Piotr Frysz
Rafał Koguciuk
Bartosz Rajkowski

prowadzący: dr inż. Tomasz Trzciński

Podstawy sztucznej inteligencji

Sprawozdanie z projektu

AE.4 Masz 10 kart ponumerowanych od 1 do 10. Znajdź przy użyciu algorytmu ewolucyjnego sposób na podział kart na dwie kupki w taki sposób, że suma kart na pierwszej kupce jest jak najbliższa wartości A, a suma kart na drugiej kupce jest jak najbliższa wartości B.

Spis treści:

1. Opis problemu
2. Decyzje projektowe
3. Pseudokod algorytmu głównego
4. Opis poszczególnych metod selekcji
5. Opis poszczególnych strategii
6. Opis metod krzyżowania
7. Opis struktury programu
8. Jak uruchomić program
9. Wnioski i podsumowanie

1. Opis problemu

Problem podziału n kart można rozumieć jako problem przydziału jednej z grup do każdej z kart. Algorytm ewolucyjny ma pomóc efektywnie rozwiązać podany problem. Specyfikacja problemu sprowadza się do dwóch punktów: dysponujemy n kartami ponumerowanymi od 1 do n. Celem jest podział kart na dwie grupy takie, że suma numerów kart z pierwszej grupy jest jak najbardziej zbliżona wartości A, a suma numerów kart z drugiej grupy jak najbliższa wartości B.

Wynika z tego, że przestrzeń możliwych rozwiązań jest rozmiaru 2^n. Wynika z tego, że zachłanne przeszukiwanie przestrzeni możliwych rozwiązań będzie działać tylko dla niewielkich n. Przy 10 kartach wymiar tej przestrzeni to 2^{10}=1024, a zatem niewiele jak na możliwości współczesnych komputerów. Dzięki temu można go wykorzystać jako punkt odniesienia przy testowaniu pozostałych algorytmów.

Problem wyboru elementów ze zbioru tak, aby ich suma była jak najbliższa danej wartości przypomina problem plecakowy. Problem plecakowy jest problemem NP-zupełnym. Dla problemu plecakowego istnieje algorytm programowania

dynamicznego, który pozwala uzyskać złożoność pseudowielomianową O(n*W), gdzie n jest liczbą elementów, a W jest oczekiwaną sumą. W przypadku algorytmu zachłannego jego złożoność jest identyczna $O(2^n)$ jak w naszym problemie. Do każdego elementu możemy przypisać dwa stany - element wybrany lub nie.

2. Decyzje projektowe

Pierwszą decyzją, którą podjęliśmy był wybór języka programowania. Wybór padł na język obiektowy Python, ponieważ uznaliśmy, że będzie to dobra okazja do pogłębienia naszej znajomości języka oraz ma on głębokie wsparcie dla zaawansowanych mechanizmów tj. programowanie obiektowe i inne, które znacznie ułatwiły nam implementację rozwiązania.

Kolejną decyzją, którą podjęliśmy, był fakt zaimplementowania różnych algorytmów ewolucyjnych, metod selekcji populacji rodzicielskiej oraz metod krzyżowania. Pozwoli nam to porównać efektywność zastosowanych rozwiązań i zdecydować, które z nich najlepiej rozwiąże nasz problem.

Zdecydowaliśmy także, że nie ograniczymy naszego problemu tylko do 10 kart, a liczba kart stanie się parametrem rozwiązania przyjmującym dowolną większą wartość. Problem dla 10 kart jest zbyt prosty, a większa liczba pozwoli wykorzystać prawdziwe możliwości zastosowanych algorytmów.

Wybór kodowania chromosomów był prosty, zastosowaliśmy kodowanie binarne. Zaletą kodowania binarnego jest prostota w stosowaniu i możliwości użycia prostych operatorów genetycznych. Jeden osobnik odzwierciedla sposób podziału kart na dwie grupy. Posiada on jeden chromosom, który składa się z zestawu n bitów, gdzie n oznacza liczbę kart do podziału. Każdy z kolejnych bitów będzie odpowiadał za przydział karty o takim samym numerze do jednej z grup. Zgodnie z przyjętą metodą kodowania, gen może przyjmować wartość 0 lub 1. Wartość 0 będzie oznaczać, że dana karta należy do grupy A, w p.p. dana karta należy do grupy B. Poniżej przedstawiony jest przykład osobnika opisującym podział 10 kart.

