Warszawa, 28.05.2020 r.

ALHE

Algorytmy heurystyczne

Skład zespołu: Patryk Łempio, Łukasz Wolanin
Opiekun: mgr inż. Kamil Deja

Dokumentacja końcowa
"Projekt z koroną"

1. Treść zadania.

W ramach projektu z koroną Państwa zadaniem jest uratowanie świata przed zabójczym wirusem.
Szalony naukowiec odkrył lek na chorobę covid-19. Niestety ze względu na jego złożoność może go dystrybuować tylko osobiście lecząc na raz wszystkich chorych znajdujących się w tym samym państwie. Zadanie utrudniają mu zamknięte lotniska i porty. W związku z tym poruszać może się jedynie pieszo bądź kajakiem pomiędzy sąsiadującymi krajami (posiadającymi granicę lądową lub morską). Podróż taka zawsze zajmuje mu 1 dzień. Korzystając z najnowszych danych dotyczących zachorowań https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-distribution-covid-19-cases-worldwide oraz z listy wszystkich granic pomiędzy krajami np: https://en.wikipedia.org/wiki/List_of_countries_and_territories_by_land_and_maritime_borders, opracuj możliwie najlepszą ścieżkę startującą z dowolnego kraju, tak by wyleczyć jak najwięcej ludzi. W ramach projektu proszę zaproponować i przetestować działanie różnych algorytmów.

2. Opis problemu.

Problem zdefiniowany w treści projektu polega na znalezieniu ścieżki w grafie reprezentującym granice lądowe i morskie pomiędzy krajami i terytoriami na kuli ziemskiej. Ścieżka ta reprezentuje trasę naukowca, który dysponuje lekiem na COVID-19 i odwiedzając dany kraj leczy wszystkich chorych w granicach kraju, przy czym leczenie nie zajmuje mu dodatkowego czasu. Do ewaluacji ścieżek zdefiniowaliśmy dwie funkcje celu:

- liczba wyleczonych pacjentów przez naukowca. Jest obliczana jako różnica sumy nowych przypadków zarejestrowanych do dnia, w którym naukowiec odwiedził kraj (włącznie) i sumy zgonów na COVID-19 przed tym dniem dla każdego państwa w ścieżce. Oczywiście w tym przypadku zadanie polega na maksymalizacji.
- liczba zgonów spowodowanych przez COVID-19. Zdefiniowana jako suma wszystkich zgonów przy założeniu, że po odwiedzeniu kraju przez naukowca nie pojawiają się w nim nowe zgony. Tę liczbę staramy się zminimalizować.

Wszystkie algorytmy, które zaimplementowaliśmy, do rozwiązania problemu przyjmują na wejściu jako argument dzień, w którym naukowiec rozpoczyna swoją podróż. Następnie rozwiązanie jest znajdowanie przez przeszukiwanie przestrzeni wszystkich ścieżek, które zaczynają się w dowolnym państwie w zadanym dniu, a kończą w ostatnim dniu dla którego dysponujemy danymi o zachorowaniach na COVID-19 lub też przez proste przeszukiwanie grafu krajów.
Dodatkowo przyjęliśmy że:

- jeśli na ścieżce znajduje się kraj dla którego nie posiadamy części bądź całości danych to w dniach dla których nie ma danych nie było nowych zachorowań czy zgonów,
- nowe przypadki Sars-CoV-2 pojawią się ponownie w kraju który został wyleczony przez doktora,
- algorytm może do wyboru kolejnego ruchu wykorzystywać dane z przyszłości,
- w przypadku drugiej funkcji celu po odwiedzeniu kraju przez doktora nie pojawiają się w nim nowe zgony. Jest to spowodowane tym, że nie posiadamy danych pozwalających stwierdzić, czy dany zgon w danych dotyczy osoby, która była uleczona przez doktora.

