Politechnika Wrocławska

Projektowanie efektywnych algorytmów

Projekt nr 2:
Implementacja i analiza efektywności algorytmu opartego na metodzie Tabu Search dla problemu komiwojażera

Autor:
Bartosz Pogoda 225988

Prowadzący:
dr inż. Jarosław Mierzwa

18 grudnia 2017

Spis treści

1 Wstęp teoretyczny **2**
- 1.1 Opis algorytmu . . . 2
- 1.2 Przeszukiwanie lokalne, sąsiedztwo . . . 2
- 1.3 Przeszukiwanie z zakazami . . . 2
  - 1.3.1 Lista Tabu . . . 2
  - 1.3.2 Dywersyfikacja . . . 2

2 Implementacja algorytmu, opis istotnych klas **3**
- 2.1 Instancja . . . 3
- 2.2 Instancja - odczyt z pliku . . . 3
- 2.3 Ścieżka - rozwiązanie . . . 4
- 2.4 Ograniczenie czasowe algorytmu . . . 4
- 2.5 Dywersyfikacja . . . 5
- 2.6 Zbieranie wyników . . . 5
- 2.7 Lista Tabu . . . 6
- 2.8 Ruchy . . . 6
  - 2.8.1 SwapMove . . . 7
  - 2.8.2 InsertMove . . . 8
  - 2.8.3 InvertMove . . . 9
- 2.9 Generatory sąsiedztwa . . . 10
- 2.10 Algorytm . . . 10
  - 2.10.1 Zależności . . . 10
  - 2.10.2 Działanie . . . 10

3 Wyniki badań **11**
- 3.1 Parametry . . . 11
- 3.2 Instancja - eil101.tsp . . . 12
  - 3.2.1 Tabela zbiorcza . . . 12
  - 3.2.2 Swap Move . . . 12
  - 3.2.3 Insert Move . . . 13
  - 3.2.4 Invert Move . . . 13
- 3.3 Instancja - rbg403.atsp . . . 14
  - 3.3.1 Tabela zbiorcza . . . 14
  - 3.3.2 Swap Move . . . 14
  - 3.3.3 Insert Move . . . 15
  - 3.3.4 Invert Move . . . 15
- 3.4 Instancja - d1291.tsp . . . 16
  - 3.4.1 Tabela zbiorcza . . . 16
  - 3.4.2 Swap Move . . . 16
  - 3.4.3 Insert Move . . . 17
  - 3.4.4 Invert Move . . . 17

4 Podsumowanie, wnioski **18**

1 Wstęp teoretyczny

1.1 Opis algorytmu

Problem komiwojażera definiowany jest dla grafów pełnych. Wierzchołki grafu nazywane są miastami, a krawędziom przypisane są wartości reprezentujące odległości między nimi. Problem polega na odnalezieniu najkrótszej ścieżki, takiej aby odwiedzić każde miasto dokładnie raz i wrócić do punktu wyjścia.
Wiele problemów, występujących w sektorach takich jak transport oraz logistyka można sprowadzić do problemu komiwojażera. Zastosowanie efektywnych algorytmów rozwiązujących ten problem nie rzadko prowadzi do redukcji kosztów i optymalizacji pewnych rzeczywistych procesów. Przykładową aplikacją mogło by być wyznaczenie optymalnej trasy dla autobusu szkolnego, tak aby odwiedził wszystkie przystanki z dziećmi w jak najkrótszym okresie czasu. Wierzchołkami grafu byłyby poszczególne przystanki, a wartościami na krawędziach szacunkowy czas przemieszczenia się autobusu między przystankami. Warto tutaj zauważyć, że czas przemieszczania się z przystanku A do B, może różnić się od czasu przemieszczenia z B do A. Różnice te są uwzględnione w asymetrycznym problemie komiwojażera, który zostanie poddany analizie w tym sprawozdaniu.

1.2 Przeszukiwanie lokalne, sąsiedztwo

Algorytm przeszukiwania lokalnego dla problemu komwijażera polega na poszukiwaniu lepszych rozwiązań zbliżonych do aktualnego rozwiązania. Dla rozwiązania definiuje się sąsiedztwo, czyli zbiór pobliskich rozwiązań. Pobliskich czyli takich, które można osiągnać poprzez wykonanie jednego ruchu na przykład może to być zamiana dwóch pozycji w aktualnej ścieżce (z wyłączeniem miasta startowego). Następnie z sąsiedztwa aktualnego rozwiązania wybiera się rozwiązanie najlepsze (najkrótsza ścieżka) i nastepuje kolejna iteracja algorytmu. Algorytm taki nie daje gwarancji znalezienia rozwiązania optymalnego, a nawet gdy takie znajdzie - nie mamy informacji o tym, że rozwiązanie to jest optymalne i możemy zakończyć poszukiwania.

1.3 Przeszukiwanie z zakazami

Algorytm przeszkukiwania lokalnego ma poważną wadę, która znacznie ogranicza jego zastosowanie. Algorytm bardzo często osiąga minimum lokalne, z którego nie jest w stanie wyjść. Ten problem rozwiązuje zastosowanie metodyki Tabu Search - przeszukiwania z zakazami.

1.3.1 Lista Tabu

Lista Tabu zawiera ruchy, które zostały wykonane w niedalekiej przeszłości. Ruchy te są zakazane, dzięki czemu algorytm otrzymuje mechanikę pozwalającą na wydostanie się z minimum lokalnego, ponieważ do raz osiągniętego minimum nie będzie mógł wrócić przez jakiś czas. Ilość kroków, podczas których ruch jest zakazany określany jest jako kadencja. Parametr ten jest bardzo istotny, jego wybór znacząco wpływa na działanie algorytmu.

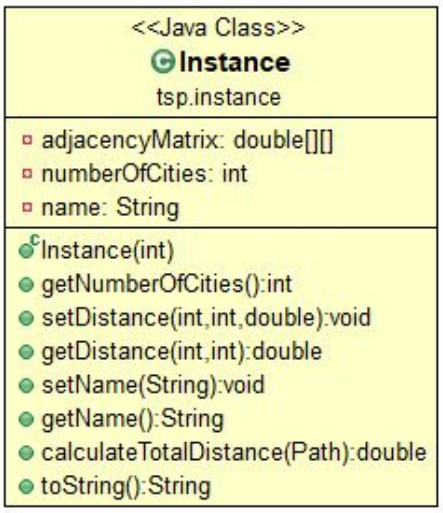
1.3.2 Dywersyfikacja

Dywersyfikacja jest kolejnym mechanizmem unikania minimum lokalnego. Polega ona na wybraniu rozwiązania potencjalnie odległego od aktualnego rozwiązania (np. rozwiązania losowego) w przypadku braku poprawy wyniku przez określony okres działania algorytmu.

2 Implementacja algorytmu, opis istotnych klas

2.1 Instancja

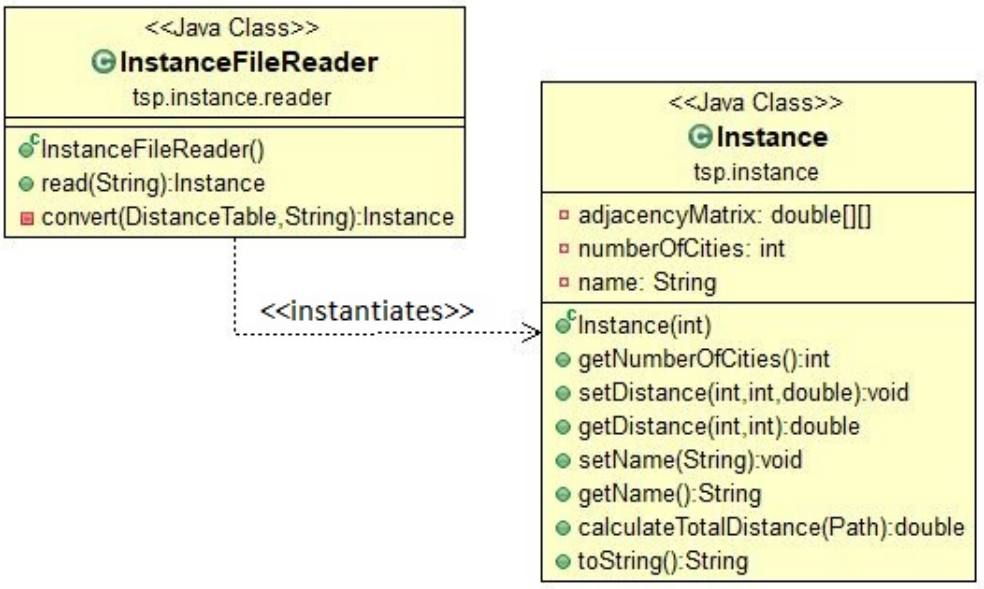
Klasa reprezentująca instancję problemu. Przechowuje informacje o krawędziach w postaci macierzy sąsiedztwa. Została również utworzona funkcja pozwalająca na obliczenie długości podanej ścieżki w obrębie instancji - calculateTotalDistance.



