AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Projekt dyplomowy

Modelowanie i optymalizacja produkcji rolnej
Modeling and Optimization of Agricultural Production

Autor: Piotr Paweł Hudaszek
Kierunek studiów: Automatyka i Robotyka
Opiekun pracy: dr inż. Piotr Kadłuczka

Kraków, 2022

Spis treści

Wstęp

Rolnictwo jest ważną dziedziną gospodarki, więc naturalnym jest wspieranie procesu uprawy przez aplikacje komputerowe mające na celu poprawę efektywności produkcji rolnej. Dzięki rozpowszechnieniu technologii takich jak między innymi GPS, zdjęcia lotnicze i czujniki wilgotności gleby, można uzyskać dokładne dane do różnego rodzaju modelów rolniczych. Rozważany w tej pracy problem dotyczy planowania procesu uprawy.

Celem pracy jest stworzenie programu który ma za zadanie zoptymalizować terminarz upraw na określonej liczbie pól tak aby maksymalizować zysk przy ograniczeniu dostępnych zasobów. Został on napisany w Pythonie jako że jest to jeden z bardziej popularnych języków programowania który wciąż się rozwija i zyskuje popularność [2]. Jako algorytm optymalizacji wybrano algorytm genetyczny.

W pierwszym rozdziale został opisany model matematyczny wraz z omówieniem użytych uproszczeń i przydatności modelu w realnych zastosowaniach. Rozdział drugi jest poświęcony adaptacji algorytmu genetycznego do rozważanego problemu w tym zdefiniowanie postaci rozwiązania, miary przystosowania oraz opisanie użytych operatorów genetycznych. W dalszej części pracy zostały zaprezentowane rozwiązania implementacyjne oraz przeprowadzone testy.

1 Opis zagadnienia

1.1 Model opisowy

Opisywane zagadnienie polega na optymalizacji zysku z uprawy w przedsiębiorstwie rolnym. Jest możliwość uprawy różnych rodzajów produktów. Każdy rodzaj posiada konieczne do przeprowadzenia zabiegi agrotechniczne na danym etapie uprawy do których potrzebuje określone zasoby. Ponadto uprawę można rozpocząć tylko w określonym dla danego typu uprawy przedziale czasu. Przedsiębiorca ma dostęp do pewnej ilości każdego rodzaju zasobów. Zasoby dzielą się na dwie klasy: zasoby dzienne które są przydzielone na dany dzień oraz zasoby całkowite przydzielone na cały rozpatrywany okres uprawy.

Jeśli uprawa zostanie przeprowadzona i zasobów nie zabraknie to przedsiębiorca uzyska określony bazowy zysk z jednostki pola, który zależny jest od rodzaju produktu. Rozpatrywane przedsiębiorstwo ma określoną liczbę pól. Każde pole można opisać jego obszarem oraz współczynnikami które określają jak dobrze dane pole pasuje do uprawy danego rodzaju produktu. Aby uzyskać rzeczywisty zysk należy przemnożyć zysk bazowy z współczynnikiem dopasowania oraz wielkością pola.

Optymalizacja polega na wyborze jaki produkt jest uprawiany na danym polu oraz kiedy rozstanie rozpoczęta uprawa. Na polu można uprawiać kilka produktów po sobie. Data rozpoczęcia uprawy jest ograniczona przez ramy czasowe danego produktu.

1.2 Model matematyczny

Model matematyczny w ścisły sposób definiuje model zagadnienia na podstawie modelu opisowego.

Dane:

T	-	długość rozpatrywanego sezonu uprawy
$t = 1, \dots, \mathrm{T}$	-	indeks dnia
$p \in P$	-	indeks pola dostępnego do uprawy
S_p	-	powierzchnia pola p
$n = 1, \dots, N_p$	-	indeks rozpoczętej uprawy na polu p
y_{pn}	-	rodzaj n-tej rozpoczętej uprawy na polu p

x_{pn} - dzień rozpoczęcia n-tej uprawy na polu p

d_y - dochód z jednostki pola na którym był uprawiany produkt y

w_{py} - współczynnik dopasowania pola p do produktu y

$z_d \in Z_d$ - dostępny zasób dzienny (odnawialny po każdym dniu)

$z_c \in Z_c$ - dostępny zasób całkowity (przydzielony na cały okres uprawy)

z_{dyt} - zasoby dzienne potrzebne do uprawy produktu y w t dniu uprawy

z_{cyt} - zasoby całkowite potrzebne do uprawy produktu y w t dniu uprawy

t_y - czas rozpoczęcia uprawy produktu

t_{y0} - pierwszy dzień w którym można zacząć uprawę produktu y

t_{yN} - ostatni dzień w którym można zacząć uprawę produktu y

o_y - czas trwania uprawy produktu y

Postać rozwiązania:

Lista dwu elementowych wektorów dla każdego pola gdzie pierwszy element x oznacza dzień rozpoczęcia a drugi y rodzaj uprawy.

$$\begin{Bmatrix} (x_{11}, y_{11}) & (x_{12}, y_{12}) & \cdots & (x_{1N_1}, y_{1N_1}) \\ (x_{21}, y_{21}) & (x_{22}, y_{22}) & \cdots & (x_{2N_2}, y_{2N_2}) \\ \vdots & \vdots & \ddots & \vdots \\ (x_{P1}, y_{P1}) & (x_{P2}, y_{P2}) & \cdots & (x_{PN_P}, y_{PN_P}) \end{Bmatrix}$$

Funkcja celu:

Zysk ze sprzedaży uzyskanych produktów .

$$f(x_{pn}, y_{pn}) = \sum_{p \in P} \sum_{n \in N_p} S_p d_{y_{pn}} w_{py_{pn}} \rightarrow \max$$

Ograniczenia:

Dla każdego dnia t oraz każdego rodzaju zasobu dziennego ilość użyta nie może przekroczyć dostępnej.

$$\forall z_d \in Z_d \quad \forall t \in T \quad \sum_{p \in P} z_{dy_p(t-x_p)} \leq z_d$$

Dla każdego rodzaju zasobu całkowitego ilość użyta przez cały okres uprawy nie może przekroczyć dostępnej.

$$\forall z_c \in Z_c \quad \sum_{t \in T} \sum_{p \in P} z_{cy_p(t-x_p)} \leq z_c$$

Uprawę produktu typu y można zacząć w wyznaczonym oknie czasowym.

$$t_{y0} \leq t_y \leq t_{yN}$$

W danym dniu na jednym polu można uprawiać tylko jeden rodzaj produktu.

$$\forall p \in P \; \forall n \in N_p \quad x_{pn} + o_y < x_{pn+1}$$

1.3 Model a rzeczywisty problem

Tak sformułowany model pozwala oddać typowe zależności podczas planowania uprawy tak jak ograniczenie terminu rozpoczęcia uprawy i konieczne zasoby. Ilość rodzajów zasobów jest konfigurowalna, co pozwala to użytkownikowi programu wprowadzić zasoby specyficzne dla danego typu uprawy i sprawić że program jest bardziej uniwersalny. Model można też zastosować do innych zagadnień niż rolnicze na przykład do bilansowania zasobów w zakładzie produkcyjnym.

Model nie bierze pod uwagę wpływu nieprzewidywalnych zmian cen oraz pogody. Natomiast użytkownik może wprowadzić odpowiednio większe potrzebne ilości zasobów lub dłuższy czas uprawy aby mieć pewien margines bezpieczeństwa.

Mimo tych ograniczeń uważam że model może być przydatny. Szczególnie w uprawie szklarniowej gdzie wpływ pogody jest ograniczony oraz w przypadku gdy optymalne wykorzystanie któregoś z zasobów jest kluczowe. Tak jest na przykład w przypadku rejonów z ograniczoną ilością wody [6].

1.4 Przegląd podobnych modeli w literaturze

Przedstawiony model nie jest oderwany od rozważanych problemów w literaturze. Poniżej opisano kilka interesujących przykładów.

Jednym z pierwszych opisanych przykładów zastosowań badań operacyjnych w rolnictwie jest praca Doktora C.W. Thornthwaite w latach 1946-50. Dotyczyła ona uprawy grochu na 7000 arowej farmie. Problemy, z jakimi borykała się farma to:

1. nie było naukowej metody na znalezienie odpowiedniego czasu na zbiory,
2. cała uprawa dojrzewała na raz, w związku z czym stało się niemożliwe nadążanie za dojrzewającym grochem nawet przy użyciu wszystkich maszyn i siły roboczej.

Pierwszy problem został rozwiązany za pomocą badania wpływów czynników pogodowych na proces dojrzewania groszku. Natomiast drugi został rozwiązany za pomocą terminarza upraw, dzięki któremu cały groch nie dojrzewał w tym samym czasie [12]. W mojej pracy nie poruszono problemu wyznaczania parametrów różnych rodzajów upraw i procesu ich uprawy, ale umożliwiono ich wygodne wprowadzenie do programu.

Bardziej skomplikowany problem przedstawia praca mająca na celu optymalizację planu upraw i hodowli w regionie Chang Qing w Chinach. Ważnym założeniem w pracy było nie wywieranie zbyt negatywnego efektu na środowisko naturalne oraz odporność planu na ekstremalne warunki pogodowe (susza lub powódź). Wykorzystano do tego celu teorię gier oraz programowanie liniowe. Uzyskane wyniki zastosowano w latach 1983-1985 i na ich skutek zysk z upraw wzrósł o 12,33%, a z hodowli o 53,77% [15].