3. Opis rozwiązania i przebiegu prac.

3.1. Zbieranie i przygotowanie danych.

Realizację rozpoczęliśmy od zebrania i przygotowania potrzebnych danych. O ile zbiór danych o przypadkach COVID-19 należało jedynie pobrać i zapisać jako plik CSV, to już z danymi o granicach między państwami i terytoriami nie było tak prosto. Należało bowiem skorzystać z techniki web scrapingu. Dane były pobrane ze strony z Wikipedii przy użyciu biblioteki request, a następnie wyłuskane z pliku html za pomocą narzędzia *Beautiful Soup*. W ciągu dalszych prac okazało się, że istnieją pewne rozbieżności pomiędzy nazwami państw w zbiorze danych o COVID-19, a zbiorze danych o granicach. Został więc stworzony słownik który mapuje nazwy na ich odpowiedniki z drugiego zbioru. Dodatkowo z racji na zaszłości kolonialne (i nie tylko) niektóre terytoria zostały usunięte ze zbioru granic, ponieważ jednocześnie stanowią część innego kraju i w kontekście zdefiniowanego problemu nie ma sensu ich wyodrębnianie.

3.2. Implementacja klas reprezentujących graf i ścieżki

Implementację grafu oparliśmy na dwóch słownikach i liście. Pierwszy z nich przechowuje listę sąsiadów dla każdego państwa lub terytorium. Jest to więc reprezentacja krawędzi w grafie. Wierzchołki grafu, czyli poszczególne państwa, są przechowywane w postaci listy. Dane dotyczące przypadków COVID-19 w państwach są przechowywane jako słownik, gdzie kluczem jest nazwa państwa, a wartością tablica (numpy.ndarray), która przechowuje wycinek zbioru danych o zachorowaniach dla danego kraju. Każdy wiersz tej tablicy zawiera datę oraz liczbę nowych przypadków i zgonów. Jeśli nie posiadamy danych dla jakiegoś kraju, to tablica zawiera zerowe liczby zgonów i zachorowań dla daty 1 stycznia 1970. Graf posiada także metody m.in. metodę zwracającą listę sąsiadów kraju, metodę zwracającą statystyki dla danego kraju i metodę zwracającą liczbę aktywnych przypadków koronawirusa w danym dniu w danym państwie.

Klasa reprezentująca ścieżki przechowuje datę startu, listę krajów na ścieżce i ocenę wartości ścieżki przez zadaną funkcję celu. W

celu ułatwienia implementacji algorytmów klasa Path posiada metody do dodawania kraju do ścieżki, do ewaluacji ścieżki, oraz do oceny zysku wynikającego z dodania kraju do ścieżki.

3.3. Implementacja algorytmu brutalnego

Algorytm brutalny został zaimplementowany jako klasa, której konstruktor przyjmuje parametry:

- kraj, od którego zaczyna się podróż,
- dzień, w którym rozpoczyna się podróż,
- graf, na którym ma działać,
- funkcja celu której będzie używać algorytm
    - 'recovered' - liczba wyleczonych pacjentów
    - 'deaths' - liczba zgonów na COVID-19.

Aby rozpocząć działanie algorytmu, należy wywołać metodę work(), która pozwoli na wyznaczenie najlepszej ścieżki znajdującej się w grafie.

Działanie algorytmu jest proste - dla podanego kraju startowego, generowane są wszystkie możliwe ścieżki poprzez kolejne dodawanie sąsiadów. Generowane w ten sposób ścieżki są przechowywane w liście, która stale się powiększa, aż do momentu wystąpienia warunku kończącego algorytm. W naszym przypadku jest to nadejście ostatniego dnia, dla którego dostępne są dane o zachorowaniach.

Algorytm, zważywszy na to, że generuje wszystkie możliwe ścieżki w grafie, zwraca ścieżkę najlepszą, jaka jest możliwa do uzyskania. Jego kosztem za to jest bardzo szybko rosnący czas wykonywania. To sprawia, że algorytm staje się bardzo nieużyteczny i mało zoptymalizowany do pracy na dużej przestrzeni przeszukiwań.

3.4. Implementacja algorytmu Best First

Algorytm Best First został zaimplementowany jako klasa, której konstruktor przyjmuje parametry:

- kraj, od którego zaczyna się podróż,
- dzień, w którym rozpoczyna się podróż,
- graf, na którym ma działać,
- funkcja celu której będzie używać algorytm
    - 'recovered' - liczba wyleczonych pacjentów

    - 'deaths' - liczba zgonów na COVID-19.

Aby rozpocząć pracę algorytmu, należy również wywołać metodę work(), która wyznacza nam ścieżkę zgodną z pracą opisywanego algorytmu.