Rysunek 1: Deklaracja klasy Instance

2.2 Instancja - odczyt z pliku

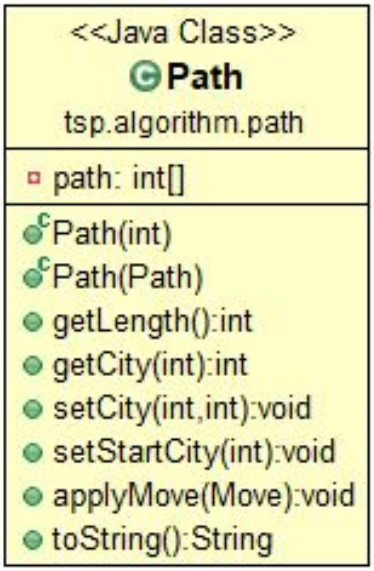
Klasa usługowa tworząca instancje problemu (typ Instance) na podstawie plików w formatach TSPLIB95 (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/). Klasa obudowuje bibliotekę jorlib, która została wykorzystana w celu odczytu wszystkich formatów plików pozyskanych z wyżej wymienionej strony. Wewnątrz metody read dosyć skomplikowany obiekt typu TSPLibInstance, otrzymany po przetworzeniu pliku przez bibliotekę jorlib, jest parsowany do obiektu typu Instance, zawierającego jedynie dane ważne z punktu widzenia projektu.



Rysunek 2: Deklaracja klasy InstanceFileReader

2.3 Ścieżka - rozwiązanie

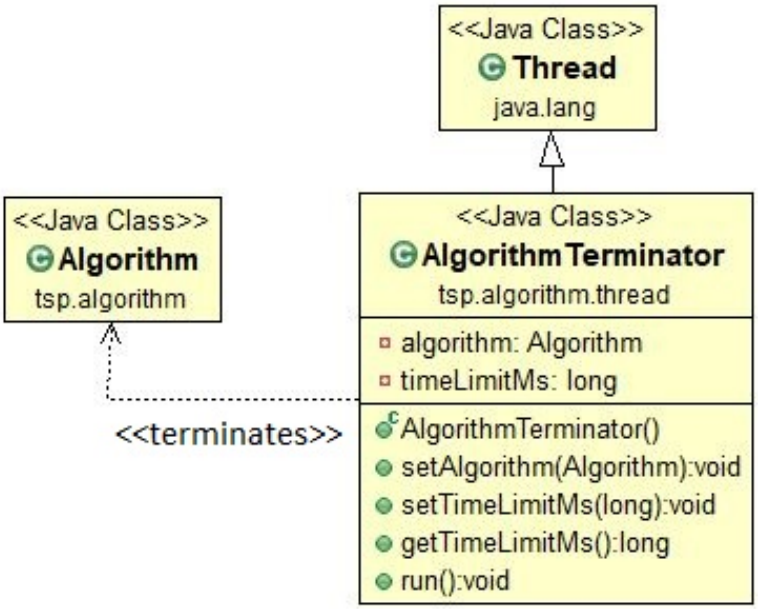
Klasa Path reprezentuje rozwiązanie algorytmu. Ścieżka jest w postaci [0, 1, 2, 3, 4, 0]. Pierwszy i ostatni element jest ustawiany za pomocą funkcji setStartCity, nie powinien on być zmieniany za pomocą funkcji setCity. Ścieżka posiada długość o jeden większą od rozmiaru instancji.



Rysunek 3: Deklaracja klasy Path

2.4 Ograniczenie czasowe algorytmu

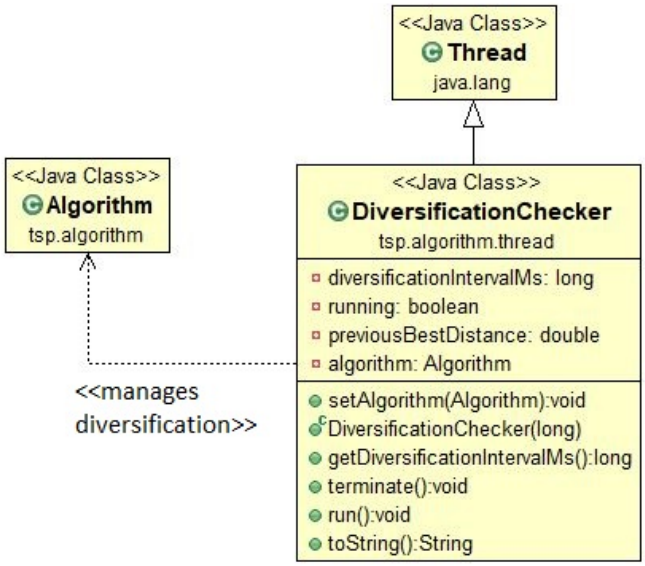
W celu umożliwienia zatrzymania algorytmu po zadanym czasie został zdefiniowany wątek AlgorithmTerminator.



Rysunek 4: Deklaracja klasy AlgorithmTerminator

2.5 Dywersyfikacja

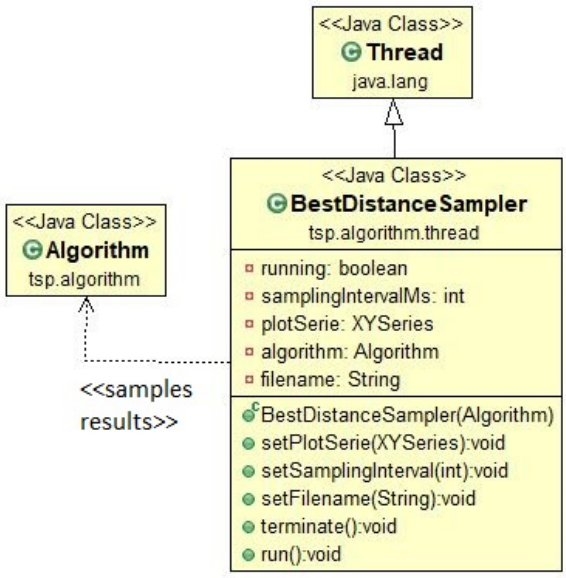
W ramach realizacji dywersyfikacji w projekcie został utworzony watek DiversificationChecker, który co ustalony czas sprawdza, czy wynik w aktualnym przedziale dywersyfikacji poprawił się (od czasu ostatniego sprawdzenia). W przypadku braku poprawy wyniku, wątek ustawia flagę dywersyfikacji na obiekcie Algorytmu, który w najbliższej iteracji przeprowadzi dywersyfikację.



Rysunek 5: Deklaracja klasy DiversificationChecker

2.6 Zbieranie wyników

Do zbierania danych został utworzony wątek, który co ustalony interwał (domyślnie- sekunda) sprawdza aktualne najlepsze znalezione rozwiązanie przez Algorytm, a następnie zapisuje je do pliku oraz, jeśli ustawiono, dodaje punkt na wyświetlanym wykresie.

