Projektowanie Efektywnych Algorytmow

Projekt

21/12/2021

252690 Karolina Nogacka

(5) Tabu search

spis treści	strona

1. **Sformułowanie zadania:**

   Zadanie polega na stworzeniu algorytmu znajdującego rozwiązanie problemu komiwojażera opartego o metodę poszukiwania z zakazami.

   Problem komiwojażera (TSP - Traveling salesman problem):

   Zakładając, że ktoś (np. kurier, sprzedawca, listonosz) ma zbiór miast/domów do odwiedzenia szukamy drogi, która zawierać będzie wszystkie wyżej wymienione miejsca i będzie drogą najkrótszą (aby listonosz nie musiał się nachodzić).

   Innymi słowy problem ten jest zagadnieniem optymalizacyjnym , polegającym na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Wszystkich cykli w takim grafie będzie $n!$ (w związku z czym złożoność dla większych grafów będzie szybko rosła), pewne z nich będą w rzeczywistości tymi samymi cyklami, zaczynającymi się jednak w różnych węzłach grafu.

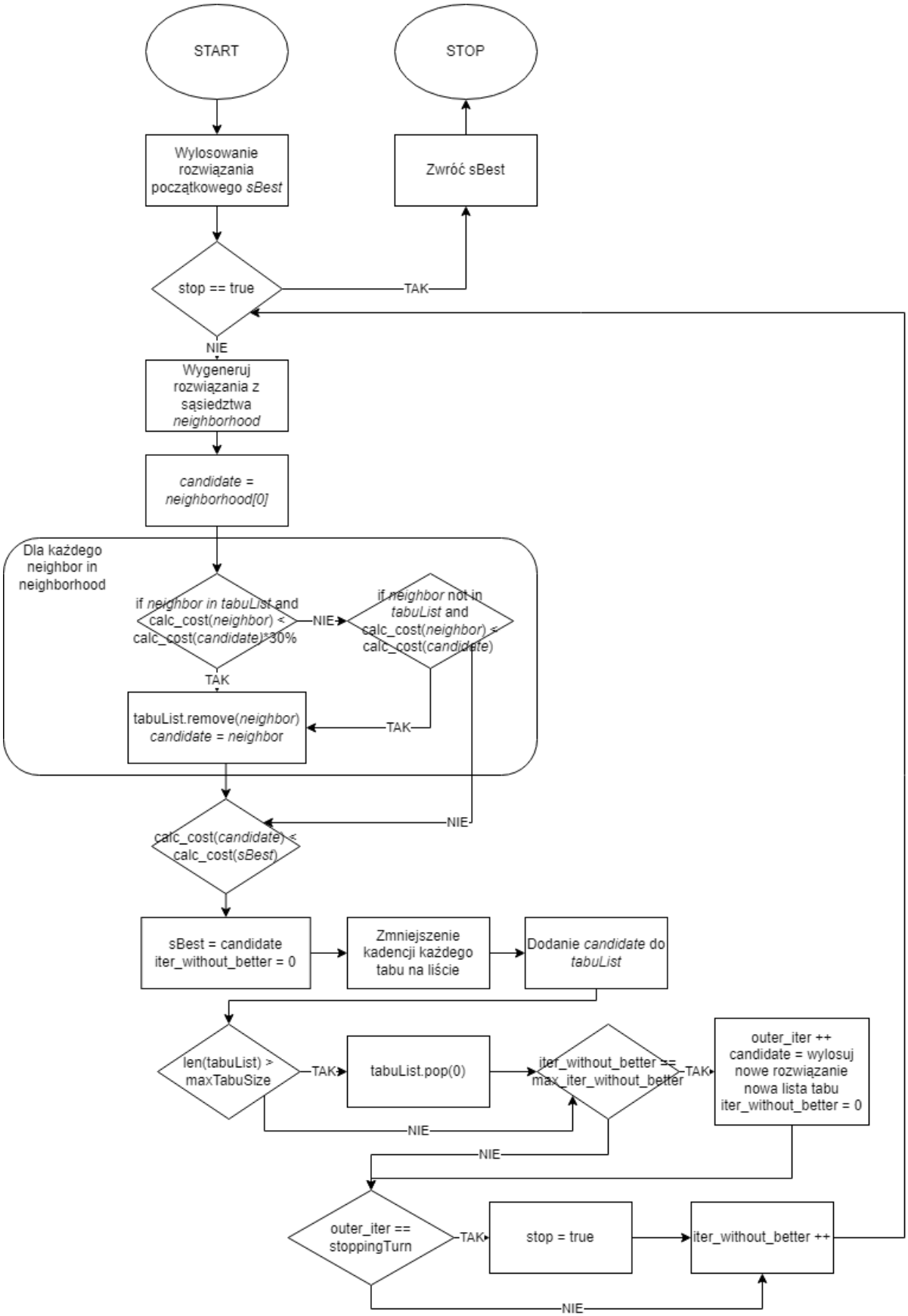
2. Metoda:

Metoda tabu search to metaheurystyka stosowana do rozwiązywania problemów optymalizacyjnych. Jej podstawową ideą jest przeszukiwanie przestrzeni, stworzonej ze wszystkich możliwych rozwiązań, za pomocą sekwencji ruchów. Istnieją ruchy niedozwolone, dodane do listy tabu. Każdy ruch znajduje się na liście pewną kadencję, która z każdą iteracją maleje, aby przy kadencji równej 0 usunąć ruch z listy tabu.

Metoda nie zapewnia znalezienia zawsze rozwiązania dokładnego.

Spodziewanym jest uzyskanie wyników dla obszerniejszych instancji niż w przypadku algorytmów opartych o: brute force, programowanie dynamiczne, czy bxb, lecz niestety wyników niedokładnych, obarczonych pewnym błędem.

3. Algorytmy:



Rysunek 1. Schemat blokowy algorytmu znajdującego rozwiązanie problemu komiwojażera, opartego na metodzie tabu search.

4. Dane wejściowe i wyjściowe

Co zawiera program badawczy:

- Plik wykonywalny tsp_ts.exe
  Aby działał poprawnie musi być w jednym katalogu z plikiem config.ini i pliki tekstowe z grafami.

- Pliki tsp_6_1.txt, tsp_6_2.txt, tsp_10.txt, tsp_12.txt, itd.
  Zawierają grafy w postaci macierzy sąsiedztwa, które będziemy badać. Źródło: http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/ [data dostępu: 16.12.2021, 17:48]

- Pliki gr21.tsp, ulysses22.tsp, br17.atsp, itd., wszystkie wypisane w wymaganiach projektu.
  Zawierają grafy w postaci macierzy sąsiedztwa, które będziemy badać. Źródło: http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/pea.php [data dostępu: 16.12.2021, 17:54]