Algorytm Best First jest algorytmem naiwnym - wybiera sobie wartość, która w danym momencie jest dla niego najlepsza. W naszym projekcie, najlepszą wartością jest sąsiad, który następnego dnia ma jak największą liczbę aktualnych zachorowań (rozważamy tu przypadek funkcji celu ‘recovered’).

Zaczynając od podanego państwa, algorytm dołącza kolejno najlepszy kraj spośród sąsiadów ostatniego kraju, który znajduje się na dotychczas wyznaczonej ścieżce. Warunkiem końcowym jest dojście do momentu, w którym nie ma już więcej danych (ostatni dzień z danymi). W ten sposób pod koniec działania algorytmu Best First, dostajemy jedną ścieżkę, która jest także naszym wynikiem.

3.5. Implementacja algorytmu Ewolucyjnego

Algorytm Ewolucyjny został zaimplementowany jako klasa, której konstruktor przyjmuje jako argumenty:

- graf, na którym ma działać,
- dzień, w którym rozpoczyna się podróż,
- liczność populacji początkowej (μ),
- liczność zbioru do reprodukcji (λ),
- prawdopodobieństwo mutacji pojedynczej ścieżki,
- sposób wyboru osobników do następnego pokolenia,
    - 'best' - μ najlepszych
    - 'roulette' - metoda koła ruletki
    - 'ranking' - selekcja rankingowa
- liczba pokoleń, przez które będzie działać algorytm,
- funkcja celu której będzie używać algorytm
    - 'recovered' - liczba wyleczonych pacjentów
    - 'deaths' - liczba zgonów na COVID-19.

Następnie uruchomienie algorytmu następuje przez wywołanie metody find_path(), która zwraca najlepszego osobnika (ścieżkę), jakiego podczas pracy znalazł algorytm.

Algorytm działa tak, że w każdym pokoleniu znajduje się μ osobników. Początkowa populacja jest losowana z przestrzeni ścieżek o zadanej długości. Następnie poprzez losowanie ze zwracaniem z aktywnej populacji wyłaniana jest grupa osobników o

liczności λ, które będą krzyżowane ze sobą. Proces krzyżowania jest zorganizowany tak, że dla każdej pary osobników jest sprawdzane czy mogą być krzyżowane. Jeśli nie, to algorytm przechodzi do kolejnej pary, jeśli tak to osobniki są krzyżowane ze sobą i nie są później rozpatrywane jako kandydaci do krzyżowania. Dzieje się tak dopóki skrzyżujemy ze sobą λ osobników lub nie będzie możliwości dalszego krzyżowania. Warunkiem jaki muszą spełnić dwie ścieżki, aby je skrzyżować jest posiadanie wspólnego państwa na nieskrajnych pozycjach. Samo krzyżowanie jest zorganizowane jak na przykładzie (w którym długość wszystkich ścieżek wynosi 5):

P1: 1-3-**4**-7-2 P2: 9-8-2-**4**-5

O1: 8-2-**4**-7-2 O2: 9-8-2-**4**-7

Jeśli wspólny kraj będzie na tej samej pozycji w obu ścieżkach, to są tylko zamieniane kraje występujące po wspólnym. W innym przypadku ścieżki są łączone z naddatkiem, a następnie jeden wariant przycinany na początku, a drugi na końcu. Taka implementacja krzyżowania sprawia, że potomstwo w dużym stopniu przypomina przodków i dziedziczy część cech. Kiedy mamy już skrzyżowane potomstwo, to w zależności od parametrów część poddajemy mutacji, która polega na podmianie jednego państwa na ścieżce. Następnym krokiem algorytmu jest selekcja osobników z reprodukowanych i tych z aktywnego pokolenia do następnego pokolenia. Zaimplementowane metody selekcji to:

- μ najlepszych,
- metoda koła ruletki,
- selekcja rankingowa.

Przy selekcji algorytm ewaluuje ścieżki i sprawdza, czy nie pojawiła się ścieżka lepsza od dotychczas najlepszej. Algorytm działa przez zadaną liczbę pokoleń.

