Projektowanie efektywnych algorytmów

Zadanie projektowe nr 3: Algorytm populacyjny

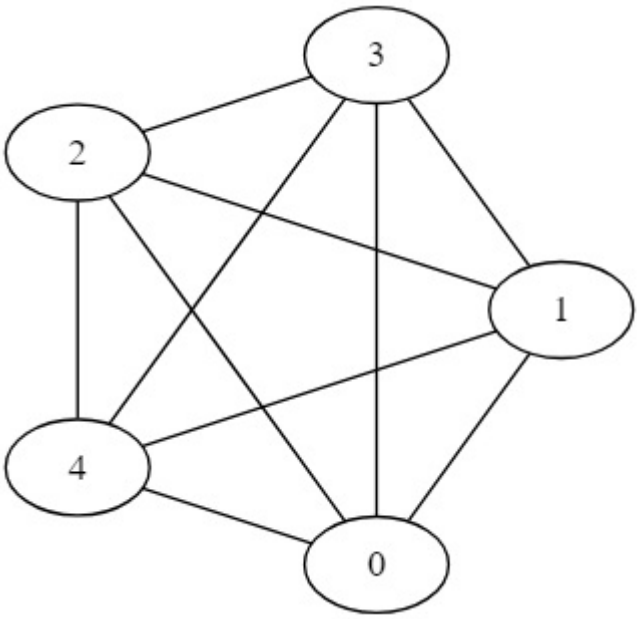
Autor	Prowadzący	Termin
Bartosz Rodziewicz (226105)	Mgr inż. Radosław Idzikowski	Piątek, 7:30

Opis problemu

Wybranym prze mnie problemem optymalizacyjnym jest Travelling Salesman Problem, czyli Problem Komiwojażera.

Polega on na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Cykl Hamiltona to taki cykl w grafie, w którym każdy wierzchołek grafu odwiedzany jest dokładnie raz (plus powrót do wierzchołka początkowego).

Jest to problem NP-trudny.



Powyżej znajduje się przykładowy graf dla 5-ciu miast. Każda krawędź posiada wagę, różną dla przejść w jedną i drugą stronę.

W programie graf jest reprezentowany jako macierz `n` x `n` z wagami przejść, gdzie `n` oznacza liczbę miast.

Metoda rozwiązania

Algorytmy Brute Force i Branch and Bound

W sprawozdaniu z etapu 1 dokładnie opisałem działanie i moją implementację powyższych algorytmów, więc nie będę tutaj tego powtarzać.

Są to algorytmy dokładne, więc zawsze znajdą najbardziej optymalny wynik, jednak posiadają bardzo dużą złożoność obliczeniową (w obu przypadkach `O(n!)`).

Algorytm Tabu Search

W sprawozdaniu z etapu 2 dokładnie opisałem działanie i moją implementację powyższego algorytmu, więc nie będę tutaj tego powtarzać.

Algorytm genetyczny

Wybranym przeze mnie algorytmem populacyjnym jest algorytm genetyczny.

W odróżnieniu od metody Tabu Search algorytm ten jest o wiele bardziej sprecyzowany.

Jego działanie wzięte zostało z biologii i tego jak poszczególne gatunki się rozwijają.

Ogólny zarys działania algorytmu prezentuje się następująco:

- Na start losowana jest początkowa populacja,
- Populacja zostaje poddana ocenie jakości i najsłabsze osobniki są odrzucane (tak, aby populacja miała stałą, z góry ustaloną, wielkość),
- Osobniki są poddawane działaniu operatorów ewolucyjnych:
  - Nowe osobniki są tworzone poprzez odpowiednie złączanie genotypów rodziców (**krzyżowanie**),
  - Osobniki są poddawane działaniu czynnika losowego (**mutacja**) delikatnie zmieniającego genotyp,

- Nowe osobniki (pokolenie) dodawane jest do ogólnej populacji. Populacja zostaje poddana ocenie i jeśli rozwiązanie spełnia wymaganą jakość algorytm zostaje przerwany, albo kontynuowany jest proces reprodukcji.

Moja implementacja charakteryzuje się następującymi cechami:

- Genotypem osobnika jest trasa komiwojażera,
- Fenotypem jest długość tej trasy,
- **Populacja startowa jest generowana całkowicie losowo**,
- Wielkość populacji jest parametrem algorytmu, domyślnie 50 osobników,
- **Kryterium spełnienia jest długość pracy algorytmu** (również parametr; domyślnie 30 sekund),
- Zastosowałem algorytm **krzyżowania OX**,
- Zaimplementowane są **dwa rodzaje mutacji - zamiana wierzchołków albo krawędzi** (parametr algorytmu; domyślnie wierzchołków),
- W przypadku krzyżowania, jak i mutacji w algorytmie są dwa parametry:
  - **Współczynnik krzyżowania** - prawdopodobieństwo, że dwa osobniki zostaną poddane reprodukcji i stworzą dwa nowe (domyślnie 0.8 - 80%),
  - **Współczynnik mutacji** - prawdopodobieństwo, że na osobniku zajdzie mutacja (domyślnie 0.01 - 1%).

