Analiza efektywności wybranych algorytmów optymalizacji

https://github.com/skowron4/CVRP-optimization-metaheuristics

Maj 2025

Spis treści

1 Wprowadzenie teoretyczne

Celem projektu było porównanie efektywności wybranych algorytmów optymalizacyjnych w rozwiązywaniu problemu CVRP.

Capacitated Vehicle Routing Problem (CVRP) jest rozszerzeniem klasycznego problemu komiwojażera (TSP), w którym należy zaplanować trasy floty pojazdów tak, aby:

- każdy klient został odwiedzony dokładnie raz,
- zapotrzebowanie każdego klienta nie przekroczyło pojemności żadnego z pojazdów,
- każda trasa zaczynała się i kończyła w tym samym magazynie (depo),
- łączny koszt (najczęściej mierzony długością tras) był minimalny.

CVRP należy do klasy problemów NP-trudnych – liczba możliwych rozwiązań rośnie wykładniczo wraz ze wzrostem liczby klientów. Metaheurystyki nie gwarantują odnalezienia optymalnego rozwiązania, lecz umożliwiają znalezienie rozwiązań wysokiej jakości w rozsądnym czasie, co jest kluczowe w praktycznych zastosowaniach logistycznych.

1.1 Funkcja oceniająca

Porównanie wyników przeprowadzono w następujący sposób. Rozwiązanie reprezentowane jest jako wektor kolejnych punktów do odwiedzenia, przy czym wartość 0 oznacza powrót do magazynu. Wartość funkcji celu obliczana jest jako łączna odległość, jaką należy przebyć, aby odwiedzić wszystkie punkty – im mniejsza, tym lepsze rozwiązanie. Podczas oceny wektora ciągi kolejnych zer traktowane są jako pojedyncze wystąpienie, co zapewnia bardziej zorganizowane przeszukiwanie przestrzeni rozwiązań i eliminuje wielokrotne „odwiedzanie" tego samego rozwiązania.

Liczba dodawanych zer odpowiada liczbie miast pomniejszonej o jeden, dzięki czemu kombinatoryczna liczba możliwych rozwiązań wzrasta z $N!$ do $(2N-1)!$. Choć znacznie powiększa to teoretyczną przestrzeń rozwiązań, pozwala wychwycić trasy, które przy zachłannym generowaniu bez użycia zer mogłyby pozostać nieodkryte. Jednak wprowadzenie dodatkowych zer powoduję również pewną redundancję. Ponieważ ciągi zer mogą być traktowane jako pojedynczy element, efektywna przestrzeń rozwiązań zostaje znacząco zredukowana, co pozwala na szybsze znalezienie tras zbliżonych do optymalnych.

1.2 Operatory sąsiedztwa

W algorytach analizowanych w tym doświadczeniu zostały zaimplementowane dwie metody mutacji : *zamiana* i *inwersja*.

W analizowanym algorytmie tabu zastosowano dwa operatory służące do próbkowania sąsiedztwa: *zamiana* oraz *inwersja*.

1.2.1 Zamiana

Losowo wybierane są dwa różne indeksy wektora rozwiązania. Procedurę powtarza się do momentu, gdy przynajmniej jeden z wybranych punktów ma wartość różną od zera (czyli dopuszcza się, aby jeden punkt był powrotem do magazynu, pod warunkiem że drugi to faktyczny klient). Po spełnieniu warunku następuje zamiana wartości na tych dwóch pozycjach.

1.2.2 Inwersja

Operator inwersji również rozpoczyna się od losowego wyboru dwóch różnych indeksów, spełniających identyczne kryterium jak przy zamianie, a także podciąg nie może składać się z samych zer. Wybrane indeksy wyznaczają początek i koniec podciągu wektora. Następnie cały ten podciąg jest odwracany i wpisywany z powrotem w to samo miejsce, co pozwala na wygenerowanie nowego sąsiedniego rozwiązania.

1.3 Algorytm Przeszukiwanie Tabu

Osobnik startowy generowany jest w sposób całkowicie losowy — wektor punktów do odwiedzenia powstaje przez losowy wybór kolejnych miast i powrotów do magazynu. Tak utworzone rozwiązanie (bieżący osobnik), ocenione za pomocą funkcji celu, stanowi punkt wyjściowy oraz podstawę do wygenerowania pierwszego zbioru sąsiedztwa.

Następnie sprawdzany jest warunek zakończenia algorytmu. W badanym podejściu kryterium zatrzymania stanowi osiągnięcie z góry określonej liczby iteracji.

Kolejnym krokiem jest wyznaczenie najlepszego rozwiązania z aktualnego sąsiedztwa. Nowe rozwiązanie dodaje się do listy Tabu, jeśli nie znajduje się na niej wcześniej. Jeżeli nowe rozwiązanie jest lepsze od dotychczas najlepszego i nie znajduje się w liście Tabu, zastępuje bieżącego osobnika. Po tej aktualizacji ponownie sprawdzany jest warunek zakończenia, co zamyka cykl algorytmu.

Algorithm 1 Algorytm Przeszukiwanie Tabu

```
1:  current ← randomIndividual
2:  best ← current
3:  add current to tabu
4:  for i = 1 to iterations do
5:      neighborhood ← generate neighborhood(current, neighbourhood_size)
6:      candidate ← best individual from neighborhood not in tabu
7:      current ← candidate
8:      if current better than best then
9:          best ← current
10:         add current to tabu
11:     end if
12: end for
13: return best
```

1.4 Algorytm Wyżarzania

Na początku, analogicznie jak w algorytmie tabu, losowo wybierane jest rozwiązanie początkowe, które następnie zostaje ocenione za pomocą funkcji celu.

W głównej pętli algorytmu sprawdzany jest warunek zakończenia. Jeżeli nie został on spełniony, generowane jest sąsiedztwo od obecnie najlepszego rozwiązania. Spośród sąsiadów wybierany jest najlepszy kandydat.

- Jeżeli jest on lepszy od obecnego rozwiązania, zostaje przyjęty automatycznie.
- Jeżeli jest gorszy, decyzja o jego akceptacji zależy od różnicy wartości funkcji celu oraz aktualnej temperatury: im wyższa temperatura, tym większe prawdopodobieństwo zaakceptowania gorszego rozwiązania.

Po ewentualnej aktualizacji rozwiązania temperatura jest obniżana zgodnie z przyjętym harmonogramem chłodzenia. Prawdopodobieństwo akceptacji gorszego rozwiązania oblicza się ze wzoru:

$$P = \exp\left(\frac{f(V_n) - f(V_c)}{T}\right),$$

gdzie:

* $f(V_n)$ — wartość funkcji celu dla nowego rozwiązania (kandydata), * $f(V_c)$ — wartość funkcji celu dla bieżącego rozwiązania, * T — bieżąca temperatura.