3.6. Implementacja algorytmu Przeszukiwania z Tabu

Algorytm Przeszukiwania z Tabu został zaimplementowany jako klasa, której konstruktor przyjmuje argumenty:

- graf na którym ma działać,
- dzień w którym naukowiec zaczyna podróż,
- długość tabu,
- funkcja celu
    - -'recovered'- liczba wyleczonych pacjentów
    - -'deaths' - liczba zgonów na COVID-19.

Algorytm ten działa tak, że na początku losuje osobnika początkowego, a następnie rozpoczyna przeszukiwanie przestrzeni ścieżek o zadanej długości wpisując kolejnych znalezionych osobników do zbioru tabu. Dzięki tabu, algorytm nie powraca do rozważanych wcześniej osobników. Chociaż zależy to od wielkości tabu, bo jeśli wielkość tabu nie jest duża, to osobniki będą je opuszczać i stanie się możliwy powrót do nich. W naszej implementacji algorytm pomija osobniki z tabu przy generowaniu sąsiadów obecnego osobnika. Przeszukiwanie przestrzeni rozwiązań przez ten algorytm polega na znajdowaniu sąsiadów aktualnego osobnika i następnie wykonuje się podmiana aktywnego osobnika na jego najlepszego sąsiada. W przypadku naszego problemu sąsiadami osobnika są wszystkie ścieżki, które różnią się od niego tylko na jednej pozycji. Algorytm w każdej iteracji zapisuje dotychczas najlepszego osobnika, aby później go zwrócić. Działanie algorytmu kończy się po określonej liczbie iteracji lub gdy aktywny osobnik nie będzie miał sąsiadów (sytuacja taka możliwa jest jedynie wtedy, kiedy wszyscy sąsiedzi będą znajdowali się w tabu).

4. Obserwacje i wnioski.

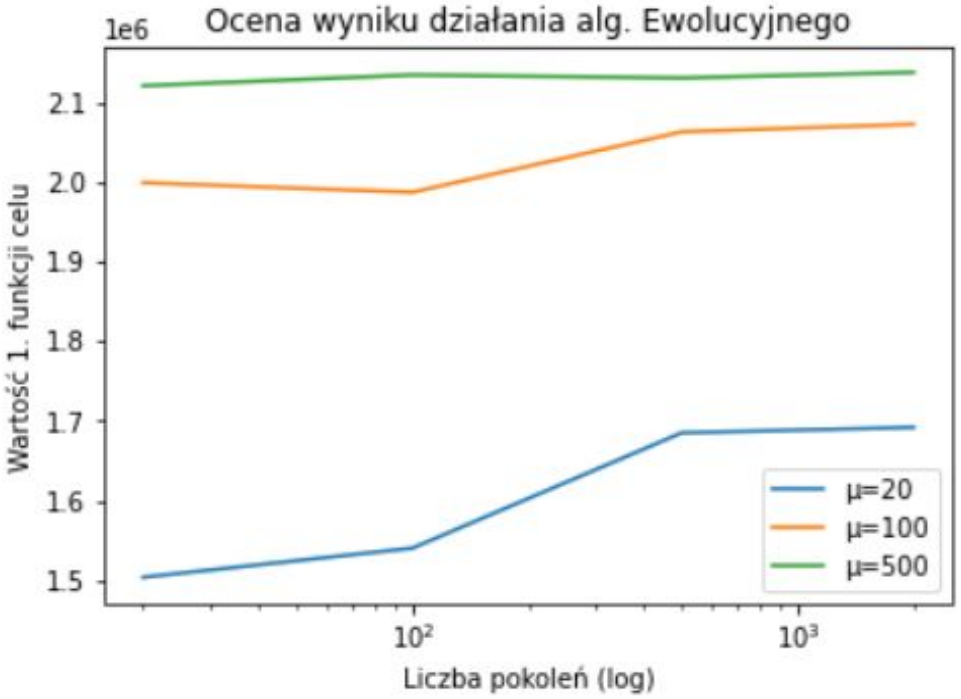
4.1. Przebieg testów

W trakcie implementacji algorytmów sprawdzaliśmy ich poprawność przeprowadzając testy poprawności, które pozwoliły znaleźć i usunąć błędy. Następnie przystąpiliśmy do właściwego testowania rozwiązań. Zbadaliśmy wpływ parametrów poszczególnych algorytmów na wyniki i czas ich działania. Testy te zajęły dużo czasu. W szczególności przetestowanie wpływu wielu parametrów na zachowanie algorytmu ewolucyjnego było długie i zajęło kilka dni.