Tak jak przeszukiwanie z zakazami, algorytm ten w żaden sposób nie gwarantuje znalezienie najbardziej optymalnego rozwiązania, jednak znacząco skraca czas wyznaczania w miarę optymalnego rozwiązania (użytkownik decyduje ile czasu algorytm pracuje).

Jeśli dla danej instancji problemu nie jest znane rozwiązanie optymalne, nie ma też żadnej możliwości aby oszacować stopnia błędu znalezionego przez algorytm rozwiązania.

Algorytm ten, od innych metod heurystycznych, wyróżnia:

- przetwarzanie populacji rozwiązań, prowadzące do równoległego przeszukiwania przestrzeni rozwiązań z różnych punktów,
- w celu ukierunkowania procesu przeszukiwania wystarczającą informacją jest jakość aktualnych rozwiązań,
- celowe wprowadzenie elementów losowych.

Ponieważ algorytm genetyczny jest algorytmem niedeterministycznym, nie można dla niego w całości określić czasowej złożoności obliczeniowej. Można jednak podać złożoność obliczeniową pojedynczego przeglądu całej populacji w celu reprodukcji, która wynosi `O(n!)`, gdzie `n` oznacza wielkość populacji.

Metoda testowania i plan eksperymentu

Wykonane przeze mnie testy można podzielić na dwa etapy.

Pierwszym etapem, w sumie głównym, jest, tak jak w przypadku Tabu Search, porównanie wpływu poszczególnych parametrów na działanie algorytmu przy czterech znanych instancjach problemu.

Za każdym razem testowany był jeden parametr zostawiając inne na ustawieniu domyślnym.

Po kolei testowane były następujące parametry:

- Czas pracy algorytmu: [1s, 10s, 30s, 60s]
- Wielkość populacji: [10, 30, 50, 100]
- Współczynnik krzyżowania: [20%, 40%, 80%, 99%]
- Współczynnik mutacji: [1%, 5%, 15%, 50%]
- Rodzaj mutacji: [wierzchołków, krawędzi]

Każdy parametr był testowany 10-cio krotnie i analizowane są znalezione wyniki. Na wykresie przedstawiona jest wartość błędu względnego znalezionego rozwiązania od rozwiązania optymalnego. Zamieszczone są trzy wartości - wartość błędu najgorszej wartości, średniej i najlepszej znalezionej w tych 10-ciu wywołaniach algorytmu.

Wybrane do testu instancje problemu to:

- `ftv33` - 34 miasta, instancja asymetryczna, trasa optymalna: 1286
- `brazil58` - 58 miasta, instancja symetryczna, trasa optymalna: 25395
- `ftv170` - 171 miasta, instancja asymetryczna, trasa optymalna: 2755
- `rbg443` - 443 miasta, instancja asymetryczna, trasa optymalna: 2720

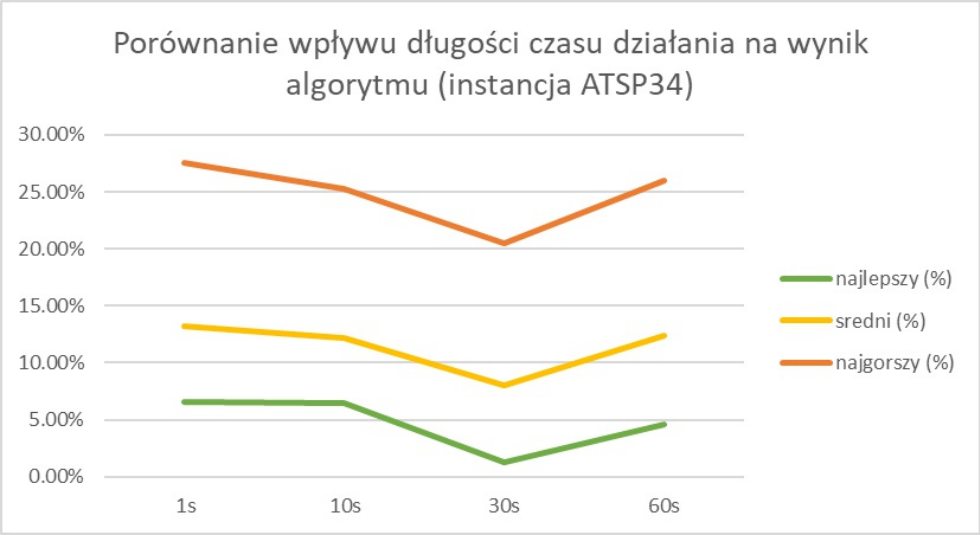
Drugą częścią było porównanie uzyskanych wyników wpływu czasu działania algorytmu genetycznego z wpływem na algorytm Tabu Search dla tej samej instancji problemu.

Pomiar czasu wykonywany był za pomocą `std::chrono::high_resolution_clock`, dostępnego w bibliotece standardowej C++11.

