób uczęszczających na zajęcia, instruktorów prowadzących zajęcia, a także dla właścicieli siłowni, warto zdefiniować dodatkowe warunki, które zwiększą komfort korzystania z zajęć, ale także zminimalizują koszt organizacji zajęć.

5) Σ 𝑛 𝑥𝑑, 𝑠, 𝑐, 𝑡, 𝑖, 𝑛 ≤ 10, ∀d ∈ 𝐷, ∀s ∈ 𝑆, ∀c ∈ 𝐶, ∀t ∈ 𝑇, ∀i ∈ 𝐼

Dla zwiększenia komfortu ćwiczeń oraz ze względu na wymogi sanitarne wynikające z pandemii COVID-19 wprowadzono limit osób biorących udział w zajęciach równy 10.

6) Σ 𝑑 𝑦𝑖, 𝑑 ≤ 3, ∀i ∈ 𝐼

Aby zmniejszyć ponoszony przez firmę koszt oraz zapewnić komfort pracy instruktorów wprowadzono ograniczenie, które powoduje, że pracownik nie musi przychodzić do pracy więcej niż 3 razy tygodniowo (ilość dni można modyfikować zależnie od zapotrzebowania).

**Funkcja celu**

fcel = ticket\_cost \* Σ 𝑛, d, t, c 𝑥𝑑, 𝑠, 𝑐, 𝑡, 𝑖, 𝑛 – hour\_pay \* Σ i, d, t, c 𝑥𝑑, 𝑠, 𝑐, 𝑡, 𝑖, 𝑛 – pay\_for\_presence \* Σ 𝑑, i 𝑦𝑖, 𝑑 – class\_renting\_cost \* Σ 𝑑 q𝑑

Gdzie:

ticket\_cost – cena biletu na zajęcia,

Σ 𝑛, d, t, c 𝑥𝑑, 𝑠, 𝑐, 𝑡, 𝑖, 𝑛 - łączna liczba uczestników ,

hour\_pay – zarobki instruktora (za godzinę),

Σ i, d, t, c 𝑥𝑑, 𝑠, 𝑐, 𝑡, 𝑖, 𝑛 - łączna liczba zajęć,

pay\_for\_presence – zarobki instruktora za pojawienie się danego dnia w pracy,

Σ 𝑑, i 𝑦𝑖, 𝑑 - łączna liczba pojawień wszystkich instruktorów,

class\_renting\_cost – cena wynajęcia sali (za dzień),

Σ 𝑑 q𝑑 - łączna liczba dni, w których wynajmujemy salę.

2 . Algorytm

1) Pseudokod algorytmu

R := R\_startowe (R – rozwiązanie aktualne)

R\_best := R (R\_best – rozwiązanie najlepsze)

T := T\_początkowe

while ((not STOP\_1) and (not STOP\_2))

for j = 0 do j = max\_iter\_one\_temp

R1 := get\_neighbour (wybór jednego z wylosowanych rozwiązań)

∆ := F(R1) – F(R)

if ∆ ≥ 0

R := R1

if F(R) > F(R\_best)

R\_best := R

end if

else

s = rand (0, 1)

if s < exp(-∆/T)

R := R1

end if

end if

T := α \* T (liniowa zmiana temperatury)

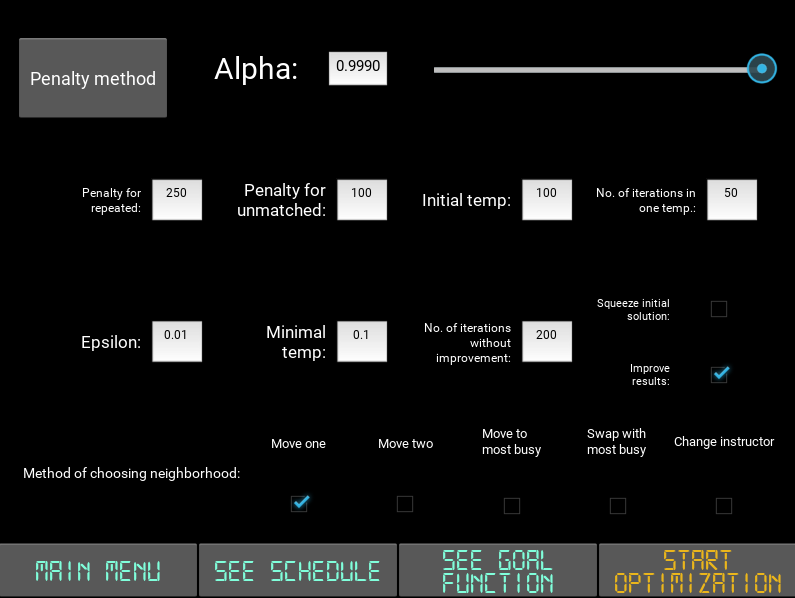
end while;

2) Przyjęte kryteria stopu

STOP\_1 – osiągnięcie temperatury minimalnej;

STOP\_2 – w zadanej liczbie iteracji wartości wskaźnika jakości (sumarycznego kosztu) dla otrzymywanych rozwiązań różnią się od siebie o nie więcej niż podana wartość błędu bezwzględnego;

3) Parametry Algorytmu



**Rysunek 1.** Ekran aplikacji służący do ustawienia parametrów

* penalty\_for\_repeated – parametr używany jedynie przy wciśnięciu przycisku “Penaty Method) - kara za pozwolenie, aby instruktor odbywał dwa zajęcia w tym samym momencie
* penalty\_for\_unmatched – parametr używany jedynie przy wciśnięciu przycisku “Penaty Method) - kara za pozwolenie, aby instruktor prowadził zajęcia, do prowadzenia których nie ma kompetencji
* alpha – liniowy współczynnik zmiany temperatury z przedziału (0, 1); domyślnie wynosi 0.9999
* inital\_temp – temperatura startowa zmniejszana w kolejnych iteracjach algorytmu; domyślnie wynosi 1000
* n\_iter\_one\_temp – liczba iteracji wykonywana dla jednej wartości temperatury; domyślnie wynosi 50
* min\_temp – kryterium stopu 1, temperatura, do której dążymy, liczba rzeczywista nieujemna; domyślnie wynosi 0.1
* epsilon – kryterium stopu 2, bezwzględna wartość różnicy między kosztami dla rozwiązań otrzymywanych w kolejnych iteracjach, która powinna spowodować zatrzymanie działania algorytmu; domyślnie wynosi 0.01
* n\_iter\_without\_improvement – liczba iteracji bez zmiany kosztu o co najmniej o epsilon, po której powinno nastąpić zatrzymanie algorytmu; domyślnie wynosi 500
* initial\_solution – informuje algorytm o tym, czy mamy już jakieś rozwiązanie początkowe (True), czy trzeba jakieś wygenerować (False); domyślnie wynosi True
* neighborhood\_type\_lst – lista rodzajów sąsiedztw (opisanych poniżej), z których powinien korzystać algorytm; domyślnie wynosi [‘move\_one’]
* Squeeze inital solution– informuje algorytm o tym, czy rozwiązanie początkowe powinno być generowane przez losowe umieszczanie zajęć w wolnych przedziałach czasowych (False), czy jedno po drugim od pierwszego wolnego przedziału czasowego (True); domyślnie wynosi False
* Improve solution – decyduje o tym, czy chcemy skorzystać z dodatkowego algorytmu poprawiającego wyniki
* Penalty method – parametr, który definiuje, czy chcemy korzystać z metody funkcji kary, czy ze sztywnych ograniczeń; domyślnie wynosi False – sztywne ograniczenia

4) Rodzaje sąsiedztw:

* ‘move\_one’ - wybiera losowo jedne zajęcia i przenosi je w inny losowo wybrany wolny przedział czasowy,
* ‘move\_two’ - wybiera losowo dwa zajęcia i przenosi je w inny losowo wybrany wolny przedział czasowy
* ‘move\_to\_most\_busy’ - wybiera jedne zajęcia z dnia, który jest najmniej zajęty i przenosi je na inny dzień, który jest najbardziej zajęty, jednak ma jeszcze jakiś wolny przedział czasowy
* ‘swap\_with\_most\_busy’ - wybiera jedne zajęcia z dnia, który jest najmniej zajęty i zamienia je z zajęciami, które odbywają się w najbardziej zajęty dzień (nie musi on mieć żadnego wolnego przedziału czasowego, ponieważ następuje zamiana)
* ‘change\_instructor’ - losowo wybiera zajęcia i zamienia dla nich prowadzącego z listy instruktorów, którzy są w stanie prowadzić dany typ zajęć

5) Dodatkowy algorytm poprawiający

W celu poprawienia ogólnego działania optymalizacji zastosowaliśmy dodatkowy algorytm poprawiający. Z poziomu GUI istnieje możliwość zastosowania tego algorytmu, lub też nie, zależnie od preferencji użytkownika.

Działanie algorytmu jest następujące: algorytm sprawdza, czy istnieją dni, w które dany instruktor przychodzi mniej razy niż w inne dni. Jeśli tak, to następnie sprawdzana jest możliwość, przeniesienia tej mniejszej liczby dni. Jeśli jest to możliwe, to algorytm przenosi je, co poprawia końcową funkcję celu, ponieważ zależy ona od ilości dni, w które prowadzący musi się pojawić w pracy. Następnie algorytm przechodzi do następnego instruktora i powtarza powyżej opisane czynności.

Funkcja, która wywołuje działanie powyżej opisanego algorytmu nie przyjmuje żadnych parametrów.

3. Aplikacja

Wymagania do uruchomienia

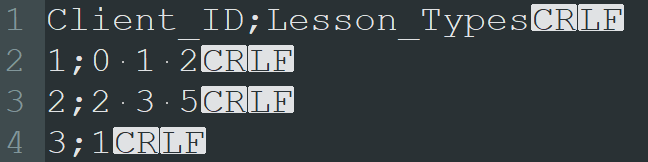
Do uruchomienia aplikacji z pliku .exe nie ma szczególnych wymagań. Przy opracowywaniu aplikacji korzystaliśmy z Pythona w wersji 3.8 wraz z zainstalowanymi bibliotekami:

* Kivy – do opracowania graficznej części aplikacji
* Pandas – do łatwiejszego importu plików .csv
* Numpy – dla ułatwienia i usprawnienia działań na macierzach
* Matplotlib – do wizualizacji wyników algorytmu

Format danych

Dane wejściowe do programu podaje się w formie plików .csv. Plik zawierający dane odnośnie klientów powinien być sformatowany w następujący sposób: w skrajnej lewej kolumnie oznaczonej Client\_ID należy wpisać id klienta, a w kolumnie Lesson\_Types oddzielone spacjami id typów lekcji wybranych przez klienta.

Przykład:



**Rysunek 2.** Przykład poprawnego napisania pliku .csv z danych klientów

Tak napisany plik wskazuje, że:

* Klient 1 wybrał typy zajęć o numerach: 0, 1, 2
* Klient 2 wybrał typy zajęć o numerach: 2, 3, 5
* Klient 3 wybrał typ zajęść o numerze: 1

Plik z danymi na temat instruktorów powinien zostać tak skonstruowany, aby w kolumnie Instructor\_ID znajdowały się id instruktorów, natomiast w kolumnie Lesson\_Types znajdowały się id typów lekcji, do prowadzenia których uprawniony jest dany instruktor.