Praca [14] dotyczy rejonu Irkutska w Rosji, w niej również zwrócono uwagę na wpływ ekstremalnych warunków pogodowych. Porównano w niej metodę programowania liniowego oraz stochastycznego programowania liniowego. Metoda stochastyczna biorąca pod uwagę ryzyko wystąpienia niepożądanych warunków pogodowych pozwoliła lepiej zaalokować dostępne zasoby. Zwrócono też uwagę na wykorzystanie technologii rolnictwa precyzyjnego [11] które pozwala gromadzić dane o przestrzennym zróżnicowaniu pola i stanu uprawy. Dzięki tym danym można między innymi wybiórczo stosować zabiegi nawożenia, co sprawia że zasoby są lepiej wykorzystywane.

Ciekawym problemem opisanym w pracy [13] jest połączenie uprawy rolnej ze zbiorem dziko rosnących dóbr. Takie podejście według autorów może mieć zastosowanie w wielu rzadziej zaludnionych rejonach.

Nie można zapomnieć o tym aby prowadzone zabiegi optymalizacyjne nie odbywały się kosztem środowiska naturalnego ani niszczenia gleby. Jednak ciężko jest te warunki uwzględnić w modelu nie posiadając bogatej wiedzy na temat rolnictwa i ekologii. Dlatego w przedstawionym programie udostępniono użytkownikowi narzędzie pozwalające w pewnym stopniu ograniczyć negatywny wpływ na środowisko. Tym narzędziem jest wprowadzanie ograniczeń. Można na przykład wprowadzić limit dzienny na zużycie nawozów lub środków ochrony roślin. Można też wprowadzić jak bardzo każdy rodzaj uprawy wyjaławia glebę i ograniczyć dopuszczalne wyjałowienie gleby.

2 Opracowanie algorytmu ewolucyjnego

2.1 Zasada działania algorytmów ewolucyjnych

Algorytmy ewolucyjne są oparte na biologicznej ewolucji sformułowanej po raz pierwszy przez Charles'a Darwina. Jego teoria tłumaczy adaptacyjne zmiany w osobnikach poprzez naturalną selekcję, która sprzyja przetrwaniu najlepiej przystosowanych do warunków środowiskowych osobników. Poza selekcją kolejnym ważnym czynnikiem rozpoznanym przez Darwina jest występowanie małych, losowych i nie ukierunkowanych zmian pomiędzy fenotypami (zespół cech organizmu). Te mutacje przeżywają w populacji jeżeli okazują się przydatne w aktualnym środowisku, w przeciwnym wypadku wymierają [3].

Siłą napędową naturalnej selekcji jest produkcja potomków, co w efekcie sprawia że wielkość populacji rośnie. Jest ona jednak ograniczona poprzez skończone zasoby dostępne dla populacji. To sprawia że osobniki lepiej wykorzystujące zasoby mają przewagę.

Jeśli zamiast osobników użyjemy rozwiązań problemu który chcemy zoptymalizować, oraz odpowiednio zdefiniujemy reguły mutacji, selekcji oraz tworzenia potomków to można naśladując naturę zbudować algorytm optymalizacji. Warto jednak zwrócić uwagę na to że proces ewolucji zachodzącej w naturze nie ma na celu optymalizacji. Jest to automatyczny proces który się cały czas toczy, ale nie dąży do niczego[10]. Optymalizacja jest efektem pobocznym ewolucji naturalnej. Dlatego natura jest inspiracją, a nie ścisłym zbiorem reguł dla algorytmów ewolucyjnych.

2.2 Schemat algorytmu

Programy ewolucyjne są zbudowane według poniższego ogólnego schematu [9] gdzie P(t) oznacza populację w iteracji t.

```
procedure program ewolucyjny
begin
  t←0
  ustal początkowe P(t)
  oceń P(t)
  while (not warunek zakończenia) do
  begin
    t←t+1
    wybierz P(t) z P(t-1)  //Selekcja
    zmień P(t)             //Mutacja oraz krzyżowanie
    oceń P(t)
```

```
  end
end
```

W tej pracy przedstawię jak przystosowałem powyższy schemat do przedstawionego problemu.

2.3 Populacja początkowa

Pierwszym krokiem w algorytmach populacyjnych jest ustalenie populacji początkowej. Wybór tej populacji zależy od problemu, ale zazwyczaj dąży się aby wszystkie rozwiązania w populacji były dopuszczalne oraz aby rozwiązania nie były skupione w jednym punkcie.

Populację początkową można wygenerować jako populację rozwiązań zerowych czyli takich bez jakiejkolwiek rozpoczętej uprawy w rozwiązaniu. Jest to jednak nie najlepszy sposób z powodu tego że część iteracji algorytmu na początku będzie musiała być poświęcona na wstępne zapełnienie rozwiązań. Można ten czas oszczędzić generując od razu populację w której każde rozwiązanie ma już kilka zaczętych upraw. Przykład takiej metody jest przedstawiony poniżej

Dla każdego rozwiązania w populacji losujemy kolejność pól zgodnie z którą będzie uzupełniane rozwiązanie. Następnie dla każdego pola losujemy uprawę oraz dzień rozpoczęcia z przedziału dopuszczalnego dla danej uprawy. Jeśli po sprawdzeniu ograniczeń zasobowych rozwiązanie z nową uprawą jest dopuszczalne to dodajemy tą uprawę, jeśli nie to przechodzimy do następnego pola w kolejności.

Plusy tej metody są następujące żadne pole ani rodzaj uprawy nie będzie faworyzowany oraz otrzymamy zawsze rozwiązanie dopuszczalne. Minusem jest to że metoda jest bardziej skomplikowana od generacji rozwiązań zerowych oraz to że metoda stworzy co najwyżej jedną uprawę na polu.

2.4 Selekcja

Selekcja polega na wyborze rozwiązań które przechodzą do następnej iteracji algorytmu. Celem tego etapu jest wprowadzenie presji ewolucyjnej. Sprawia ona że rozwiązania dążą do poprawy funkcji celu. Bez selekcji populacja rozwiązań losowo błądziłaby po przestrzeni rozwiązań. W metodach selekcji opisanych poniżej lepsze rozwiązania mają większe prawdopodobieństwo przejścia do następnej iteracji.

W metodzie koła ruletki prawdopodobieństwo wyboru danego osobnika jest wprost proporcjonalne wartości jego funkcji celu. Aby wykorzystać tą metodę należy najpierw zsumować wartości funkcji celu wszystkich osobników w populacji, a następnie dla każdego osobnika podzielić wartość funkcji celu przez tą sumę. Otrzymamy w ten sposób

prawdopodobieństwa wyboru dla każdego osobnika sumujące się do 1. Prawdopodobieństwo wyboru osobnika opisuje wzór:

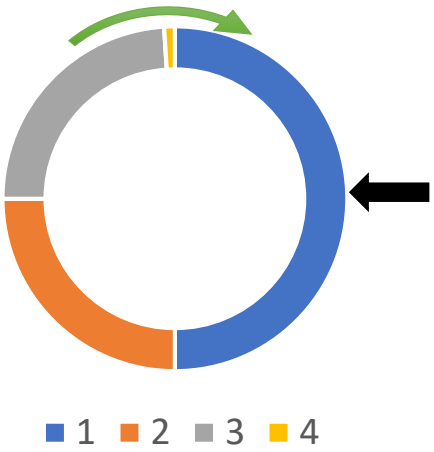
$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

gdzie:

f_i - wartość funkcji celu osobnika i,

N - liczba osobników w populacji.

Następnie należy przypisać każdemu rozwianiu część koła zgodnie z prawdopodobieństwem wyboru oraz określić punkt wyboru rozwiązania. Po zakręceniu kołem osobnik na którym znajdzie się ten punkt jest wybierany. Należy zakręcić kołem tylekrotnie ile chcemy wybrać osobników.



Osobnik	Wartość funkcji celu	Prawdopodobieństwo wyboru
1	25	0,5
2	12,5	0,25
3	12	0,24
4	0,5	0,01

Rys. 1 Metoda ruletki, gdzie czarna strzałka oznacza punkt wyboru rozwiązania, a tabela po prawej przedstawia obliczenia prawdopodobieństwa.

Standardowa wersja tej metody nie pozwala na zastosowanie w przypadku ujemnych wartości funkcji celu. Można jednak zdefiniować podobną metodę która różni się tylko tym że na początku dodaje do wszystkich wartości funkcji celu najbardziej ujemną wartość funkcji celu w populacji. Niestety tracimy wtedy własność że prawdopodobieństwo wyboru jest wprost proporcjonalne wartości funkcji celu. Jeśli nie występują wartości ujemne to ta metoda działa tak samo jak standardowa. Minus tej metody uwidacznia się kiedy wszystkie osobniki mają bardzo bliską wartość funkcji celu a przez to bliskie prawdopodobieństwo wyboru. Wtedy presja ewolucyjna jest efektywnie tracona.

