Sprawozdanie końcowe
Model prowadzenia Winnicy

Autorzy:

Julia Nowak

Paweł Radzik

Filip Huczek

Spis treści

Podział pracy

Etap	Osoba	Wykonane zadania
Model zagadnienia	Julia	**Funkcja celu (Command_files.py)** Ocena() isOK_size() wypisz() 150/150 [100%] **Model zagadnienia [100%]** **Struktury danych (Generators.py):** generate_solution(), vine_price_generator(), plant_price_generator(), soil_quality_generator() gen() - 200/400 [50%]
	Paweł	**Sąsiedztwo (cześć generatorów)** generateSolutionFromNumber() generateAntiNum() przelicz() generateAllsolutions() generateAllsolutionsFromAspi() 100/400 **Postać Rozwiązania**
Algorytm - opracowanie	Paweł	**Tabu Search [90%]** -Pamięć średnioterminowa -Kryterium Aspiracji -Pamięć długoterminowa -Pomiar czasu i wydajności pamięciowej -Naprawienaie błędów -Ograniczenia górne **Implementacja sąsiedztwa [100%]** **Parametryzacja Sąsiedztwa i TabuSearch[100%]**
	Julia	**Tabu Search [10%]** - pierwsza wersja algorytmu - przerzucenie do App **Rozwiązanie początkowe [100%]**

Implementacja Aplikacji	Julia	**GUI [90%]** 900/900 – cały plik ui_app.py Plik app.ui Dostosowanie kodu App.py do app.ui **Zapis i odczyt z plików [100%]** **Tworzenie wykresów [100%]** canvas.py 100/100 Wykresy.py 80/80
	Paweł	**Dodawanie Flag Aplikacji [10%]** W pliku App/py
Testy	Paweł	**Testy list** **Testy pamięci**
	Filip	**Testy Generatorów 500 /500 [100%]** Ceny roślin, nawozu, obsadzonego pola, gen, antinum, pomocnicze Plik Testowanie.py
	Julia	**Testowanie** Plik Testy.py 200/200 [100%] Tryb testowy w aplikacji [100%] Testy sąsiedztwa Testy rozwiązania początkowego, epsilona, iteracji, list Pliki i foldery z zadaniami testowymi
Dokumentacja	Filip	**Wersja początkowa [100%]** Rozdział 1-3
	Julia	**Wersja końcowa [75%]** Rozdziały 1, 3, 4, 5 (dopisanie i korekta)
	Paweł	**Wersja końcowa [25%]** Rozdział 2 (dopisanie i korekta)

Model zagadnienia

Opis

Zagadnienie, które rozważamy, to problem optymalizacji bilansu zysków i strat pewnej winnicy. Oczywistym jest, że każdy przedsiębiorca chce, aby jego zyski były jak największe, a straty – jak najmniejsze . Głównym celem naszego zagadnienia jest więc maksymalizacja, a tym, co chcemy zoptymalizować, jest ilość jednostek winogron danego typu na posiadanych polach. Postać naszego rozwiązania jest więc „kostką" – obiektem 3D. Dzięki takiemu rozwiązaniu prawie można poczuć na własnej skórze jak to byłoby mieć własną, prosperującą winnicę produkującą wychwalane na cały świat wino.

W modelu znalazły się najważniejsze etapy produkcji wina:

- Sadzenie winogron na polu, które jest ograniczone przez powierzchnię i ograniczenie dolne
- Nawożenie
- Zbiory winogron z pól, uwzględniające jakość gleby i pory roku
- Butelkowanie
- Magazynowanie butelek, decyzja o ich sprzedaży lub przechowywaniu w zależności od zapotrzebowania sklepu oraz pojemności magazynu
- Transport do sklepu

Przyjęliśmy, że winogrona są sadzone w marcu, lipcu i listopadzie. Mamy całkowity zakaz sadzenia w zimie, pozostałe miesiące są wysoce nieoptymalne.

Winogrona, które nie utworzą jedności, są uznawane jako straty i wyrzucana. Nie generują one jednak strat pieniężnych, a jedynie straty liczebne.

Nadmiar winogron w magazynie obarczony jest kosztem, ale także naliczany jest zysk – nadwyżki nie wyrzucamy, a jedynie sprzedajemy za 40% wartości.

Trójpolówka nalicza bonus za sadzenie w losowo określony, stały sposób.

Model matematyczny

Ogólna postać modelu matematycznego to:

$$\sum_{m=0}^{12*years-1}\left(\sum_{f=0}^{fields-1}\left(\sum_{t=0}^{types-1}(monthGain_m - monthCost_m - monthRent)\right)\right) \rightarrow max$$

Model posiada jednak dość złożoną strukturę, która znacznie rozszerza jego prostą na pierwszy rzut oka formę. Poniżej przedstawiono elementy, składające się na monthGain i monthCost.

$$plantingCost = \begin{cases} x_{mft} * KObs_t + IsFert * fertCost * x_{mft}, & \boldsymbol{m} mod 12 \notin \{0,1,11\} \\ 0, & \boldsymbol{m} mod 12 \in \{0,1,11\} \end{cases}$$

$$fieldPunish = \begin{cases} x_{mft} * 50.00, & x_{mft} < lower_f \\ x_{mft} * 100.00, & x_{mft} > upper_f \end{cases}$$

$$gather = \begin{cases} 0, & \boldsymbol{m} mod 12 \in \{0,1,11\} \\ x_{mft} * \left(soilQuality_{mft} + IsFert * fertBonus\right), & \boldsymbol{m} mod 12 \notin \{0,1,11\} \end{cases}$$

$$harvestCost = \begin{cases} 0, & \boldsymbol{m} mod 12 \in \{0,1,11\} \\ gather * harCost, & \boldsymbol{m} mod 12 \notin \{0,1,11\} \end{cases}$$

$$bottles = \frac{gather}{ppb} + remains_t$$

$$bottlingCost = \frac{gather}{ppb} * bottlingPrice$$

$$bottlesSelled = \begin{cases} bottles, & bottles \leq storeNeeds_t \\ storeNeeds_t, & bottles > storeNeeds_t \end{cases}$$

$$bottlesRemained = \begin{cases} 0, & bottles \leq storeNeeds_t \\ bottles - storeNeeds_t, & bottles > storeNeeds_t \end{cases}$$

$$BotTransCost = bottlingCost + transCost * bottlesSelled$$

$$bottleGain = bottlesSelled * bottlePrice_{tm}$$

$$magOverflowCost = \begin{cases} (sum(remains) - magazineCapacity) * 0.4 * bottlePrice_{tm}, & magazineCapacity < sum(remains) \\ 0, & magazineCapacity \geq sum(remains) \end{cases}$$

$$magOverflowGain = \begin{cases} (sum(remains) - magazineCapacity) * 0.8 * bottlePrice_{tm}, & magazineCapacity < sum(remains) \\ 0, & magazineCapacity \geq sum(remains) \end{cases}$$

$$\begin{aligned} monthCost = {} & plantingCost + fieldPunish + harvestCost + BotTransCost \\ & + bottlesRemained * magazineCost + storeNeeds_t * NotAchievedGoalPunish \\ & + magOverflowCost \end{aligned}$$

$$monthGain = bottleGain + magOverflowGain$$

Algorytm

Schemat algorytmu

1. Inicjacja flag i potrzebnych zmiennych
2. Inicjacja zmiennych wyjściowych i zmiennych do tworzenia wykresów
3. Wygeneruj sąsiedztwo na podstawie flag (szczegóły w sekcji generacja rozwiązania)
4. Ustawienie wartości najlepszego rozwiązania na –inf
5. Przejdź do następnego rozwiązania
6. Jeżeli rozwiązanie jest na tabu liście pójdź do kroku 5
7. Oblicz wartość funkcji dla rozwiązania
8. Porównaj aktualnie obliczoną wartość z poprzednią najlepszą wartością uwzględniając zniechęcenie wynikające z pamięci długoterminowej
    8.1. Jeżeli jest większa zamień największe rozwiązanie na aktualne
9. Jeżeli są inne rozwiązania przejdź do punktu 5
10. Jeżeli Spadek występował przez n kolejnych iteracji, gdzie n to parametr ustawiany w ustawieniach
    10.1. Zregeneruj rozwiązania z listy tabu
    10.2. Przejdź do następnego rozwiązania z listy tabu
    10.3. Oblicz wartość funkcji dla rozwiązania
    10.4. Porównaj aktualnie obliczoną wartość z poprzednią najlepszą wartością uwzględniając zniechęcenie wynikające z pamięci długoterminowej
        10.4.1. Jeżeli jest większa o pewną wartość ustawioną w ustawieniach zamień największe rozwiązanie na aktualne
    10.5. Jeżeli są inne rozwiązania na liście tabu przejdź do punktu 10.2
11. Jeżeli rozwiązanie jest gorsze od poprzedniego zwiększ liczniki aspiracji i pamięci średnioterminowej
12. Jeżeli Spadek występował przez m kolejnych iteracji, gdzie m to parametr ustawiany w ustawieniach zresetuj rozwiązanie do losowego poprzedniego rozwiązania zapomnij część tabu listy, zmieniasz pamięć krótkoterminową.
13. Zapisz rozwiązanie i jego wartość do listy rozwiązań
14. Dodaj do pamięci długoterminowej aktualne rozwiązanie
15. Dodaj do tabu listy rozwiązanie przeciwne
16. Sprawdź warunki stopu (epsilon i liczba iteracji)
17. Jeżeli nie STOP, przejdź do punktu 3
18. Zapisz potrzebne dane statystyczne do pamięci programu i stwórz wykresy