Przykład:



**Rysunek 3.** Przykład poprawnego napisania pliku \*.csv z danych instruktorów

Tak napisany plik wskazuje, że:

* Instruktorzy o id od 0 do 4 mogą prowadzić wszystkie 10 typów zajęć
* Instruktor 5 może prowadzić typy zajęć 2, 5, 6, 8

Format wyników

Wyniki można podejrzeć w aplikacji. Menu “See Schedule” pozwala podejrzeć terminarz wybrany przez algorytm. Wyświetlany jest terminarz tylko dla jednej klasy (domyślnie o id: 0). Wyboru wyświtlanej klasy można dokonać wciskając przycisk “Displayed Classroom” w lewym górnym rogu programu. Menu “Goal Function” umożliwia prześledzenie przebiegu funkcji celu. Dodatkowo parametry analizowanego problemu jak również koszty wyliczone w kolejnych etapach pracy programu można sprawdzić w menu “See Algorithm Parameters”.

Opis funkcjonalności:

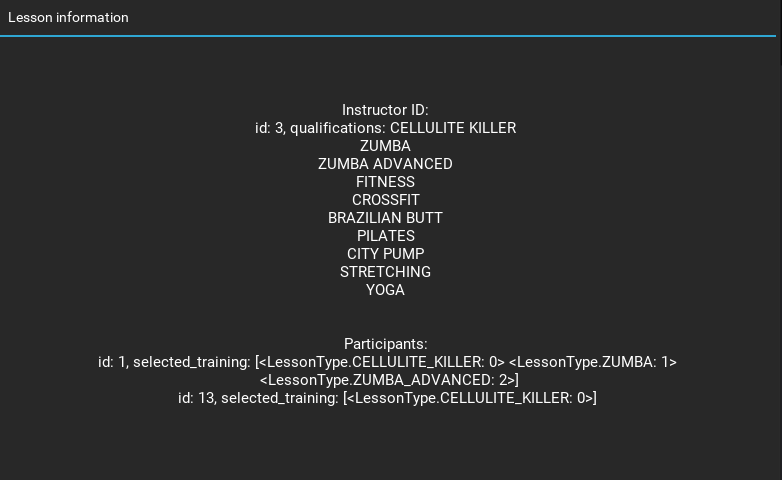


**Rysunek 4.** Główny ekran aplikacji

Główny ekran aplikacji zawiera 7 przycisków spełniających następujące funkcje:

* **Optimize** -Menu pozwalające ustawić parametry algorytmu (przedstawione na Rysunku 1.)
* **Schedule Options** –Menu pozwalające wybrać ścieżkę do plików .csv zawierających informacje o klientach i o instruktorach (jeżeli zostały umieszczone w folderach client\_data i instructor\_data program powinien wykryć je automatycznie, w innym przypadku należy wybrać właściwe ścieżki). Dodatkowo w tym menu można zmienić parametry terminarza: liczbę klas, liczbę dni pracy, ilość godzin treningów w dniu, liczbę klientów danego dnia, cenę biletów, płace godzinową dla instruktora, płace dla instruktora za obecność, koszt dzienny wynajęcia sali.
* **See Schedule** –Menu wyświetlające terminarz dla kolejnych klas. (wyświetlaną klasę można zmienić przyciskiem “Displayed Classrom” w lewym górnym rogu)
* **Goal Function –** Menu wyświetlające wykres funkcji celu w kolejnych iteracjach
* **See Algorithm Parameters –** Menu wyświetlające: koszt początkowy, koszt optymalizacji, koszt po uruchomieniu algorytmu poprawiającego, liczbę iteracji, liczbę klientów, liczbę instruktorów, czas optymalizacji.
* **About Organizer –** informacje na temat aplikacji, link do repozytorium projektu na GitHub
* **Exit -** wyjście z programu

Dodatkowo, warto zaprezentować w jaki sposób przechowywana w programie jest informacja o organizowanych zajęciach:



**Rysunek 5.** Przykładowy ekran informujący o prowadzącym i uczestnikach zajęć

3. Testy

Wykaz scenariuszy

1) Podział ze względu na dane wejściowe

1. Test dla niedużej liczby uczestników - przypadek najprostszy
2. Test dla powiększonej liczby uczestników

Dla powyższych przypadków zostały przeprowadzone osobne testy ze względu na **dobrane parametry algorytmu**. Testowane były między innymi:

1. Pojedyncze otoczenia,
2. Kombinacje otoczeń podzielone ze względu na ich naturę: losowe, oparte na wartościach maksymalnych itp.
3. Powtarzalność działania algorytmu, na którą wpływ ma parametr **n\_iter\_without\_improvement**.
4. Zbieżność algorytmu.
5. Zastosowanie funkcji kary lub stałych ograniczeń zapobiegających powstawaniu przypadków niedozwolonych.
6. Inne.

Testy zostały przeprowadzone także dla różnych **parametrów harmonogramu:**

1. cena biletu na zajęcia
2. stawka za godzinę
3. liczba zajęć w ciągu dnia itd.

Przypadki złośliwe:

1. Zbyt duża liczba grup dla wybranej liczby klas
2. Zbyt mała liczba instruktorów

Opis metodyki badań

Badania były prowadzone w następujący sposób. Najpierw, odseparowaliśmy od siebie przepadki testowe, które dotyczą małego oraz bardziej licznego zbioru danych wejściowych. Następnie, dla każdego z wyżej wymienionych zbiorów przeprowadziliśmy testy, które dotyczyły różnego doboru parametrów algorytmu. Badaliśmy poruszanie się algorytmu w różnych rodzajach sąsiedztw, podzielonych nie tylko ze względu na liczbę doboru, ale także na rodzaj, stopień losowości i inne właściwości. Badania te prowadziliśmy dla typowych (docelowych, realistycznych) parametrów naszego harmonogramu, czyli:

**Tab. 1.** Domyślne parametry harmonogramu

|  |  |  |
| --- | --- | --- |
| Parametr | Mały zbiór testowy (36 os.) | Duży zbiór testowy (100 os.) |
| Liczba klas zajęciowych | 1 | 2 |
| Liczba dni tygodnia | 6 | 6 |
| Liczba zajęć dziennie | 6 | 6 |
| Maksymalna liczba klientów | 5 | 5 |
| Cena biletu [zł] | 40 | 40 |
| Stawka godzinowa [zł] | 50 | 50 |
| Zarobki trenera, za pojawienie się danego dnia w klubie fitness [zł] | 50 | 50 |
| Dzienna cena wynajęcia sali [zł] | 200 | 200 |

Badaliśmy także inne parametry algorytmu. Warto dodać, że badania były prowadzone w sposób **wyrywkowy** - zmienialiśmy jedną z wcześniej opisywanych domyślnych wartości parametrów i sprawdzaliśmy jak ta zmiana wpływa na działanie algorytmu. Dla przypomnienia warto jeszcze raz przyjrzeć się domyślnym parametrom, które na podstawie wielu przeprowadzonych testów uznaliśmy za najlepsze:

**Tab. 2.** Domyślne parametry algorytmu

|  |  |
| --- | --- |
| Parametr | Wartość domyślna |
| alpha | 0.999 |
| Initial temp | 1000 |
| No. of iterations in one temp | 50 |
| Epsilon | 0.01 |
| Minimal temp | 0.1 |
| No of iterations without improvement | 200 |
| Squeeze initial solution | False |
| Improve results | True |
| Method of choosing neighborhood | Move one |
| Penalty method | False |

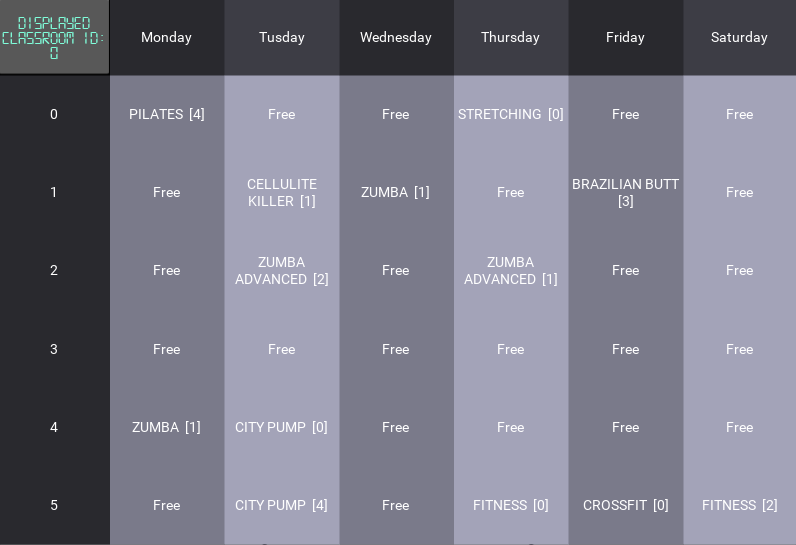
Warto zaznaczyć, że na tym etapie testów dane wejściowe były generowane **losowo** – tzn. Wybory klientów dotyczące ilości oraz rodzajów treningów były losowane tzn. Nie powinniśmy obserwować żadnej tendencji co do wyboru treningów - były one wybierane mniej więcej równomiernie.

Następnie przeszliśmy do drugiej fazy testów - sprawdzany był wpływ doboru parametrów harmonogramu na działanie algorytmu. Sprawdzaliśmy, jak algorytm zachowa się w takich sytuacjach, jak zmniejszona dzienna liczba zajęć, zmniejszona cena wynajęcia sali, zwiększone zarobki trenera za samo pojawienie się danego dnia w klubie.

Ostatnia – trzecia faza testów polegała na sprawdzeniu działania algorytmu dla złośliwych danych wejściowych (wiele osób wybrało ten sam trening, brak wystarczającej liczby instruktorów, zbyt duża łączna liczba wybranych treningów).

