Algorytm Genetyczny – problem komiwojażera

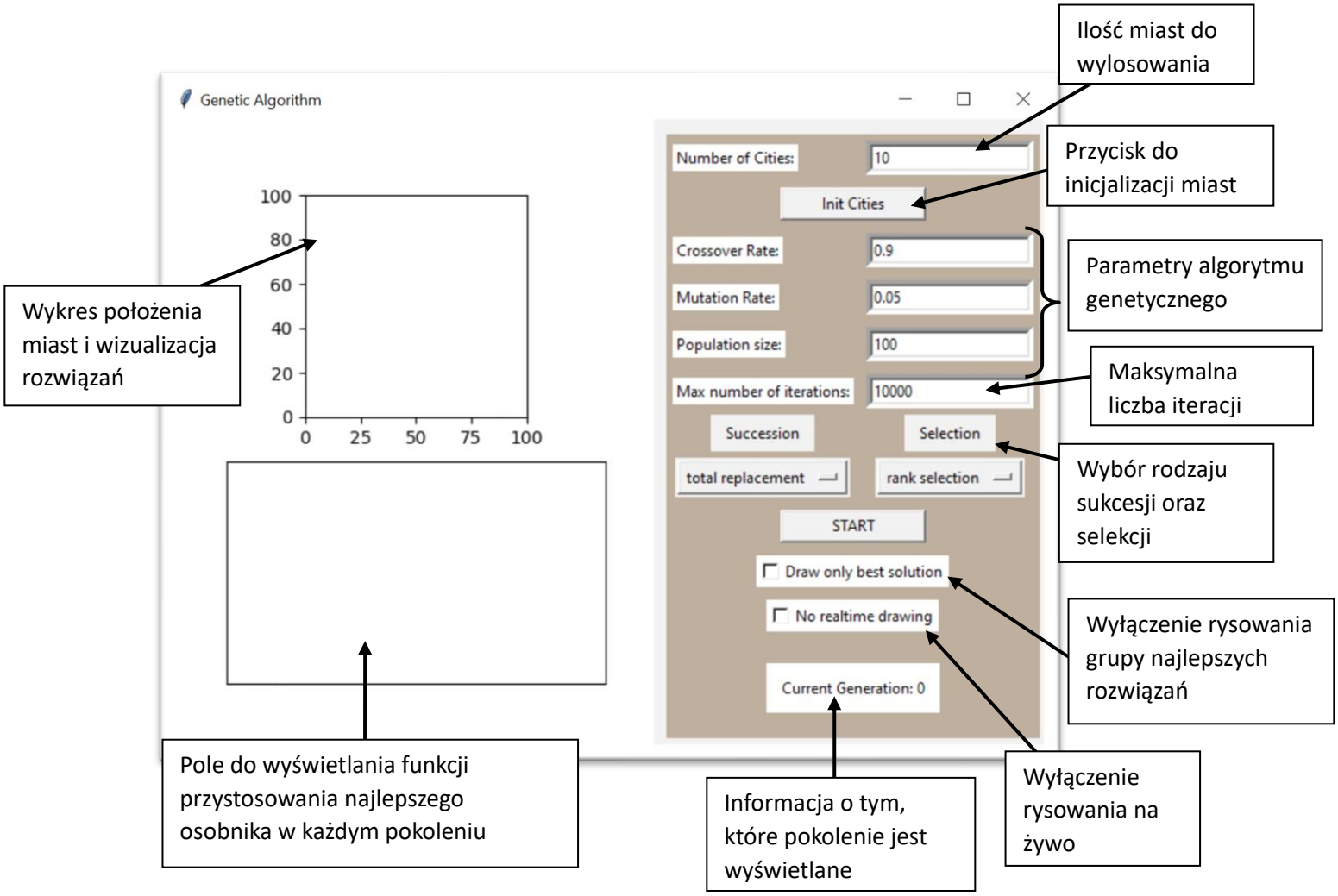
Natalia Kołodziejczyk, Inżynieria i Analiza Danych, nr albumu 409695

Wstęp

Gdy zastanawiałam się w jaki sposób chciałabym ten projekt zrealizować, pomyślałam, że byłoby dobrze w jakiś sposób oglądać działanie algorytmu oraz w łatwy sposób zmieniać jego parametry. Wpadłam więc na pomysł stworzenia prostej aplikacji okienkowej, która pozwalałaby na wizualizację działania algorytmu, śledzenie w jaki sposób zmienia się funkcja przystosowania, losowanie punktów na wykresie oraz intuicyjne ustawianie parametrów. Wiedziałam, że zadanie może nie będzie łatwe, ale ostatecznie skusiłam się na tą opcję. Przy pomocy pythonowej biblioteki tkinter stworzyłam prosty interfejs. Od razu chciałabym podkreślić, że sama aplikacja jest prototypem – nie jest może najpiękniejsza wizualnie, raczej prosta. Samo rysowanie dla dużej ilości miast nie jest zbyt wydajne. Oraz nie obsłużonych jest parę niestandardowych sytuacji, jak np. ponowna inicjalizacja miast podczas działania algorytmu. Ale chciałam się tą aplikacją podzielić w ramach tego projektu, ponieważ naprawdę sprawia to dużo radości. Algorytm genetyczny oczywiście jest kompletny, tylko dodatkowo opakowany w część wizualną.

Aplikacja okienkowa

Może nie jest to konieczne, ale przejdę do krótkiego opisu, jak aplikację najlepiej używać. Poniżej przedstawiony interfejs:



Myślę, że cały interfejs jest w miarę intuicyjny, ale zamieszczam drobny opis.

