

**Wydział Elektrotechniki Automatyki Informatyki i Inżynierii Biomedycznej**

***Algorytm Ewolucyjny w zastosowaniu w optymalizacji na szeregach czasowych.***

Autorzy: Dawid Bugajny

*Wojciech Woszczek*

*Igor Ratajczyk*

Kierunek studiów: Automatyka i Robotyka

Przedmiot: *Badania Operacyjne 2*

Kraków, 2021/2022

Spis treści

**1**. **Zarys** **problematyki**3

**2. Model matematyczny problemu4**

2.1 Przyjęte oznaczenia4

2.2 Funkcja celu5

2.3 Ograniczenia oraz postać rozwiązania 6

**3. Algorytm ewolucyjny7**

**4. Opis elementów opracowanych8**

4.1 Kodowanie rozwiązania8

4.2 Konstrukcja rozwiązań początkowych8

4.3 Operatory krzyżowania9

4.4 Operatory mutacji10

4.5 Selekcja11

4.6 Operator rzutowania na rozwiązania dopuszczalne12

4.7 Hybrydowy algorytm ewolucyjny13

**5. Parametry14**

5.1 Parametry funkcji przystosowania14

5.2 Parametry algorytmu ewolucyjnego14

5.3 Parametry rozwiązania15

5.4 Parametry mutacji15

5.5 Parametry krzyżowania15

5.6 Parametry selekcji16

**6. Aplikacja algorytmu17**

**7. Testy algorytmu18**

**7. Podsumowanie36**

7.1 Wnioski36

7.2 Stwierdzone problemy36

7.3 Kierunki dalszego rozwoju36

1. **Zarys Problematyki**

Na podstawie historycznych danych zachorowalności na COVID-19 można stwierdzić znaczną sezonowość tego procesu. Aby skutecznie walczyć z zarazą należy skutecznie pobierać wymazy osób podejrzanych o chorobę, aby móc skutecznie przeciwdziałać rozwojowi epidemii. Jako że żadna służba zdrowia na świecie nie jest przygotowana na tego typu wydarzenia, ale sytuacja wymaga znacznie większej liczby laborantów pobierających wymazy. Praca i szkolenie laborantów kosztują.

Problem optymalizacyjny polega na takim sterowaniu liczbą przyjmowanych laborantów, aby zminimalizować koszty szkolenia i pracy personelu medycznego oraz koszty śmierci nieprzebadanego pacjenta.

Dzięki ubiegło sezonowym danym i modelowi SARIMA będziemy prognozować przyszłą liczbę dziennych zachorowań .

Zakładamy, że każdy laborant pobierający wymazy jest w stanie pobrać wymazów dziennie. Szkolenie nowego laboranta trwa jeden dzień i kosztuje a dzień jego pracy jest warty . Pacjent chory, który nie zostanie przebadany kosztuje państwo .

Z racji ograniczonej pojemności sal i mając na celu wydajność szkolenia w danym dniu nie można szkolić więcej niż laborantów.

1. **Model Matematyczny Problemu.**

2.1. Przyjęte oznaczenia.

– operator różnicowy

– stała do schematu kosztu oczekujących pacjentów

predykcja dziennej liczby zachorowań (liczba naturalna)

– backshift operator – operator opóźnienia

górne ograniczenie dziennej liczby szkolonych laborantów

– funkcja Heavyside’a

– operator sumy kumulowanej

– warunek początkowy liczby laborantów

– liczba jednostek czasu w której przeprowadzamy optymalizację

– liczba wymazów, jaką w jednostce czasu pobiera laborant

– liczba jednostek czasu, po których niewymazany pacjent nie zostanie wymazany

– szereg czasowy pacjentów, którzy nie otrzymali wymazu

Koszt szkolenia jednego laboranta

2.2. Funkcja celu.

Implementacja algorytmu pozwala na optymalizację problemów, gdzie koszt może być liczony na dwa sposoby.

Funkcja celu (Time Series Covid Problem Naive):

Można to interpretować jako sumę kosztów po wszystkich dniach, gdzie uwzględniamy koszt pracy wszystkich laborantów, oczekiwany koszt życia niewymazanych pacjentów oraz koszt szkolenia nowych laborantów.

Ten model funkcji kosztu jest znacznie prostszy w zrozumieniu, stanowi sensowny sposób badania zachowania algorytmu, gdzie wymagana jest znajomość rozwiązania oraz może być traktowana jako punkt wyjściowy do zrozumienia bardziej zaawansowanej funkcji kosztu z punktu drugiego.

Funkcja celu (Time Series Covid Problem):

gdzie:

takie, że:

Ogólny zamysł drugiej funkcji kosztu implementuje scenariusz, gdzie pacjent niewymazany może zostać przebadany jeszcze przez dni od pierwszego dnia zjawienia się w przychodni. Jeżeli liczba pacjentów przekracza możliwości badawcze zakładu ustawia się on w kolejce na oczekujących a kolejka oczekujących „opóźnia się” o jeden dzień. Jeżeli liczba pacjentów nie przekracza możliwości przetwórczych zakładu to wymazywani są pacjenci oczekujący, ale najpierw wymazywani są pacjenci oczekujący najdłużej ( odpowiada za zmniejszanie liczby oczekujących pacjentów).