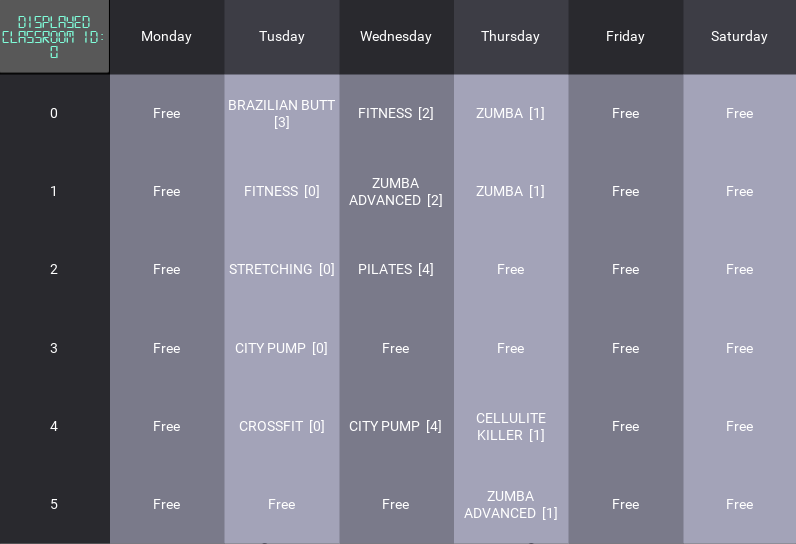
Testy dla danych wejściowych składających się z mniejszej liczby uczestników - 36 osób, 5 instruktorów

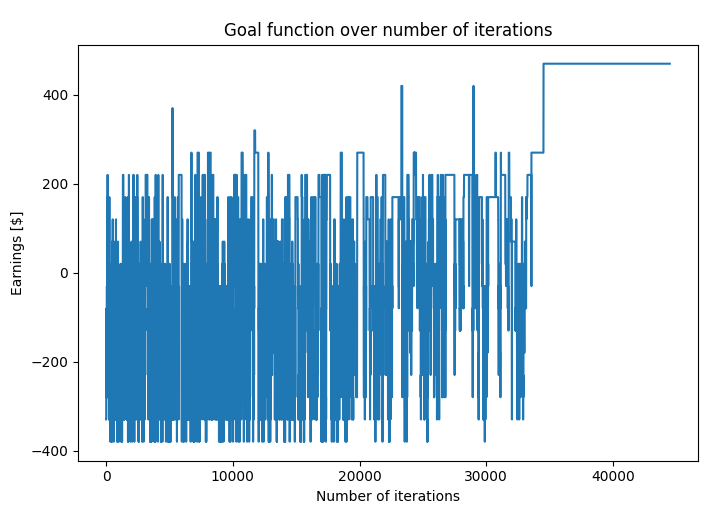
1. **Rodzaje sąsiedztw - pojedyncze sąsiedztwa**



**Rysunek 6.** Rozwiązanie początkowe - losowy harmonogram, taki sam dla kolejnych pięciu testów - liczby w nawiasach kwadratowych oznaczają id instruktora

1. Sąsiedztwo **move one –** losowe przesunięcie jednych zajęć w wolne miejsce

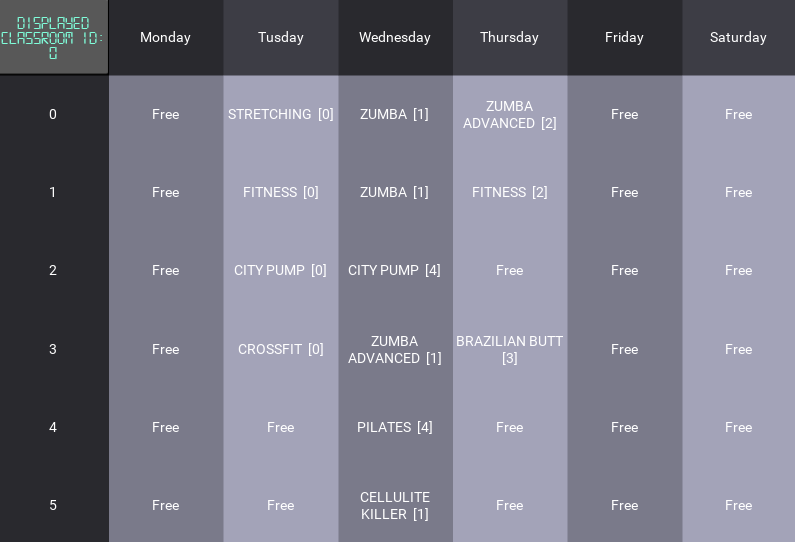


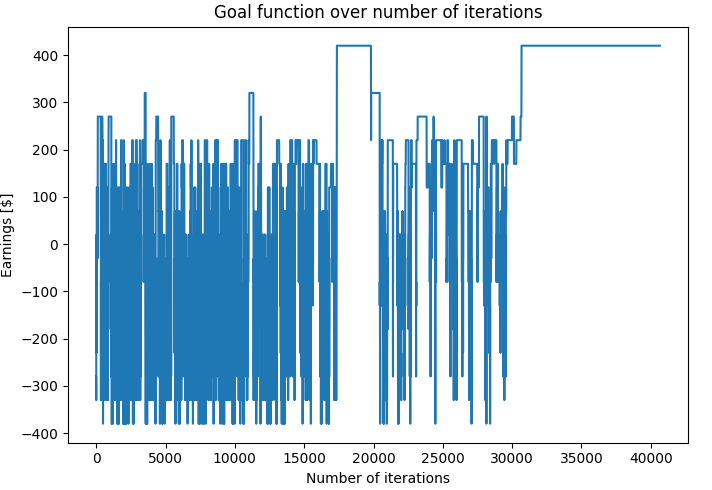


**Zarobki**: -330 -> 470 -> 620

**Wnioski do przebiegu:** Widać, że na początku występują spore oscylacje – algorytm odkładnie sprawdza różne rozwiązania w sąsiedztwie, jednak pod koniec oscylacje zaczynają gasnąć i algorytm znajduje rozwiązanie.

1. Sąsiedztwo **move two –** losowe przesunięcie dwóch zajęć w dwa wolne miejsca

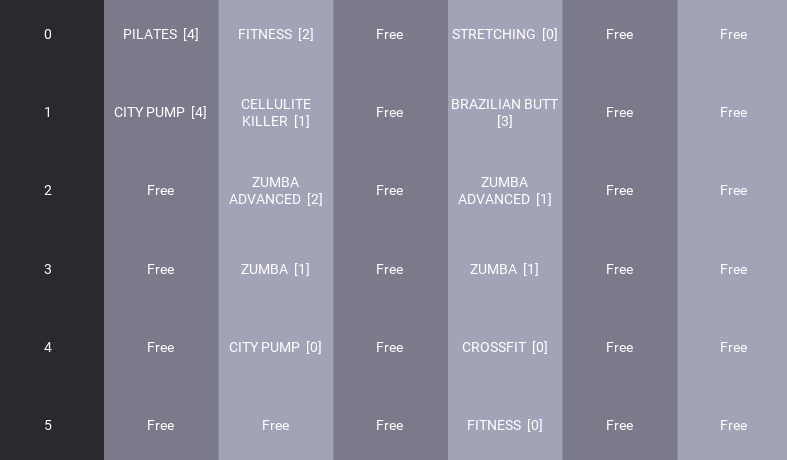


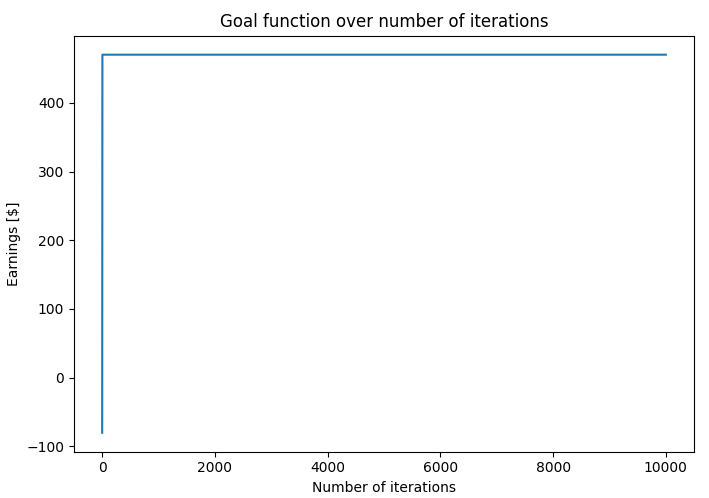


**Zarobki**: -330 -> 420-> 620

Wnioski do przebiegu: Przebieg nie różni się znacznie od poprzedniego. Jest to rozsądne zachowanie - dobór sąsiedztwa jest bardzo podobny w obu przypadkach, różni się tylko ilością przesuwanych zajęć.

1. Sąsiedztwo **Move to most busy** - przesunięcie zajęć z dnia, w które odbywa się ich najmniej w dzień, w którym odbywa się ich najwięcej

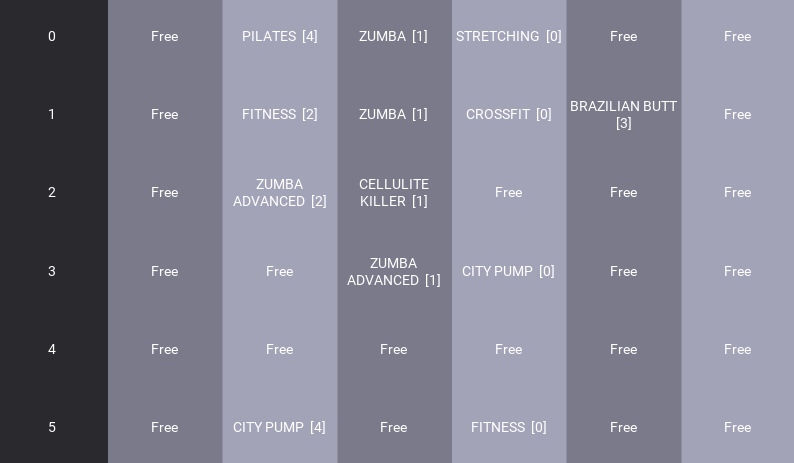


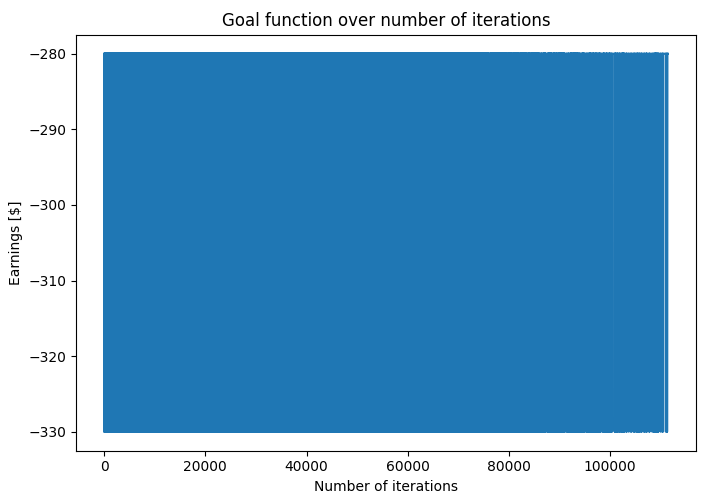


**Zarobki**: -330 -> 470 -> 520

**Wnioski do przebiegu:** W tym przypadku wykres jest bardzo ciekawy, jednak zachowanie funkcji jest dość łatwe do wytłumaczenia. W tym przypadku dobieramy sąsiedztwo w taki sposób, że przenosimy zajęcia z dni, w które odbywa się ich mała ilość na dni, w których jest dużo zajęć. Dlatego też już po kilku iteracjach wybrana metoda doboru sąsiedztwa nie ma zbyt dużego pola do manewru (wszystkie zajęcia są już upchane w 2-3 dni) i choć rozwiązanie jest dość dobre, to przestrzeń rozwiązań jest bardzo niedokładnie przeszukana i prawdopodobnie istnieje wiele lepszych rozwiązań. Warto zwrócić jednak uwagę, że ten dobór sąsiedztwa współpracuje bardzo dobrze z innymi sąsiedztwami.