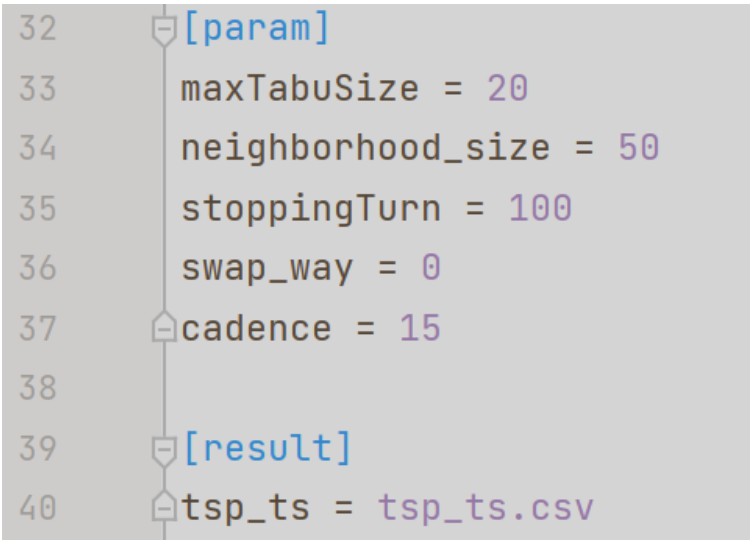
- Plik inicjujący config.ini

```
[data]
tsp_6_1 = tsp_6_1.txt; 132; [0, 1, 2, 3, 4, 5, 0]
tsp_6_2 = tsp_6_2.txt; 80; [0, 5, 1, 2, 3, 4, 0]
tsp_10 = tsp_10.txt; 212; [0, 3, 4, 2, 8, 7, 6, 9, 1, 5, 0]
tsp_12 = tsp_12.txt; 264; [0, 1, 8, 4, 6, 2, 11, 9, 7, 5, 3, 10, 0]
tsp_13 = tsp_13.txt; 269; [0, 10, 3, 5, 7, 9, 11, 2, 6, 4, 8, 1, 12, 0]
tsp_14 = tsp_14.txt; 282; [0, 10, 3, 5, 7, 9, 13, 11, 2, 6, 4, 8, 1, 12, 0]
tsp_15 = tsp_15.txt; 291; [0, 12, 1, 14, 8, 4, 6, 2, 11, 13, 9, 7, 5, 3, 10, 0]
tsp_17 = tsp_17.txt; 39; [0, 11, 13, 2, 9, 10, 1, 12, 15, 14, 5, 6, 3, 4, 7, 8, 16, 0]

gr21 = gr21.tsp; 2707; []
ulysses22 = ulysses22.tsp; 7013; []
gr24 = gr24.tsp; 1272; []
fri26 = fri26.tsp; 937; []
bays29 = bays29.tsp; 2020; []
att48 = att48.tsp; 10628; []
eil51 = eil51.tsp; 426; []
berlin52 = berlin52.tsp; 7542; []
br17 = br17.atsp; 39; []
ftv33 = ftv33.atsp; 1286; []
ftv35 = ftv35.atsp; 1473; []
ftv38 = ftv38.atsp; 1530; []
p43 = p43.atsp; 5620; []

gr96 = gr96.tsp; 55209; []
kroA100 = kroA100.tsp; 21282; []
kroB150 = kroB150.tsp; 26130; []
pr152 = pr152.tsp; 73682; []
ftv170 = ftv170.atsp; 2755; []
rbg323 = rbg323.atsp; 1326; []
```

Rysunek 2. Zawartość pliku config.ini.



Rysunek 3. Zawartość pliku config.ini c.d.

Pierwsze zdjęcie zawiera nazwy plików ze strony http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/, http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/pea.php oraz optymalne rozwiązania dla grafów w nich zawartych. Drugie zdjęcie zawiera parametry algorytmu oraz plik nazwę pliku wynikowego.

Sekcja [data] zawiera nazwę plików wejściowych z grafami.
Sekcja [param] zawiera parametry algorytmu.
Sekcja [result] zawiera nazwę pliku wyjściowego.

- Skrypty tsp_ts.py, my_writer.py:

Skrypt tsp_sa.py jest głównym plikiem programu, zawiera on wywołania metod z innych plików. my_writer.py zapewnia poprawny zapis do plików wynikowych.

- Plik tsp_ts_out-analiza.csv:

Zawiera zbiorcze dane i wykresy.

5. Procedura badawcza

- **Kolejność wykonywanych badań:**
    - Pobieranie danych z pliku inicjującego,
    - Wczytanie grafów z plików wejściowych,
    - Uruchomienie badań dla grafów.

Wynikiem działania programu (zapisywanym do pliku) jest czas znalezienia rozwiązania dla każdej instancji, długość najkrótszego cyklu, oraz sam cykl (w postaci listy węzłów).

Wyniki zostały opracowane po 20-krotnym uruchomieniu programu (algorytm wykonał się 20 razy dla każdej instancji). Jako górną granicę czasową dla algorytmu przyjęto godzinę działania całego programu oraz maksymalnie godzina pracy nad jedną instancją.

Algorytm był w stanie policzyć wyniki dla każdej zadanej instancji w zadanym czasie. Wyniki czasowe jednak znacznie różnią się w zależności od zadanych parametrów.

Specyfikacja sprzętu:

a. Procesor Intel i7-10510U, 1.80GHz – 2.30 GHz
b. 16,0 GB pamięci ram
c. System Windows 10 Home Edition

- **Metoda badania zużycia czasu:**

```
253    work(matrix: instance):
254    print("Liczę dla: " + matrix.name)
255
256    start_time = time.time()
257    path, cost = tabu_search(matrix.matrix)
258    end_time = time.time() - start_time
259
260    tsp_result = [str(matrix.name), int(matrix.length), end_time, cost, path]
261
262    # return tsp_result, tsp_ts_params
263    return tsp_result
```

Rysunek 4. Metoda pomiaru czasu.

W funkcji *work()* przed i po uruchomieniu funkcji *tabu_search()* mierzony jest czas za pomocą funkcji z biblioteki python'a *time.* Różnica zapisywana jest do pliku wyjściowego jako czas wykonania algorytmu dla zadanej instancji.

Wybór parametrów:

5.1. Rozmiar listy tabu (dwa sposoby):

- Taki sam dla każdej instancji (zadany w pliku .ini),
- maxTabuSize = 3*n (według Knox'a Tabu Search Performance on the Symmetric Traveling Salesman Problem. Computer Operations Research, Vol. 21, No. 8, pp. 867-876.).

5.2. Kryterium aspiracji:

- Jeśli ruch znajduje się na liście tabu ale gwarantuje rozwiązanie lepsze o 30% od aktualnego, to usunięty zostanie z listy tabu i zostanie wzięty pod uwagę jako kandydat na nowe najlepsze rozwiązanie. Kryterium 30% ma zapobiec zbyt częstemu usuwaniu rozwiązań z listy tabu.

5.3. Kryterium zakończenia (dwa pierwsze zamienne):

- Wykonano maksymalną ilość dozwolonych iteracji (*stoppingTurn* z pliku .ini),
- Wykonano maksymalną ilość dozwolonych iteracji (*stoppingTurn* obliczone programowo jako dziesięciokrotność wielkości instancji),
- Minął dozwolony czas 1h.