Koszt naliczany jest względem oczekującej kolejki pacjentów: każdy kolejny dzień oczekiwania kosztuje wykładniczo (domyślnie: mniej) niż poprzedni, co odzwierciedla rzeczywistość, gdzie pacjent zaraz po zakażeniu czuje się na tyle dobrze, że jest w stanie dalej rozprzestrzeniać wirusa.

2.3. Ograniczenia oraz postać rozwiązania

Rozwiązanie jest postaci:

Gdzie każdemu dniu roku jest przypisywana liczba.

Ograniczenia są postaci:

Oznacza to, że:

* Liczba laborantów na początku roku jest zadania
* Przyrost laborantów jest ograniczony
* Liczba laborantów jest nieujemna

1. **Algorytm Ewolucyjny.**

Pseudokod algorytmu:

t: = 0

Population = new Population ()

Fitness = FitnessFunction ()

While (not Stop):

t++

for random f, g in population:

allowed:= draw

if allowed:

x, y = Cross(f,g)

Population += x

Population += y

for random f in population:

allowed:= draw

if allowed:

f = Mutate(f)

f = CastFeasible(f)

Fitness = FitnessFunction ()

Population = Select (Population+Elite)

Elite = SelectElite (Population+Elite)

If allowHybrid:

BestSolution = HybridOptimizer (BestSolution)

return BestSolution

1. **Opis elementów opracowanych.**

Należy zwrócić szczególną uwagę, że z racji wielowymiarowości problemu, klasyczne podejście nakazuje położenie szczególnego nacisku na zachowywanie dopuszczalności rozwiązania. Tak też zostało poczynione, dlatego zaprojektowane operatory genetyczne, pseudogenetyczne oraz inne elementy rozwiązania skupiają się na zachowywaniu dopuszczalności rozwiązania. Dokładniejsze omówienie zastosowanych elementów znajduje się odpowiadających im podrozdziałach.

4.1. Kodowanie rozwiązania.

Genotyp jest kodowany jako wektor liczb całkowitych, liczby zatrudnionych laborantów każdego dnia:

Fenotyp jest kodowany jako wektor liczb całkowitych, liczby laborantów na każdy dzień:

4.2. Konstrukcja rozwiązań początkowych.

Domyślna metoda konstrukcji rozwiązań początkowych obejmuje konstruowanie rozwiązania dopuszczalnego. Można ją zapisać jako:

Co oznacza, że rozwiązania są konstruowane w przód, próbkowane z rozkładu jednostajnego w dopuszczalnym zakresie (tj. tak aby przyrost laborantów był ograniczony liczba laborantów była nieujemna).

Jest możliwość konstrukcji rozwiązania dopuszczalnego z wykorzystaniem innych rozkładów prawdopodobieństwa.

Konstrukcja rozwiązania niedopuszczalnego obejmuje próbkowanie z rozkładu nieograniczonego.

4.3. Operatory Krzyżowania.

Zasadniczym operatorem krzyżowania jest operator kombinacji wypukłej z parametrem dany wzorem:

Wypada rozważyć, czy zachodzi:

(4.2.3)

tj. czy z dopuszczalności rozwiązań rodzicielskich wynika dopuszczalność rozwiązań potomnych.

Z uwagi na liniowe ograniczenia genotypu i liniowe ograniczenia liniowych przekształceń genotypu (suma kumulowana i operator różnicowy) ograniczenia 2 i 3 są również zachowane.

4.4. Operatory Mutacji.

Zaimplementowane operatory mutacji obejmują:

Mutacja typu Gaussowskiego

Do każdego elementu wektora rozwiązania dodawana jest zmienna o rozkładzie Gaussowskim:

Szumy są między sobą nieskorelowane.

Mutacja typu Cauchy’ego - Lorenza

Do każdego elementu wektora rozwiązania dodawana jest zmienna o rozkładzie Gaussowskim:

Gdzie rozkład prawdopodobieństwa jest dany wzorem:

Szumy są między sobą nieskorelowane.

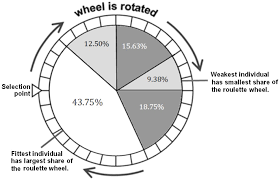
Szum Cauchy’ego – Lorenz ‘a jest wprowadzony dla uzyskania bardziej zróżnicowanych genotypów (dla mutacji Gaussowskiej na komputerach trudno uzyskać mutację odległą bardziej niż .

Z uwagi na to, że żadna z zaproponowanych mutacji nie zapewnia zachowania dopuszczalności rozwiązania, po mutacjach rozwiązania są rzutowane na rozwiązania dopuszczalne.