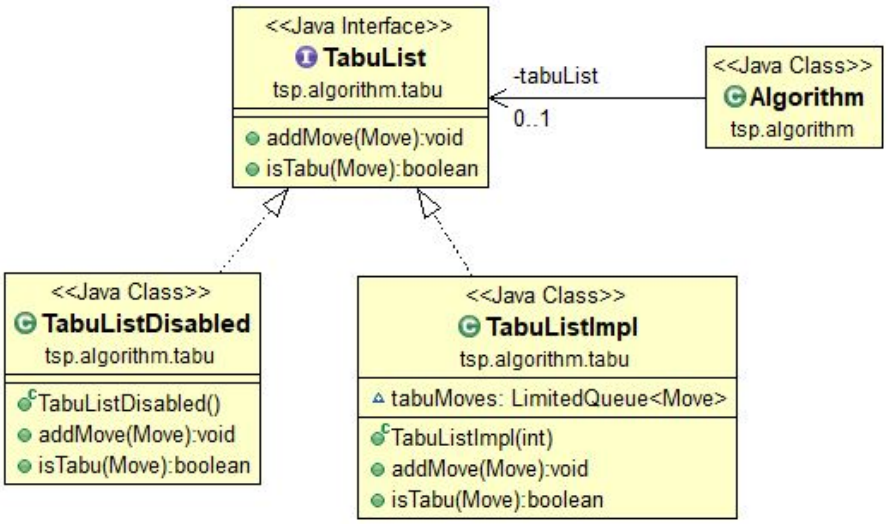


Rysunek 6: Deklaracja klasy BestDistanceSampler

2.7 Lista Tabu

Został zdefiniowany interfejs definiujący operacje, które można wynonać na liście tabu, a więc, dodanie ruchu oraz sprawdzenie czy dany ruch jest tabu. Taka struktura pozwala na dynamiczną podmianę używanej implementacji listy zakazów. W ramach projektu zdefiniowano dwie implementacje:

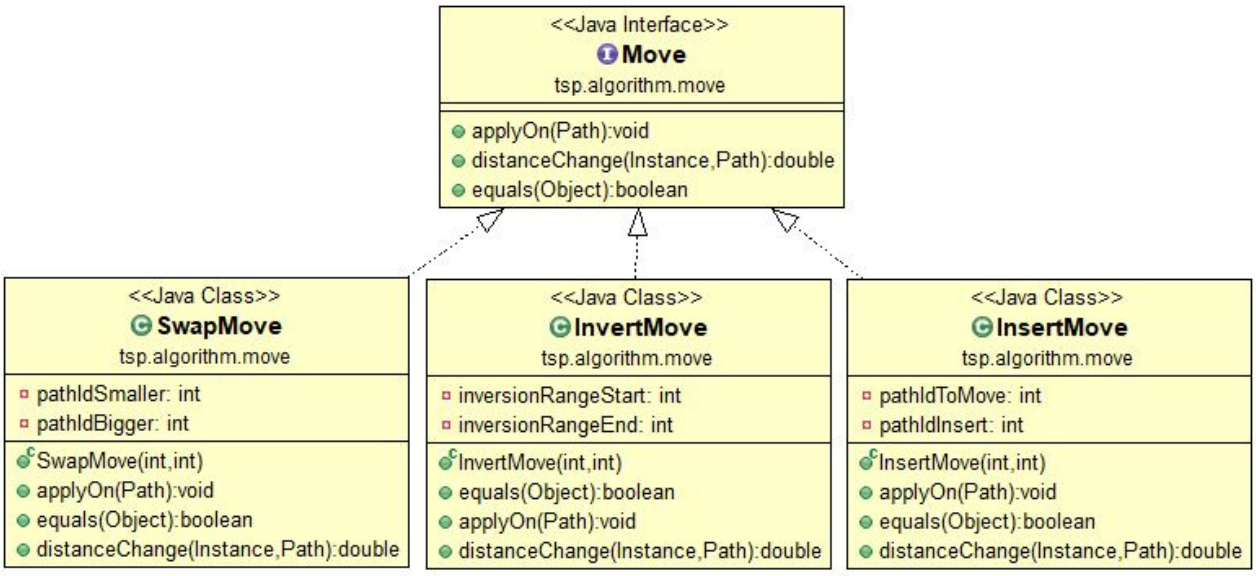
- TabuListDisabled - implementacja, która przy dodaniu ruchu do listy nie robi nic, a sprawdzenie czy ruch jest tabu zawsze zwraca odpowiedź "fałsz". Ta implementacja została zdefiniowana w celu umożliwienia wyłączenia listy tabu - algorytm staje się tradycyjnym przeszukiwaniem lokalnym.
- TabuListImpl - kolejka o ograniczonej wielkości



Rysunek 7: Interfejs Tabu List oraz jego implementacje

2.8 Ruchy

W projekcie zostały zdefiniowane trzy rodzaje ruchów (zmian aktualnego rozwiązania). Zostaną one opisane w kolejnych podpunktach.



Rysunek 8: Interfejs Move oraz jego implementacje

2.8.1 SwapMove

Ruch polega na zamianie miejscami dwóch miast w ścieżce (zadanych jako ich pozycje).

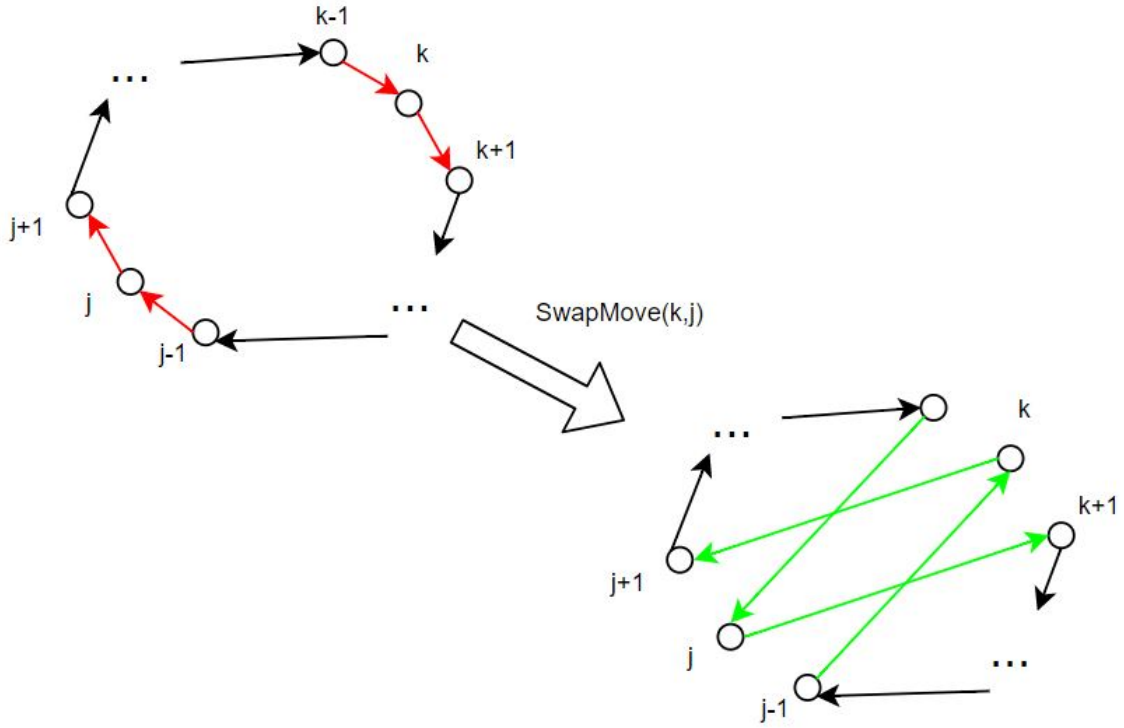
Parametry:

- pathIdSmaller - id pierwszego z elementów ścieżki do wymiany (mniejsze)
- pathIdBigger - id drugiego z elementów ścieżki do wymiany (większe)

Dziedzina:

- pathIdSmaller > 0, pathIdSmaller < path.size() - 2
- pathIdBigger > pathIdSmaller, pathIdBigger < path.size() - 1

Przykład:



Rysunek 9: Przykład ruchu SwapMove (źródło własne)

2.8.2 InsertMove

Ruch polega na wstawieniu miasta będącego na zadanej pozycji w inną zadaną pozycję w ścieżce.