Po tej operacji cykl algorytmu wyżarzania zostaje zamknięty.

Algorithm 2 Algorytm Symulowanego Wyżarzania

```
current ← randomIndividual
best ← current
T ← T_initial
for i = 1 to iterations do
    neighborhood ← generate neighborhood(current, neighbourhood_size)
    for all ind in neighborhood do
        if ind better than current or P_acceptance(ind, current, T) > random(0, 1) then
            current ← ind
        end if
    end for
    if current better than best then
        best ← current
    end if
    T ← α × T                    ▷ example cooling schedule with coefficient α = 0.999
end for
return best
```

1.5 Hybryda Wyżarzanie–Tabu (Hartowanie)

Hybryda łączy mechanizmy algorytmu wyżarzania z pamięcią tabu – pozwalamy na wielokrotne „podgrzewanie" systemu, jednocześnie unikając powrotów do niedawno odwiedzonych rozwiązań dzięki liście tabu. Zwiększenie temperatury ma na celu „wybicie" algorytmu z lokalnego optimum, a lista tabu zapobiega cyklom i wspiera eksplorację nowych rejonów przestrzeni rozwiązań. Celem hybrydy jest maksymalizacja eksploracji sąsiedztwa po wybiciu z lokalnego optimum.

Po osiągnięciu minimalnej temperatury rozpoczyna się odliczanie kolejnych iteracji (np. 100). Po upływie tej liczby iteracji system jest ponownie podgrzewany — temperatura zostaje ustawiona na zdefiniowany poziom (który może różnić się od temperatury początkowej). Cykl schładzania i podgrzewania, wspierany przez mechanizm tabu, powtarza się aż do osiągnięcia z góry określonej liczby iteracji.

Algorithm 3 Algorytm Hybrydowy Tabu-Wyżarzanie

$current \leftarrow randomIndividual$
$best \leftarrow current$
add *current* to tabu list
$T \leftarrow T_{initial}$
$isCooling \leftarrow true$
$iterToHeat \leftarrow iterations\_to\_start\_heating$
for $i = 1$ to *iterations* **do**
    $neighborhood \leftarrow$ generate neighborhood($current$, $neighbourhood\_size$)
    for all *ind* in *neighborhood* **do**
        if *ind* not in tabu **then**
            if *ind* better than *current* **or** $P_{acceptance}(ind, current, T) > random(0, 1)$ **then**
                $current \leftarrow ind$
            end if
        end if
    end for
    if *current* better than *best* **then**
        $best \leftarrow current$
        add *current* to tabu
    end if
    if *isCooling* **then**
        if $iterToHeat = 0$ **then**
            $iterToHeat \leftarrow iteration\_to\_start\_heating$
        end if
        $T \leftarrow \alpha \times T$ ▷ example cooling schedule with coefficient $\alpha = 0.995$
        if $T = T_{final}$ **then**
            $isCooling \leftarrow$ false
        end if
    else
        if $iterToHeat > 0$ **then**
            $iterToHeat \leftarrow iterToHeat - 1$
        else
            $T \leftarrow \beta \times T$ ▷ example heating schedule with coefficient $\beta = 1.15$
            if $T = T_{max}$ **then**
                $isCooling \leftarrow$ true
            end if
        end if
    end if
end for
return *best*

2 Dane testowe

Do przeprowadzenia eksperymentów użyto standardowych danych testowych pochodzących z repozytorium problemów VRP dostępnego publicznie pod adresem:

`http://vrp.atd-lab.inf.puc-rio.br/index.php/en/`

Wykorzystano instancje ze **zbioru A (Augerat's Set A)**, które są powszechnie stosowane w literaturze do oceny efektywności heurystyk i metaheurystyk dla problemu CVRP.

2.1 Struktura pliku danych

Każdy plik instancji zawiera komplet informacji w następującym formacie:

- **Nagłówek (Header)** – podstawowe informacje:
  - `NAME` – nazwa instancji,
  - `COMMENT` – komentarz (np. liczba pojazdów, wartość optymalna),
  - `TYPE` – typ problemu (`CVRP`),
  - `DIMENSION` – liczba wierzchołków (depo + klienci),
  - `EDGE_WEIGHT_TYPE` – typ metryki (np. `EUC_2D`),
  - `CAPACITY` – pojemność każdego pojazdu.
- **NODE_COORD_SECTION** – współrzędne każdego wierzchołka (X, Y).
- **DEMAND_SECTION** – zapotrzebowanie każdego wierzchołka.
- **DEPOT_SECTION** – numer wierchołka depo/magazynu (zwykle 1), zakończony wartością `-1`.
- **EOF** – znacznik końca pliku.

Przykład fragmentu pliku:

```
NAME : A-n32-k5
TYPE : CVRP
DIMENSION : 32
CAPACITY : 100
NODE_COORD_SECTION
1 82 76
2 96 44
...
DEMAND_SECTION
1 0
2 19
...
DEPOT_SECTION
1
-1
EOF
```

2.2 Wykorzystane instancje

W eksperymentach zastosowano następujące instancje dla problemu CVRP z zestawu A:

- `A-n32-k5` – 32 wierzchołki, 5 pojazdów
- `A-n37-k6` – 37 wierzchołków, 6 pojazdów
- `A-n39-k5` – 39 wierzchołków, 5 pojazdów
- `A-n45-k6` – 45 wierzchołków, 6 pojazdów
- `A-n48-k7` – 48 wierzchołków, 7 pojazdów
- `A-n54-k7` – 54 wierzchołki, 7 pojazdów
- `A-n60-k9` – 60 wierzchołków, 9 pojazdów
- `A-n61-k9` – 61 wierzchołków, 9 pojazdów

Każda instancja różni się rozmiarem i stopniem złożoności, co umożliwia wszechstronną ocenę algorytmów oraz przeprowadzenie porównawczej analizy jakości i skalowalności każdej zastosowanej metaheurystyki.

3 Strojenie metod

Strojenie metod polegało na analizie zachowania się algorytmów w różnych konfiguracjach parametrów. Proces ten miał na celu identyfikację ustawień, które prowadzą do możliwie najlepszych wyników. W szczególności skupiono się na analizie wyników osiąganych przez algorytmy w 50 000 iteracjach, z populacją o wielkości 40 osobników w każdej iteracji. Pozwoliło to na wiarygodne porównanie efektywności poszczególnych strategii oraz dostrojenie parametrów w taki sposób, aby maksymalizować skuteczność algorytmów.