4.5. Selekcja.

Domyślną selekcją w to selekcja ruletkowa. W każdej iteracji algorytmu losowana jest określona liczba osobników ze zbioru otrzymanego po ew. krzyżówce oraz mutacji. Prawdopodobieństwo wybrania osobnika jest proporcjonalne do jego przystosowania (wyliczonego według zadanej funkcji celu) i wyraża się wzorem:

Zakładając rozkład jednostajny, sytuacja ta jest analogiczna do kręcenia tarczą, gdzie pole każdej opcji jest proporcjonalne do jego wartości przystosowania:



Zaimplementowana została również selekcję turniejową. Polega ona na przeprowadzeniu paru „turniejów” pośród losowo wybranych osobników z populacji. Zwycięzca każdego turnieju jest wybierany do następnej populacji (w kolejnej iteracji). Metoda turniejowa może być opisana przez poniższy pseudokod:

Dopóki nie osiągniemy pożądanej liczności populacji:

Wybierz losowo k osobników z poprzedniej populacji,

Najlepszy (funkcja kosztu) osobnik jest wybrany z prawd. p

N – ty najlepszy osobnik jest wybierany z prawdopodobieństwem p(1-p)^n

4.6. Operator rzutowania na rozwiązania dopuszczalne.

Jako że operatory mutacji nie zawsze zachowują dopuszczalność rozwiązania opracowany został operator rzutowania na rozwiązania dopuszczalne. Jest on postaci:

Podobnie jak operator konstrukcji rozwiązania, buduje on rozwiązania w przód.

Łatwo sprawdzić, że dla rozwiązania dopuszczalnego nie zmienia on postaci rozwiązania:

ale:

co nie zachodzi dla rozwiązań dopuszczalnych.

4.7. Hybrydowy algorytm ewolucyjny.

Z uwagi na wielowymiarowość problemu i dynamikę algorymu ewolucyjnego zostały zaimplementowane dodatkowe metody optymalizacji najlepszego rozwiązania.

Obydwie metody optymalizacji można zakfalifikować do metod zachłannych. Optymalizuje ona przechodząc po rozwiązaniu w przód przeszukując wszystkie dopuszczalne wartości i wstawia w jej miejsce wartość minimalizującą funkcję celu.

For t = 1: end

Solution = min (Solution(t) = d for d in feasible)

Druga metoda zachłannej optymalizacji iteruje wiele razy po tym samym roziwiązaniu dopóki następują jakieś zmiany.

While Solution has changed

For t = 1: end

Solution = min (Solution(t) = d for d in feasible)

1. **Parametry**

5.1. Parametry funkcji celu.

* ***name\_of\_fitness\_function****: str –* parameter określający pożądaną funkcję celu. Możliwe wartości: “TimeSeriesCovidProblemNaive”, “TimeSeriesCovidProblem” (obie funkcje opisane w rozdziale 2.1);
* worker\_cost – ***worker\_cost*** *: float–* koszt każdego pracownika w jednostce czasu;
* death\_probability***death\_probablity****: float –* prawodpodobieństwo śmierci pacjenta, który nie został zbadany wymazem;
* ***cost\_of\_death***: float – oczekiwany koszt śmierci pacjenta;
* ***training\_cost****: float –* koszt trenowania pracownika;
* ***swabs\_per\_day****: int –* ilość wymazów wykonywana przez pracownika w jednostce czasu;
* ***timer\_series****: np.ndarray –* szereg czasowy określający przewidywana ilość zakażeń na następny rok – otrzymany przy pomocy danych historycznych oraz modelu ARIMA;

Dodatkowe parametry używane wyłącznie w przypadku używania funkcji celu “TimeSeriesCovidProblem”:

* ***cost\_of\_non\_immediate\_swab****: float –* oczekiwany koszt nieprzeprowadzenia natychmiastowego wymazu;
* ***days\_for\_swab****: int –* jednostki czasu (dni, tygodnie) w których można przeprowadzić wymaz;
* ***delayed\_cost****: bool –* oczekiwany koszt opóźnienia wykonania wymazu;

5.2. Parametry algorytmu ewolucyjnego.

* ***allow\_eps\_ff\_stop****: bool* – czy ograniczenie związane z funkcją przystosowania to kryterium stopu;
* ***esp\_ff****: float* – o ile zmienić się może funkcja celu abu uznać że warto zatrzymać algorytm;
* ***eps\_ff\_type****: str* – Przyjmuje wartości Best albo Average
* ***no\_back:*** *int –* liczba iteracji w tył, w których musi następić zmiana o ***esp\_ff*** w przeciwnym wypadku może nastąpić stop z kryterium ***allow\_eps\_ff\_stop***
* ***allow\_no\_iter\_stop****: bool –* czy ograniczenie liczby iteracji to kryterium stopu;
* ***no\_iter****: int –* liczba iteracji (kryterium stopu);
* ***allow\_indifferent\_population\_stop****: bool –* czy ograniczenie związane z różnorodnością populacji to kryterium stopu;
* ***population\_diversity\_measure****: str –* sposób liczenia różnorodności populacji (Np odchylenie standardowe wartości funkcji przystosowania w populacji)
* ***pop\_div\_eps:*** *float* ***–*** minimalna wartość miary różnorodności funkcji przystosowania
* ***population\_number****: int – ­*liczność zbioru populacji w każdej iteracji (liczba rozwiązań);
* ***hybrid****: bool –* czy stosować optymalizator hybrydowy dla ostatecznego rozwiązania.