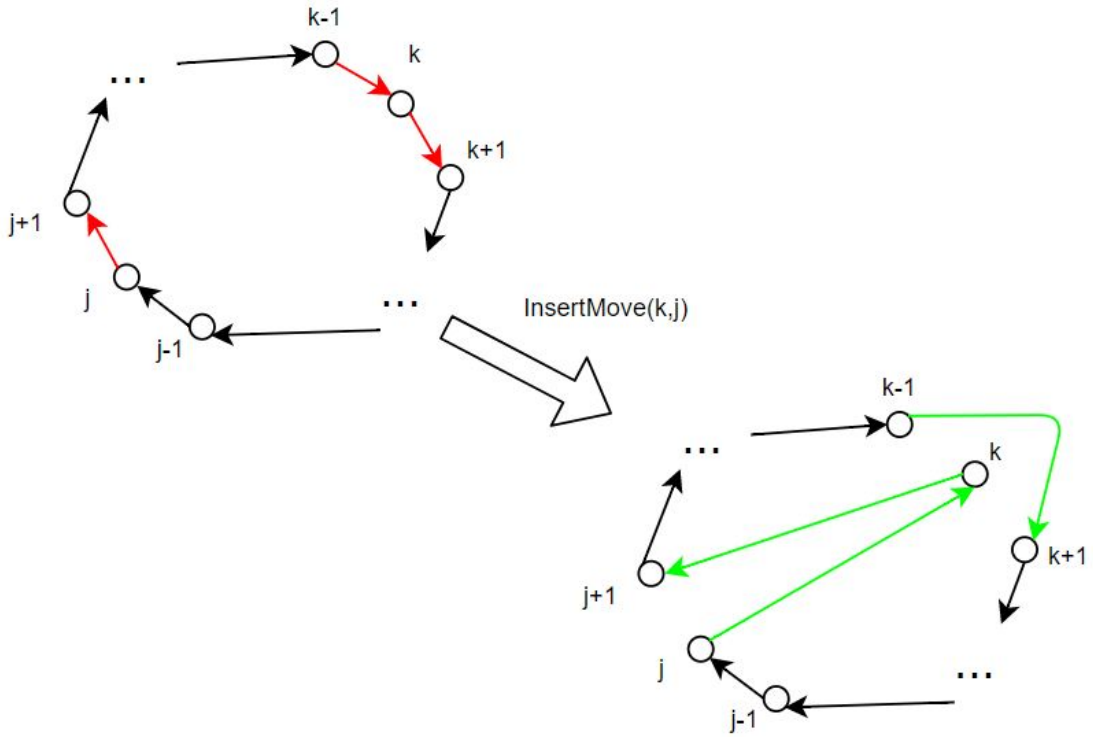
Parametry:

- pathIdToMove - id elementu ścieżki do przeniesienia
- pathIdInsert - id elementu ścieżki, w które ma zostać wstawiony element przenoszony

Dziedzina:

- pathIdToMove > 0, pathIdInsert > 0, pathIdToMove != pathIdInsert

Przykład:



Rysunek 10: Przykład ruchu InsertMove (źródło własne)

2.8.3 InvertMove

Ruch polega na odwróceniu kolejności w podścieżce (sekwencja miast) zawartej w ścieżce.

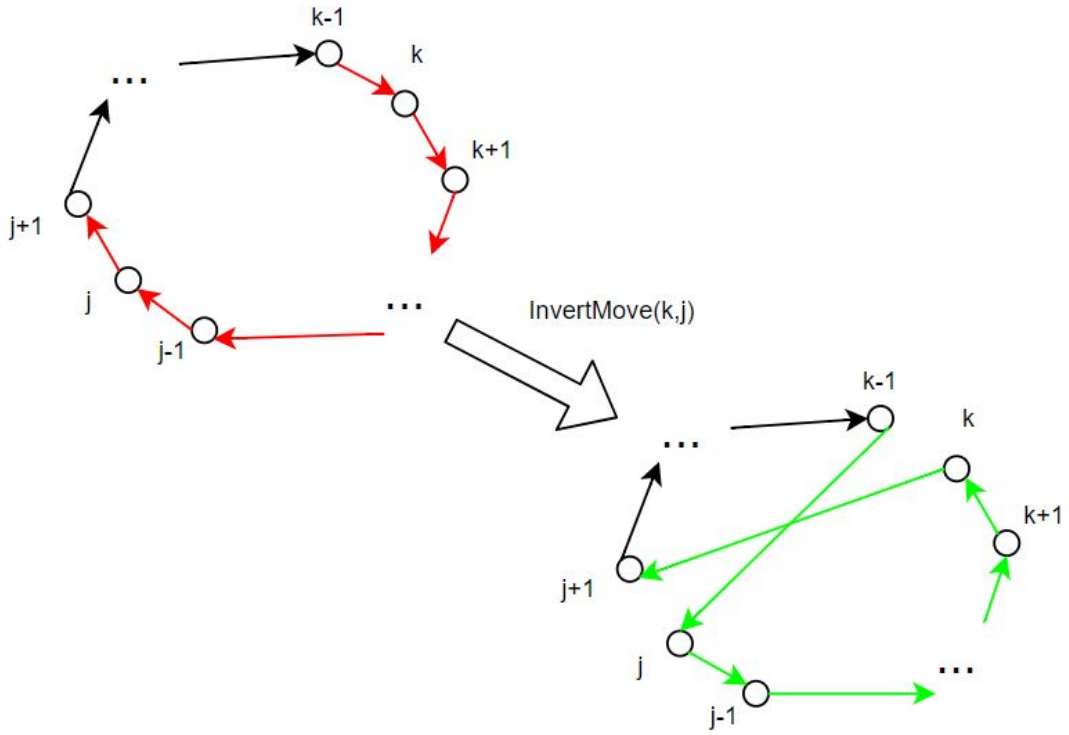
Parametry:

- inversionRangeStart - id elementu będącego początkiem podścieżki
- inversionRangeEnd - id elementu będącego końcem podścieżki

Dziedzina:

- inversionRangeStart > 0, inversionRangeStart < path.size() - 2
- inversionRangeEnd > inversionRangeStart, inversionRangeEnd < path.size() - 1

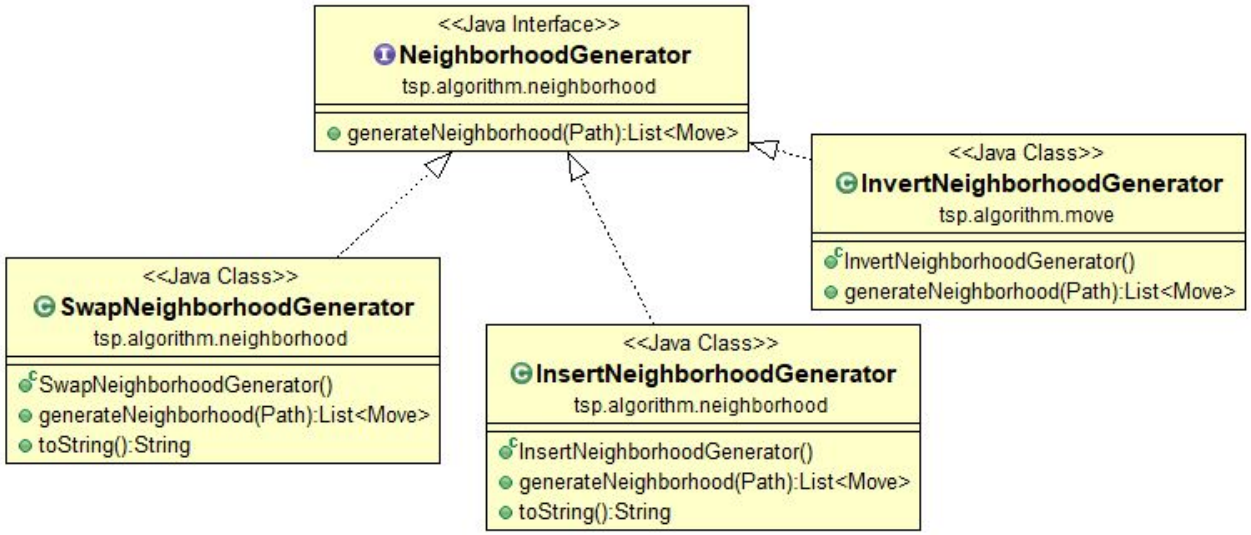
Przykład:



Rysunek 11: Przykład ruchu InvertMove (źródło własne)

2.9 Generatory sąsiedztwa

Dla każdego z wyżej opisanych ruchów został utworzony generator sąsiedztwa, który zajmuje się generowaniem listy możliwych ruchów dla danej ścieżki (z uwzględnieniem dziedziny konkretnego ruchu).