A teraz: jak efektywnie korzystać z aplikacji. Pierwszym krokiem po uruchomieniu musi być wylosowanie miast (bez tego algorytm oczywiście nie będzie mógł poprawnie wystartować). Wartość domyślna to 10 miast, możemy do okna wprowadzić inną liczbę i nacisnąć przycisk „Init Cities", w wyniku czego wylosujemy podaną ilość miast. Ułożenie wyświetli się na wykresie obok. Bez obaw możemy zmieniać ilość miast i losować je tak długo, aż uznamy ustawienie za odpowiednie.

Po wylosowaniu miast możemy przejść do ustawiania parametrów algorytmu. Starałam się sparametryzować jak najwięcej opcji mogłam. Podstawowymi parametrami są: prawdopodobieństwo krzyżowania, prawdopodobieństwo mutacji oraz rozmiar populacji. Do tego możemy wybrać dwie różne metody selekcji: metodę rankingową lub ruletki. Możemy też określić sposób selekcji: czy cała populacja zastępowana jest przez całe potomstwo, czy jednak jeden najlepszy rodzic zawsze przejdzie do populacji potomnej. Możemy też ustalić, po jakiej maksymalnie liczbie iteracji chcemy, by algorytm się zatrzymał. W praktyce jest to bardziej dla nas (w razie czego, jakby okazało się, że algorytm radzi sobie tragicznie to żeby nie chodził w kółko), bo w programie „na sztywno", po wielu eksperymentach określiłam po jakiej liczbie pokoleń bez zmiany algorytm ma się zatrzymać (ale głownie sprawdza się przy 10-30 miastach, wyżej może być wymagana niekiedy korekta).

Dwie pozostałe opcje są do wizualizacji. Możemy je zmieniać w dowolnym momencie działania algorytmu (ale pod żadnym pozorem nie można tego robić z parametrami algorytmu, one są

wczytane do niego na początku działania!). Opcja „Draw only best solution” oznacza, że rysowany będzie tylko jeden, najlepszy przedstawiciel każdego pokolenia. Domyślnie oprócz niego rysowane jest również 20 najlepszych osobników. Opcja może to być przydatna z dwóch powodów: przy dużej ilości miast, by nie zaciemniać ekranu, lub w celu przyspieszenia algorytmu. Opcja „No realtime drawing” sprawia, że nie widzimy na bieżąco co dzieje się w algorytmie. W przypadku, kiedy mamy 20 miast samo zaznaczenie opcji „Draw only best solution” sprawia, że algorytm działa w miarę sprawnie, ale już przy 40 miastach algorytm zaczyna chodzić bardzo powolnie. Spowodowane jest to tym, że z każdą iteracją odbywa się nowa wizualizacja. Wtedy można zaznaczyć tą opcję, dzięki czemu nie obciążamy dodatkowo komputera. Co przydatne – możemy regularnie sprawdzać na jakim etapie jest algorytm przez odznaczenie opcji w dowolnym momencie. Można pooglądać parę populacji i ponownie ją zaznaczyć.

Na koniec, bez względu na to czy wizualizowaliśmy proces, wyświetli się ostateczny wykres z najlepszym znalezionym rozwiązaniem, oraz wykres ilustrujący jak zmieniała się funkcja przystosowania najlepszego osobnika.

Sam algorytm

Algorytm zaimplementowałam przy pomocy dwóch klas: Individual, która reprezentuje konkretnego osobnika. Każdy z osobników ma atrybuty: długość chromosomu, chromosom jako np.array oraz wartość funkcji przystosowania. Funkcja przystosowania liczona jest na podstawie dostarczonej osobnikowi tablicy dystansów; chciałam zrobić coś analogicznego do tego co było pokazane na wykładzie. Oprócz tego, osobnik może się skrzyżować z innym, owocem czego jest dwóch potomków. Krzyżowanie odbywa się poprzez wylosowanie locusu początku oraz końca, wymianie tych dwóch fragmentów chromosomu, oraz wyeliminowaniu konfliktów, jakie w wyniku tego wystąpiły. Natomiast mutacja osobnika odbywa się poprzez przesunięcie dowolnego allelu na wybrane miejsce do przodu.

Cała logika algorytmu genetycznego zawarta jest w klasie GeneticAlgorithm. W takiej pojedynczej instancji zawarte są wszystkie parametry algorytmu oraz inne potrzebne dane. Tutaj przechowywane są między innymi: cała populacja, wybrani rodzice, ich dzieci, tablica dystansów, najlepszy osobnik z każdego pokolenia. Właściwa pętla jest nieco zacieniona przez funkcje rysujące wykresy, ale wygląda następująco:

```
alg = GeneticAlgorithm(self.cities, self.num, self.crossover_rate, self.mutation_rate,
                       self.population_size,self.max_iter,succesion,selection)
# Populacja początkowa
alg.initialize_population()
alg.count_distance_table()
# Ocena wszystkich osobników populacji
alg.mark_the_population()
```

Pierwsze następuje inicjalizacja populacji początkowej. Tworzona jest podana liczba osobników o losowych chromosomach. Jednokrotnie obliczamy wspomnianą wcześniej tablicę odległości, która jest potrzebna do obliczenia funkcji przystosowania. Następnie populacja jest oceniana. Wchodzimy do właściwej pętli:

```
while True:
    print(alg.current_generation)
    alg.current_generation += 1
    self.label_generation.config(text=f"Current Generation: {alg.current_generation}")
    alg.selection()
    alg.crossover()
    alg.mutation()
    alg.succession()
    alg.mark_the_population()
    if alg.best_solutions[alg.current_generation-1] == alg.best_solutions[alg.current_generation]:
        pop_without_improvement += 1
    else:
        pop_without_improvement = 0
    if pop_without_improvement == max_pop_without_improvement:
        break
    if alg.max_iterations <= alg.current_generation:
        break
```