5.3. Parametry rozwiązania.

* ***problem\_name****: str –* rodzaj rozwiązywanego problemu. Możliwe wartości: „TSOCOV19D”;
* ***create\_feasible****: bool –* Wartość logiczna określająca czy początkowe rozwiązanie ma być konstruowane jako dopuszczalne;
* ***solution\_size****: int* – rozmiar rozwiązania (długość wektora);
* ***L\_limit****: int –* dolny limit zatrudnionych pracowników, wartość >=0;
* ***DL\_limit****: int –* górny limit dodatniego przyrostu liczby pracowników;
* ***L0****: int –* liczba zatrudnionych pracowników w ostatni dzień poprzedniego roku.

5.4. Parametry mutacji.

* *type\_of\_mutation: str –* typ mutacji uzywanej w algorytmie. Możliwe wartości: „Cauchy”, „Gaussian”;
* *mutation\_probability: float –* prawdopodobieństwo przeprowadzenia przez algorytm mutacji w każdej iteracji;

Parametry używane tylko w przypadku używania mutacji Cauchy’ego:

* *mu –* parametrśredniej rozkładu Cauchy’ego;
* *gamma –* parametrgamma (określający rozrzut) rozkładu Cauchy’ego;

Parametry używane tylko w przypadku używania mutacji gaussowskiej:

* *mean –* parametrśredniej rozkładu Gaussa;
* *std –* odchylenie standardowe rozkładu Gaussa;

5.5. Parametry krzyżowania

* ***type\_of\_crossover****: str –* typ pożadanej krzyżówki. Możliwe wartości: "One Point", "Two Points", "Average", "Convex Combination", "Uniform";
* ***crossover\_probability****: float –* prawdopodobieństwo wykonania krzyżówki przez algorytm w każdej iteracji;

Parametr używany wyłącznie w przypadku używania krzyżówek *One Point* oraz *Two Points*:

* ***distribution\_of\_cut****: str –* typ rozkładu, według którego przeprowadzane jest cięcie. Możliwe wartości: „Uniform”;

Parametr używany wyłącznie w przypadku używania krzyżówki *Convex Combination*:

* ***alpha\_distribution****: str* – typ rozkładu, według którego w każdej iteracji wybierany jest parametr alfa. Możliwe wartości: „Uniform”, „Arcsine”.

5.6. Parametry selekcji

* ***type\_of\_selection****: str–* typ przeprowadzanej w każdej iteracji algorytmu selekcji. Możliwe wartości: „Roulette”, „Tournament”;
* ***elite****: bool –* wartość logiczna określająca, czy chcemy użyć selekcji elitarnej;
* ***truncation****: bool –* wartość logiczna określająca, czy chcemy użyć mechanizmu ucinania;
* ***elite\_count****: int –* liczność zbioru elitarnego;
* ***proportion****: float –* proporcja najlepszych osobników używany w mechanizmie cięcia;

Parametry używane tylko w przypadku używania selekcji turniejowej:

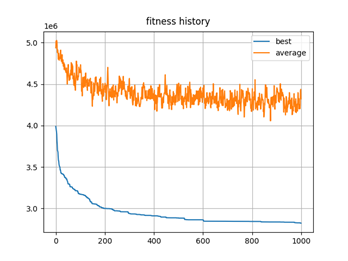
* ***k****: int –* wielkość turnieju (liczba uczestników);
* ***p****: float –* prawdopodobieństwo wygrania najlepszego osobnika w turnieju.

1. **Aplikacja Algorytmu**
2. **Testy algorytmu**

Testy przeprowadziliśmy dla różnych rodzajów krzyżówek, mutacji oraz selekcji. W tytułach podrozdziałów wypisane jest jakie z nich są użyte. Dla pierwszych ośmiu testów użyta została funkcja kosztu „naiwna”, 1000 iteracji, "worker\_cost": 5, "death\_probability": 0.15, "cost\_of\_death": 100, "training\_cost": 4, "swabs\_per\_day": 1, "solution\_size": 52, "L\_limit": 3000, "DL\_limit": 8000, "L0": 5000.

