Zastosowanie programowania ewolucyjnego w celu rozwiązania problemu komiwojażera

Maciej Wieczór-Retman, Kacper Stojek

1 Wstęp

Problem komiwojażera (ang. *Traveling salesman problem*) jest problemem minimalizacji, polegającym na odnalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Cykl Hamiltona to taki cykl w grafie, w którym każdy wierzchołek grafu jest odwiedzony dokładnie raz. Czyli rozwiązaniem problemu komiwojażera jest cykl o najmniejszej sumie wag krawędzi grafu. Liczba różnych cyklów Hamiltona H dla pełnego grafu nieskierowanego o n wierzchołkach jest wyrażana równaniem eq. (1).

$$H = \frac{(n-1)!}{2} \tag{1}$$

Z uwagi na liczbę różnych cyklów, przeszukiwanie wszystkich rozwiązań jest fizycznie niemożliwe dla większych grafów. Przykładowo stosunkowo mały graf o stu wierzchołkach, daje w przybliżeniu $4,7 \times 10^{155}$ rozwiązań. Dla przybliżenia zakładając, że sprawdzenie jednej ścieżki zajmuje $1\,\mu s$, przeanalizowanie wszystkich ścieżek zajęłoby $1,48 \times 10^{142}\,lat$. Z tego powodu ważne jest zastosowanie bardziej skomplikowanych algorytmów minimalizacyjnych, takich jak właśnie programowanie genetyczne. Dobrze skonstruowany algorytm powinien być w stanie znaleźć minimum globalne dla większości przypadków lub przynajmniej optymalne rozwiązanie.

Napisany algorytm będzie testowany na danych z www.math.uwaterloo.ca w celu weryfikacji jego działania.

2 Implementacja

Jak zostało wspomniane na wstępie, rozwiązaniem problemu komiwojażera jest najkrótszy cykl Hamiltona dla danego pełnego, ważonego i nieskierowanego grafu. Z tych informacji wynikają szczegóły implementacji takie jak funkcja przystosowania, reprezentacja osobników oraz ograniczenia co do sposobu mutacji, oraz krzyżowania osobników.

Ponieważ szukany jest cykl Hamiltona, w rozwiązaniu każdy wierzchołek grafu pojawi się tylko raz, to znaczy, że mając graf G o N wierzchołkach:

$$V(G) = \{v_1\, v_2\, \cdots v_N\}, \tag{2}$$

rozwiązaniem będzie pewna permutacja zbioru $V(G)$. Z uwagi na to, najłatwiej jest wykorzystać kodowanie całkowitoliczbowe. Z tym kodowaniem każdy wierzchołek dostaje swój indeks,

a następnie każdy osobnik składa się z pewnej permutacji wierzchołków grafu, reprezentując pojedynczy cykl Hamiltona.

Mając zdefiniowany sposób zapisu osobników, można w oparciu o to zdefiniować funkcję przystosowania $f(x)$ dla osobnika x. Najpierw jednak należy zdefiniować funkcję kosztu $u(x)$. Ponieważ pojedynczy osobnik składa się z kolejnych indeksów wierzchołków, funkcją kosztu jest suma wag krawędzi pomiędzy kolejnymi wierzchołkami, oraz dodatkowo pomiędzy pierwszym a ostatnim wierzchołkiem w osobniku (szukany jest cykl grafu). Następnie funkcja przystosowania $f(x)$ jest następująca:

$$f(x) = C_{max} - u(x), \tag{3}$$

gdzie:

$f(x)$ - funkcja przystosowania,
$u(x)$ - funkcja kosztu,
C_{max} - parametr nie mniejszy niż maksymalna wartość $u(x)$ dla danej aktualnej populacji,
x - osobnik.

Z uwagi na naturę rozwiązania, metody krzyżowania osobników są ograniczone, ponieważ allele w nowopowstałym osobniku nie mogą się powtarzać. Dlatego przykładowo krzyżowanie arytmetycznie się nie nadaje. Możliwe sposoby krzyżowania to:

- PMX,
- OX,
- CX.

W projekcie użyto krzyżowania OX, ponieważ ta metoda krzyżowania najbardziej zachowuje kolejność alleli w osobniku. Zamysł jest taki, aby osobniki wymieniały się krótkimi trasami, zachowując ich kolejność. Dany osobnik na początku działania algorytmu będzie miał zoptymalizowany tylko fragment trasy. Krzyżując dwa osobniki, jest spora szansa, że ten fragment się zachowa i trafi do kolejnego osobnika. Za mieszanie kolejności raczej powinny być odpowiedzialne mutacje osobników. Metody krzyżowania PMX oraz OX za bardzo zmieniają szyk alleli, przez co nie zostały wykorzystane.