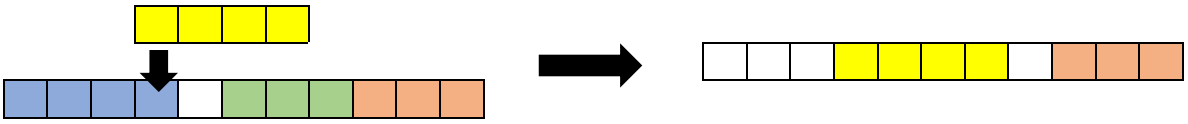
W selekcji rankingowej osobniki w populacji są sortowane rosnąco ze względu na wartość funkcji celu. Selekcja polega na wyborze określonej liczby osobników z końca posortowanej listy. Plusem tej metody jest zachowanie presji ewolucyjnej również wtedy gdy osobniki mają bliskie wartości funkcji celu. Metoda działa zarówna dla ujemnych i dodatnich wartości funkcji celu. Minusem tej metody jest brak losowości w wyborze, przez co gorsze rozwiązania nie maja możliwości bycia wybranym. Może to negatywnie wpływać na różnorodność w populacji.

W selekcji turniejowej przeprowadzamy tyle turniejów ile osobników potrzebujemy wybrać. Turniej polega na wyborze najlepszego rozwiązania z grupy losowych rozwiązań. Wielkość tej grupy wynosi dwa lub więcej osobników. Metoda działa zarówna dla ujemnych i dodatnich wartości funkcji celu.

2.5 Mutacja

Mutacja w algorytmach ewolucyjnych ma na celu zróżnicowanie rozwiązań i przeciwdziała utknięciu populacji w lokalnym ekstremum. Polega ona na małej zmianie losowej części rozwiązania.

Proponowana metoda mutacji polega na losowym wyborze typu uprawy, następnie dnia rozpoczęcia z możliwych dla danego typu. Należy też wybrać pole na którym zostanie rozpoczęta ta uprawa. Można wybrać pole losowo gdzie każde pola ma równe prawdopodobieństwo, lub uwzględnić współczynnik dopasowania pola do typu produktu. Dobrze nadaje się do tego do tego metoda ruletki w której prawdopodobieństwo wyboru jest wprost proporcjonalne do współczynnika dopasowania. Jeśli na danym polu i terminie są już uprawy to należy je usunąć z rozwiązania.



Rys. 2 Zasada wstawiania nowej uprawy (kolor żółty) do harmonogramu upraw na polu.

Przedstawiona metoda mutacji nie jest w stanie usunąć wszystkich upraw z danego pola. Dodatkowo jeśli rozpoczętych upraw jest niewiele w stosunku do pól to bardziej prawdopodobne jest dodanie nowej uprawy niż usunięcie istniejącej na skutek mutacji. Nie jest to problemem w przypadku stosowania funkcji poprawy, ale może być w przypadku funkcji kary. Więcej o informacji tych funkcjach i ich działaniu w podrozdziale Ocena rozwiązania.

Dlatego zaproponowano też drugą metodę mutacji która najpierw losuje z równym prawdopodobieństwem czy dodajemy nową uprawę czy usuwamy już istniejącą. W przypadku usuwania wszystkie uprawy mają równą szansę wyboru.

Mutacji poddawana jest w każdej iteracji część osobników. Osobniki do mutacji są wybierane losowo zgodnie z pewnym prawdopodobieństwem. Jest ono takie samo dla każdego osobnika i zazwyczaj takie samo dla wszystkich iteracji algorytmu. Właściwy dobór prawdopodobieństwa mutacji jest ważnym zagadnieniem i istotnie wpływa na skuteczność algorytmu.

2.6 Krzyżowanie

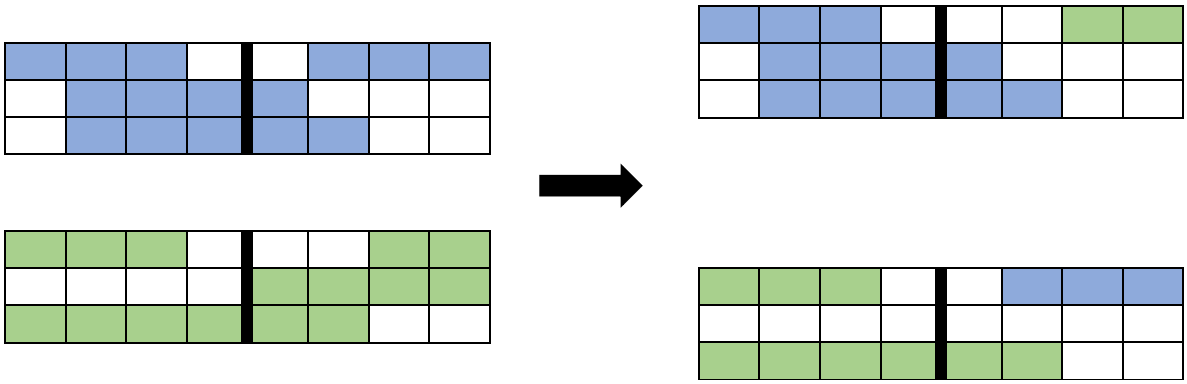
Krzyżowanie ma za zadanie stworzenie nowego rozwiązania wykorzystując do jego budowy dwa inne rozwiązania. Dzięki krzyżowaniu uzyskujemy rozwiązania o kombinacji cech rodziców. Jeśli cechy te są dobre pod względem funkcji celu to otrzymamy osobnika lepszego od każdego z rodziców. Co prawda możliwe jest również otrzymanie osobnika gorszego od obu rodziców, ale takie rozwiązanie zostanie prawdopodobnie odrzucone na etapie selekcji.

W przedstawionym problemie nasuwają się dwa główne sposoby na przeprowadzenie krzyżowania. Krzyżowanie ze względu na:

- dni,
- pola.

Krzyżowanie ze względu na dni polega na wyborze części dni z jednego rozwiązania i reszty z drugiego. Problemem który pojawi się podczas przeprowadzania tego typu krzyżowania są uprawy będące w trakcie na granicy łączenia. Aby zminimalizować ten problem zastosowano krzyżowanie jednopunktowe. Jednak nie eliminuje to całkowicie tego problemu.

Proponowane rozwiązanie polega na wzięciu wszystkich upraw do punktu podziału z pierwszego rodzica, a po tym punkcie z drugiego rodzica. W ten sposób otrzymamy część upraw z pierwszego rodzica nie skończonych, a część z drugiego nie zaczętych. Uprawy rozpoczęte kończymy, a jeśli koliduje to z innymi uprawami to je usuwamy. Uprawy nie zaczęte usuwamy. Powtarzamy to zamieniając kolejnością rodziców aby otrzymać drugie rozwiązanie. Proces jest przedstawiony na rysunku poniżej.



Rys. 3 Zasada krzyżowania rozwiązań. Z rozwiązań po lewej otrzymano nowe po prawej.

Krzyżowanie ze względu na pola jest prostsze ponieważ kolejność pól nie ma znaczenia. Wybrano metodę krzyżowania biorącą parzyste pola z pierwszego rozwiązania oraz nieparzyste z drugiego i tworzące w ten sposób nowe rozwiązanie. Aby utrzymać symetrię należy powtórzyć proces zamieniając najpierw kolejnością rozwiązania, wtedy uzyskamy drugie nowe rozwiązanie.

2.7 Ocena rozwiązania

Na skutek zdefiniowanych powyżej metod krzyżowania lub mutacji mogą się pojawić rozwiązanie nie spełniające ograniczeń zasobowych. Inne ograniczenia będą zawsze spełnione. Można ten problem rozwiązać stosując między innymi metodę funkcji kary lub funkcji poprawy.

„Ich [metody funkcji kary] istota polega na modyfikacji funkcji celu przez włączenie do niej ograniczeń za pomocą tak zwanej funkcji kary. Wprowadzenie funkcji kary przekształca zagadnienie optymalizacji z ograniczeniami w zadanie optymalizacji bez ograniczeń. Dla zadania minimalizacji funkcja kary ma wartość dodatnią, a wartość zmodyfikowanej funkcji celu rośnie proporcjonalnie do wielkości o jaką naruszone zostały ograniczenia. […] Tak więc funkcja [kary] przypisuje pewną karę liczbową każdemu punktowi spoza zbioru rozwiązań dopuszczalnych”[1].

Jako że mamy do czynienia z maksymalizacją w przedstawionym problemie to funkcja kary musi mieć wartość ujemną. Należy najpierw znaleźć zbiór zasobów które zostały przekroczone K. Zasoby dzienne danego rodzaju są liczone osobno dla każdego dnia i traktowane jako osobny zasób. Jest wiele możliwości konstrukcji tej funkcji. W rozpatrywanym problemie przyjęto że każdy rodzaj zasobu będzie miał taki sam wpływ na funkcję kary oraz że wartość kary będzie zależna od wielkości względnej przekroczenia zasobu. Te założenia spełnia funkcja kary liczona według poniższego wzoru:

$$f = \sum_{k \in K} \frac{d_k - z_k}{d_k} f_{max}$$

gdzie:

k - rodzaj przekroczonego zasobu,

d_k - dostępna ilość zasobu k,

z_k - zużyta ilość zasobu k,

f_{max} - maksymalna uzyskana dotychczas wartość funkcji celu.

Funkcja kary może być zależna od numeru iteracji. Ma to na celu sprawienie aby na początku działania algorytmu funkcja kary nie wpływała znacząco na wartość funkcji celu, ale wraz z czasem coraz bardziej. Pozwala to na lepsze przeszukanie zbioru rozwiązań w początkowych iteracjach, a jednocześnie sprawia że prawdopodobieństwo otrzymania rozwiązania niedopuszczalnego pod koniec działania algorytmu jest mniejsze.