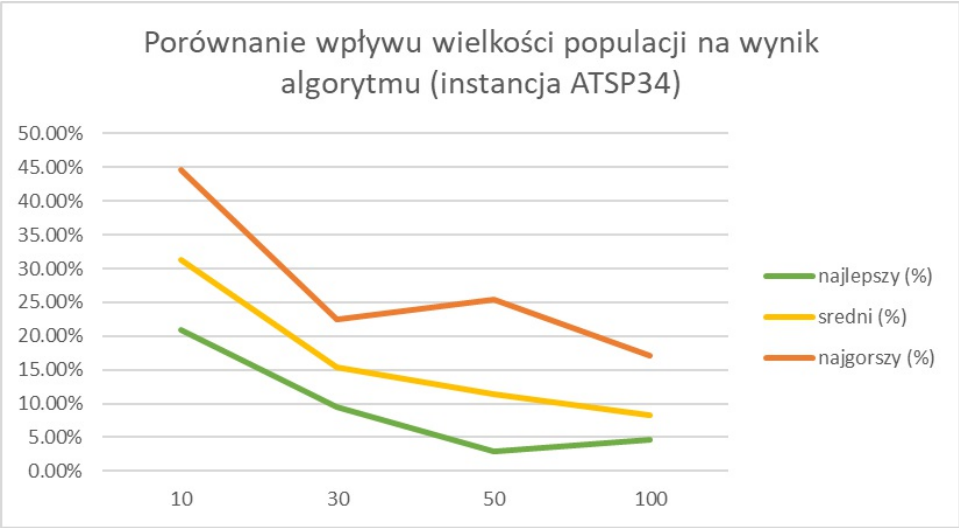
Wyniki pomiarów

Wpływ parametrów algorytmu na jego dokładność

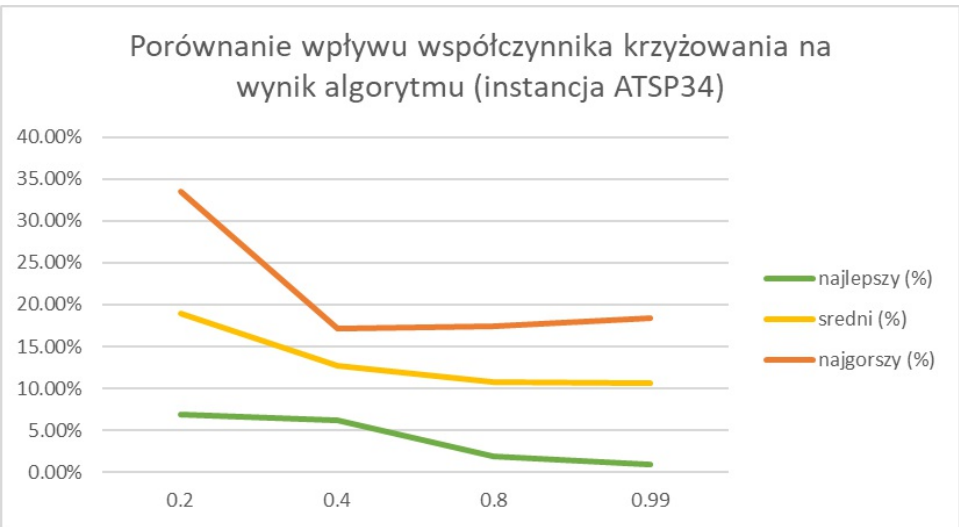
Instancja `ftv33` - ATSP34



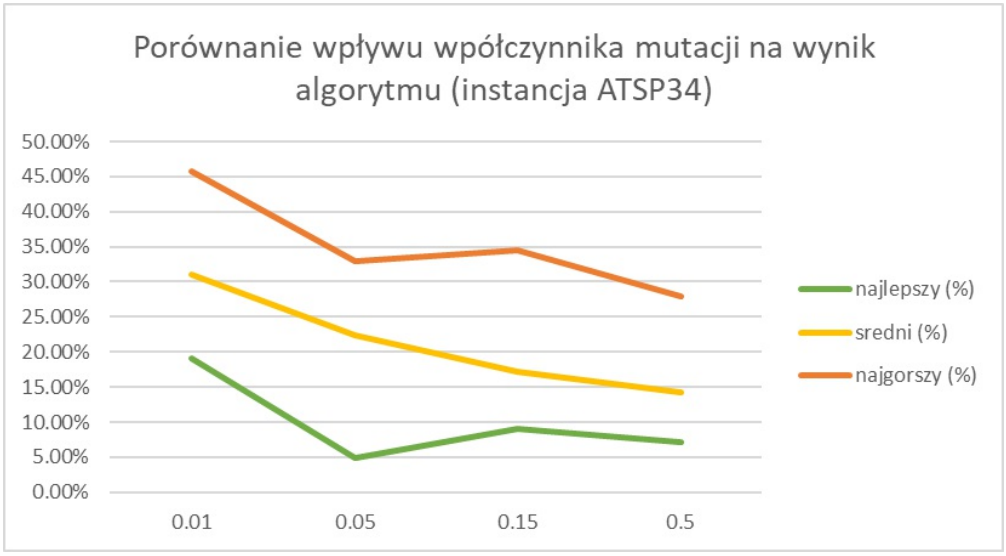
Ten wykres jest chyba najłatwiejszy do przewidzenia i generalnie zgadza się z przewidywaniem, czyli jakość rozwiązania poprawia się wraz z zwiększeniem długości pracy algorytmu. Zadziwiająca jest tutaj stosunkowo mała poprawa, mimo znaczącego zwiększania czasu, jednak o wiele lepiej widoczne będzie to przy większych instancjach. Ciężki do wytłumaczenia jest natomiast spadek jakości rozwiązania przy skoku z 30s na 60s i jedyny mój strzał może być w kierunku tego, że w momencie tamtego testu komputer był bardziej obciążony.