Gen chromosomu	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**
Numer karty	1	2	3	4	5	6	7	8	9	10
Grupa do której należy karta	**A**	**B**	**A**	**B**	**B**	**A**	**A**	**A**	**A**	**B**

Przykładowo chcemy, by suma numerów kart w grupie A wynosiła 10, a w grupie B 45. Funkcja przystosowania, którą przyjęliśmy jako:

$$Fitness = |expected\_sum_A - sum_A| + |expected\_sum_B - sum_B|$$

gdzie:
$expected\_sum_A$ - oczekiwana suma numerów kart w grupie A
$expected\_sum_B$ - oczekiwana suma numerów kart w grupie B
sum_A - suma numerów kart w grupie A dla danego osobnika
sum_B - suma numerów kart w grupie B dla danego osobnika

dla tego osobnika wynosi:
sum_A=1+3+6+7+8+9=34
sum_B=2+4+5+10=21
Fitness = |10-34|+|45-21|=24+24=48

Inne czynniki, mogące mieć wpływ na jakość rozwiązań algorytmów, także zostały sparametryzowane. Użytkownik ma prawo zmieniać następujące parametry:

- liczba kart do dystrybucji = rozmiar populacji
- oczekiwana suma numerów kart w grupie A
- oczekiwana suma numerów kart w grupie B
- prawdopodobieństwo krzyżowania
- prawdopodobieństwo mutacji
- rozmiar populacji rodzicielskiej λ
- rozmiar nowej populacji μ
- maksymalna liczba iteracji algorytmu

3. Pseudokod algorytmu głównego

Krok 0: Wygeneruj losową populację początkową.
Krok 1: Oceń przystosowanie osobników populacji.
Krok 2: Jeśli jest spełniony warunek stopu to zakończ działania algorytmu.
Krok 3: W p.p. wygeneruj populację rodzicielską poprzez selekcję.
Krok 4: Przeprowadź operacje genetyczne za pomocą operatorów
Krok 5: Utwórz nową populację i skocz do punktu 1.

4. Opis poszczególnych metod selekcji

4.1. Selekcja metodą koła ruletki

Metoda polega na tym, że każdemu osobnikowi można przydzielić wycinek koła ruletki o wielkości proporcjonalnej do wartości funkcji przystosowania danego osobnika. W ogólnym przypadku im większa jest wartość funkcji przystosowania, tym większy jest wycinek (sektor) na kole ruletki. W naszym przypadku dążymy do minimalizacji funkcji przystosowania, więc wycinek koła będzie odwrotnie proporcjonalny do funkcji przystosowania danego osobnika. W związku z tym potrzebny jest wskaźnik, który będzie służył do określenia wpływu danego osobnika.

Najpierw dokonujemy odwrócenia wartości funkcji przystosowania poprzez zastosowanie fomuły: $max + 1 - fitness$. Pozwala to na odwrócenie skali. Aby suma wpływów całej populacji była równa 1 należy ją podzielić przez sumę nowootrzymanych wartości. Można zauważyć że ze wszystkich wartości otrzymamy $n * (max + 1) - sum$

$$Influence = \frac{(max+1)-fitness}{n*(max+1)-sum}$$

gdzie:
max - wartość funkcji przystosowania najgorszego osobnika w populacji
fitness - wartość funkcji przystosowania osobnika, dla którego chcemy policzyć influence
n - liczba osobników populacji
sum - suma wartości funkcji przystosowania wszystkich osobników populacji.

