Politechnika Wrocławska

Algorytmy optymalizacji inspirowane naturą – projekt startowy

Antoni Toczyński, 263914

Cel projektu

Celem projektu było zastosowanie metaheurystyk do rozwiązania problemu cVRP (ang. Capacitated Vehicle Routing Problem). Na podstawie metaheurystyk zaimplementowano algorytmy, które następnie, przy równej liczbie iteracji, poddano próbie na tych samych instancjach problemu.

Problem

cVRP to problem minimalizacyjny NP-trudny. Instancja problemu zawiera n miast (klientów) oraz koordynaty tych lokalizacji. Celem jest odnalezienie trasy, która odwiedzi wszystkie miasta dokładnie raz. Problem utrudnia kwestia zasobów. Każde miasto ma zapotrzebowanie na zasoby, a pojazdy na trasie, które je rozwożą mają ograniczoną pojemność. Dostępne są specjalne lokalizacje (magazyny), które nie mają zapotrzebowania. Magazyn musi być początkiem i końcem trasy.

Rozwiązaniem problemu jest zbiór tras z magazynu, przez miasta i z powrotem do magazynu. Koszt rozwiązania to suma odległości między lokalizacjami.

Algorytmy

W ramach rozwiązywania problemu zaimplementowano 4 algorytmy.

Algorytm losowy

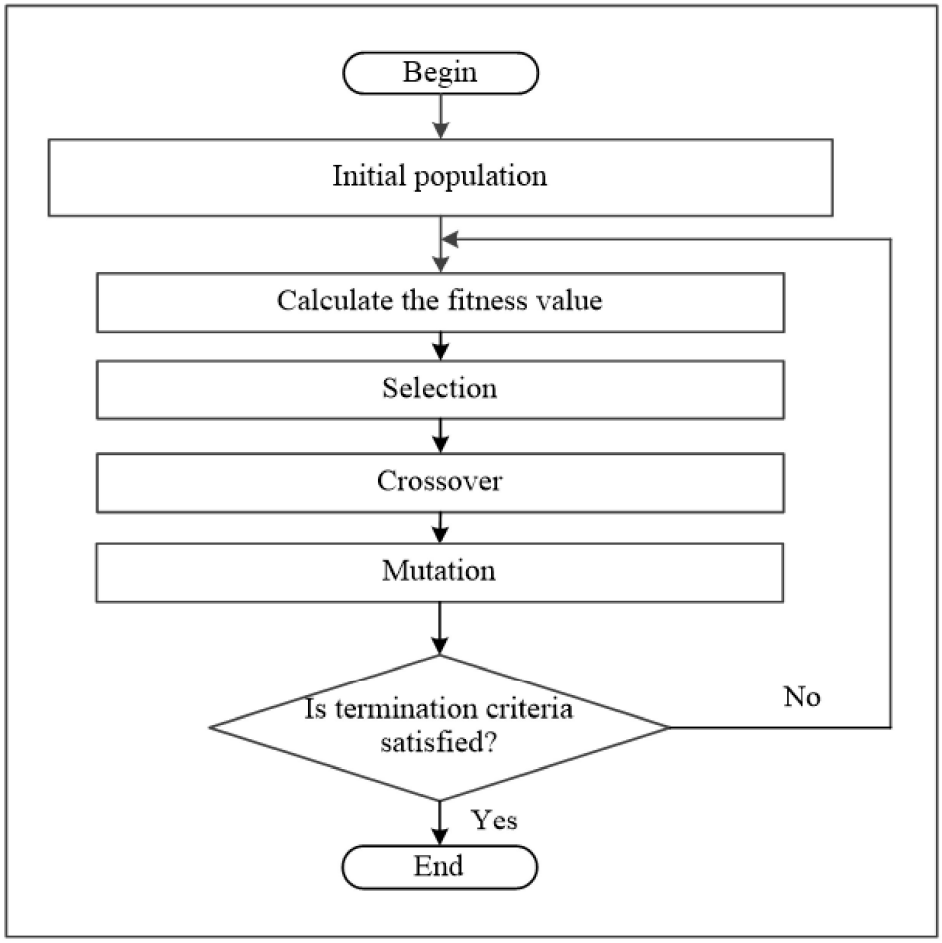
Algorytm ten polega na losowym ułożeniu ciągu miast. Następnie ciąg ten jest dzielony na ścieżki na podstawie zapotrzebowania. Kiedy następne miasto w ciągu doprowadziłoby do przekroczenia pojemności pojazdu, wydzielana jest trasa, a do jej kosztu dolicza się powrót do magazynu. Algorytm uruchamiany jest X razy, a jako wynik przyjmowane jest najlepsze spośród znalezionych rozwiązań.