W programie jest możliwość uwzględnienia zależności funkcji kary od iteracji poprzez podanie dwóch współczynników. Pierwszy określa przez jaką liczbę należy przemnożyć funkcję kary w pierwszej iteracji, a drugi przez jaką należy przemnożyć w ostatniej. Dla pozostałych iteracji współczynnik przez który należy przemnożyć jest uzyskany za pomocą interpolacji liniowej.

Procedura poprawy ma na celu zmianę rozwiązania niedopuszczalnego na dopuszczalne. W przedstawionym problemie polega to na usunięciu upraw które zużywają zasoby, których brakuje. Należy wybrać uprawy do usunięcia w losowy sposób oraz przerwać usuwanie kiedy rozwiązanie stanie się dopuszczalne.

Dla przekroczonych zasobów dziennych w danym dniu można to uzyskać w następujący sposób. Należy ustawić pola w losowej kolejności a następnie po kolei sprawdzać czy na polu w danym dniu jest uprawa która zużywa dany zasób. Jeśli tak to usunąć tą uprawę. Następnie sprawdzić czy zasób dalej jest przekroczony, jeśli już nie to przerwać sprawdzanie. W przypadku zasobów całkowitych czyli przydzielonych na wszystkie dni, można postępować podobnie tylko należy dodatkowo losowo wybrać kolejność dni.

2.8 Parametry algorytmu

W opracowanym algorytmie jest możliwość wyboru:

- rodzaju i wielkości populacji początkowej,
- rodzaju krzyżowania,
- rodzaju oraz prawdopodobieństwa mutacji,

- funkcji kary lub poprawy,
- rodzaju selekcji.

Algorytm zakończy pracę kiedy wystąpi jeden z poniższych warunków:

- maksymalna liczba iteracji zostanie przekroczona,
- maksymalna liczba iteracji bez poprawy wartości funkcji celu zostanie przekroczona,

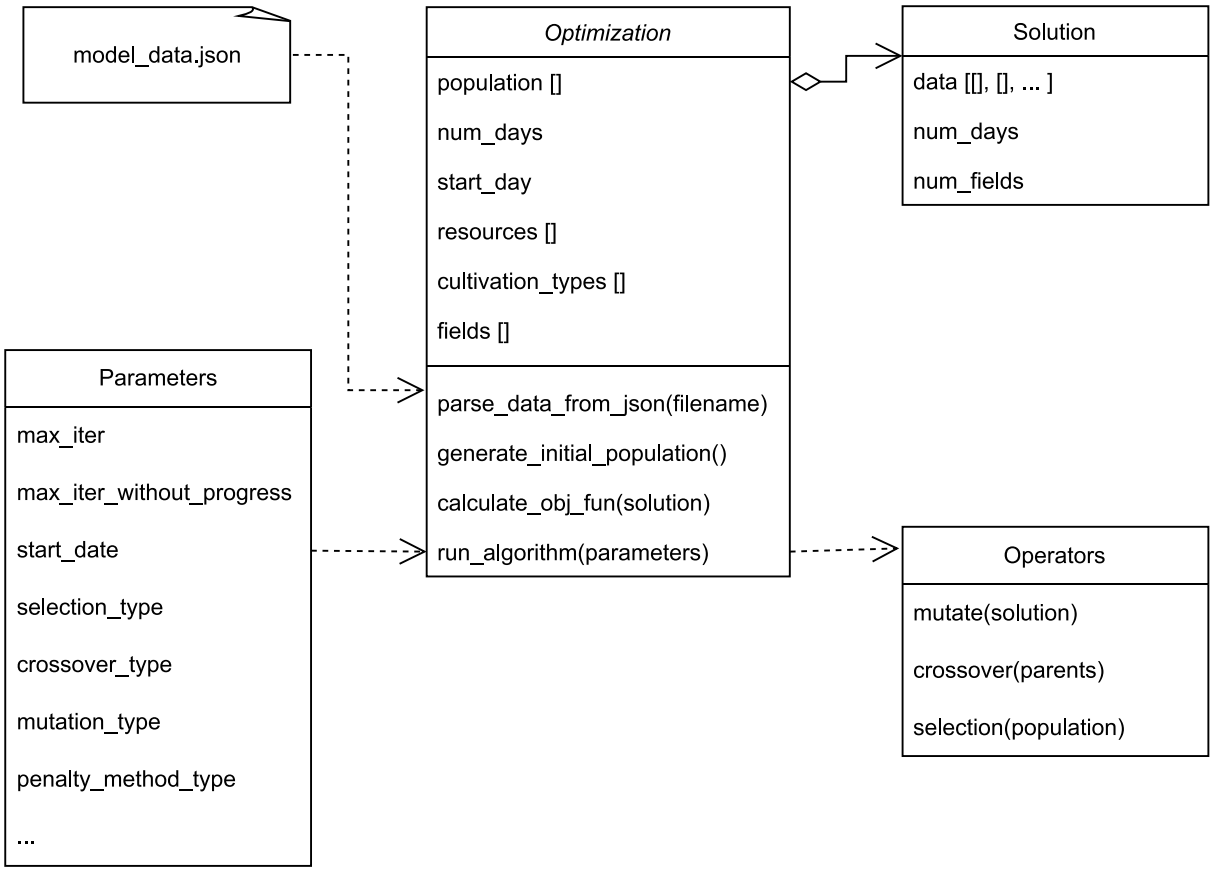
Są to standardowe warunki zakończenia pracy algorytmów optymalizacyjnych.

3 Opis aplikacji

3.1 Struktura programu

Program został podzielony na trzy zasadnicze części:

- interfejs graficzny,
- algorytm genetyczny,
- obsługę plików danych

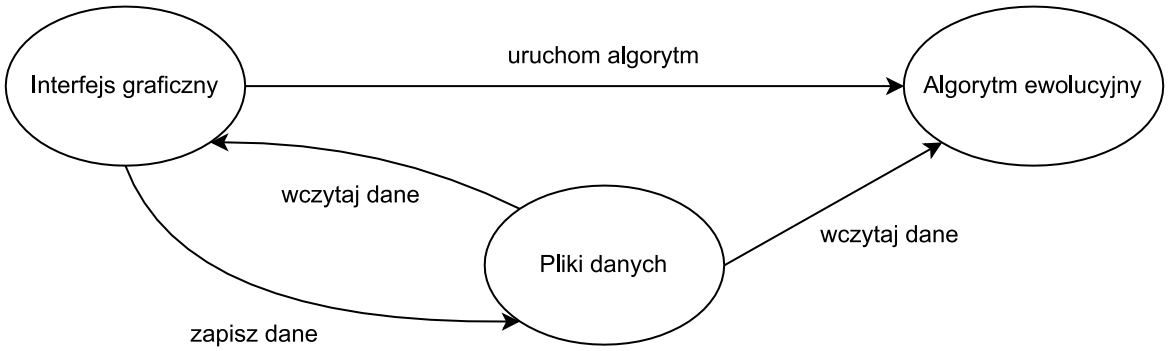


Rys. 4 Struktura programu

Do zapisu danych użyto pliku typu JSON z powodu jego popularności oraz dobrej współpracy z Pythonem. Zdecydowano się umieścić dane pól, zasobów i typów upraw w jednym pliku, aby uprościć późniejsze jego wczytywanie lub zapisywanie. Dane modelu są przekształcane w strukturę zagnieżdżonych list i słowników zaraz przed uruchomieniem algorytmu oraz zapisywane jako pola głównej klasy (Optimization). Przekształcenie pliku na

strukturę danych jest konieczne ponieważ korzystnie bezpośrednio z pliku podczas działania algorytmu jest bardzo czasochłonne.

Dzięki podanej strukturze programu udało się wyraźnie oddzielić interfejs graficzny od algorytmu. Schemat współpracy między tymi komponentami przedstawiony jest poniżej. Każda strzałka odpowiada wywołaniu jednej metody.



Rys. 5 Schemat współpracy pomiędzy komponentami systemu

3.2 Reprezentacja rozwiązania

Rozwiązanie jest obiektem klasy `Solution` który zawiera dla każdego pola listę upraw na nim rozpoczętych. Rozpoczętą uprawę przedstawia krotka w której pierwszy element oznacza typ uprawy a drugi dzień jej rozpoczęcia.

Początkowo rozwiązanie było przedstawione w formie macierzowej o wierszach odpowiadających polom i kolumnach odpowiadających dniom. Jednak zrezygnowano z takiego przedstawienia rozwiązania ponieważ zajmuje ono dużo miejsca, a przez to tworzenie nowych osobników jest czasochłonne.

3.3 Implementacja interfejsu graficznego

Zdecydowałem się na użycie Pythona również do utworzenia interfejsu graficznego. Nie jest to typowo język wykorzystywany do tego celu, ale taki wybór pozwala wykorzystać jego biblioteki jak między innymi matplotlib i pandas do tworzenia wykresów. Dodatkową zaletą wykorzystania jednego języka jest uproszczenie procesu budowania i instalacji programu.

Dwa najpopularniejsze narzędzia do tworzenia interfejsu graficznego w Pythonie to Tkinter [4] oraz Qt [5].