4 Porównanie wykorzystanych algorytmów

Każdy algorytm uruchomiono 100 razy, co pozwoliło zminimalizować wpływ losowości na wyniki. W dalszej części przedstawiono parametry początkowe oraz podsumowanie wyników dla poszczególnych metod.

4.1 Parametry początkowe Przeszukiwania Tabu

Parametr	Wartość
Liczba iteracji	15000
Operator sąsiedztwa	inwersja
Rozmiar tablicy tabu	200
Rozmiar sąsiedztwa	40

4.2 Parametry początkowe Algorytmu Wyżarzania

Parametr	Wartość
Liczba iteracji	15000
Operator sąsiedztwa	inwersja
Rozmiar tablicy tabu	200
Rozmiar sąsiedztwa	40
Temperatura początkowa	700.0
Temperatura końcowa	0.0
Sposób schładzania	geometryczny
Prędkość schładzania	0.9987

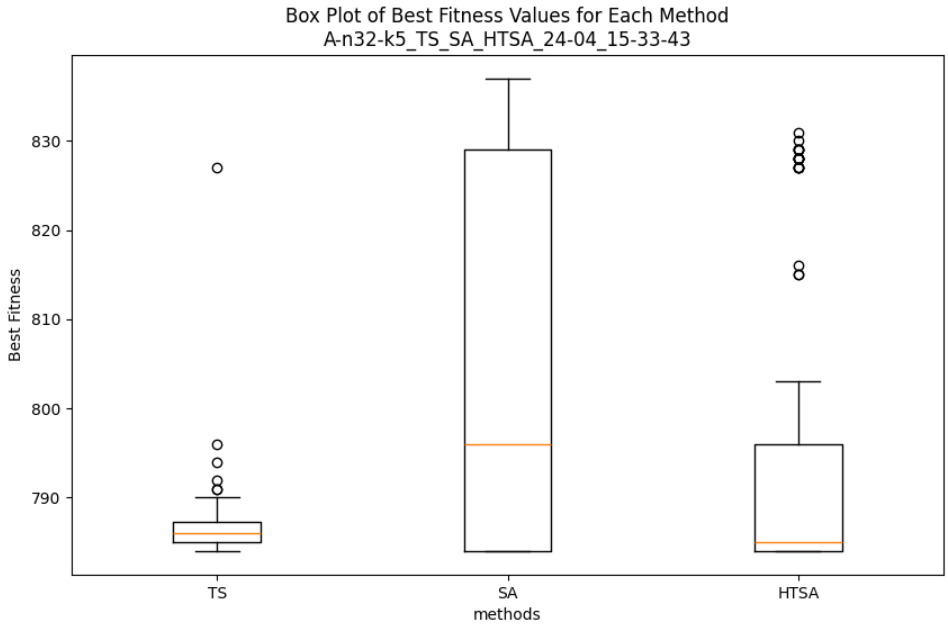
4.3 Parametry początkowe Hybrydy Tabu-Wyżarzanie

Parametr	Wartość
Liczba iteracji	15000
Liczba iteracji do rozpoczęcia podgrzewania	50
Operator sąsiedztwa	inwersja
Rozmiar tablicy tabu	250
Rozmiar sąsiedztwa	40
Temperatura początkowa	700.0
Temperatura końcowa	0.001
Maksymalna temperatura podgrzania	20.0
Sposób schładzania	geometryczny
Prędkość schładzania	0.995
Sposób podgrzewania	geometryczny
Prędkość podgrzewania	1.5

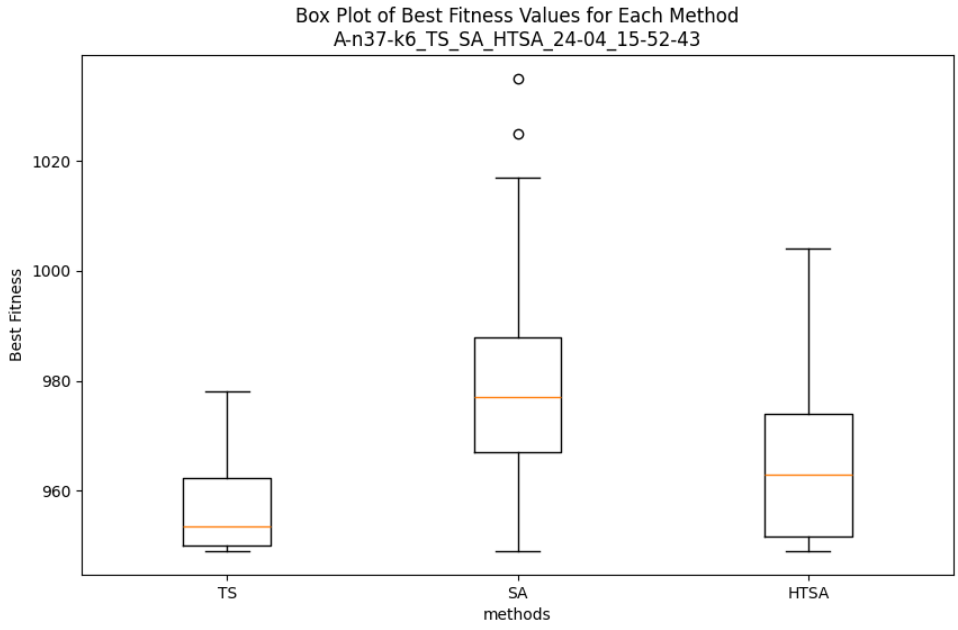
Instacja	Opt. wynik	Tabu (100x)				SA (100x)				Tabu+SA (100x)			
		Best	Worst	Avg	Std	Best	Worst	Avg	Std	Best	Worst	Avg	Std
N32-k5	**784**	784	827	**786.94**	4.54	784	837	802.78	20.85	784	831	793.78	16.34
N37-k6	**949**	949	978	**956.75**	8.26	949	1035	978.47	18.11	949	1004	963.98	13.13
N39-k5	**822**	825	849	837.08	4.84	822	905	839.71	15.81	822	857	**830.27**	5.93
N45-k6	**944**	965	1018	987.26	11.89	944	1019	977.51	13.79	947	994	**965.9**	9.41
N48-k7	**1073**	1106	1185	1160.44	13.5	1091	1178	1128.24	16.21	1074	1150	**1114.03**	13.65
N54-k7	**1167**	1216	1288	1255.59	15.91	1177	1307	1227.45	26.43	1172	1244	**1201.7**	16.26
N60-k9	**1354**	1394	1503	1449.34	22.73	1376	1572	1440.74	33.38	1364	1464	**1414.46**	22.68
N61-k9	**1034**	1069	1133	1099.9	14.2	1040	1136	1079.94	21.25	1036	1102	**1067.41**	14.35