Rysunek 12: Interfejs NeighborhoodGenerator oraz jego implementacje

2.10 Algorytm

2.10.1 Zależności

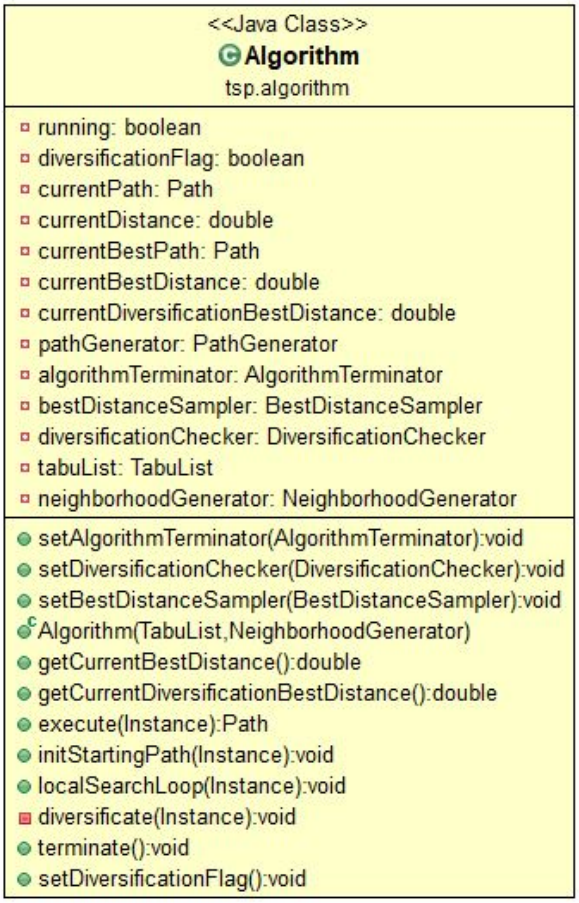
Zależności algorytmu (wyżej opisane klasy) muszą zostać ustawione przy konstrukcji instancji klasy Algorithm.

2.10.2 Działanie

W pierwszej kolejności algorytm uruchamia wątki, do których referencje zostały ustawione przy jego konkstrukcji. Uruchamiany jest wątek typu AlgorithmTerminator, który zatrzyma algorytm po ustalonym czasie, wątek BestDistanceSampler służący do próbkowania wyników oraz, jeśli ustawiono, wątek zarządzający dywersyfikacją.

Następnie zostaje wygenerowane rozwiązanie za pomocą metody zachłannej. W sąsiedztwie tak otrzymanego rozwiązania poszukiwane są co raz to lepsze rozwiązania. Za pomocą generatora możliwych ruchów w kazdej iteracji generowane są wszystkie możliwe ruchy, a następnie spośród tych, które nie są w danym momencie zakazane, wybiera się rozwiązanie które najlepiej wpłynie na wynik. Ruch, który został wykonany w celu uzyskania danego rozwiązania staje się ruchem zakazanym.

W przypadku wykrycia flagi dywersyfikacji, generowane jest losowe rozwiązanie. W przypadku zakończenia pracy przez wątek AlgorithmTerminator, najlepszy uzyskany do tej pory wynik jest zwracany.



Rysunek 13: Metody i atrybuty klasy Algorithm

3 Wyniki badań

Po zaimplementowaniu algorytmu oraz klas pomocniczych zostały przeprowadzone testy jakościowe. Testy zostały przeprowadzone na trzech instancjach o różnej wielkości. Dla każdej z instancji zbadano wyniki otrzymane w przeciągu 6 minut dla różnych rodzajów sąsiedztwa, z zastosowaniem dywersyfikacji oraz bez.

3.1 Parametry

- Kadencja listy zakazów została ustawiona na trzykrotność wielkości problemu (3*N).
- Ograniczenie czasowe algorytmu zostało ustawione na 6 minut.
- Interwał sprwadzenia dywersyfikacji został ustawiony na 30 sekund.

3.2 Instancja - eil101.tsp

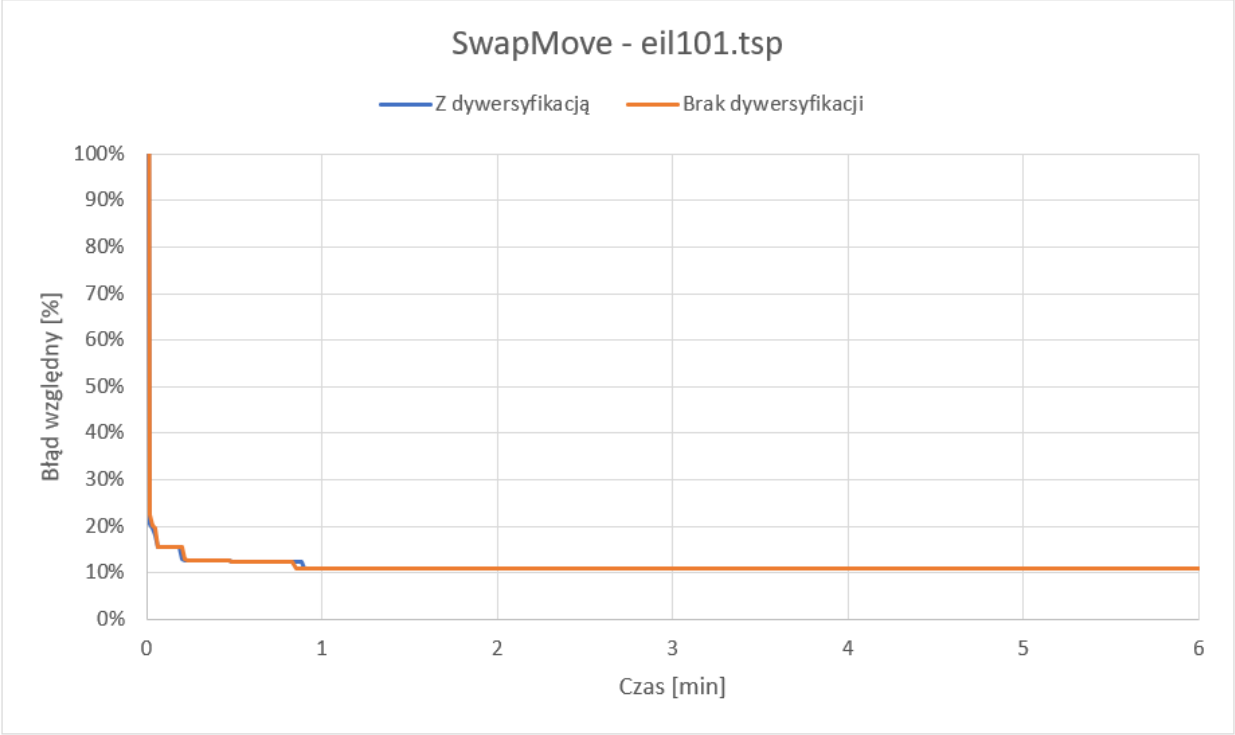
Optymalna długość ścieżki: 629

3.2.1 Tabela zbiorcza

Rodzaj ruchu		SwapMove		InsertMove		InvertMove	
Dywersyfikacja		Tak	Nie	Tak	Nie	Tak	Nie
Czas [min]	0	369.95%	369.95%	369.95%	369.95%	369.95%	369.95%
	0.5	12.56%	12.40%	14.79%	14.79%	0.00%	0.00%
	1	10.81%	10.81%	14.79%	14.79%	0.00%	0.00%
	1.5	10.81%	10.81%	10.81%	14.79%	0.00%	0.00%
	2	10.81%	10.81%	10.81%	14.79%	0.00%	0.00%
	2.5	10.81%	10.81%	10.81%	14.79%	0.00%	0.00%
	3	10.81%	10.81%	9.38%	14.79%	0.00%	0.00%
	3.5	10.81%	10.81%	9.38%	14.79%	0.00%	0.00%
	4	10.81%	10.81%	9.38%	14.79%	0.00%	0.00%
	4.5	10.81%	10.81%	9.38%	14.79%	0.00%	0.00%
	5	10.81%	10.81%	6.68%	14.79%	0.00%	0.00%
	5.5	10.81%	10.81%	6.20%	14.79%	0.00%	0.00%
	6	10.81%	10.81%	6.20%	14.79%	0.00%	0.00%