Algorytm Zachłanny

Algorytm polega na wybieraniu w danym momencie najbliższego miasta, które nie doprowadzi do przekroczenia pojemności pojazdu. W przypadku braku dostępnych miast spełniających ten warunek, do ścieżki dodawany jest koszt powrotu do magazynu i trasa zostaje zakończona. Pozostałe miasta będą dobrane w ścieżki w taki sam sposób. Algorytm jest deterministyczny, co oznacza, że dla tej samej instancji zawsze zwróci identyczne rozwiązanie.

Algorytm Genetyczny

Algorytm został zaimplementowany na podstawie metaheurystyki Algorytmu Genetycznego (GA), którego logika przedstawiona została na Rys 1.



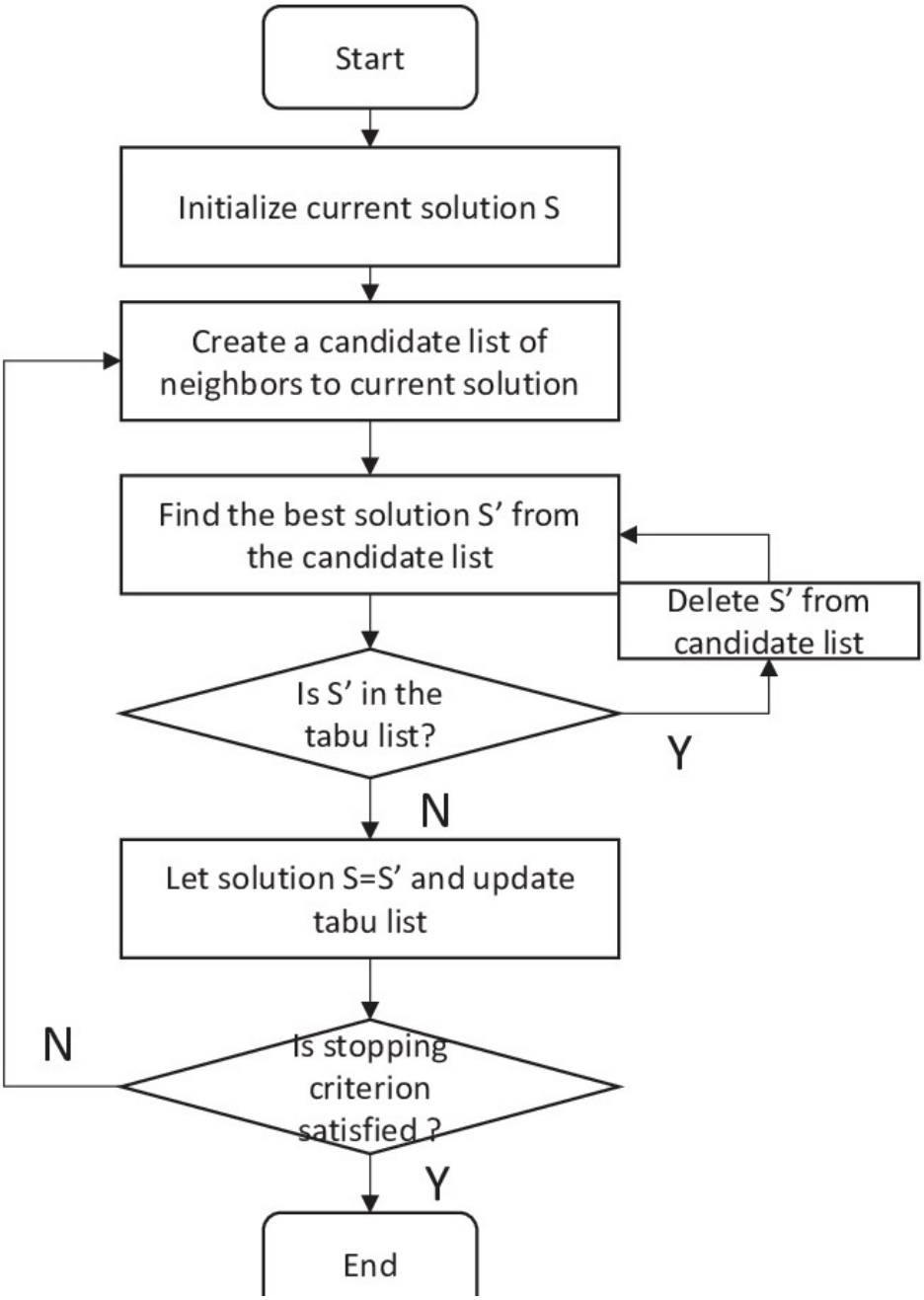
Rysunek 1 https://www.mdpi.com/2073-8994/12/11/1758