Z naszej populacji dokonujemy selekcji rodziców. W zależności od wybranej przez nas opcji wywoływana jest odpowiednia metoda. Wybrani rodzice się krzyżują. Ponieważ już sama selekcja stanowiła losowanie i kolejność wyboru jest przypadkowa, to jako pary rodziców wybieram po prostu kolejne osobniki z grupy rodzicielskiej. Dla każdej pary rodziców istnieje określone prawdopodobieństwo tego, że wydadzą oni na świat potomstwo. Jeśli jednak los nie chce by tak się stało, do populacji dzieci dodawanych jest dwóch osobników identycznych z rodzicami – nie dochodzi do krzyżowania.

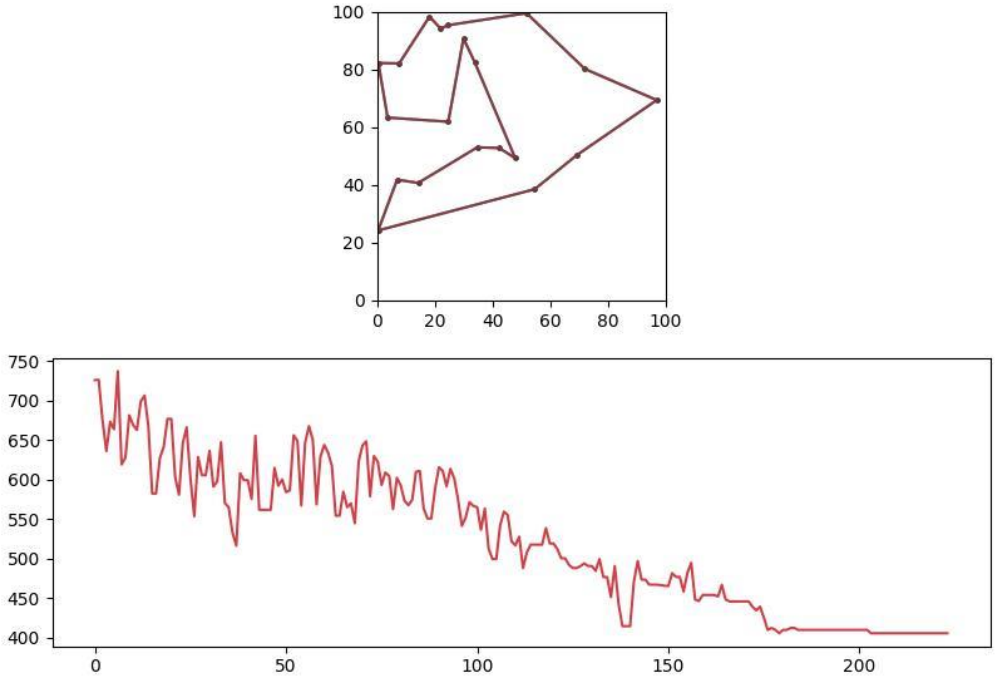
Następuje mutacja samej grupy potomnej. Następnie odbywa się sukcesja: może trochę sztucznie to oddzieliłam, ale było to konieczne by móc zastosować zarówno sukcesję elitarną jak i z całkowitym zastępowaniem. Nowa populacja jest oceniana. Następnie sprawdzany jest warunek stopu (oraz, czy nie przekroczona została maksymalna ilość iteracji). Jeśli warunek jest spełniony, algorytm się kończy i dostajemy ostateczny wynik. W przeciwnym wypadku cała pętla się powtarza.

Analiza wyników

Zacznijmy naszą analizę dla mniejszej ilości miast i zobaczmy, jakie kombinacje parametrów wydają się obiecujące. Zobaczmy, jak działa algorytm dla 20 miast.

20 miast – porównanie rodzajów selekcji

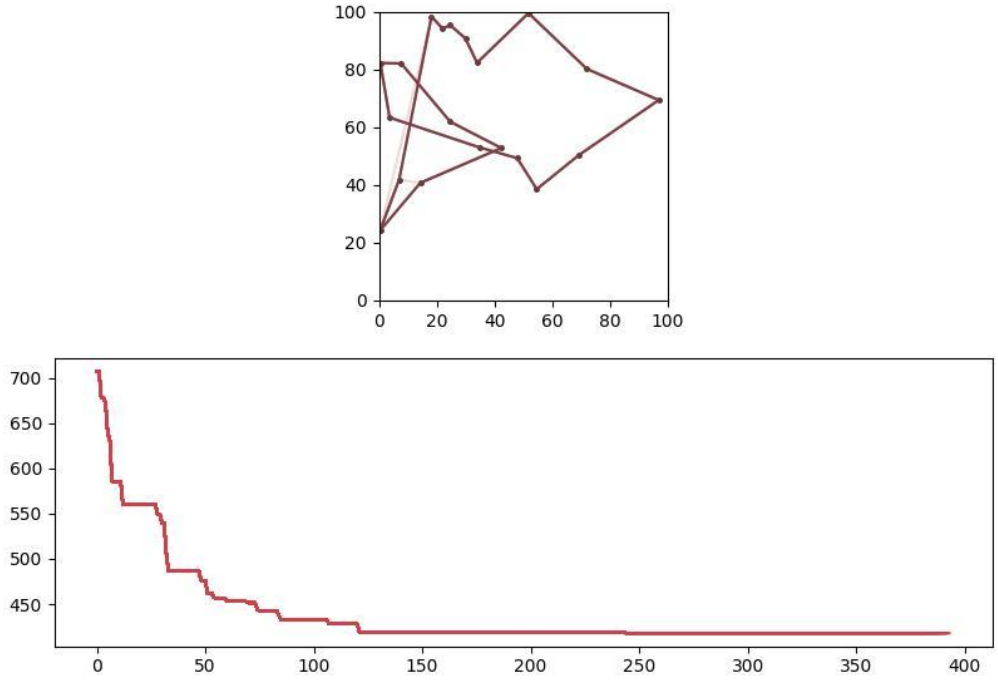
Zacznijmy od selekcji rankingowej z całkowitym zastępowaniem, przy użyciu domyślnych parametrów:



Ilość generacji: 223, wartość funkcji przystosowania: 405.52286686.