1. Sąsiedztwo **Swap with most busy –** zamiana zajęć pomiędzy dniami z największą i najmniejszą liczbą zajęć

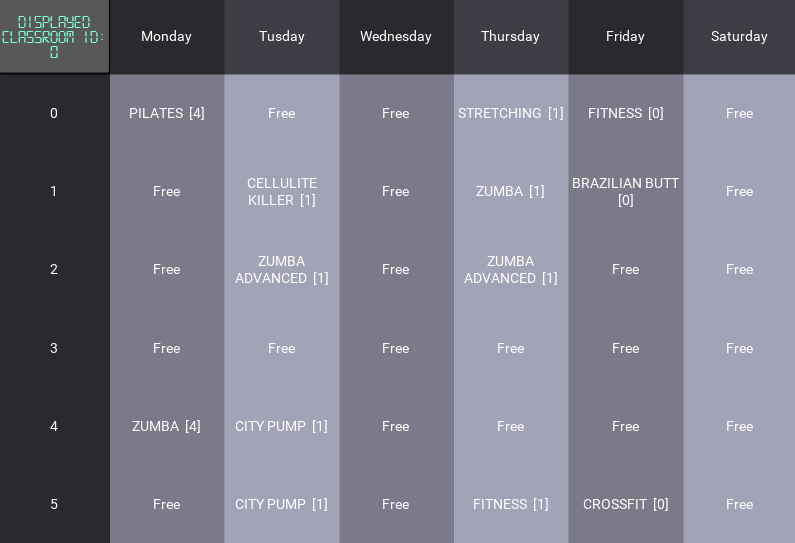


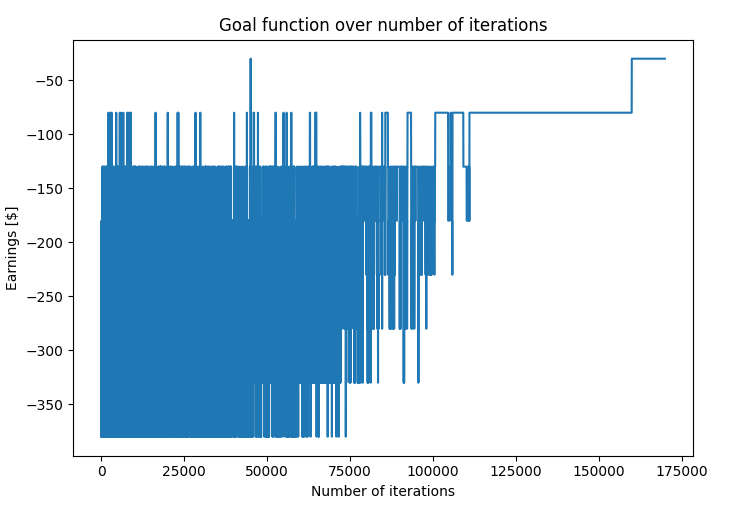


**Zarobki**: -330 -> -280-> 420

**Wnioski do przebiegu:** W tym przypadku przebieg funkcji też jest ciekawy, jednak łatwy do wytłumaczenia. Działa ona w taki sposób, że wybiera dwa dni, w które odbywa się dużo zajęć, wybiera po jednych zajęciach z obu dni i zamienia je miejscami. Dlatego też występują niegasnące oscylacje. Warto dodać, że ta metoda doboru sąsiedztwa także dobrze współpracuje z innymi sąsiedztwami, które wprowadzają losowość.

1. Sąsiedztwo **Change instructor –** losowa zmiana instruktora z zachowaniem ograniczeń.

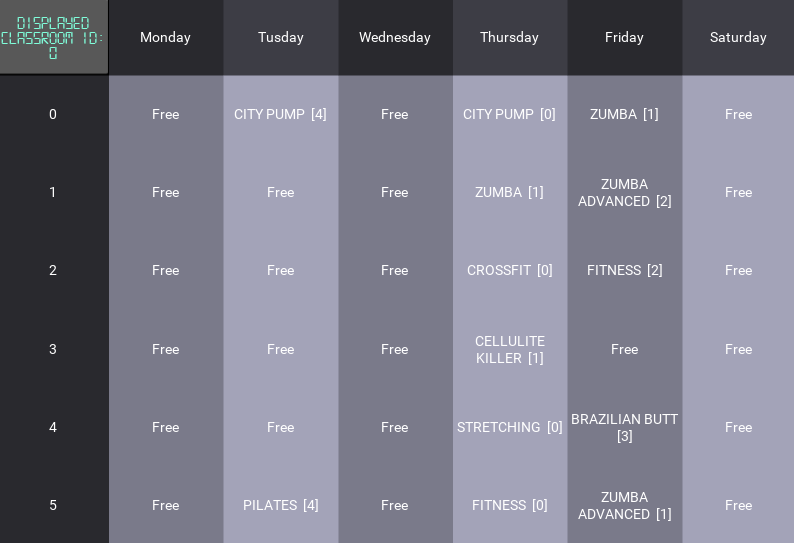


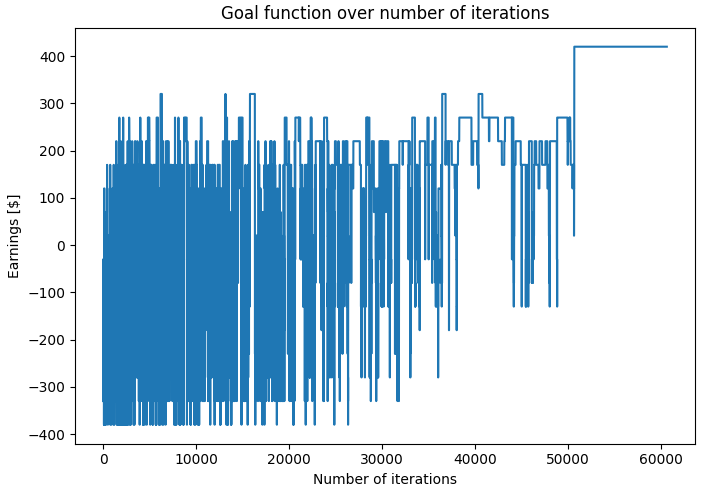


**Zarobki**: -330 -> -30 -> 470

**Wnioski do przebiegu:** Przebieg ten najbardziej przypomina typowy dla algorytmu Symulowanego Wyżarzania przebieg funkcji celu. Oscylacje na początku są bardzo duże, szybko zaczynają maleć, żeby w końcu znaleźć rozwiązanie. Uzyskane tą metodą zarobki wynoszą tylko –30 zł. Jest to spowodowane tym, że przy tym doborze sąsiedztwa algorytm nie może przenosić zajęć pomiędzy sobą, możliwe jest jedynie zmienianie prowadzących, co nie daje wystarczającej swobody do poszukiwań.

1. **Rodzaje sąsiedztw - kombinacje sąsiedztw**
2. Sąsiedztwa losowe – move one, move two

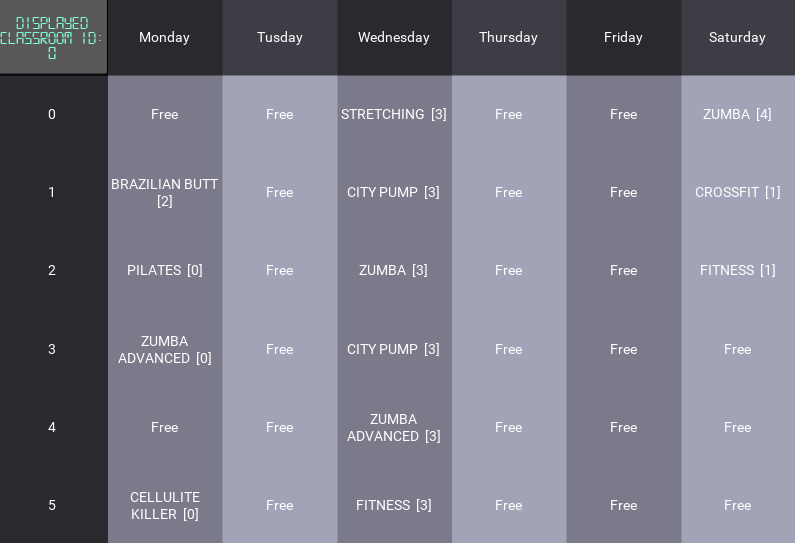


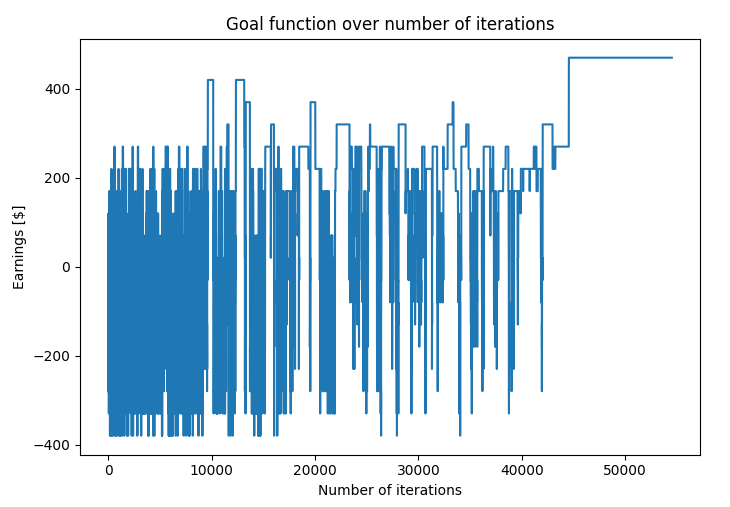


**Zarobki** –330 -> 420 -> 570

**Wnioski do przebiegu:** Kombinacja tych dwóch metod doboru sąsiedztwa nie zmienia wiele, otrzymujemy podobny przebieg jak w przypadku, gdy zastosujemy te metody osobno. Jest to typowy przebieg dla algorytmu SA.