**7.1 Mutation: Gaussian, Convex crossover α distribution: Uniform, Selection: Roulette**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L. p.** | **Wartość f kosztu** | **Dopuszcza-lność** | **Rozwiązanie** | **Uwagi** |
| 1 | 2828036 | TAK |  |  |
| 2 | 2900566 | TAK |  | worst |
| 3 | 2801381 | TAK |  |  |
| 4 | 2646780 | TAK |  | best |
| 5 | 2720261 | TAK |  |  |

Przebieg optymalizacji:

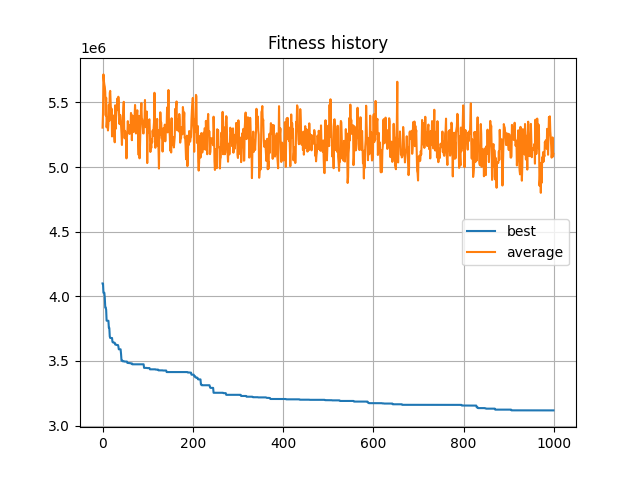
Uwagi:

Małe różnice w różnorodności rozwiązań

**7.2 M: Cauchy, convex X α dist: Uniform, S: Roulette**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L. p.** | **Wartość f kosztu** | **Dopuszcza-lność** | **Rozwiązanie** | **Uwagi** |
| 1 | 3054010 | TAK |  | worst |
| 2 | 3025378 | TAK |  |  |
| 3 | 3000134 | TAK |  |  |
| 4 | 2987398 | TAK |  |  |
| 5 | 2938525 | TAK |  | best |

Przebieg optymalizacji:



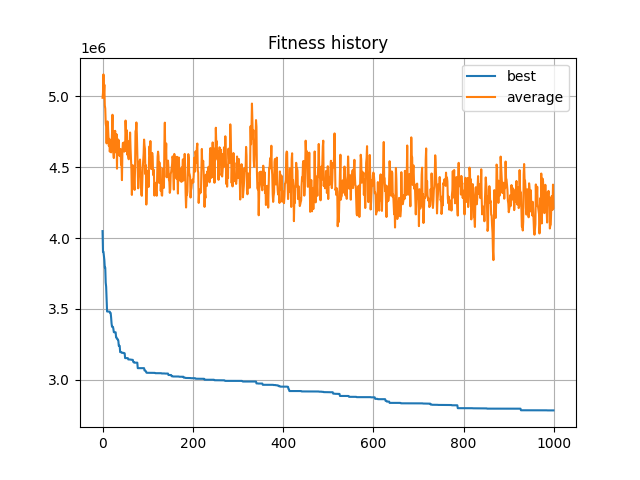
Uwagi:

Mutacja cauchy’ego powoduje, że różnorodność zwracanych rozwiązań jest większa oraz zmniejsza gładkość krzywej optymalizacji. Co ciekawe, wartości funkcji przystosowania są wyższe niż w przypadku mutacji gaussowskiej – zatem ten rodzaj mutacji spisuje się nieco gorzej.

**7.3 M: Gaussian, convex X α dist: Arcsine, S: Roulette**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L. p.** | **Wartość f kosztu** | **Dopuszcza-lność** | **Rozwiązanie** | **Uwagi** |
| 1 | 2655109 | TAK |  | best |
| 2 | 2659938 | TAK |  |  |
| 3 | 2817041 | TAK |  |  |
| 4 | 3006226 | TAK |  | worst |
| 5 | 2663099 | TAK |  |  |

Przebieg optymalizacji:



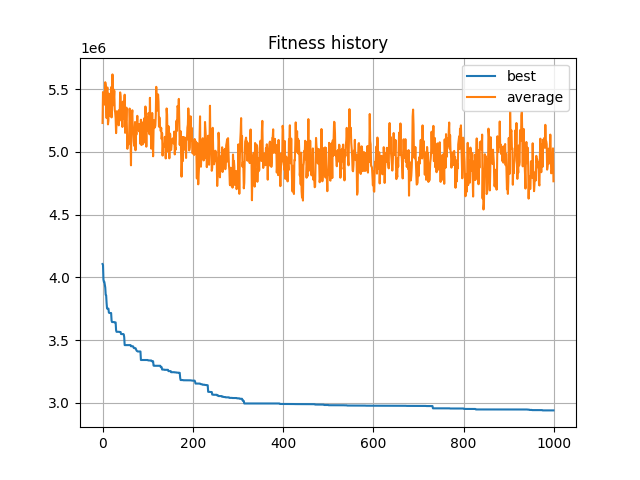
Uwagi:

Na przebiegu optymalizacji widać, że zmiana rozkładu parametru α powoduje szybszy spadek funkcji kosztu dla małych iteracji w porównaniu do rozkładu jednostajnego. Jednakże, samo rozwiązanie najlepsze jest gorsze niż dla uniform.