5.4. Długość kadencji:

- Taka sama dla każdej instancji (*cadence* zadana w pliku .ini),
- Obliczona programowo jako *cadence* = 20% ** n.*

5.5. Sposób wyboru rozwiązania w sąsiedztwie obecnego:

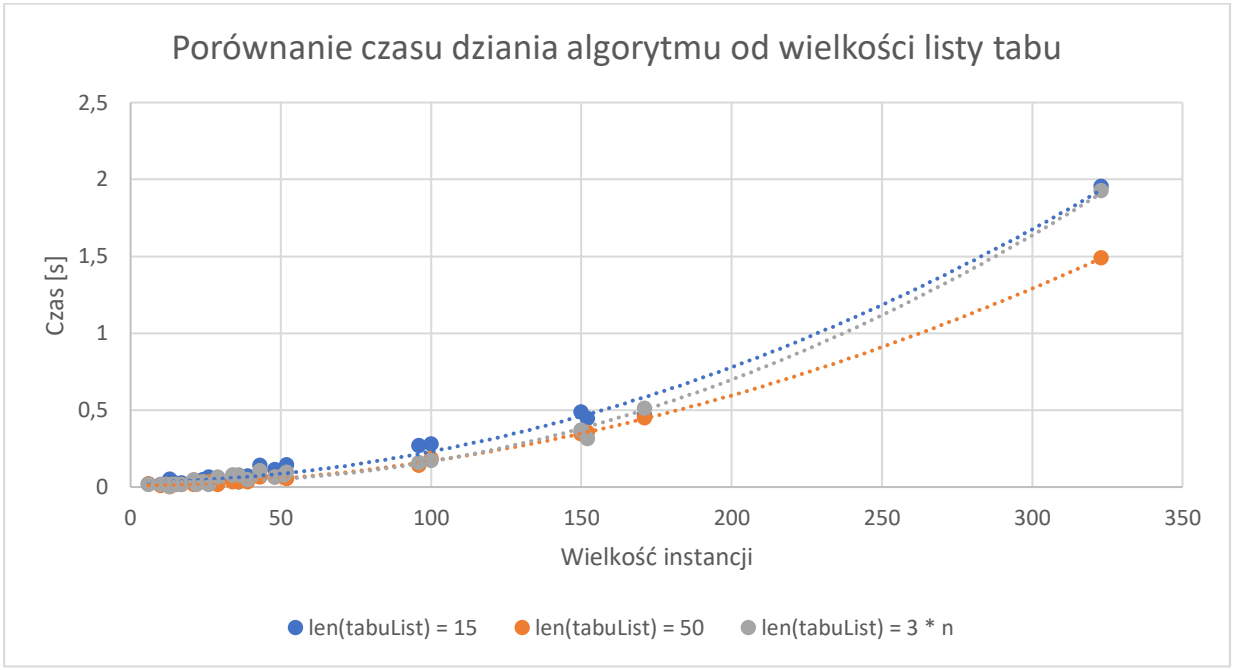
- 2-zamiana (swap_way = 0 w pliku .ini),
- Zamiana dwóch części tablicy (swap_way = 1 w pliku .ini),
- Invert (swap_way = 2 w pliku .ini),
- Hill climbing (swap_way = 3 w pliku .ini).

5.6. Zdarzenia krytyczne (dwa warianty):

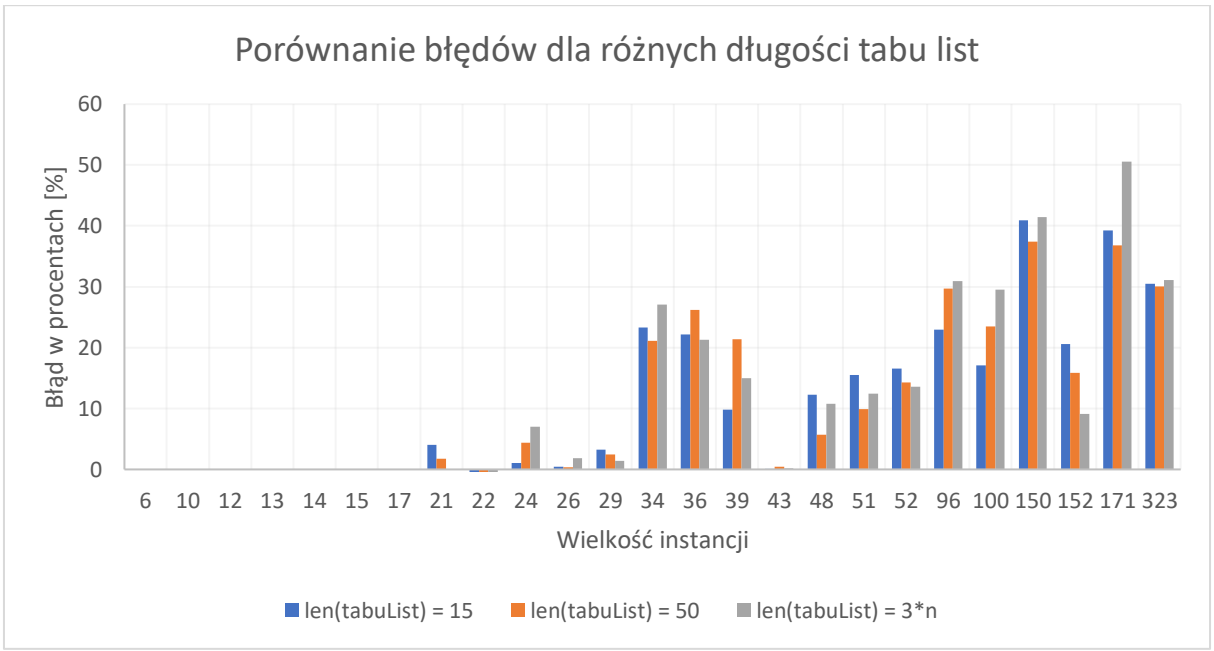
- Brak obsługi zdarzeń krytycznych,
- Jeśli wewnętrzny licznik pętli (ten dla pracy w pewnym sąsiedztwie), współgrał będzie z licznikiem braku poprawy rozwiązania (nastąpi stagnacja, która dojdzie do zadanego maksimum), wylosowane zostanie nowe rozwiązanie, które będzie potencjalnie lepsze od aktualnego. Ma to zapobiec utknięciom w minimach lokalnych.

6. Wyniki

Porównanie dla długości listy tabu:

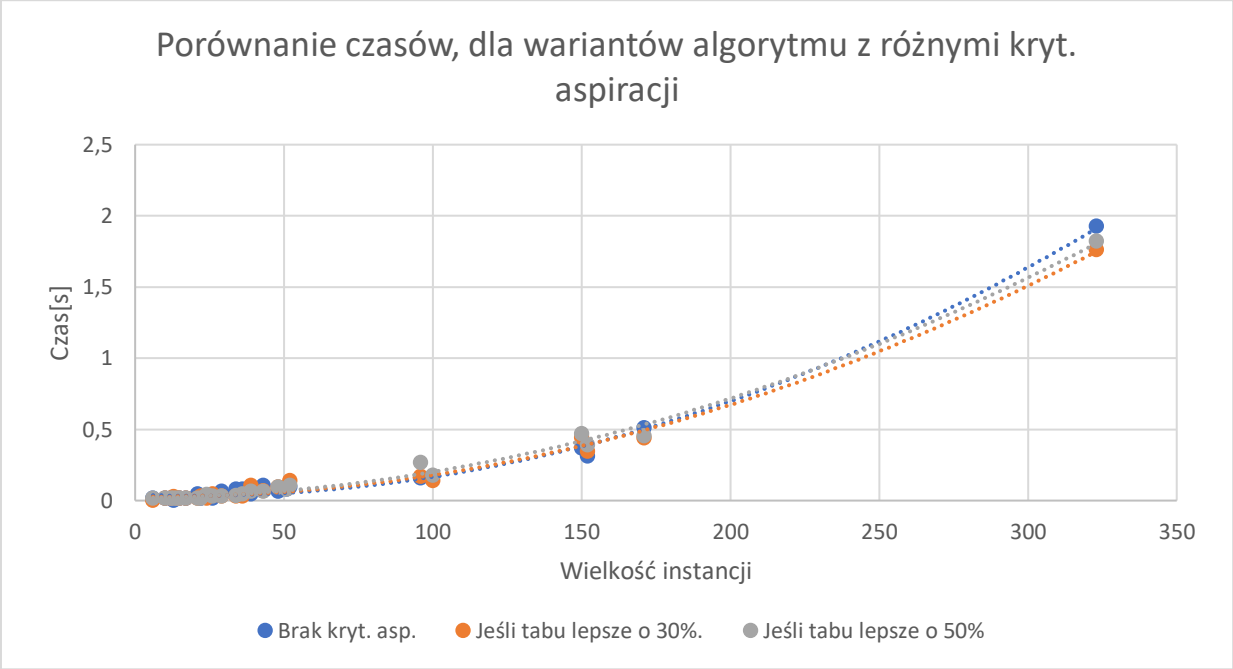


Rysunek 5. Porównanie czasu działania w zależności od długości listy tabu. Iteracje = 50, sposób generacji roz. w sąs. 2-zamiana, kadencji 15.