4.2. Obserwacje i wnioski dotyczące algorytmu Ewolucyjnego.

Pierwszą rzeczą która została sprawdzona był wpływ liczebności populacji na wyniki działania algorytmu. Dla parametru μ reprezentującego liczbę osobników w populacji o wartościach odpowiednio 20, 100, 500 sprawdziliśmy jaką wartość funkcji celu osiągnie najlepszy znaleziony osobnik. Przy czym reszta parametrów była następująca: metoda selekcji - μ najlepszych, $\lambda = 1,2\mu$, prawdopodobieństwo mutacji = 0,1.

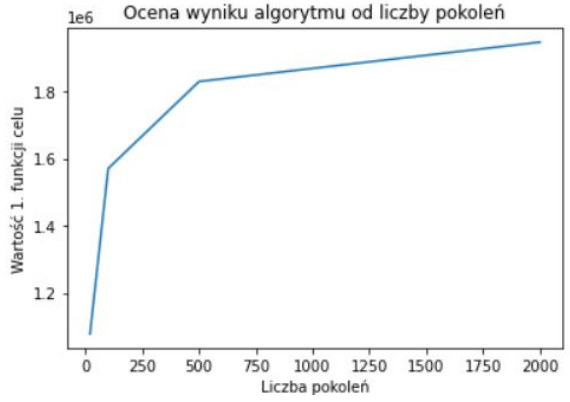
Uzyskane wyniki zostały przedstawione na wykresie poniżej:

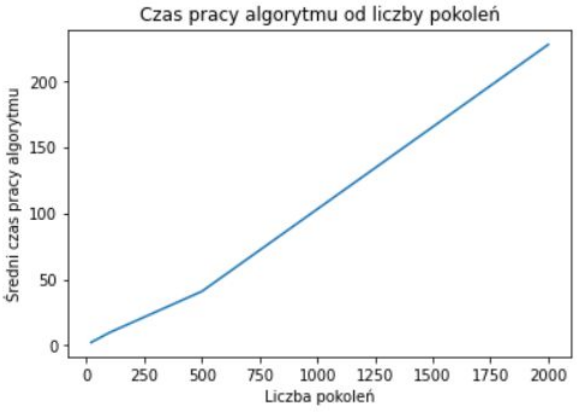


Jak widać liczebność populacji w algorytmie ma bardzo duży wpływ na wynik. Największa rozbieżność jest przy małej ilości pokoleń, natomiast dla większej liczby pokoleń wynik niewiele zmienia się dla μ = 500 i poprawia się o wiele bardziej dla μ = 20. Wniosek z tego taki, że dla przedstawionej w zadaniu przestrzeni przeszukiwania wylosowanie populacji zawierającej 500 osobników daje spore szanse, że w ciągu kilku pokoleń znajdziemy w niej osobnika o dużej wartości przystosowania. Krótko mówiąc, kiedy mamy dużą populację to szybciej przeszukujemy możliwe ścieżki. Przedłużanie poszukiwań nie zwiększa już tak mocno wyniku. Natomiast im mniejsza populacja tym dłużej będziemy szukać dobrego osobnika. Przy μ równych 20 i 100 widać ustabilizowanie po 500 pokoleniach. Jest to prawdopodobnie spowodowane tym, że algorytm zaczął eksploatować maksimum lokalne funkcji celu (metoda selekcji μ najlepszych nie najlepiej radzi sobie z wychodzeniem z maksimum lokalnego). Przy μ = 500 populacja jest na tyle duża, że po kilku pokoleniach algorytm zaczyna eksploatację wielu maksimów, w tym także tego globalnego.

Na kolejnych dwóch wykresach przedstawiono porównanie skuteczności działania algorytmu Ewolucyjnego w zależności od zastosowanej metody selekcji osobników do następnego pokolenia. Wpływ metody selekcji zbadaliśmy przy reszcie parametrów algorytmu odpowiednio: μ = 500, λ = 1,2μ, prawdopodobieństwa mutacji = 0,5 i liczbie pokoleń = 100.