**7.4 M: Cauchy, convex X α dist: Arcsine, S: Roulette**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L. p.** | **Wartość f kosztu** | **Dopuszcza-lność** | **Rozwiązanie** | **Uwagi** |
| 1 | 2891677 | TAK |  | worst |
| 2 | 2869047 | TAK |  |  |
| 3 | 2753894 | TAK |  | best |
| 4 | 2819538 | TAK |  |  |
| 5 | 2877978 | TAK |  |  |

Przebieg optymalizacji:



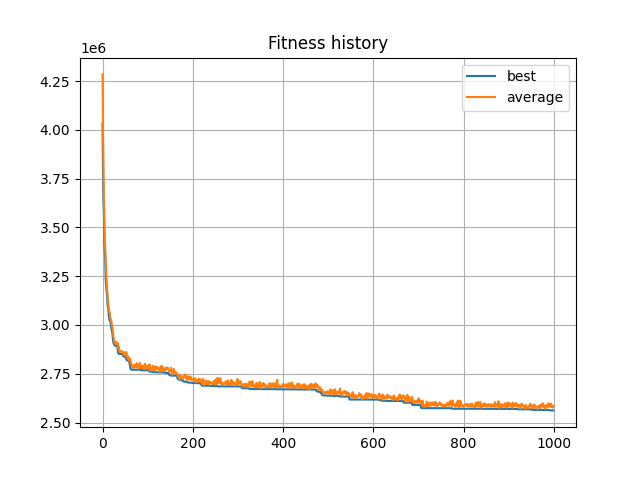
Uwagi:

Mutacja Cauchy’ego znów poradziła sobie gorzej od gaussowskiej – mniejsza wartość funkcji kosztu

**7.5 M: Gaussian, convex X α dist: Uniform, S: Tournament**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L. p.** | **Wartość f kosztu** | **Dopuszcza-lność** | **Rozwiązanie** | **Uwagi** |
| 1 | 2510166 | TAK |  | best |
| 2 | 2521959 | TAK |  |  |
| 3 | 2549202 | TAK |  |  |
| 4 | 2533015 | TAK |  |  |
| 5 | 2604863 | TAK |  | worst |

Przebieg optymalizacji:



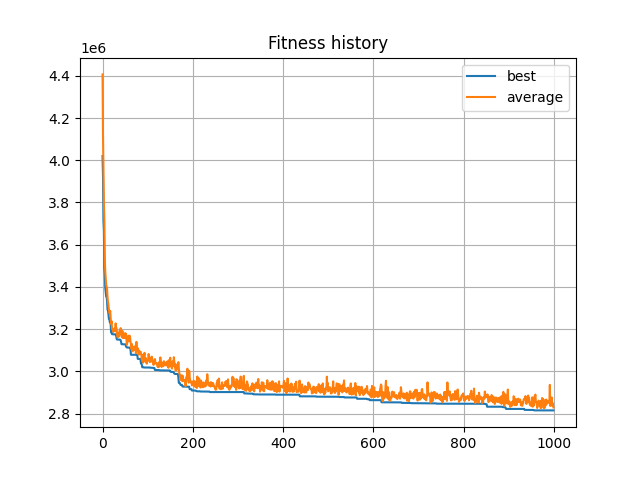
Uwagi:

Zmiana selekcji na turniejową drastycznie zmienia przebieg średniej funkcji celu w danej populacji – rozwiązania w populacji są bardzo zbliżone do siebie i ich srednia wartość funkcji przystosowania jest bardzo zbliżona do najlepszego osobnika. Samo rozwiązanie nieco lepsze od selekcji ruletkowej – patrząc na najlepszy wynik f kosztu.

**7.6 M: Cauchy, convex X α dist: Uniform, S: Tournament**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L. p.** | **Wartość f kosztu** | **Dopuszcza-lność** | **Rozwiązanie** | **Uwagi** |
| 1 | 2651842 | TAK |  | best |
| 2 | 2706900 | TAK |  |  |
| 3 | 2801768 | TAK |  |  |
| 4 | 2800755 | TAK |  |  |
| 5 | 2831708 | TAK |  | worst |

Przebieg optymalizacji:



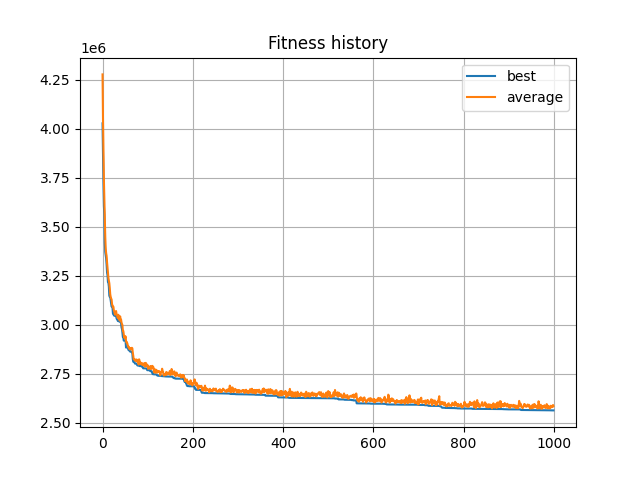
Uwagi:

Po raz kolejny potwierdza się, że mutacja Cauchy’ego zwraca bardziej zróżnicowane, jednak gorsze wyniki od gaussowskiej.