Podobnie jak w przypadku krzyżowania, osobniki po mutacji dalej muszą tworzyć poprawny ciąg Hamiltona. Z tego powodu mutacje ograniczają się do podmian elementów w strukturze osobnika, bez zmian ich wartości. Dopuszczalne sposoby mutacji to:

- inwersja (ang. inversion),
- wstawianie (ang. insertion),
- przenoszenie (ang. displacement),
- wzajemna wymiana (ang. reciprocal exchange).

W projekcie użyto inwersji oraz wstawiania. Przy mutacji, szczególnie w początkowej fazie działania algorytmu, ważne jest, aby mutacje miały charakter globalny. Inwersja polega na wylosowaniu

podciągu w osobniku i odwrócenie w nim kolejności alleli, natomiast wstawianie polega na wylosowaniu pojedynczego allela, i wstawienie go w innym, losowo wybranym miejscu. Wstawianie pozwala na stworzenie nowych tras w osobnikach, tak zmiany są małe i algorytm przy tylko takim rodzaju mutacji, szybko się zbiega do minimum lokalnego. Dlatego, aby troch zwiększyć szansę na znalezienie innego, lepszego minimum, została dodana inwersja. Ponieważ zmiany przy inwersji są bardziej drastyczne, jest większa szansa, że populacja nie utknie od razu w minimum lokalnym. To czy zostanie wykonana inwersja czy wstawianie też zależy od obecnej iteracji. Im wyższa iteracja, tym większa szansa, że zostanie wykonane wstawianie zamiast inwersji.

Przy tworzeniu nowych osobników również trzeba uważać, aby nie tworzyć elity. Zostawianie i krzyżowanie tylko najlepszych osobników, prowadzi do przedwczesnego zbiegania się algorytmu do nieoptymalnego rozwiązania. Dlatego osobniki są niszowane w ten sposób, że osobniki z gorszym przystosowaniu mają większą szansę na trafienie do puli rodzicielskiej. W odwrotnej konfiguracji, gdzie lepsze osobniki miały większą szansę na trafienie do puli rodzicielskiej, algorytm przedwcześnie się zbiegał do suboptymlanego rozwiązania. Następnie po przeprowadzeniu krzyżowania i wyznaczeniu nowych osobników, są one poddawane mutacji. Po krzyżowaniu oraz mutacji są pozostawiani rodzice lub potomkowie, w zależności od tego, który osobnik jest lepszy.

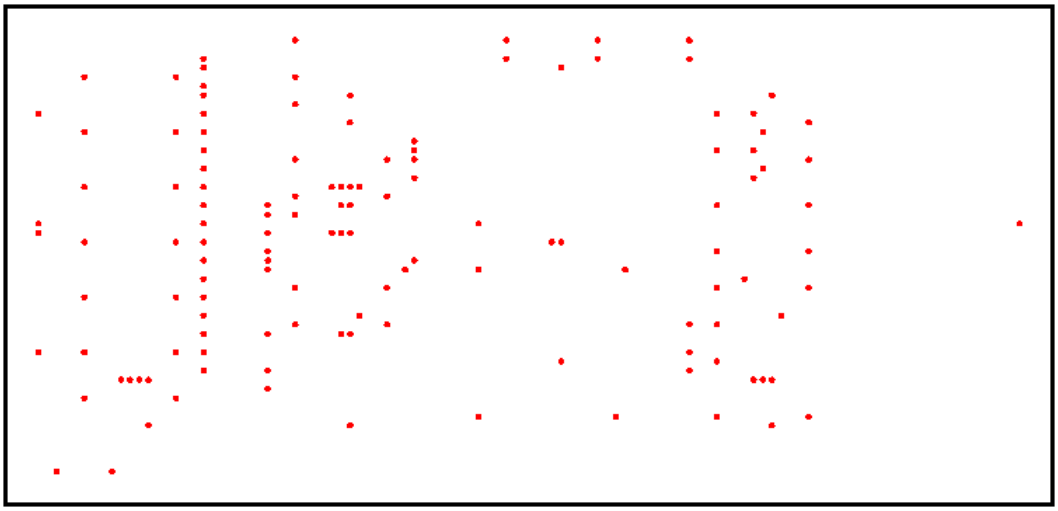
3 Wyniki pracy

Jak wspomniano na wstępie, algorytm został przetestowany na danych z www.math.uwaterloo.ca w celu weryfikacji jego działania. Pierwszym testowym grafem jest xqf131, którego wierzchołki są przedstawione na rysunku 1. Natomiast drugim grafem testowym jest xqg237, którego wierzchołki są przedstawione na rysunku 2.

