Szymon Pilawka
Karol Krzekotowski

Etap 3 – Algorytmy populacyjne

Wprowadzenie

Wraz z kursem kończą się rozważania na temat potencjalnych heurystyk do rozwiązywania problemu komiwojażera. Jako ostatnią metaheurystykę, rozważać będziemy algorytm genetyczny.

Algorytm genetyczny, jak sama nazwa wskazuje, czerpie inspirację z natury, generyki, faktu krzyżowania się osobników w celu wydania na świat najlepszego potomstwa. Algorytm metaheurystyczny GA operuje też na określeniu populacja – zbiorze permutacji skazanych na współżycie, rywalizację i łączenie.

Jednostką czasu w algorytmie jest generacja – zbiór zdarzeń gwarantujący kontynuację – utworzenie kolejnej generacji, możliwie najmocniejszej. W każdej generacji pewne są dane kroki: **Krzyżowanie**, **Mutacja**, **Wybór nowej populacji**. Aby zapobiec stagnacji i zwiększyć różnorodność zaimplementowano kroki jak: **Migracja, Wymieranie.**

Implementacja

Nasza implementacja celuje w mniejsze, lecz bardziej dynamiczne populacje. Krzyżowanie ma w sobie element losowości aby nie doszło do stagnacji. Aby populacja nie była zbyt zbyt dynamiczna, wprowadziliśmy osobników „Elitarnych”, mających stu procentową szansę uczestnictwa w kolejnej generacji. Osobnicy mają określony wiek – zbyt starzy osobnicy zostają wykluczeni (czyt. zabici).

Cykl każdej generacji prezentuje się następująco:

A. Wybór rodziców i krzyżowanie
B. Utworzenie nowej populacji
C. Mutacje

A. Wybór rodziców odbywa się za pomocą tzw. ruletki. Wpierw osobnicy klasyfikowani są na podstawie ich potencjału – funkcji celu. Ruletka działa w następujący sposób: Osobnicy z większym potencjałem mają większe szanse na zostanie wybranym. W programie mamy dostępne trzy metody krzyżowania, domyślną metodą jest Single Point Crossover (SPX).

B. Nowa populacja tworzona jest w stosunku: 3 + 1:2:2, gdzie 3 to liczba elit przechodzących dalej (chyba że ich czas nadszedł), następnie jedna piąta z pozostałych członków to losowi przedstawiciele z poprzedniej generacji, a pozostałe miejsca są na pół podzielone pomiędzy rodziców wybranych ruletką a ich dzieci.

C. Mutacje są istotne w aspekcie utrzymywania różnorodności populacji, poprawna mutacja będzie opłacalna dla osobnika w dalszym terminie.

Poza przepływem głównym, na wyspach z mniejszą szansą wystąpienia mogą dziać się wpływowe zdarzenia. Pierwszym z nich jest **migracja** pomiędzy wyspami. Ponownie ruletką, wybieramy najbardziej obiecujących osobników do emigracji (faktycznie opuszczają wyspę). Drugim z nich jest **wymieranie** – najmniej obiecująca wyspa zostaje zniszczona, a przeżywa jedynie mała część populacji. Braki w populacji wypełnione zostają nowo wygenerowanymi osobnikami.

Warunkiem stopu algorytmu jest osiągnięcie maksymalnej ilości generacji. Z punktu widzenia algorytmu nie jesteśmy w stanie określić, kiedy zostanie osiągnięte minimum globalne w celu prędszego zakończenia pracy.

Badanie złożoności

Teoretyczna

Czas działania algorytmu jest zależny od wielu zmiennych występujących w programie. Podstawowym czynnikiem jak i wyznacznikiem jest **ilość generacji** (G), podawana na wejściu. Do innych czynników należą: **rozmiar grafu** (N), **rozmiar populacji** (P). W implementacji główny przepływ każdej generacji dzieli się na 5 kroków:

A. Wybór rodziców
B. Krzyżowanie
C. Utworzenie nowej populacji
D. Mutacje
E. Sortowanie

Zaczynamy od kroku **A**: Wyboru rodziców. Losowanie odbywa się za pomocą **ruletki**. Ruletka dobiera losową liczbę do przedziałów w taki sposób, że wartości z większą wagą mają lepsze szanse bycia wybranym. Wygenerowanie ruletki odbywa się przy inicjacji algorytmu, i ma złożoność czasową zależną od ilości przedziałów – czyli stały. Tak samo losowanie pojedynczej wartości z wygenerowanej ruletki również ma złożoność stałą.

$$O_A(n)=O(P)$$

Krok **B**: Krzyżowanie jest niewątpliwie najważniejszą częścią algorytmu. W programie mamy dostępne 3 metody: Order Crossover (OX), Single Point Crossover (SPX), Cycle Crossover(CX). Wygenerowanie obiecującego potomstwa definiuje dalsze działanie algorytmu.
W najgorszym przypadku krzyżowaniu może być poddany cały zbiór rodziców. Przeanalizuję konkretne metody krzyżowania.

a) **OX** możemy podzielić na kilka etapów. Pierwszym jest „pocięcie” rodziców zajmujące O(1)*O(N), następnie przeklejenie korpusu z pierwszego rodzica, a następnie wygenerowanie listy pozostałych elementów w oparciu o wartości nie obecne w nowym dziecku – ten krok jest zdecydowanie najtrudniejszy złożeniowo, dla każdego elementu należy sprawdzić czy nie znajdował on się w korpusie. Krok sprawdzenia elementu dla k – rozmiaru korpusu ma złożoność O(N*k), gdzie k w najgorszym przypadku jest równe N. Następnie rozładowujemy kolejkę co ma złożoność O(N).
Dla OX złożoność jednego krzyżowania wynosi $O(N^2)$

b) **SPX** jest zdecydowanie prostszy, dzielimy pierwszego rodzica na dwie części, bierzemy początkową część, a resztę wypełniamy drugim rodzicem. Przy wypełnianiu sprawdzamy, czy punkt nie znajduje się już na liście. Najgorszym przypadkiem jest wybranie punktu na samym początku, mamy wtedy po N kroków o średniej długości N/2. Złożoność:
$O(N^2)$

c) **CX** polega na znalezienu cykli, a następnie sklejeniu dzieci używając tych właśnie cykli. Przy kroku generowaniu tabliczki w najgorszym przypadku przejdziemy po każdym wierzchołku co najwyżej 1.5 raza (przypadek że mamy same cykle dwuelementowe). Każdy krok zawiera istotny krok znalezienia indeksu wartości z drugiego rodzica w pierwszym, co może zająć O(N). Dlatego krok generowania tabliczki wynosi $O(N^2)$. Następnym krokiem jest wygenerowanie dzieci z dostępnej tabliczki, co powinno wyniesie O(N). Złożoność całego krzyżowania wynosi $O(N^2)$.

Jak przeanalizowaliśmy, złożoność każdej z metod jest taka sama asymptotycznie, więc krok wygenerowania potomstwa wynosi $O_B(n)=O(P*N^2)$.

Krok **C**: Nowa populacja. Mamy dostępne 4 zbiory: Elity, Stara Populacja, Rodzice, Dzieci. Proces doboru trzech ostatnich zbiorów to losowe wybieranie osobników. Utworzenie nowego zbioru to krok zależny jedynie od rozmiaru populacji (oraz kosztu przekopiowania danych, który pomijamy).
$O_C(n)=O(P)$

Krok **D**: Mutacje. Każdy osobnik może być poddany maksymalnie 1 mutacji. Domyślną metodą mutacji jest invert (złożoność O(N)), jednak istnieje możliwość wykonania mutacji swap (złożoność O(1)). Skupimy się na worst case scenario (invert).
Wpierw wybieramy osobnika oraz usuwamy go z populacji (złożoność zależna od struktury danych, w liście wskaźnikowej użytej w programie jest to złożoność O(N) – dostępu do elementu i usunięcia - O(1)).
Po wykonaniu mutacji następuje ponowne policzenie funkcji celu - (O(N)). W tym momencie proces mutacji kończy się.
Następnie osobnik wraca do populacji, zostaje doklejony na koniec.
$O_D(n)=O(P*N)$

Krok **E**: Sortowanie. Jak możemy zobaczyć, w procesie utworzenia nowej populacji oraz mutacji lista traci kolejność rosnącą, wymaganą do ruletki przy procesie wybierania rodziców. Sortowanie jest wbudowane w bibliotekę Pythona i ma złożoność $O_E(n)=O(P*\log(P))$

Poza głównym przepływem mogą przytrafić się zdarzenia mniej prawdopodobne: **Migracje** oraz **wymieranie**.