Otoczenie i jego generacja:

Z powodu tego, że nasza przestrzeń rozwiązań jest ogromna (trójwymiarowa macierz) podjęliśmy decyzję deweloperską, że naszym rozwiązaniem co prawda będzie

macierz, ale za to do tabu listy będziemy dodawać drogę przeciwną do tej która nas do niej zaprowadziła np. Zwiększając wartość na polu nr 1 w maju dla jakiegoś typu wina na tabu listę wrzucamy rozwiązanie przeciwne, czyli dla pola nr 1 i w maju odejmowanie danego rodzaju wina.

Odpowiednio dla listy długoterminowej też nie zniechęcamy do poszczególnych rozwiązań tylko zmian na danych polach reprezentowanych przez liczby jednoznacznie mówiące jaka zmiana zaszła.

Funkcja generująca otoczenie działa w ten sposób że generuje każdą możliwą liczbę z przedziału <0, IlośćPólWMacierzy*2>liczby te jednoznacznie mówią o miejscu gdzie dokonywana jest zmiana a ostatni bit czy jest to odejmowanie czy dodawanie dzięki czemu żeby wygenerować rozwiązanie przeciwne wystarczy zmienić wartość ostatniego bitu. Jeżeli jest włączona generacja tylko pewnej części sąsiedztwa dla każdej liczby jest losowane czy ona wystąpi zgodnie ze współczynnikiem umieszczonym we fladze dzięki temu występuje tylko część z całej przestrzeni rozwiązań . Dodatkowym Parametrem jest sposób określania wartości może być deterministyczna czyli odejmujemy i dodajemy tą samą wartość lub losowa ograniczona przez parametry z ustawień.

Pamięci

W wyniku zmiany koncepcji z rozwiązania na “drogę” wynika parę konsekwencji dla założeń algorytmu:

Pamięć krótkoterminowa - TL nie zabrania konkretnych rozwiązań tylko drogę do nich wynika z tego że teoretycznie możliwe byłoby dojście do poprzedniego rozwiązania poprzez inną drogę przy wykorzystaniu innej metody generowania niż klasyczna. Wynika z tego że zabronione są też permutacje drogi która została odbyta.

Pamięć średnioterminowa - Pamięć średnioterminowa służy do wyciągania nas z beznadziejnych rozwiązań i gdy nie uzyskamy poprawy rozwiązania przez pewien okres “cofa” nas do jednego z poprzednich rozwiązań

Pamięć długoterminowa - zamiast zniechęcać do konkretnych rozwiązań zniechęca do podejmowania tej samej ścieżki dzięki czemu program podejmuje sprawdzanie większej ilości rozwiązań niekoniecznie najbardziej optymalnych może to pomagać w szukaniu innych dróg niż te z minimum lokalnego.

Funkcja Aspiracji - jest aktywowana podobnie do pamięci średnioterminowej jednak wyliczenie wartości funkcji musi się odbyć ponownie. Następnie wybieramy rozwiązanie z wcześniej wygenerowanych, jeżeli przekroczą poprzednie najlepsze o pewną wartość.

Generator rozwiązań początkowych

Zaimplementowaliśmy następujące metody wyboru rozwiązania początkowego:

I. Losujące dowolne rozwiązanie z dozwolonego przedziału
II. Wybierające rozwiązanie z dolnej połowy dozwolonego obszaru
III. Generujące rozwiązanie z minimalnymi dozwolonymi wartościami
IV. Generujące rozwiązanie na podstawie zapotrzebowania na wino danego typu

Parametry funkcji celu

- **Ilość lat** – okres lat, przez które planujemy sadzić winogrona. Sadzenie odbywa się zawsze w marcu, lipcu i listopadzie – w pozostałych miesiącach nie liczymy.
- **Liczba pól** – ile pól uprawnych posiadamy. Każde pole ma określoną powierzchnię (wysokość x szerokość), będącą ograniczeniem górnym oraz ograniczenie dolne, ustalane przez użytkownika w tabeli.
- **Rodzaje winogron** – jakie rodzaje winogron są rozważane do zasadzenia. Każdy typ posiada swoje własne koszty sadzenia oraz sprzedaży.
- **Trójpolówka** – specjalny, opcjonalny sposób generacji macierzy jakości gleby, która opiera się o specyficzny schemat sadzenia
- **Nawóz** – decyzja podejmowana przez użytkownika – umożliwia zwiększenie jakości gleby przy dodatkowych kosztach
- **Koszt zbioru** – koszt zbioru 1 jednostki winogron
- **Pojemność magazynu –** ograniczenie górne pojemności magazynu – nie może się zmieścić więcej butelek, niż jest to założone, gdyż otrzymujemy za to karę. Nadwyżkę butelek możemy jednak odsprzedać za 40% wartości.
- **Koszt magazynowania –** opłata za magazynowanie niesprzedanych butelek
- **Ilość winogron na butelkę –** ile jednostek utworzy jedną butelkę wina
- **Koszt transportu** – koszt transportu 1 butelki wina do sklepu
- **Koszt butelkowania** – kosz zabutelkowania 1 butelki
- **Zapotrzebowanie sklepu** – lista zapotrzebowania sklepu wczytywana z tabeli. Określa zapotrzebowanie sklepu w danym miesiącu na dany typ wina. Wprowadza karę za niedotrzymanie zobowiązań wobec sklepu.

Złożoność obliczeniowa

Złożoność obliczeniowa algorytmu wynika z dwóch głównych złożoności funkcji celu i złożoności samego algorytmu. Funkcja celu ma złożoność na poziomie $O(N^3)$ ponieważ są tam 3 zagnieżdżone pętle których długość zależy po kolei od: ilości pól, ilości winogron, ilości miesięcy. Sam algorytm ma złożoność $O(N^2)$ jednak to dla stałego czasu generowania w naszym przypadku eskaluje to do $O(N^5)$ co jest i tak mniejsze od algorytmu sprawdzając ego każde rozwiązanie który ma ekspotencjalną złożoność obliczeniową.

Aplikacja

Wymagania odnośnie uruchomienia

Do uruchomienia programu niezbędne będzie środowisko obsługujące język programowania „Python” wersji 3.10. W trakcie tworzenia Aplikacji korzystano ze środowiska PyCharm Professional, które jest zalcanym środowiskiem, jednak nie koniecznym.
Do poprawnego otworzenia programu niezbędne będzie zainstalowanie następujących bibliotek:

- PyQt6
- Numpy
- Matplotlib
- Pandas
- Os
- Re
- Xlsxwriter
- winsound

Po instalacji, aby uruchomić program należy nacisnąć przycisk „ ▶ ” dla pliku „App.py”. Po jego naciśnięciu powinno nam się ukazać okno aplikacji.

Na start otrzymujemy gotowe parametry, które umożliwiają natychmiastowe uruchomienie programu dla przykładowych danych. W trybie symulacji nie jest potrzebne wczytanie żadnego pliku, problemy mogą się jednak pojawić przy nieprawidłowym użyciu trybu testowania.

Format danych i wyników

Wczytywanie z pliku zostało zaimplementowane w taki sposób, że użytkownik może wczytać pliki, podając nazwę folderu, w którym znajdują się wszystkie 4 plik (ułożone w odpowiedniej kolejności). Wczytywać można pliki w formacie .txt.

Ta opcja jest jednak dość niewygodna, dlatego zalecana jest początkowa generacja rozwiązania przykładowego dla zadanej ilości miesięcy, pól i typów winogron, a następnie jego edycja. Znacznie zmniejsza to możliwość popełnienia błędu.