Po uruchomieniu algorytmu parokrotnie mogę stwierdzić, że: algorytm kończy się przy około 200-400 pokoleniu a funkcja przystosowania wynosi: 330 – 405. Warunek stopu to 20 pokoleń bez zmiany.

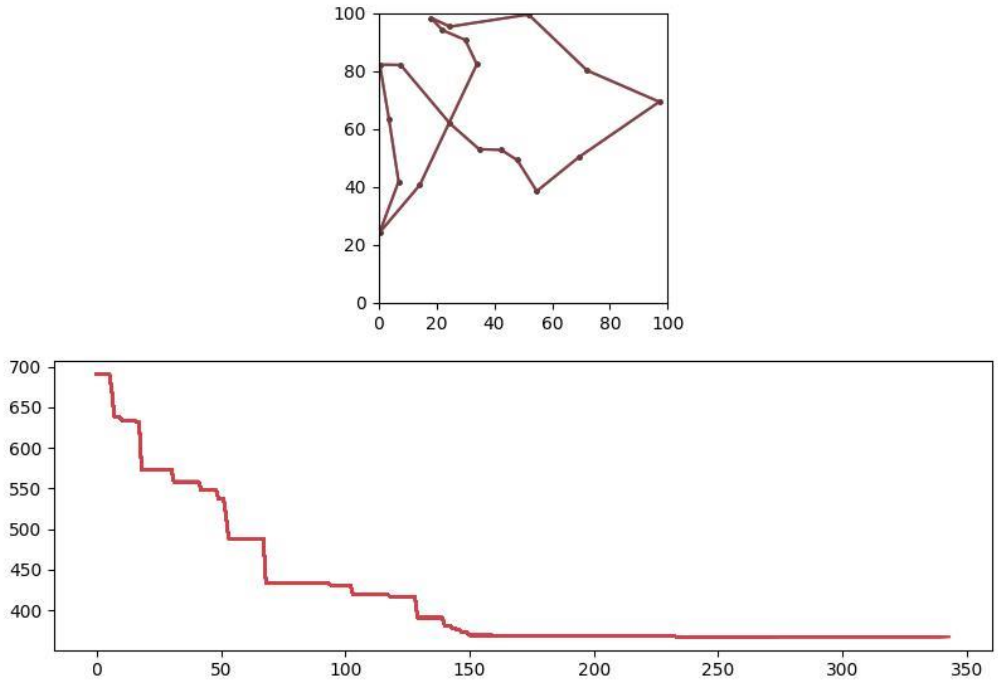
Zobaczmy, jak wygląda przebieg algorytmu gdy wybierzemy sukcesję elitarną.



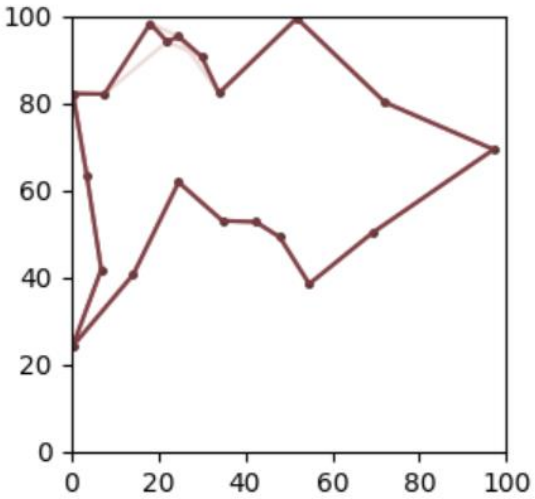
Dla tego typu dałam bardziej restrykcyjny warunek stopu, ponieważ rotacja najlepszego osobnika nie jest tak duża jak w przypadku całkowitego zastępowania. Dla tego typu wynosi on 150 pokoleń bez zmian. Jak widzimy tak duża liczba jest dobra, ponieważ przy mniejszym warunku nie doszłoby do poprawy przy około 240 pokoleniu, jak widzimy na wykresie, tylko algorytm mógłby już się zatrzymać. Liczba pokoleń do przejścia będzie więc w tym wypadku dużo większa. Natomiast widzimy, że dojście do już całkiem przyzwoitego wyniku zajęło tylko 125 pokoleń – co jest dużo szybsze w porównaniu do całkowitego zastępowania, gdzie wynik stabilizuje się przy 200.