Migracja: Grupy migrujące pomiędzy wyspami są dobierane za pomocą ruletki, należy jednak sprawdzić, czy na liście indeksów migrujących osobników nie występują duplikaty. Niech Mig – liczba osobników migrujących. Złożoność generowania grup wynosi $O(Mig^2)$, w przypadku naszego programu wartość Mig zależna jest od rozmiaru populacji (P/10).
Podobnie jak przy mutacji, następuje wklejenie nowych osobników oraz posortowanie $O(P*Mig+P*\log(P))$.
W implementacji krok ma złożoność $O(P^2+P*\log(P))$.

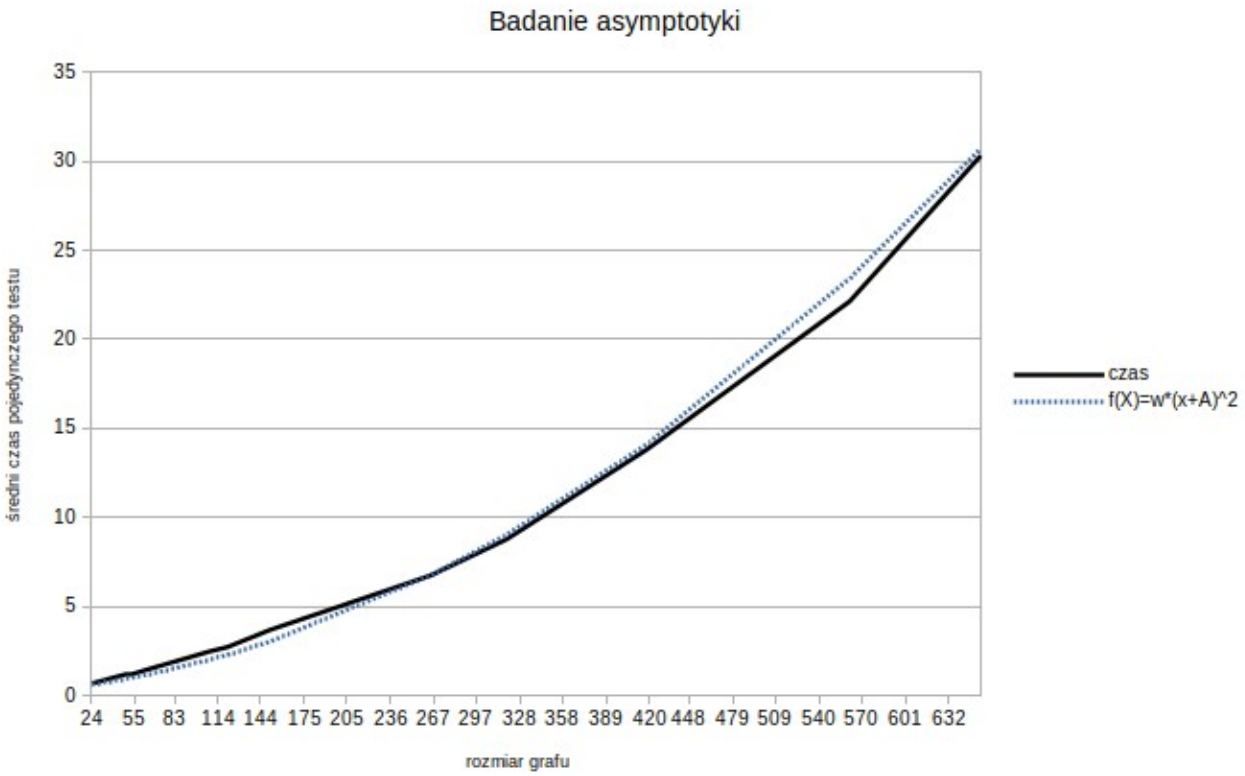
Wymieranie: Proces wymierania losowo eliminuje część populacji (może być nawet cała), a następnie generuje nowych członków – generacja jest mieszanką metod krandom oraz NearestNeighbour, zatem maksymalnie generacja takich członków kosztuje czasowo $O(N^2)$.
Krok ma złożoność $O(P*N^2)$.