Jak widać najlpsze wyniki dawała medoda selekcji koła ruletki, a najsłabiej wypadła metoda μ najlepszych. Jest to najprawdopodobniej spowodowane zbyt silną presją selekcyjną w przypadku metody μ najlepszych oraz tym, że metoda koła ruletki pozwoliła na lepszą eksplorację przestrzeni rozwiązań co częściej doprowadzało do znalezienia lepszych rezultatów. Jeśli chodzi o czasy wykonania, to znów najgorzej wypadła metoda μ najlepszych, ponieważ przy niej trzeba sortować osobniki przed dokonaniem selekcji. Najlepiej czasowo wypadła metoda selekcji rankingowa i była ona także niewiele gorsza od metody koła ruletki, jeśli chodzi o wyniki. To sprawia że najtrafniejszą metodą selekcji dla problemu zdefiniowanego w zadaniu wydaje się metoda rankingowa.





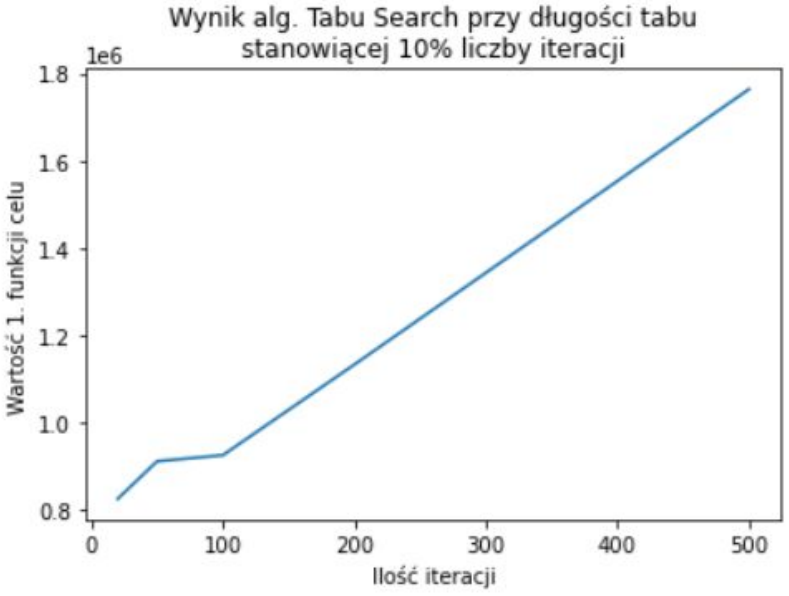
Powyżej znajdują się dwa wykresy ukazujące zależność kolejno wartości funkcji celu dla najlepszego znalezionego osobnika i czasu pracy w zależności od liczby pokoleń. Wartości te zostały zmierzone przy parametrach: selekcja rankingowa, prawdopodobieństwo mutacji 0,5, μ = 20 i λ = 24. Jak widać wartość funkcji celu gwałtownie rośnie na początku przeszukiwania, a następnie rośnie już wolniej. Można to interpretować w ten sposób, że na początku startujemy z losowymi osobnikami, których przystosowanie nie jest szczególnie wysoko, więc łatwo jest przez krzyżowanie i mutację znaleźć osobników lepiej przystosowanych. Im lepszych mamy osobników tym wolniej rośnie wartość

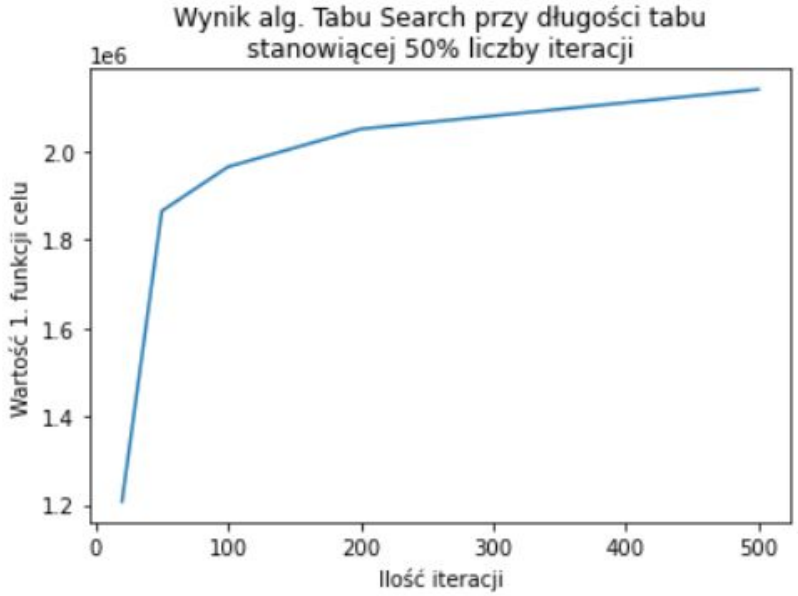
funkcji celu, ponieważ coraz trudniej jest znaleźć lepszych osobników. Oczywiście liczba pokoleń, po której wartość funkcji celu zacznie się stabilizować jest zależna od przestrzeni rozwiązań. W tym przypadku była to przestrzeń ścieżek o długości 14 dni rozpoczynających się 11 maja. Zależność czasu wykonania od liczby pokoleń jest w przybliżeniu liniowa, czemu trudno się dziwić.

