AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI AUTOMATYKI INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

KIERUNEK AUTOMATYKA I ROBOTYKA II STOPIEŃ

GŁĘBOKIE UCZENIE I INTELIGENCJA OBLICZENIOWA
2022/2023

Tennis League Tournament Optimization

Grupa: ***1 (11.15 – 13.30 środa)***

L.p.	Członkowie grupy:	Nr. albumu	Adres email
1	Filip Gąciarz	400267	filipgaciarz@student.agh.edu.pl
2	Maria Mazur	403592	mariam@student.agh.edu.pl
3	Artur Morys-Magiera	402111	npodgorski@student.agh.edu.pl
4	Michał Motyl	401943	michamotyl@student.agh.edu.pl
5	Norbert Podgórski	402111	npodgorski@student.agh.edu.pl
6	Bartosz Sroka	400490	srokab@student.agh.edu.pl

Opiekun: dr hab. inż. Joanna Kwiecień

Spis Treści

Wstęp

Projekt polega na stworzeniu oprogramowania, które będzie generowało drabinki turniejowe zawodów tenisowych na podstawie strategii ewolucyjnej. Algorytm znajdzie również zastosowanie w innego rodzaju turniejach wymagających utworzenie drabinki, takich jak szachy turniejowe czy brydż, natomiast należałoby wtedy uwzględnić inne cechy indywidualne zawodników.

Algorytm genetyczny jest metodą heurystyczną, która naśladuje procesy ewolucyjne zachodzące w naturze, takie jak selekcja, mutacja i krzyżowanie, aby otrzymać optymalne rozwiązania problemów.

Przyjęta strategia jest modyfikacją klasycznego modelu algorytmu genetycznego. Pierwszym operatorem działającym na populacji jest mutacja a kolejnym krzyżowanie. Nowa populacja jest akutalizowana w każdej operacji działającej na populacji. Kolejne operatory wykorzystują zmodyfikowany stały zestaw osobników określony w konfiguracji systemu.

Projekt ma na celu stworzenie łatwego i skutecznego sposobu generowania drabinek turniejowych dla zawodów tenisowych, co może zaoszczędzić czas i wysiłek organizatorów. Ostatecznym celem projektu jest stworzenie intuicyjnego interfejsu użytkownika, który pozwoli na łatwe wprowadzenie danych wejściowych i wygenerowanie optymalnej drabinki turniejowej w możliwie krótkim czasie.

Opis problemu

Celem projektu jest stworzenie programu, który pozwoli na wygenerowanie drabinki turniejowej dla dowolnej liczby uczestników turnieju tenisowego, z uwzględnieniem różnych sposobów rozstawienia graczy. Algorytm ewolucyjny będzie działał na podstawie oceny jakości meczy między zawodnikami dla wygenerowanych drabinek turniejowych, tak aby otrzymać suboptymalne rozwiązanie. Zadaniem funkcji celu będzie maksymalizowanie *„widowiskowości”* przeprowadzonego turnieju, bazując na dostępnych informacjach o historii ostatnich spotkań zawodników.

Termin *“widowiskowości”* powiązany jest przykładowo z faktem, kiedy posiadamy zawodnika, który wykazuje bardzo dobrą passę (wygrywa kilkanaście razy z rzędu) i ma się zmierzyć z zawodnikiem, który ma wysokie elo. W takiej sytuacji pojawiają się pytania czy *“mistrz”* przerwie passę tego zawodnika czy nie. Analizowany aspekt zwiększa poziom *“widowiskowości”* spotkania.

Struktura danych wejściowych

Dane wejściowe kodują informacje o następujących elementach:

- liczba graczy (zawodników), ich dane osobowe, ranking ELO (liczba zmiennoprzecinkowa proporcjonalna do ich poziomu gry)
- liczba rund w turnieju
- składy osobowe oraz statystyki i wyniki meczy w poszczególnych rundach

Do zapisu danych wykorzystano format JSON z uwagi na przejrzystość, znajomość przez wszystkich członków zespołu, jak i prostą możliwość przeniesienia 1:1 ze skryptu generującego testowe scenariusze turniejowe, do podania na wejście skryptu, dostępu przez API REST, jak i wgrania poprzez interfejs webowy przez użytkownika końcowego. Z uwagi na możliwość dostępu przez API REST, skorzystano z formatu JSON, gdyż ciało żądania typu POST może być - z ustawieniem nagłówka Content-Type - zakodowane bezpośrednio w tym formacie.

Zbiór danych jest listą list, z których każda zawiera obiekty - mecze. Każdy z meczy koduje informacje o ilości zdobytych przez graczy setów i asów serwisowych, jak i informacje o obu graczach: ich ID, imieniu i nazwisku oraz wartości rankingu ELO.

Poniżej przedstawiono przykładowy fragment wygenerowanych danych:

```
[ <lista rund
        [ < runda 1 - lista meczy w tej rundzie
                { < mecz 1 w rundzie 1
                "p1": {
                                "id": 3,
                                "name": "Nancy Hall",
                                "elo": 10
                },
                "p2": {
                                "id": 1,
                                "name": "Kimberly Schwartz",
                                "elo": 90
                },
                "s1": 0,
                "s2": 1,
                "a1": 0,
                "a2": 2
                }.
                { … < mecz 2 w rundzie 1
                }
        ],
        [ < runda 2
                { … < mecz 1 w rundzie 2
                },
                …
        ],
        …
]
```

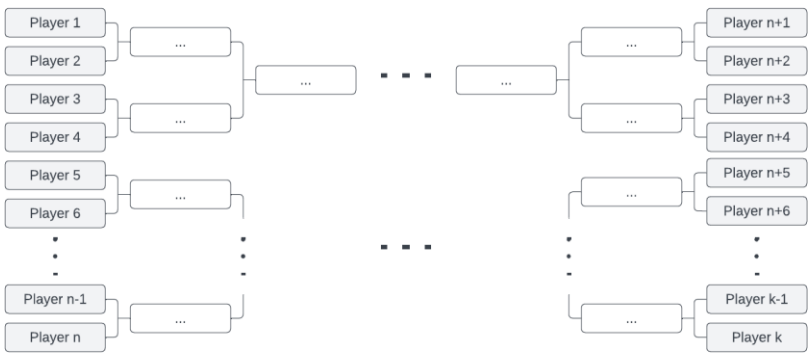
gdzie: a1, a2 oraz s1, s2 to - odpowiednio - liczba asów serwisowych graczy 1 i 2, liczba wygranych setów graczy 1 i 2.

Deserializację oraz serializację danych wykonano zgodnie z paradygmatem OOP, opracowywując klasy-wrappery pozwalające na parsowanie, operowanie na w postaci obiektów - instancji odpowiednich klas, a ostatecznie na serializację do postaci JSON, wszystkich w.w. struktur danych.

Struktura rozwiązania

Wynikiem działania algorytmu jest ustawiona drabinka turniejowa. W strukturze projektu jest to List[GeneratedMatch] gdzie GeneratedMatch to lista dwóch zawodników wraz z ich statystykami ostatnich spotkań. Dodatkowo każdy z osobników posiada parametr *lifetime*, który odpowiada za długość życia danego osobnika w algorytmie. Jeśli osobnik nie zmutuje ani nie zostanie odrzucony przez selekcje, to po określonej ilości iteracji zostanie usunięty z rozwiązania.

W przedstawionym poniżej rozwiązaniu interesuje nas dobranie *k* zawodników (player) w pary, czyli generacja pierwszego strzebla drabinki turniejowej. Kolejne etapy drabinki nie muszą być wynikiem wygranego meczu między dwoma zawodnikami. Każdy ze strzebli może zostać wygenerowany ponownie przy pomocy stworzonego algorytmu bazując dodatkowo na historii przebiegu spotkań we wcześniejszym etapie spotkania.



Elementy oznaczone szarym kolorem są istotnym rozwiązaniem, natomiast kolejne etapy oznaczone '…' są już uzależnione od decyzji dotyczącej przebiegu całego turnieju.

Funkcja celu

Dla każdego z zawodnika $\boldsymbol{z_i}$ na podstawie $\boldsymbol{k}$ ostatnich meczów, z których posiadamy statystyki takie jak $\boldsymbol{a1}, \boldsymbol{a2}$ - liczby trafionych asów podczas spotkania oraz $\boldsymbol{s1}, \boldsymbol{s2}$ - liczba wygranych setów wyznaczamy 3 parametry, które będą istotne w funkcji celu:

$\boldsymbol{as}$ – Współczynnik wygranych setów *(Avarange_stes_win_ratio)*

$$\boldsymbol{as_{z_1}} = \frac{\sum_{\boldsymbol{i=1}}^{\boldsymbol{k}} (\boldsymbol{s1_i} - \boldsymbol{s2_i})}{\boldsymbol{k}}$$

$\boldsymbol{aa}$ – Średnia wykonanych asów *(Averange_aces_played)*

$$\boldsymbol{aa_{z_1}} = \frac{\sum_{\boldsymbol{i=1}}^{\boldsymbol{k}} \boldsymbol{a1_i}}{\boldsymbol{k}}$$

$\boldsymbol{aw}$ – Współczynnik wygranych meczów *(Averange_of_win_matches)*

$$(s1_i - s2_i) < 0 \rightarrow w1_i = 0 - przegrana$$
$$(s1_i - s2_i) = 0 \rightarrow w1_i = 1 - remis$$
$$(s1_i - s2_i) > 0 \rightarrow w1_i = 2 - wygrana$$

$$\boldsymbol{aw_{z_1}} = \frac{\sum_{\boldsymbol{i=1}}^{\boldsymbol{k}} \boldsymbol{w1_i}}{\boldsymbol{k}}$$