1. Sąsiedztwa losowe – move one, move two + change instructor

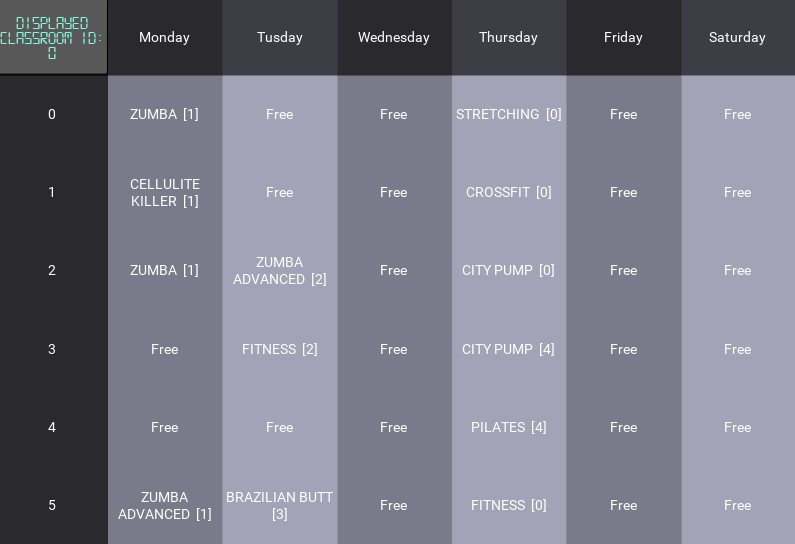


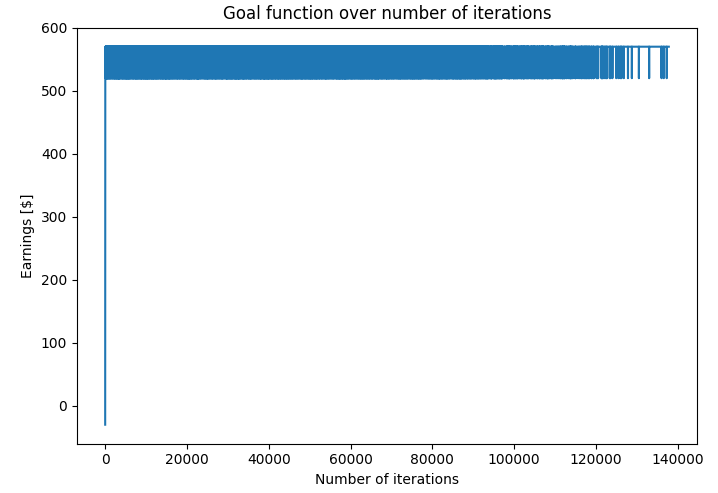


**Zarobki** –330 -> 420 -> 570

**Wnioski do przebiegu:** Kombinacja wszystkich trzech losowych metod doboru sąsiedztwa. Przebieg typowy dla Algorytmu SA. Długa niezmienny przebieg na końcu działania algorytmu wynika z zastosowanego parametru **n\_iter\_without\_improvement**.

1. Sąsiedztwa oparte na wartościach maksymalnych – move to / swap with most busy

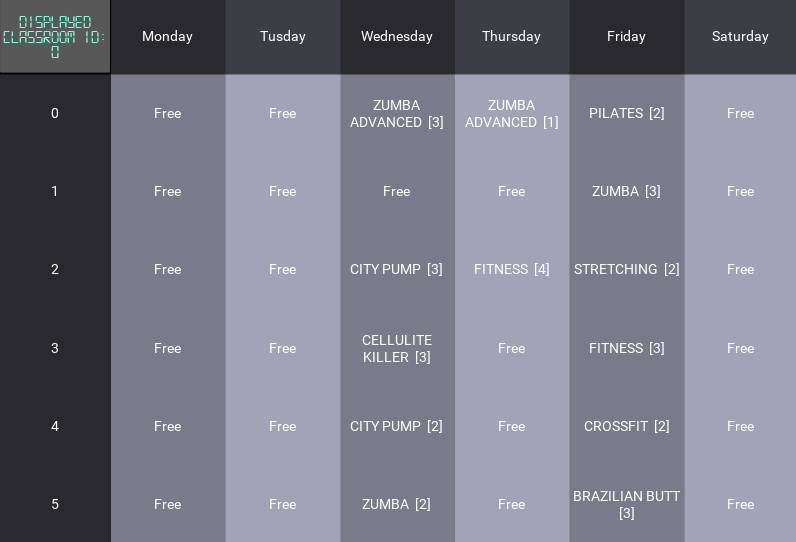


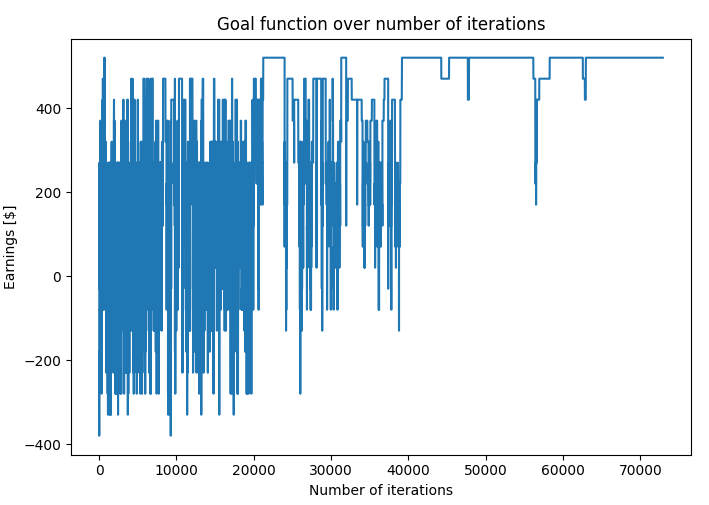


**Zarobki** –20-> 580-> 580

**Wnioski do przebiegu:** Kombinacja obu metod bazujących na wartościach maksymalnych. Jak widać z wykresu, posiada ona cechy obu metod doboru sąsiedztwa zastosowanych osobno. Z jednej strony można zaobserwować szybki wzrost wartości funkcji celu na początku działania algorytmu, z drugiej strony przez długi czas następują oscylacje.

1. Wszystkie sąsiedztwa naraz



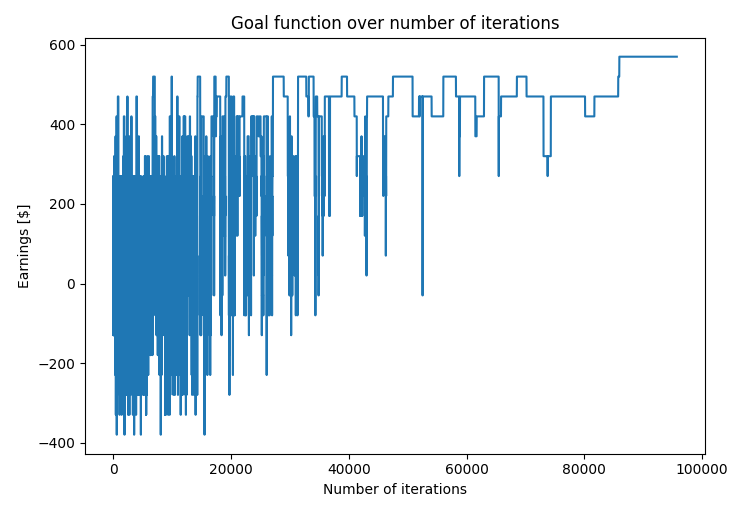


**Zarobki** –200-> 450-> 580

**Wnioski do przebiegu:** Kombinacja wszystkich metod. Otrzymujemy typowy dla Algorytmu SA przebieg, jedynie w okolicy 60000 iteracji obserwujemy “szpilkę” oznaczającą nagłą, dużą zmianę wartości funkcji celu.

1. Wszystkie sąsiedztwa naraz + zastosowanie funkcji kary



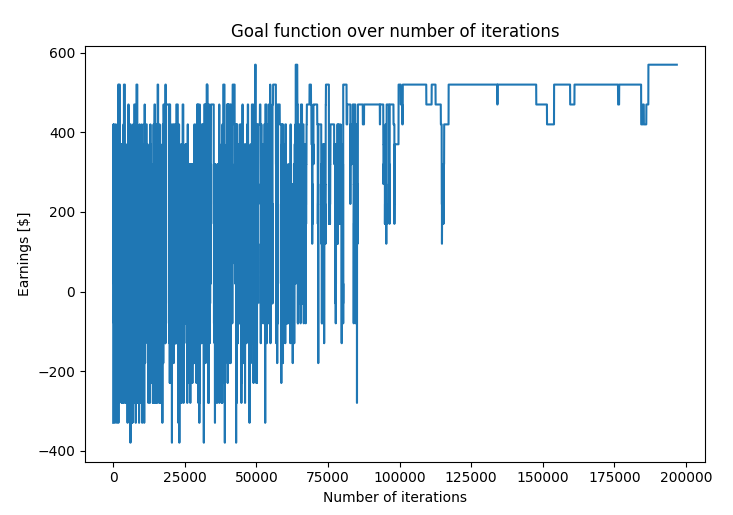


**Zarobki:** -100 -> 580 -> 620

**Wnioski do przebiegu:** Kombinacja wszystkich metod wraz z zastosowaniem funkcji kary zamiast sztywnych ograniczeń. Jak widać, przebieg algorytmu jest praktycznie perfekcyjny, działa on dokładnie tak jak symulowane wyżarzanie działać powinno. W dodatku otrzymujemy najlepszą dotychczas wartość funkcji celu.

1. Wszystkie sąsiedztwa naraz + zastosowanie funkcji kary + alpha = 0.9995



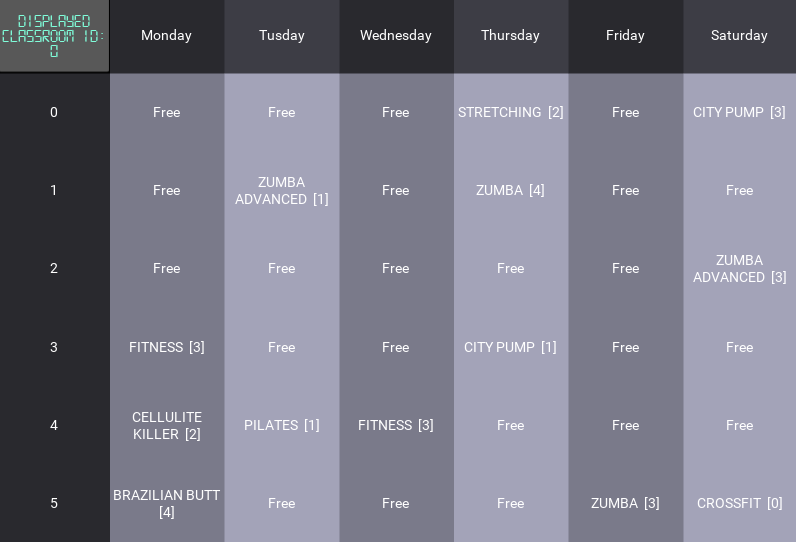


**Zarobki:** -100 -> 580 -> 620

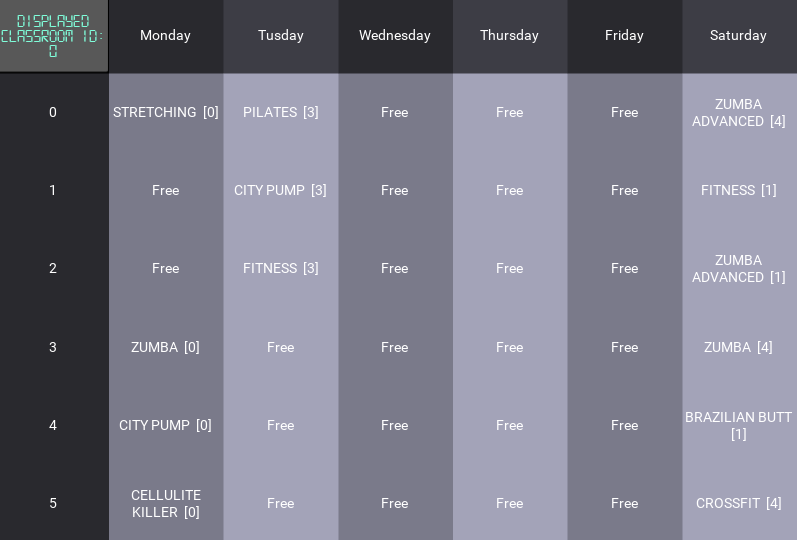
**Wnioski do przebiegu:** Kombinacja wszystkich metod wraz z zastosowaniem funkcji kary zamiast sztywnych ograniczeń. Jak widać, przebieg algorytmu jest praktycznie perfekcyjny, działa on dokładnie tak jak symulowane wyżarzanie działać powinno. Dodatkowe zwiększenie parametru alpha do 0.9995 powoduje dokładniejsze niż wcześniej przeszukiwanie, wolniejsze malenie wartości temperatury.