**7.7 M:Gaussian, convex X α dist: Arcsine, S: Tournament**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L. p.** | **Wartość f kosztu** | **Dopuszcza-lność** | **Rozwiązanie** | **Uwagi** |
| 1 | 2526126 | TAK |  |  |
| 2 | 2625436 | TAK |  | worst |
| 3 | 2571570 | TAK |  |  |
| 4 | 2486618 | TAK |  |  |
| 5 | 2463980 | TAK |  | best |

Przebieg optymalizacji:



Uwagi:

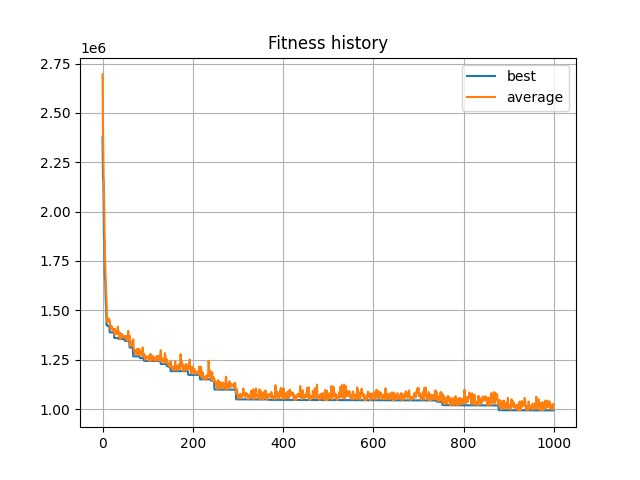
Co ciekawe, przeciwnie do testów w selekcji ruletkowej, tym razem dystrybucja arcsine parametru α kombinacji wypukłej zwraca lepsze wyniki od rozkładu jednostajnego. Funkcja kosztu tu osiągnięta jest najlepsza z badanych dotychczas kombinacji parametrów. **Tę kombinację przyjmę zatem w dalszych testach.**

**7.8 Przypadek skrajny – zerowy koszt śmierci**

Reszta parametrów bez zmian (w porównaniu z testami z poprzedniego podrozdziału).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L. p.** | **Wartość f kosztu** | **Dopuszcza-lność** | **Rozwiązanie** | **Uwagi** |
| 1 | 987459 | TAK |  | worst |
| 2 | 986401 | TAK |  | best |

Przebieg optymalizacji:



Uwagi:

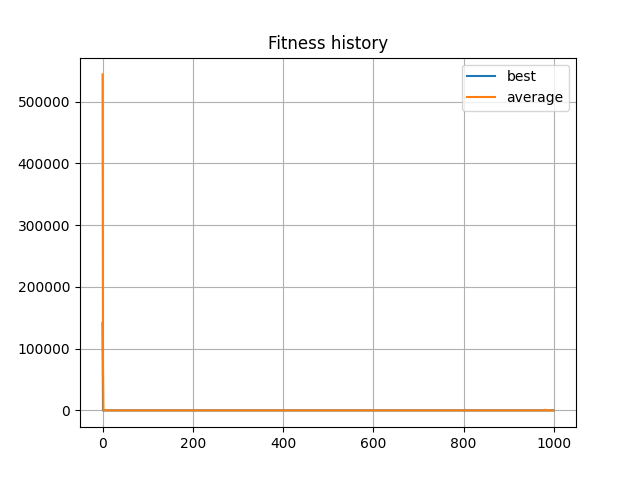
Tak jak się tego można było spodziewać, przy zerowym koszcie śmierci algorytm zwraca wektor, którego wartości są w przybliżeniu stałe i bliskie dolnemu ograniczeniu liczby zatrudnionych medyków – w tym przypadku 3 tysiące. Eksperyment potwierdza zatem poprawne działanie algorytmu.

**7.9 Przypadek skrajny – zerowe kosztu trenowania oraz utrzymania pracowników**

Reszta parametrów bez zmian (w porównaniu z testami z podrozdziału 7.7).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L. p.** | **Wartość f kosztu** | **Dopuszcza-lność** | **Rozwiązanie** | **Uwagi** |
| 1 | 0 | TAK |  | ideal |
| 2 | 0 | TAK |  | ideal |

Przebieg optymalizacji:



Uwagi:

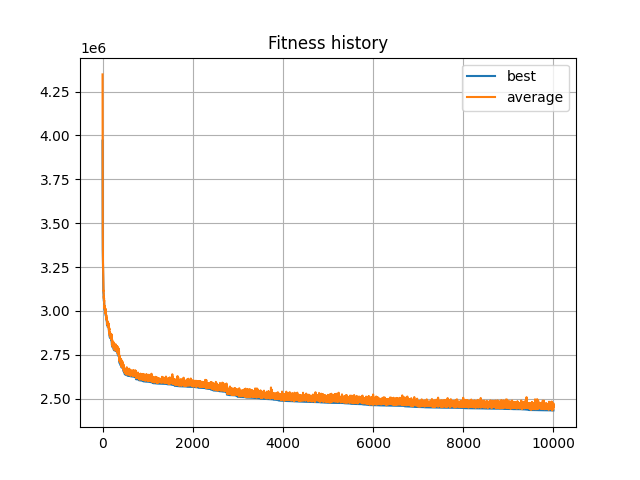
Tak jak się tego można było spodziewać, przy zerowym koszcie treningu oraz utrzymania pracowników algorytm zwraca rozwiązanie, w którym liczba medyków zawsze jest większa od liczby przypadków – wtedy zagwarantowany jest zerowy koszt.