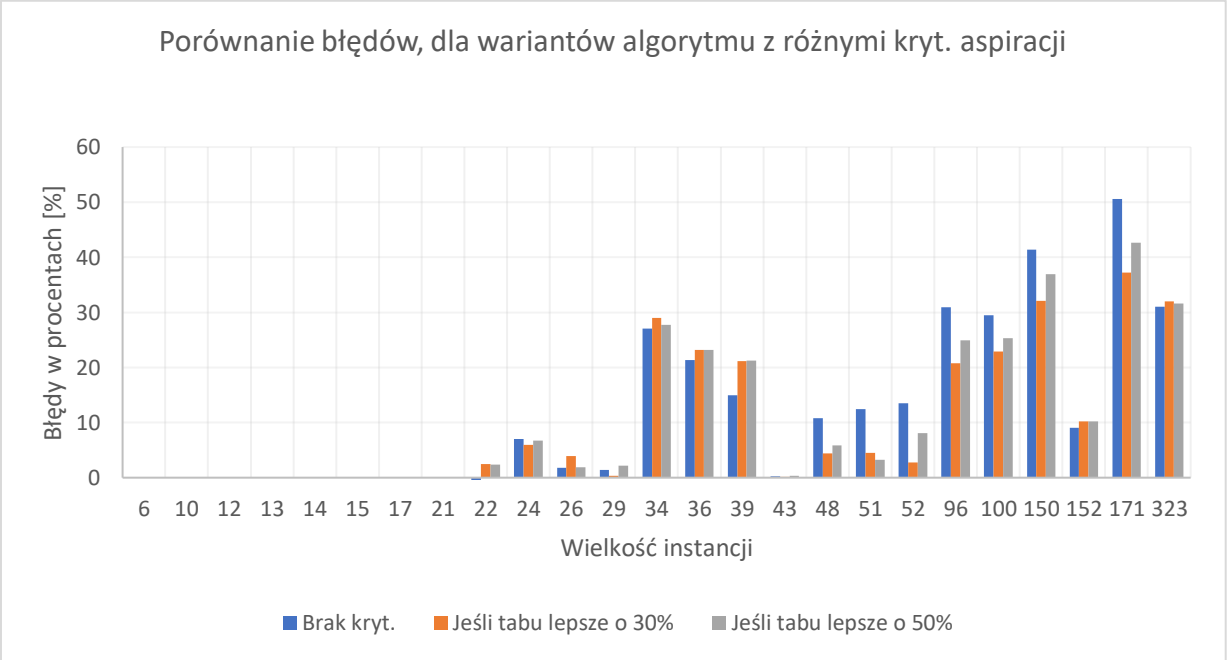


Rysunek 6. Porównanie błędu w zależności od długości listy tabu. Iteracje = 50, sposób generacji roz. w sąs. 2-zamiana, kadencji 15.

Porównanie po wprowadzeniu kryterium aspiracji:

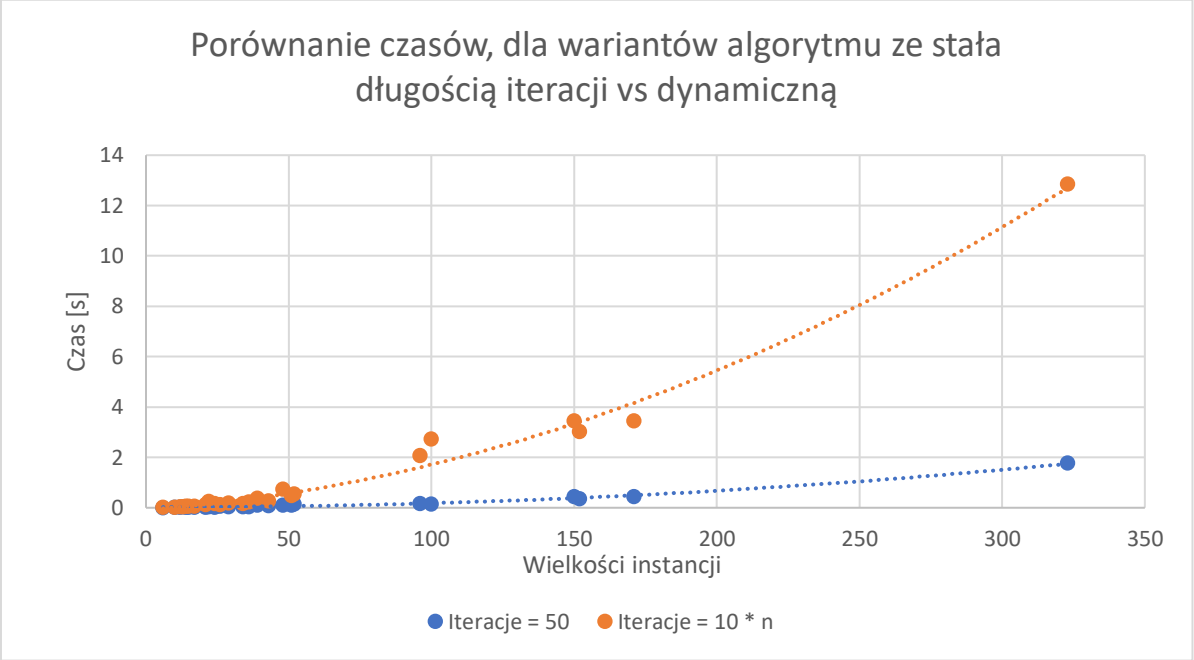


*Rysunek 7. Porównanie czasu działania w zależności od kryt. aspiracji. Iteracje = 50, sposób generacji roz. w sąs. 2-zamiana, kadencji 15, długość listy tabu = 3*n.*

