|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Projekt z BMSSI | | | | | | |
| Rok akademicki | Termin | Rodzaj studiów | Kierunek | Prowadzący | Grupa | Sekcja |
| 2012/2013 | Wtorek (N) | NSI | INF | GB | 1 | 1 |
| 18:00 – 21:00 |

**Wykorzystanie algorytmów genetycznych do rozwiązywania problemów NP, na przykładzie problemu komiwojażera.**

Wykonawca projektu:

PAWŁOWSKI Robert

MAZIARZ Przemysław

**1. Cel projektu**

W projekcie należało przestawić praktyczne wykorzystanie metod rozwiązywania problemów obliczeniowych motywowanych obserwacjami z zakresu biologii. Podstawowymi metodami są tutaj sieci neuronowe lub algorytmy genetyczne.

W niniejszym raporcie przedstawiono realizację aplikacji, która rozwiązuje problem NP trudy z użyciem algorytmów genetycznych. Jako przykład problemu NP trudnego wybrano problem komiwojażera.

**2. Podstawy teoretyczne**

Algorytmy genetyczne są heurystyczną metodą optymalizacji inspirowaną genetyką i procesem ewolucji. Wykorzystując dalej analogię do procesu ewolucji, populacja to fragment przestrzeni poszukiwań w pewnym przedziale czasu, nazywanym dalej generacją. Kolejne generacje to kolejne iteracje z algorytmicznego punktu widzenia. Każdy osobnik opisany jest przez wybrany zestaw cech. Załóżmy, że liczba tych cech wynosi N. Wtedy charakterystyczna kombinacja N cech osobnika stanowi pojedyncze rozwiązanie problemu w N-wymiarowej przestrzeni rozwiązań.

W procesie ewolucji giną osobniki gorzej przystosowane, a przeżywaj osobniki lepiej przystosowane, posiadające zestaw cech korzystniejszy, np. ze względu na czynniki środowiska. Osobnik może nieco zmienić niektóre ze swoich cech, na skutek mutacji, które - jak to mutacje - mają charakter punktowy i losowy. Osobnik może ponadto powielić część swoich cech poprzez skojarzenie (rekombinację) z innym osobnikiem i wydanie potomstwa. Jednakże byt osobników nie jest łatwy. Prędzej czy później przychodzą zdarzenia, które stanowią element presji ewolucyjnej - selekcja, w wyniku której - zazwyczaj, choć niekoniecznie - osobniki najsłabsze muszą zginąć z populacji. Wraz z kolejnymi pokoleniami, populacja staje się coraz silniejsza, osobniki coraz lepiej przystosowane. O tym kiedy zakończyć proces ewolucji i wybrać najlepszego osobnika w populacji, jako rozwiązanie problemu, decyduje tylko przyjęte przez nas kryterium. Kryterium może być tak proste jak określona z góry liczba iteracji. A jest to tylko jeden z parametrów jakie można modyfikować w algorytmach genetycznych. Użytkownik algorytmu genetycznego powinien mieć możliwość wyboru szeregu algorytmów, które będą definiować przebieg mutacji, krzyżowania i selekcji. Co więcej na różnych etapach działania algorytmu wprowadzane mogą być pewne heurystyczne modyfikatory wybranych algorytmów, np. selekcja elitarna lub eliminacja osobników najstarszych.

Przejdźmy teraz do problemu komiwojażera. Jest to zagadnienie relatywnie młode. Pierwsze publikacje na temat problemu komiwojażera pojawiły się w larach 20-tych XX wieku [2]. Wówczas matematyk i ekonom Karl Menger opublikował broszurę poruszającą zagadnienie problemu komiwojażera. W 1932 Menger wydał publikację pod tytułem: „Das Botenproblem”, w której definiował on „problem posłańca” jako: "zadanie znalezienia, dla skończonej liczby punktów, których, parami, odległości są znane, najkrótszej ścieżki łączącej te punkty… Zasada, wg której dla punktu startowego powinno się wybrać punkt leżący najbliżej niego, itd., nie daje w rezultacie najkrótszej ścieżki." Dopiero w latach 50-tych pojawiła się próba rozwiązania problemu, zaproponowana przez George’a B. Dantzig’a, Fulkerson’a i Johnson’a (1954). Przez dziesięciolecia metoda ta pozostawała jedyną dla liczby miast rzędu kilkuset. W późniejszym okresie powstało kilka algorytmów dających rozwiązanie przybliżone, dające co najwyżej 3, 2 lub 1,5-razy gorsze rozwiązanie od rozwiązania optymalnego [2]:

* algorytm 3-przybliżony, 1955, G. Morton i A.H. Land,
* algorytm 2-przybliżony, 1956, Merill M. Flood,
* algorytm 1,5-przybliżony, 1975, N. Christofides.

Rozwój komputerów umożliwił opracowanie metod heurystycznych, w tym przedstawionych pokrótce algorytmów genetycznych, które potrafią znaleźć lepsze rozwiązania w bardzo przyzwoitym czasie.

Formalnie problem komiwojażera jest problemem optymalizacji, w którym należy znaleźć minimalny cykl Hamiltona w pełnym grafie ważonym. Graf jest tworzony przez połączenia pomiędzy miastami. Pełność grafu oznacza, że pomiędzy każdymi dwoma miastami istnieje bezpośrednie połączenie. Wagami grafu może być czas podróży, koszt podróży lub dowolna inna ilościowa miara jakości przebytej drogi. Natomiast cykl Hamiltona jest cyklem w grafie, w którym każdy wierzchołek grafu przechodzony jest tylko jeden raz, za wyjątkiem pierwszego wierzchołka.

**3. Zaimplementowane operatory mutacji**

**3.1. Insert mutation - mutacja przez wstawianie.**

W mutacja przez wstawianie wybieramy dwa losowe allele i ustawiamy jest obok siebie, przy czym jeden z alleli zachowuje swoją wcześniejszą pozycję. Pozostałe allele są przesuwane tak by wypełnić powstałe wolne miejsce i zachować sąsiedztwo oraz porządek. Pod względem wspomnianych dwóch cech (sąsiedztwa i porządku), jest to najbardziej konserwatywny z zaimplementowanych operatorów mutacji.



Rys. 1. Przykład mutacji przez wstawianie.

**3.2. Swap mutation - mutacja przez zamianę**

W tej metodzie mutacji chromosomów reprezentujących permutacje wybieramy dwa losowe allele i zamieniamy je miejscami. Zaburzenie sąsiedztwa i porządku jest tutaj większe niż w przypadku mutacji przez wstawianie.



Rys. 2. Przykład mutacji przez zamianę.

**3.3. Inversion mutation - mutacja przez odwracanie**

Mutacja przez odwracanie polega na wybraniu w sposób losowy fragmentu chromosomu i odwrócenie kolejności alleli w nim występujących. Ta metoda zaburza przede wszystkim informację o sąsiedztwie alleli.



Rys. 3. Przykład mutacji przez odwrócenie.

**3.4. Scramble mutation - mutacja przez wymieszanie**

Operator mutacji przez wymieszanie wybiera losowy fragment chromosomu i przestawia allele w nim występujące w sposób losowy. Ta metoda mutacji potencjalnie wprowadza najwięcej zmian w informacji o sąsiedztwie i kolejności alleli.

****

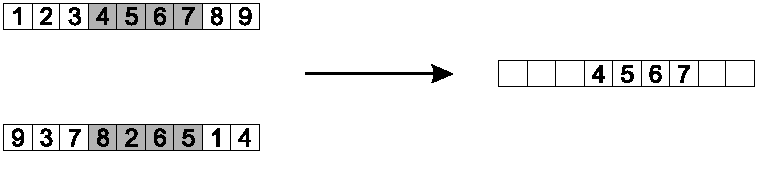
Rys. 4. Przykład mutacji przez wymieszanie.