Na podstawie tych parametrów, a także posiadanego przez zawodnika rankingu ELO wyznaczamy wartość przystosowania dla konkretnego meczu:

$$\boldsymbol{m_{ij}} = \boldsymbol{W_{elo}} \cdot \left|\boldsymbol{elo_{z_i}} - \boldsymbol{elo_{z_j}}\right| + \boldsymbol{W_{as}} \cdot \left|\boldsymbol{as_{z_i}} - \boldsymbol{as_{z_j}}\right| + \boldsymbol{W_{aa}} \cdot \left|\boldsymbol{aa_{z_i}} - \boldsymbol{aa_{z_j}}\right| + \boldsymbol{W_{aw}} \cdot \left|\boldsymbol{aw_{z_i}} - \boldsymbol{aw_{z_j}}\right|$$

gdzie:

m_{ij} – wartość przystosowania na podstawie meczu pomiędzy zawodnikami: $\boldsymbol{z_i}$ i $\boldsymbol{z_i}$
W_{elo} – waga różnicy *elo* pomiędzy graczami
W_{as} – waga współczynnika wygranych setów pomiędzy graczami
W_{aa} – waga wykonanych asów pomiędzy graczami
W_{aw} – Waga współczynnika zwycięstw pomiędzy graczami

Funkcja celu to suma wartości przystosowania dla wszystkich meczów które zostaną rozegrane:

$$f_c = \sum_{i=1}^{l} m_i$$

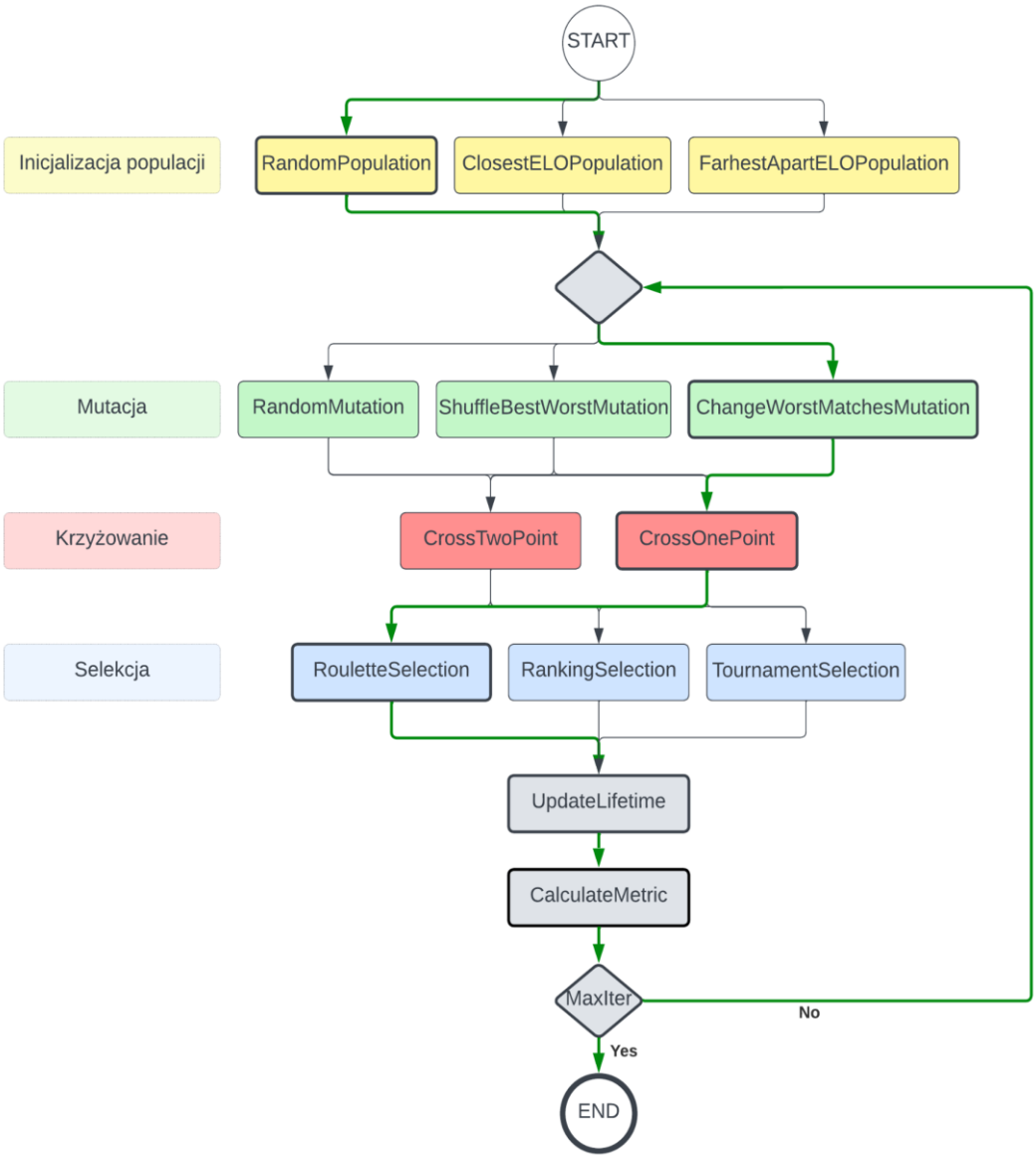
gdzie:

l – liczba zawodników podzielona przez 2

Propozycja rozwiązania – prezentacja własności algorytmu

Rozwiązanie przy pomocy algorytmu ewolucyjnego składa się z głównych kroków takich jak inicjalizacja populacji początkowej, operatory mutacji i krzyżowania oraz selekcja osobników do nowej populacji. Każdy z tych operatorów zostanie dokładnie omówiony w tym rozdziale.

Algorytm posiada warunek stopu w przypadku osiągnięcia maksymalnej liczby iteracji, ze względu na wprowadzony *lifetime*, przy dużej liczbie iteracji algorytm powinien radzić sobie z wyjściem z minimum lokalnego co pozwala na długie działanie algorytmu bez poprawy w ostatnich *n* iteracjach.



Na powyższym schemacie przedstawiono wszystkie dostępne operatory algorytmu ewolucyjnego. Ścieżka oznaczona kolorem zielonym jest to jedna z możliwości wyboru operatorów w celu otrzymania najlepszych rezultatów. Każdy z operatorów posiada swoje własne parametry, które również powinny zostać dostosowane do badanego problemu.

Tworzenie populacji

Pierwszym krokiem w procesie optymalizacji metodą algorytmu ewolucyjnego, jest odpowiednie dobranie problemu początkowego. Ważne jest, aby wybrać jak najbardziej różnorodne rozwiązania co spowoduje lepsze rezultaty algorytmu. Jeśli utworzymy zbiór populacji z tych samych osobników niekoniecznie otrzymamy zadowalające nasz rezultaty.

W realizowanym problemie wykorzystano dwa typy tworzenia populacji, jedna tworzy populację losowo, druga wykorzystuję specyfikę zawodników, czyli ich ELO.

1. RandomPopulation - jest to metoda tworząca losowe osobniki w populacji. Na podstawie listy osobników w losowy sposób przydziela osobniki w mecze. Prezentowana metoda pozwala na uzyskanie bardzo złych jak i dobrych rozwiązań w populacji początkowej.

2. ClosestELOPopulation - jest to metoda, która wykorzystuje jedną z wartości wchodzącą do funkcji celu. Zawodnicy zostali posortowani w kolejności względem parametru ELO. Następnie zostają dobrane ze sobą w pary. Pierwszy osobnik zawiera mecze, których różnica *elo* między zawodnikami jest najmniejsza. Kolejne osobniki wybierane z są z listy wybierając co 2, co 3 itd. element do pary:

$$S_1 = [M(p_1, p_2), M(p_3, p_4), M(p_5, p_6), M(p_7, p_8), \dots M(p_{k-1}, p_k)]$$
$$S_2 = [M(p_1, p_3), M(p_5, p_7), \dots M(p_{\frac{k-3}{2}}, p_{\frac{k-1}{2}}), M(p_2, p_4), M(p_6, p_8), \dots M(p_{\frac{k-2}{2}}, p_{\frac{k}{2}})]$$
$$S_3 = [M(p_1, p_4), \dots M(p_{\frac{k-5}{3}}, p_{\frac{k-2}{3}}), M(p_2, p_5), \dots M(p_{\frac{k-4}{2}}, p_{\frac{k-1}{2}}), M(p_3, p_6), \dots M(p_{\frac{k-3}{3}}, p_{\frac{k}{3}})]$$

Powyższy przykład pokazuje wyłącznie 3 kolejne osobniki (S_i) utworzone tą metodą, gdzie $M(\dots)$ to mecz a p_i to kolejni zawodnicy ze zbioru danych. Po przekroczeniu osobnika o numerze $n = \frac{k}{2}$ osobniki zostają generowane od nowa. Dla małych zbiorów zawodników generuje to powtarzalność osobników w populacji.

3. FarthestApartELOPopulation - metoda tworzy populację podobnie do drugiej metody jednak mecze wybierane są na podstawie największych różnic ELO między zawodnikami. Metoda ma na celu generowanie gorszych rozwiązań w populacji, aby zminimalizować szanse wpadnięcia w lokalne minimum na początku działania algorytmu.

Mutacja

Następnym krokiem algorytmu ewolucyjnego jest przeprowadzenie mutacji na poszczególnych osobnikach. Polega ona na wprowadzeniu zmian w genotypie osobników populacji. Zapewnienie takiej różnorodności genetycznej, może prowadzić do odkrycia lepszych rozwiązań w trakcie procesu ewolucji. Dodatkowo dzięki temu operatorowi genetycznemu dbamy o to, aby algorytm nie wpadł w lokalne ekstremum, które prowadziłoby do zakończenia procesu poszukiwania rozwiązania przed osiągnięciem optymalnego rozwiązania.

W projekcie stworzono trzy typy mutacji. Jedna z nich działała całkowicie losowo, druga wprowadzała zmiany wśród najlepszego i najgorszego rozwiązania natomiast trzecia starała się poprawiać zbieżność algorytmu.

1. RandomMutation - jest to metoda, która zmienia graczy w wylosowanych meczach w sposób losowy. Jako parametr przyjmuje “n” par między którymi dokona zamiany graczy. Parametr wejściowy reprezentuje ilość par, a nie informacje o tym, które dokładnie pary mają zostać poddane zamianie graczy. Gdy funkcja wylosuje “n” meczy z danego osobnika wtedy wśród tych meczy dokonywana jest losowa zmiana pomiędzy graczami.

2. ShuffleBestWorstMutation - jest to metoda, która ma za zadanie zamienić ze sobą graczy w najlepszej i najgorszej parze meczowej. Jedna osoba z najgorszej pary jest wymieniana na osobę z najlepszej pary i odwrotnie.