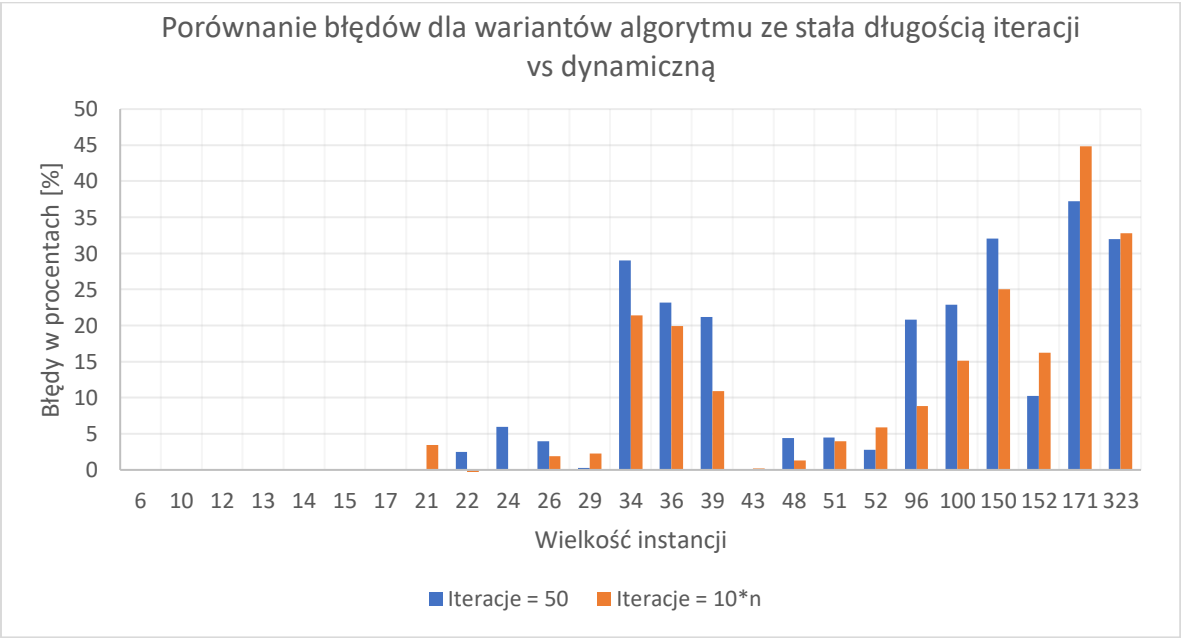


*Rysunek 8. Porównanie błędu w zależności od kryt. aspiracji. Iteracje = 50, sposób generacji roz. w sąs. 2-zamiana, kadencji 15, długość listy tabu = 3*n.*

Porównanie dla różnych długości iteracji:

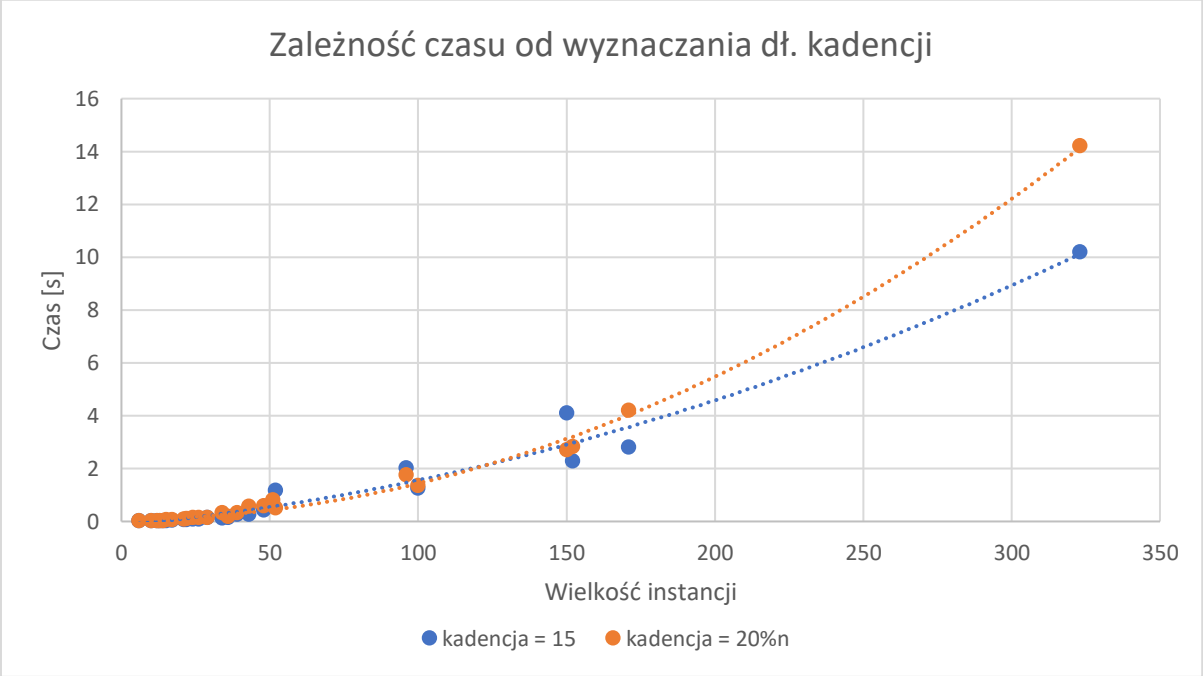


*Rysunek 9. Porównanie czasu działania w zależności od liczby iteracji. Iteracje = 50, sposób generacji roz. w sąs. 2-zamiana, kadencji 15, długość listy tabu = 3*n, kryt. asp. = 30%.*

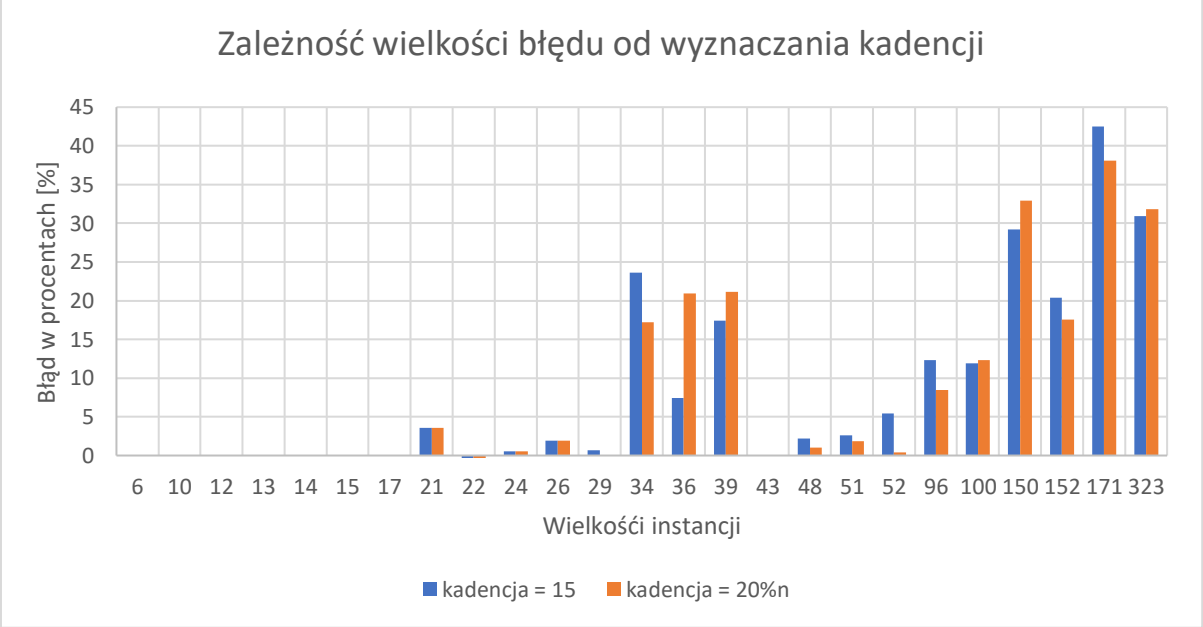


*Rysunek 10. Porównanie wielkości błędu w zależności od liczby iteracji. Iteracje = 50, sposób generacji roz. w sąs. 2-zamiana, kadencji 15, długość listy tabu = 3*n, kryt. asp. = 30%.*