Całe koło ruletki odpowiada sumie wartości influence wszystkich osobników rozważanej populacji. Każdemu osobnikowi oznaczonemu przez ch_i dla i=0,1,...,K gdzie K jest liczebnością populacji, odpowiada wycinek koła $v(ch_i)$ stanowiący część całego koła, wyrażony w procentach, zgodnie ze wzorem:

$$v(ch_i) = p_s(ch_i) * 100\%$$

w którym

$$p_s(ch_i) = \frac{F(ch_i)}{\sum_{j=1}^{K} F(ch_i)}$$

przy czym $F(ch_i)$ oznacza wartość funkcji przystosowania osobnika ch_i, a $p_s(ch_i)$ jest prawdopodobieństwem selekcji chromosomu ch_i. Selekcja osobnika odbywa się w ten sposób, że losuje się liczbę rzeczywistą z zakresu <0;1>, która odpowiada konkretnemu punktowi na okręgu koła ruletki. Im mniejsza jest wartość funkcji przystosowania osobnika, tym większy jest parametr Influence, a co za tym idzie tym większy jest wycinek koła ruletki odpowiadający temu osobnikowi. Im większy wycinek koła tym większe prawdopodobieństwo wylosowania tego osobnika do populacji rodzicielskiej.

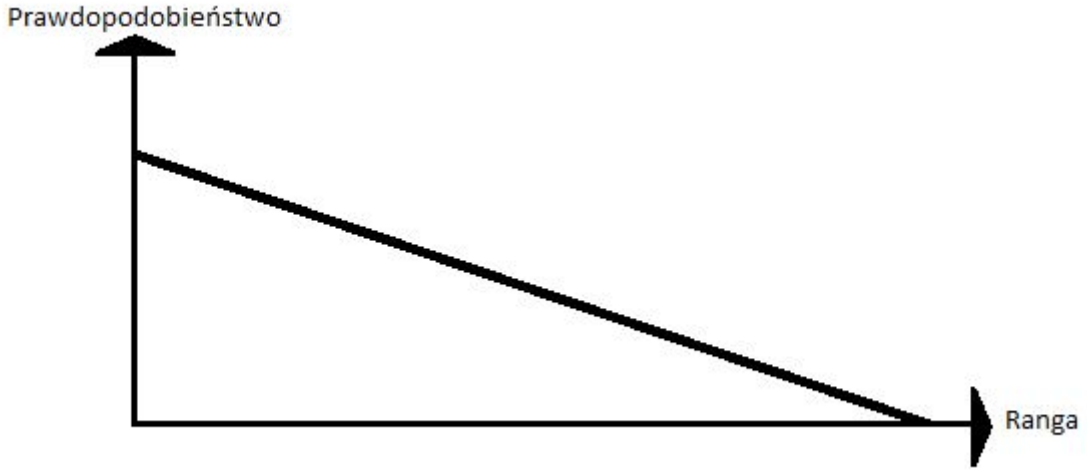
W wyniku selekcji zostaje utworzona populacja rodzicielska, nazywana też pulą rodzicielską. Następnie ta pula rodzicielska jest przekazywana dalej, gdzie wystąpią operacje genetyczne.

4.2. Selekcja metodą turniejową

Ta metoda polega na tym, że całą populację dzieli się na 2 lub 3 osobnikowe grupy tzw. grupy turniejowe. Z każdej grupy turniejowej najlepiej dopasowany (z najmniejszą wartością funkcji przystosowania) osobnik wybierany jest do populacji rodzicielskiej. Turniej powtarza się dotąd, aż populacja rodzicielska będzie miała pożądaną wielkość. W naszym przypadku przyjęliśmy, że grupę turniejową tworzą 3 losowo wybrane osobniki z całej populacji. Zaletą tej metody jest to, że nie potrzebujemy żadnych dodatkowych parametrów osobników, oprócz funkcji dopasowania, żeby stwierdzić, który z 3 osobników jest najlepiej dopasowany.

