Projektowanie Efektywnych Algorytmow

Projekt

18/01/2022

252690 Karolina Nogacka

(7) Ant Colony Optimalization

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:

W tym etapie projektu ro rozwiązania problemu komiwojażera skonstruowany został algorytm mrówkowy („*Ant Colony Optimalization*"). Jak sama nazwa wskazuje, sposób ten został zainspirowany zachowaniem mrówek.

Mrówki w nowopoznanej przestrzeni poruszają się w sposób losowy (ich głównym celem jest znajdowanie pożywienia dla kolonii). Z upływem czasu, mrówki poruszają się w sposób bardziej metodyczny, a odpowiadają za to feromony. Mrówka poruszając się zostawia za sobą feromon, jest to jeden z kilku sposobów komunikacji pomiędzy nimi. Im bardziej zadowalająca jest trasa (w przyrodzie: im bardziej obfita w pożywienie jest trasa), tym więcej feromonu pozostawia mrówka dla pozostałych członków kolonii.

W algorytmie opartym o metodę kolonii mrówek, mamy do czynienia ze sztucznymi mrówkami. Jest kilka metod wydzielania feromonów przez takie mrówki (m. in. DAS, QAS, CAS). Z czasem feromon pozostawiony na trasie wyparowuje (tak jak w prawdziwym życiu).

Algorytm oparty o taką metodę składa się więc z następujących kroków:

1. Stworzone zostaje *m* mrówek, dla każdej losowany jest wierzchołek startowy (miasto startowe),
2. Mrówki poruszają się po krawędziach grafu (dysponują listą tabu odwiedzonych wierzchołków),
3. Zawartość feromonu na krawędzi *E(i, j)* w chwili t oznaczana jest przez $\tau_{ij}(t)$,
4. Ilość feromonu w chwili startowej $t = 0$, jest stosunkowo mała i taka sama na każdej krawędzi (sprawi to, że feromon nie będzie istotny w chwili startowej, a mrówki będą poruszały się w losowy sposób),
5. Ilość feromonu na krawędzi może być aktualizowana zaraz po przejściu przez nią mrówki lub np. po przejściu całego cyklu przez mrówkę.

Istotnym elementem algorytmu jest prawdopodobieństwo wyboru kolejnego wierzchołka (miasta) odwiedzonego przez mrówkę:

Prawdopodobieństwo wyboru miasta j przez k-tą mrówkę w mieście i dane jest wzorem:

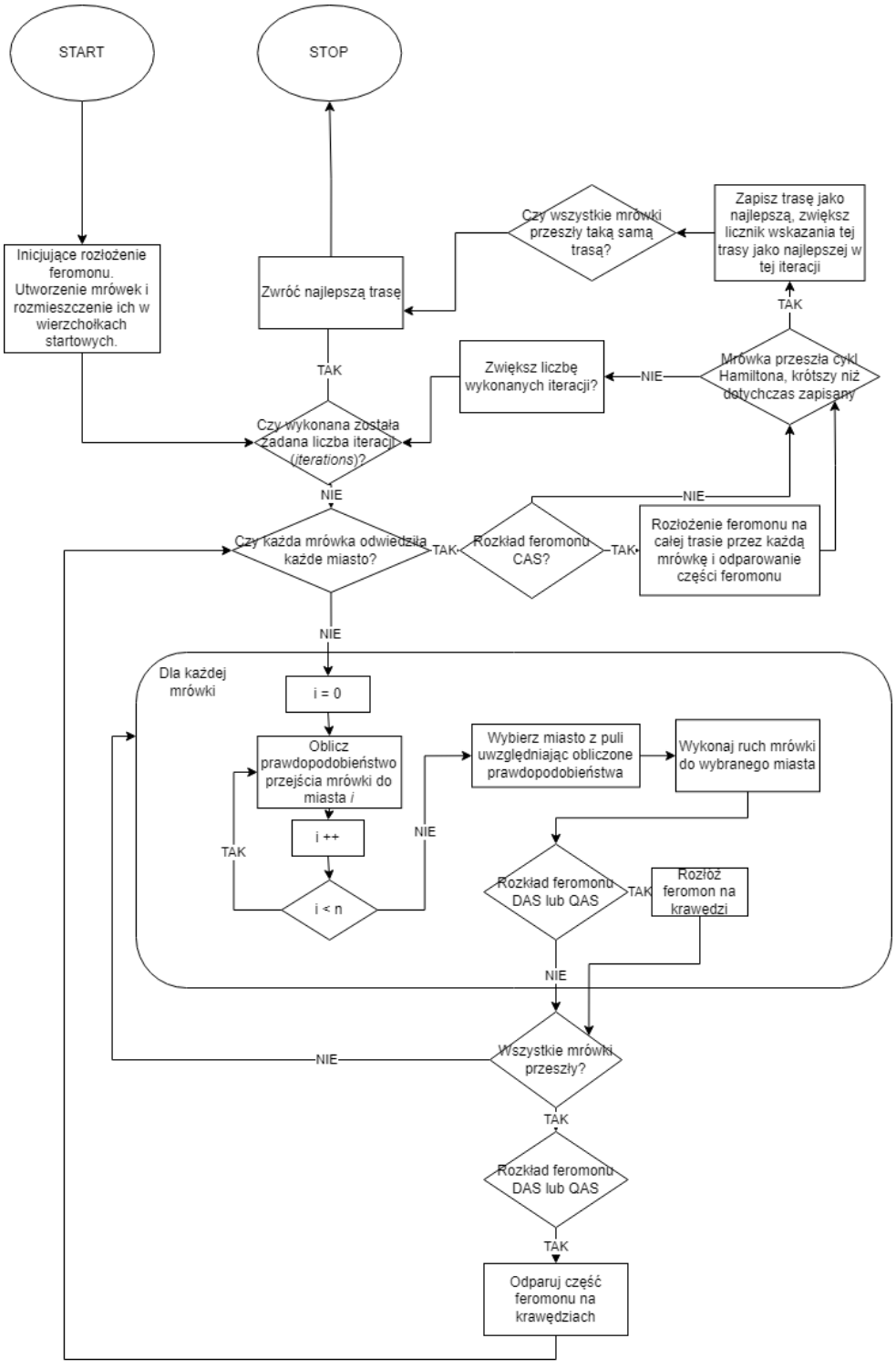
$$p_{ij} = \begin{cases} \dfrac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{c_{i,l} \notin \Omega}(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}} & \bigvee c_{i,l} \in \Omega \\ 0 & \bigvee c_{i,l} \notin \Omega \end{cases}$$