**7.10 Większa liczba iteracji – 10 tysięcy**

Reszta parametrów bez zmian (w porównaniu z testami z podrozdziału 7.7).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L. p.** | **Wartość f kosztu** | **Dopuszcza-lność** | **Rozwiązanie** | **Uwagi** |
| 1 | 2440535 | TAK |  | worst |
| 2 | 2375645 | TAK |  | best |
| 3 | 2461927 | TAK |  |  |
| 4 | 2404942 | TAK |  |  |
| 5 | 2437540 | TAK |  |  |

Przebieg optymalizacji:



Uwagi:

Większa liczba iteracji powoduje poprawę funkcji kosztu najlepszego rozwiązania – jednak prędkość spadku krzywej optymalizacji jest mniejsza wraz ze zwiększającą się ilością iteracji.

**Podsumowanie.**

8.1. Wnioski.

Algorytm spełnia oczekiwania odnośnie do wydajności i jakości otrzymanych wyników. Uwagi na wielowymiarowość problemu algorytm został zaprojektowany z położeniem szczególnego nacisku na zachowywanie dopuszczalności rozwiązań. Jeżeli choćby jedno rozwiązanie w trakcie procesu optymalizacji przestało być dopuszczalne to można byłoby po już więcej nie doszukać się dopuszczalnych rozwiązań, to jest dalej iterować po rozwiązaniach bez większego sensu. Pomaga w tym implementacja krzyżowań zachowujących dopuszczalność, operatora rzutowania dla rozwiązania dopuszczalne, czy przede wszystkim konstruktora rozwiązań dopuszczalnych.

Można by się spodziewać, że rozwiązanie bliskie optymalnemu będzie mniej więcej odwzorowywać przebieg liczby zachorowań zakładając sensownie dobrane parametry funkcji kosztów. Otrzymane wyniki zdają się to potwierdzać. Również przeprowadzone testy dla specyficznie dobranej funkcji przystosowania dokładniej mówiąc jej parametrów znajdują rozwiązania bliskie tym, które można byłoby przewidywać po jako rozwiązania optymalne. Algorytm po dostrojeniu parametrów ogranicza czas bezsensownego błądzenia na rzecz celowej eksploracji, a potem eksploatacji rozwiązań.

Jako miarę lokalnej optymalności rozwiązań zostało przyjęte sprawdzenie o ile od otrzymanego wyniku różni się wynik optymalizowany dodatkowo hybrydowym algorytmem.

8.2. Stwierdzone problemy

Zasadniczym problemem była zbyt duża złożoność zagadnienia rozpatrywanego według dni tj. o wymiarze 365. W celu otrzymywania sensownych wyników po należało sprowadzić zagadnienie z podziałem na tygodnie tj. o wymiarze 52.

Umożliwiło to ograniczenie błądzenia algorytmu po nazbyt dużym zagadnieniu. Po zmianie otrzymywane wyniki zaczęły być bardziej satysfakcjonujące a ich polepszanie z użyciem algorytmu hybrydowego nie przynosiły znaczącej poprawy, co może świadczyć o ich lokalnej optymalności.

Problemem była również zbytnia złożoność obliczeniowa symulacji (nienaiwnej funkcji celu), z tego powodu zastosowane zostało podejście amalgamowane (optymalizacja przybliżonej funkcji celu).

8.3. Kierunki dalszego rozwoju

Algorytm został tak zaimplementowany, aby przede wszystkim umożliwić jego ciągły i dalszy rozwój. Funkcje zostały napisane w sposób możliwie ogólny co może przydać się nie tylko w trakcie przyszłych implementacji przyszłych popraw, ale również przydało nam się w trakcie transponowania problemu z dziennego na tygodniowy. Zamiast zmiany wielu parametrów w wielu klasach wystarczyło zmienić parametr długości rozwiązania i podać stosowny szereg czasowy.

Taka budowa pozwala również na przyszłe modyfikacje, które na razie mogłyby mu obejmować wprowadzenie mechanizmu uczenia Baldwina i Larmarcian’a. Nie ma również przeszkód, aby podana implementacja służyła do optymalizacji zgoła innego problemu. Wartym rozważenia są różne rodzaje sukcesji jak również uogólnienia czy po każdej mutacji ma następować rzutowanie na rozwiązania optymalne. Ciekawym byłby również mechanizm program wyspowego algorytmu ewolucyjnego, czy możliwość wyboru spośród kilku rodzajów mutacji i skrzyżowań w trakcie jednego przebiegu algorytmu.