1. **Parametry harmonogramu**
2. Cena wynajęcia sali ustawiona na 1000 (zamiast 200), Sąsiedztwa: move one, move to most busy, change instructor. Zastosowana funkcja kary.

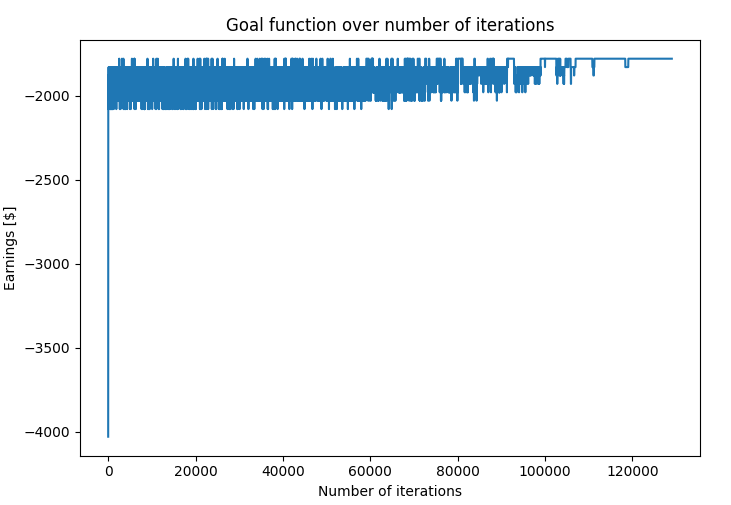
Przed:



Po:



Przebieg:

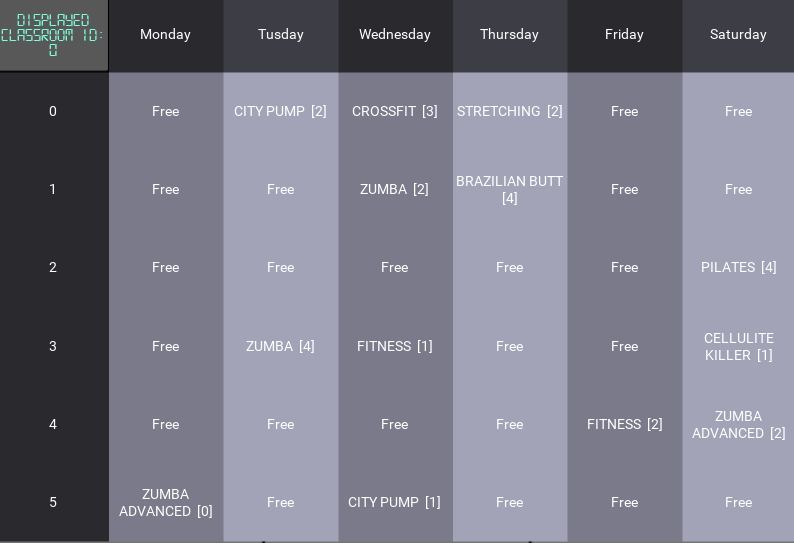


**Zarobki:** -4100-> -1920

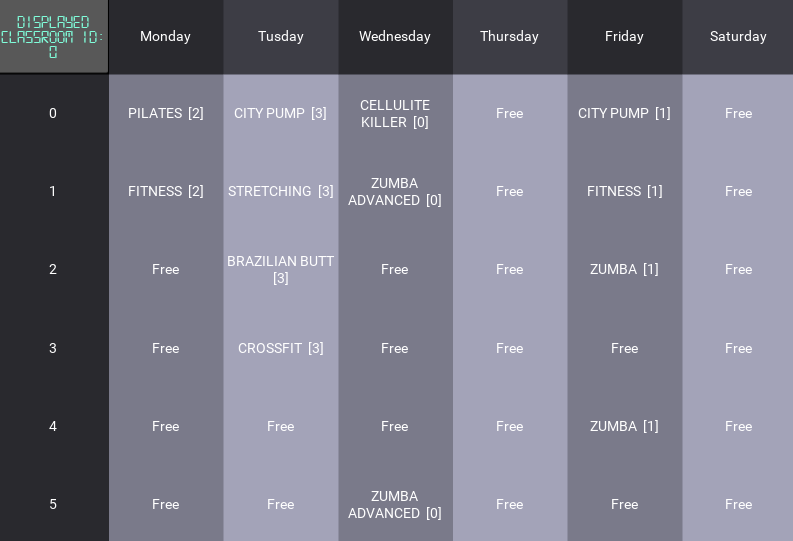
**Wnioski do przebiegu:** Na początku duży wzrost, ze względu na to, że wiele zależy od ilości zajętych sal (koszt wynajęcia wynosi aż 1000). Następnie przebieg dość typowy dla algorytmu SA.

b) Zarobki trenera za pojawienie się w klubie zwiększone z 50 do 250, zmniejszono koszt wynajęcia sali z 200 do 10, zezwolono na zmianę instruktora dla danych zajęć

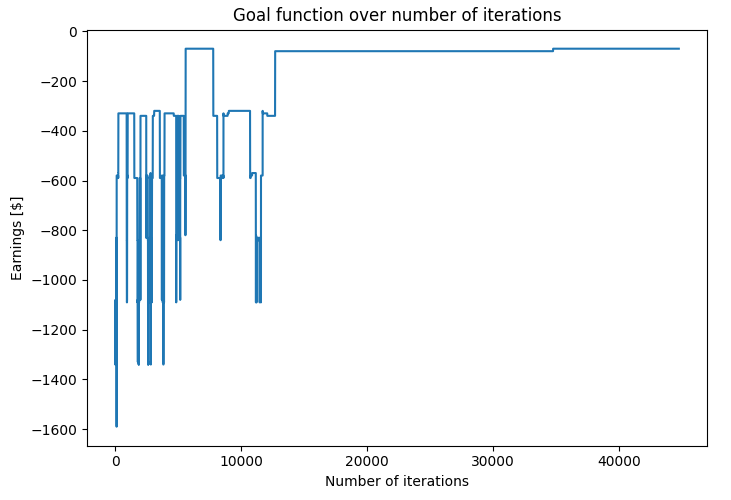
Przed:



Po:



Przebieg:



Zarobki: -1590 -> -70 -> 430

**Wniosek do przebiegu:** Algorytm zadziałał zgodnie z oczekiwaniami. Przywiązywał mniejszą wagę do ilości dni, w które odbywają się treningi, jednak każdy instruktor musi przyjść do pracy tylko raz w tygodniu.

1. **Przypadek złośliwy**: zbyt mała liczba przedziałów czasowych lub brak wykwalifikowanych instruktorów, aby zorganizować wszystkie zajęcia.

**Rezultat**: Program przestał działać, aplikacja wyłączyła się. Należy to rozwiązać w przyszłych wersjach oprogramowania. Możliwe rozwiązanie: algorytm automatycznie zwiększa liczbę zajęć organizowanych każdego dnia, aby pomieścić wszystkie zajęcia, które mają się odbywać.

1. **Podsumowanie testów dla mniejszej liczby klientów**

Wnioski dla poszczególnych przypadków testowych zostały zawarte pod wykresami ich dotyczącymi. Poniżej znajduje się tabela, która zawiera zbiorcze wnioski dotyczące działania algorytmu w przypadku całych grup testów.

**Tab. 3.** Ogólne wnioski dotyczące działania algorytmu

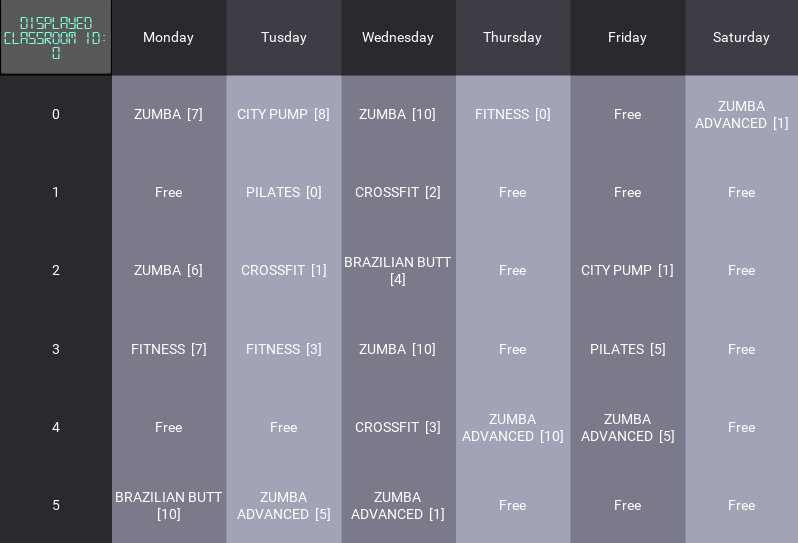
|  |  |
| --- | --- |
| **Grupa testów** | **Wnioski podsumowujące** |
| Pojedyncze sąsiedztwa | Pojedyncze sąsiedztwa działają generalnie najgorzej. Wielu rodzajom sąsiedztwa zastosowanym osobno brakuje pewnych cech. Niektóre sąsiedztwa powodują zbyt szybkie dążenie do rozwiązania co nie jest dobrym zjawiskiem, ponieważ niedokładnie przeszukujemy wtedy możliwe rozwiązania, inne sąsiedztwa powodują oscylacje, które objawiają się błądzeniem algorytmu pomiędzy kilkoma rozwiązaniami. |
| Kombinacje sąsiedztw | Kombinacje sąsiedztw działają lepiej niż pojedyncze sąsiedztwa. Najbardziej opłaca się łączyć ze sobą sąsiedztwa, które pochodzą z dwóch osobnych grup: działające losowo oraz na podstawie największej wartości. Gwarantuje to z jednej strony przeszukiwanie różnych rozwiązań - dzięki losowości, z drugiej strony poprawianie parametrów harmonogramu podczas działania - dzięki największej wartości. Te dwa rodzaje metod przeszukiwania sąsiedztwa dopełniają się nawzajem gwarantując dobre działanie algorytmu. |
| Kombinacje sąsiedztw z zastosowaniem funkcji kary | Takie ustawienie parametrów gwarantuje uzyskanie najlepszych wyników. Metoda funkcji kary, dzięki temu, że pozwala na naruszenie ograniczeń, daje większą swobodę algorytmowi w przeglądaniu możliwych rozwiązań, dzięki czemu jest on w stanie dotrzeć do takich ułożeń harmonogramu, do których nie dotarłby bez przejścia przez rozwiązania niedozwolone. |
| Wpływ parametrów harmonogramu | Algorytm reaguje na zmiany parametrów harmonogramu bardzo dobrze, sensownie, tak jak zareagowałby człowiek. Gdy zwiększymy cenę wynajmu sali algorytm zwraca dużo większą uwagę na to, aby sala była zajęta najmniej razy w tygodniu. |
| Przypadki złośliwe | Program nie radzi sobie najlepiej, gdy natrafi na złośliwe przypadki. Najczęściej kończy się to przedwczesnym zakończeniem działania algorytmu bądź wyłączeniem programu. Jest to negatywna cecha, którą należy poprawić w przyszłych wersjach oprogramowania. |