3. ChangeWorstMatchesMutation - jest to metoda, która zamienia pary meczowe tak aby dopasować graczy pod względem ich ELO. Parametrem wejściowym do funkcji jest liczba par, w których ma nastąpić zamiana. Pary meczowe wybierane są na podstawie ich wartości funkcji celu. Jako że najgorsze dopasowanie zawodników w meczu określamy wysoką wartością funkcji celu to właśnie takie mecze wybieramy do późniejszej zamiany. Następnie tworzymy nowe pary w miejsce starych w osobniku dobierając zawodników pod względem zbliżonej wartości ich ELO. Jako że wartość ELO odgrywa dużą rolę w obliczaniu funkcji celu, to dzięki zastosowaniu takiego typu mutacji możemy poprawić zbieżność całego algorytmu.

Krzyżowanie

Kolejnym krokiem w algorytmie jest przeprowadzenie krzyżowania w populacji. Ze względów implementacyjnych zdecydowaliśmy się na usuwanie rodziców z populacji i zastępowanie ich dziećmi. Potencjalną stroną negatywną takiego rozwiązania jest możliwość pozbycia się rozwiązań lepszych ze zbioru i zastąpienie ich rozwiązaniami potencjalnie gorszymi. Ryzyko to jest niwelowane przez zapamiętywanie najlepszych wyników co populacje. Dzięki czemu nie tracimy rozwiązań potencjalnie najlepszych i jednocześnie zmniejszamy szanse algorytmu na zatrzymanie się w lokalnym minimum.

Do operacji krzyżowania losowane jest 40% osobników w populacji. W losowaniu każdy osobnik ma równą szansę na zostanie wybranym. Następnie, również całkowicie losowo, wybrane 40% populacji jest parowane ze sobą w celu utworzenia par rodziców.

Kluczowym elementem krzyżowania dla naszego przypadku było zapewnienie dopuszczalności rozwiązań. W tym celu zastosowana została wariacja na temat krzyżowania PMX. Dla naszego rozwiązania kolejność par w wytworzonym osobniku nie ma znaczenia, lecz na rzecz poprawnego krzyżowania założone zostało, że ma. Po wylosowaniu indeksów krzyżowania dla osobników w oryginalnej formie genotypu przechodzimy do zmiany reprezentacji na rzecz zapewnienia dopuszczalności rozwiązań. Każda para w każdym z dwóch rodziców rozłożona jest na gracza pierwszego oraz gracza drugiego, którzy zostają kolejno dopisani do listy graczy odpowiednio rodzica pierwszego oraz rodzica drugiego. Dalej stosujemy krzyżowanie PMX na listach graczy rodziców. W ten sposób, jeżeli po wymianie wynikającej z krzyżowania two-point lub one-point wykryty zostanie powtarzający się gracz, dzięki słownikowi utworzonemu na bazie metody PMX możemy podmienić powtarzającego się gracza. Takie rozwiązanie daje nam pewność, że wszyscy gracze pojawią się w potomkach, ale także zapewnia, że każdy gracz pojawi się dokładnie raz. Jednocześnie takie podejście zapewnia niezmienną liczbę meczy.

Po utworzeniu list graczy potomków, dokonana zostanie zmiana z list graczy, na ustalony w projekcie genotyp osobnika. Gracze parowani są ze sobą, gdzie gracz z n-tym indeksem w liście parowany jest z graczem z indexem n+1, zaczynając od n równego zero. Z tych par tworzone są obiekty GeneratedMatch, które następnie są łączone w listy. Utworzone listy to odpowiednio potomek pierwszy oraz potomek drugi.

Ostatecznie potomkowie zastępują rodziców w populacji. Rodzice zostają usunięci na rzecz nowych osobników, tak, aby liczebność populacji została bez zmian.

Opis operatorów krzyżowania:

Dla obydwu z powyższych wylosowane indeksy nie mogą być sobie równe - tak, aby na pewno zaszło krzyżowanie osobników.

1. CrossOnePoint - wyżej wymieniony mechanizm krzyżowania, gdzie losowany jest tylko jeden indeks do krzyżowania - drugi domyślnie ustawiony jest na początek listy.

2. CrossTwoPoint - wyżej wymieniony mechanizm krzyżowania, gdzie losowane są obydwa indeksy do krzyżowania.

Dla obydwu z powyższych wylosowane indeksy nie mogą być sobie równe - tak, aby na pewno zaszło krzyżowanie osobników.

Selekcja

Aby zapewnić optymalizację funkcji celu, a więc generowanie w kolejnych iteracjach algorytmu coraz lepszych rozwiązań niezbędna jest ocena przystosowania poszczególnych osobników tworzących populację. Istnieje wiele różnych metod selekcji osobników - część z nich w głównej mierze opiera się na losowości, inne w bardziej restrykcyjny sposób dokonują wyboru najlepiej

przystosowanej części populacji. W projekcie zaimplementowane zostały następujące strategie dokonywania selekcji:

1. Roulette Selection (Selekcja Kołem Ruletki) - wszystkie osobniki z populacji umieszczane są na kole ruletki, a część jaka przypada danemu osobnikowi jest wprost proporcjonalna do wartości jego funkcji przystosowania. W następnym kroku następuje losowanie osobników, aż do wyboru populacji równolicznej z populacją wejściową. Jest to najbardziej popularna z metod selekcji, posiada jednak zasadniczą wadę - jeżeli już na początku działania algorytmu pojawi się osobnik zdecydowanie dominujący i zajmujący większość koła ruletki to wyprze on z populacji wszystkie inne osobniki, co może skutkować utknięciem algorytmu w lokalnym minimum.

2. Ranking Selection (Selekcja Rankingowa) - w tym podejściu osobniki również umieszczane są na kole ruletki, jednak część koła jaka przypada danemu osobnikowi nie jest proporcjonalna do wartości funkcji przystosowania, a do pozycji jaką zajmuje on wśród pozostałych osobników:

	Osobnik 1	Osobnik 2	Osobnik 3
Wartośc funkcji przystosowania	7	2	1
Miejsce w rankingu	1	2	3
Procent koła w selekcji ruletkowej	70	20	10
Procent koła w selekcji rankingowej	50	33.(3)	16.(6)

Dodatkowo selekcja rankingowa umożliwia umieszczenie na kole ruletki jedynie określonej części spośród najlepszych osobników - w ten sposób użytkownik może kontrolować szybkość zbieżności algorytmu.

3. Tournament Selection (Selekcja Turniejowa) - osobniki z populacji wejściowej są dzielone losowo na podgrupy o liczebności określonej przez użytkownika. Do następnej populacji wybierany jest najlepszy osobnik z każdej podgrupy.

Aplikacja

W ramach projektu stworzyliśmy dedykowaną stronę internetową, która umożliwia użytkownikowi załadowanie danych na podstawie, których generowana jest kolejna runda turnieju oraz wizualizację wyniku działania algorytmu.

Na podstawie przeprowadzonych testów algorytmu, ustaliliśmy optymalne parametry, które gwarantują najlepsze wyniki. Dzięki temu użytkownik może korzystać z gotowego rozwiązania, bez konieczności dokonywania wyborów, które mogłyby być dla niego skomplikowane bez znajomości podstaw algorytmów ewolucyjnych.

Możliwość zmiany operatorów genetycznych oraz ich parametrów została dodana w pliku config w strukturze projektu, aby w łatwy sposób przeprowadzić analizę działania algorytmu. Wszystkie testy zostały wykonane na podstawie losowo wygenerowanych danych testowych.

Generowanie danych testowych

Z powodu braku dostępności danych na niebudzącej wątpliwości licencji, zdecydowano się na wygenerowanie zbioru danych. Wykonano skrypt, pozwalający na generowanie scenariuszy turniejowych w zależności od dwóch parametrów: ilości graczy oraz ilości rund. Korzystając z pakietu faker, skrypt opracowuje listę graczy (obiektów - instancji klasy Player) o losowych danych osobowych wraz z losowym wynikiem ELO.

W drugim kroku iteracyjnie losowane są mecze w kolejnych rundach, a co rundę lista graczy jest mieszana dla zwiększonej randomizacji. W każdej z rund generowana jest ilość meczy dla każdej pary graczy bądź - w przypadku nieparzystej ich ilości - dla par i jednego zawodnika zwyciężającego walkowerem. Każdemu z meczy przypisywane są wylosowane statystyki: asów każdego z graczy, jak i wyniku setowego dla każdego z nich. Zwycięzcą naturalnie jest gracz o większej liczbie setów; może

zajść również remis. Dane zapisywane są następnie do pliku. Passa (ilość wygranych lub zremisowanych meczów) każdego z graczy nie są zawarte w danych wejściowych, ale wyliczane z danych na późniejszym etapie.

Ustawienie parametrów algorytmu - zmienne procesowe

Do konfiguracji w algorytmie są zarówno typy wybranych metod (np. sposób generacji populacji startowej) jak i ich parametry (np. maksymalna długość życia osobnika w selekcji). Zmian możemy dokonywać poprzez zmiany w pliku config.yaml, gdzie w odpowiednich miejscach wpisujemy wybrane przez nas wartości. Opis kolejnych konfiguracji działania algorytmu poprzez plik config:

1. Tworzenie populacji
   a. ***make_population_type***: wybór rodzaju tworzenia populacji, do wyboru
      - **"RandomPopulation"** (tworzenie zupełnie losowe),
      - **"ClosestELOPopulation"** (tworzenie populacji startowej, gdzie parowani są gracze o zbliżonym *elo*)
      - **"FarthestApartELOPopulation"** (tworzenie populacji startowej, gdzie parowani są gracze o znacząco różnym od siebie ELO)
   b. ***numer_of_specimens***: liczba dodatnia całkowita, zdefiniowanie ilości osobników w populacji, liczba ta jest stała przez całość trwania algorytmu - żadna z późniejszych operacji nie zmienia liczebności populacji
   c. ***percent_mutation_in_population*****:** podawana jako liczba z zakresu 1-100, definiuje jaki procent populacji może zostać poddana mutacji

2. Mutacja
   a. ***mutation_type***: rodzaj typu mutacji, do wyboru
      - **"RandomMutation"** (zamienia graczy losowo pomiędzy n wybranymi meczami),
      - **"ShuffleBestWorstMutation"** (mutacja zmieniająca graczy z najlepszych meczy oraz najgorszych meczów),
      - **"ChangeWorstMatchesMutation"** (mutacja zmienia mecze z najgorszą wyliczoną wartością funkcji celu - rozdziela graczy, aby następnie sparować ich na podstawie ELO)
   b. ***numberOfPairs***: liczba dodatnia całkowita, ilość par meczów, które ulegną mutacji (e.g. ile zostanie przelosowanych między sobą lub zmieniony w odpowiedni sposób do wybranej mutacji)

3. Krzyżowanie
   a. ***crossover_type***: rodzaj przeprowadzonego krzyżowania, do wyboru
      - **"OnePoint"** (krzyżowanie one-point-like z wariacją na temat PMX)
      - **"TwoPoint"** (krzyżowanie two-point-like z wariacją na temat PMX)
   b. ***percent_crossover_in_population:*** podawana jako liczba z zakresu 1-100, definiuje jaki procent populacji zostaje poddany krzyżowaniu.