W algorytmie operujemy na populacji rozwiązań, z której poprzez kolejne generacje uzyskuje się coraz lepsze osobniki. Główne etapy działania obejmują:

- Selekcję – wybór rodziców z populacji w oparciu o turniej lub ruletkę.
- Krzyżowanie – tworzenie potomków na podstawie dwóch rozwiązań z populacji (rodziców). Potomstwo dziedziczy część cech z jednego rodzica i część z drugiego.
- Mutację – wprowadzenie losowych zmian w populacji. Pozwala rozszerzyć przestrzeń przeszukiwanych rozwiązań.
- Elitaryzm – dopuszczanie najlepszych rozwiązań bezpośrednio do kolejnej generacji, aby mogły dalej uczestniczyć w operacjach krzyżowania i mutacji.

Tabu Search

Algorytm został zaimplementowany na podstawie metaheurystyki Tabu Search, której logika została przedstawiona na Rysunku 2.



Rysunek 2 https://www.researchgate.net/figure/Flowchart-of-tabu-search-algorithm_fig1_320508257

W metodzie tej przestrzeń rozwiązań jest eksplorowana poprzez generowanie rozwiązań sąsiednich względem aktualnego rozwiązania (np. poprzez zamianę kolejności dwóch miast). Spośród sąsiadów wybierane jest najlepsze rozwiązanie, które nie znajduje się na liście tabu.

Lista tabu pełni funkcję pamięci krótkoterminowej i zapobiega powrotowi do wcześniej odwiedzonych rozwiązań (tym samym zmniejszając ryzyko utknięcia w lokalnych minimach). Każdy wykonany ruch,

który prowadzi do nowego rozwiązania, zostaje zapisany w liście tabu. Z czasem starsze ruchy są z niej usuwane, dzięki czemu mogą być ponownie rozważane.

Jeżeli jednak rozwiązanie uzyskane poprzez ruch tabu spełnia kryterium aspiracji (np. jest najlepszym dotychczas znalezionym rozwiązaniem), może ono zostać zaakceptowane pomimo zakazu.

Implementacja

Wszystkie algorytmy zostały zaimplementowane w języku C++. Aby przyspieszyć badania, każdy test działał w osobnym wątku. Do agregacji, przetwarzania i przedstawienia wyników wykorzystano język Python.

Podstawą rozwiązywania instancji problemu cVRP w zaimplementowanym kodzie jest klasa ProblemInstance (Rys. 3). Zawiera ona najważniejsze informacje, takie jak vector<Node>, czyli id klienta, jego współrzędne oraz zapotrzebowanie.

```
21  class ProblemInstance {
22  public:
23      std::string name;
24      std::string comment;
25      std::string type;
26      int dimension = 0;
27      int capacity = 0;
28      std::string edge_weight_type;
29
30      std::vector<utils::Node> nodes;
31      std::vector<int> depots;
32
33      ProblemInstance() = default;
34
35      explicit ProblemInstance(const std::string& filename) {
36          loadFromFile(filename);
37      }
38
39      void loadFromFile(const std::string& filename);
```

Rysunek 3 Klasa ProblemInstance

Algorytm losowy

Zaimplementowany algorytm układał w sposób losowy wektor N miast. Następnie dzielił ten wektor na ścieżki z uwagi na zapotrzebowanie.