Tabela 1: Wyniki algorytmów dla różnych instancji

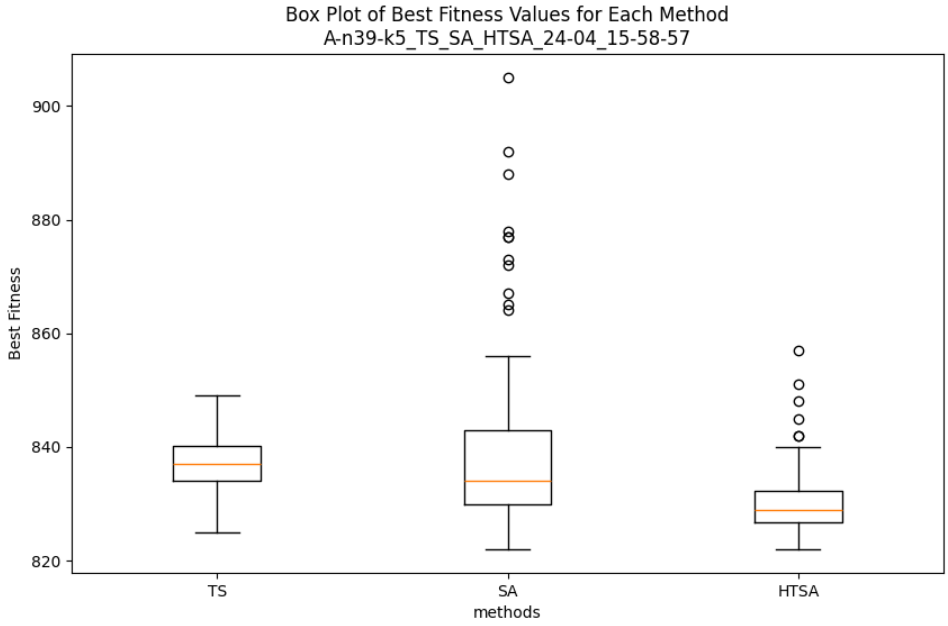
4.4 Zachowanie algorytmów



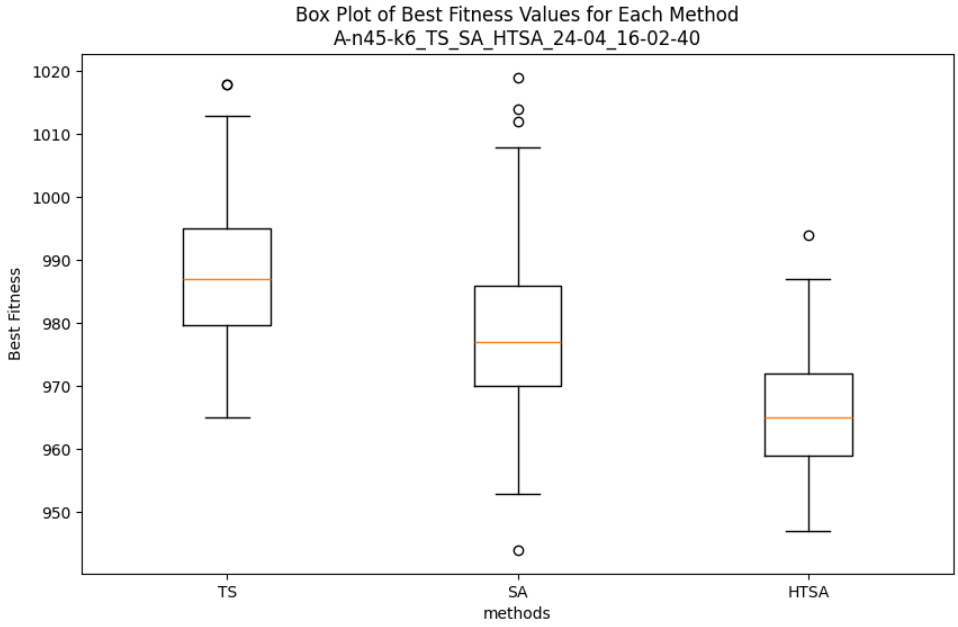
Rysunek 1: Wykres pudełkowy dla danych A-n32-k5



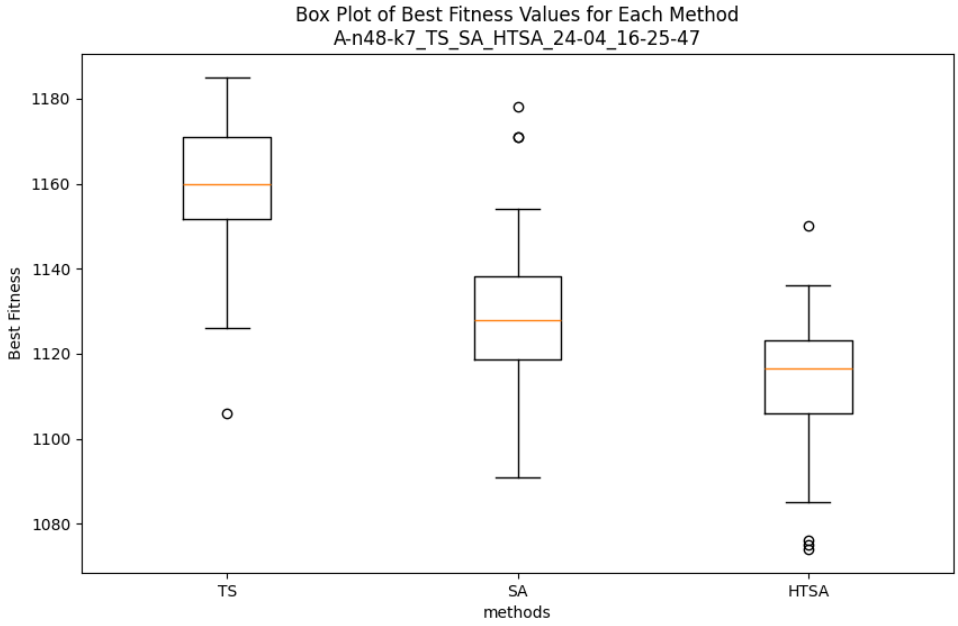
Rysunek 2: Wykres pudełkowy dla danych A-n37-k6