Rysunek 14: Tabela zbiorcza dla instancji eil101.tsp

3.2.2 Swap Move

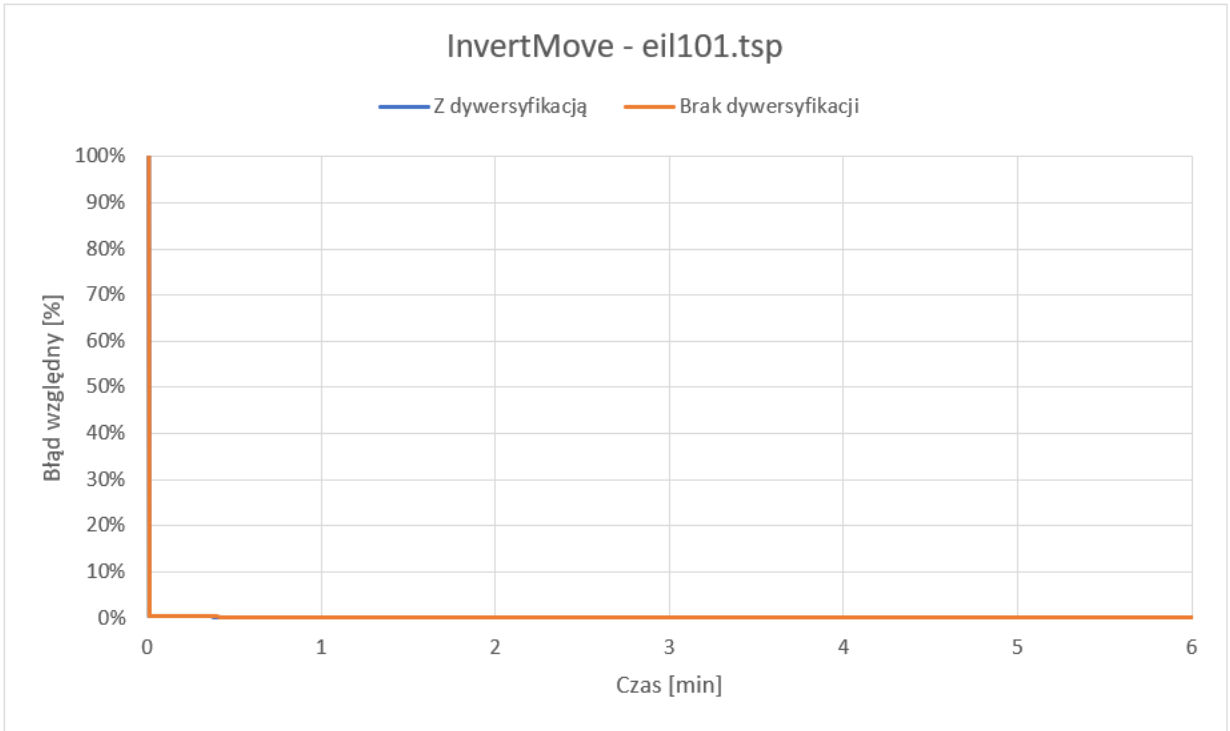


Rysunek 15: Zależnośc błędu względnego od czasu dla SwapMove (eil101.tsp)

3.2.3 Insert Move

Rysunek 16: Zależnośc błędu względnego od czasu dla InsertMove (eil101.tsp)

3.2.4 Invert Move



Rysunek 17: Zależnośc błędu względnego od czasu dla InvertMove (eil101.tsp)

3.3 Instancja - rbg403.atsp

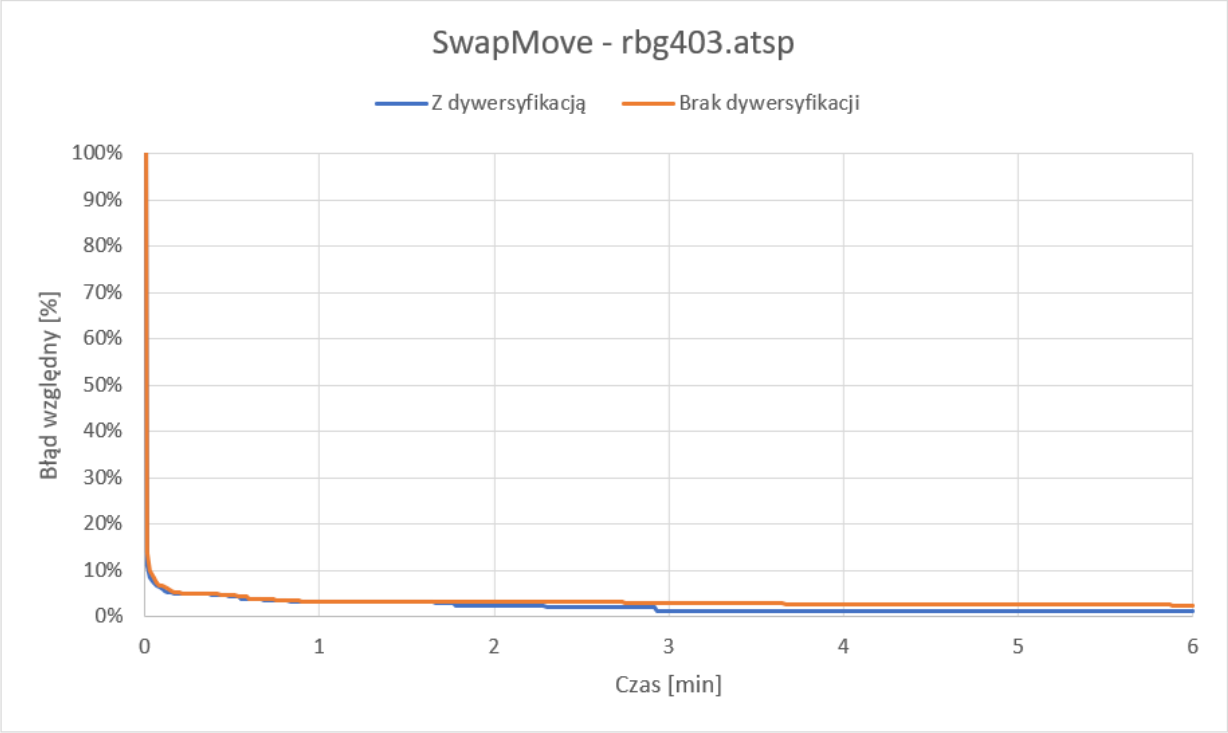
Optymalna długość ścieżki: 629

3.3.1 Tabela zbiorcza

Rodzaj ruchu		SwapMove		InsertMove		InvertMove	
Dywersyfikacja		Tak	Nie	Tak	Nie	Tak	Nie
Czas [min]	0	369.95%	369.95%	369.95%	369.95%	369.95%	369.95%
	0.5	12.56%	12.40%	14.79%	14.79%	0.00%	0.00%
	1	10.81%	10.81%	14.79%	14.79%	0.00%	0.00%
	1.5	10.81%	10.81%	10.81%	14.79%	0.00%	0.00%
	2	10.81%	10.81%	10.81%	14.79%	0.00%	0.00%
	2.5	10.81%	10.81%	10.81%	14.79%	0.00%	0.00%
	3	10.81%	10.81%	9.38%	14.79%	0.00%	0.00%
	3.5	10.81%	10.81%	9.38%	14.79%	0.00%	0.00%
	4	10.81%	10.81%	9.38%	14.79%	0.00%	0.00%
	4.5	10.81%	10.81%	9.38%	14.79%	0.00%	0.00%
	5	10.81%	10.81%	6.68%	14.79%	0.00%	0.00%
	5.5	10.81%	10.81%	6.20%	14.79%	0.00%	0.00%
	6	10.81%	10.81%	6.20%	14.79%	0.00%	0.00%

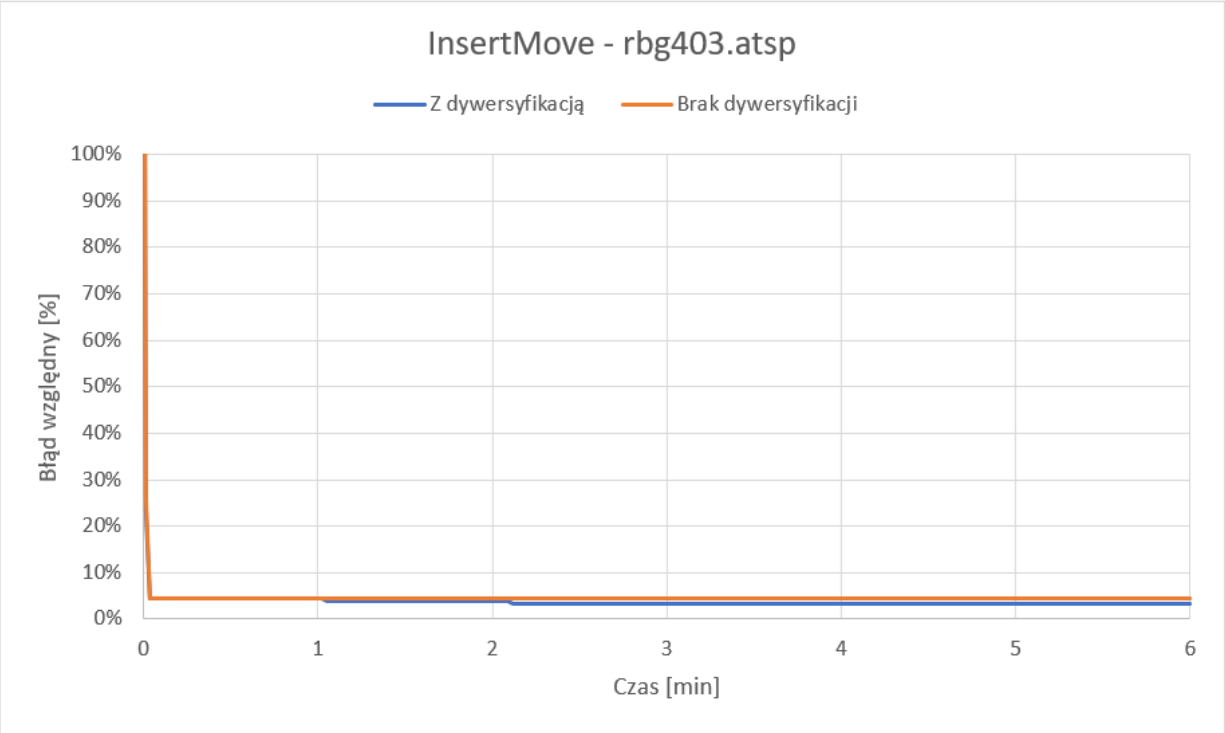
Rysunek 18: Tabela zbiorcza dla instancji rbg403.atsp