Dane to w większości macierze, listy oraz zmienne typu float, bool, string i int. Te ostatnie wczytywane są z interfejsu aplikacji. Z tabel przechwytywane są dane, które wchodzą do algorytmu w postaci listy. Niektóre elementy algorytmu mogły zostać

wyłączone lub zmienione. W aplikacji można było np. zadecydować o wystąpieniu danego typu listy lub rodzaju rozwiązania początkowego czy sąsiedztwa. Warunki ograniczające również są ustalane przez użytkownika, a błędy logiki są i wskazywane użytkownikowi

Zadania testowe były w głównej mierze zadawane ręcznie przy użyciu trybu testowego i edytora plików.

Wyniki przedstawione są w zróżnicowanej formie. Najważniejsze dane zostały przedstawione w postaci wykresów, które dają nam możliwość zobrazowania, jak zachowuje się algorytm.

Wykresy znajdują się w zakładce „Wykresy" i umożliwiają zaobserwowanie zmian wartości funkcji celu w trakcie zadanego okresu czasu i iteracji. Bezpośrednia prezentacja dopuszczalności rozwiązania została pominięta, gdyż jest to widoczne w zakładce „Statystyki" w tabeli zbiorczej.

Śledzenie najważniejszych wyników oraz wystąpień kryterium aspiracji, czy wywołań listy średnioterminowej jest możliwe w zakładce „Statystyki". W zakładce tej mamy przedstawione chociażby liczbę pogorszeń/popraw funkcji celu, wykorzystane kryterium STOP, czy też porównanie najlepszego i aktualnego rozwiązania.

Złożoność obliczeniowa została przedstawiona w zakładce „Statystyki".

Wyniki można również zobaczyć w folderze „Wyniki". Wykorzystywane tabele można podejrzeć w folderze „Tabele".

Warto zaznaczyć, że w trybie testowym nie jest możliwe zobaczenie wykresów. Statystyki są również bardziej ograniczone, ze względu na możliwość przeprowadzenia dużej liczby iteracji. W tym trybie możemy zobaczyć wykresy i statystyki tylko wówczas, gdy ustawimy ilość iteracji na 1.

W trybie testowym istnieje możliwość zapisu otrzymanych rozwiązań do pliku. Przeprowadzenie dużej liczby iteracji oraz zmienianie parametrów umożliwia utworzenie zestawu danych do testów. Po zakończeniu testowania bardzo ważne jest, aby nacisnąć przycisk „S" – w innym wypadku wyniki przepadną.

Funkcjonalność

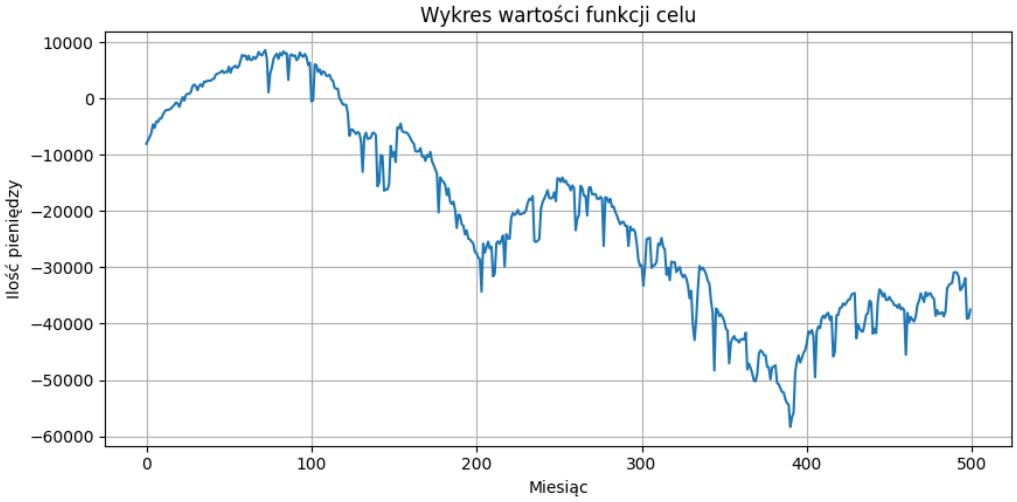
- Wprowadzenie parametrów funkcji celu
- Wprowadzenie parametrów algorytmu
- Możliwość podglądu plików
- Możliwość edycji plików
- Możliwość monitorowania zmian i przebiegu algorytmu
- Generacja wykresów do późniejszego wykorzystania
- Zapis wyników do pliku

Testy

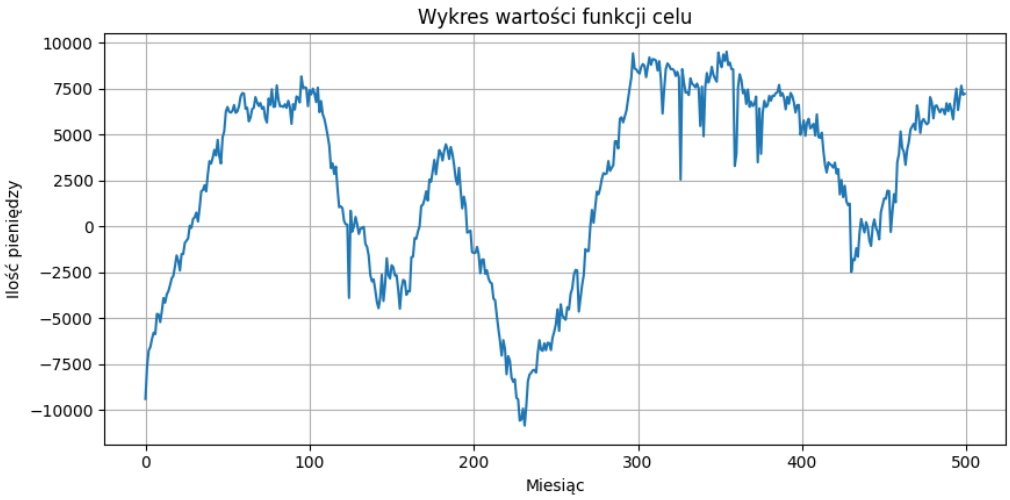
Wpływ wielkości listy Tabu – badanie długości listy Tabu i jej wpływu na rozwiązanie (Julia/Filip)

Wartości testowe (nieuwzględnione są domyślne):

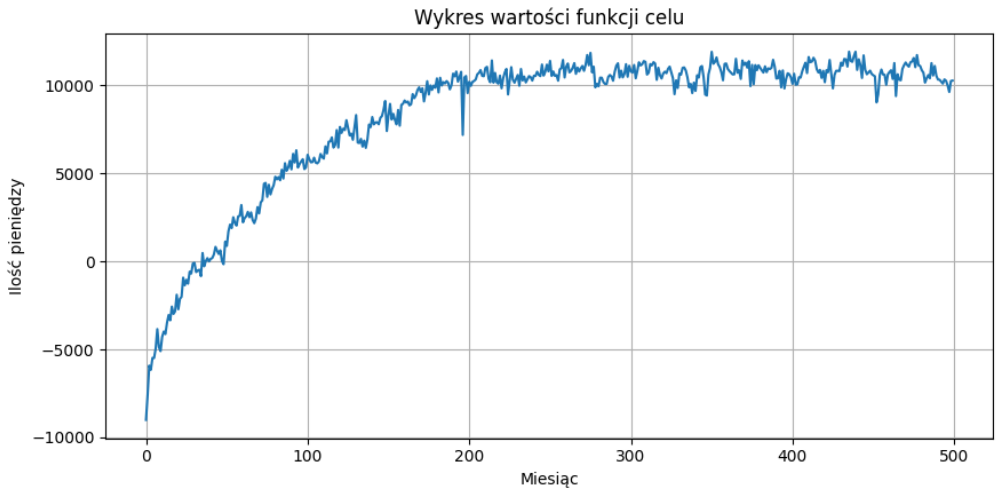
- Iteracje: 300
- Powtórzenia 50
- Lista krótkoterminowa: 5/25/50
- Krok: stały-20
- Wartość predefiniowana: 1000000
- Rozwiązanie początkowe: Typ IV
- Sąsiedztwo: 0.3