**4. Zaimplementowane operatory krzyżowania**

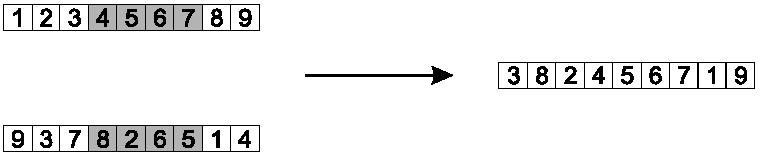
**4.1. Order Crossover - krzyżowanie porządkowe**

Krzyżowanie porządkowe zachowuje głównie informację o kolejności alleli w chromosomie. Algorytm można opisać w czterech punktach:

1. Wybierz arbitralnie (np. losowo) fragment chromosomu pierwszego rodzica.
2. Skopiuj wybrany fragment wprost do chromosomu pierwszego dziecka. Zachowaj pozycję alleli przy kopiowaniu.
3. Skopiuj allele, które nie występują w wybranym fragmencie do pierwszego dziecka:
   1. zaczynając zaraz za skopiowanym fragmentem,
   2. uwzględniając kolejność występowania alleli w chromosomie drugiego rodzica,
   3. „zawijając” allele, gdy osiągniesz koniec chromosomu.
4. Postępuj analogicznie dla drugiego dziecka, zamieniając rolami chromosomy rodzicielskie.

****

Rys. 5. Krzyżowanie porządkowe. Kopiowanie wybranego fragmentu chromosomu.

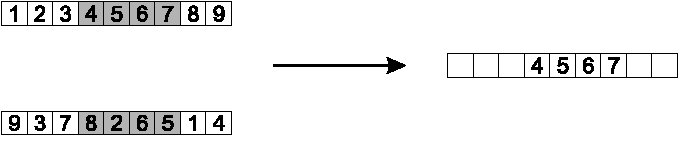
****

Rys. 6. Krzyżowanie porządkowe. Kopiowanie pozostałych alleli od drugiego z rodziców z zachowaniem porządku.

**4.2. Partially Mapped Crossover (PMX) - krzyżowanie z częściowym mapowaniem**

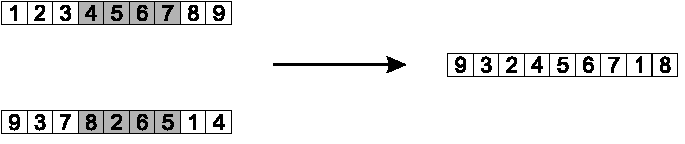
Algorytm krzyżowania z częściowym mapowaniem jest raczej złożony:

1. Wybierz losowo fragment chromosomu pierwszego rodzica i skopiuj go wprost do chromosomu pierwszego dziecka.
2. Zaczynając od pierwszego skopiowanego allelu, spójrz na allele występujące w tym samym fragmencie chromosomu drugiego rodzica, które nie zostały skopiowane.
3. Dla każdego z nieskopiowanych alleli, oznaczmy go *i*, spójrz na chromosom dziecka jaki element *j* został skopiowany na jego pozycji od pierwszego rodzica.
4. Ustaw *i* na pozycji zajmowanej przez *j* w chromosomie drugiego rodzica, chyba że wiemy, że nie będziemy *i* występuje już w chromosomie dziecka.
5. Jeżeli pozycja zajmowana przez *j* w chromosomie drugiego rodzica została już zajęta w chromosomie dziecka przez allel *k*, ustaw *i* na pozycji zajmowanej przez *k* w chromosomie drugiego rodzica.
6. Jeżeli rozważymy wszystkie elementy skopiowane w punkcie 1., reszta pozycji zostaje wypełniona na podstawie chromosomu drugiego rodzica.
7. Drugiego dziecko tworzymy analogicznie.

****

Rys. 7. Krzyżowanie z częściowym mapowanie. Kopiowanie wybranego fragmentu chromosomu.