4.3. Selekcja metodą rankingową

W selekcji rankingowej, zwanej też selekcją rangową, osobniki populacji są ustawiane kolejno w zależności od wartości ich funkcji przystosowania. Można sobie to wyobrazić jako listę rankingową osobników uszeregowanych od najlepiej do najgorzej przystosowanego, gdzie każdemu osobnikowi przypisana jest liczba określająca jego kolejność na liście i nazywana rangą. Prawdopodobieństwo wybrania danego osobnika do populacji rodzicielskiej jest odwrotnie proporcjonalne do rangi osobnika, im niższa ranga (mniejsza wartość funkcji przystosowania) tym prawdopodobieństwo jest większe. Ta metoda jest bardzo podobna do metody koła ruletki. Modyfikacja polega jedynie na zmianie funkcji określającej prawdopodobieństwo wyboru danego osobnika. Funkcja ta wygląda następująco:



Metoda ta ma taką przewagę nad metodą koła ruletki, że nie napotyka konieczności skalowania w związku z problemem przedwczesnej zbieżności, co może wystąpić przy stosowaniu metody ruletki.

5. Opis poszczególnych strategii

5.1. Strategia (1+1)

Według tej strategii, populacja składa się tylko z jednego osobnika, którego nazywa się osobnikiem bazowym **x**. Algorytm rozpoczyna się od losowego ustalenia wartości poszczególnych składowych chromosomu tego osobnika. W każdej generacji w wyniku mutacji powstaje nowy osobnik **y**. Porównując wartości funkcji przystosowania obydwu osobników, tzn. F(**x**) i F(**y**), wybierany jest ten z mniejszą wartością funkcji przystosowania. To właśnie on staje się w następnej generacji nowym osobnikiem bazowym **x**. W algorytmie tym nie występuje operator krzyżowania. Osobnik **y** powstaje poprzez mutację na losowym genie osobnika bazowego.

5.2. Strategia (1+1) zrównoleglona

Jest to strategia oparta na strategii (1+1). Polega ona na tym, że z jednego osobnika bazowego **x** powstaje równolegle **μ** osobników, które powstają niezależnie w wyniku mutacji na losowym genie chromosomu osobnika bazowego. Jako nowy osobnik bazowy wybierany jest ten z **μ** nowo utworzonych osobników, który jest najlepiej przystosowany, czyli jego wartość funkcji przystosowania jest najmniejsza. Procedura ta jest powtarzana dopóki nie wystąpi warunek stopu.

5.3. Strategia (μ+λ)

Rozwinięciem strategii ewolucyjnej (1+1) jest strategia (μ+λ). Algorytm ten, dzięki większej liczbie osobników i w konsekwencji większej różnorodności genotypów, pozwala uniknąć końcowego rozwiązania w postaci minimum lokalnego, często spotykanego w poprzednio opisanej strategii (1+1). Algorytm rozpoczynamy od losowego wygenerowania początkowej populacji rodzicielskiej **P**, zawierającej **μ** osobników. Następnie tworzona jest, poprzez reprodukcję, populacja tymczasowa **T** zawierająca **λ** osobników, przy czym **μ>=λ.** Reprodukcja polega na jednej z metod selekcji poprzednio opisanych. Osobnicy populacji **T** podlegają operacjom krzyżowania i mutacji, w wyniku czego powstaje populacja potomno **O** o liczebności **λ.** Ostatnim krokiem jest wybór **μ** najlepszych potomków z obydwu populacji **P∪O**, które w nowej generacji będą stanowiły nową populację rodzicielską **P**.

5.4. Strategia (μ,λ)

Ta strategia jest bardzo podobna do strategii (μ+λ). Działanie obydwu algorytmów jest prawie identyczne. Różnica polega na tym, że nową populację **P** zawierającą **μ** osobników wybiera się tylko spośród **λ** najlepszych osobników populacji **O**. Aby było to możliwe, musi być spełniony warunek **μ<λ**. Metoda ta ma przewagę nad strategią (μ+λ) w

jednym dość ważnym punkcie, a mianowicie populacja do tej pory mogła być zdominowana jednym osobnikiem o małej wartości funkcji przystosowania, utrudniając znalezienie lepszych rozwiązań. Strategia **(μ,λ)** jest pozbawiona tej wady, ponieważ stare osobniki nie przechodzą do nowej puli rodzicielskiej.