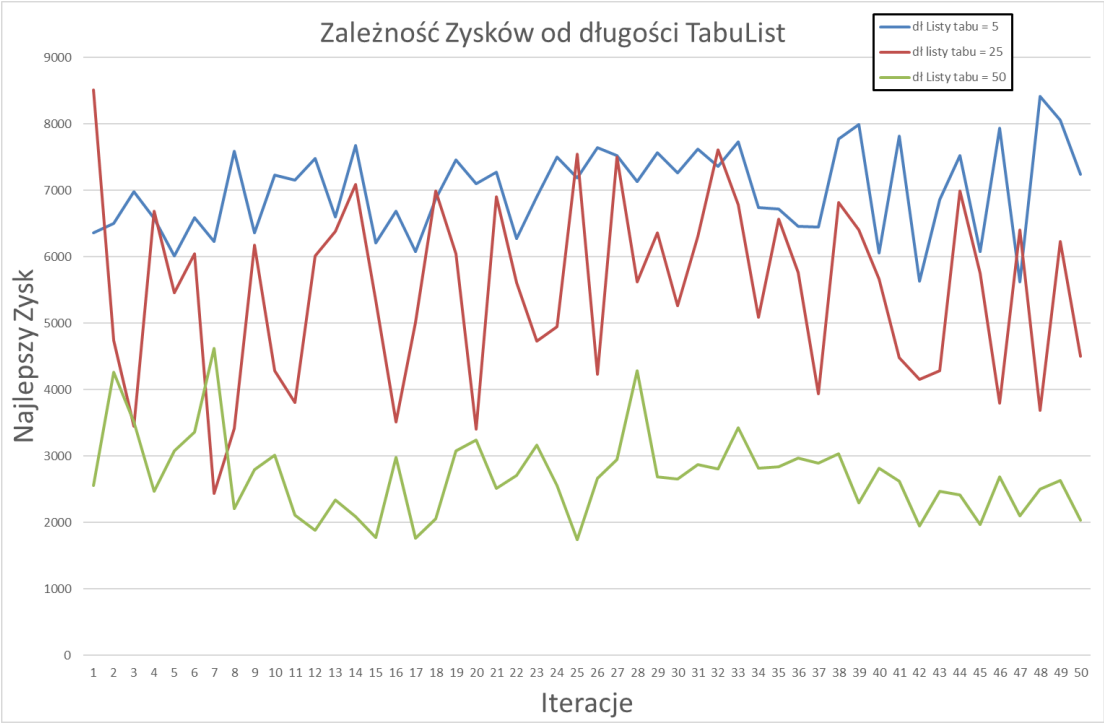
Rys. 1 Wykres Tabu50



Rys. 2 Wykres Tabu25



Rys.3 Wykres Tabu5

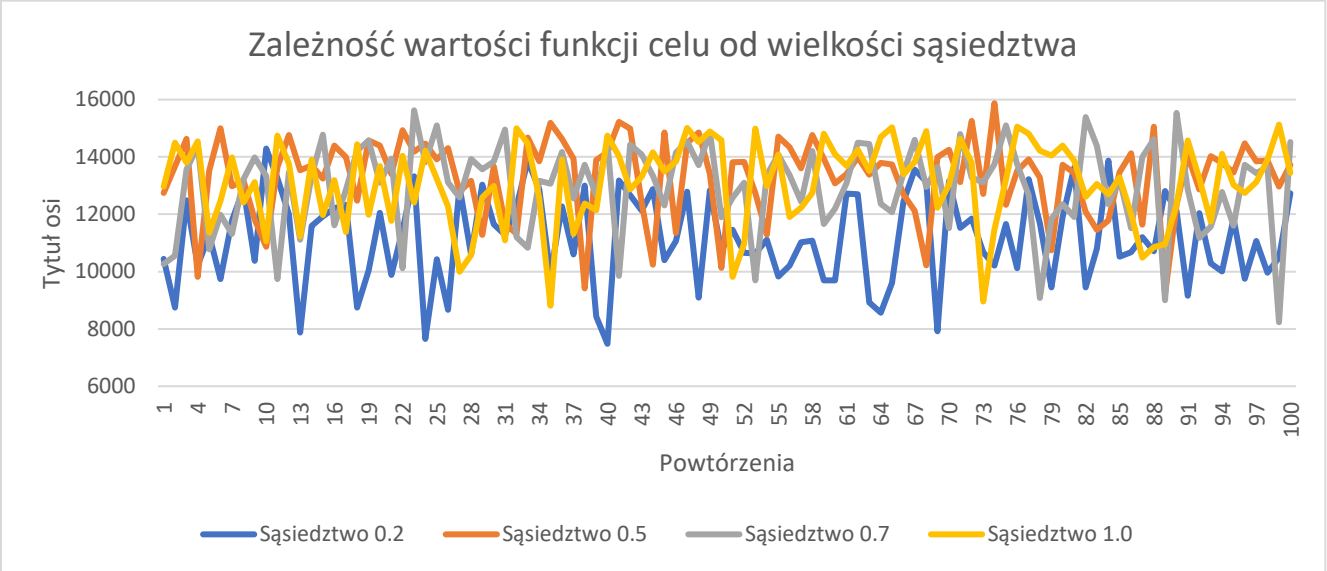


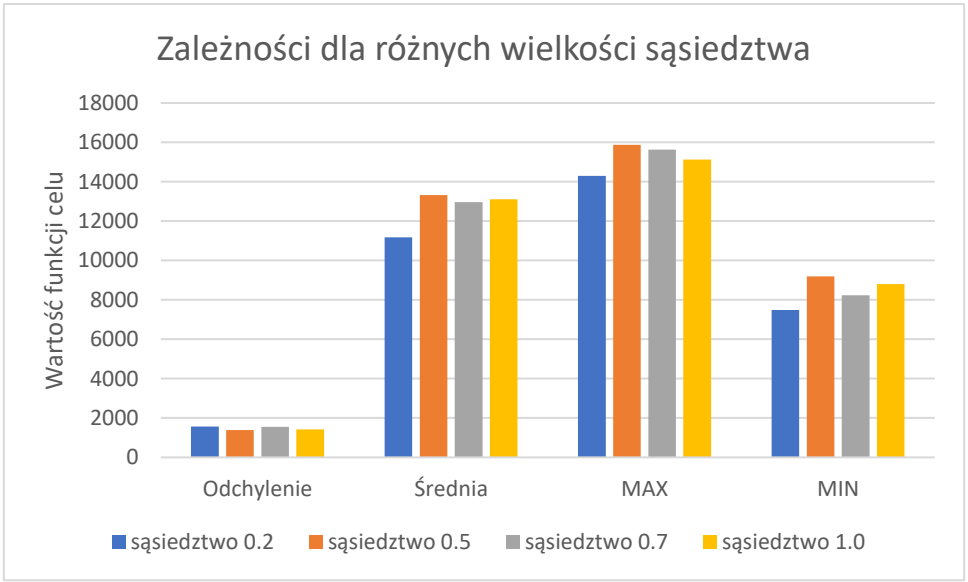
Obserwując wykresy można zauważyć, że im dłuższa Tabu Lista, tym zazwyczaj gorsze rozwiązania. Wynika to z tego, że przy dłuższej liście Tabu i niedostatecznym kryterium aspiracji, funkcja może „Utknąć” w miejscu. Dla Tabu Listy skrajności również nie są dobre – przy zbyt krótkiej liście może dojść do krążenia wokół jednego rozwiązania, dla za długiej – zbytniego przyblokowania. Najlepiej wybierać wartość pośrednią, która umożliwia dużą dywersyfikację rozwiązań i szansę na znalezienie lepszych rozwiązań, niż przy zbyt krótkiej i zbyt długiej liście.

Wpływ wielkości sąsiedztwa na rozwiązanie – sprawdzenie, jak wielkość sąsiedztwa przekłada się na jakość rozwiązania (Julia)

Wartości testowe (nieuwzględnione są domyślne):

- Iteracje: 300
- Powtórzenia 100
- Lista krótkoterminowa: 5/25/50
- Krok: stały-20
- Wartość predefiniowana: 1000000
- Rozwiązanie początkowe: Typ IV
- Sąsiedztwo: 0.2/0.5/0/7/1.0



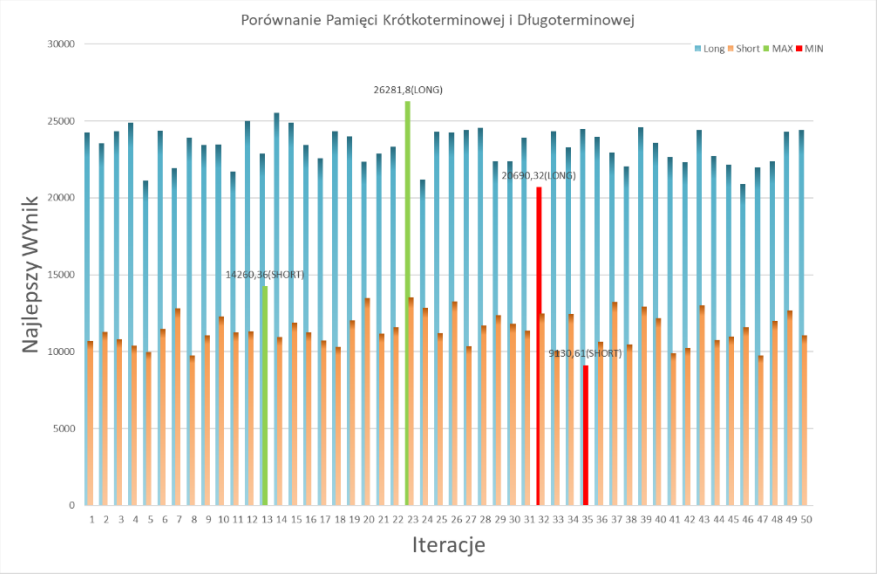


Można zaobserwować, że najlepiej sprawdza się sąsiedztwo 0.5. Wykonanie go nie zajmuje tak dużo czasu, jak wykonanie sąsiedztwa 1.0. W teorii, sąsiedztwo 1.0 ma większe prawdopodobieństwo dostarczeni najlepszego rozwiązania, jednak ze względu na ograniczenie w postaci liczby iteracji i mechanizmu zabronień – większości rozwiązań możemy wówczas nie odwiedzić.