Uzyskany wynik to: 417.80792318 przy 394 pokoleniach. A ogólne zakresy po paru eksperymentach: 300-650 pokoleń a funkcja przystosowania 335 – 417 (co jest bardzo podobnym zakresem do wcześniejszego)

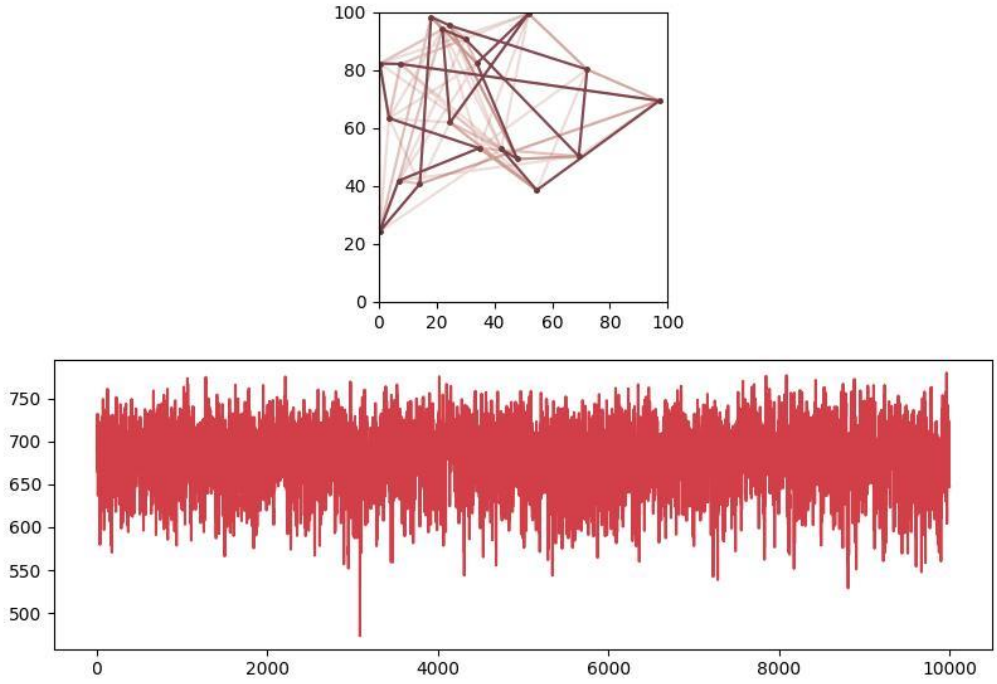
Teraz sukcesja elitarna wraz z selekcją ruletki:



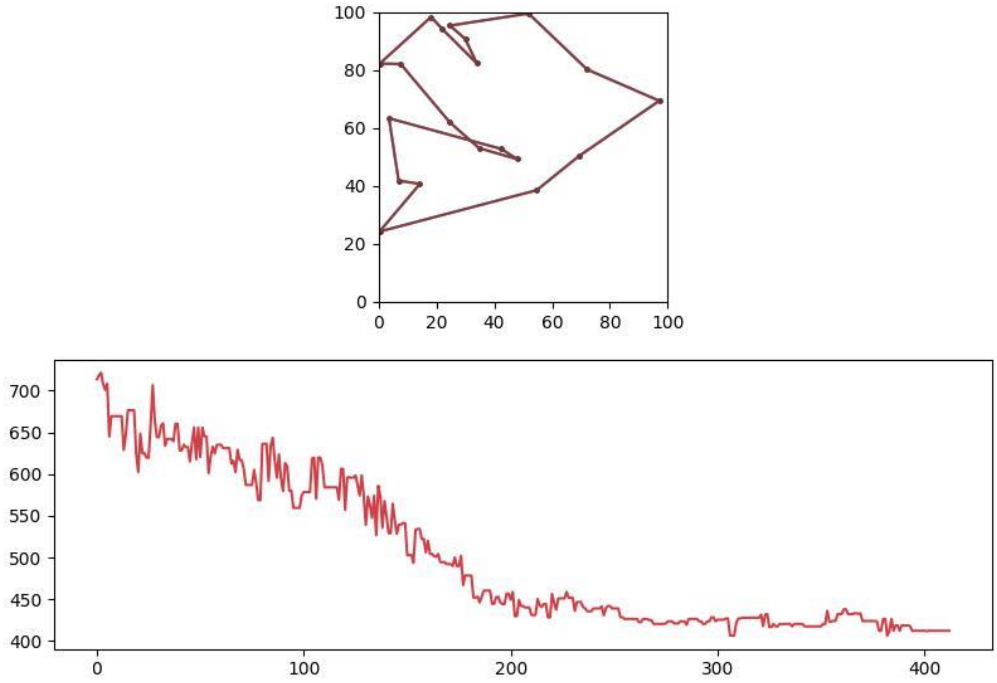
Warunek stopu dałam tu trochę mniejszy: 100 pokoleń bez zmiany. Uzyskany wynik to 367.1821356 przy 344 pokoleniach. Tutaj również stabilizacja nastąpiła przy około 150 pokoleniu, ale ten typ selekcji przy sukcesji elitarnej wydaje się być trochę szybszy. Po wykonaniu paru eksperymentów mamy: liczba pokoleń 317-700, funkcja przystosowania: 329-447. Rozrzut wydaje się być znaczny dla tej metody, ale za to znalezione zostało najlepsze jak do tej pory rozwiązanie:



Ostatni typ to sukcesja z całkowitym zastępowaniem i metoda ruletki. Nie wiem dlaczego, ale z jakiegoś powodu ten rodzaj działa tragicznie. O ile z sukcesją elitarną metoda ruletki ewidentnie radzi sobie dobrze, tak z całkowitym zastępowaniem algorytm w ogóle nie dochodzi do stabilizacji. Dla selekcji rankingowej populacja stabilizowała się po 200 pokoleniu. A tutaj do tego nie dochodzi – mimo ustawienia warunku stopu na tylko 10 pokoleń bez zmiany. Wydaje się, że nie dochodzi do żadnej poprawy nawet przy 2000 pokoleniu. Algorytm jakby działał całkowicie losowo, nie widać żadnej tendencji spadkowej… Musiałam długo się naczekać, by dojść do maksymalnego 10000 pokolenia (po to też dodałam opcję jego ustawienia, bo w tym przypadku wyniku można się nie doczekać).