Rys. 8. Krzyżowanie z częściowym mapowanie. Mapowanie.

****

Rys. 9. Krzyżowanie z częściowym mapowanie. Uzupełnianie brakujących elementów na podstawie chromosomu drugiego rodzica.

**4.3. Cyclic Crossover - krzyżowanie cykliczne**

Podstawowe założenie algorytmu to pochodzenie każdego allelu od jednego z rodziców z informacją o jego położeniu. Procedura realizacji algorytmu:

1. Utwórz cykl alleli od pierwszego rodzica:
   1. zacznij od pierwszego allelu pierwszego rodzica,
   2. spójrz na allel na tej samej pozycji u drugiego z rodziców,
   3. idź do pozycji z tym samym allelem u pierwszego z rodziców,
   4. dodaj ten allel do cyklu,
   5. powtarzaj kroki b-d aż dojdziesz do allelu z kroku a.
2. Dodaj allele tworzące cykl do chromosomu pierwszego dziecka, na pozycjach jakie zajmowały one u pierwszego rodzica.
3. Drugi cykl weź od drugiego z rodziców, postępując w analogiczny sposób.

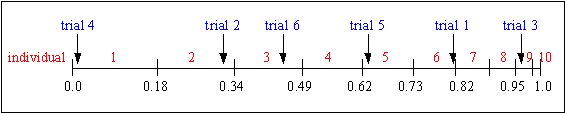
Rys. 10. Krzyżowanie cykliczne. Detekcja cykli.

Rys. 11. Krzyżowanie cykliczne. Kopiowanie cykli.

**5. Zaimplementowane operatory selekcji**

**5.1. Selekcja metodą koła ruletki**

Selekcja metodą koła ruletki mapuje w przedział [0; 1] udział oceny dopasowania danego osobnika w sumie dopasować wszystkich osobników. W ten sposób, w przedziale [0; 1] każdemu osobnik odpowiada pewien podprzedział. Obrazowo, każdemu osobnik odpowiada powierzchnia koła ruletki proporcjonalna do jego oceny dopasowania. Dysponując tak przygotowanymi danymi o populacji, dokonujemy wyboru N osobników, którzy utworzą nową populację, przez N generacji liczby losowej z przedziału [0; 1] (N obrotów kołem ruletki).



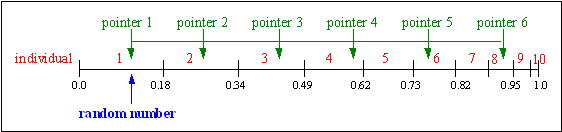
Rys. 11. Selekcja metodą koła ruletki.

Wadą metody koła ruletki jest przede wszystkim możliwość wyboru tego samego osobnika N krotnie.

**5.2. Stochastic Universal Sampling (SUS)**

Metoda o akronimie SUS nazywana jest czasem modą koła ruletki z wieloma wskaźnikami. Metoda SUS posiada lepsze właściwości statystyczne niż standardowa metoda koła ruletki. Wybranie tego samego osobnika N krotnie jest tutaj mniej prawdopodobne.

Analogicznie do metody koła ruletki, w algorytmie SUS każdemu osobnik odpowiada powierzchnia koła ruletki proporcjonalna do jego oceny dopasowania. Inaczej przebiega tutaj wybieranie osobników. Mianowicie losowanie odbywa się tylko raz. Losowana jest liczba z zakresu [0; <wartość największego udziału>]. Natomiast pozostałe N-1 osobników wybierane jest przez dodawanie do wylosowanej liczby wartości 1/N.



Rys. 11. Selekcja metodą SUS.

**5.3. Selekcja turniejowa**

Selekcja turniejowa swoim przebiegiem przypomina rozgrywanie zawodów. Populacja jest dzielona na podgrupy, gdzie rozmiar podgrupy jest parametrem algorytmu. Z każdej podgrupy wybierany jest najlepszy osobnik. Liczba podgrup jest równa liczbie osobników w nowej populacji.