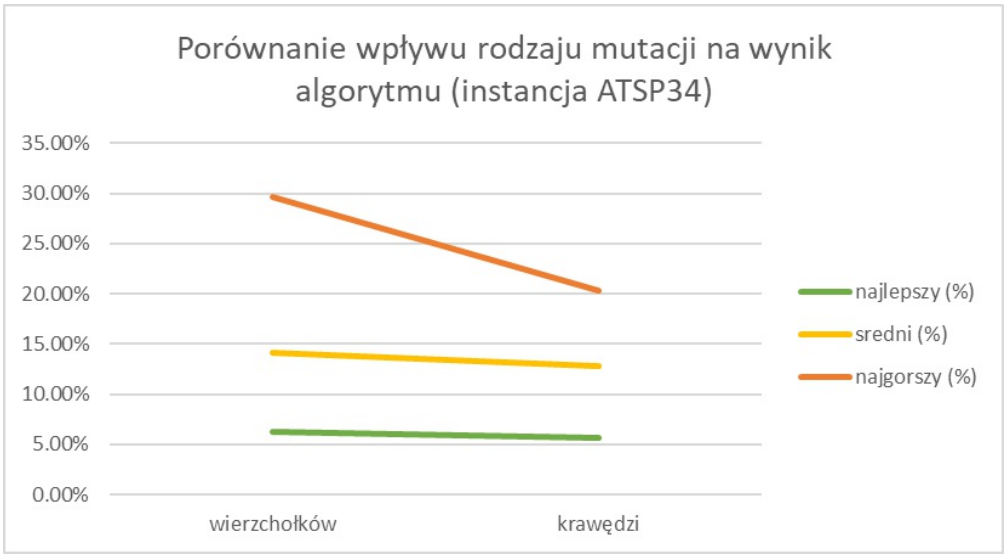
W tym przypadku również wynik był w miarę oczywisty - większa populacja daje większą liczbę miejsc w przestrzeni rozwiązań, która jest aktualnie sprawdzana, więc musiała powodować polepszenie rozwiązania.



Wpływ współczynnika krzyżowania, który tak na prawdę jest prawdopodobieństwem nie był już taki przewidywalny, jednak widać, że generalnie najlepszy efekt daje krzyżowanie wszystkiego z wszystkim.

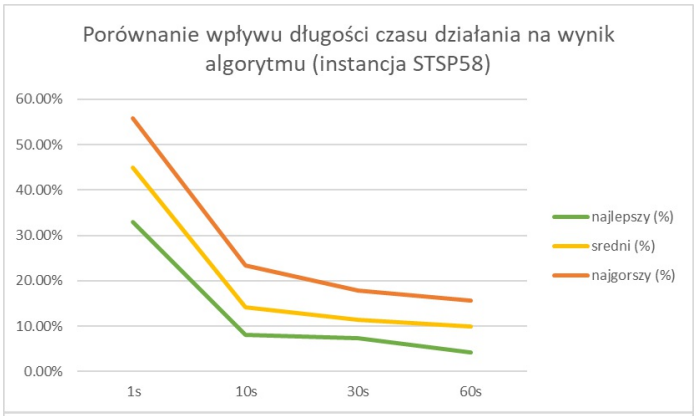


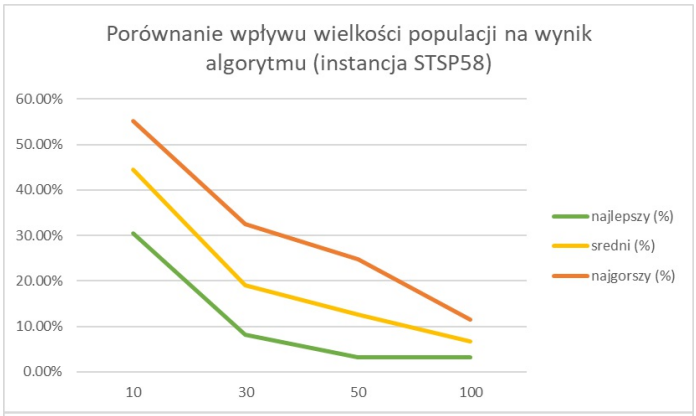
Wykres ukazujący współczynnik mutacji pokazuje, że wybrane przeze mnie domyślne ustawienie nie było najlepszym wyborem. Wygląda, że zwiększenie mutacji, czyli wprowadzenie większej losowości w rozwiązania dość sporo pomaga.