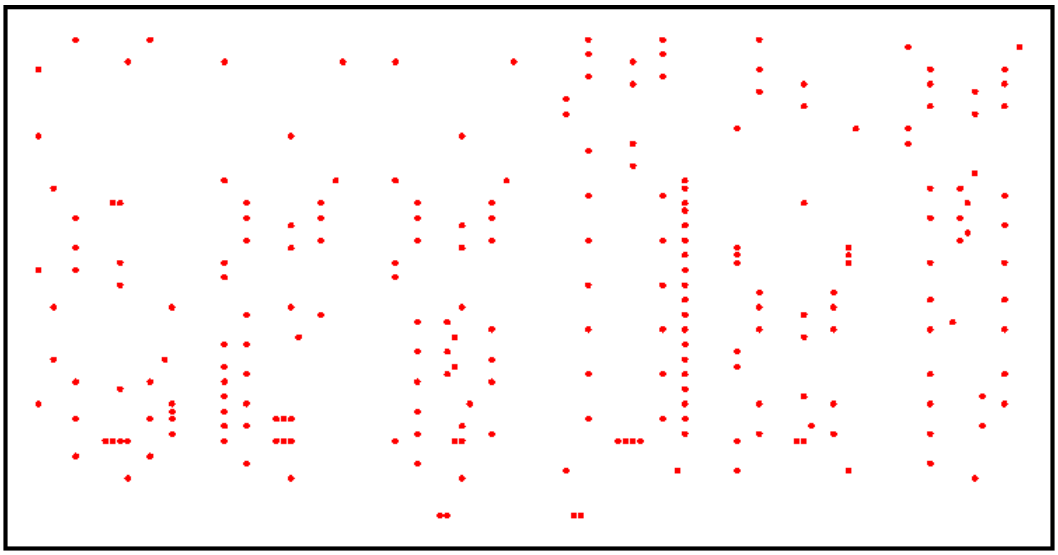
Początkowo algorytm do mutacji osobników wykorzystywał tylko wstawianie, bez inwersji oraz dla stosunkowo małej liczby osobników w populacji, wyniki przedstawiono na rysunku 3. Jak widać, algorytm bardzo szybko zbiega się do jakiegoś rozwiązania. Również widać, że zwiększenie liczby osobników pozwala na odnalezienie lepszego rozwiązania, dzięki rozważaniu większej ilości potencjalnych ścieżek. Następnie ten sam algorytm został sprawdzony na większym grafie xqg237. Wyniki są przedstawione na rysunku Jest ta sama sytuacja jak w przypadku mniejszego grafu, rozwiązanie szybko się zbiega, a większa liczba osobników pomaga znaleźć lepsze rozwiązanie. Znalezione rozwiązania przedstawiono na rysunku 5

W kolejnej wersji programu, do mutacji została dodana inwersja, z myślą o tym, aby ograniczyć wpadanie do minimum lokalnego. Wpływ tej modyfikacji jest przedstawiony na rysunku 7. Jak można zauważyć, dodanie inwersji faktycznie spowalnia zbieganie się algorytmu, jednak ostatecznie rozwiązanie jest podobne.