Algorytm zachłanny

Zaimplementowany algorytm zachłanny pozwalał wybrać, które miasto ma zostać odwiedzone jako pierwsze, bez względu na jego odległość od magazynu. Kolejne wybierane miasta były tymi, które miały najmniejszą odległość od aktualnego miasta i zarazem ich odwiedzenie nie przekroczyłoby limitu zapotrzebowania. Gdy nie było dostępnego miasta, algorytm zapisywał ścieżkę i rozpoczynał nową.

Algorytm genetyczny

Zaimplementowany algorytm genetyczny polegał na następujących parametrach, które wpływały na jego działanie:

- `max_number_of_generations` – liczba iteracji głównej pętli algorytmu (10tys.)
- `max_population_size` – liczba osobników początkowej populacji i limit liczebności następnych pokoleń (500)
- `mutation_factor` – liczba rzeczywista z przedziału (0, 1) określająca prawdopodobieństwo mutacji danego osobnika (0.1)
- `crossover_factor` - liczba rzeczywista z przedziału (0, 1) określająca prawdopodobieństwo wytworzenia potomstwa przez daną parę osobników (0.7)
- `ALLOW_INTO_NEXT_GENERATION_THRESHOLD` – zmienna odpowiedzialna za wprowadzenie elitaryzmu – osobniki, których fitness przekracza tą wartość dopuszczani są bez zmian do kolejnego pokolenia (0.97)
- `MINIMAL_REQUIRED_FITNESS` – osobniki, których wynik fitness jest mniejszy niż ta wartość usuwane są z populacji, (0.7)

W nawiasach przy poszczególnych parametrach podano ich wartości podczas testów.

Populacja inicjalizowana była poprzez utworzenie jednego rozwiązania z wykorzystaniem algorytmu zachłannego, a pozostałych przy użyciu algorytmu losowego. Następnie dla każdego osobnika obliczano wartość fitness z przedziału od 0 do 1. 0 oznacza najgorsze rozwiązanie w populacji, a 1 najlepsze.

Następnie z populacji usuwani są nadmiarowe osobniki lub te które mają zbyt niską wartość fitness. Kolejno rozpoczynana jest pętla przechodząca po wszystkich pozostałych osobnikach. Do następnej generacji dopuszczane są te, których fitness przekracza wartość `ALLOW_INTO_NEXT_GENERATION_THRESHOLD`. Losowane są również rozwiązania, które będą brały udział w operacjach mutacji i krzyżowania.

Zaimplementowana metoda krzyżowania na początku iteruje po wszystkich ścieżkach rodzica A. Dla każdej z tych ścieżek występuje szansa 50%, że zostanie ona przeniesiona do potomka. Następnie tworzony jest wektor `remaining`, który zawiera miasta nieuwzględnione w ścieżkach potomka. Klienci w tym wektorze są ustawieni w tej samej kolejności co w ścieżkach rodzica B. Z wektora `remaining` powstają ścieżki, które dodawane są do potomka. Wadą zaimplementowanego operatora jest to, że potomek może być kopią rodzica A.

Powyższe procesy powtarzane są do czasu osiągniecia kryterium zatrzymania.

Tabu Search

Zaimplementowany algorytm TS jako argument przyjmował jedynie maksymalną liczbę iteracji. Maksymalny rozmiar sąsiedztwa definiowany był jako 5-cio krotność rozmiaru instancji problemu.

Pierwsze rozwiązanie uzyskane jest przy użyciu algorytmu zachłannego, po czym rozpoczyna się główna pętla algorytmu. Generowana jest lista sąsiadów, której rozmiar wynosi maksymalnie 5-cio krotność rozmiaru instancji. Przy wyznaczaniu sąsiadów ścieżki aktualnego rozwiązania spłaszczane są do jednego wektora. Następnie dwa losowe elementy w tym wektorze zamieniane są miejscami. Wektor następnie dzielony jest na trasy, aby nie przekroczyć pojemności ciężarówki. Tak powstałe rozwiązanie jest zapisywane w liście wraz z ruchem, który doprowadził do jego utworzenia.