Tab 1 Porównanie popularnych narzędzi do tworzenia interfejsu graficznego w Pythonie

	Tkinter	Qt
Instalacja	Nie jest konieczna, jest to standardowa biblioteka.	Konieczna.
Prosty do zrozumienia	Tak, mała liczba funkcjonalności.	Nie, zrozumienie całej biblioteki zajmuje dużo czasu
Dostępne widżety	Tylko podstawowe widżety	Szeroki wybór
Graficzny konstruktor interfejsu	Brak	Dostępne narzędzie Qt Designer pozwalające dodawać widżety metodą przeciągnij i upuść (ang. drag and drop)
Licencja	Darmowa	Płatna dla użytku komercyjnego

Zdecydowano się na wybór QT głównie z powodu dostępności narzędzia QT Designer które znacznie przyspiesza proces tworzenia interfejsu. Program nie będzie używany komercyjnie więc nie potrzebna jest płatna licencja.

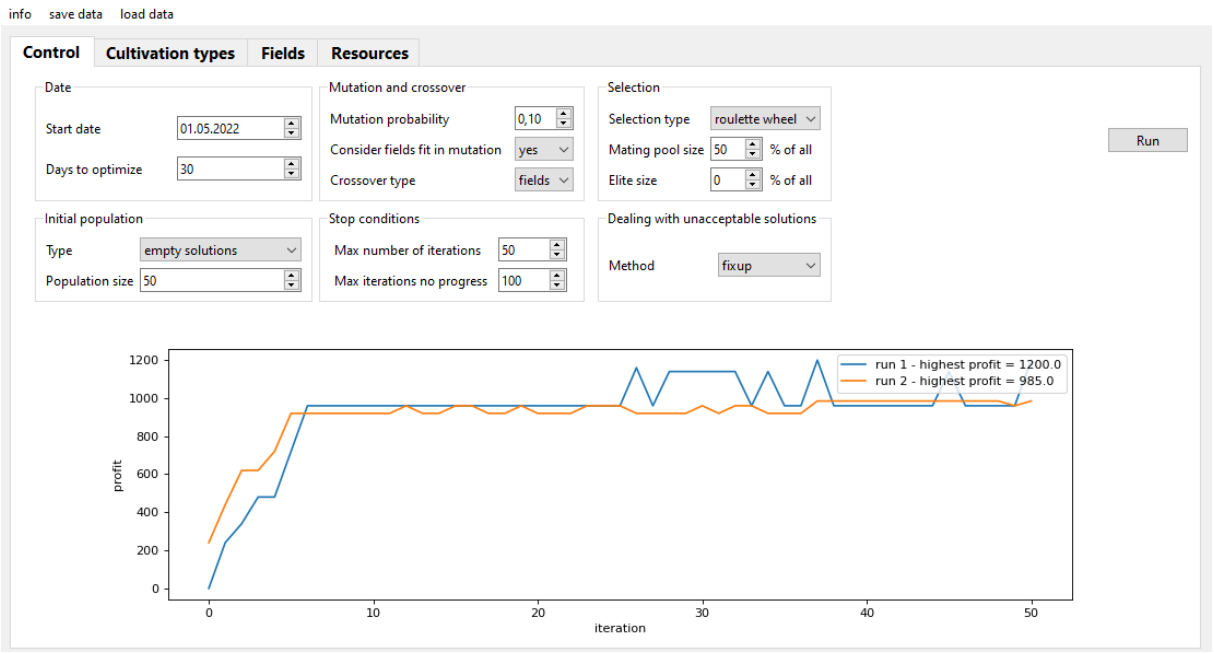
Interfejs składa się z następujących części:

- pasek narzędzi,
- zakładka control,
- zakładka cultivation types,
- zakładka fields,
- zakładka resources,
- okno rozwiązania.

Pasek narzędzi jest widoczny u góry okna programu. Pozwala on wczytać wybrany przez użytkownika plik z danymi modelu, zapisać wprowadzone dane do pliku. Program pozwala wczytać dowolny plik typu JSON, jeśli plik nie jest spełnia warunków tego typu zostanie wyświetlony komunikat o błędzie. Po poprawnym wczytaniu pliku zostanie wyświetlony komunikat mówiący ile typów upraw, zasobów i pól udało się wczytać.

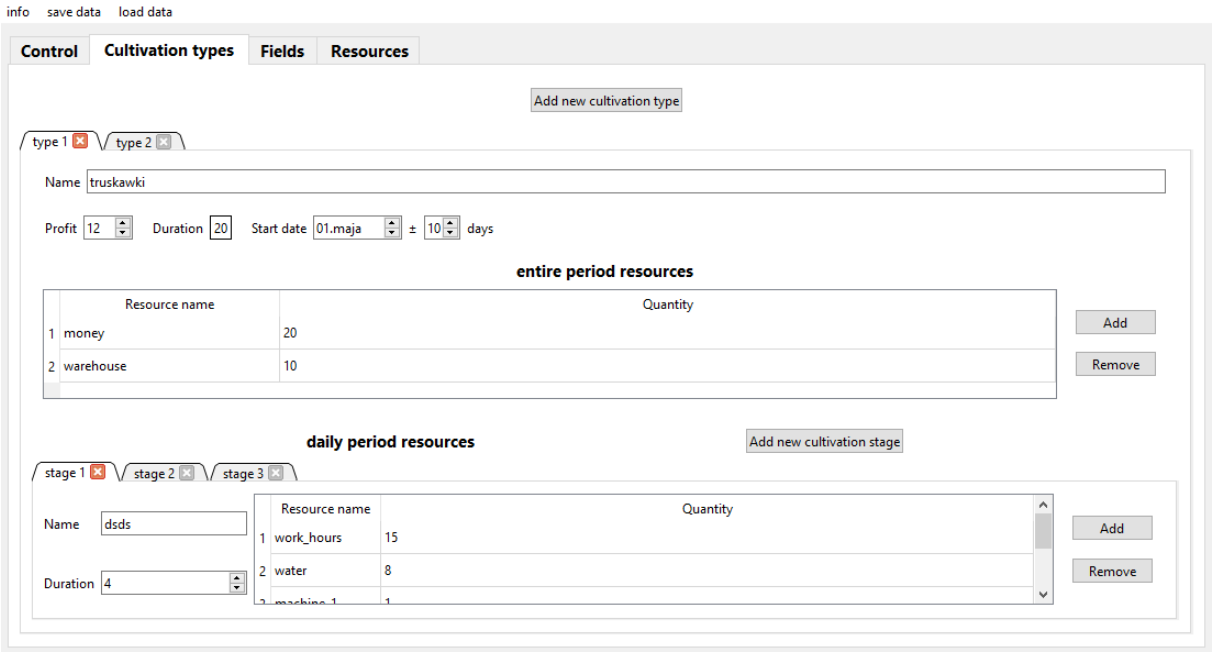
Zakładka control pozwala ustawić parametry algorytmu oraz go uruchomić za pomocą przycisku Run. Ponadto po zakończeniu pracy algorytmu zostanie wyświetlony wykres

przedstawiający zysk z najlepszego rozwiązania w każdej iteracji algorytmu. Dzięki temu wykresowi można porównać jak zmiana parametrów wpłynęła na uzyskane rozwiązania.



Rys. 6 Zakładka control pozwalająca ustawić parametry algorytmu

Zakładka cultivation types pozwala wprowadzić dane poszczególnych rodzajów upraw. Możliwy przedział czasu w którym można rozpocząć uprawę jest określony poprzez datę w środku przedziału oraz ilość dni o ile można maksymalnie przesunąć datę.

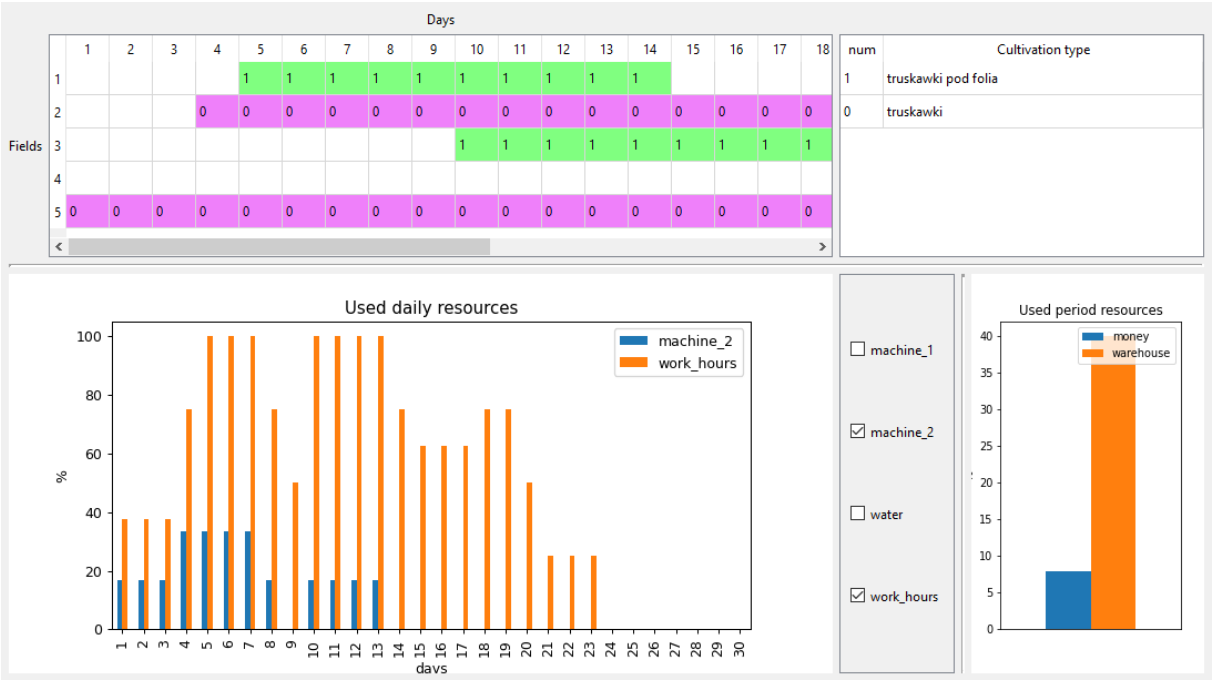


Rys. 7 Zakładka cultivation types pozwalająca wprowadzić dane procesu uprawy

Zakładka fields pozwala wprowadzić dane pól, w tym ich powierzchnię oraz współczynniki dopasowania danego pola do wszystkich typów upraw. Zakładka resources

pozwala wprowadzić dostępne zasoby do tabel z podziałem na zasoby dzienne i te przydzielone na cały okres uprawy.