5.5. Programowanie ewolucyjne

Strategia ta także jest podobna do (μ+λ), lecz istnieje dość znacząca różnica. Podczas każdej generacji algorytmu programowania ewolucyjnego nowa populacja **O** jest tworzona poprzez mutację każdego z osobników populacji rodzicielskiej **P**. Natomiast w strategii ewolucyjnej (μ+λ) każdy z osobników ma szansę na pojawienie się w populacji tymczasowej **T**, zgodnie z jedną z metod selekcji**,** na której są wykonywane operacje genetyczne, przy czym μ<=λ. W programowaniu ewolucyjnym populacje **P** oraz **O** są tak samo liczne, tzn. μ=λ. Ostatecznie nowa populacja rodzicielska **P** jest tworzona za pomocą selekcji rankingowej, której podlegają zarówno osobniki ze starej jak i nowej populacji.

6. Opis metod krzyżowania

6.1. Krzyżowanie jednopunktowe

Pierwszym etapem krzyżowania jest wybór par osobników z populacji rodzicielskiej (puli rodzicielskiej). Jest to tymczasowa populacja złożona z osobników wybranych metodą selekcji i przeznaczonych do dalszego przetwarzania za pomocą operacji krzyżowania i mutacji w celu utworzenia nowej populacji potomków. Na tym etapie osobniki z populacji rodzicielskiej kojarzone są w pary. Dokonuje się tego w sposób losowy, zgodnie z prawdopodobieństwem krzyżowania p_k. Następnie dla każdej z pary rodziców losuje się pozycję genu (locus) w chromosomie, określającą tzw. punkt krzyżowania. Jeżeli chromosom każdego z rodziców składa się z L genów, to oczywiście punkt krzyżowania l_k jest liczbą naturalną mniejszą od L. Zatem wybór punktu krzyżowania sprowadza się do wylosowania liczby z przedziału [0, L-1]. W wyniku krzyżowania pary chromosomów rodzicielskich otrzymuje się następującą parę:

A. potomek, którego chromosom składa się z genów na pozycjach od 1 do l_k. pochodzących od pierwszego rodzica, i następnych genów, od pozycji l_k+1 do L, pochodzących od drugiego rodzica;
B. potomek, którego chromosom składa się z genów na pozycjach od 1 do l_k pochodzących od drugiego rodzica, i następnych genów, od pozycji l_k+1 do L, pochodzących od drugiego rodzica.

6.2. Krzyżowanie dwupunktowe

Krzyżowanie dwupunktowe, jak sama nazwa wskazuje, tym różni się od krzyżowania jednopunktowego, że potomkowie dziedziczą fragmenty chromosomów rodzicielskich wyznaczone przez 2 wylosowane punkty krzyżowania.

6.3. Krzyżowanie równomierne

Krzyżowanie równomierne, nazywane też krzyżowaniem jednolitym lub jednostajnym, odbywa się zgodnie z wylosowanym wzorcem wskazującym, które geny dziedziczone są od pierwszego z rodziców (pozostałe pochodzą od drugiego). Krzyżowanie to można stosować do różnych rodzajów kodowania chromosomu. Musi być spełniony jeden warunek - chromosomy muszą być tej samej długości.

7. Opis struktury programu

Struktura programu zostało podzielona na 3 główne pliki: phenotype.py, generation.py, solution.py. Podstawowa klasa ***Phenotype*** reprezentująca pojedynczego osobnika populacji. Pola jakie posiada ta klasa to:

- ***genotype*** - lista, której liczba elementów odpowiada ilości kart do dystrybucji, każdy element tej listy należy do zbioru [0,1];
- ***fitness*** - liczba całkowita reprezentująca wartość funkcji przystosowania osobnika
- ***influence*** - liczba rzeczywista, odwzorowuje wycinek koła ruletki należący do osobnika, wykorzystana przy metodzie selekcji koła ruletki.

Umożliwia ona takie operacje na osobnikach jak: mutacja na genie o indeksie przekazanym jako argument do funkcji, krzyżowanie dwóch osobników metodą przekazaną jako argument funkcji, obliczenie funkcji przystosowania oraz liczby ***influence***.