**5.4. Selekcja elitarna**

Selekcja elitarna jest została zrealizowana w projekcie jako opcjonalny modyfikator selekcji, który może zostać dodany do każdej z metod opisanych w punktach 5.1. ÷ 5.3. Selekcja elitarna polega wyborze określonej liczby najlepiej dopasowanych osobników i przeniesieniu ich wprost do nowej populacji. W ten sposób mamy gwarancję, że przypadkowo nie utracimy dobrego rozwiązania. Z drugiej jednak strony możemy dzięki temu utknąć w pewnym ekstremum lokalnym rozwiązywanego problemu optymalizacji.

**5.5. Eliminacja osobników najstarszych**

Jest to kolejny z zaimplementowanych w projekcie modyfikatorów selekcji. Modyfikator może zostać dodany do każdej z metod opisanych w punktach 5.1. ÷ 5.3. Modyfikator usuwa z populacji określoną liczbę osobników przebywających w populacji najdłużej.

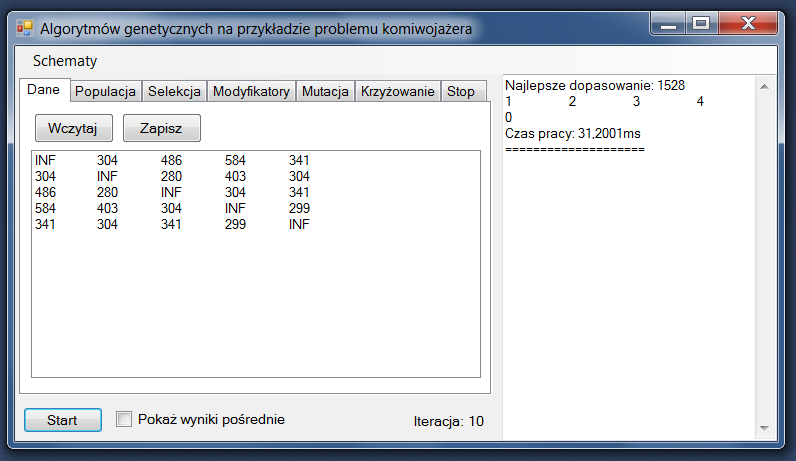
Osobniki przebywające w populacji najdłużej są przeważnie osobniki dobrze dopasowanymi. Zastosowanie tego modyfikatora z jednej strony spowalnia proces ewolucji. Aczkolwiek należy zauważyć, że za pomocą tego operatora możemy szybko wydostać się z ekstremum lokalnego. Ten modyfikator może być szczególnie przydatny w początkowych iteracjach procesu ewolucji, gdy chcemy „przeczesać” przestrzeń poszukiwań.

**6. Specyfikacja zewnętrzna**

Aplikacja została napisana w języku C# wykorzystującego platformę .NET Framework 4. Po uruchomieniu programu ukazuje się nam okno. Po lewej stronie widoczne są zakładki (rys. 14), w których możemy modyfikować parametry inicjalizujące algorytm genetyczny. Program umożliwia wczytywanie z pliku wcześniej przygotowanych grafów z pliku, po przez kliknięcie na przycisk ‘Wczytaj’. Parametry które można modyfikować to:

* rozmiar populacji,
* algorytm selekcji,
* modyfikator selekcji,
* algorytm mutacji,
* algorytm krzyżowania.

W momencie wybrania prawidłowych parametrów, możemy dokonać przeszukania najkrótszej drogi, po przez kliknięcie w przycisk ‘Start’. W oknie umieszczonym w prawej części programu wyświetlone zostaną wynik wyszukiwania.



Rys. 14. Główne okno programu.

**7. Testowanie algorytmu i wyniki**

**7.1. Przykład nr 1**

Testowanie algorytmu polegało na przeprowadzeniu pięć prób dla każdego zaimplementowanego wariantu algorytmu genetycznego. Poddawanymi testowi wartościami było położenie pięciu głównych polskich miast zgodnych z rysunkiem 2.