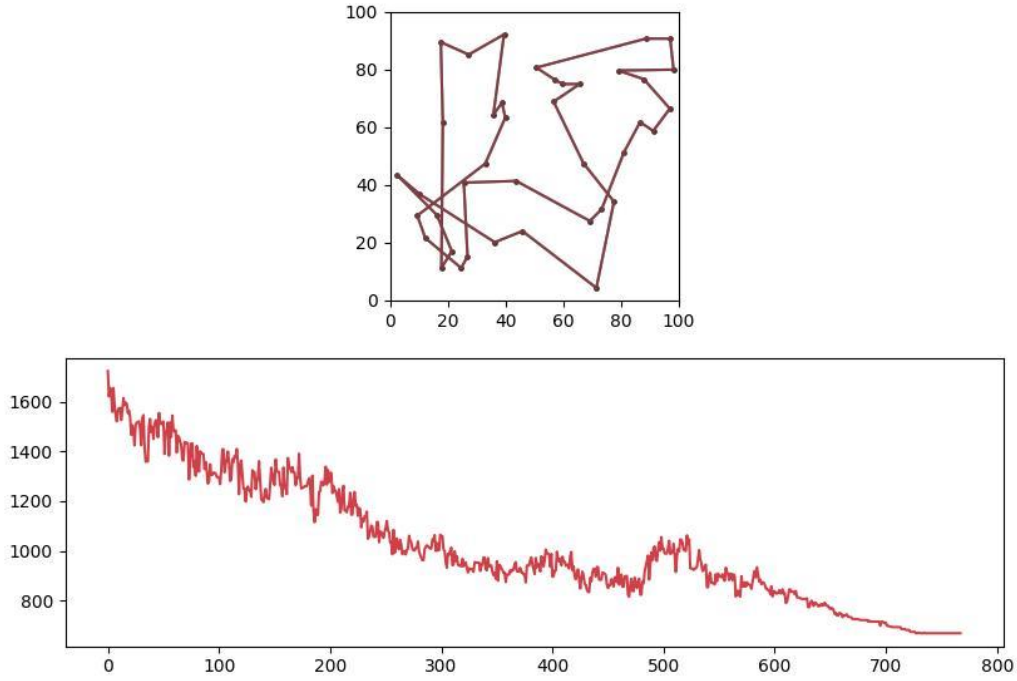
Postanowiłam więc trochę pokombinować z innymi parametrami algorytmu. Do poprawy doszło po drastycznym zmniejszeniu prawdopodobieństwa krzyżowania do 40%, zmniejszenia prawdopodobieństwa mutacji do 3% oraz zwiększeniu rozmiaru populacji do 150.



Wynik: 412 a liczba pokoleń nieco ponad 400. Jednak ewidentnie ten typ selekcji nie jest w przypadku naszego problemu stabilny i szybki. Rozwiązania znalezione w ten sposób nie były w stanie przejść poniżej 400 (co innym rozwiązaniom udawało się bardzo często). Stanowczo lepsze, szybsze i stabilniejsze wydają mi się powyższe typy.

40 miast

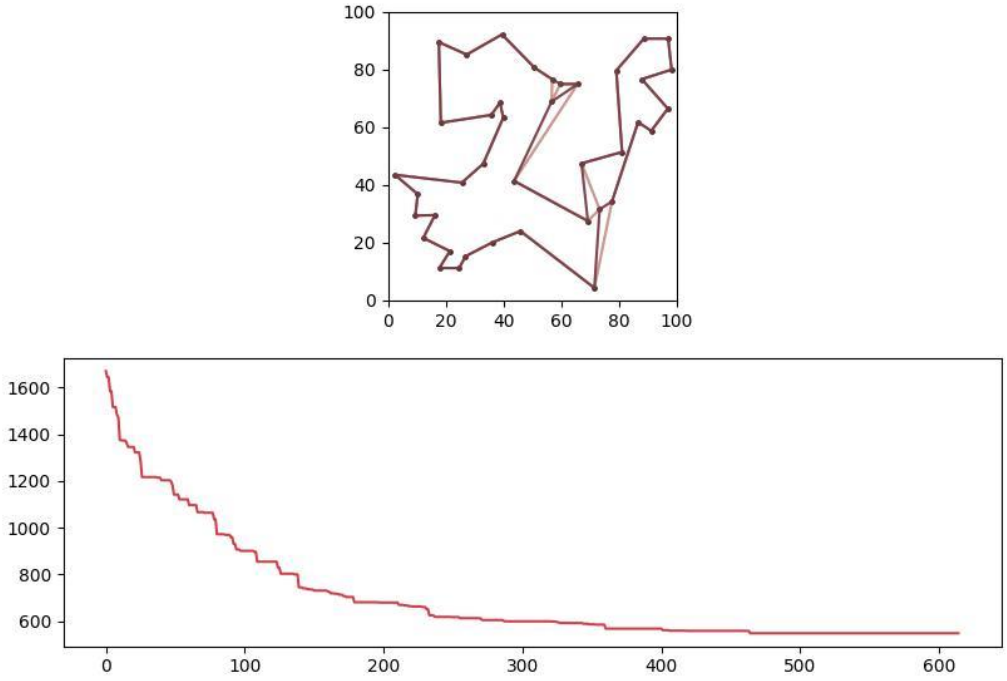
Na pierwszy rzut algorytm genetyczny z całkowitym zastępowaniem, stosujący selekcję rankingową.