Po uzyskaniu listy sąsiadów jako kolejne rozwiązanie wybierane jest to, które ma najniższy koszt, a ruch, który doprowadził do jego utworzenia, nie znajduje się na liście tabu. Warunek listy tabu może zostać pominięty, jeśli dane rozwiązanie spełnia kryterium aspiracji (jest lepsze niż jakiekolwiek dotychczas znalezione rozwiązanie).

Następnie, gdy wybrane zostanie kolejne rozwiązanie, dodawane jest ono do listy tabu na obliczoną liczbę iteracji. Na końcu pętli zmniejszany jest czas przez jaki ruchy pozostają w liście tabu lub są z niej usuwane, gdy upłynie wystarczająca liczba iteracji.

Wyniki

Algorytm zachłanny był uruchamiany tyle razy, ile było miast w danej instancji. Za każdym razem pierwszym miastem był inny punkt.

Algorytm genetyczny, losowy oraz tabu search działały przez 10 tys. iteracji. Algorytmy te były również uruchomione 10 razy niezależnie.

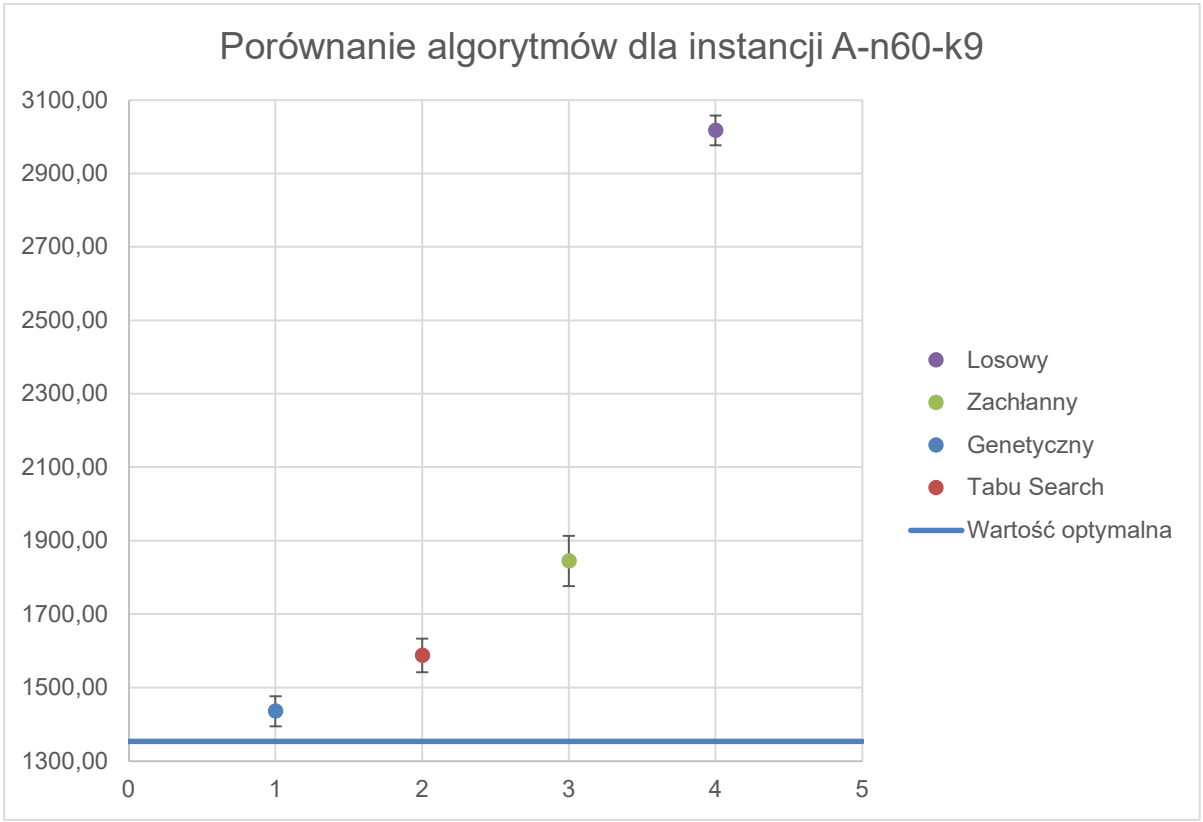
Wyniki w tabelach przedstawiają koszty najlepszego znalezionego rozwiązania (Best), najgorszego znalezionego rozwiązania (Worst), średnią kosztów (Avg) oraz odchylenie standardowe (Std).