4. Selekcja:
   a. *selection_type*: rodzaj przeprowadzanej selekcji, do wyboru
      - **"RouletteSelection"** (selekcja ruletkowa)'
      - **"RankingSelection"** (selekcja rankingowa),
      - **"TournamentSelection"** (selekcja turniejowa)
   b. ***clash_participants_number***: całkowita dodatnia liczba, liczba osobników do turnieju
   c. ***maximum_lifetime***: całkowita dodatnia liczba, maksymalna długość życia osobnika w niezmienionej formie, osobniki żyjące dłużej niż parametr nie będą brane pod uwagę w selekcji
   d. ***percent_selected_from_best:*** całkowita dodatnia liczba, określa jaka część spośród najlepszych osobników populacji weźmie udział w selekcji rankingowej.

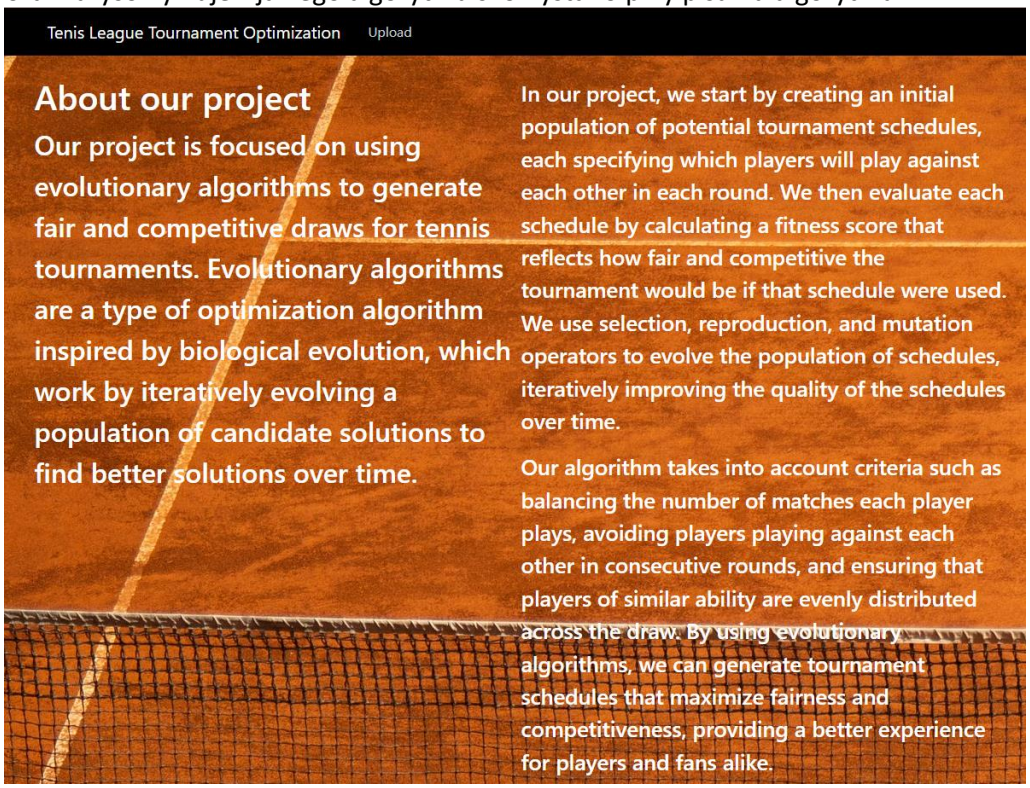
5. Parametry odpowiedzialne za przebieg algorytmu:
   a. ***iterations***: liczba iteracji działania algorytmu

Opis działania aplikacji

Tworząc stronę internetową chcieliśmy, aby była przystosowana do użytkowania jej przez osoby niezwiązane z algorytmiką, a jedynie chcące skorzystać z przygotowanego przez nas algorytmu. Taki użytkownik ma dostępne trzy elementy reprezentowane w formie zakładek na stronie:

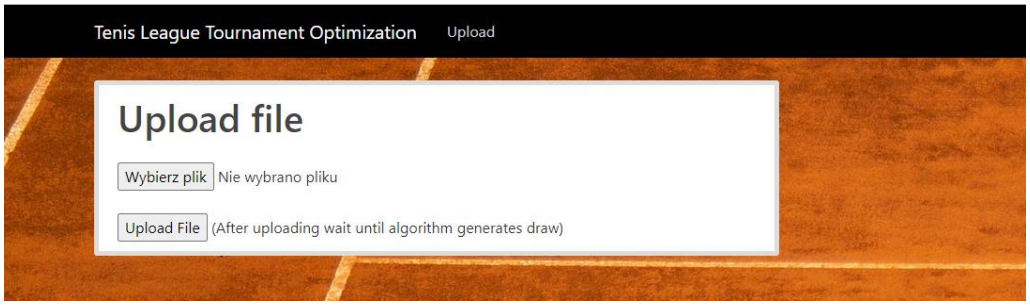
1. Informacje o projekcie
2. Wgrywanie pliku .json
3. Wizualizacja wyników

Pierwsza zakładka, która wyświetla się po uruchomieniu strony opowiada o tym czego dotyczy projekt oraz zarysowywuje z jakiego algorytmu skorzystano przy pisaniu algorytmu.



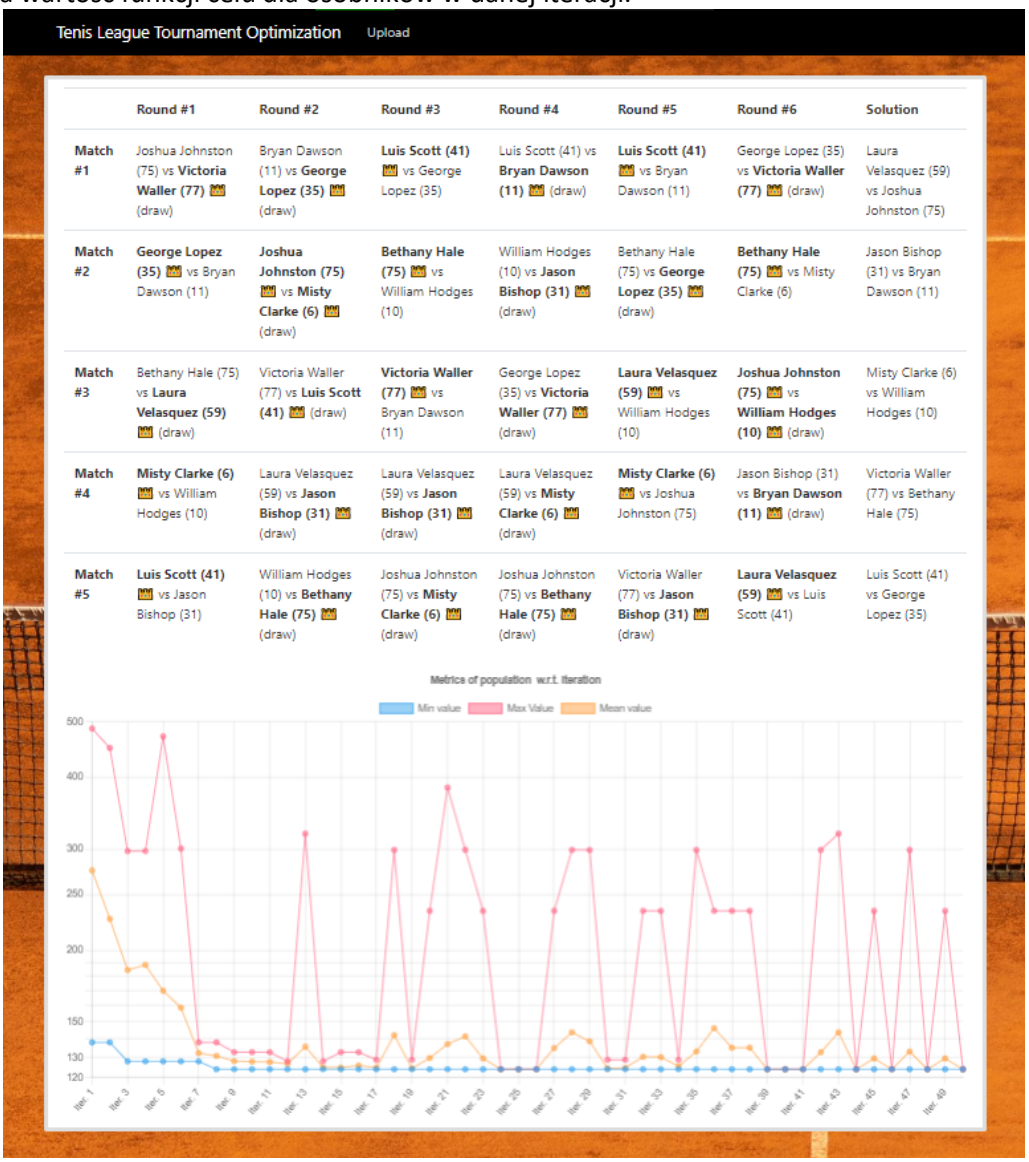
Rys1. Zakładka z informacjami o projekcie

W drugiej zakładce "Upload" istnieje możliwość wgrania pliku z danymi zawodników na podstawie, których generowane jest rozwiązanie, czyli rozpiska meczy. Po kliknięciu w "Wybierz plik" i przesłaniu pliku należy kliknąć "Upload File" i poczekać aż program wylczy najoptymalniejsze rozwiązanie.