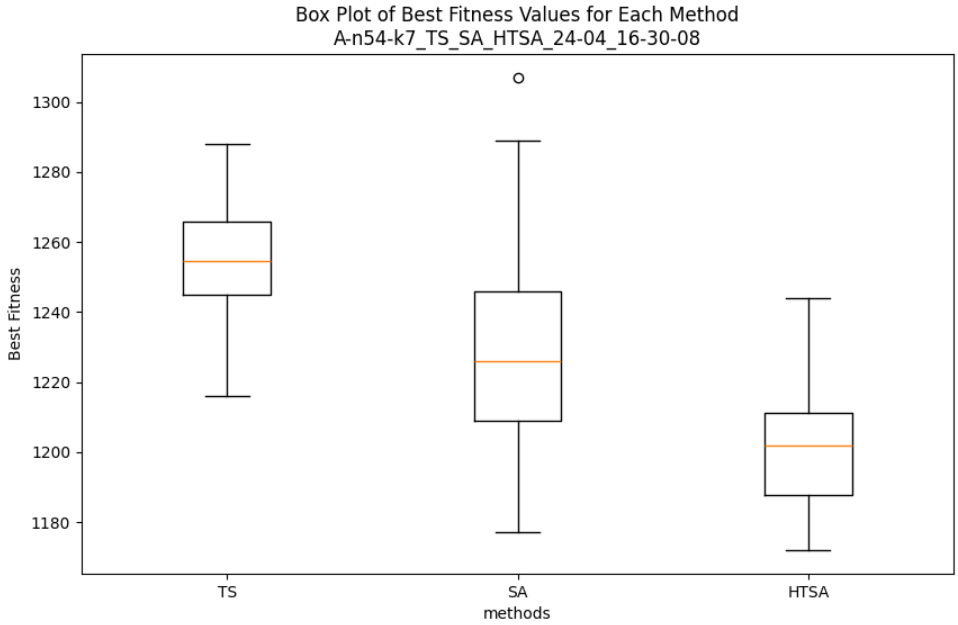
Rysunek 3: Wykres pudełkowy dla danych A-n39-k5



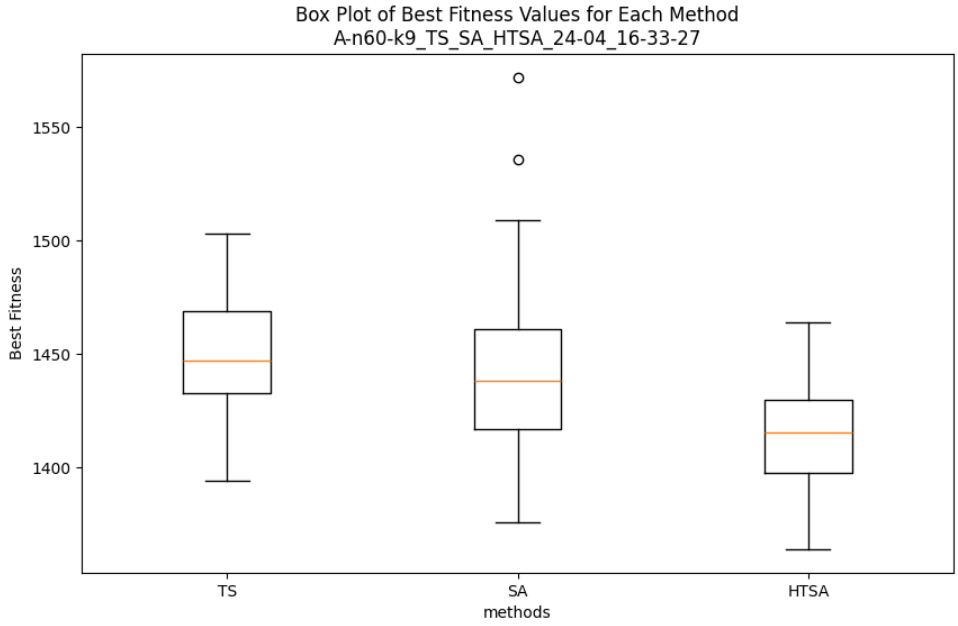
Rysunek 4: Wykres pudełkowy dla danych A-n45-k6



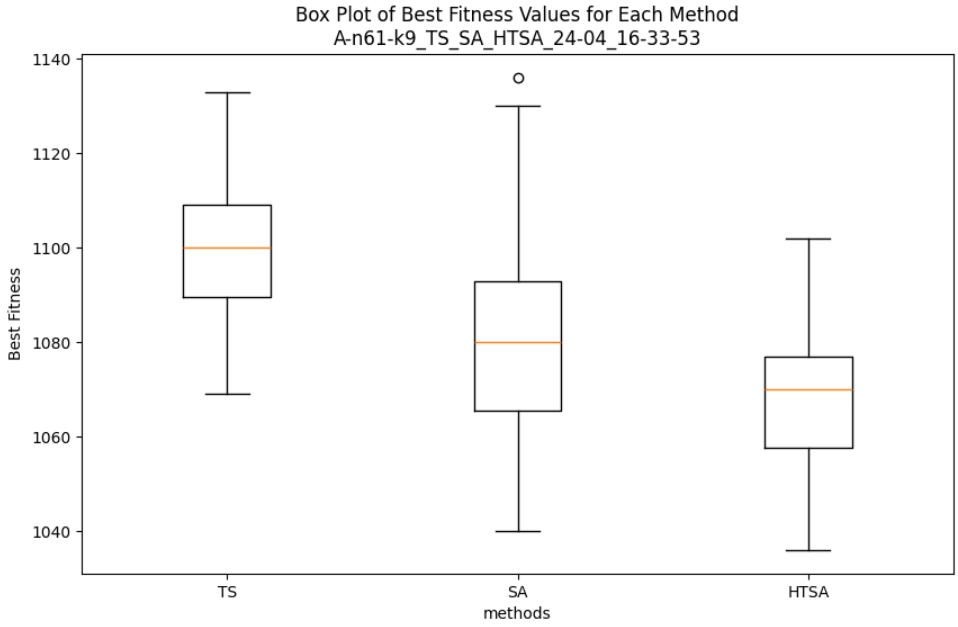
Rysunek 5: Wykres pudełkowy dla danych A-n48-k7