3.3.2 Swap Move



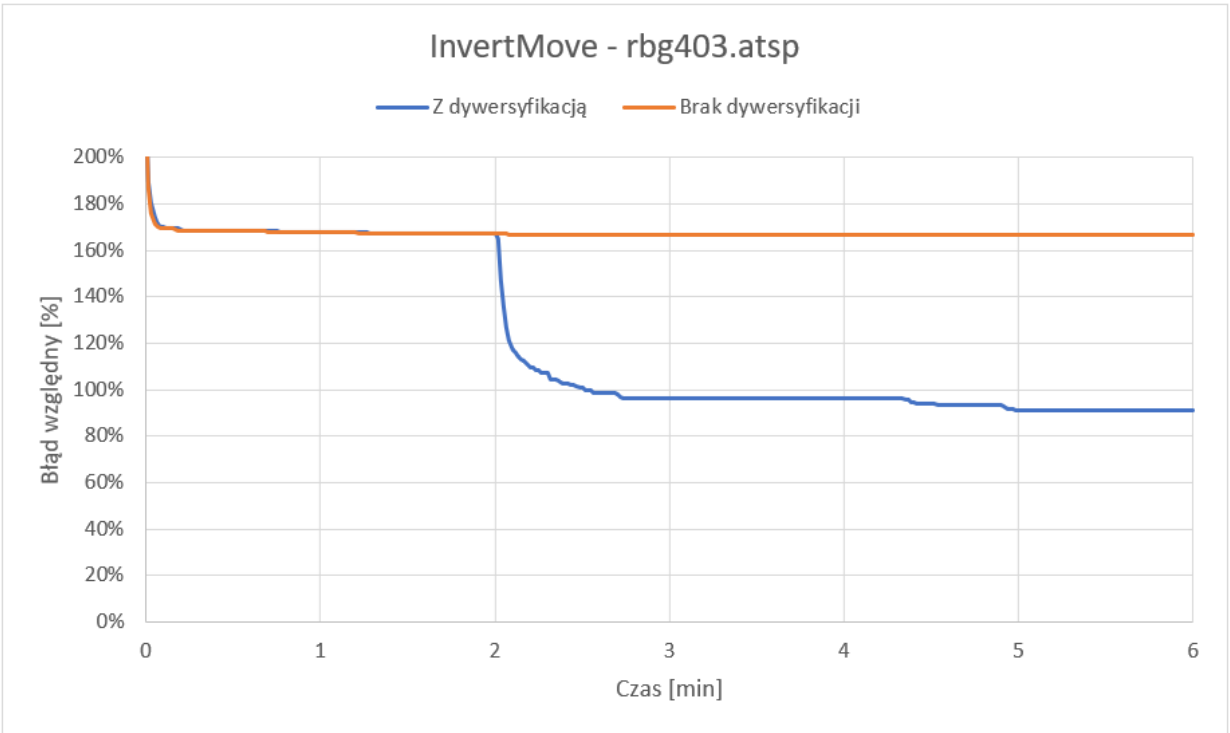
Rysunek 19: Zależnośc błędu względnego od czasu dla SwapMove (rbg403.atsp)

3.3.3 Insert Move



Rysunek 20: Zależnośc błędu względnego od czasu dla InsertMove (rbg403.atsp)

3.3.4 Invert Move



Rysunek 21: Zależnośc błędu względnego od czasu dla InvertMove (rbg403.atsp)

3.4 Instancja - d1291.tsp

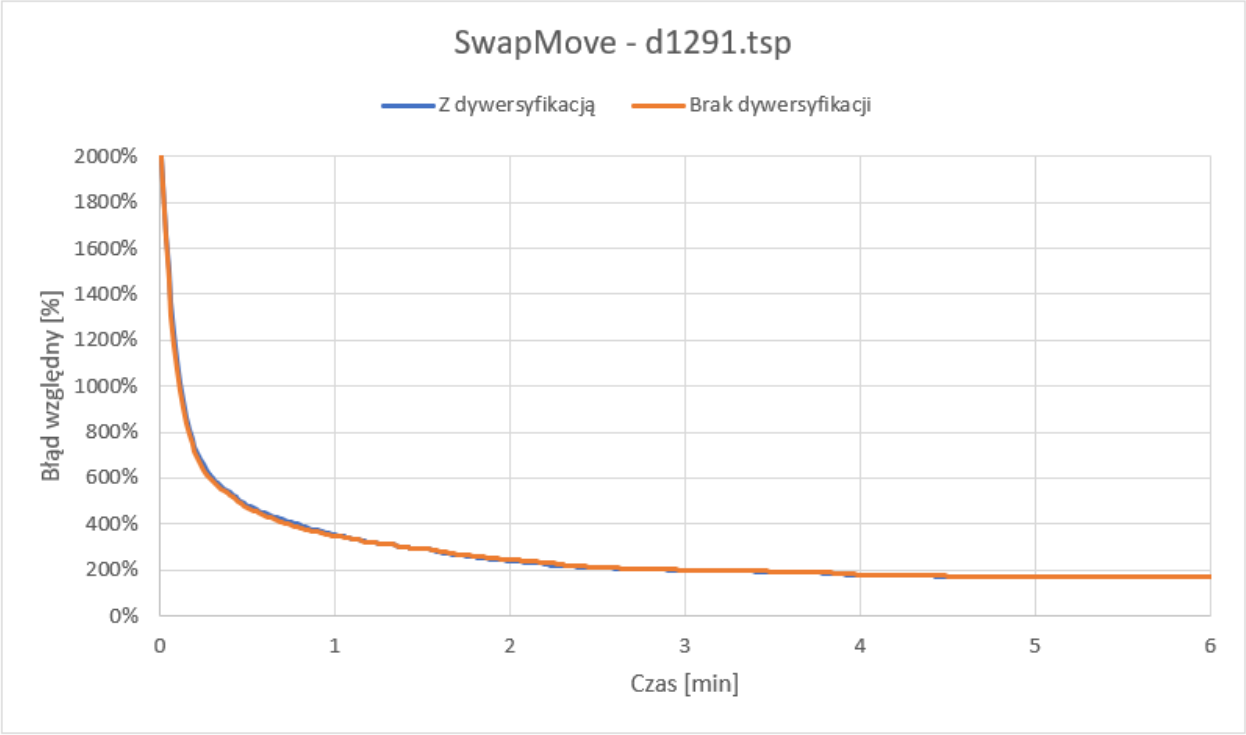
Optymalna długość ścieżki: 50801

3.4.1 Tabela zbiorcza

Rodzaj ruchu		SwapMove		InsertMove		InvertMove	
Dywersyfikacja		Tak	Nie	Tak	Nie	Tak	Nie
Czas [min]	0	2117%	2117%	2117%	2117%	2117%	2117%
	0.5	479%	471%	917%	993%	2010%	1992%
	1	352%	349%	606%	625%	1929%	1886%
	1.5	292%	293%	385%	432%	1811%	1772%
	2	240%	244%	254%	301%	1704%	1675%
	2.5	211%	214%	186%	212%	1604%	1584%
	3	198%	200%	135%	160%	1512%	1493%
	3.5	193%	193%	129%	134%	1420%	1408%
	4	180%	180%	128%	128%	1333%	1328%
	4.5	174%	174%	127%	128%	1263%	1247%
	5	173%	173%	127%	127%	1185%	1175%
	5.5	172%	172%	116%	127%	1116%	1106%
	6	171%	171%	115%	127%	1080%	1089%