Drugą, większą klasą zaimplementowaną w naszym rozwiązaniu jest klasa ***Generation***. Reprezentuje ona generację osobników, na której możliwe są przeróżne operacje. Stanowi ona bazę dla wszystkich strategii ewolucyjnych wykorzystywanych w projekcie. Posiada takie pola jak:

- ***population*** - lista obiektów klasy ***Phenotype***, odzwierciedla wszystkich osobników danej populacji.
- ***num_iterations*** - obrazuje liczbę iteracji danego algorytmu strategii
- ***number_of_individuals*** - liczba osobników populacji
- ***expected_sum_A*** - oczekiwana suma numerów kart w grupie A
- ***expected_sum_B*** - oczekiwana suma numerów kart w grupie B

- ***max_iterations*** - liczba iteracji danego algorytmu strategii, po przekroczeniu której algorytm się zatrzymuje

W ramach tej klasy dodatkowo zaimplementowane są funkcje usprawniające rozwiązania problemu, tj. obliczenie średniego dopasowania populacji, posortowanie populacji, wyłonienie najlepszego osobnika, obliczenie funkcji przystosowania dla każdego osobnika. Dostępne są również 3 funkcje: ***RouletteSelection***, ***TournamentSelection***, ***RankingSelection***. Realizują one wcześniej opisane metody selekcji populacji tymczasowej, z której później będą wyłonieni potomkowie.

Klasami, które dziedziczą po klasie ***Generation***, są klasy, które reprezentują poszczególne strategie ewolucyjne. Zdecydowaliśmy się na takie rozwiązanie, ponieważ wszystkie one będą miały bardzo podobną funkcjonalność, a różnica jest tylko w samym algorytmie postępowania z populacją osobników. Aby jednak zaimplementować te różnice, w każdej z nich stworzyliśmy metodę ***step***, której wykonanie jest jednoznaczne z jedną iteracją algorytmu strategii.

8. Jak uruchomić program

Żeby uruchomić program, należy przede wszystkim pobrać kod źródłowy, który jest dostępny pod tym adresem:

https://github.com/Rajtek/PSZT-cards-divider.git

Ze względu na wykorzystane narzędzia, niezbędne jest zainstalowanie interpretera języka Python w wersji 2.7 wraz z bibliotekami: ***numpy***, ***pylab***, ***matplotlib***. Kiedy już będziemy w posiadaniu wyżej wymienionych, możemy uruchomić program na dwa sposoby:

1. Poprzez polecenie konsolowe:

   ./uruchom nazwa_strategii [nazwa_metody_selekcji] [metoda_krzyzowania]

   gdzie

nazwa_strategi oznacza jedną z następujących: *1Plus1*, *1Plus1Paralleled*, *MiLambda*, *MiPlusLambda*, *EvolutionaryProgramming*.
nazwa_metody_selekcji oznacza: *RouletteSelection, TournamentSelection, RankingSelection.*
metoda_krzyzowania oznacza: *single-point, two-point, uniform*

Wynikiem tej metody uruchomienia programu jest plik jpg z wykresem, który ilustruje jak zmieniała się funkcja przystosowania w kolejnych iteracjach algorytmu.

2. Poprzez polecenie konsolowe:

./solution.py nazwa_strategii [nazwa_metody_selekcji] [metoda_krzyzowania]

gdzie znaczenie poszczególnych argumentów jest takie same jak wyżej opisane. Wynikiem tej metody na wyjściu stdout są wiersze z taką samą informacją jak wyżej, tylko bez wykresu. Wynik tej metody jest trudniejszy w analizie, gdyż to co wyświetla program to średnie dopasowanie populacji oraz funkcja dopasowania najlepszego osobnika po każdej iteracji algorytmu.

Przed uruchomieniem programu możemy zmienić parametry algorytmów, które znajdują się w górnej części pliku solution.py.