Porównanie dla różnych sposobów wyznaczania długości kadencji:

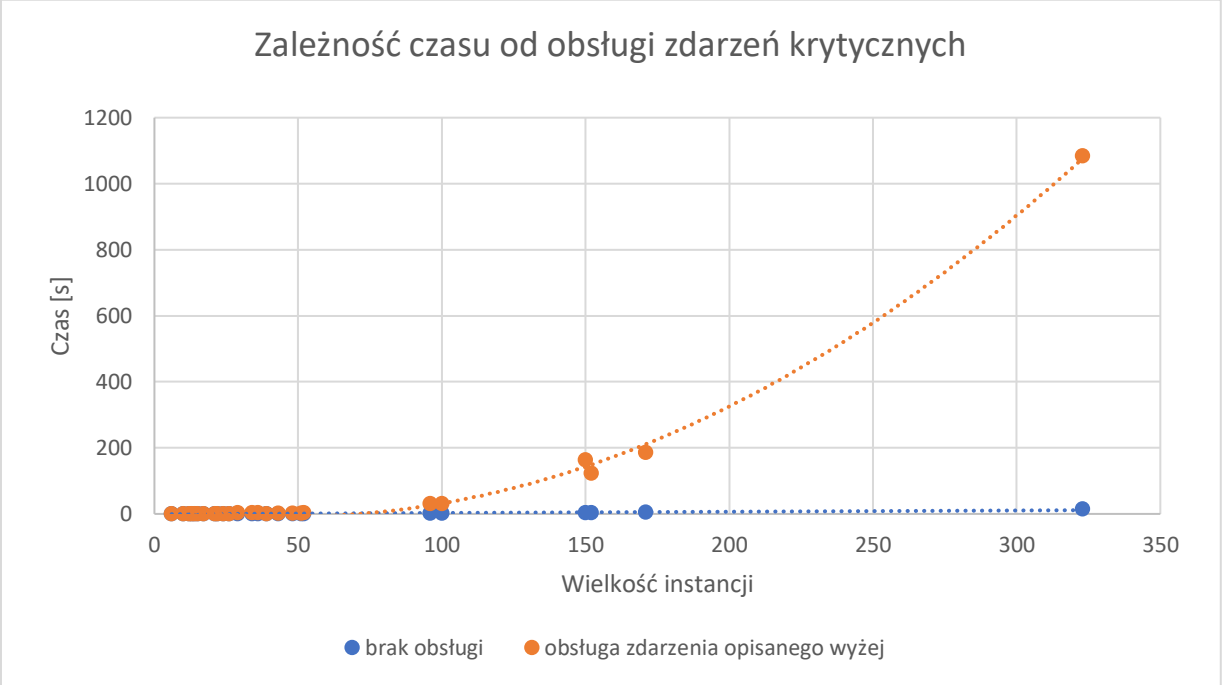


*Rysunek 11. Porównanie czasu działania w zależności od liczby kadencji. Iteracje = 50, sposób generacji roz. w sąs. 2-zamiana, kadencji 15, długość listy tabu = 3*n, kryt. asp. = 30%, iteracje = 10n.*

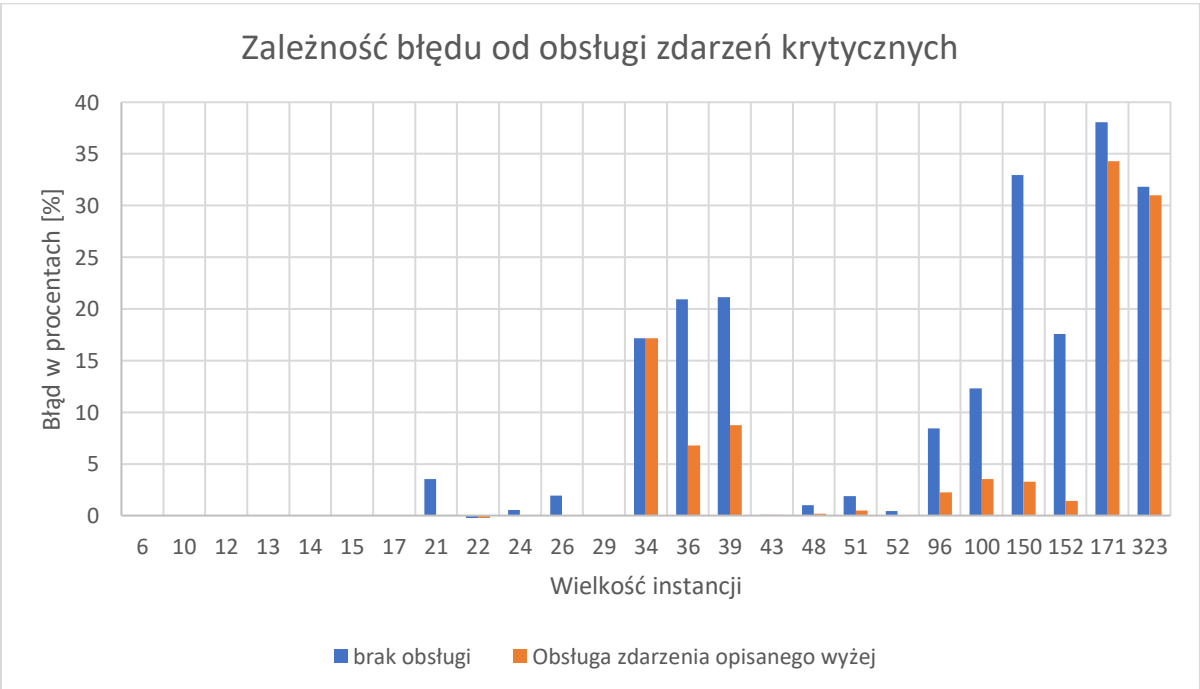


*Rysunek 12. Porównanie wielkości błędów w zależności od liczby kadencji. Iteracje = 50, sposób generacji roz. w sąs. 2-zamiana, kadencji 15, długość listy tabu = 3*n, kryt. asp. = 30%, iteracje = 10n.*

Porównanie dla obsługi zdarzenia krytycznego:



*Rysunek 13. Porównanie czasu działania dla obsługi zdarzenia krytycznego. Iteracje = 50, sposób generacji roz. w sąs. 2-zamiana, kadencji 15, długość listy tabu = 3*n, kryt. asp. = 30%, iteracje = 10n, kadencje = 20%n.*