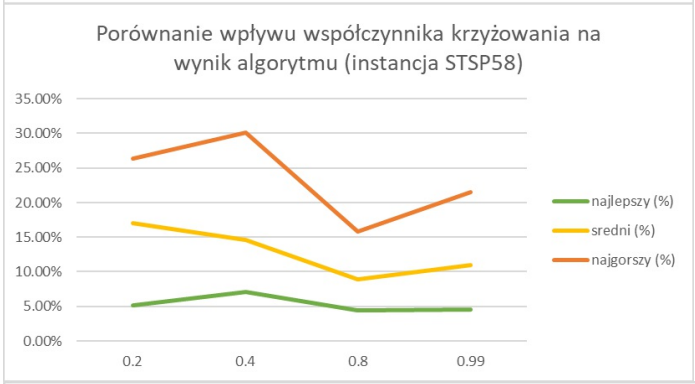


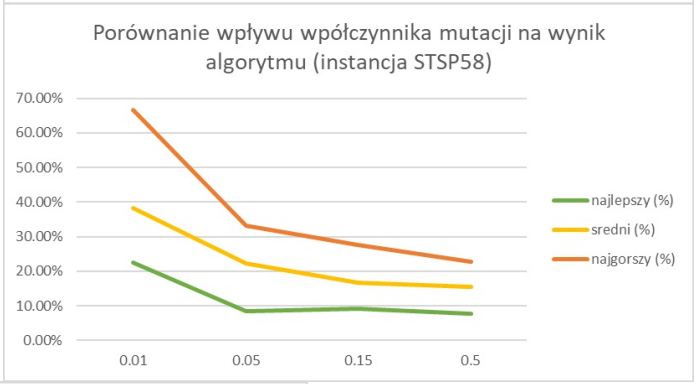
Z uwagi na słaby wynik poprzedniego parametru i jego zły wybór wydaje mi się, że ten test nie jest w żaden sposób miarodajny, zwłaszcza, że wynik jest praktycznie identyczny.