Rozwiązanie: 668.22827288, ilość pokoleń: 767.

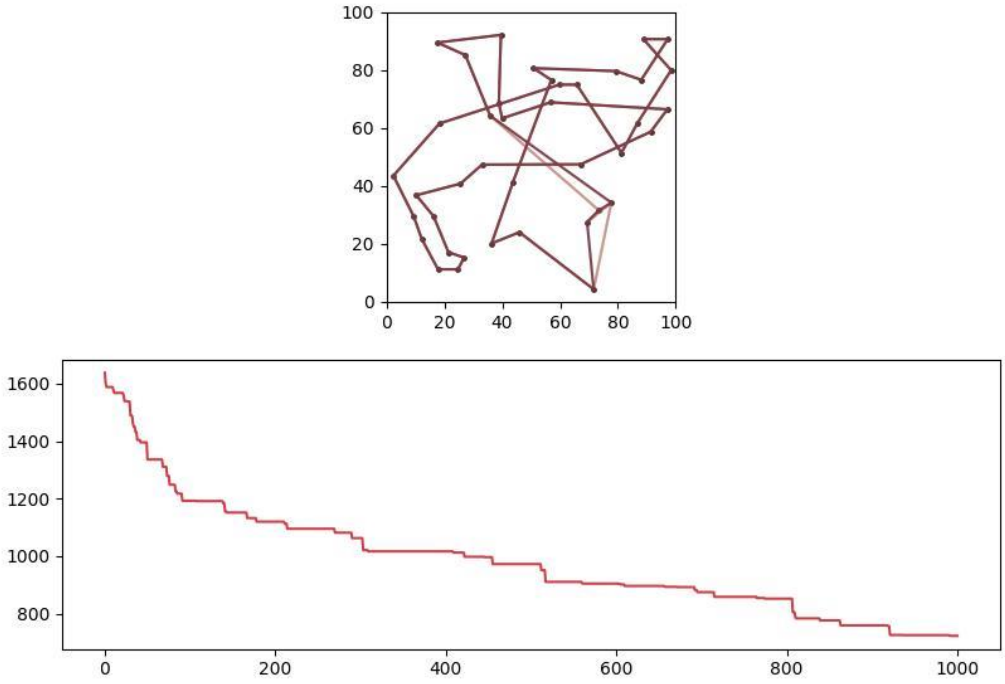
Jak na dwukrotne zwiększenie liczby miast, szybkość uzyskania wyniku bardzo przyzwoita! Jednak czy jest on najlepszy? Wydawałoby się, że dałoby się go trochę zoptymalizować… Co jednak jest niewątpliwie dobre to to, że algorytm potrafi znaleźć dobre rozwiązanie poniżej 1000 pokoleń (sprawdziłam parokrotnie).

Zobaczymy, czy przy sukcesji elitarnej algorytm działa lepiej.



Mamy dużo lepsze rozwiązanie 548.11920518 znalezione po zaledwie 614! Algorytm z sukcesją elitarną zadziałał bardzo dobrze po zwiększeniu prawdopodobieństwa mutacji do 10%. Pozytywnie na działanie algorytmu wpłynęło też zmniejszenie prawdopodobieństwa krzyżowania do 80%. Bardzo szybko dochodzi do dobrego rozwiązania, już przy 300 pokoleniu a potem dokonuje poprawek. Zdarza się, że działa czasem dużo dłużej ale zazwyczaj dochodzi do bardzo dobrego rozwiązania. Wydaje się, że sukcesja elitarna działa bardzo dobrze z większymi wartościami prawdopodobieństwa mutacji, a z mniejszymi prawdopodobieństwa krzyżowania.

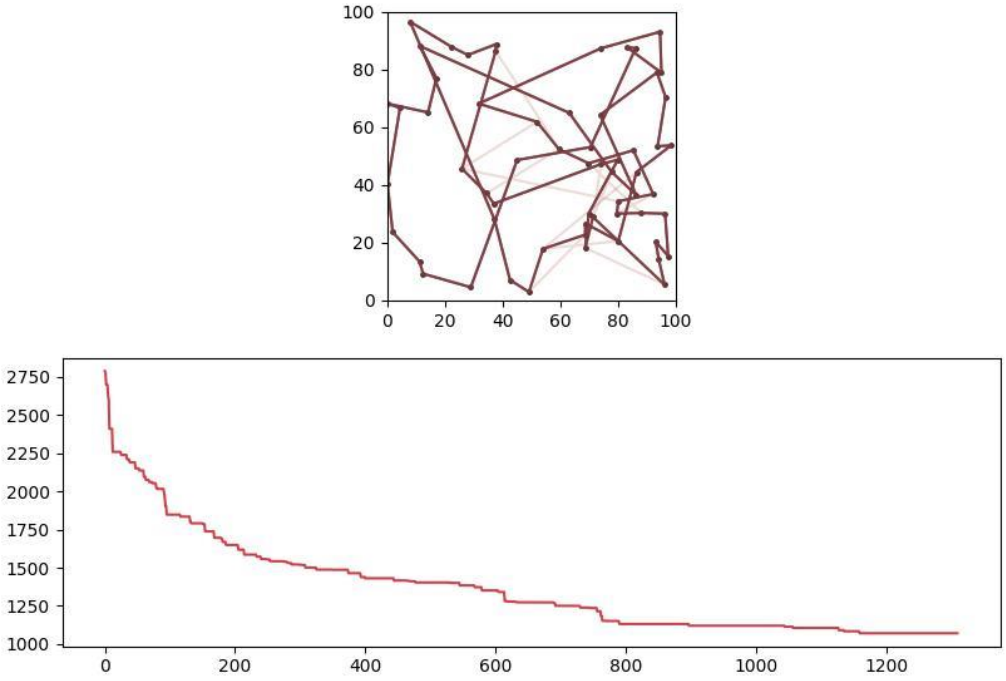
Dla sukcesji elitarnej z metodą ruletki mamy natomiast:



No tutaj nie mamy za bardzo wybitnych wyników. Nieznacznie pomaga zwiększenie rozmiaru populacji do 120, ale wtedy całe działanie algorytmu jest sporo spowolnione. Mamy wynik 722.78395105 w 1000 pokoleń (ustawiłam limit). Przy sukcesji elitarnej dużo lepsze wyniki uzyskała metoda selekcji rankingowej

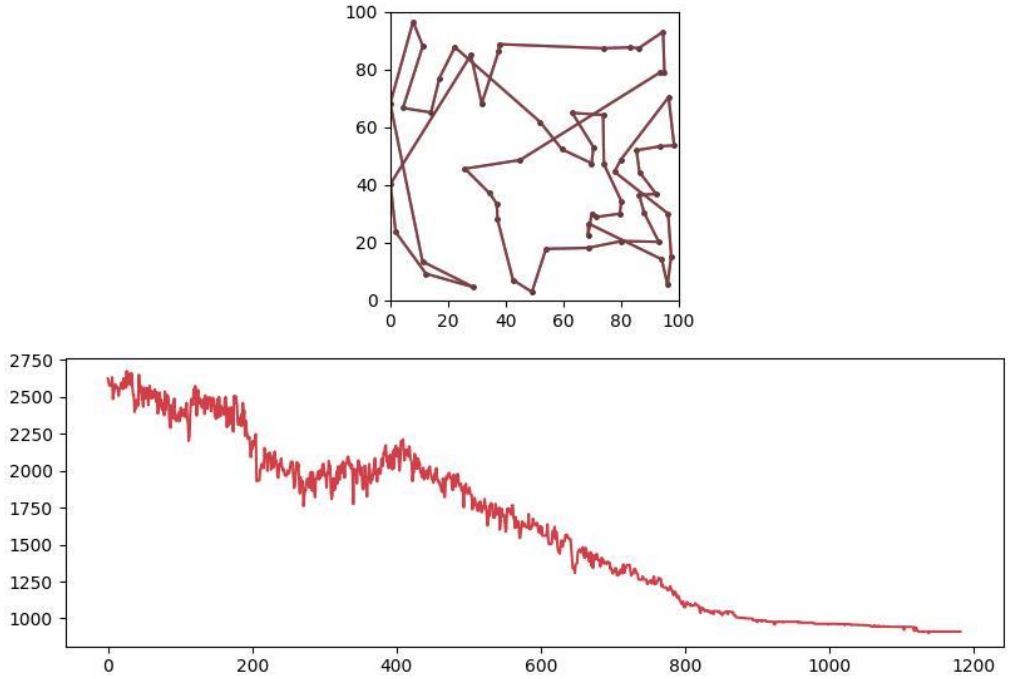
60 miast

Postanowiłam również przetestować działanie algorytmu w skrajnych warunkach – dla 60 miast. Tu już jakiejś poważniejszej analizy nie wykonałam, ponieważ czas trwania takiego jednego algorytmu jest już bardzo spory. Tak więc to bardziej jako ciekawostka, żeby zobaczyć co wyjdzie. Chciałam tylko zobaczyć, czy algorytm poradzi sobie z taką ilością.



Mamy sukcesję elitarną z metodą rankingową, prawdopodobieństwo krzyżowania 80% a mutacji 30%.
Rozwiązanie: 1071.12077123

Metoda rankingowa przy całkowitym zastępowaniu dała natomiast wynik:



Funkcja przystosowania: 910.85788246 przy 1181 pokoleniach. Dla parametrów 95% prawdopodobieństwo krzyżowania i 3% prawdopodobieństwo mutacji, a do tego rozmiar populacji zwiększony do 110. Tu już widać, że rozwiązanie wygląda w miarę przyzwoicie.

Wnioski

Po przeanalizowaniu różnych typów algorytmu genetycznego oraz ich parametrów, stwierdzam, że dla problemu komiwojażera najlepiej spisuje się selekcja rankingowa z całkowitym zastępowaniem lub selekcja rankingowa z sukcesją elitarną. Wydaje się, że selekcja ruletkowa nie spisuje się najlepiej. Ponad to można zauważyć, że przy całkowitym zastępowaniu korzystna jest duża wartość prawdopodobieństwa krzyżowania (około 90%) i nie za duża dla mutacji (3-5%). Mimo tego, że często zdarza się że w następnym pokoleniu wyginie najlepsze rozwiązanie, to ostatecznie sytuacja stabilizuje się po pewnej liczbie pokoleń i do dyspozycji mamy dobre rozwiązania – w wyniku ich krzyżowania ostatecznie dochodzimy do wyniku.

Dla sukcesji elitarnej, gdzie jeden najlepszy osobnik zawsze przechodzi do następnego pokolenia, spisuje się lepiej większa wartość prawdopodobieństwa mutacji (nawet 10-20%) a mniejsza krzyżowania. Skoro i tak najlepszy osobnik przechodzi dalej bez zmiany, to możemy sobie pozwolić na dużo częstsze mutacje potomstwa, co może szybko doprowadzić do rozwiązania. Dzięki mniejszemu prawdopodobieństwu krzyżowania część osobników przechodzi dalej bez zmiany chromosomu, ale za to jest podatna na mutację – dobre rozwiązania może udać się jeszcze bardziej polepszyć.
910.85788246