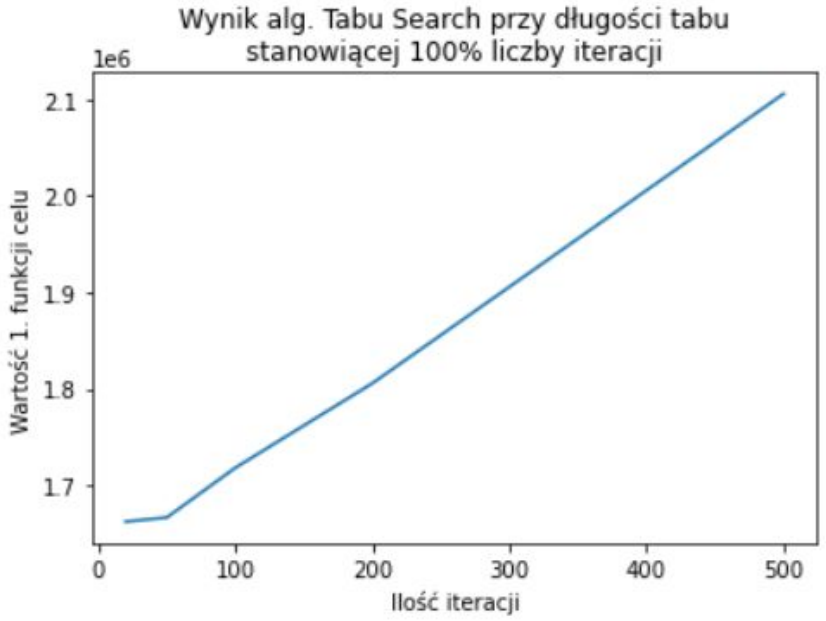
Zbadaliśmy również zależność jakości wyniku działania algorytmu od czynnika reprezentującego p-ństwo mutacji mutacji osobników. Przy wartości innych parametrów: μ = 20, λ = 24, liczbie generacji = 60 i metodzie selekcji rankingowej Uzyskane wyniki są dość zaskakujące. Najlepiej algorytm sprawował się przy prawdopodobieństwie równym 0,25, ale już dla prawdopodobieństwa równego 0,5 wyniki były o wiele gorsze, żeby znów się poprawić dla wyższych wartości parametru. Na pewno przy częstszej mutacji więcej nowych krajów będzie pojawiać się w ścieżkach, dla tego przeszukane będzie większa część przestrzeni rozwiązań. Natomiast dla niższych wartości parametru mutacji jeśli populacja bazowa będzie słaba to ciężko będzie znaleźć dobre rozwiązanie. Większa liczebność populacji obniża wpływ mutacji na wynik działania algorytmu.

4.3. Obserwacje i wnioski dotyczące przeszukiwania z tabu

Przy algorytmie przeszukiwania z tabu jedynymi parametrami jakie możemy ustawić są długość tabu i ilość iteracji. Oczywiście można dodatkowo dołączyć do tego definicję sąsiedztwa, czy będzie ona szersza lub węższa. Mu jednak przyjęliśmy, że dwie ścieżki sąsiadują jeśli różnią się tylko jednym krajem (oczywiście muszą mieć tą samą długość i daty rozpoczęcia). W ramach testów algorytmu przeszukiwania z tabu zbadaliśmy wpływ ilości iteracji dla różnych długości tabu.