****

Rys. 15. Testowany problem.

Rysunek przedstawia odległość pomiędzy poszczególnymi matami. Każde miasto posiada przyporządkowany identyfikator, po którym można go rozpoznać w grafie. Przykładowo, Gdańsk = 0, Wrocław = 2.

W wyniku testowania algorytmu otrzymujemy zbiór liczb, które przedstawiają numer miasta od którego należy wyruszyć aby droga podczas której chcemy odwiedzić wszystkie miasta i wrócić do miasta początkowego była najkrótsza.

Poniżej zostały przestawione testy dla różnych ustawień algorytmu genetycznego:

**Testowanie 1**

**liczba populacji 100**

**Rodzaj Selekcji Stochastic Universal Sampling (SUS) StochasticUniversalSamplingStrategy**

**Rodzaj Mutacji mutacja przez wstawianie Insert mutation**

**Rodzaj krzyżowania krzyżowanie cykliczne Cyclic Crossover**

**Nr Proby**

**1 1 2 3 4 0**

**2 0 4 3 2 1**

**3 3 4 0 1 2**

**4 0 4 3 2 1**

**5 3 4 0 1 2**

**Testowanie 2**

**liczba populacji 100**

**Rodzaj Selekcji Stochastic Universal Sampling (SUS) StochasticUniversalSamplingStrategy**

**Rodzaj Mutacji mutacja przez wstawianie Insert mutation**

**Rodzaj krzyżowania krzyżowanie z częściowym mapowaniem Partially Mapped Crossover (PMX)**

**Nr Proby**

**1 2 3 4 0 1**

**2 3 4 0 1 2**

**3 0 1 2 3 4**

**4 0 1 2 3 4**

**5 3 2 1 0 4**

**Testowanie 3**

**liczba populacji 100**

**Rodzaj Selekcji Stochastic Universal Sampling (SUS) StochasticUniversalSamplingStrategy**

**Rodzaj Mutacji mutacja przez zamianę Swap mutation**

**Rodzaj krzyżowania krzyżowanie cykliczne Cyclic Crossover**

**Nr Proby**

**1 2 1 0 4 3**

**2 3 2 1 0 4**

**3 2 1 0 4 3**

**4 3 2 1 0 4**

**5 4 3 2 1 0**

**Testowanie 4**

**liczba populacji 100**

**Rodzaj Selekcji Stochastic Universal Sampling (SUS) StochasticUniversalSamplingStrategy**

**Rodzaj Mutacji mutacja przez zamianę Swap mutation**

**Rodzaj krzyżowania krzyżowanie z częściowym mapowaniem Partially Mapped Crossover (PMX)**

**Nr Proby**

**1 1 2 3 4 0**

**2 1 0 4 3 2**

**3 1 2 3 4 0**

**4 0 1 2 3 4**

**5 1 2 3 4 0**

**Testowanie 5**

**liczba populacji 100**

**Rodzaj Selekcji Stochastic Universal Sampling (SUS) StochasticUniversalSamplingStrategy**

**Rodzaj Mutacji mutacja przez odwracanie Inversion mutation**

**Rodzaj krzyżowania krzyżowanie cykliczne Cyclic Crossover**

**Nr Proby**

**1 1 0 4 3 2**

**2 1 2 3 4 0**

**3 3 2 1 0 4**

**4 4 0 1 2 3**

**5 2 3 4 0 1**

**Testowanie 6**

**liczba populacji 100**

**Rodzaj Selekcji Stochastic Universal Sampling (SUS) StochasticUniversalSamplingStrategy**

**Rodzaj Mutacji mutacja przez odwracanie Inversion mutation**

**Rodzaj krzyżowania krzyżowanie z częściowym mapowaniem Partially Mapped Crossover (PMX)**

**Nr Proby**

**1 2 1 0 4 3**

**2 1 2 3 4 0**

**3 0 1 2 3 4**

**4 2 1 0 4 3**

**5 1 0 4 