Rys2. Zakładka wgrywania pliku z danymi uczestników

Następnie wyświetla się zakładka, w której widoczne są wcześniejsze rundy turnieju oraz nasze rozwiązanie w kolumnie “Solution”. Każdy z graczy ma podaną przy swoim imieniu i nazwisku wartość jego ELO. We wcześniejszych rundach gracze, którzy wygrali swój mecz są oznaczeni emotikonem korony oraz ich dane są pogrubione. Dodatkowo został umieszczony wykres poszczególnych metryk populacji w zależności od numeru iteracji algorytmu. Wśród metryk jest minimalna, maksymalna i średnia wartość funkcji celu dla osobników w danej iteracji.



Rys3. Wizualizacja wyników

Wykorzystane narzędzia

Do wykonania aplikacji zespół skorzystał ze środowiska Visual Studio Code, runtime’u oraz języka Python 3.10, a także z następujących bibliotek:

- pyyaml - do serializacji/deserializacji hiperparametrów algorytmu oraz konfiguracji aplikacji / serwera
- numpy - do obliczeń numerycznych oraz generowania danych
- flask - do uruchomienia serwera HTTP dla API REST oraz aplikacji webowej
- WTForms, Flask-WTF - warstwy abstrakcji do opracowywania formularzy webowych w Flask
- Jinja - silnik szablonów HTML dla Flaska do aplikacji webowej
- Faker - do generowania realistycznych danych testowych (imiona, nazwiska)

Styl kodu był utrzymany dzięki linterowi autopep8, a kod oraz historia działań zespołu były utrzymane w repozytorium GIT. Uproszczoną dokumentację kodu m. in. z instrukcjami uruchomienia zapisano w pliku Markdown. Wykorzystane pakiety wraz z wersjami zrealizowano w pliku requirements.txt dla menedżera pakietów PIP.

Eksperymenty /testy skuteczności algorytmu

Aby sprawdzić poprawność działania naszego algorytmu w pełni zostały wykonany szereg testów skuteczności. Testy zostały podzielone na:

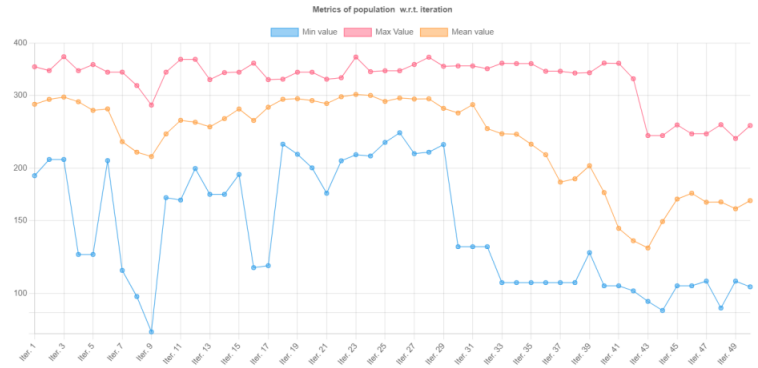
1. Testy poszukiwania wyniku docelowego przy różnych metodach:
   - tworzenia populacji,
   - selekcji,
   - krzyżowania,
   - mutacji
2. Testy dla różnych wielkości problemu
3. Testy dla różnych wag funkcji celu

1. Testy dla różnych metod

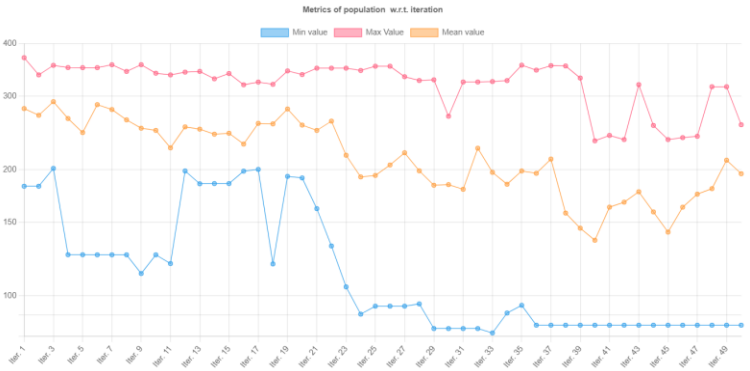
W celach testowych został przyjęty przykład, w którym chcemy dopasować 16 zawodników na podstawie ich ostatnich 8 rund.

Krzyżowanie

Przyjęte reguły: (Populacja: RandomPopulation, Mutacja: RandomMutation, Selekcja: RouletteSelection)

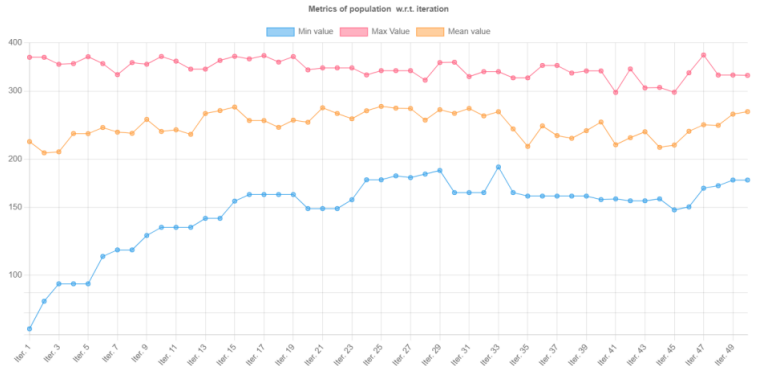


OnePoint *(Wynik najlepszy 81 Wynik końcowy 105)*

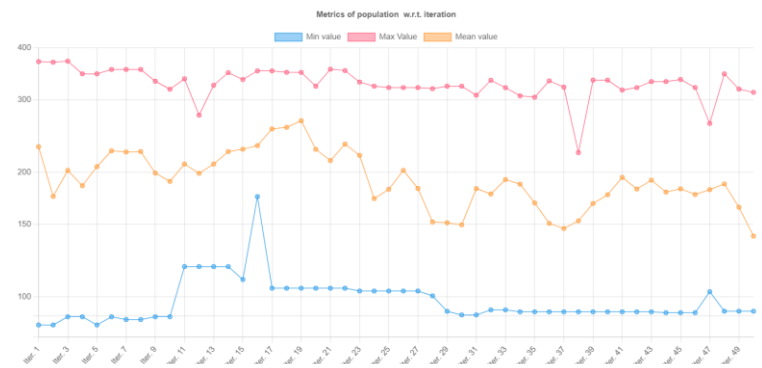


TwoPoint *(Wynik najlepszy 101 Wynik końcowy 105)*

Przyjęte reguły: (Populacja: ClosestELOPopulation, Mutacja: RandomMutation, Selekcja: RouletteSelection)



OnePoint (Wynik najlepszy 72 Wynik końcowy 170)

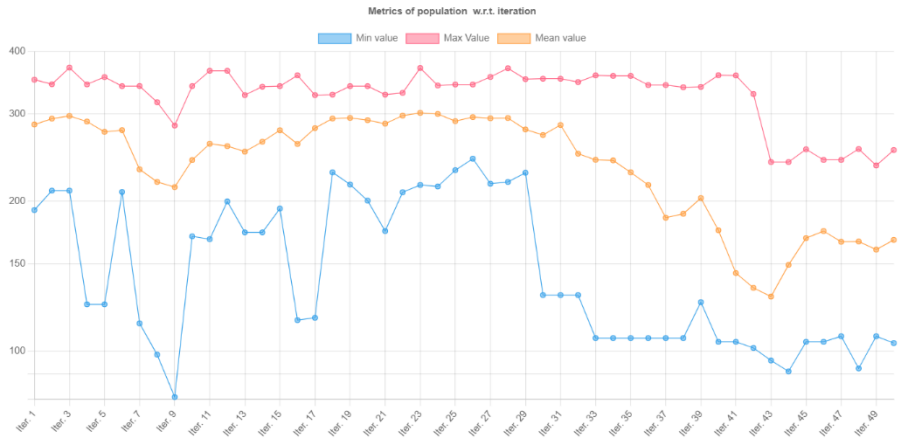


TwoPoint (Wynik najlepszy 72 Wynik końcowy 91)

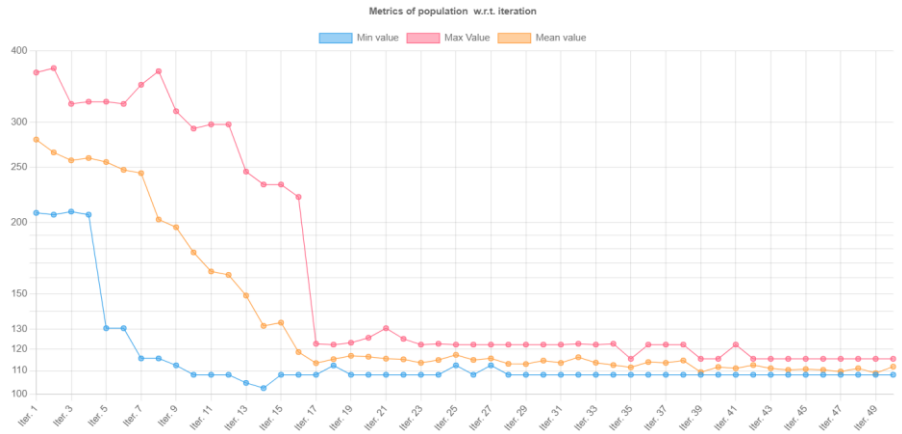
Jak możemy zaobserwować na powyższych wykresach krzyżowanie TwoPoint dużo lepiej radzi sobie ze znalezieniem wyniku suboptymalnego. Jego funkcja celu dąży do minimum dużo szybciej niż gdy algorytm korzysta z krzyżowania OnePoint. Metoda jednopunktowa nie zawsze też powoduje poprawę wyniku w kolejnej iteracji. Jak możemy też zaobserwować, kiedy korzystamy z Tworzenia populacji metodą ClosestELOPopulation metoda jednopunktowa odeszła od wyniku najlepszego swoimi krzyżowaniami, gdzie metoda dwupunktowa starała się znaleźć kolejne minimum.