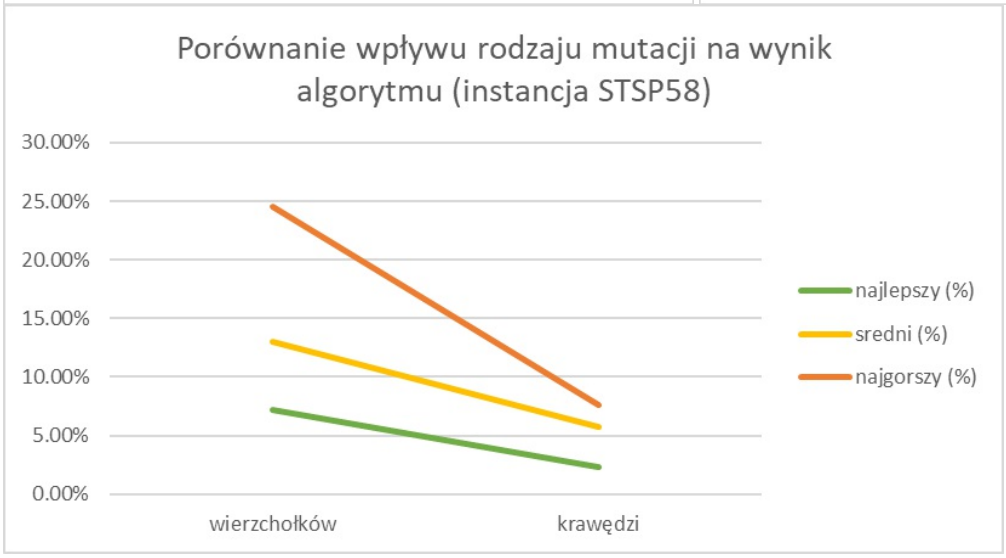
Instancja `brazil58` - STSP58





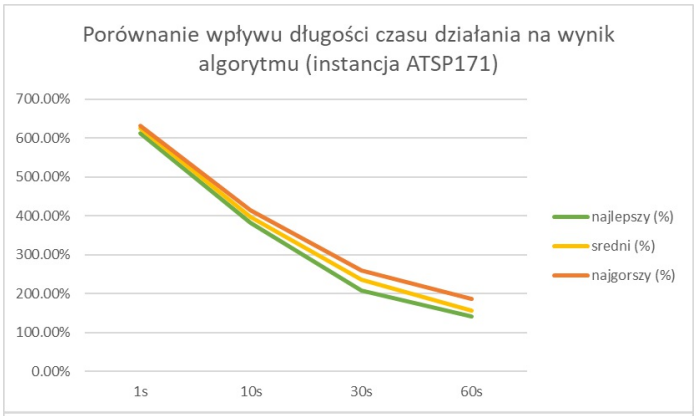


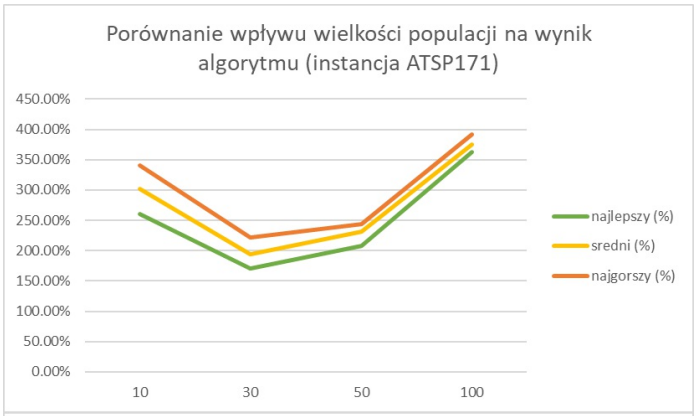


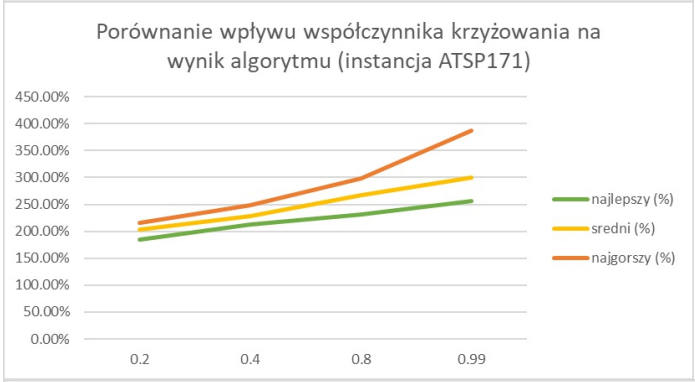


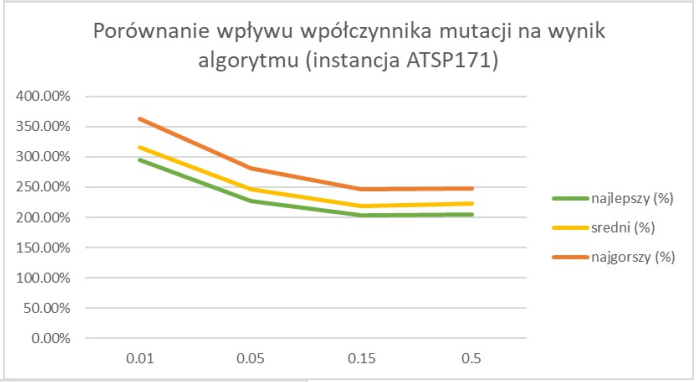
Generalnie powyższe wyniki potwierdzają obserwacje z poprzedniej instancji. Zauważyć można jednak lekki spadek wzrostu jakości. Wykres wpływu rodzaju mutacji na wynik algorytmu zdaje pokazywać, że wybór zamiany wierzchołków był również nie trafioną decyzją.