Okno rozwiązania pojawia się po zakończeniu pracy algorytmu pokazuje najlepsze uzyskane rozwiązanie oraz zużycie zasobów w tym rozwiązaniu. W górnej części okna widzimy terminarz upraw wraz z legendą. Terminarz jest kolorowany dla czytelności, każdy kolor odpowiada innemu typowi uprawy. Poniżej przedstawione jest zużycie zasobów dziennych w każdym dniu oraz zużycie zasobów przydzielonych na cały okres uprawy. Jako że ilość rodzajów zasobów może być duża udostępniono możliwość wyboru które zasoby są wyświetlane na wykresie.



Rys. 8 Okno rozwiązania przedstawia najlepsze rozwiązanie oraz zużyte zasoby

4 Eksperymenty obliczeniowe

4.1 Testy poprawności działania aplikacji

Przeprowadzono testy jednostkowe większości metod w programie. Wiele z nich ma sobie element losowy, co jest problemem podczas przeprowadzania testów. Aby zniwelować ten problem wykorzystano technikę mockowania. Polega ona na zastąpieniu problematycznej funkcji, w tym wypadku losującej, funkcją która zwraca znaną oraz zawszę tą samą wartość.

Testy jednostkowe nie są wystarczające aby stwierdzić poprawność działania. Dlatego przeprowadzono również manualne testy poniższych podstawowych scenariuszy end-to-end czyli przedstawiających cały proces użycia programu.

1. Uruchomienie programu, wczytanie przykładowych danych, uruchomienie algorytmu.
2. Uruchomienie programu, wczytanie przykładowych danych, zmiana danych, uruchomienie algorytmu, zapisanie zmienionych danych do plików.
3. Uruchomienie programu, wczytanie zapisanych wcześniej danych, uruchomienie algorytmu.
4. Uruchomienie programu wpisanie danych od zera, uruchomienie algorytmu.

Program testowano głównie na systemie Windows 10 64 bit, ale odpalono program również na systemie Windows 11 i przeprowadzono podstawowe testy. Zaobserwowano jedynie zmiany graficzne.

Sprawdzono również odporność programu na wprowadzanie nie poprawnych danych. Na skutek testów między innymi zablokowano możliwość wprowadzenia ujemnych ilości zasobów. Wprowadzono też uzupełnianie listy zasobów to znaczy jeśli dany zasób jest używany w uprawie ale nie obecny w liście dostępnych to jest on dodawany z ilością równa zero.

4.2 Instancje testowe

Zdefiniowano poniższe instancje testowe. Dla wszystkich rozmiar populacji wynosił 100 osobników, w populacji początkowej wykorzystano częściowo zapełnione rozwiązania oraz wielkość puli osobników rozmnażających się to 40.

Zestaw danych 1. Dane modelu zawierają 8 pól, 8 typów upraw, 10 typów zasobów. Średnia ilość dostępnych zasobów. Optymalizację przeprowadzono na przedziale 60 dni. Użyto krzyżowanie ze względu na pola, selekcję rankingową, funkcję poprawy oraz mutację dodającą.

Zestaw danych 2. Dane modelu zawierają 8 pól, 8 typów upraw, 10 typów zasobów. Średnia ilość dostępnych zasobów. Optymalizację przeprowadzono na przedziale 60 dni, Prawdopodobieństwo mutacji jest równe 0,5. Użyto selekcję rankingową, funkcję poprawy oraz mutację dodającą.

Zestaw danych 3. Dane modelu zawierają 8 pól, 8 typów upraw, 10 typów zasobów. W ramach tego zestawu zdefiniowano trzy przypadki dostępności zasobów, czyli mała, średnia oraz duża. Optymalizację przeprowadzono na przedziale 60 dni. Prawdopodobieństwo mutacji jest równe 0,5. Użyto krzyżowanie ze względu na pola oraz selekcję rankingową.

Zestaw danych 4. Dane modelu zawierają 8 pól, 8 typów upraw, 10 typów zasobów. Średnia ilość dostępnych zasobów. Optymalizację przeprowadzono na przedziale 60 dni. Prawdopodobieństwo mutacji jest równe 0,5. Użyto krzyżowanie ze względu na pola, funkcję poprawy oraz mutację dodającą.

Zestaw danych 5. Dane modelu zawierają 30 pól, 15 typów upraw, 12 typów zasobów. Optymalizację przeprowadzono na przedziale 180 dni. Prawdopodobieństwo mutacji jest równe 0,5. Użyto krzyżowanie ze względu na pola, selekcję rankingową, funkcję poprawy oraz mutację dodającą.

4.3 Dobór parametrów mutacji oraz krzyżowania

Najważniejszym parametrem mutacji jest prawdopodobieństwo jej wystąpienia. Przeprowadzono testy dla prawdopodobieństw mutacji oraz liczby iteracji przedstawionych w tabeli. Pozostałe parametry oraz dane modelu są określone w zestawie danych 1. Są to dane o średnim rozmiarze czyli 10 pól, 8 typów upraw, 10 typów zasobów.

Z powodu losowości w algorytmie przeprowadzono 10 testów dla każdej wartości prawdopodobieństwa i liczby iteracji oraz wyliczono z tych danych średnią. Wyniki są przedstawione w tabeli 2.

Tab 2 Uzyskany zysk w zależności od prawdopodobieństwa mutacji p_m oraz liczby iteracji

	Liczba iteracji				
	100	250	500	1000	2000
$p_m = 0{,}3$	1002	1039	1085	1113	1159
$p_m = 0{,}4$	1010	1081	1118	1152	1182
$p_m = 0{,}5$	1036	1085	1118	1164	1172
$p_m = 0{,}6$	1047	1102	1125	1153	1175
$p_m = 0{,}7$	1030	1106	1130	1145	1159

Jak widać w tabeli powyżej, dla 100 iteracji najlepszy wynik daje prawdopodobieństwo równe 0,6. Natomiast dla 2000 iteracji najlepszy wynik daje prawdopodobieństwo równe 0,4. Warto też zauważyć że prawdopodobieństwo mutacji nie ma aż tak dużego wpływu na końcowy wynik szczególnie dla dużych ilości iteracji. Ponadto najlepsze wyniki osiągnięto dla zaskakująco dużych wartości p_m.

Można spróbować wyciągnąć wniosek że wraz ze wzrostem ilości iteracji lepsze wyniki daje mniejsza wartość prawdopodobieństwa. Jednak dane nie w całości to potwierdzają to znaczy dla iteracji 250 i 500 najlepszy wynik jest dla $p_m = 0{,}7$. Ponadto średnią wyliczano tylko dla 10 powtórzeń z powodu długich obliczeń dla wysokich wartości iteracji, więc nie są to zbyt wiarygodne dane.

Niemniej jednak ten wniosek ma podstawy teoretyczne jako że większa wartość prawdopodobieństwa mutacji sprawia że populacja szybciej, ale mniej dokładnie przeszukuje przestrzeń rozwiązań. Dla tego dla małej liczby iteracji lepiej się sprawdza duża wartość p_m a dla większej liczby iteracji mniejsza

Ciekawą alternatywą do manualnego wyznaczania parametrów mutacji jest wyznaczanie ich automatycznie podczas pracy algorytmu [8]. Pozwala to uprościć proces korzystania z algorytmów genetycznych. W przedstawionym w pracy algorytmie nie zastosowano tej metody.

Sprawdzono jak zachowują się dwie zaimplementowane metody krzyżowania. Testy przeprowadzono dla zestawu danych 2. Wybrano prawdopodobieństwo mutacji równe 0,5. Tabela poniżej przedstawia wartości średnie dla 10 uruchomień algorytmu.

Tab 3 Porównanie wyników działania dwóch metod krzyżowania

	Liczba iteracji				
	100	250	500	1000	2000
Krzyżowanie ze względu na dni	954	985	1003	1003	1003
Krzyżowanie ze względu na pola	1029	1080	1106	1147	1165

Niezalenie od ilości iteracji krzyżowanie ze względu na pola dało lepsze wyniki. Jest to najprawdopodobniej spowodowane tym że krzyżowanie ze względu na dni usuwa jedną z nakładających się uprawy na granicy podziału rozwiązań. Krzyżowanie ze względu na pola nie ma tego problemu.

4.4 Dobór parametrów funkcji kary i poprawy

Porównano funkcję kary z funkcją poprawy. Dla obu sprawdzono też która metoda mutacji jest lepsza. Dla przypomnienia są zdefiniowane dwie metody mutacji:

- Dodająca - dodaje nową uprawę w losowe miejsce.
- Usuwająca lub dodająca - usuwa jedną z istniejących lub doje nową uprawę w losowe miejsce.