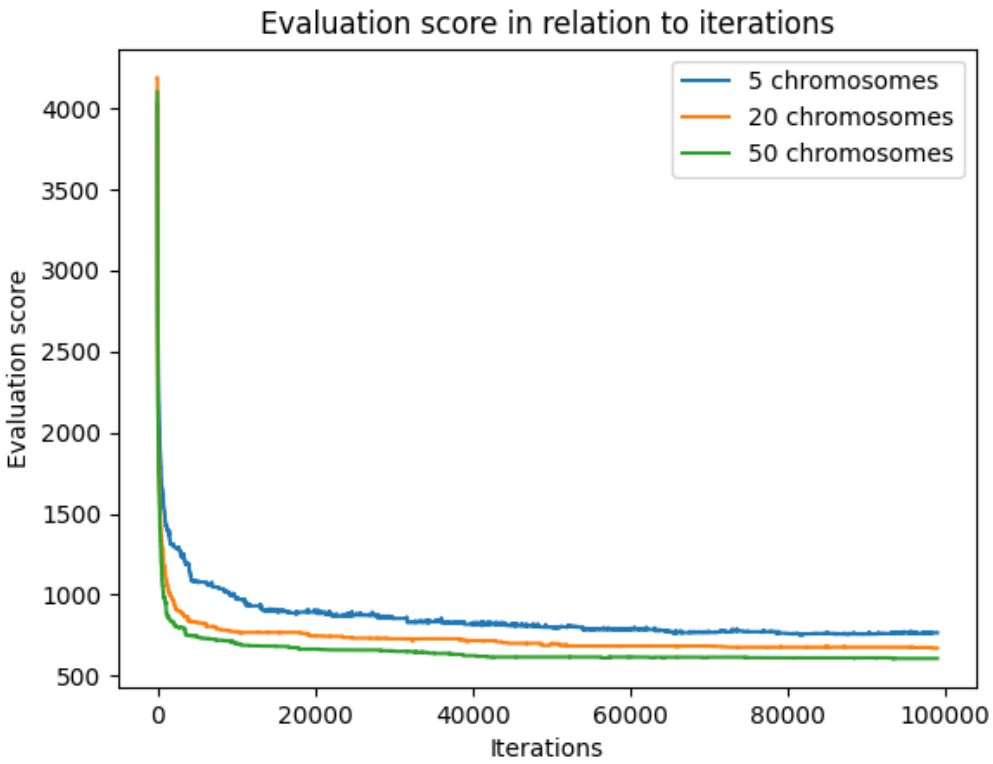
Poza inwersją zostały użyte większe populacje. Przekłada się to na dłuższy czas szukania rozwiązania, ale w zamian jest odszukane lepsze rozwiązanie. Na rysunku 8 jest przedstawiony przebieg algorytmu po modyfikacji dla grafu xqf131. Algorytm dość szybko zbiega się do rozwiązania, pomimo dodania nowej mutacji, ale jest to spowodowane większą ilością osobników. Samo rozwiązanie jest też lepsze w porównaniu do poprzedniej wersji programu, bez inwersji. Na



Rysunek 1: Wierzchołki grafu xqf131



Rysunek 2: Wierzchołki grafu xqg237

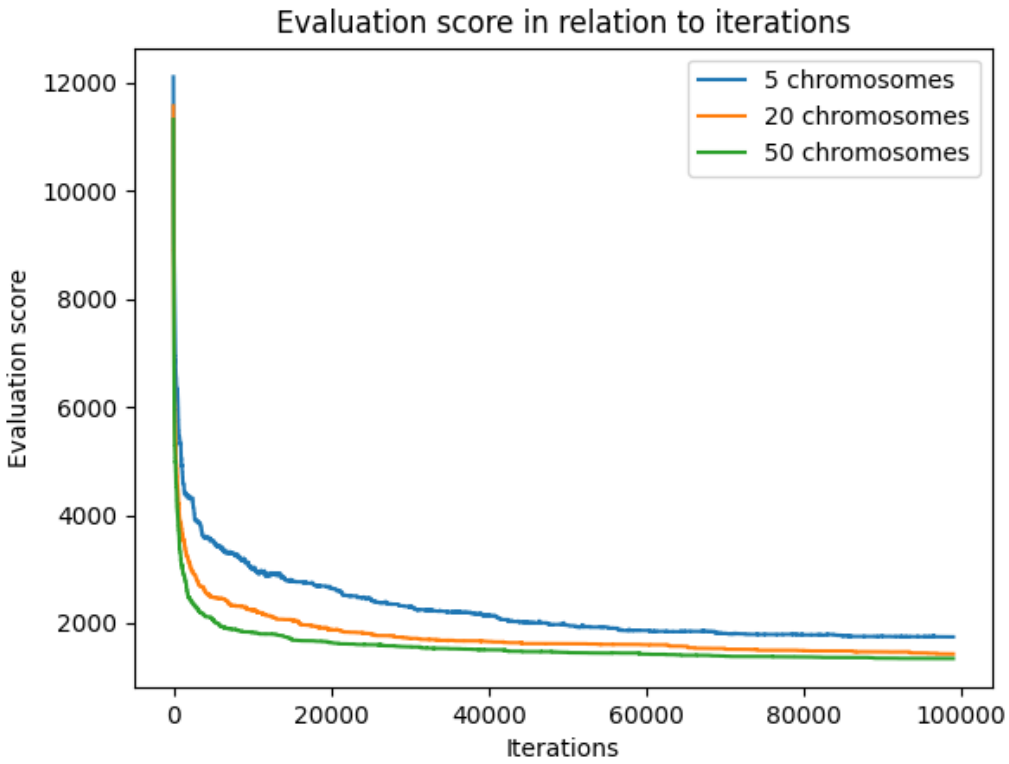


Rysunek 3: Przebieg pierwszej wersji algorytmu dla grafu xqf131

rysunku 9 przedstawiono znalezione ścieżki dla po dodaniu inwersji do mutacji. Wyniki dla obu grafów są przedstawione w tebeli 1 oraz 2.