**Tab. 4.** Zestawienie wyników działania samego algorytmu SA w poszczególnych testach

|  |  |  |
| --- | --- | --- |
| **Sąsiedztwo** | **Uzyskany wynik** | **Liczba iteracji** |
| Move one | 470 zł | 35 tyś. |
| Move two | 420 zł | 31 tyś. |
| Move to most busy | 470 zł | < 100 |
| Swap with most busy | -280 zł | > 100 tyś. |
| Change Instructor | -30 zł | 175 tyś. |

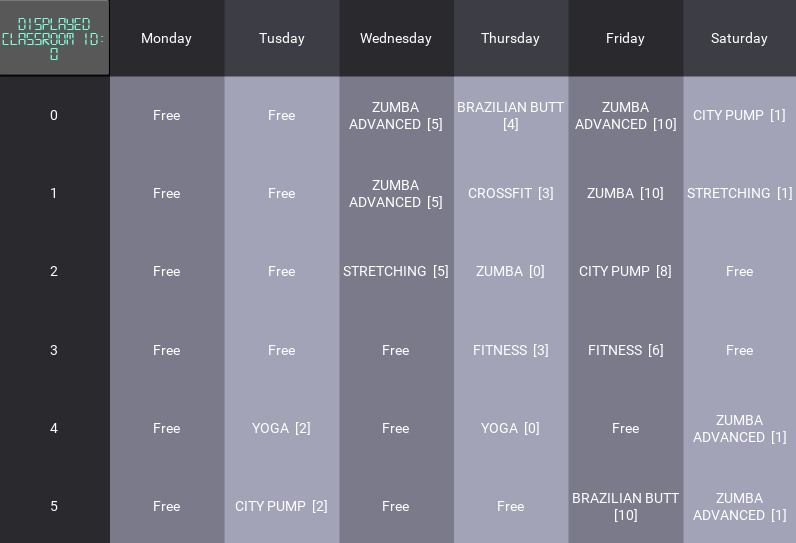
**2) Problem większych rozmiarów**

1. ****Początkowe ułożenie (bez optymalizacji)****

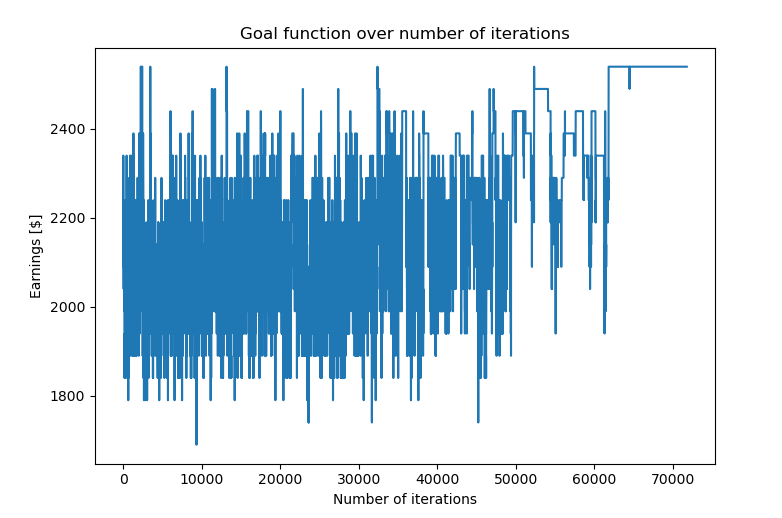

1. Sąsiedztwo move one – losowe przesunięcie jednych zajęć w wolne miejsce

**Pierwsza sala zajęciowa:**



**Druga sala zajęciowa:**





**Wnioski do przebiegu:** Dość typowy przebieg dla algorytmu SA, widać mocne oscylacje oraz pod koniec stopniowe schodzenie do wartości najlepszej.

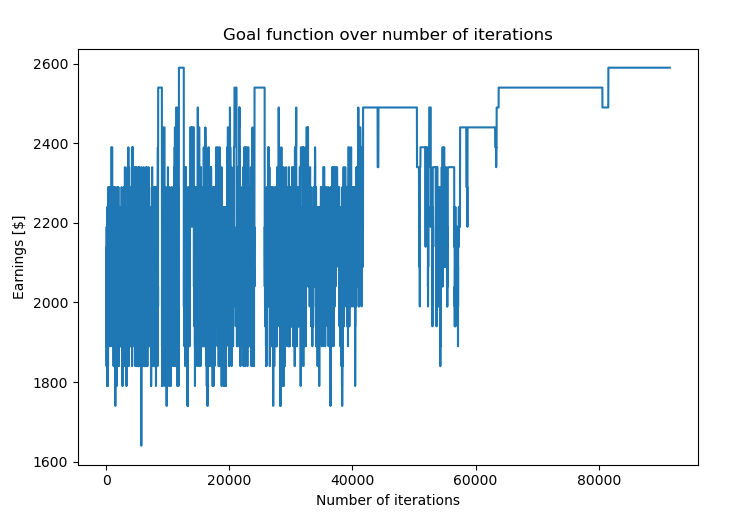
1. Sąsiedztwo move two – losowe przesunięcie dwóch zajęć w dwa wolne miejsca

**Pierwsza sala zajęciowa:**



**Druga sala zajęciowa:**





**Wnioski do przebiegu:** Podobnie jak poprzedni przykład dość typowy przebieg dla algorytmu SA, widać mocne oscylacje oraz pod koniec stopniowe schodzenie do wartości najlepszej.

1. Sąsiedztwo Move to most busy - przesunięcie zajęć z dnia, w które odbywa się ich najmniej w dzień, w którym odbywa się ich najwięcej

**Pierwsza sala zajęciowa:**

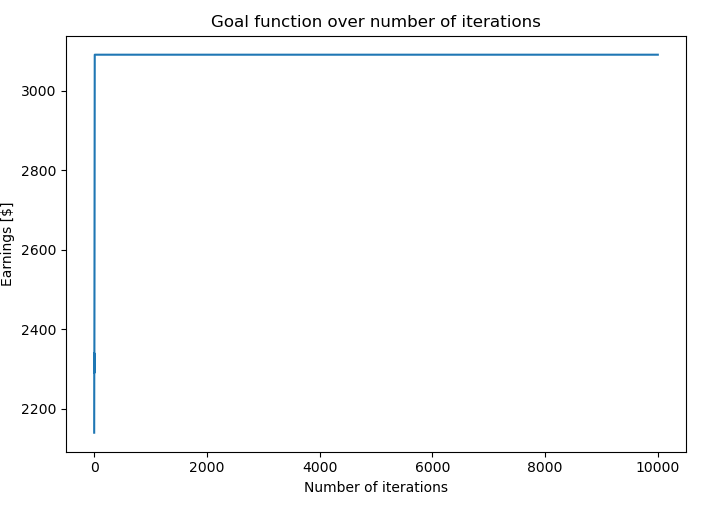
Obraz zawierający stół

Opis wygenerowany automatycznie

**Druga sala zajęciowa:**

Obraz zawierający stół

Opis wygenerowany automatycznie



**Wnioski do przebiegu:** Jak widać algorytm szybko dąży do wartości największej, jest to związane z tym, że w tej metodzie doboru sąsiedztwa staramy się “upychać” zajęcia z dni względnie luźnych do tych bardziej zajętych co naturalnie zwiększa funkcję celu poprzez redukcję kosztu wynajęcia sali oraz kosztu opłacenia obecności instruktorów. Takie podejście jednak prowadzi do dość szybkiego zablokowania możliwości ruchu.

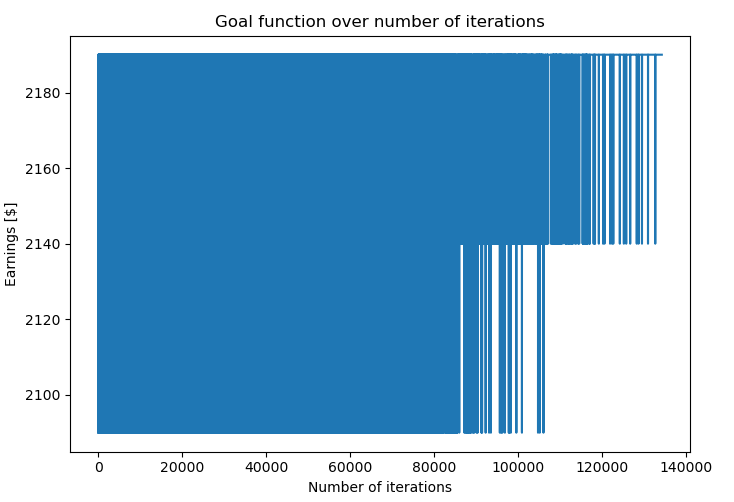
1. **Sąsiedztwo **Swap with most busy –** zamiana zajęć pomiędzy dniami z największą i najmniejszą liczbą zajęć.**