9. Wyniki działania i porównanie metod

Aby wyłonić najlepszy algorytm i przetestować, jak sobie radzą poszczególne algorytmy w zbliżonych warunkach, przeprowadziliśmy 2 testy wszystkich algorytmów. Parametry obu testów były jednakowe:

- liczba kart: 1000
- oczekiwana suma numerów kart w grupie A: 124951
- oczekiwana suma numerów kart w grupie B: 375549
- prawdopodobieństwo krzyżowania: 0.8
- prawdopodobieństwo mutacji dla strategi **(μ,λ) i (μ+λ)**: 0.04
- maksymalna liczba iteracji algorytmu: 1000
- wielkość populacji początkowej (**μ**): 40
- wielkość populacji tymczasowej (**λ**): 50

Porównamy, ile iteracji zajęło algorytmom na znalezienie stosunkowo dobrego rozwiązania i w jakim czasie to się odbyło. Aby miało ono sens, porównamy wyniki ze względu na rodzaj krzyżowania, metodę selekcji i strategię ewolucyjną. Analiza metody krzyżowania i sposobu selekcji ma sens tylko dla algorytmów (μ+λ) i (μ,λ), więc porównajmy na początku wyniki ze względu na rodzaj krzyżowania i metodę selekcji.

1 Runda:

krzyżowanie	(μ+λ) Ranking		(μ+λ) Turniej		(μ+λ) Ruletka		(μ,λ) Ranking		(μ,λ) Turniej		(μ,λ) Ruletka	
	Czas	Iteracje	Czas	Iteracje	Czas	Iteracje	Czas	Iteracje	Czas	Iteracje	Czas	Iteracje
1-punktowe	16	430	16	490	18	400	40	>1000	16	500	31	530
2-punktowe	8	370	13	400	23	370	40	>1000	13	380	19	520
równomierne	2	50	6.5	140	7	70	56	>1000	3	50	8	140

2 Runda:

krzyżowanie	(μ+λ) Ranking		(μ+λ) Turniej		(μ+λ) Ruletka		(μ,λ) Ranking		(μ,λ) Turniej		(μ,λ) Ruletka	
	Czas	Iteracje	Czas	Iteracje	Czas	Iteracje	Czas	Iteracje	Czas	Iteracje	Czas	Iteracje
1-punktowe	16	470	13	430	15	440	34	>1000	16	520	17	580
2-punktowe	15	330	12	370	10	300	32	>1000	12	450	18	550
równomierne	3	60	5	100	5	105	45	>1000	2	37	6	140

Jeśli chodzi o metody krzyżowania, to jednoznacznie wygrywa krzyżowanie równomierne, nie powinniśmy mieć co do tego żadnych wątpliwości. Wykorzystując tą metodę zdecydowanie najszybciej program znalazł optymalne rozwiązanie. Krzyżowanie dwupunktowe okazało się delikatnie lepszym rozwiązaniem niż jednopunktowe.

Jeśli chcemy rozstrzygnąć, która metoda selekcji najlepiej rozwiązuje nasze zadanie, to tutaj odpowiedź nie jest już taka jednoznaczna. Mimo tego, można zauważyć, że, poza jednym wyjątkiem jakim jest zestawienie strategii (μ,λ) z rankingową metodą selekcji, najgorzej radzi sobie algorytm selekcji metodą koła ruletki. Zarówno w 1 jak i 2 rundzie ta metoda potrzebuje najwięcej czasu, chociaż różnica jest niewielka. Jeśli chodzi o wyłonienie lepszej metody z turniejowej i rankingowej, to zadanie w tym przypadku nie jest proste. Biorąc pod uwagę fakt, że strategia (μ,λ) z rankingową metodą selekcji nie zdołała znaleźć optymalnego rozwiązania podczas 1000 iteracji, zwycięzcą zostaje metoda selekcji turniejowej. Najlepszym zestawieniem metod selekcji wraz z metodą krzyżowania jest metoda turniejowa + krzyżowanie równomierne, gdyż taka kombinacja zdumiewająco dobrze poradziła sobie z rozwiązaniem tego problemu.