Z wyłączeniem zdarzeń mało prawdopodobnych możemy obliczyć złożoność algorytmu następująco:

$$O_{GA}(n)=G*(O_A(n)+O_B(n)+O_C(n)+O_D(n)+O_E(n))$$
$$O_{GA}(n)=G*(O(P)+O(P*N^2)+O(P)+O(P*N)+O(P*\log(P)))$$
$$O_{GA}(n)=O(GPN^2+GPlog(P))$$

Praktyczna

Zbadanie złożoności w zależności od N odbyło się używając tych samych parametrów dla macierzy różnych rozmiarów. Otrzymane wyniki porównujemy do funkcji $f(x)=w*(x+A)^2$, gdzie w=0.00057, A=80.

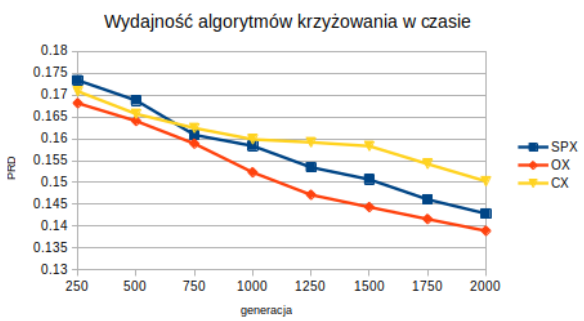


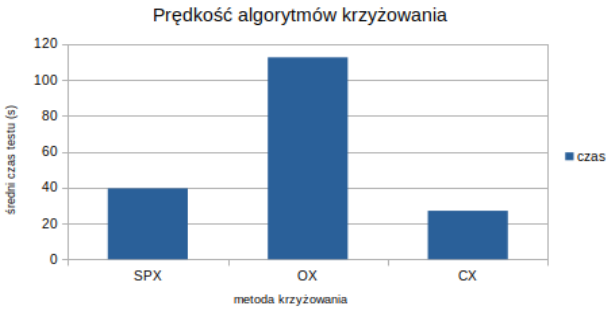
Zbiór danych do N=24 do N=724 wskazuje pokrywanie się faktycznego czasu z funkcją kwadratową f(x) oraz potwierdza oszacowania wykonane w części teoretycznej.

Test I – porównanie metod krzyżowania

Metody krzyżowania odgrywają istotną rolę przy tworzeniu kolejnych generacji. Z badania asymptotyki wiemy, że trzy dostępne metody (CX,OX,SPX) mają podobną złożoność, jednak czy któraś z nich jest znacząco lepsza od pozostałych?

Test ma na celu zbadanie jakiej poprawy możemy spodziewać się od wybranej metody. Duża poprawa wyników jest oznaką efektywnego krzyżowania osobników.