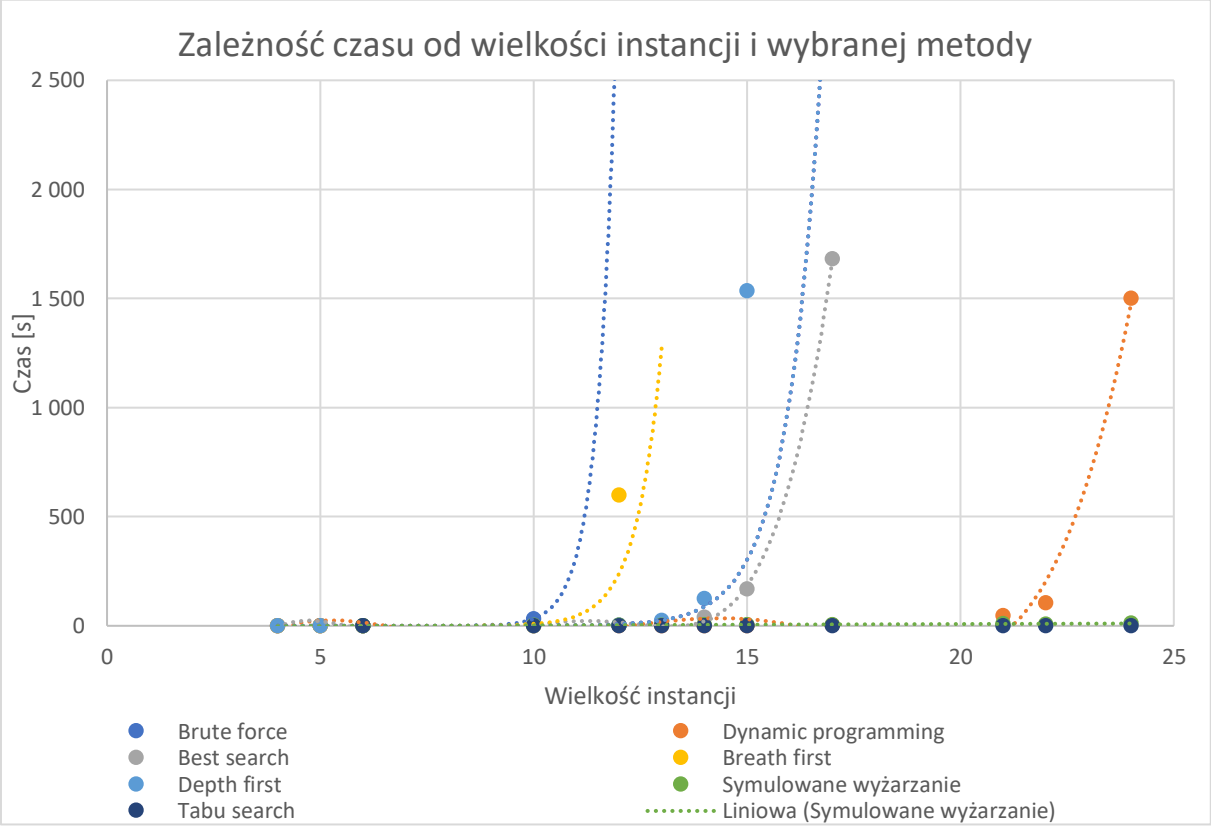
*Rysunek 14. Porównanie błędu dla obsługi zdarzenia krytycznego. Iteracje = 50, sposób generacji roz. w sąs. 2-zamiana, kadencji 15, długość listy tabu = 3*n, kryt. asp. = 30%, iteracje = 10n, kadencje = 20%n.*

Porównanie działania algorytmu opartego o metodę tabu search z innymi:

Porównywane są najlepsze uzyskane wyniki tabu search.

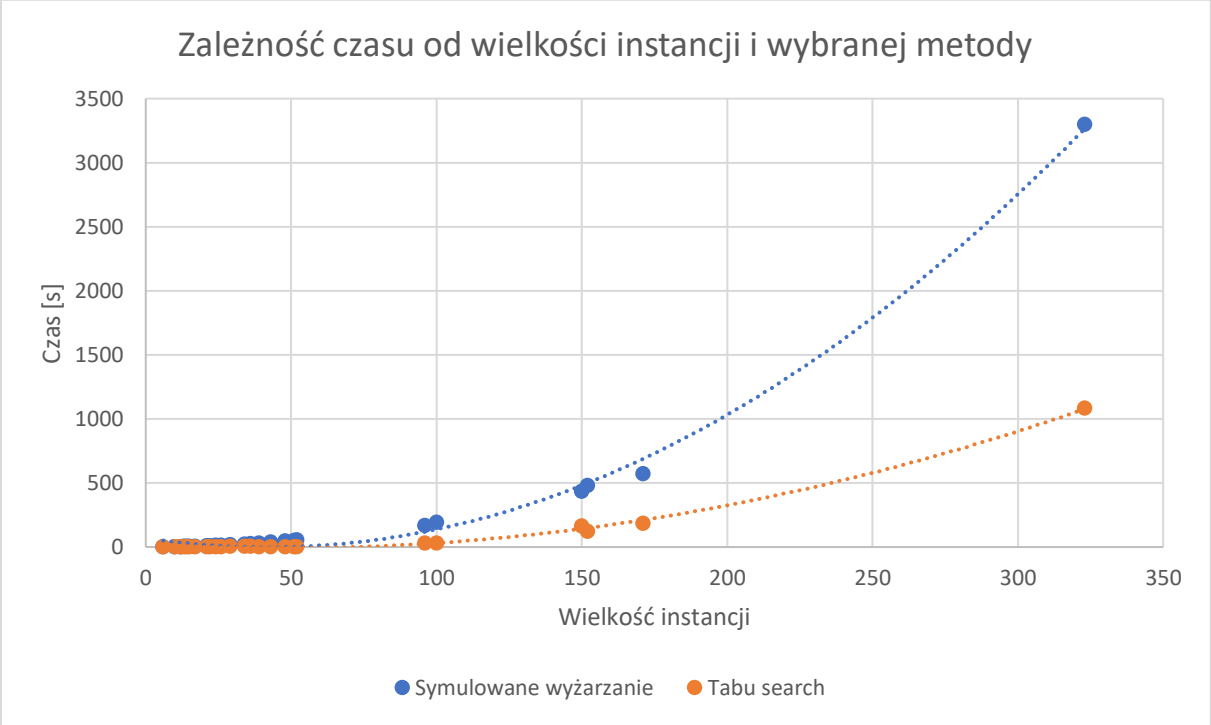
Takie wyniki uzyskano dla parametrów:

- Długość listy tabu: 3*n,*
- Liczba iteracji: 100,
- Sposób generowania rozw. w sąsiedztwie: 2-zamiana,
- Długość kadencji: 20%*n*.