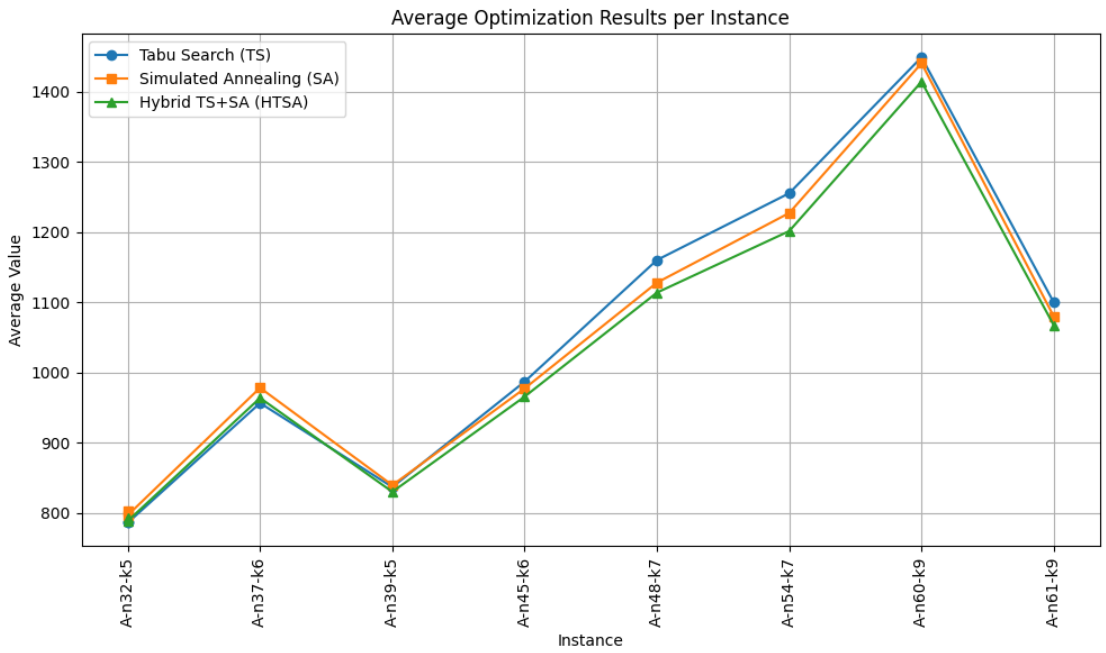
Rysunek 6: Wykres pudełkowy dla danych A-n54-k7



Rysunek 7: Wykres pudełkowy dla danych A-n60-k9

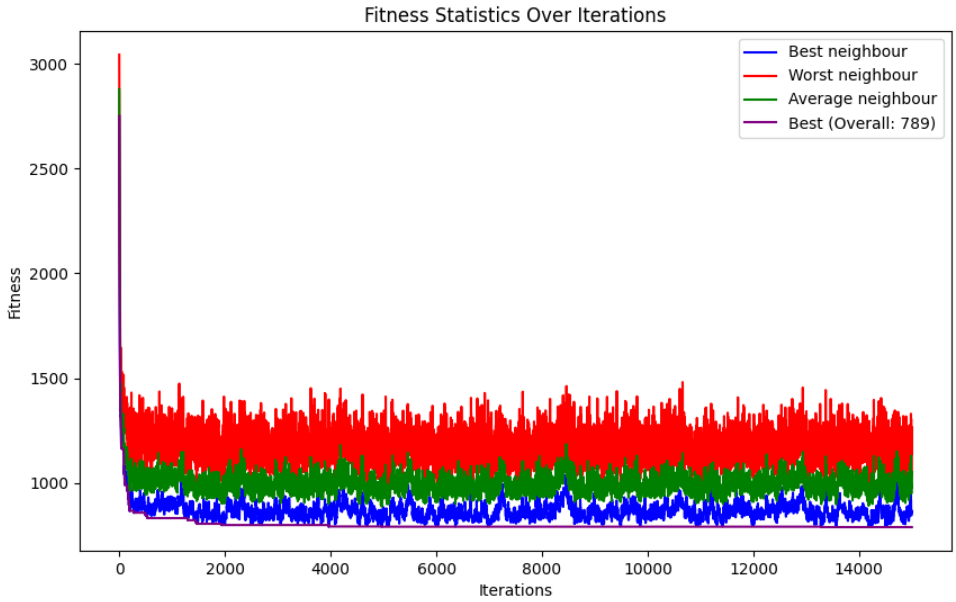


Rysunek 8: Wykres pudełkowy dla danych A-n61-k9



Rysunek 9: Wykres uśrednionych wyników metod dla różnych instancji problemu

4.5 Przeszukiwanie Tabu



Rysunek 10: Wizualizacja zachowania algorytmu preszukiwania tabu na małym problemie

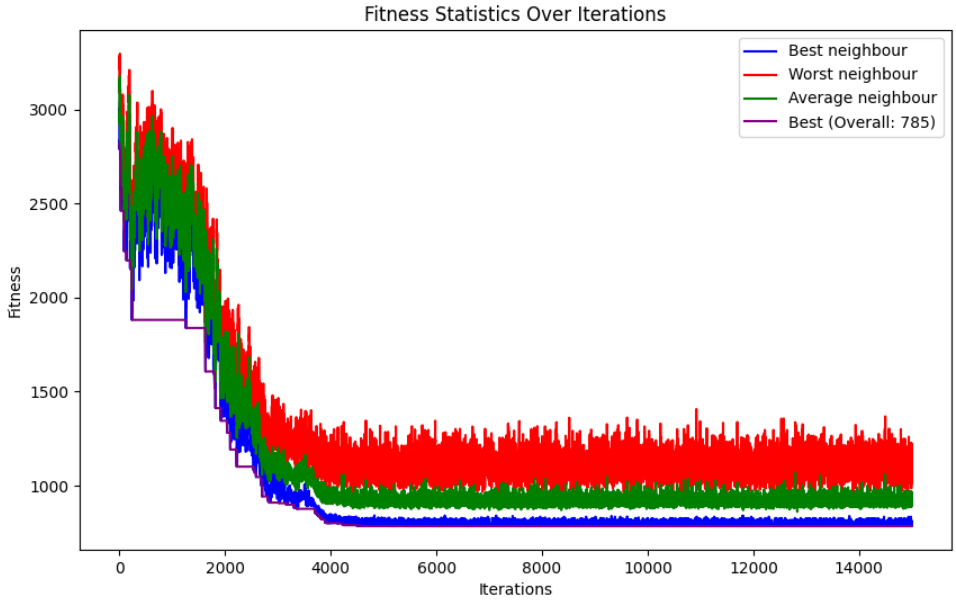
Rysunek 11: Wizualizacja zachowania algorytmu preszukiwania tabu na średnim problemie

Rysunek 12: Wizualizacja zachowania algorytmu preszukiwania tabu na dużym problemie