3 2**

**Testowanie 7**

**liczba populacji 100**

**Rodzaj Selekcji Stochastic Universal Sampling (SUS) StochasticUniversalSamplingStrategy**

**Rodzaj Mutacji mutacja przez wymieszanie Scramble mutation**

**Rodzaj krzyżowania krzyżowanie cykliczne Cyclic Crossover**

**Nr Proby**

**1 4 3 2 1 0**

**2 1 0 4 3 2**

**3 0 4 3 2 1**

**4 3 2 1 0 4**

**5 3 2 1 0 4**

**Testowanie 1**

**liczba populacji 100**

**Rodzaj Selekcji metodą koła ruletki RouletteWheelSelectionStrategy**

**Rodzaj Mutacji mutacja przez wstawianie Insert mutation**

**Rodzaj krzyżowania krzyżowanie cykliczne Cyclic Crossover**

**Nr Proby**

**1 0 1 2 3 4**

**2 2 3 4 0 1**

**3 4 3 2 1 0**

**4 2 3 4 0 1**

**5 0 4 3 2 1**

**Testowanie 2**

**liczba populacji 100**

**Rodzaj Selekcji metodą koła ruletki RouletteWheelSelectionStrategy**

**Rodzaj Mutacji mutacja przez wstawianie Insert mutation**

**Rodzaj krzyżowania krzyżowanie z częściowym mapowaniem Partially Mapped Crossover (PMX)**

**Nr Proby**

**1 3 4 0 1 2**

**2 1 2 3 4 0**

**3 3 4 0 1 2**

**4 1 2 3 4 0**

**5 3 2 1 0 4**

**Testowanie 3**

**liczba populacji 100**

**Rodzaj Selekcji metodą koła ruletki RouletteWheelSelectionStrategy**

**Rodzaj Mutacji mutacja przez zamianę Swap mutation**

**Rodzaj krzyżowania krzyżowanie cykliczne Cyclic Crossover**

**Nr Proby**

**1 4 3 2 1 0**

**2 1 2 3 4 0**

**3 3 4 0 1 2**

**4 1 0 4 3 2**

**5 4 3 2 1 0**

**Testowanie 4**

**liczba populacji 100**

**Rodzaj Selekcji metodą koła ruletki RouletteWheelSelectionStrategy**

**Rodzaj Mutacji mutacja przez zamianę Swap mutation**

**Rodzaj krzyżowania krzyżowanie z częściowym mapowaniem Partially Mapped Crossover (PMX)**

**Nr Proby**

**1 1 2 3 4 0**

**2 1 0 4 3 2**

**3 1 2 3 4 0**

**4 0 1 2 3 4**

**5 1 2 3 4 0**

**Testowanie 5**

**liczba populacji 100**

**Rodzaj Selekcji metodą koła ruletki RouletteWheelSelectionStrategy**

**Rodzaj Mutacji mutacja przez odwracanie Inversion mutation**

**Rodzaj krzyżowania krzyżowanie cykliczne Cyclic Crossover**

**Nr Proby**

**1 2 3 4 0 1**

**2 1 0 4 3 2**

**3 2 3 4 0 1**

**4 3 2 1 0 4**

**5 4 3 2 1 0**

**Testowanie 6**

**liczba populacji 100**

**Rodzaj Selekcji metodą koła ruletki RouletteWheelSelectionStrategy**

**Rodzaj Mutacji mutacja przez odwracanie Inversion mutation**

**Rodzaj krzyżowania krzyżowanie z częściowym mapowaniem Partially Mapped Crossover (PMX)**

**Nr Proby**

**1 1 0 4 3 2**

**2 0 4 3 2 1**

**3 2 1 0 4 3**

**4 0 1 2 3 4**

**5 1 0 4 3 2**

**Testowanie 7**

**liczba populacji 100**

**Rodzaj Selekcji metodą koła ruletki RouletteWheelSelectionStrategy**

**Rodzaj Mutacji mutacja przez wymieszanie Scramble mutation**

**Rodzaj krzyżowania krzyżowanie cykliczne Cyclic Crossover**

**Nr Proby**

**1 0 1 2 3 4**

**2 1 0 4 3 2**

**3 3 4 0 1 2**

**4 3 2 1 0 4**

**5 4 3 2 1 0**

**7.2. Przykład nr 2**

Ze względu na założenia naszego programu i samego algorytmu genetycznego, w którym to początkowa droga startowa jest dobierana losowo (w zależności jak się chromosomy wymieszają) trudno jest w rozsądny sposób dokonać pomiarów i zweryfikować działanie programu.