Wpływ pamięci długoterminowej i średnioterminowej - (Filip)

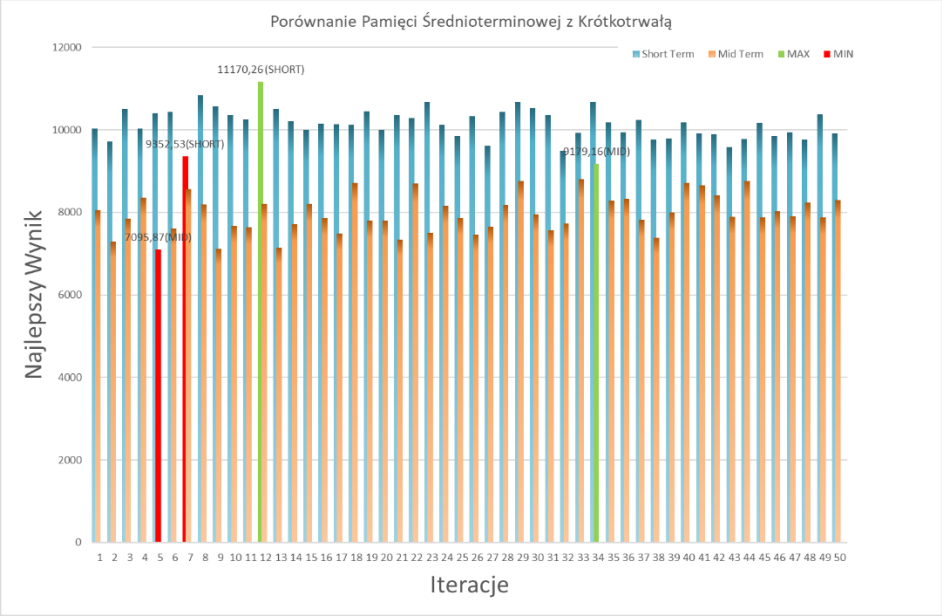
Wartości testowe (nieuwzględnione są domyślne):

- Iteracje: 400
- Powtórzenia 50
- Krok: stały-20
- Ilość lat: 2
- Sąsiedztwo: 0.2/0.5/0/7/1.0
- Lista średnia i krótka/ Długa i krótka



Odchylenie średniej dla pamięci krótkoterminowej: 942,389072

Odchylenie średniej dla pamięci długoterminowej: 1039,654768



Odchylenie średniej dla pamięci krótkoterminowej: 289,6862

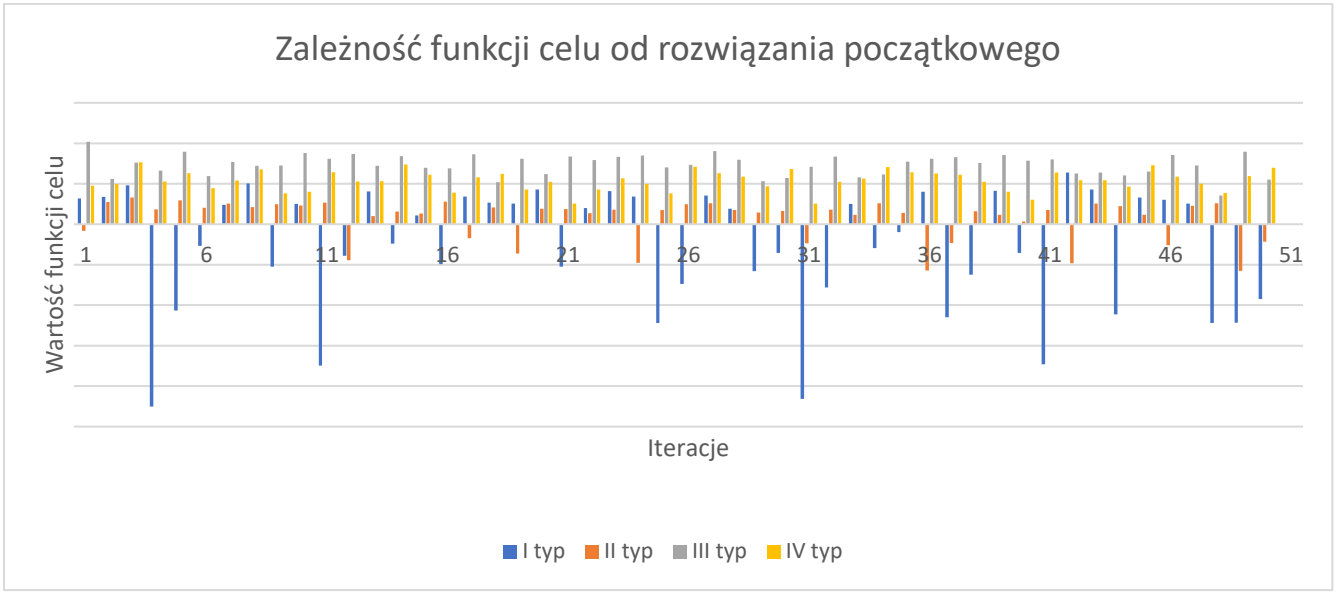
Odchylenie średniej dla pamięci średnioterminowej: 397,095784

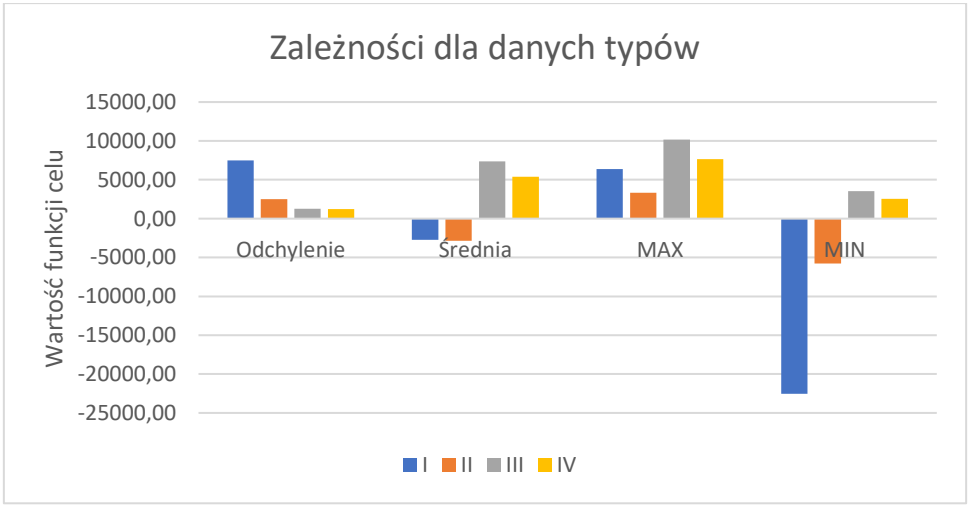
Wpływ rozwiązania początkowego – jak wybór rozwiązania początkowego przekłada się na rozwiązanie końcowe (Julia)

Wartości testowe (nieuwzględnione są domyślne):

- Iteracje: 300
- Powtórzenia 50
- Lista krótkoterminowa: 15
- Krok: stały-10
- Wartość predefiniowana: 1000000
- Rozwiązanie początkowe: Typ I/II/III/Iv
- Sąsiedztwo: 1.0

UWAGA – do tego wykresu nie utworzono plików, ze względu na generację nowych rozwiązań.