Aby porównać strategie ewolucyjne, do zestawienia ze strategiami 1+1, 1+1 zrównolegloną oraz programowaniem ewolucyjnym, wybraliśmy najlepsze warianty strategii (μ+λ) oraz (μ,λ). Dla (μ+λ) była to metoda selekcji rankingowej z krzyżowaniem równomiernym, a dla (μ,λ) metoda selekcji turniejowej również z krzyżowaniem równomiernym. Wyniki prezentują się następująco:

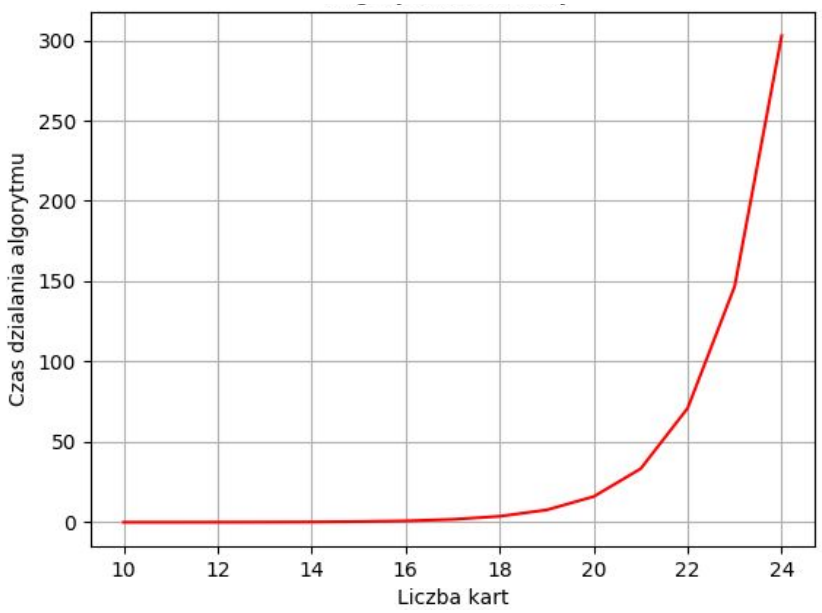
	Strategia:	1+1	1+1 równ	Prog. ew.	(μ+λ)	(μ,λ)
1 Runda	Czas:	0.47	3.2	15	2	3
	Iteracje:	680	160	310	50	50
2 Runda	Czas:	0.6	3	13	3	2
	Iteracje:	640	160	290	60	37

Jeżeli byśmy brali pod uwagę czas, potrzebny na rozwiązanie problemu, to najlepszy okazał się być algorytm (1+1). Z drugiej strony potrzebuje on zdecydowanie więcej iteracji niż np. (μ+λ) czy (μ,λ). Te dwie strategie poradziły sobie równie dobrze, chociaż z delikatną przewagą dla (μ,λ). Zdecydowanie najgorzej radzi sobie algorytm programowania ewolucyjnego.

10. Wnioski i spostrzeżenia

Stworzone algorytmy działają niespodziewanie dobrze. Przegląd 1000 kart jest zagadnieniem bardzo mocno złożonym. Dla porównania został napisany algorytm wykorzystujący przegląd zupełny. Zostały przeprowadzone dwa testy - jeden gdy suma kart była równa A+B i jeden gdy znalezienie takiego rozwiązania nie było możliwe.

Drugi przypadek zmuszał algorytm do przeglądu całej dziedziny. Czasy wykonania dla poszczególnych n zostały przedstawione na wykresie.



Dobrze widać na nim złożoność wykładniczą. Dla 24 kart przegląd trwał już ponad 5 minut. Natomiast dla 1000 kart zajęłoby około $6.3 * 10^{293}$ raza dłużej. To znacznie więcej niż dotychczasowa długość życia wszechświata.

Jasno to pokazuje przewagę algorytmów ewolucyjnych. Nawet najgorsze z nich działają w dopuszczalnym czasie. Inną sprawą jest to, że algorytmy ewolucyjne nie zawsze znajdowały idealne rozwiązanie. Jednak możliwe było kilkukrotne ich uruchomienie po których zwykle optymalna wartość była zwracana. Problemem może być sytuacja kiedy nie znamy wartości globalnego minimum. Nie mamy wtedy pewności czy znaleziona podczas wszystkich uruchomień wartość jest optymalna.