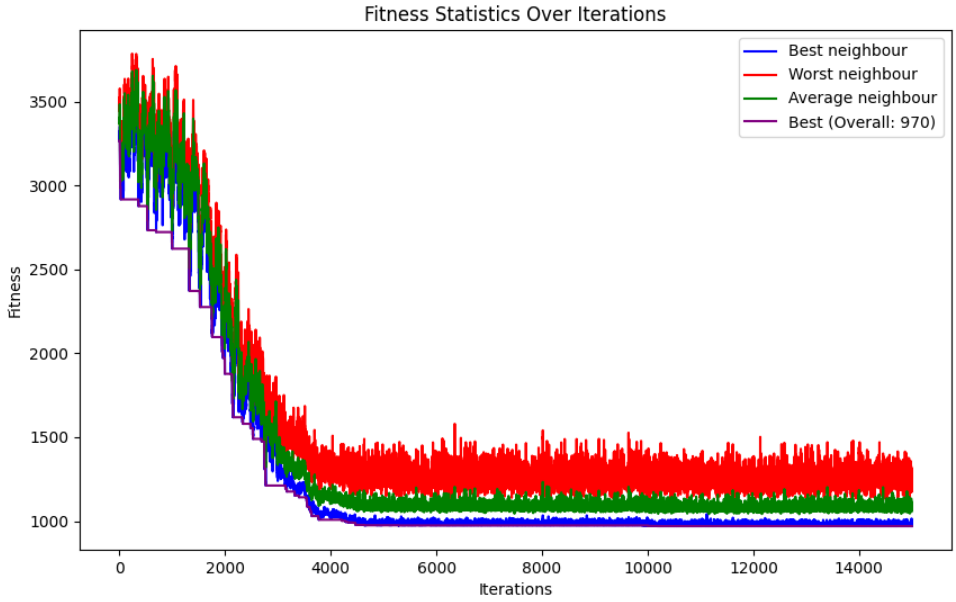
Na zamieszczonych wykresach widać charakterystyczne „skoki” w przebiegu przeszukiwania przestrzeni rozwiązań algorytmu przeszukiwania Tabu. Świadczy to o przechodzeniu między kolejnymi lokalnymi optimami: ścieżka wartości bieżącego rozwiązania nie oddala się znacząco od najlepszego rozwiązania dotychczasowego, a jednocześnie nie pokrywa z nim w całości. Oznacza to, że algorytm skutecznie eksploruje przestrzeń rozwią-

zań, unikając zarówno przedwczesnej zbierzności, jak i niepotrzebnego dryfowania po gorszych rozwiązaniach. Taki przebieg potwierdza właściwe dobranie parametrów wielkości listy tabu oraz strategii wyboru rozwiązań.

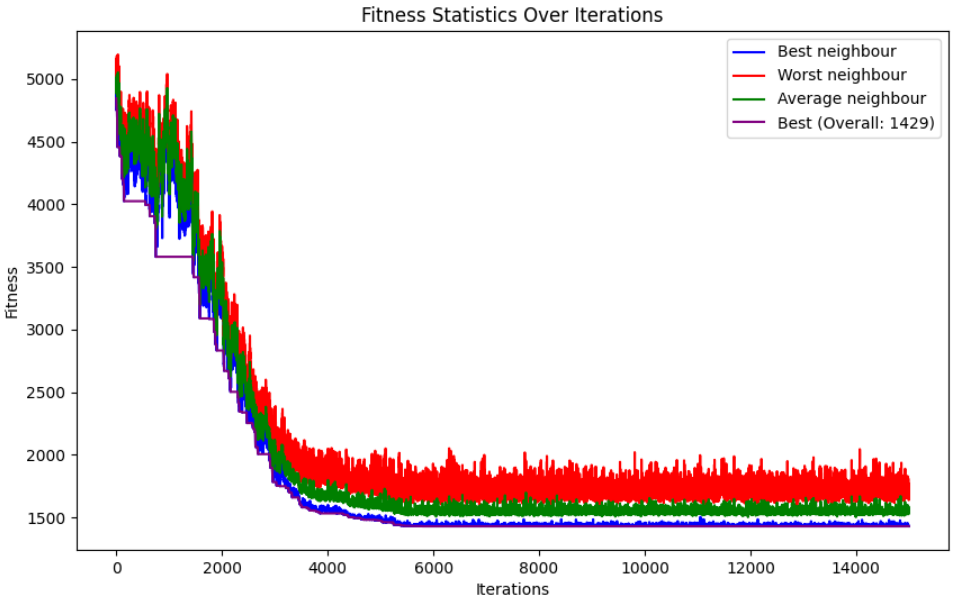
4.6 Algorytm Wyżarzania



Rysunek 13: Wizualizacja zachowania algorytmu wyżarzania na małym problemie



Rysunek 14: Wizualizacja zachowania algorytmu wyżarzania na średnim problemie



Rysunek 15: Wizualizacja zachowania algorytmu wyżarzania na dużym problemie

Na zamieszczonych wykresach widać typowy przebieg działania algorytmu symulowanego wyżarzania. W początkowej fazie, przy wysokiej temperaturze, eksploracja przestrzeni rozwiązań przypomina proces stochastyczny — algorytm często akceptuje gorsze rozwiązania, co zapobiega przedwczesnemu utknięciu w lokalnych minimach.

W miarę obniżania temperatury poszukiwania stają się coraz bardziej ukierunkowane. Coraz rzadziej akceptowane są gorsze rozwiązania, a jakość generowanych rozwiązań stopniowo rośnie. W końcowej fazie dominuje przeszukiwanie lokalne: ścieżka wartości bieżącego rozwiązania niemal pokrywa się ze ścieżką najlepszego sąsiada, co świadczy o stabilizacji i dopracowywaniu finalnego rozwiązania.

4.7 Hybryda Tabu-Wyżarzanie

Rysunek 16: Wizualizacja zachowania hybrydy tabu+sa+podgrzanie (hartowanie) na małym problemie

Rysunek 17: Wizualizacja zachowania hybrydy tabu+sa+podgrzanie (hartowanie) na średnim problemie

Rysunek 18: Wizualizacja zachowania hybrydy tabu+sa+podgrzanie (hartowanie) na dużym problemie

Algorytm hybrydowy łączący symulowane wyżarzanie z przeszukiwaniem tabu wykazuje podobny schemat działania do klasycznego wyżarzania, jednak wprowadza iteracyjne „podgrzewanie" systemu, co nadaje mu cykliczny charakter.