Tabela 1 Wyniki dla algorytmu losowego i zachłannego

Instancja	Optymalny wynik	Alg. losowy [10k]				Alg. zachłanny [n]			
		Best	Worst	Avg	Std	Best	Worst	Avg	Std
A-n32-k5.vrp	784	1460	1628	1536,40	48,58	1029	1245,0	1128,2	56,57
A-n37-k6.vrp	949	1638	1753	1701,20	33,38	1224	1466,0	1333,5	64,42
A-n39-k5.vrp	822	1638	1690	1664,60	17,19	990	1239,0	1105,0	51,73
A-n45-k6.vrp	944	2106	2209	2147,60	34,77	1212	1733,0	1432,2	121,47
A-n48-k7.vrp	1073	2203	2341	2287,80	45,51	1371	1679,0	1481,1	71,00
A-n54-k7.vrp	1167	2529	2687	2634,10	49,20	1349	1612,0	1504,8	64,07
A-n60-k9.vrp	1354	2938	3054	3017,50	40,23	1693	2012,0	1845,0	68,05

Tabela 2 Wyniki dla algorytmu genetycznego i tabu search

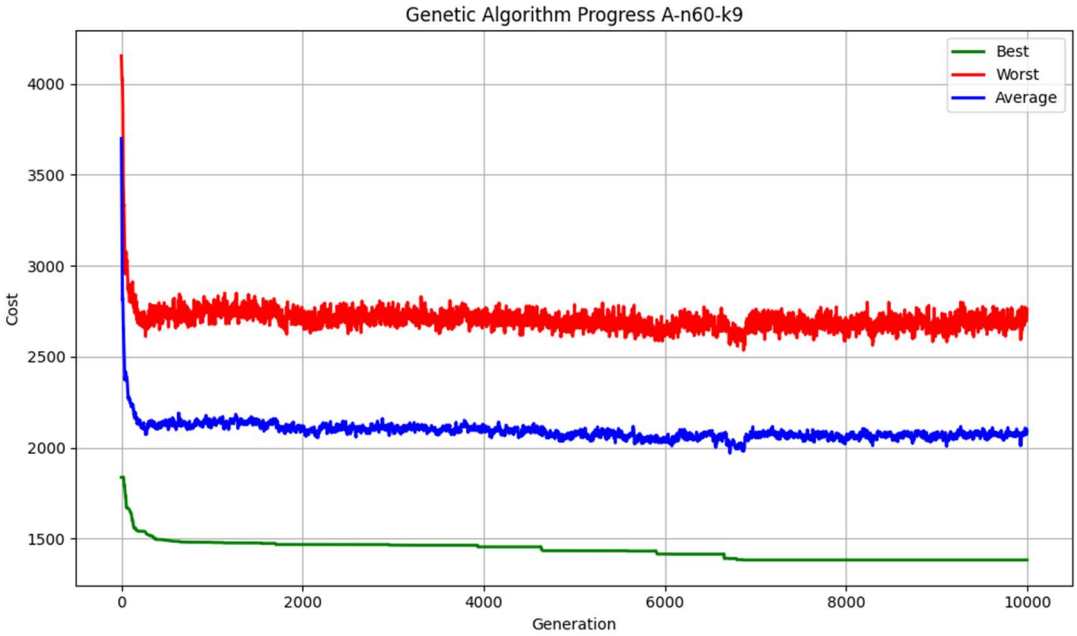
Instancja	Optymalny wynik	Alg. Genetyczny [10k]				Alg. Tabu Search[10k]			
		Best	Worst	Avg	Std	Best	Worst	Avg	Std
A-n32-k5.vrp	784	784	883	805,6	29,38	817	1007	874,3	60,66
A-n37-k6.vrp	949	950	1040	991,5	25,52	1049	1190	1146,3	38,54
A-n39-k5.vrp	822	828	878	855,3	16,22	915	983	948,4	19,27
A-n45-k6.vrp	944	1006	1090	1040,5	23,10	1106	1298	1176,8	57,31
A-n48-k7.vrp	1073	1168	1261	1195	28,15	1197	1319	1251,3	37,25
A-n54-k7.vrp	1167	1242	1317	1286	23,05	1296	1326	1308,2	9,78
A-n60-k9.vrp	1354	1383	1516	1435,8	40,80	1492	1651	1587,7	46,11