xqf131		
populacja	inwersja	wynik
5	nie	758
20	nie	670
50	nie	606
50	tak	615
75	tak	588
100	tak	614

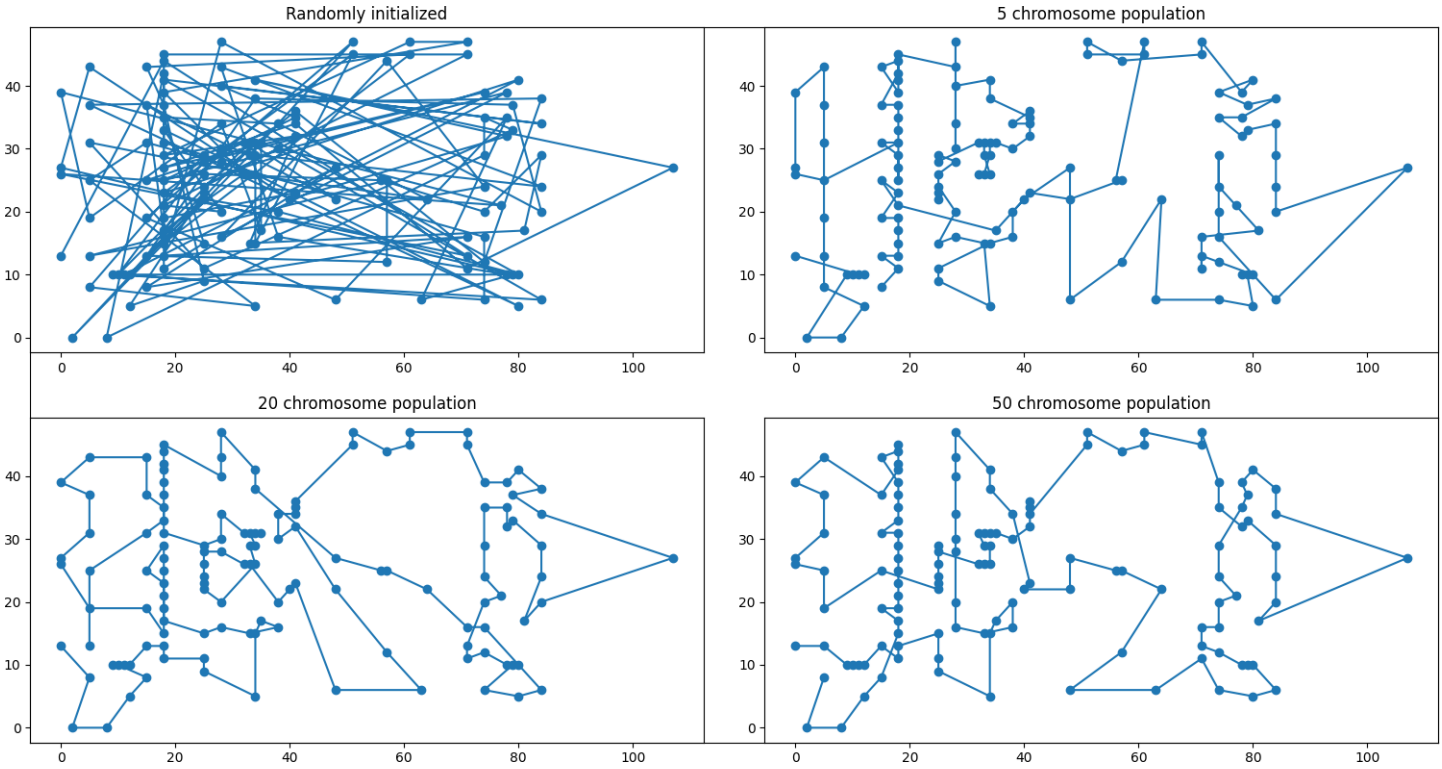
Tabela 1: Otrzymane wyniki dla grafu xqf131 po 10^6 iteracjach