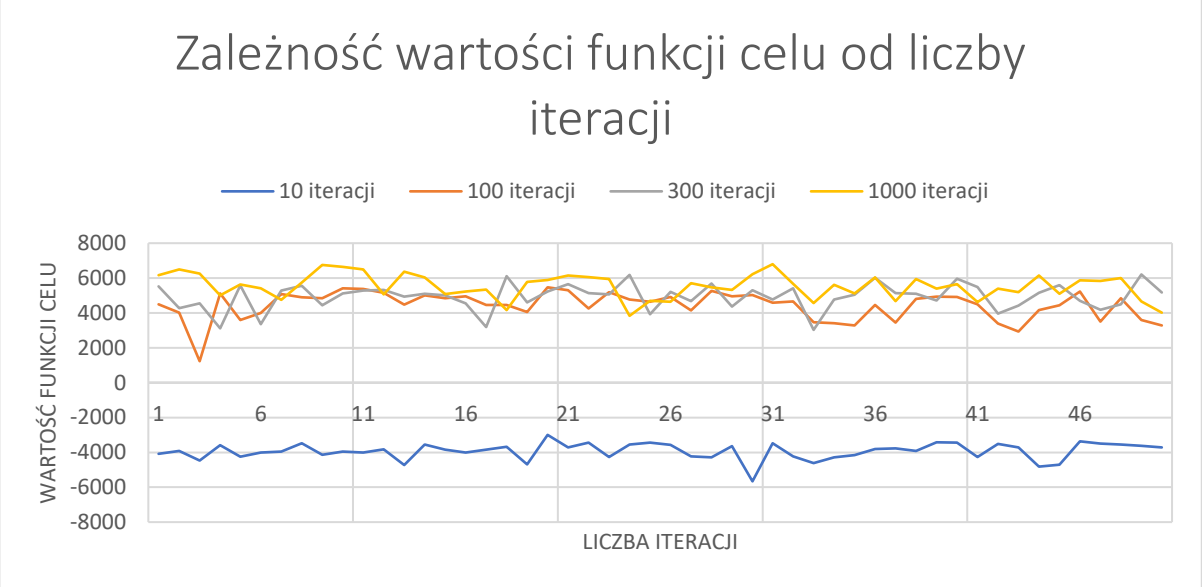


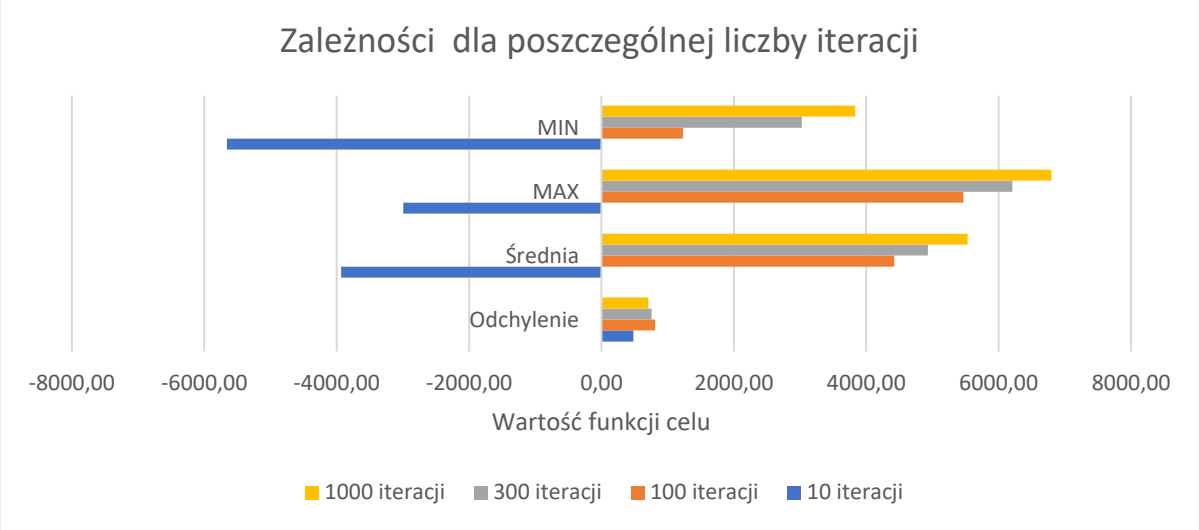
Widzimy, że najlepsze wartości uzyskano dla typu III i IV – były to typy bazujące na ograniczeniu dolnym i zapotrzebowaniu sklepu. Najgorsze okazało się losowo wygenerowane rozwiązanie typu I. Łatwo więc zauważyć, jak dużą rolę odgrywa zapotrzebowanie w naszym modelu.

Wpływ ilości iteracji – sprawdzenie, jak na rozwiązanie wpływa liczba iteracji (Julia)

Wartości testowe (nieuwzględnione są domyślne):

- Iteracje: 10/100/300/1000 na 50 powtórzeń
- Lista krótkoterminowa: 10
- Krok: stały-10
- Wartość predefiniowana: 1000000
- Rozwiązanie początkowe: Typ III
- Sąsiedztwo: 1.0





Test był jednym z podstawowych testów, które umożliwiły nam właściwe oszacowanie niezbędnej ilości iteracji do przeprowadzenia sensownych testów. Wartości niewielkie dawały bardzo złe wyniki, dlatego zazwyczaj skupialiśmy się na iteracjach 100-1000. Za duża liczba iteracji zajmowała bardzo dużo czasu, więc musieliśmy ograniczyć się do mniejszych liczb iteracji i powtórzeń.

Kryteria STOP – badanie funkcjonowania kryteriów (Julia)

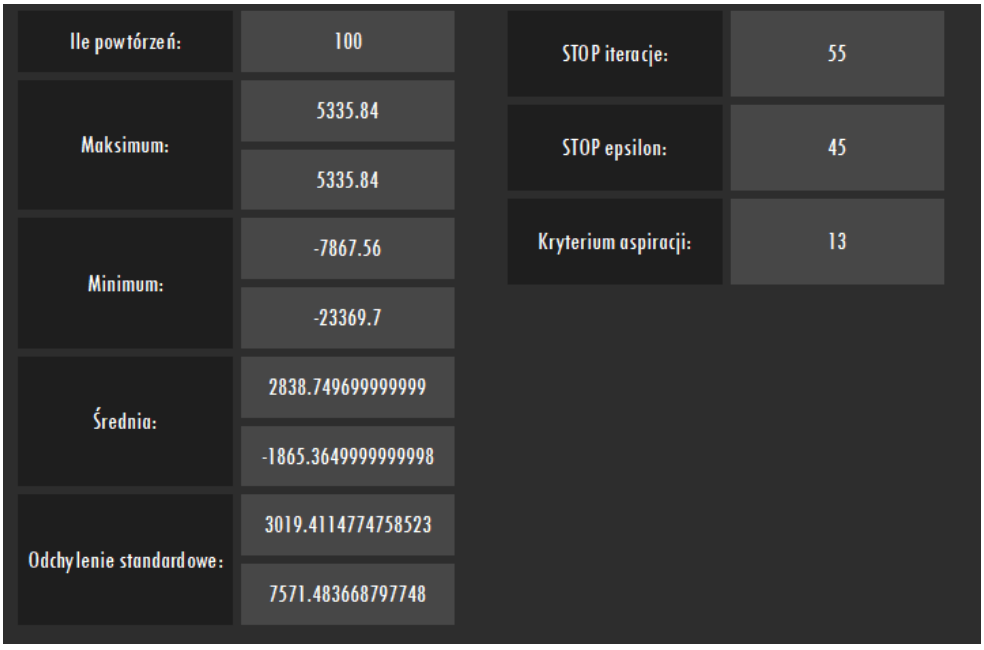
Wartości testowe (nieuwzględnione są domyślne):

- Epsilon: 0.1
- Iteracje: 300
- Ilość powtórzeń: 100
- Krok: stały-20
- Wartość predefiniowana: 5000
- Rozwiązanie początkowe: Typ I
- Sąsiedztwo 0.2

Dla przeprowadzenia tego testu przydatniejsza okazała się zakładka statystyk, która zawierała informacje zbiorcze, dotyczące wartości statystycznych oraz ilości wykorzystanych kryteriów STOP: kryterium iteracji i kryterium epsilon.

Działanie obu kryteriów zostało potwierdzono, jednak przy kryterium epsilon ważne jest odpowiednie ustawienie wartości epsilon oraz wartości predefiniowanej. Nie zawsze bowiem udaje się osiągnąć epsilon, który jest rzędu 10^{-5}.

Kryterium dokładności jest ważne, gdyż z niektórych przyczyn możemy chcieć, aby obliczanie algorytmu zakończyło się jak najbliżej zadanej dokładności, a nie w zależności od iteracji.