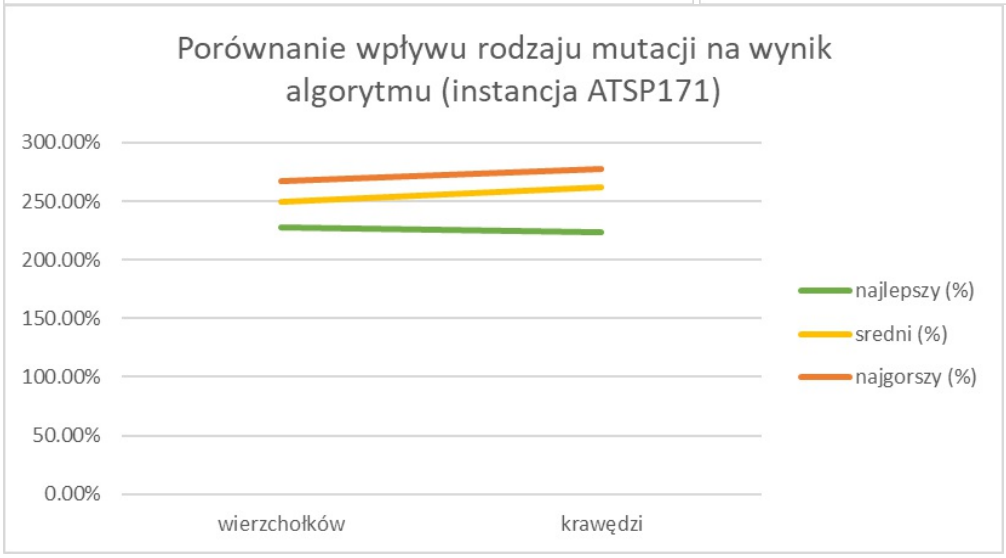
Instancja `ftv170` - ATSP171





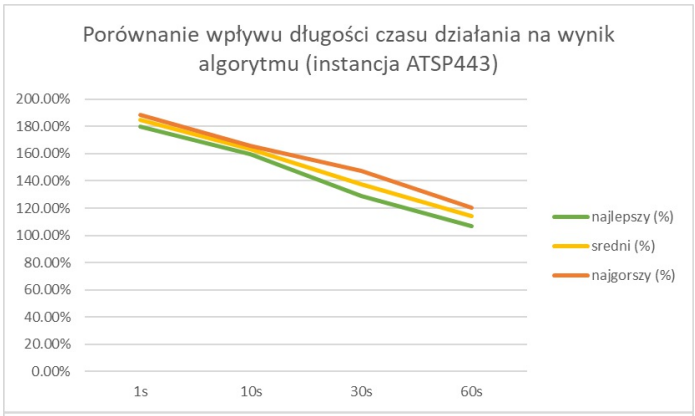


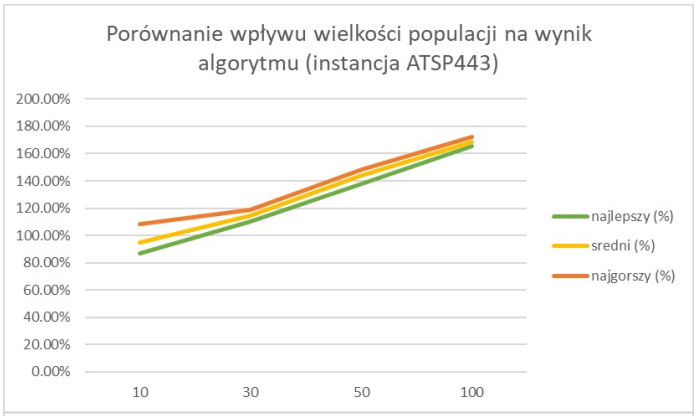


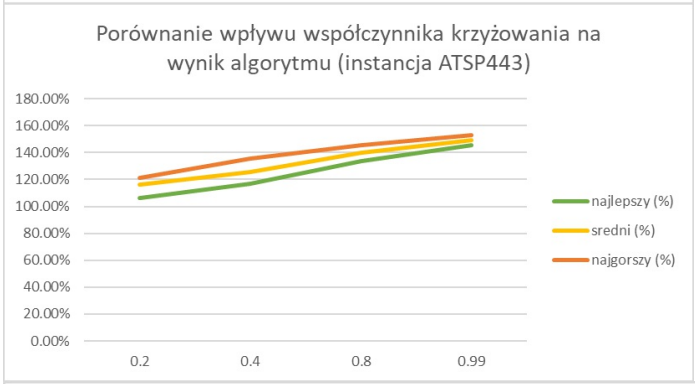


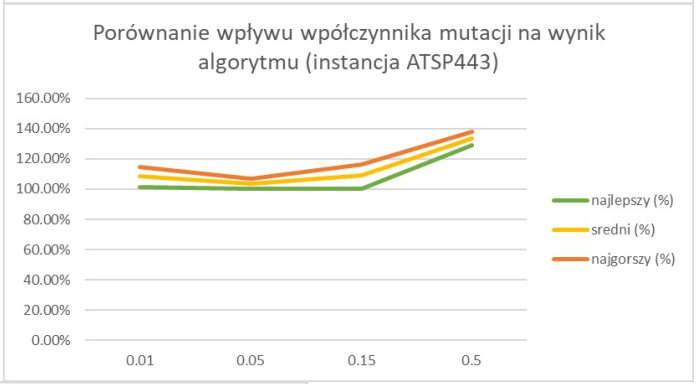
W tym wypadku zauważyć należy znaczący wzrost wielkości instancji, jak i ogromny spadek jakości wyników. W oczy od razu rzuca się również odwrócenie trendu przy którym zwiększenie ilości osobników, czy prawdopodobieństwa krzyżowania (co bezpośrednio zwiększa się na rozmiar nowej populacji przy każdym wywołaniu). Wyjaśnienie tego problemu znajduje się w mojej implementacji metody krzyżowania OX, której złożoność obliczeniowa drastycznie wzrasta przy większej liczbie miast.