Mutacja

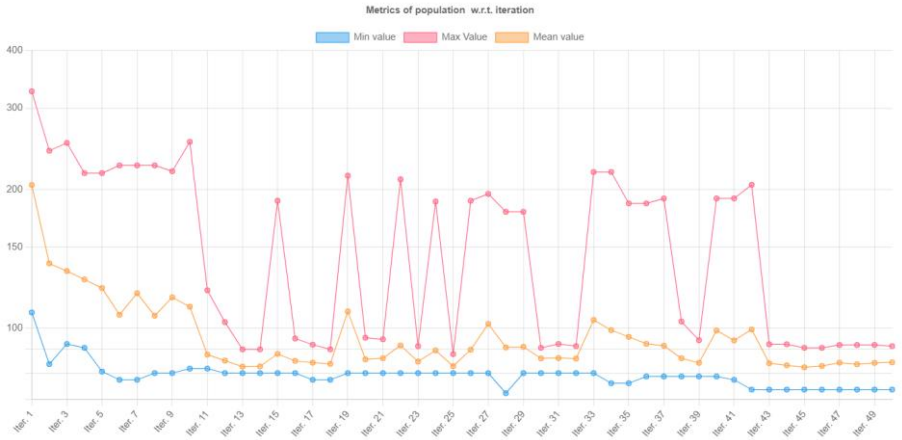
Przyjęte reguły: (Populacja: RandomPopulation, Krzyżowanie: OnePoint, Selekcja: RouletteSelection)



RandomMutation (Wynik najlepszy 81, Wynik końcowy 101)

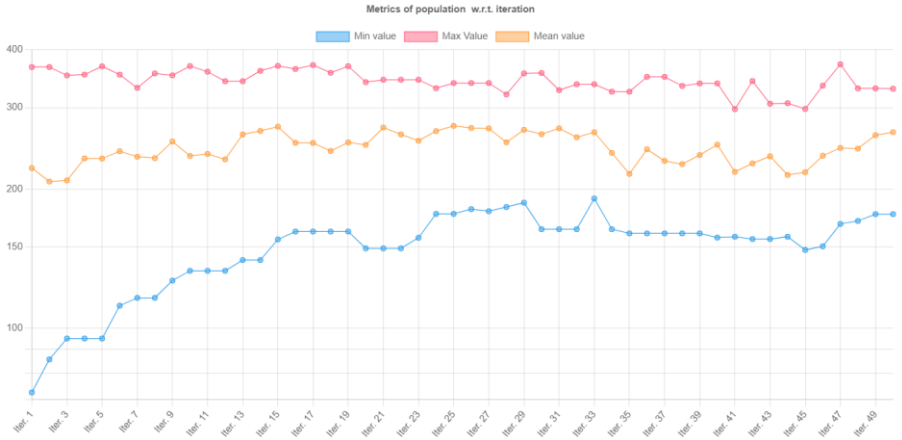


ShuffleBestWorstMutation (Wynik najlepszy 101, Wynik końcowy 110)

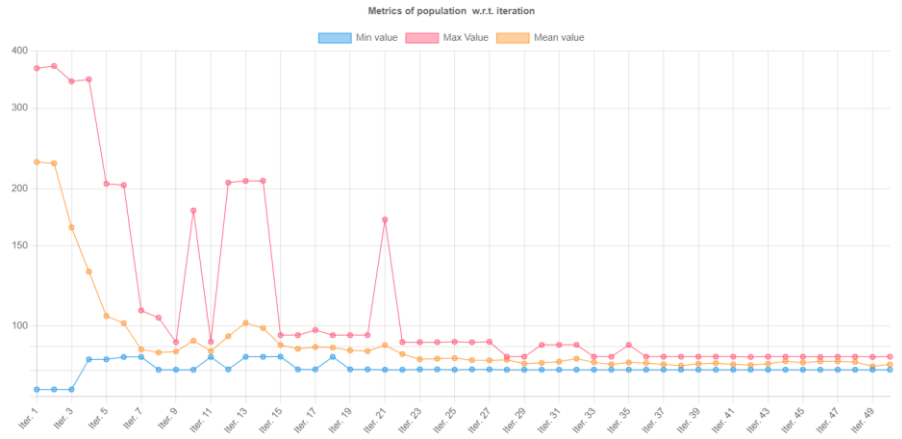


ChangeWorstMatchesMutation (Wynik najlepszy 73.2, Wynik końcowy 74.3)

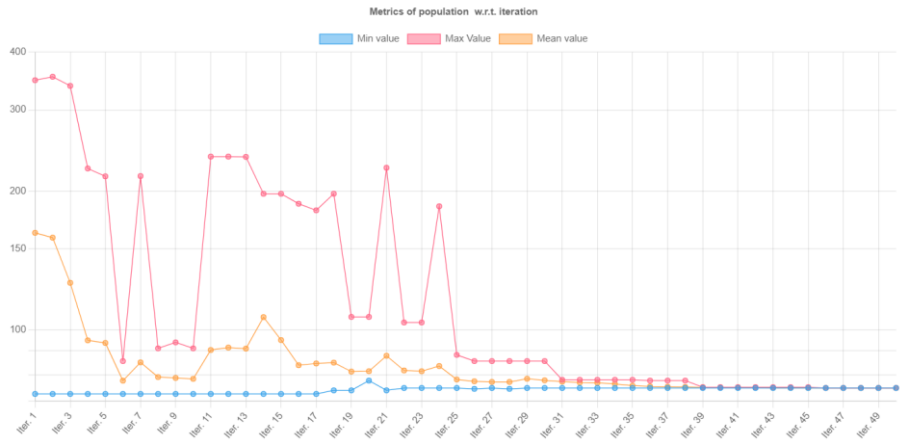
Przyjęte reguły: (Populacja: ClosestELOPopulation, Krzyżowanie: OnePoint, Selekcja: RouletteSelection)



RandomMutation (Wynik najlepszy 72, Wynik końcowy 150)



ShuffleBestWorstMutation (Wynik najlepszy 72, Wynik końcowy 81.2)



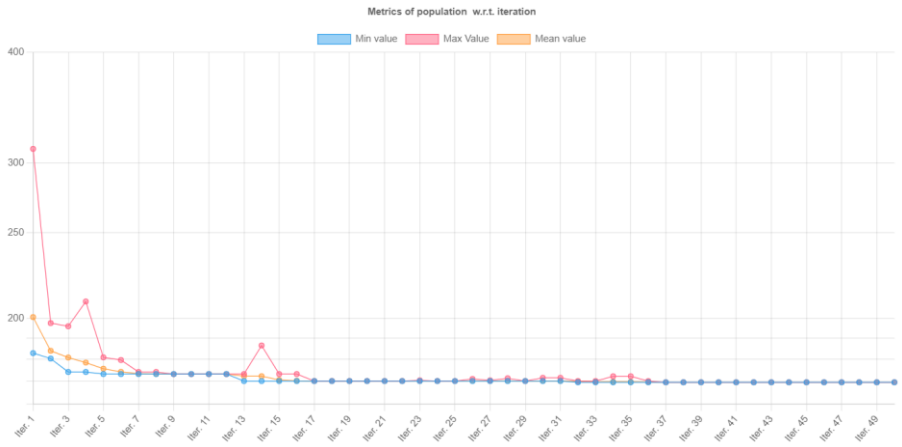
ChangeWorstMatchesMutation (Wynik najlepszy 72, Wynik końcowy 72)

Po porównaniu metod mutacji można stwierdzić, że najlepszą metodą jest Mutacja ChangeWorstMatchesMutation, która to poprawia za każdym razem tylko najgorszy mecz w serii przez co prowadzi najszybciej do zbieżności algorytmu. Mutacja ShuffleBestWorstMutation poprzez swoje działanie powoduje mniej kontrolowaną mutację, z której mogą być zarówno lepsze jak i gorsze wyniki jednak także prowadzi do zbieżności w jakimś optimum lokalnym. Natomiast mutacja RandomMutation poprzez swoją losowość może proponować nowe osobniki, które mogą się okazać rozwiązaniami lepszymi niż te które oferują pozostałe mutacje jednak nie jest to pewne. Natomiast

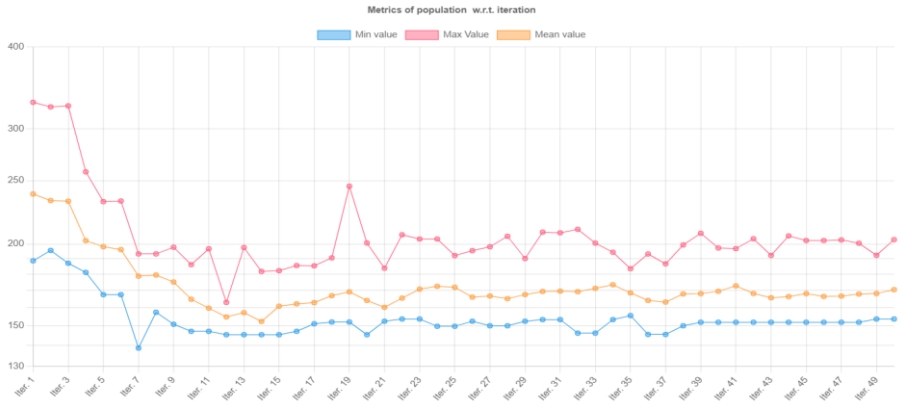
przy populacji startowej ClosestELOPopulation mutacja RandomMutation spowodowała oddalenie się od szukanego wyniku, gdzie pozostałe były nadal szukały wyniku suboptymalnego w bliskiej odległości od osobnika startowego.