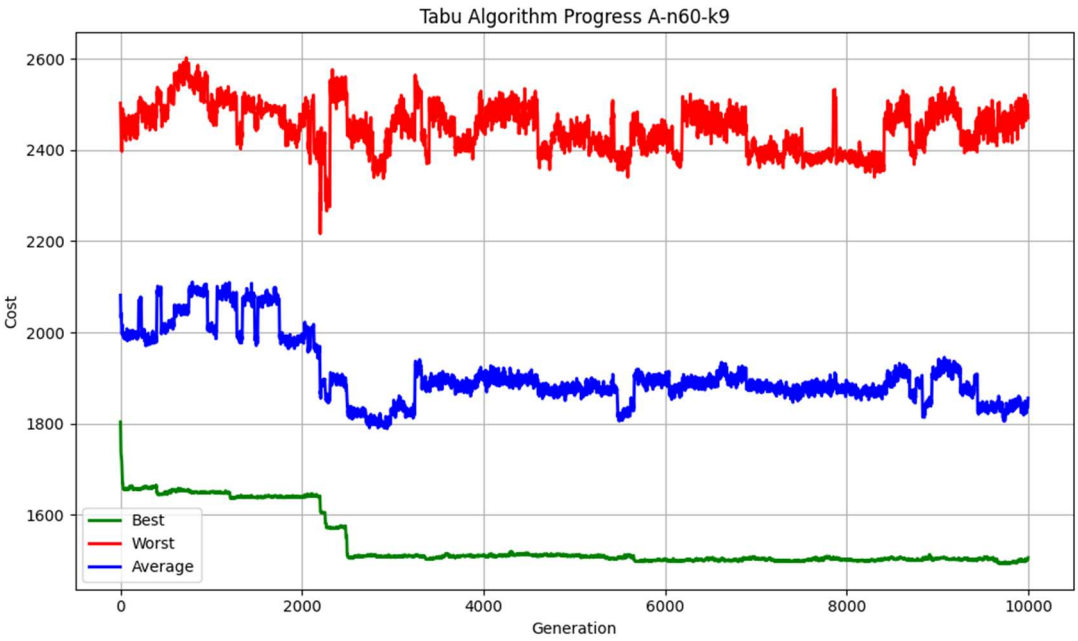
Rysunek 4 Średnie wyniki algorytmów dla instancji A-n60-k9

Z tabel 1. oraz 2. widać, że algorytm genetyczny dla każdej instancji zwracał najlepsze wyniki. Dla instancji A-n32-k5 znalazł też rozwiązanie optymalne. Tabu search znajdował rozwiązania gorsze niż algorytm genetyczny, ale i tak były to rozwiązania lepsze niż te otrzymane przy użyciu algorytmu zachłannego. Algorytm losowy ani razu nie znalazł rozwiązania lepszego niż te otrzymywane przy użyciu metody zachłannej.

Rys. 4 przedstawia średnie wartości rozwiązań dla poszczególnych algorytmów. Słupki błędu to średnia wartość pomniejszona lub powiększona o odchylenie standardowe.



Rysunek 5 Wartości rozwiązań we wszystkich generacjach algorytmu genetycznego



Rysunek 6 Wartości rozwiązań we wszystkich iteracjach algorytmu genetycznego

Rysunki 5. i 6. przedstawiają, jak wyglądał rozkład rozpatrywanych rozwiązań w poszczególnych iteracjach odpowiednio dla algorytmu genetycznego oraz TS. Z wykresów można zauważyć, że w TS występują większe wahania kosztów niż dla algorytmu genetycznego, w którym wartość średnia rozwiązań po początkowym spadku utrzymuje się na podobnym poziomie.