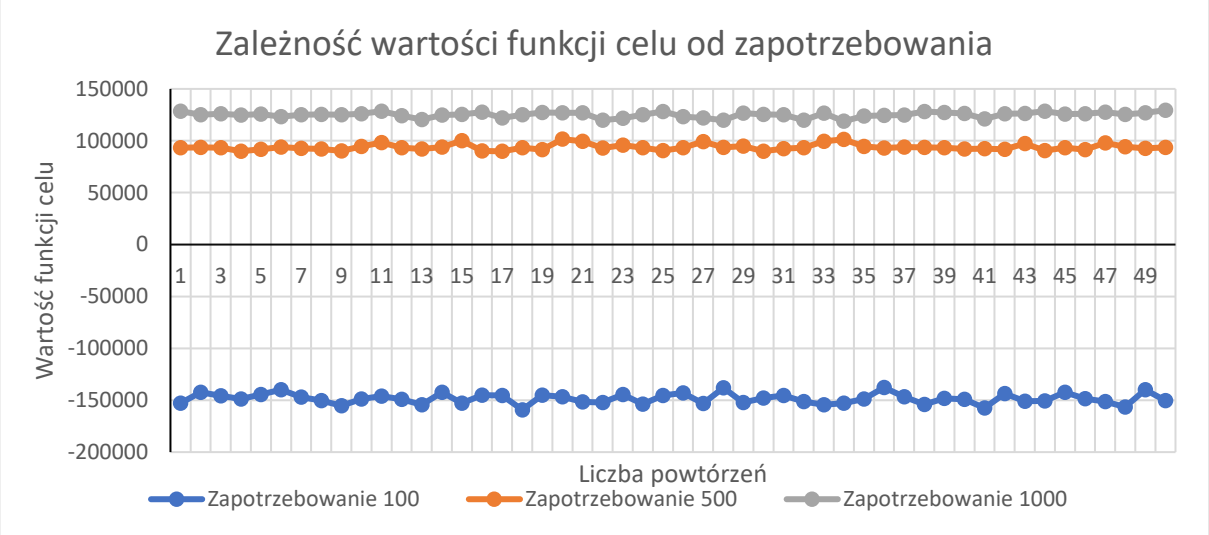
W statystykach przedstawiono wartości dla rozwiązania najlepszego (góra) oraz aktualnego (dół).

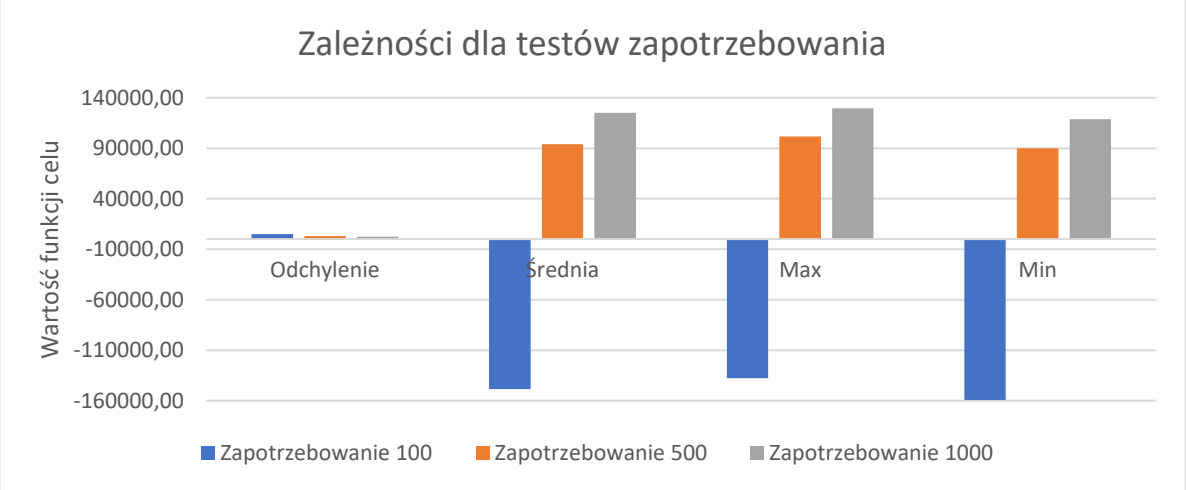
Przeprowadzony test uzasadnił także, że kryterium aspiracji również jest wykorzystywane przez algorytm.

Wpływ zapotrzebowania/ograniczeń – wpływ działania ograniczeń (Julia)

Wartości testowe (nieuwzględnione są domyślne):

- Ograniczenie górne dla 1 pola: 1000 (100x10)
- Ograniczenie dolne dla 1 pola: 500
  (dla wszystkich ograniczenia takie same)
- 4 pola
- Pojemność magazynu: 2000
- Zapotrzebowanie: dla każdego z 3 typów 1000
- Krok: stały-20
- Sąsiedztwo: 0.5
- 300 iteracji/ 50 powtórzeń



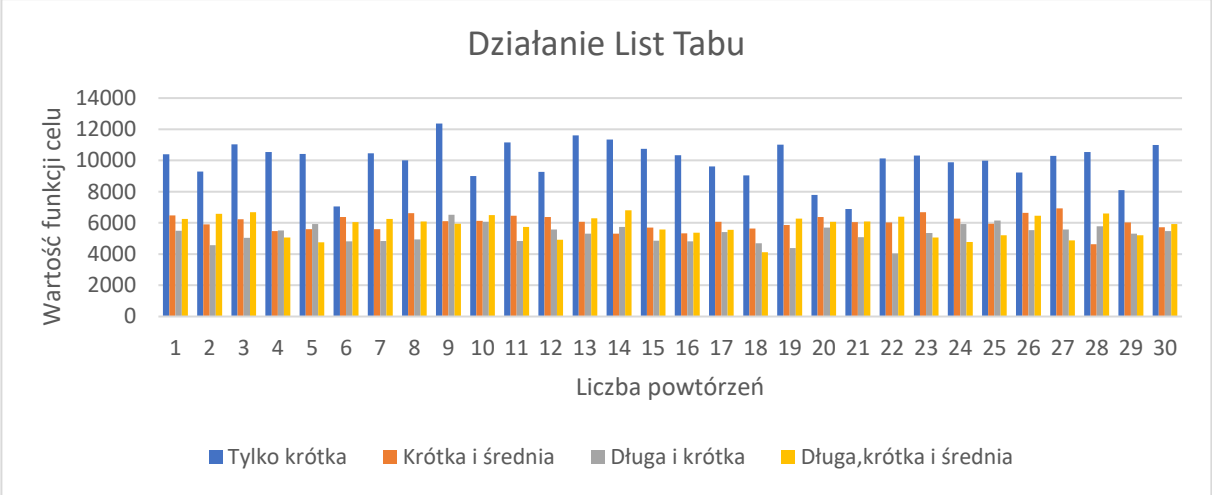


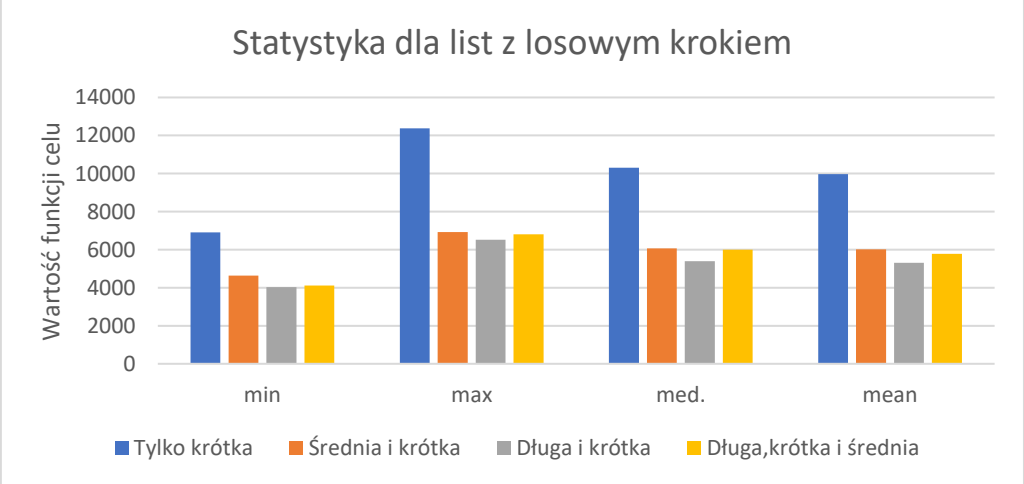
Zależność poprawnego działania algorytmu jest silnie oparta o zapotrzebowanie sklepu. Jeśli jest ono małe, to butelki wina się marnują – zostają w większości sprzedane po niekorzystnych cenach, a magazyn się przepełnia. Im większe zapotrzebowanie, tym większa szansa, na lepszy zysk. Trzeba mieć jednak na uwadze, że zbyt duża wartość zapotrzebowania zacznie przynosić straty.