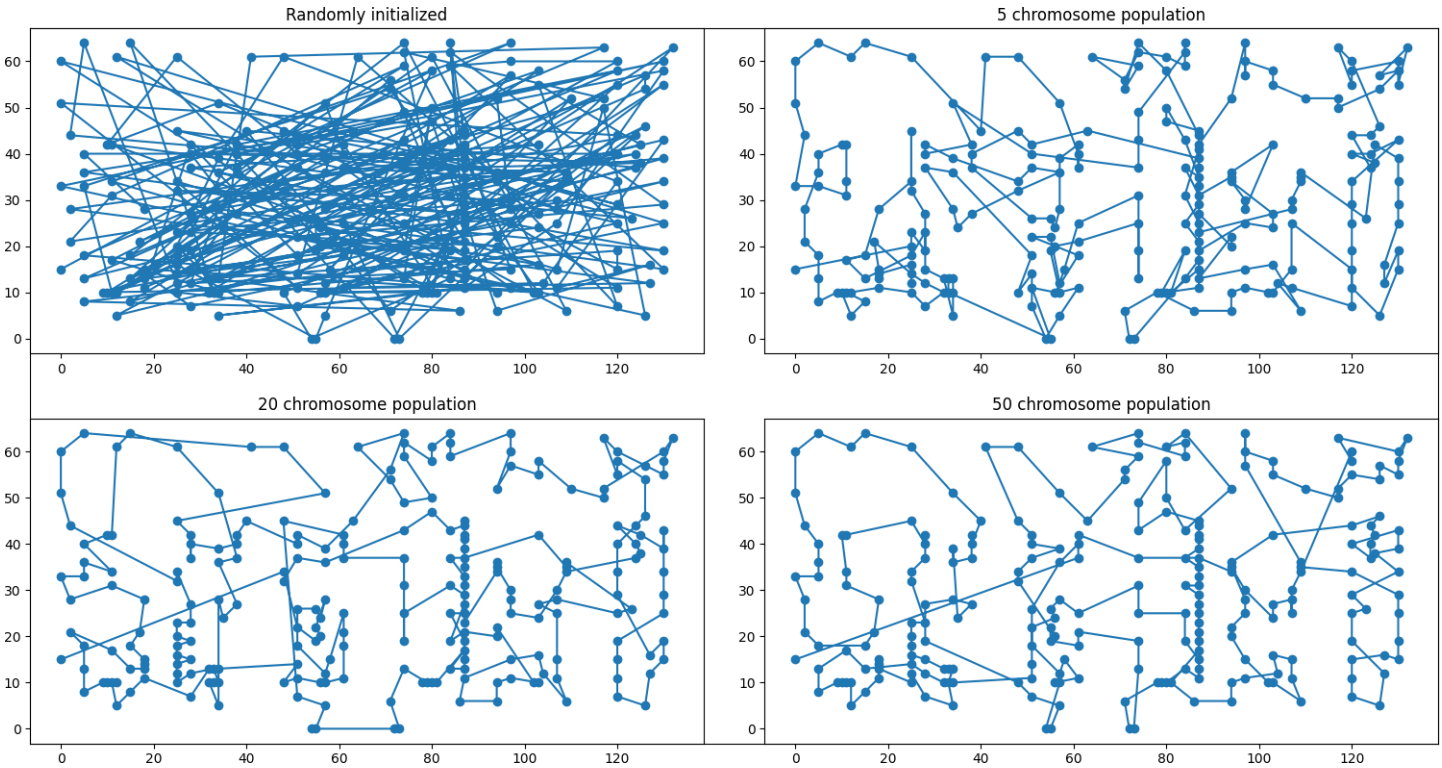
Rysunek 4: Przebieg pierwszej wersji algorytmu dla grafu xqg237