W początkowej fazie każdego cyklu następuje szybkie schładzanie systemu, co sprzyja intensywnej eksploracji przestrzeni rozwiązań. Po upływie określonej liczby iteracji (parametr definiowany przez użytkownika) system zostaje ponownie „podgrzany" — temperatura wzrasta, choć zwykle do poziomu niższego niż początkowy. Na wykresie wyraźnie widać, że maksymalne temperatury kolejnych cykli stopniowo maleją. Po osiągnięciu kolejnego progu cykl schładzania powtarza się, a cały proces trwa aż do wyczerpania zdefiniowanej liczby iteracji.

W tej hybrydzie lista tabu działa tak jak w klasycznym algorytmie przeszukiwania tabu — przechowuje niedawno odwiedzone rozwiązania, by zapobiec ich powtarzaniu. Dzięki temu algorytm unika cyklicznego wracania do tych samych tras i jest zmuszony eksplorować nowe obszary przestrzeni rozwiązań.

Takie podejście umożliwia balans pomiędzy globalną eksploracją (dzięki cyklicznemu podgrzewaniu) a lokalnym doskonaleniem rozwiązań (podczas fazy chłodzenia). W efekcie algorytm może efektywnie przeskakiwać pomiędzy różnymi obszarami przestrzeni rozwiązań, jednocześnie dokładnie badając lokalne sąsiedztwa.

5 Profilowanie programu

W celu analizy wydajności algorytmu zastosowano technikę **profilowania**, pozwalającą na precyzyjne zmierzenie czasu wykonania poszczególnych fragmentów kodu oraz identyfikację potencjalnych wąskich gardeł (ang. *bottlenecks*).

5.1 Czym jest profiler?

Profiler to narzędzie umożliwiające monitorowanie działania programu w czasie rzeczywistym lub po jego zakończeniu. Dostarcza informacji o:

- czasie wykonywania funkcji,
- liczbie wywołań poszczególnych fragmentów kodu,
- zużyciu pamięci,
- kolejności wykonywania instrukcji.

Dzięki temu można zoptymalizować program poprzez:

- eliminację nieefektywnych operacji,
- redukcję złożoności obliczeniowej krytycznych sekcji,
- lepsze zarządzanie zasobami.

5.2 Wykorzystanie profilera w projekcie

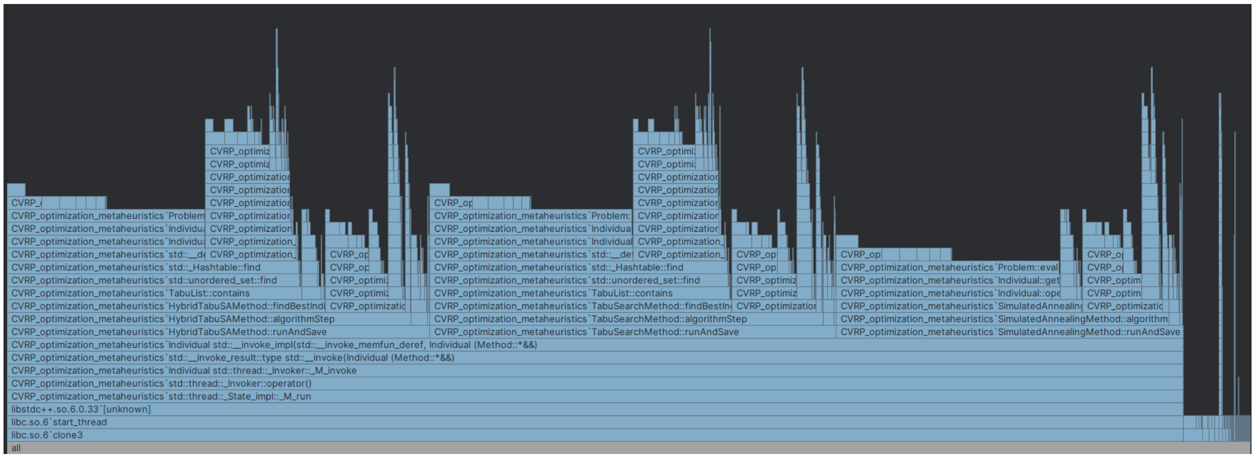
Do profilowania zaimplementowanego algorytmu użyto narzędzia dostępnego w środowisku CLion (przez WSL). Pozwoliło ono na szczegółową analizę funkcji odpowiedzialnych za:

- generowanie rozwiązań początkowych,
- ocenę funkcji celu (długość trasy),
- działanie operatorów mutacji,
- wybór najlepszych osobników do kolejnych iteracji.

Na podstawie wyników profilowania zidentyfikowano fragmenty kodu najbardziej obciążające obliczeniowo. W efekcie wprowadzono optymalizacje, m.in.:

- uproszczenie funkcji wczytywania i zapisywania danych,
- optymalizację procedur mutacji (wybór indeksów),
- usprawnienie zarządzania listą tabu — zastosowano kombinację `unordered_set` (do sprawdzania, czy rozwiązanie jest na liście) oraz `deque` (do szybkiego usuwania najstarszych wpisów w czasie $O(1)$).

Profilowanie przeprowadzono na różnych instancjach (np. `A-n32-k5`, `A-n61-k9`), aby ocenić wpływ rozmiaru danych na wydajność algorytmu.



Rysunek 19: Widok wyników profilowania w środowisku CLion (WSL)

6 Podsumowanie

W przedstawionych eksperymentach porównano trzy podejścia do rozwiązywania problemu CVRP: klasyczny algorytm wyżarzania, przeszukiwanie tabu oraz hybrydę łączącą mechanizm symulowanego wyżarzania z cyklicznym podgrzewaniem i listą tabu.

Dla dwóch najmniejszych instancji (A-n32-k5, A-n37-k6) najlepsze wyniki uzyskało przeszukiwanie tabu — charakteryzowało się ono szybkim dochodzeniem do dobrych lokalnych minimów oraz wysoką stabilnością wyników. W przypadku większych i bardziej złożonych instancji przewagę wykazał algorytm hybrydowy, łączący zalety globalnej eksploracji (wyżarzanie) z mechanizmami tabu. Hybryda skuteczniej unikała pułapek lokalnych minimów i osiągała lepsze wartości funkcji celu.

Ostatecznie podejście hybrydowe okazało się najbardziej efektywne dla trudniejszych instancji CVRP, co potwierdza, że łączenie różnych strategii optymalizacji może prowadzić do istotnej poprawy jakości rozwiązań.

Warto zauważyć, że wszystkie metody zostały starannie dostrojone; jedynie klasyczne wyżarzanie dla najmniejszej instancji wykazywało większą zmienność wyników.