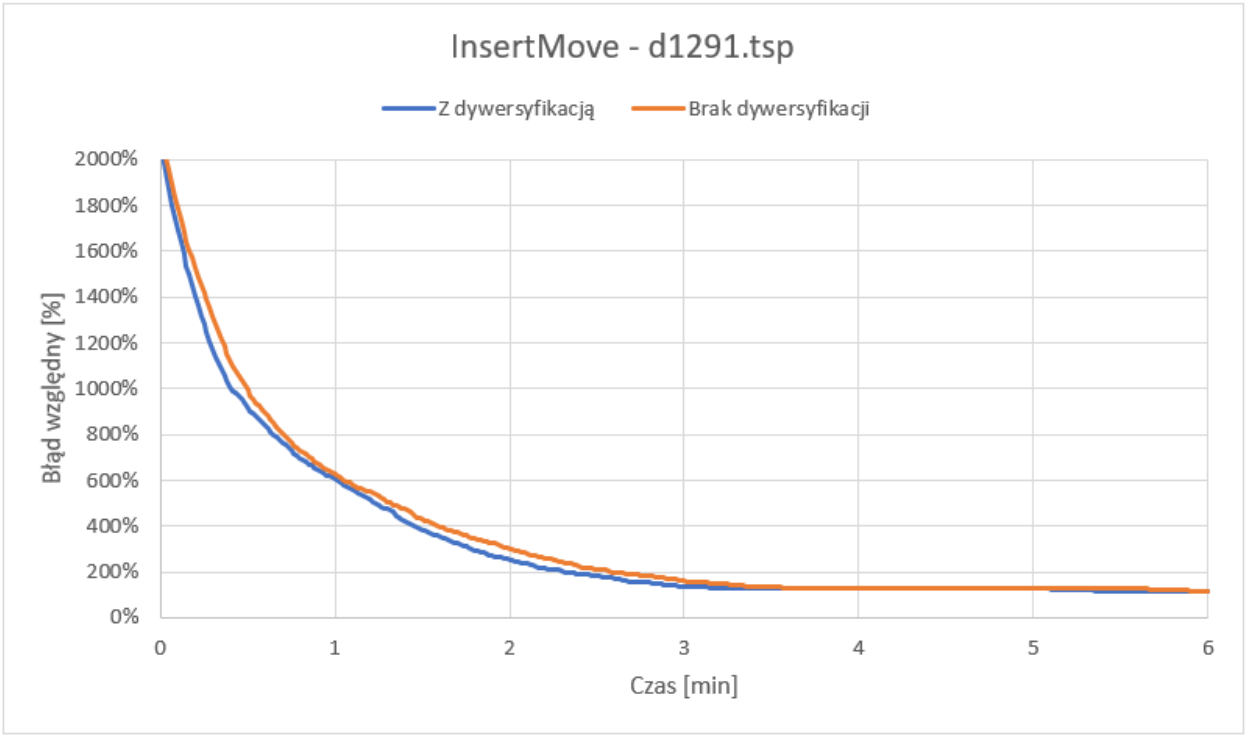
Rysunek 22: Tabela zbiorcza dla instancji d1291.tsp

3.4.2 Swap Move



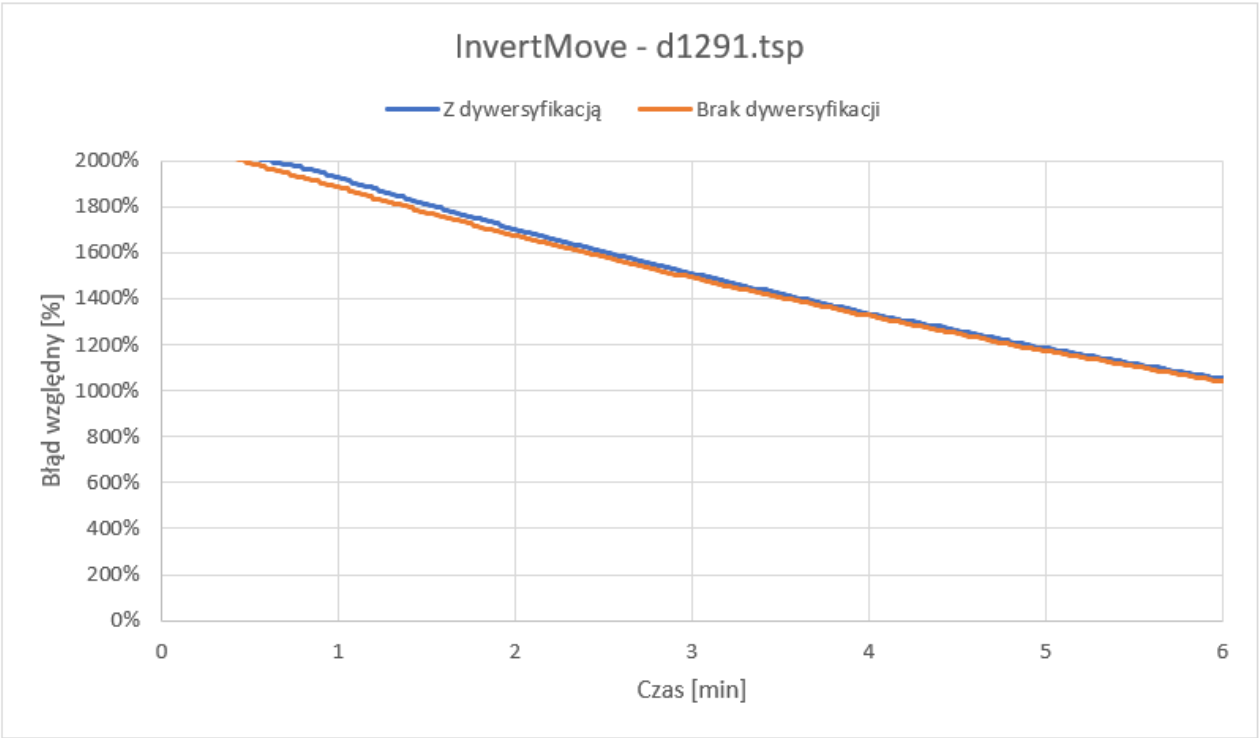
Rysunek 23: Zależnośc błędu względnego od czasu dla SwapMove (d1291.tsp)

3.4.3 Insert Move



Rysunek 24: Zależnośc błędu względnego od czasu dla InsertMove (d1291.tsp)

3.4.4 Invert Move



Rysunek 25: Zależnośc błędu względnego od czasu dla InvertMove (d1291.tsp)

4 Podsumowanie, wnioski

Zaimplementowany algorytm oparty na metodyce przeszukiwania z zakazami skutecznie radził sobie z znajdywaniem rozwiązań bliskich optymalnemu w krótkim czasie, mimo dużych rozmiarów instancji.

- Dla instancji eil101.tsp oraz sąsiedztwa InvertMove udało się znaleźć rozwiązanie, które zostało uznane za optymalne na podstawie danych zamieszczonych na stronie TSPLIB95.
- Dla instancji o większym rozmiarze sąsiedztwo InvertMove okazało się być najmniej skuteczne. Być może jakość rozwiązań zwiększyłoby ustawienie mniejszej wartości kadencji.
- Dla instancji o rozmiarze 1291 wyniki z użyciem dywersyfikacji nie różnią się znacząco od wyników bez jej użycia. Wynika to z krótkiego czasu badania, podczas którego nie udało się osiągnąć minimum lokalnego, z którego algorytm nie byłby stanie wyjść.
- Dla pozostałych przypadków mechanizm dywersyfikacji pozytywnie wpłynął na jakość wyników. Znaczną poprawę wyniku przy zastosowaniu dywersyfikacji widać na Rysunku 21.
- Nie można jednoznacznie określić, które sąsiedztwo jest najlepsze. Skuteczność sąsiedztwa zależy od konkretnego problemu. W celu uzyskania najlepszych wyników powinno się dobrać sąsiedztwo oraz kadencję listy zakazów, np. doświadczalnie. Sąsiedztwa można również wymieszać ze sobą, jak również jego wybór pozostawić jako element losowy w iteracji algorytmu.
- Metody aproksymacyjne pozwalają na otrzymywanie (często) zadowalających wynikow nawet dla dużych instancji problemu.