Według [9] skuteczność funkcji kary zależy w dużym stopniu od tego w jak dużym stopniu przestrzeń rozwiązań jest niedopuszczalna. Dlatego dla każdego zestawu parametrów przeprowadzono testy na trzech przypadkach dostępności zasobów.

- Mała ilość zasobów - oznacza to możliwość uprawy tylko na części pól z powodu braku zasobów.
- Średnia ilość zasobów - zasoby pozwalają na każdym polu uprawiać, ograniczają jakie uprawy mogą być na polach jednocześnie.
- Duża ilość zasobów - zasoby nie są ograniczeniem, tylko ilość pól i dni.

Testy przeprowadzono dla 1000 iteracji algorytmu, dla danych z zestawu 3. Tabela poniżej przedstawia wartości średnie dla 10 uruchomień algorytmu.

Tab 4 Porównanie działania funkcji kary i naprawy

	Mała ilość zasobów	Średnia ilość zasobów	Duża ilość zasobów
Funkcja kary Mutacja dodająca	287	893	1128
Funkcja kary Mutacja dodająca lub usuwająca	298	867	1115
Funkcja poprawy Mutacja dodająca	321	897	1118
Funkcja poprawy Mutacja dodająca lub usuwająca	303	878	1119

Jak widać w tabeli dla dużej ilości zasobów nie ma większej różnicy pomiędzy tymi metodami czego należało się spodziewać ponieważ rozwiązanie nigdy nie będzie niedopuszczalne. W tym wypadku te funkcje nie zostaną wywołane. Dla pozostałych przypadków obie funkcje dają podobny wynik, ale funkcja poprawy daje minimalnie lepszy wynik.

Trzeba zauważyć że dla funkcji kary uruchomiono program wielokrotnie w celu doboru współczynników funkcji kary. Przy zbyt małej wartości funkcji kary dostawano rozwiązania niedopuszczalne szczególnie dla mutacji dodającej. A dla zbyt dużej dostawano wyraźnie gorsze wyniki. Co sprawia że metoda funkcji poprawy jest wygodniejsza w użyciu. Mutacja dodająca sprawdziła się minimalnie lepiej jeśli używana razem funkcją poprawy.

4.5 Dobór metody selekcji

Zbadano poniższe metody selekcji, zastosowano zestaw danych 4.

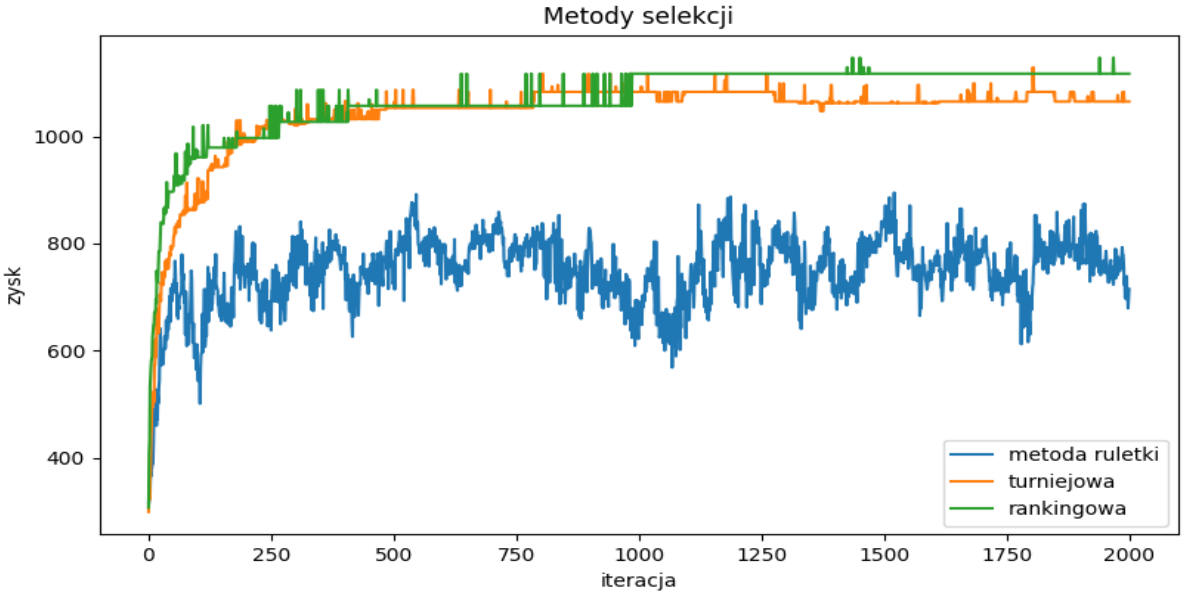
- Selekcja metodą ruletki.
- Selekcja turniejowa z wielkością turnieju równą 2.
- Selekcja rankingowa.

Prawdopodobieństwo mutacji równe 0,5, krzyżowanie ze względu na pola i funkcja poprawy. Tabela poniżej przedstawia wartości średnie dla 10 uruchomień algorytmu.

Tab 5 Porównanie różnych metod selekcji

	Liczba iteracji				
	100	250	500	1000	2000
Selekcja metodą ruletki	782	837	871	910	924
Selekcja turniejowa	968	1066	1098	1144	1163
Selekcja rankingowa	1036	1113	1119	1150	1184

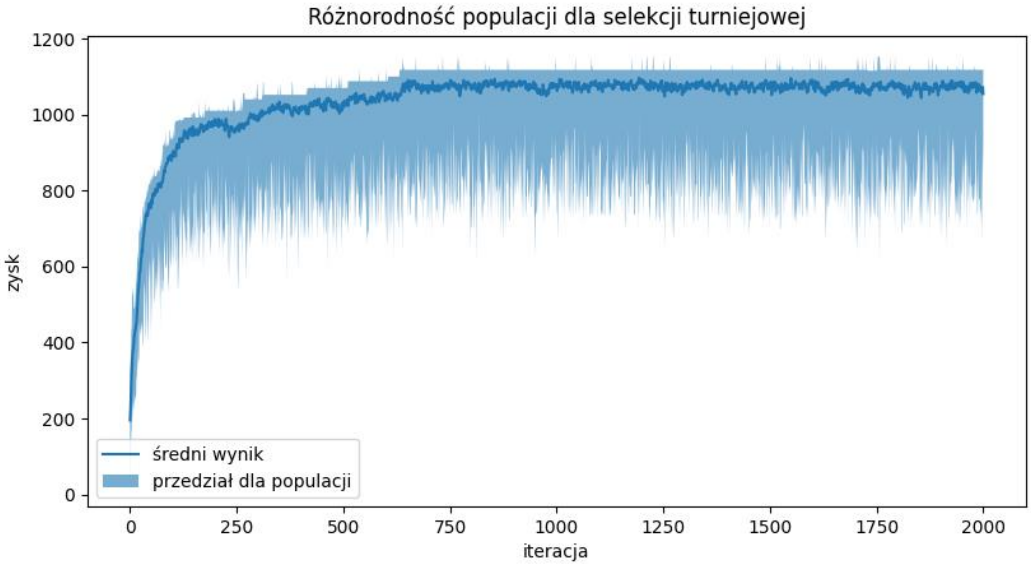
Najlepiej sprawdziła się selekcja rankingowa niezależnie od numeru iteracji. Selekcja turniejowa dawało nieznacznie gorsze wyniki, a selekcja metodą ruletki wyraźnie gorsze.



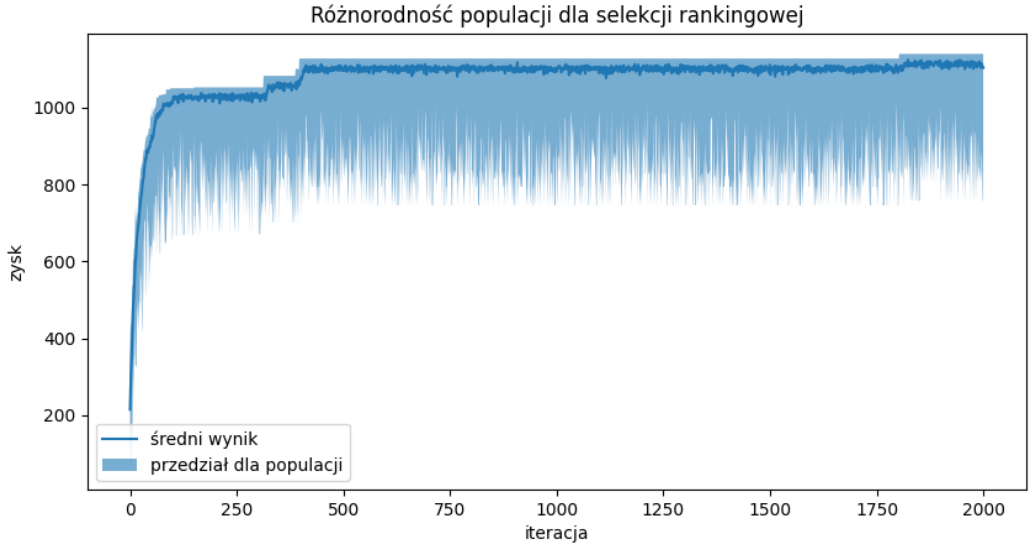
Rys. 9 Porównanie działania metod selekcji

Na rysunku Rys. 10 i Rys. 11 przedstawiono jak wygląda rozkład osobników w populacji. Jak widać nawet dla dużych numerów iteracji są w populacji osobniki o niskiej wartości funkcji

celu, ale większość osobników (średnia) jest bliżej najlepszych osobników. Wykresy dla selekcji turniejowej i rankingowej są podobne, ale można zauważyć że dolna granica wykresu selekcji rankingowej jest mniej poszarpana. Jest tak dlatego że selekcja turniejowa nie pozwala najgorszym rozwiązaniom się rozmnażać.