**Pierwsza sala zajęciowa:**



**Druga sala zajęciowa:**

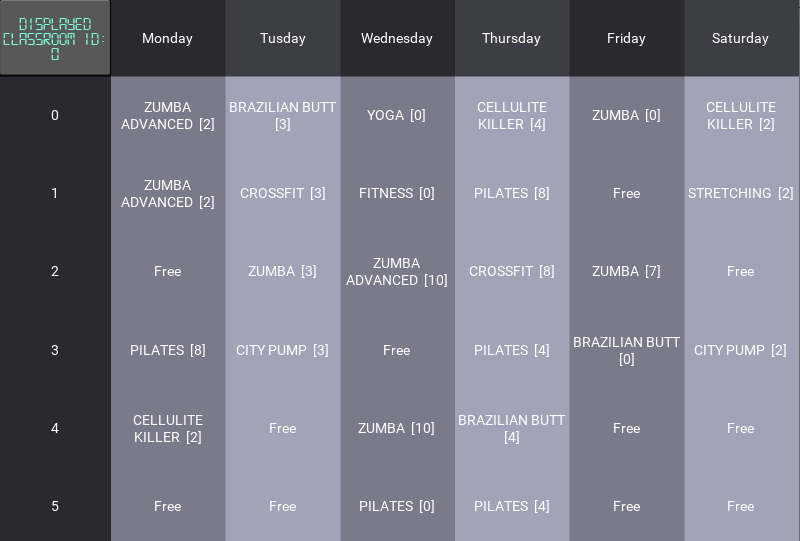




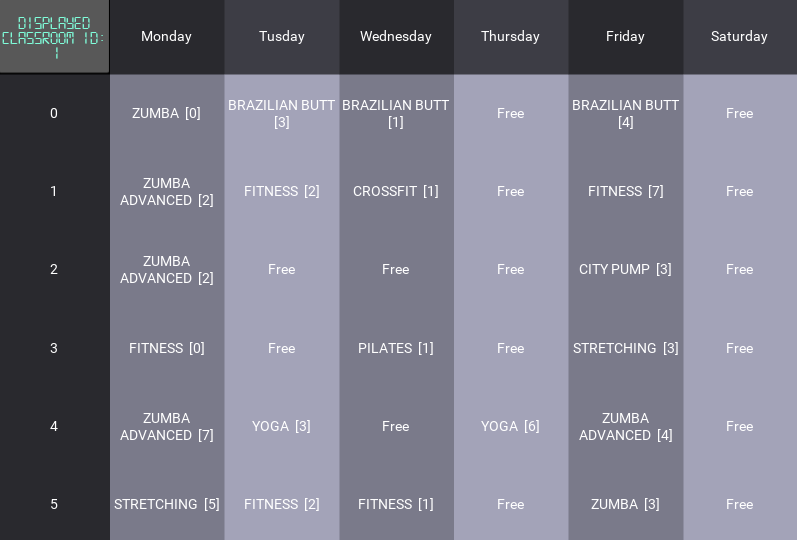
**Wnioski do przebiegu:** Widoczne na wykresie duże oscylacje są wynikiem tego, że przy tej metodzie wyboru sąsiedztwa algorytm może jedynie zamieniać miejscami zajęcia, nie mogąc przy tym przenieść zajęć w puste miejsca. Przez to algorytm może kilkukrotnie wpadać w te same miejsca.

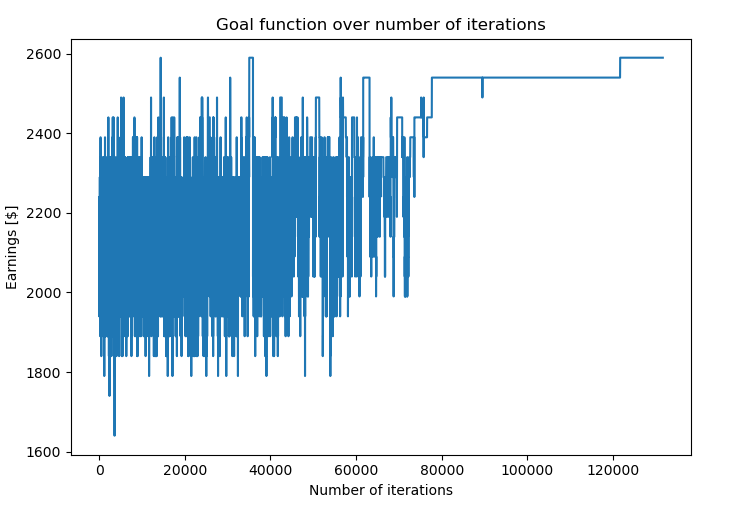
1. **Sąsiedztwo **Change instructor** - sąsiedztwo wybierane poprzez losową zmianę instruktora z zachowaniem ograniczeń.**

**Pierwsza sala zajęciowa:**



**Druga sala zajęciowa:**

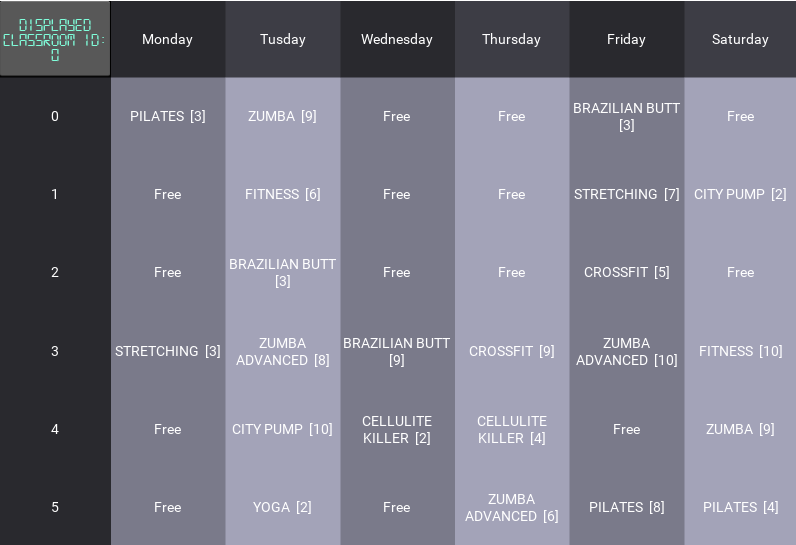




**Wnioski do przebiegu:** Widoczny przebieg jest zbliżony do dwóch pierwszych przykładów, zauważalne jest natomiast szybsze zbliżanie się do wartości największej.

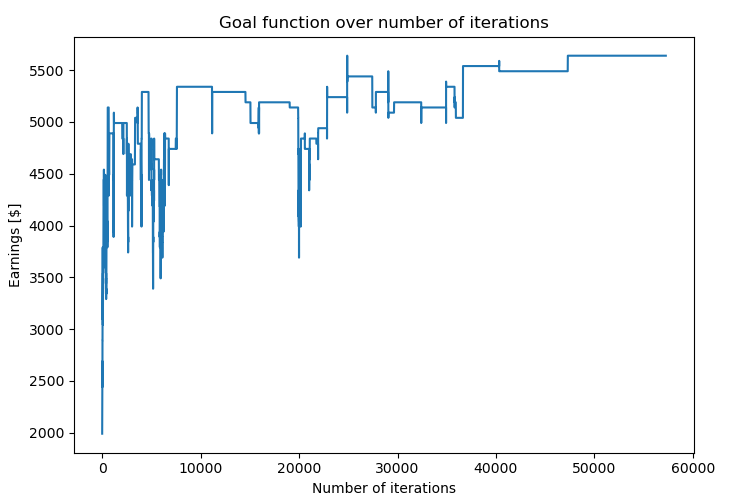
1. **Sąsiedztwo **Change instructor with penalty method**  - sąsiedztwo wybierane poprzez losową zmianę instruktora, możliwe zmiany naruszające ograniczenia kwalifikacji oraz obecności w dwóch miejscach naraz.**

**Pierwsza sala zajęciowa:**



**Druga sala zajęciowa:**





**Wnioski do przebiegu:** Jak widać przebieg w tym przypadku jest dość nietypowy jest to związane z możliwością naruszania ograniczeń przez algorytm. Ma to również pozytywne skutki - uzyskany wynik jest jednym z najlepszych. Dobry rezultat spowodowany jest większą swobodą metody – algorytm może poprzez rozwiązania niedopuszczalne trafić na normalnie nieosiągalne rozwiązanie.

1. **Podsumowanie testów dla większej liczby klientów**

**Tab. 5.** Zestawienie wyników działania algorytmu SA bez dodatkowej funkcji poprawiającej

|  |  |  |
| --- | --- | --- |
| **Sąsiedztwo** | **Uzyskany wynik** | **Liczba iteracji** |
| Wartość początkowa | $1940 | - |
| Move one | $2540 | 71800 |
| Move two | $2690 | 88800 |
| Move to most busy | $3090 | 10000 |
| Swap with most busy | $2190 | 134200 |
| Change Instructor | $2590 | 131650 |
| Change Instructor with penalty method | $5640 | 57250 |

**Wnioski podsumowujące:** Najlepsze wyniki zdecydowanie osiąga metoda z wykorzystaniem funkcji kary, okupione jest to możliwością uzyskania rozwiązania niedopuszczalnego. Takie rozwiązanie można jednak dość łatwo poprawić dodatkową funkcją “sprzątającą” - np. zbliżony efekt daje nasza funkcja *improve\_results.* Drugie najlepsze rezultaty uzyskaliśmy przy wykorzystaniu move to most busy, która w krótkim czasie dochodzi do przyzwoitych rezultatów

4. Podsumowanie

Wnioski

Wiele wniosków zostało zawartych w rozdziale dotyczącym testów. Aplikacja działa bardzo dobrze dla przypadków naturalnych, rzeczywistych, mało złośliwych. Warto zwrócić uwagę na to, że odejmuje ona wiele pracy osobom, które musiałyby ułożyć taki harmonogram. Jedyne co aplikacja dostaje jako dane wejściowe to parametry algorytmu - które i tak można pozostawić domyślne, ponieważ działają one poprawnie dla większości przypadków, parametry harmonogramu – indywidualne dla każdej instytucji a także dwa proste pliki wejściowe - format \*.csv - które mogą być zbierane automatycznie przez takie narzędzia jak Google Forms. Oznacza to, że jedyne, czego wymaga się od użytkownika to ustawienie parametrów harmonogramu i przygotowanie formularza zgłoszeniowego (który i tak musiałby zostać dla takiej instytucji przygotowany). Resztę pracy wykonuje algorytm, który zwraca gotowy harmonogram, który jest często ułożony bezbłędnie bądź wymaga drobnej korekty. Dzięki aplikacji, którą napisaliśmy, człowiek jest w stanie zaoszczędzić wiele godzin pracy, a także dodatkowo zostać poinformowany o potencjalnych zarobkach, które wynikałyby z takiego ułożenia harmonogramu.

Stwierdzone problemy

Aplikacja na ten moment posiada kilka bug’ów, które zostaną naprawione w jej następnych wersjach. Jednym z takich błędów jest wyłączanie się aplikacji, gdy natrafi ona na zbyt liczny zbiór testowy (brak możliwości rozlokowania tak dużej liczby osób). Możliwe rozwiązanie - automatyczne zwiększanie liczby klas, w których mogą odbywać się zajęcia. Oprócz tego występują czasami problemy z pakietem kivy garden, które wymagają zakomentowania jednej linijki kodu w pliku backend\_kivy.py - konkretnie **from matplotlib import \_png**. Problem ten pojawia się jednak bardzo rzadko.

Kierunki dalszego rozwoju

Potencjalnym kierunkiem dalszego rozwoju aplikacji byłoby dodanie większej liczby algorytmów optymalizacyjnych, takich jak **tabu search** oraz **algorytm genetyczny** i porównanie działania z obecnie zaimplementowanych algorytmem symulowanego wyżarzania. Kolejną opcją jest dorobienie możliwości generowania pliku wyjściowego - tabela w programie Excel a także możliwość przesuwania pojedynczych zajęć w wynikowym harmonogramie z poziomu programu.