xqg237		
populacja	inwersja	wynik
5	nie	1744
20	nie	1428
50	nie	1344

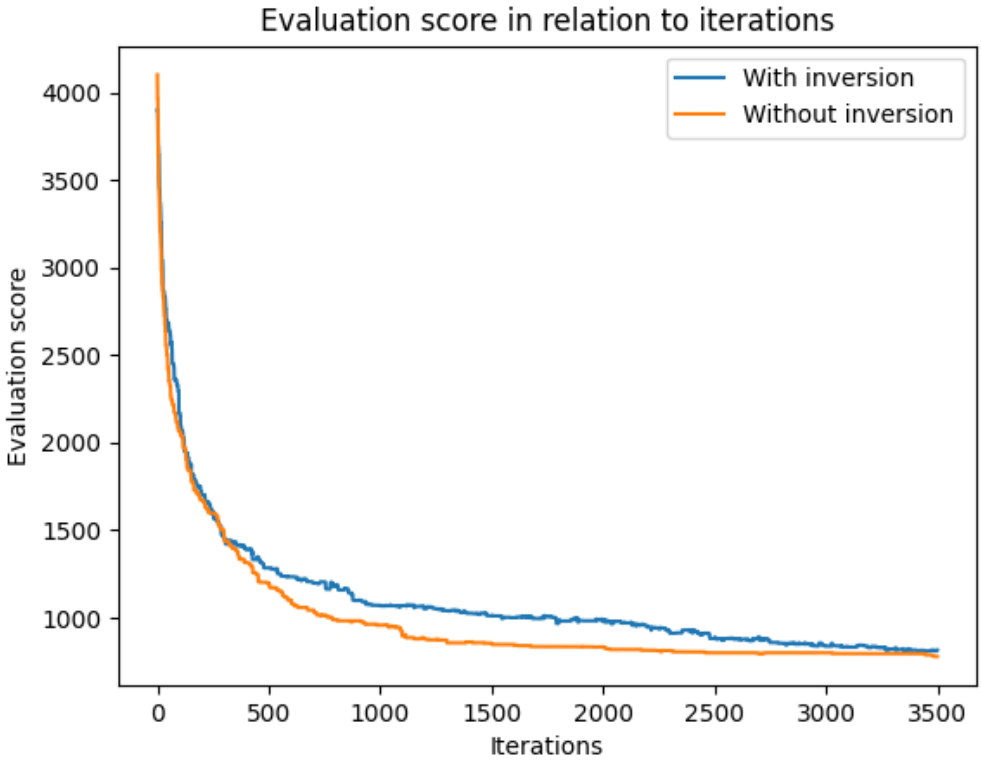
Tabela 2: Otrzymane wyniki dla grafu xqg237 po 10^6 iteracjach



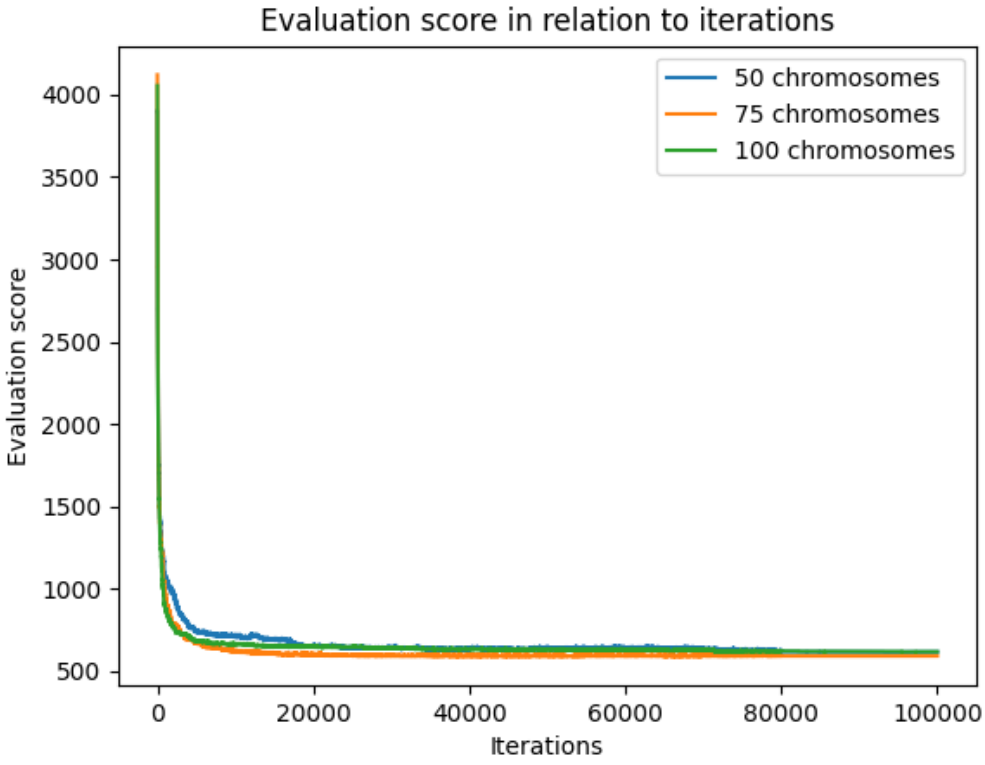
Rysunek 5: Znalezione rozwiązania grafu xqf131 dla poszczególnych populacji bez inwersji