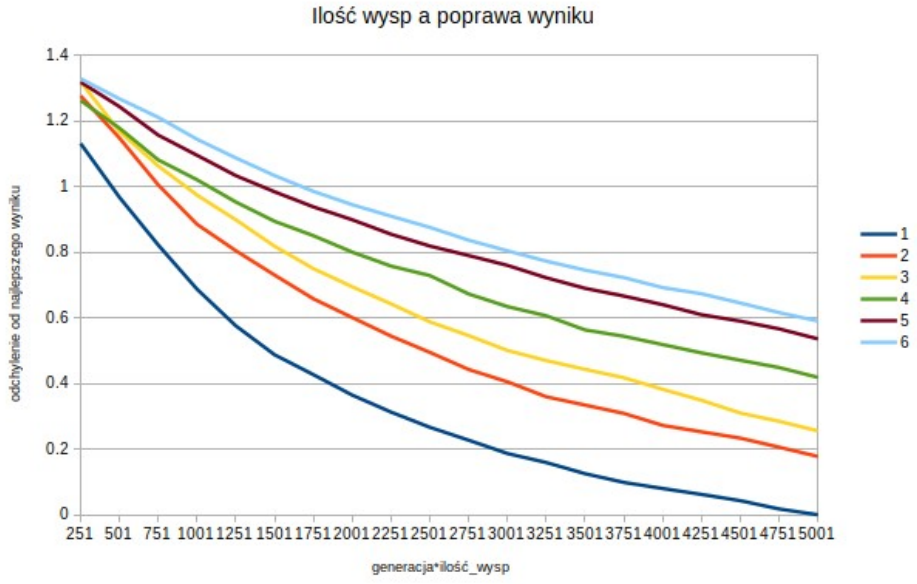
Wyniki wskazują, iż odpowiedź na pytanie która metoda krzyżowania jest najlepsza nie jest jednoznaczna. Widzimy mocne i słabe strony, Order Crossover mimo najlepszych wyników działa stanowczo najwolniej. Dobre wyniki zawdzięczamy dzięki dobremu połączeniu pomiędzy sklejaniem naturalnym oraz skutecznym. Metoda Cycle Crossover wydaje się najbardziej naturalną metodą krzyżowania, jednak generowane dzieci są po części miszmaszem złożonym z losowych pętli. Jest natomiast najszybsza. Metoda Single Point Crossover wydaje się dobrym pośrednikiem pomiędzy.

Przy dobieraniu optymalnej metody, pomijałbym CX z powodu znacznie słabiej rokujących wyników, natomiast pozostałe dwie metody wybierałbym w zależności od skupienia na krótkim czasie działania czy poprawie przy mniejszej ilości generacji.

Test II – dobranie ilości wysp

Algorytm oferuje utworzenie tzw. wysp – czyli osobnych populacji, które raz na jakiś czas wymieniają się genotypem. Do zalet tworzenia wysp należy znacznie zwiększona różnorodność populacji, zmniejszona szansa stagnacji. Wyspy mogą jednak spowolnić proces poprawy, obiecujący osobnicy nie mają aż tak wielkiego wpływu na ogół symulacji.

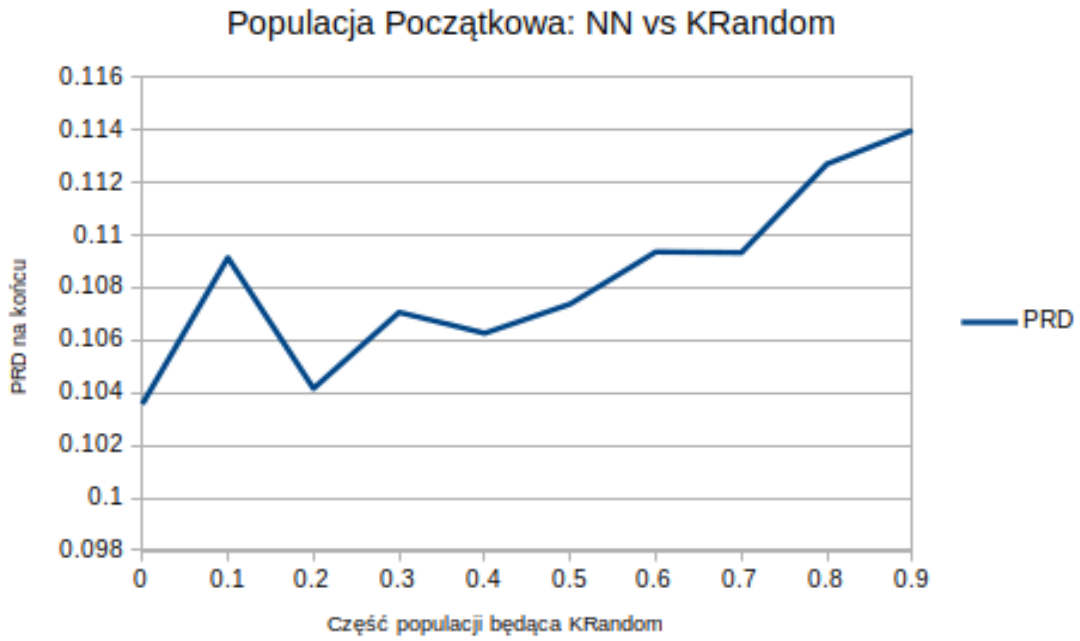
W teście będę badał współczynnik Wyspy*Generacje. Będzie on stały dla każdej ilości wysp.