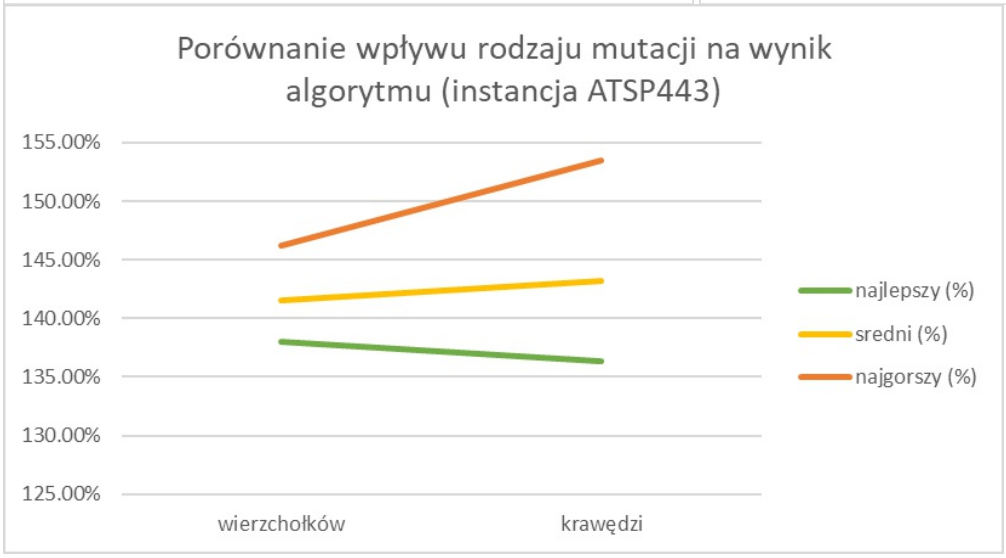
Instancja `rbg443` - ATSP443





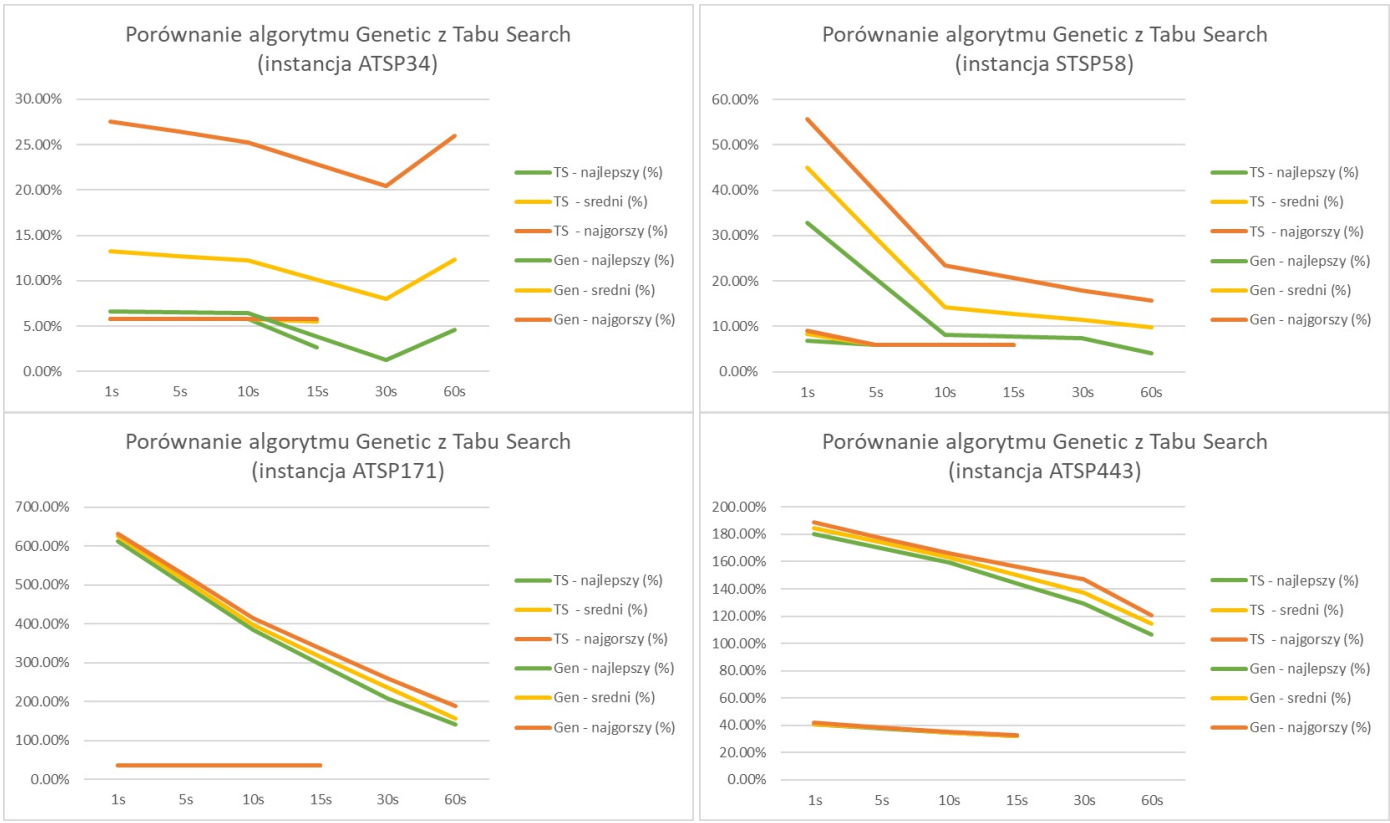






Ta instancja również zachowuje zauważone trendy, gdzie się da oraz bardzo dobrze pokazuje problem, który został zauważony w poprzedniej instancji.

Genetic vs Tabu Search



Na etapie planowania testów porównanie tych dwóch algorytmów wydawało się być sensowne, jednak po odkryciu wspomnianego błędu w implementacji zdaje się bezcelowe. Bez problemu widać, że algorytm Tabu Search znajduje lepsze rozwiązania, a genetyczny tylko dla na prawdę małych instancji zdaje się być w stanie konkurować z nim. Dodatkowo bardzo ładnie widać o wiele większą losowość w algorytmie genetycznym.

Wnioski

- Problem z optymalizacją algorytmu krzyżowania w znaczący sposób szkodzi całemu algorytmowi. Skalę zjawiska reprezentuje umieszczone w moim kodzie zliczanie sprawdzonych rozwiązań, które dla Tabu Search przy 30s zwraca 50 milionów, podczas gdy Genetic sprawdza jedynie 150 tysięcy.
- Wybranie populacji startowej w mniej losowy sposób, mogłoby dać lepsze efekty algorytmu.
- Bardziej odpowiedne wybranie domyślnych parametrów (zwiększenie szans na mutację i zmiana rodzaju mutacji) powinno przynieść poprawę działania całego algorytmu.
- Algorytm genetyczny jest o wiele bardziej losowy, co pokazuje różnica pomiędzy najlepszym, a najsłabszym wynikiem w próbie, która jest kilkakrotnie większa niż w przypadku Tabu Search.
- Algorytm genetyczny powinien lepiej radzić sobie z lokalnymi minimum, w których moja implementacja Tabu Search utykała.