Z powyższych wykresów wynika, że dla pierwszych 50 iteracji wartość funkcji celu najszybciej rośnie dla długości tabu równej 250 (połowa z 500 iteracji) jest to sytuacja, kiedy w tabu wciąż znajdują się wszystkie ścieżki rozpatrywane wcześniej. Później następuje zwolnienie tempa wzrostu, które raczej nie jest spowodowane długością tabu, a po prostu tym, że osiągnięto w miarę wysokie wartości funkcji celu. W ciągu 500 iteracji największą wartość funkcji celu uzyskał algorytm z tabu długości równej liczbie iteracji. Był on najskuteczniejszy ponieważ z racji na długość tabu nie mógł się nigdy cofnąć do poprzedniej ścieżki i przeszukał ich najwięcej. Przy długości ścieżek równej 14 każda ma całkiem spory zbiór sąsiadów, jednak gdyby był on mniejszy algorytm mógłby zakończyć pracę przedwcześnie i być może powinniśmy rozważyć wykorzystanie krótszego tabu.

4.4. Obserwacje i wnioski dotyczące algorytmów brutalnego oraz Best First

Zważywszy uwagę na fakt, że oba z tych algorytmów nie posiadają zależności wynikających z istnienia zmiennych parametrów, testy oraz badania tych algorytmów nie przyniosłyby dużej ilości informacji. Wiemy na pewno, że porównując czasy ich działania, to algorytm brutalny wykonuje się zdecydowanie dłużej od algorytmu BestFirst. Wadą jest również to, że zajmuje on o wiele większą część zasobów obliczeniowych maszyny, na której działa. Daje on jednak czasami znacząco lepsze wyniki, niż drugi z nich.

4.5. Podsumowanie wniosków

Z przeprowadzonych testów wynika, że algorytm przeszukiwania przestrzeni z tabu lepiej radził sobie ze znajdowaniem dobrych ścieżek od algorytmu Ewolucyjnego. Średnio zwracał lepsze wyniki, przy krótszym czasie przeszukiwania. Jednak nie można na tej podstawie stwierdzić, że algorytm Ewolucyjny jest gorszy od algorytmu przeszukiwania z tabu, ponieważ ten pierwszy jest bardziej uniwersalny.

Jeśli chodzi o algorytmy nieheurystyczne, to ich działanie w projektach jak ten pozostawia wiele do życzenia. Występuje tu duży brak zbalansowania pomiędzy czasem ich wykonywania (brutalny), jak i wynikiem ich pracy (algorytm Best First często daje bardzo słabe wyniki w porównaniu do 3 innych zaimplementowanych algorytmów).

Warto jest więc wykorzystywać algorytmy heurystyczne, które są pewnym złotym środkiem na problemy optymalnego poruszania się po przestrzeni przeszukiwań.

5. Informacje dodatkowe.

5.1. Wymagania

Do implementacji algorytmów i ich uruchomienia są wykorzystywane:

- Python 3.6
- numpy 1.18.1
- pandas 1.0.1

Dodatkowo do przygotowania danych i testowania wykorzystano:

- jupyter 1.0.0
- beautifulsoup4 4.9.0
- requests 2.23.0
- matplotlib 3.2.0

5.2. Użycie

Aby użyć któregoś z algorytmów należy najpierw wczytać graf za pomocą funkcji get_graph(). Można także użyć funkcji refresh_covid_data() aby odświeżyć zbiór danych o przypadkach COVID-19.

Użycie algorytmu Ewolucyjnego jest najłatwiejsze przez funkcję Evolutionary.search(), informacja o przyjmowanych argumentach w pliku Evolutionary.py.

Użycie algorytmu przeszukiwania z tabu jest najłatwiejsze przez funkcję Tabu.search(), informacja o przyjmowanych argumentach w pliku Tabu.py.

Użycie algorytmu brutalnego jest najłatwiejsze przez funkcję BruteAlgorithm.work(), informacja o przyjmowanych argumentach w pliku BruteAlgorithm.py.
Użycie algorytmu Best First jest najłatwiejsze przez funkcję BestFirst.py, informacja o przyjmowanych argumentach w pliku BestFirst.py.