Wpływ kroku losowego na pamięci - Paweł

Wartości testowe (nieuwzględnione są domyślne):

- Iteracje: 500
- Powtórzenia 30
- Lista krótkoterminowa:
- Lista średnioterminowa: 5
- Lista długoterminowa: 50
- Krok: stały-20
- Brak kryterium aspiracji
- Rozwiązanie początkowe: Typ I
- Sąsiedztwo: 0.5





Pamięci długo i średnioterminowe dywersyfikują przeglądaną przestrzeń rozwiązań co pozwala uniknąć zatrzymywania się w minimach lokalnych i szerszą eksplorację rozwiązań jednak powoduje to że może w niektórych przypadkach wyjść z optimum globalnego co można zaobserwować na załączonych powyżej danych statystycznych jednak dla bardziej złożonych funkcji celu z wieloma minimami lokalnymi pozwala to na uzyskanie o wiele lepszych wyników.

Testy Generatorów – testy jednostkowe, jednak ważne do działania algorytmu

Test_soil_quality_Trojpolowka:

Test generuje mnożnik nawozu dla każdego pola bez uwzględnienia mnożnika Trójpolówki

Użyte warunki dla testów:

- Poprawna struktura tablicy trójwymiarowej
- czy podczas generacji nie powstał żaden element nie będący przewidzianym typem liczbowym
- Czy ceny są zgodne z zakresem generatora nawozu z uwzględnieniem

Test_soil_quality_without_Trojpolowka:

Test generuje mnożnik nawozu dla każdego pola z uwzględnieniem mnożnika Trójpolówki

Użyte warunki dla testów:

- Poprawna struktura tablicy trójwymiarowej
- czy podczas generacji nie powstał żaden element nie będący przewidzianym typem liczbowym
- Czy ceny są zgodne z zakresem generatora nawozu z uwzględnieniem mnożnika trójpolówki

Test_plant_price:

Test sprawdza dla różnych przypadków wygenerowanie losowych cen z nałożonego przedziału

Użyte warunki dla testów:

- Poprawna struktura tablicy
- czy podczas generacji nie powstał żaden element nie będący przewidzianym typem liczbowym
- Czy ceny są zgodne z zakresem generatora cen

Test_generate_Solution:

Test sprawdzał jak dla różnych przypadków danych, generator będzie zwracał macierze zawierające obsiane pola oraz o jakich typach winogron. Warto zaznaczyć tu warunek, iż na każdym polu może być posiany tylko jeden rodzaj winogron

Użyte warunki dla testów:

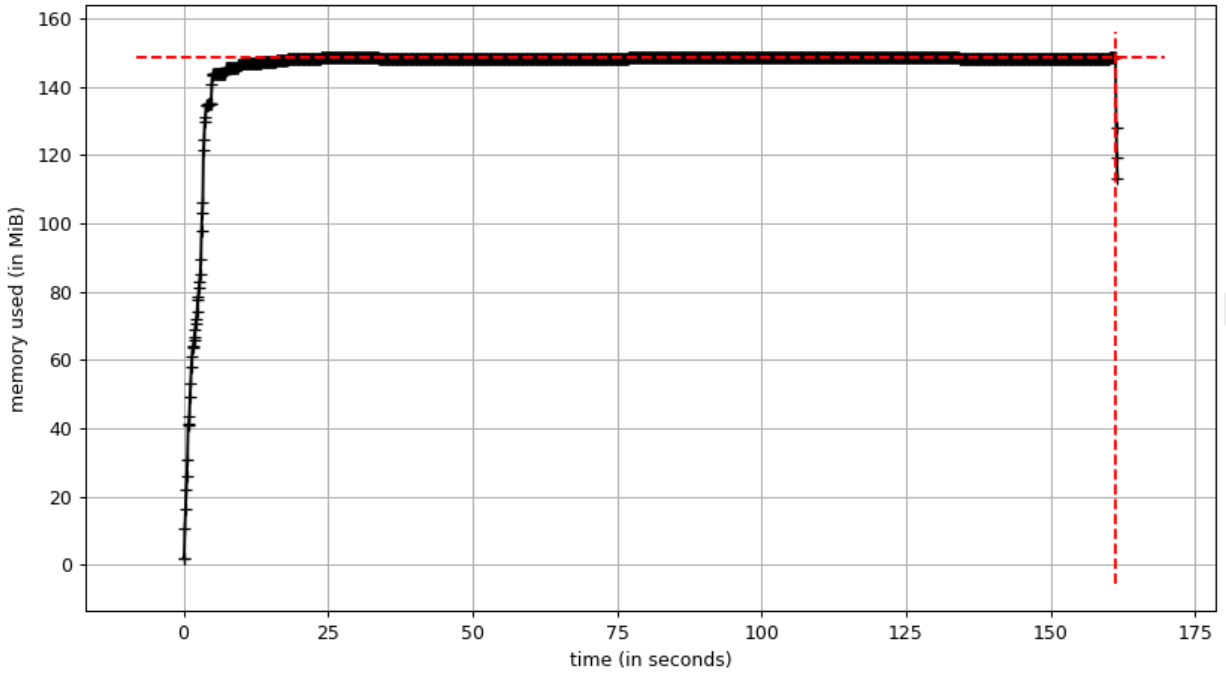
- Poprawna struktura tablicy trójwymiarowej
- czy podczas generacji nie powstał żaden element nie będący przewidzianym typem liczbowym
- Czy ilość posadzonych roślin zgadza się z pojemnością odpowiedniego pola i czy nazwy winogron zgadzają się z bazę

Testy – słowo końcowe

Warto przede wszystkim wspomnieć, że wiele testów wykonywanych w trakcie projektowania aplikacji zostało zawartych w odpowiednim panelu na interfejsie. Warto również zagłębić się w pliki testowe, które dadzą większe spojrzenie na poprawność implementacji algorytmu, funkcji celu, jak i też samej aplikacji.

W zakładce „Statystki" znajdują się obliczenia dotyczące czasu wykonania danych elementów algorytmu Tabu Search, a także zaimplementowana została możliwość monitorowania rozwiązań.

Złożoność obliczeniową przedstawiamy natomiast poniżej:



Rys. 4 Złożoność pamięciowa aplikacji

Wynika z tego benchmarku, że główne zużycie pamięci przez aplikacje wynika z użycia GUI natomiast zużycie powodowane przez sam algorytm jest nieistotną częścią.

Podsumowanie

Wnioski

Pisanie aplikacji umożliwiło nam dogłębne zapoznanie się z algorytmem Tabu Search oraz wszelkimi strukturami, które są z nim związane. Pisanie aplikacji opartej na naszym własnym modelu, wykorzystującej algorytm heurystyczny do maksymalizacji okazało się zadaniem ciężkim i czasochłonnym, ale także bardzo ciekawym.

Wykorzystanie algorytmu umożliwiło nam zapoznanie się z praktycznym zastosowaniem Tabu Search. Uświadomiliśmy sobie, że algorytm ten nie jest taki jednoznaczny, jak początkowo mogłoby się wydawać. Posiada on wiele możliwości dostosowania pod rozważany problem.

Tworzenie projektu było również świetną okazją do zapoznania się z biblioteką PyQt6 oraz aplikacją QtCreator, które służą do tworzenia interfejsu aplikacji. Początkowo, tworzenie interfejsu było ciężkie, jednak z czasem, po zgłębieniu dokumentacji, okazało się przyjemnym zajęciem.

Stwierdzone problemy

- Aplikacja działa wolno i czasami wyświetla informację o tym, że okno „Nie odpowiada" – wymaga to odpowiedniej implementacji QThread, która zapobiegnie zawieszaniu się okna (Program działa w tle)
- Wczytywanie z pliku potrafi być uciążliwe, należałoby to zoptymalizować
- Jeżeli mamy skrajnie małe sąsiedztwo, istnieje ryzyko, że nie otrzymamy prawidłowego rozwiązania
- Niektóre błędne wejścia nie są opisane – należałoby utworzyć więcej przypadków
- Czasami oscylacje mogą nie być dobrze widoczne – należy wówczas odpowiednio dostosować parametry algorytmu

Kierunki dalszego rozwoju

Aplikacje można rozwijać pod kątem optymalizacji bilansu zysków i strat dla różnych przedsiębiorstw – nie tylko tych zajmujących się rolnictwem.

Optymalizacja dla rzeczywistych przedsiębiorstw musiałaby zostać dostosowana pod kątem generatorów, gdyż mają nam one jedynie symulować wartości rzeczywiste. Niektóre z parametrów musiałyby zostać pominięte, gdyż w rzeczywistości byłyby ciężkie do sprawdzenia.

Na pewno warto urozmaicić i rozwinąć funkcję celu, a także bardziej usprawnić interfejs aplikacji, aby był bardziej przyjazny użytkownikowi. Napisanie instrukcji do programu również byłoby wskazane.