Najlepsze znalezione rozwiązania

Wszystkie poniższe rozwiązania zostały uzyskane przez algorytm genetyczny.

- A-n32-k5

```
Route #1: 30 16 1 12
Route #2: 21 31 19 17 13 7 26
Route #3: 6 3 2 23 4 11 28 14
Route #4: 20 5 25 10 15 22 9 8 18 29
Route #5: 27 24
Cost 784
```

- A-n37-k6

```
Route #1: 6 31 34 17 18 14
Route #2: 27 32 15 30 26 4
Route #3: 13 35 25
Route #4: 19 21 9 5 3 8
Route #5: 20 1 33 2 28 23 22 12 11 10
Route #6: 36 29 24 16 7
Cost 950
```

- A-n39-k5

```
Route #1: 8 11 26 34 37 35 24 9
Route #2: 23 20 29 16 7 3 22 2 17
Route #3: 6 36 27 28 13 30 21
Route #4: 31 1 5 32 10 15 38 18
Route #5: 4 12 33 25 19 14
Cost 828
```

- A-n45-k6

```
Route #1: 14 31 35 1 44 6
Route #2: 24 37 34 30 40 11 26
Route #3: 23 43 18 17 19 27 29
Route #4: 5 21 33 41 8 10 16 4 42
Route #5: 28 7 32 13 20 3 9
Route #6: 38 2 25 15 22 36 39 12
Cost 1006
```

- A-n48-k7

```
Route #1: 31 43 23 44 18
Route #2: 28 30 46 42 11 26 39 45
Route #3: 14 17 16 10 47
Route #4: 40 27 15 8 20 3 7 35
Route #5: 32 38 19 25 37 36
Route #6: 12 5 1 6 22
Route #7: 41 2 33 21 13 4 24 29 9 34
Cost 1168
```

- A-n54-k7

```
Route #1: 30 52 16 6 13 11 38 26 35
Route #2: 18 5 50 39 7 28 4
Route #3: 32 44 9 33 21 45 17
```

```
Route #4: 53 3 22 31 19
Route #5: 23 29 15 10 1 36 49 20 43
Route #6: 25 47 51 2 12 27 14
Route #7: 34 41 46 42 24 48 40 8 37
Cost 1242
```

- A-n60-k9

```
Route #1: 16 20 3 11 4 21 40 25
Route #2: 14 47 23 2 28 31 6
Route #3: 39 42 45 58
Route #4: 34 24 1 44 49 30 53 46
Route #5: 50 43 56 12 51 9 32
Route #6: 35 55 15 26 17 37 57 27
Route #7: 41 33 38 59 52 18
Route #8: 7 29 13 8 19
Route #9: 36 22 10 54 5 48
Cost 1383
```

Wnioski

Dzięki przeprowadzonym badaniom można sformułować następujące wnioski:

- Z zaimplementowanych algorytmów najlepiej sprawdził się algorytm genetyczny, dla najmniejszej badanej instancji odnaleziono przy jego użyciu rozwiązanie optymalne
- Algorytm losowy nie jest dobrą metodą znajdowania rozwiązań o niskim koszcie. Przydatny jest w wyznaczaniu początkowej przestrzeni rozwiązań w algorytmach, które jej wymagają
- Wynik algorytmu zachłannego jest dobrym punktem wyjściowym do sprawdzenia jakości innych algorytmów. Sprawdza się również jako wstępne rozwiązanie w algorytmach, ponieważ jest prosty w implementacji
- W zaimplementowanych algorytmie genetycznym i TS widać, jak dla GA średni koszt mniej waha się w czasie niż dla TS
- Wykorzystanie wielowątkowości w C++ ułatwiło testowanie, ponieważ zmniejszyło czas oczekiwania na wyniki testów

Repozytorium

Kod napisany w ramach projektu dostępny jest na repozytorium GitHub.