Rys. 10 Różnorodność populacji dla selekcji turniejowej



Rys. 11 Różnorodność populacji dla selekcji rankingowej

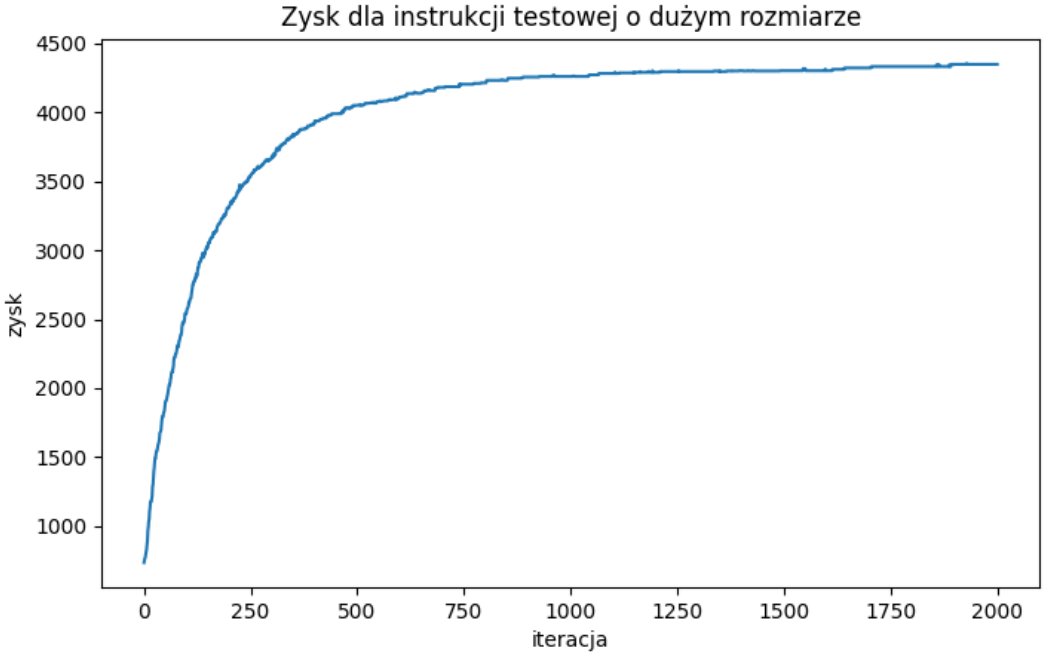
4.6 Wyniki dla instrukcji testowej o dużym rozmiarze

W tym rozdziale zbadano jak program poradzi sobie z bardzo rozbudowanymi danymi. Dane zawierają 30 pól, 15 rodzajów upraw i 12 rodzajów zasobów. Okres optymalizacji wynosi 180

dni. Jest to zestaw danych 5. Czas trwania został wyznaczony tylko dla działania algorytmu, pominięto czas zużywany przez interfejs graficzny na wyświetlenie wyników.

Tab 6 Uzyskany zysk i czas działania dla problemu o dużym rozmiarze

	Liczba iteracji				
	100	250	500	1000	2000
Uzyskany zysk	2441	3226	3756	3913	4074
Czas trwania obliczeń [s]	6.42	20.79	51.24	123.16	266.05



Rys. 12 Przebieg procesu optymalizacji dla instrukcji testowej o dużym rozmiarze

Różnica pomiędzy wynikiem dla 1000 i 2000 iteracji jest niewielka więc taka ilość iteracji jest w przybliżeniu wystarczająca dla tak dużego problemu. Dla 2000 iteracji czas trwania obliczeń jest poniżej 5 minut, co jest czasem w mojej opinii akceptowalnym. Na rysunku poniżej widać 25% terminarza rozwiązania. Warto zauważyć że rozwiązywany problem jest sztucznie utrudniony względem rzeczywistego problemu ponieważ uprawy zazwyczaj trwają znacznie dłużej niż zdefiniowane w problemie.

Rys. 13 Część terminarza rozwiązania dla problemu o dużym rozmiarze

5 Podsumowanie

Celem pracy było stworzenie programu, który ma za zadanie optymalizować terminarz upraw na określonej liczbie pól tak aby maksymalizować zysk przy ograniczeniu dostępnych zasobów. W tym celu należało zdefiniować model matematyczny oraz przystosować algorytm ewolucyjny do specyfiki problemu.

Cel udało się osiągnąć. Stworzono program w języku Python, który pozwala za pomocą interfejsu graficzne go wprowadzić dane modelu oraz ustawić parametry algorytmu. Dodatkowo program pozwala zwizualizować zużycie zasobów oraz zapisać lub wczytać dane modelu z pliku. Z narzędzi zastosowano między innymi bibliotekę Pandas oraz QT.

Napotkano problem z sposobem przedstawienia rozwiązania. Początkowo zaimplementowano rozwiązanie jako macierz, gdzie każdy element odpowiadał danemu dniu na danym polu. Jednak dla problemu o większym rozmiarze np. 180 dni oraz 20 pól dawało to nieakceptowalnie długi czas realizacji. Dlatego zastosowano postać rozwiązania jako listę list, gdzie każdy element odpowiadał uprawie. Jest to znacznie szybsze ponieważ zazwyczaj liczba upraw jest znacznie mniejsza niż liczba dni.

Zdefiniowany model matematyczny pozwala wprowadzić dowolną liczbę rodzajów zasobów. Oraz ustalić ile danego rodzaju zasobu jest potrzebne na danym etapie uprawy. Dzięki temu uzyskano elastyczny model nadający się nie tylko do optymalizacji upraw, ale też produkcji przemysłowej.

Jednak na skutek tak dużej elastyczności modelu, program wymaga wprowadzenia dużej ilości danych co może być minusem dla użytkowników. Ważnym kierunkiem rozwoju tego programu byłoby wprowadzenie bazy danych opisującej proces uprawy różnych rodzajów produktów. Wymaga to jednak dużej wiedzy rolniczej i doświadczenia, dlatego nie podjęto się tego zagadnienia. Kolejnym aspektem, który można by rozwinąć to wpływ pogody na proces uprawy. Modyfikowałby on długość wegetacji, wielkość plonu (zysk) oraz wymagane zasoby (np. woda).

6 Literatura

[1] Anna Danielewska-Tułecka, Jan Kusiak, Piotr Oprocha, Optymalizacja Wybrane metody z przykładami zastosowań, Wydawnictwo Naukowe PWN 2009

[2] Badanie popularności języków programowania PYPL PopularitY of Programming Language Index:
https://pypl.github.io/PYPL.html (28.11.2022)

[3] Bäck, Thomas, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms (New York, 1996; online edn, Oxford Academic, 12 Nov. 2020),

[4] Dokumentacja biblioteki Tkinker:
https://docs.python.org/3/library/tkinter.html (28.11.2022).

[5] Dokumentacja narzędzia QT Designer:
https://doc.qt.io/qt-6/qtdesigner-manual.html (28.11.2022).

[6] Ibrahim M. Al-Harkan, et al, A Decision Support System for Optimal Use of Irrigation Water and Crop Selection, Journal of King Saud University - Engineering Sciences, Volume 21, Issue 2, 2009, Pages 77-84, ISSN 1018-3639,

[7] Maffezzoli, et al, (2022). Agriculture 4.0: A systematic literature review on the paradigm, technologies and benefits. Futures. 142. 102998. 10.1016/j.futures.2022.102998.

[8] Mariusz Makuchowski, Adam Tyński. Automatyczna mutacja w algorytmach ewolucyjnych. Wydawnictwo AGH Automatyka 2009 tom 13 Zeszyt 2

[9] Michalewicz Zbigniew, Algorytmy genetyczne + struktury danych = programy ewolucyjne z angielskiego przełożył Zbigniew Nahorski.Wydanie trzecie. Warszawa : Wydawnictwa Naukowo-Techniczne,

[10] Richard Dawkins, The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe without Design, W. W. Norton & Company, 28.09.2015

[11] Stanisław Samborski, Rolnictwo precyzyjne, Wydawnictwo Naukowe PWN, 2007 ISBN: 9788301198985

[12] Urja Thakkar, et al, Application of Operations Research in Agriculture, International Journal of Scientific & Engineering Research Volume 10, Issue 10, October-2019

[13] Ya Ivanyo, et al, Models of optimization of combination of production of agrarian products and harvesting of wild food resources, E3S Web of Conf. Volume 222, 2020 https://doi.org/10.1051/e3sconf/202022201016

[14] Ya Ivanyo et al Optimization models of agricultural production with heterogeneous land resources, 2021 J. Phys.: Conf. Ser. 1989 012041 https://doi.org/10.1088/1742-6596/1989/1/012041

[15]Zhao Qingzhen, et al, The Application of Operations Research in the Optimization of Agricultural Production. Operations Research (1991) 39(2):194-205. https://doi.org/10.1287/opre.39.2.194