Program działa poprawnie dla wszystkich typów parametrów, sposób jego działania jest bardzo szybki, ponieważ przerwanie działania programu można w każdej chwili (czasowo lub po ilości iteracji).

Kolejnym z testów został przeprowadzony dla odległości 69 miast Polskich, (plik z odległościami został dołączony do sprawozdania w formacie .xls oraz .txt). Dla których chcielibyśmy iteracyjne znaleźć najkrótszą drogę dla wszystkich miast, musielibyśmy sprawdzić 69! Możliwości czyli : 1.71\*10^98. Zrealizowany program z kolei rozwiązuje ten problem w skrajnym przypadku da rozmiaru populacji wynoszącym 100 oraz maksymalnej liczbie iteracji: 1000.

*Po 2098 iteracjach, z wynikami*

*Najlepsze dopasowanie: 21277*

*27 66 34 3 7 9 40 26 58 25 55 36 43 39 13 28 64 35 32 59 53 63 12 52 14 45 37 31 33 21 51 42 22 24 60 44 2 23 4 20 15 5 65 8 50 57 6 17 68 61 1 29 41 62 19 38 0 47 16 46 10 18 48 49 54 56 11 30 67*

*Czas pracy:2959,1693ms*

*Lub po 1531 iteracjach z wynikami:*

*Najlepsze dopasowanie: 21588*

*67 44 54 56 50 17 63 58 65 64 25 68 28 11 2 52 16 36 35 27 40 20 57 3 53 29 30 15 14 10 49 34 9 38 47 48 66 43 23 55 4 32 31 18 12 1 42 21 8 59 26 6 61 5 33 51 13 41 60 37 0 7 46 39 19 22 62 24 45*

*Czas pracy:2336,1337ms*

**8. Wnioski**

W ramach projektu stworzono podstawy zrębu implementującego algorytmy genetyczne, który mógłby stać się częścią większej biblioteki metod obliczeniowych motywowanych biologicznie. W projekcie stworzono program wykorzystujący zręb dla rozwiązania najpopularniejszego problemu z rodziny problemów NP trudnych - problemu komiwojażera.

Opracowany program pozwala na modyfikację wielu parametrów algorytmu genetycznego, jak i dobór odpowiedniej kombinacji operatorów ewolucji. Aplikacja realizuje przetwarzanie wielowątkowe, dzięki czemu czas znajdowania suboptymalnego rozwiązania jest jeszcze krótszy!

**9. Literatura**

1. Internet: http://kaims.pl/~deren/gms/wyklady/04\_kom\_opt.pdf, dostęp: 9.12.12.
2. Internet: http://www.binboy.org/algorytmy-grafy/articles/140/Problem\_komiwojazera\_Sformuowanie\_problemu\_Rys\_historyczny\_Algorytm\_Christofidesa\_i\_jego\_analiza.html, dostęp: 17.12.12.
3. Internet: http://www.cs.vu.nl/~gusz/ecbook/slides/Genetic\_Algorithms.ppt, dostęp: 9.12.12.
4. Randy L. Haupt, Sue Ellen Haupt: Practical Genetic Algorithms. Wiley 2004.
5. Zbigniew Michalewicz: Algorytmy genetyczne + struktury danych = programy ewolucyjne. WNT, Warszawa 1996.