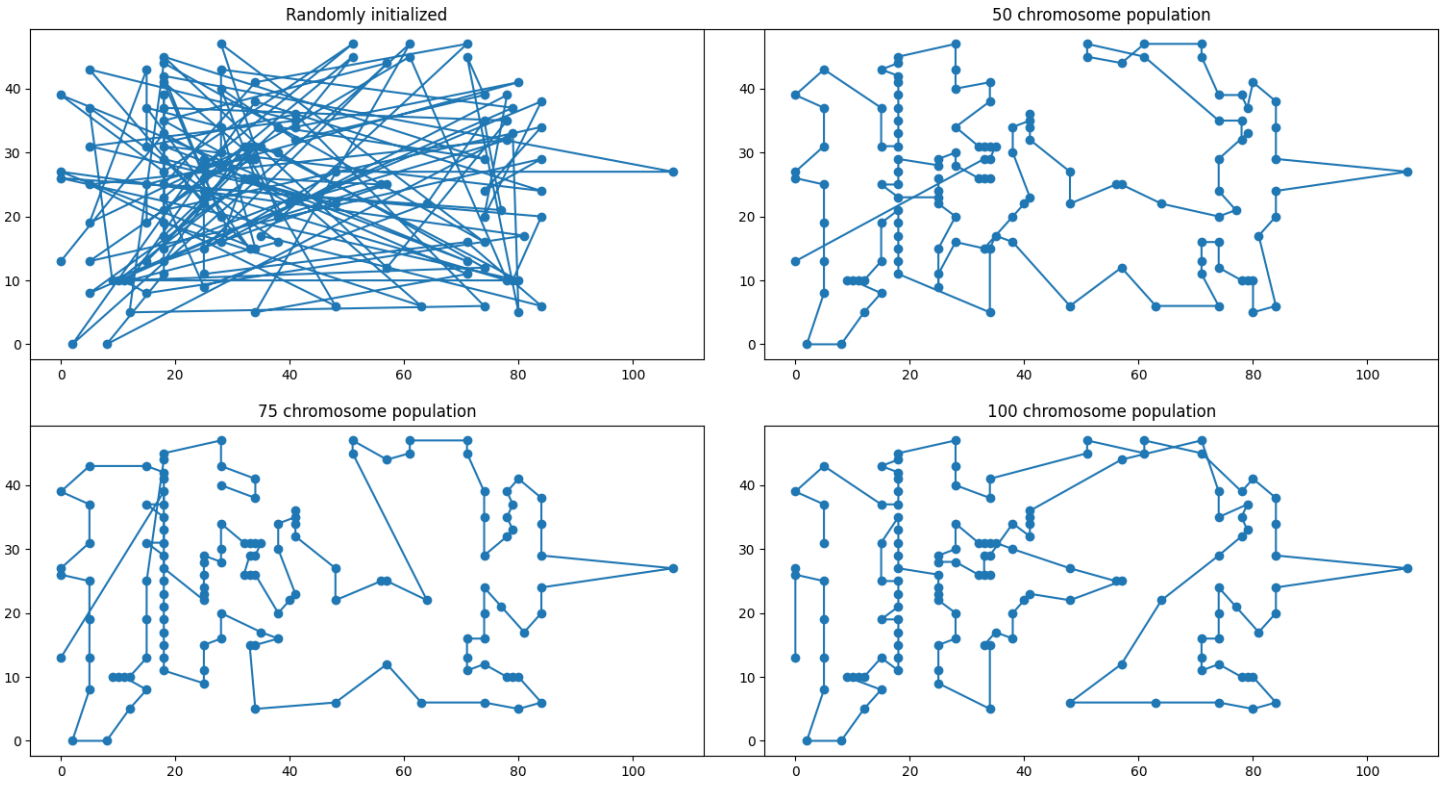
Rysunek 6: Znalezione rozwiązania grafu xqg237 dla poszczególnych populacji bez inwersji



Rysunek 7: Porównanie dwóch wersji algorytmu dla grafu xqf131 i 50 osobników



Rysunek 8: Przebieg drugiej wersji algorytmu dla grafu xqf131



Rysunek 9: Znalezione rozwiązanie grafu xqf131 dla poszczególnych populacji po dodaniu inwersji