Selekcja

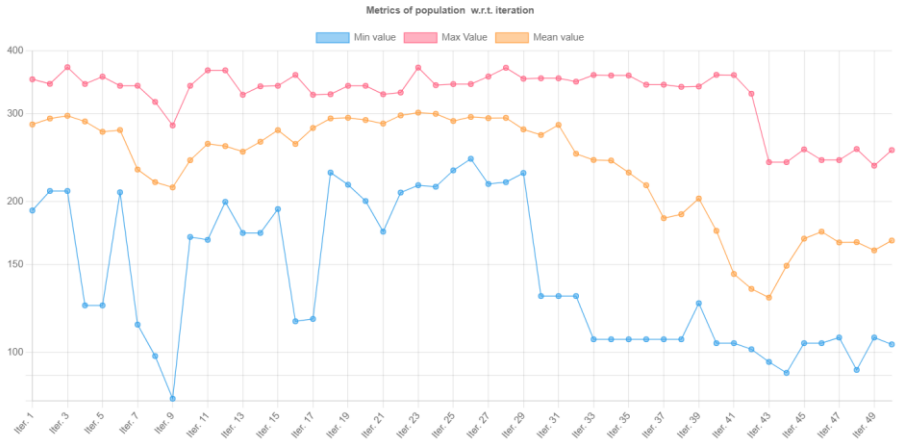
Przyjęte reguły: (Populacja: RandomPopulation, Mutacja: RandomMutation, Krzyżowanie: OnePoint)



RankingSelection (Wynik najlepszy 170, Wynik końcowy 170)

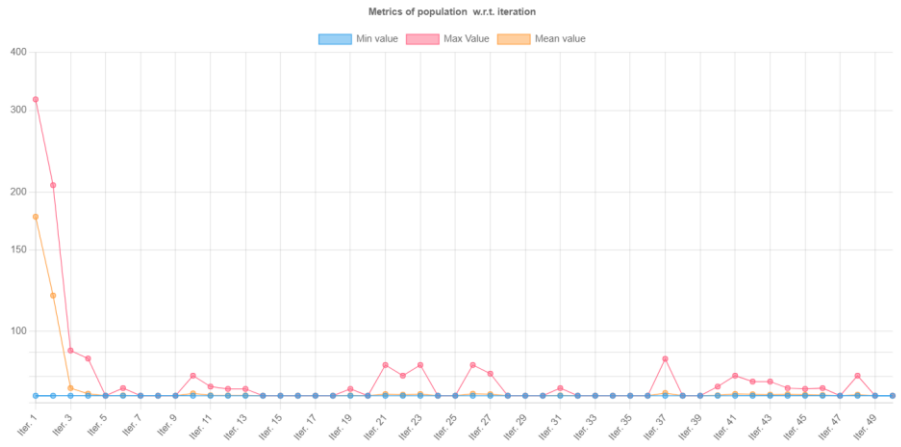


TournamentSelection (Wynik najlepszy 133, Wynik końcowy 152)

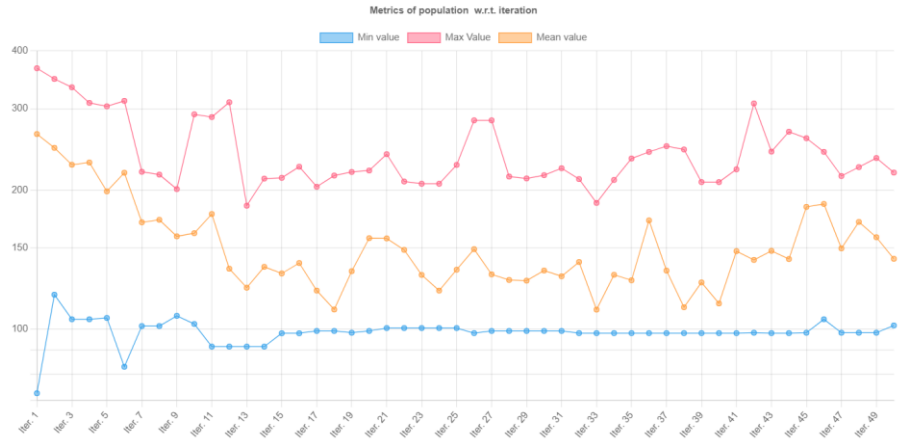


RouletteSelection (Wynik najlepszy 80, Wynik końcowy 105)

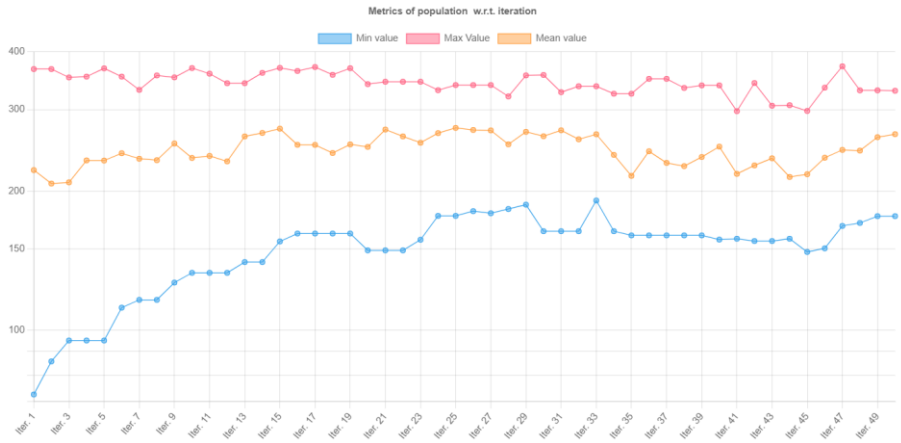
Przyjęte reguły: (Populacja: ClosestELOPopulation, Mutacja: RandomMutation, Krzyżowanie: OnePoint)



RankingSelection (Wynik najlepszy 72, Wynik końcowy 72)



TournamentSelection (Wynik najlepszy 72, Wynik końcowy 101)



RouletteSelection (Wynik najlepszy 72, Wynik końcowy 170)

Porównując trzy metody selekcji możliwe do wykorzystania w naszym algorytmie możemy stwierdzić, że metoda selekcji kołem ruletki jest najlepsza w przypadku, kiedy generujemy losową populację startową, natomiast nie radzi sobie dobrze przy specyficznej populacji, ponieważ może odrzucić wynik najlepszy. Metoda selekcji rankingowa jest najbardziej zbieżna i zatrzymuje się w pierwszym napotkanym minimum lokalnym przez co nie pozwala nam na poszukiwanie lepszych rozwiązań natomiast w momencie, kiedy trafi do właściwego utrzymuje je. Metoda Turniejowa jest

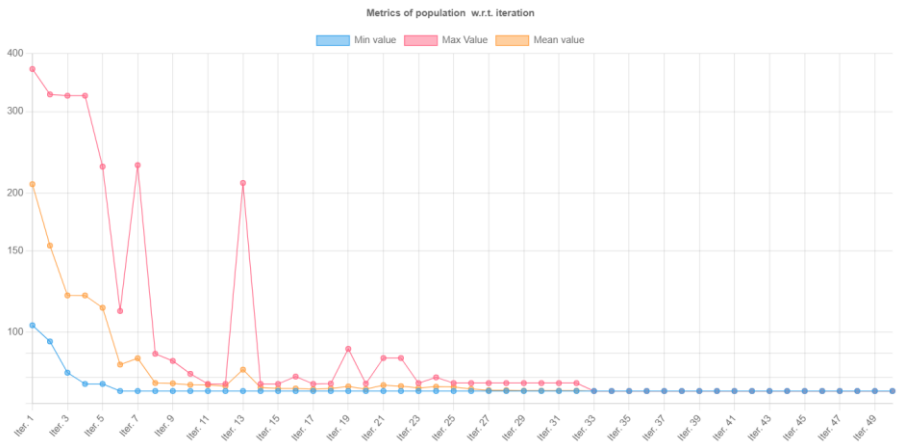
mniej agresywna niż metoda rankingowa, ponieważ utrzymuje najlepszy wynik, ale pozwala też na poszukiwanie nowego, ale nie w tak dużym stopniu jak metoda ruletki.

Populacja

Na podstawie powyższych wykresów można stwierdzić, że populacja generowana losowo pozwala na przeszukanie większej ilości minimum, a także nie zamyka się na pierwsze napotkane rozwiązanie. Natomiast w przypadku, kiedy chcemy wyszukać pary po rankingu ELO populacja ClosestELOPopulation przedstawia nam wynik optymalny i ciężko jest znaleźć lepszy. Jednak niekoniecznie sprawdzi się to w momencie, kiedy inne parametry naszej funkcji celu będą miały większą wagę oraz zastosowanie tej populacji startowej zamyka szansę niektórym metodą na znalezienie jeszcze lepszego wyniku.

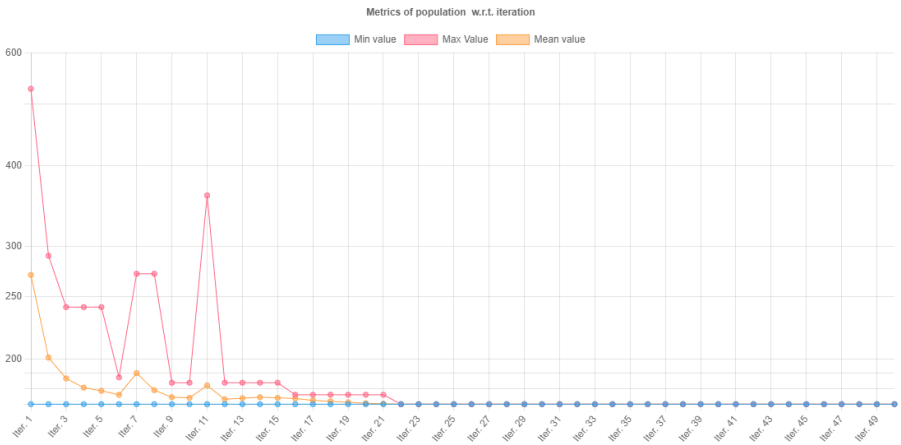
Najlepsze połączenie metod

Najlepszym połączeniem metod jest użycie populacji RandomPopulation, mutacji ChangeWorstMatchesMutation, krzyżowania TwoPoint oraz selekcji RouletteSelection, ponieważ pozwala ono na znalezienie rozwiązania suboptymalnego nie zamykając się od razu na pierwszy napotkany wynik, a także jest elastyczne dla różnych założeń użytkownika. Jeśli użytkownik dobierze inne wagi dla parametrów to powyższe połączenie metod powinno najbardziej sprostać jego oczekiwaniom.