Z testu widzimy słabość wyboru wielu wysp – osobnicy obiecujący zostają odizolowani i dodatkowo nie mają szansy rozwoju przez zmniejszoną liczbę generacji.

Test III – populacja początkowa

Implementacja zezwala na określenie podziału populacji początkowej (oraz populacji przy zdarzeniu wymierania). Generacja zostanie podzielona na dwie części – stworzoną za pomocą metody k-random, oraz za pomocą Nearest Neighbour z mutacjami. Utrzymanie losowych permutacji w populacji gwarantuje zwiększoną różnorodność i zapobiega stagnacji.



W prezentacji wyniku pominąłem pomiar dla 1.0, dla którego średnie PRD wynosiło 0.893. Pokazuje to, że nawet w przypadku gdy mała część populacji wykazuje optymalne zachowanie (np. dla stosunku 0.9) i tak otrzymamy rozwiązanie porównywalnie dobre (0.114 a 0.104). Zauważamy obniżenie średniego momentu do osiągnięcia stosunku 0.2. Co ciekawe, Wariant z samym NN, czyli stosunek 0, wykazuje najlepsze wyniki. Należy pamiętać że członkowie generowani NN zostają losowo zmutowani przy tworzeniu populacji.

Możemy wywnioskować, iż nawet mała kolebka obiecujących osobników koniec końców odniesie sukces i przetrwa, wyznaczając szlak kolejnym pokoleniom.

Test IV – szansa mutacji

Mutacja jest jednym z ważniejszych czynników gwarantującym różnorodność populacji jak i gwarantujących poprawę populacji. Należy jednak uważać przy doborze czynnika, zbyt wiele mutacji w populacji może doprowadzić do straty ważnego genotypu.

Jak można się było spodziewać, brak mutacji gwarantuje najsłabszy wynik z przedziału [0,0.1]. Minimum lokalne funkcja osiągnie w okolicach 5-6%. Należy zauważyć, iż dalszy przebieg funkcji sugerowałby że dla wartości powyżej 0.1 otrzymamy rezultaty gorsze od wartości 0.

Test (stat.) - Wymieranie

Zjawisko wymierania zdarza się naprawdę rzadko, i można by zastanowić się czy aby na pewno jest potrzebne. W teście stawiam następującą tezę: Wymieranie nie ma wpływu na wynik końcowy. W teście otrzymuję dwa zbiory danych: Jeden dla symulacji ze standardową szansą wymierania, drugi z zerową szansą wymierania. Wykonano 50 powtórzeń dla obu metod dla 5000 iteracji i 3 wysp na grafie berlin52.

Otrzymane wyniki:

w = 601.0
p = 0.725

Zatem nie możemy jednoznacznie stwierdzić poprawności tezy, wysoka ilość powtórzeń nie jest w stanie zrekompensować rozpiętości wyników, co jest głównym czynnikiem dużego p-value.

Podsumowanie

Z dotychczas badanych algorytmów, GA zezwala na zdecydowanie najwięcej dostrojeń dzięki bardzo dużej ilości parametrów. Z drugiej strony, dobranie idealnych parametrów może sprawiać duży kłopot. W metaheurystyce większy nacisk kładziemy na poprawę w dłuższym czasie, aniżeli na poprawę tymczasową, jak było w algorytmie TabuSearch, gdzie często zdarzało nam się zachłannie utknąć w pewnym dołku. Miarę niskim kosztem oferowana jest świetna różnorodność co można uznać jako mocną stronę algorytmu. Do minusów należy fakt, że dążymy do poprawy w spowolnionym tempie, kładąc nacisk na aspekt ewolucyjny – długotrwały, aniżeli rewolucyjny – chwilowy.