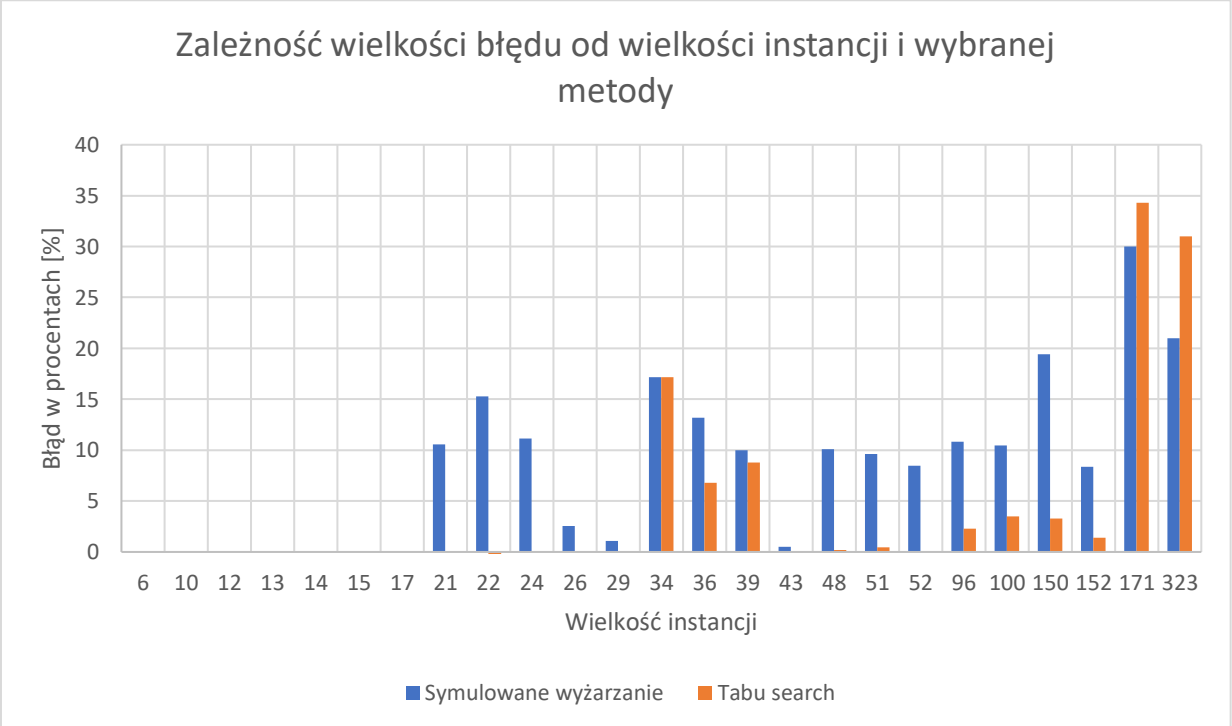


Rysunek 15. Porównanie czasowe tabu search z innymi metodami.

Porównanie działania algorytmu opartego o metodę tabu search z symulowanym wyżarzaniem:



Rysunek 16. Porównanie czasu działania tabu search i symulowanego wyżarzania. Dla obu metod wybrano wartości parametrów dające najlepsze wyniki.



Rysunek 17. Porównanie błędów tabu search i symulowanego wyżarzania. Dla obu metod wybrano wartości parametrów dające najlepsze wyniki.

7. Analiza wyników i wnioski

Zgodnie z założeniami, za pomocą algorytmu opartego o metodę *tabu search* udało się otrzymać wyniki dla większych instancji (TSP) niż w przypadku algorytmów opartych o metody z poprzednich etapów projektu (*brute force, programowanie dynamiczne, bxb*), mając do dyspozycji takie same zasoby sprzętowe i taki sam czas (*Strona 14*) (*Rysunek 15*).

Porównując najlepsze wyniki (czyli co za tym idzie dla parametrów dających najlepsze wyniki) dla algorytmu opartego na *symulowanym wyżarzaniu* i dla algorytmu opartego o *tabu search* (*Strona 15*) (*Rysunki 16, 17*), wersja *tabu search* okazała się być ponad dwukrotnie szybsza. Analizując wartość błędów dla wszystkich zbadanych instancji (*Rysunek 17*), można zauważyć że sumarycznie algorytm z symulowanym wyżarzaniem uzyskał większy błąd. Niemalże dla każdej instancji (pomijając wielkość 34, 39, 171 oraz 323), wartość błędu dla *symulowanego wyżarzania* była kilkukrotnie większa niż dla *tabu search*. Natomiast przy dwóch największych instancjach (wielkość 171 i 323), widać odwrotny trend. Wartość błędu dla *tabu search* jest większa niż dla *symulowanego wyżarzania*. Może to być spowodowane specyficznym ukształtowaniem tych instancji lub na przykład dobranymi parametrami. Istnieje możliwość, że dla jeszcze większych instancji trend ten utrzymałby się. Nie można więc wyciągnąć jednoznacznych wniosków, która metoda gwarantuje mniejszy błąd wyniku.

Analizując wpływ długości listy tabu na działanie algorytmu (*Strona 9*) (*Rysunki 5, 6*), można wysnuć wniosek, że jest to mocno powiązane z konkretną instancją. Nie można dostrzec żadnego konkretnego trendu na wykresie błędów.

Dodanie kryteriów aspiracji do algorytmu, zmieniło jego czas wykonywania nieznacznie (*Strona 10*) (*Rysunek 7*). Dla instancji większych od 40, widać poprawę wyników (*Rysunek 8*), jeśli zastosowane są kryteria aspiracji. Najbardziej obiecujące wydało się kryterium: „Jeżeli rozwiązanie na liście tabu jest lepsze od aktualnego o 30%, to usuń je z listy tabu", dlatego właśnie ono zostało wykorzystane w dalszych badaniach algorytmu.

Zmieniając dobór liczby iteracji, można dostrzec (*Strona 11*) (*Rysunek 9*), że jeśli liczba iteracji jest stała, to czas rośnie liniowo (wraz z wielkością instancji). Jeśli zaś liczba iteracji zależna jest od *n* (samej wielkości instancji), to czas rośnie szybciej, krzywoliniowo. Jest to zachowanie zgodne z intuicją oraz założeniami. Dla instancji mniejszych od 150, widać lepszą jakość wyników (*Rysunek 10*), jeśli ilość iteracji uzależniona jest od wielkości instancji (*iteracje* = 10*n*).

Została zbadana stała i jednakowa dla każdej instancji długość kadencji (15) oraz długość zależna od wielkości instancji (20%*n*) (*Strona 12*) (*Rysunki 11, 12*). Dla instancji do wielkości 96, długość kadencji wyznaczana drugim sposobem była krótsza niż 15, od instancji wielkości 96 drugi sposób wyznaczania kadencji daje większe wartości niż 15. Dla instancji wielkości 36 możemy dostrzec dużo większy błąd dla *kadencji* = 20%*n* ≈ *7,* niż dla *kadencji* = 15. Może być to spowodowane wpadnięciem w cykl w okolicach minimum lokalnego (zwiększona intensyfikacja). Nie można wyciągnąć jednoznacznych wniosków co do tego, który sposób wyboru długości kadencji jest lepszy. Jeśli kadencja jest krótka, intensyfikacja przeszukiwań rośnie, jeśli kadencja jest dłuższa rośnie dywersyfikacja (mniejsze ryzyka na wpadnięcie w minimum lokalne, jednak co za tym idzie mniej dokładne przeszukiwanie sąsiedztwa, co może spowodować ominięcie obiecującego rozwiązania). Sposób wyboru zależy, w dużej mierze od oczekiwań i wymagań stawianych algorytmowi.

Zbadano wpływ obsługi zdarzenia krytycznego, jakim jest stagnacja z pewnym rozwiązaniem, którego od pewnej liczby iteracji algorytm nie poprawił (pewna liczba iteracji tutaj: wartość zadana w pliku .ini *stoppingTurn*, czyli maksymalna liczba instancji bez poprawy rozwiązania). Jeśli takie wydarzenie ma

miejsce, losowane jest zupełnie nowe rozwiązanie (albo pierwszy jego wierzchołek, a sąsiedzi ustalani są wedle zasady „najbliższego sąsiada”, metoda wyboru rozwiązania taka sama jak w raporcie *symulowane wyżarzanie*). Taka czynność zapobiega pozostaniu w ewentualnych minimach lokalnych i wzmaga dywersyfikację przeszukiwań. Znacznie wydłużyła ona czas badania instancji (*Strona 13*) (*Rysunek 13*), widać również znaczną poprawę jakości rozwiązań (szczególnie dla instancji do wielkości ok 170 węzłów). Jest to bardzo obiecujący wynik badania. Jednak implementacja w dużej mierze zależy od założeń i wymagań stawianych algorytmowi. Wszystko zależy od tego co jest bardziej pożądane: krótki czas wykonania czy dokładność wyników.