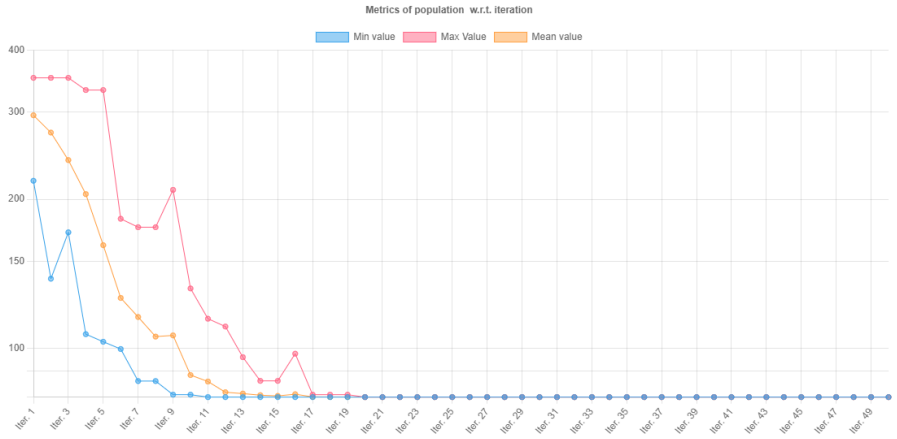


2. Testy dla różnych wielkości problemu

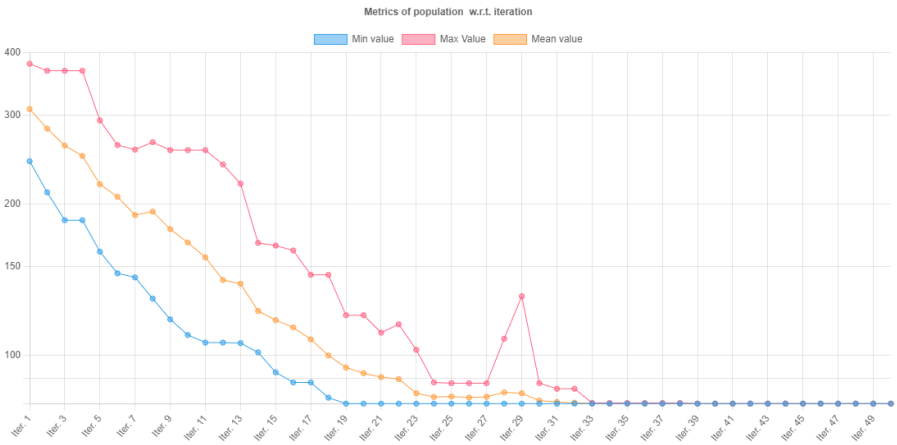
W tej części wykonano testy dla najlepszego zestawienia metod oraz różnych wielkości turniejów. Wybrano 10 rund oraz odpowiednie ilości zawodników: 10, 32 oraz 64.



Turniej 1: 10 zawodników



Turniej 2: 32 zawodników

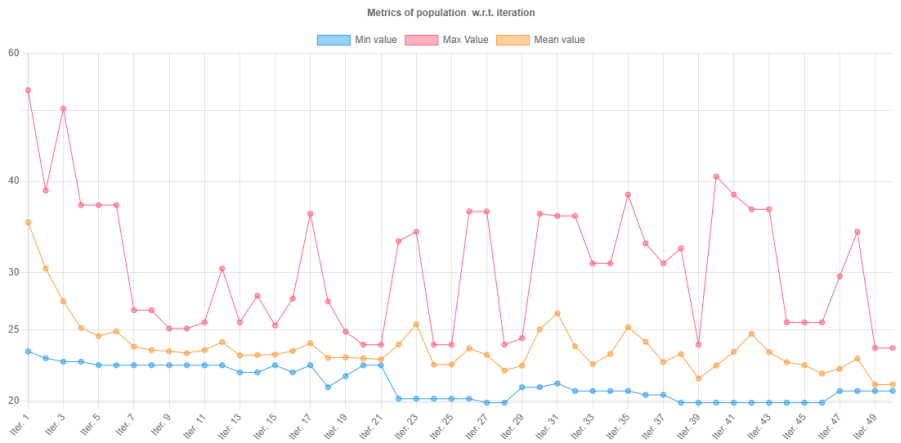


Turniej 3: 64 zawodników

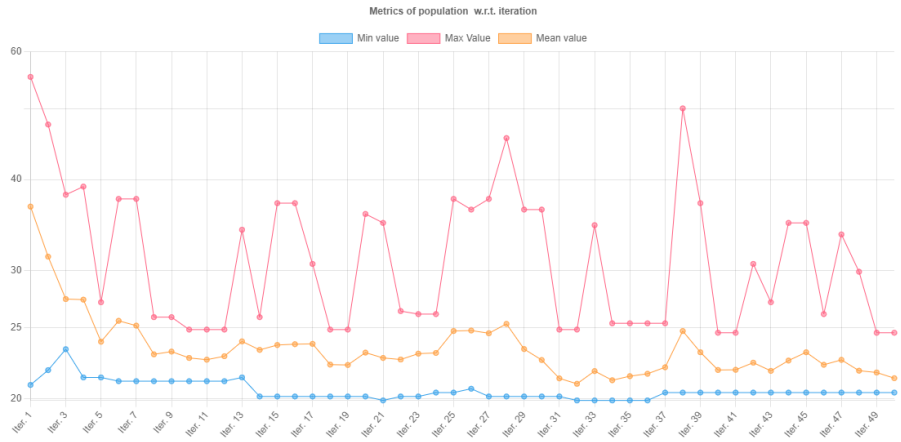
Jak można zauważyć program poradził sobie z wyznaczeniem par turniejowych dla dużej liczby zawodników i dla każdego z przypadków znalazł rozwiązanie suboptymalne które można byłby użyć przy przygotowaniu turnieju tenisowego.

3. Testy dla różnych wag funkcji celu

Kiedy wartość wagi *ELO* jest bardzo niska $W_{elo} = 1, W_{as} = 8, W_{aa} = 8, W_{aw} = 8$



Funkcja celu dla RandomPopulation



Funkcja celu dla ClosestELOPopulation

Pokazuje to, że dobranie wag ma duże znaczenia dla poszukiwanego wyniku, zastosowano tutaj minimalnej wagi dla ELO spowodowało, że inne parametry zaczęły mieć dużo większe znaczenie oraz ClosestELOPopulation już nie daje od razu najlepszego wyniku w pierwszej iteracji.

Podsumowanie / wnioski

Na podstawie otrzymanych wyników możemy stwierdzić, że algorytm działa poprawnie. Proponuje pary turniejowe pozwalają zmierzyć się zawodnikom o najbardziej zbliżonych warunkach, przez co oglądanie lub uczestniczenie w takich meczach może być interesujące.

Duże znaczenie na funkcję celu ma dobór odpowiednich wag parametrów. Użytkownik sam może tym sterować oraz położyć nacisk na najbardziej interesujący go element.

Dobór odpowiednich metod wyznaczenia populacji, mutacji, krzyżowania czy selekcji też leży w rękach użytkownika i zastosowanie każdej z nich może przynieść konkretne skutki. Optymalnym połączeniem, kiedy nie mamy konkretnych preferencji co do szybkości otrzymanego wyniku bądź zbieżności jest użycie metod populacji RandomPopulation, mutacji ChangeWorstMatchesMutation, krzyżowania TwoPoint oraz selekcji RouletteSelection.

Algorytm nie ma problemów z wyznaczeniem wyniku dla dużego przypadku natomiast jest to czasochłonne.

Spis literatury

1. Głębokie uczenie i inteligencja obliczeniowa. Materiały z wykładów. 2023.
2. Vijini Mallawaarachchi. Introduction to Genetic Algorithms — Including Example Code. Towards Data Science. 2017.
3. Ryan Champlin. Selection Methods of Genetic Algorithms. Olivet Edu. 2018.
4. Apar Garg. Crossover Operators in Genetic Algorithm. Medium. 2021.
5. Genetic Algorithms - Mutation. Tutorialspoint.

Podział pracy

Zagadnienie	Waga	Udział [%]	Wykonawca
Generacja przykładowego zbioru danych / struktura danych.	5	100%	Artur
Stworzenie klasy osobnika / struktura osobnika.	2	80%	Bartosz
		20%	Norbert
Dodanie czasu życia osobnika - lifetime	1	40%	Norbert
		20%	Bartosz
		20%	Maria
		20%	Michał
Obliczenie statystyk dla zawodnika / funkcja celu.	2	90%	Filip
		10%	Bartosz
Stworzenie metod generacji populacji początkowej / struktura populacji	3	100%	Bartosz
Mutacja osobników	4	100%	Maria
Krzyżowanie osobników	5	100%	Michał
Selekcja osobników	4	100%	Norbert
Debugowanie kodu	2	70%	Michał
		30%	Maria
Testy skuteczności algorytmu	5	90%	Filip
		10%	Norbert
Złożenie algorytmu / Stworzenie szablonu projektu	3	80%	Norbert
		10%	Artur
		10%	Bartosz
Wizualizacja wyników / aplikacja	5	50%	Maria
		50%	Artur
Zarządzanie projektem	2	100%	Bartosz
Przygotowanie sprawozdania	5	20%	Michał
		30%	Filip
		20%	Bartosz
		10%	Maria
		10%	Norbert
		10%	Artur

Powyższa tabela sprowadza się do przemnożeniu wag każdego zadania przez procentowy udział każdej z osób. Przyjęte wagi są wyłącznie naszą estymacją czasu oraz wkładu włożonego w wykonywanie przypisanych zadań. Wynikiem tego przeliczenia jest poniższa tabela, wartości przedstawione poniżej są wyłącznie estymacją i próbą zmierzenia nakładu pracy każdej z osób jednak nie odzwierciedla w 100% wkładu każdej osoby.

Podsumowanie udziału osób w projekcie	
Artur	**8,3**
Norbert	**8,2**
Bartosz	**8,3**
Maria	**7,8**
Michał	**7,6**
Filip	**7,8**