gdzie:
c - kolejne możliwe (nie znajdujące się na liście $tabu_k$ miasto),
Ω - dopuszczalne rozwiązanie (nieodwiedzone miasta, nienależące do $tabu_k$),
η_{ij} -wartość lokalnej funkcji kryterium; np. $\eta = \frac{1}{d_{ij}}$ (*visibility*), czyli odwrotność odległości pomiędzy miastami,
α - parametr regulujący wpływ τ_{ij},
β - parametr regulujący wpływ η_{ij}.

Rysunek 1. Źródło: Prezentacja "ACO", autor: dr. Tomasz Kapłon.

Szacowana złożoność: $\mathbf{O}(\boldsymbol{CC} * \boldsymbol{n^3})$**.** Gdzie CC to liczba iteracji, a n liczba miast.

3. Algorytmy:



Rysunek 2. Schemat blokowy algorytmu znajdującego rozwiązanie problemu komiwojażera, opartego na metodzie kolonii mrówek.

4. Dane wejściowe i wyjściowe

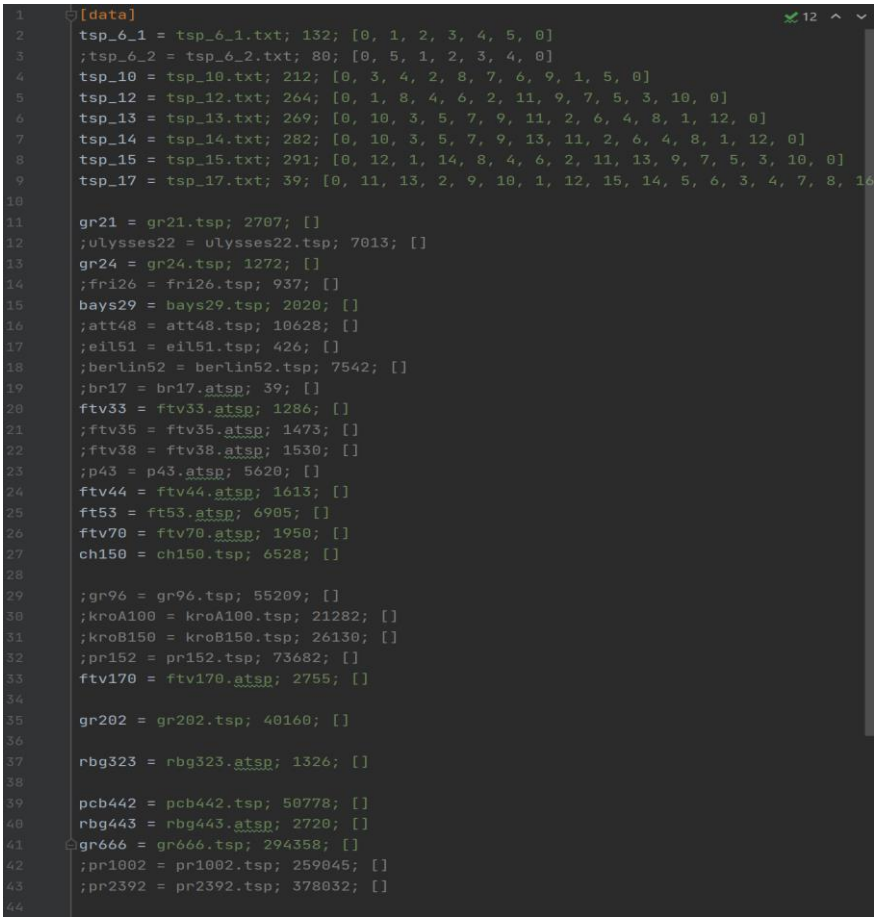
Co zawiera program badawczy:

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

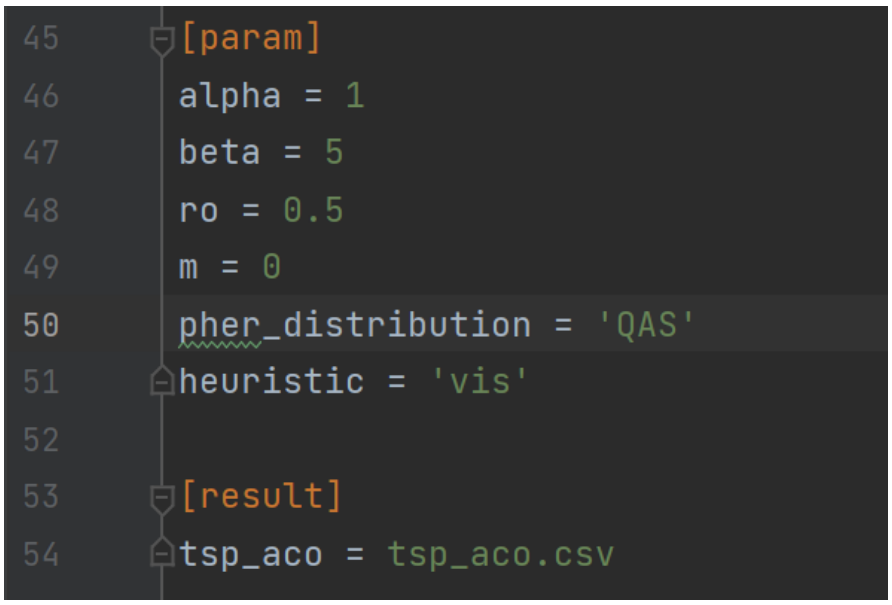
- Pliki tsp_6_1.txt, tsp_6_2.txt, tsp_10.txt, tsp_12.txt, itd.
  Zawierają grafy w postaci macierzy sąsiedztwa, które będziemy badać.
  Źródło: http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/ [data dostępu: 16.12.2021, 17:48]

- 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



Rysunek 3. Zawartość pliku config.ini.



Rysunek 4. Zawartość pliku config.ini c.d., liczba mrówek jest równa liczbie wierzchołków, zadawana z pliki .ini jedynie w celach testowych.

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_aco_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 10-krotnym uruchomieniu programu (algorytm wykonał się 10 razy dla każdej instancji). Jako górną granicę czasową dla algorytmu przyjęto godzinę działania całego programu oraz maksymalnie godzina pracy nad jedną instancją.

Nie znaleziono instancji 600 w postaci odpowiedniej do badań. Własnoręcznie przekonwertowano instancje wielkości powyżej 600, najprawdopodobniej jednak w zły sposób. Wielkość błędu była dla nich ujemna, co sugeruje znalezienie cyklu Hamiltona krótszego niż optymalny. Pod uwagę wzięto więc jedynie czas wykonania algorytmu (powinien on być zbliżony do czasu wykonania gdyby instancja była odpowiednia).

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:**

```
203  def work(matrix: instance):
204      print("Liczę dla: " + matrix.name)
205
206      start_time = time.time()
207      path, cost = aco(matrix)
208      end_time = time.time() - start_time
209
210      tsp_result = [str(matrix.name), int(matrix.length), end_time, cost, path, memory()]
211      print("Name: " + str(tsp_result[0]) + ", size: " + str(tsp_result[1]) + ", time: " + str(end_time) +
212            ", cost: " + str(cost) + ", memory: " + str(memory()))
213
214      # return tsp_result, tsp_ts_params
215      return tsp_result
```

Rysunek 5. Metoda pomiaru czasu.

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

```
31  def memory():
32      w = wmi.WMI('.')
33      result = w.query("SELECT WorkingSet FROM Win32_PerfRawData_PerfProc_Process "
34                       "WHERE IDProcess=%d" % os.getpid())
35      return int(result[0].WorkingSet)
```

Rysunek 6. Pomiar pamięci zabieranej przez uruchomiony proces.

Wybór parametrów:

5.1. Parametr α (im większy tym większa rola feromonu w wyborze kolejnego wierzchołka):

- Sprawdzono wpływ wartość parametru *alpha* jako: 1 (zalecane przez Dorigo), 2, 3.

5.2. Parametr β (im większy tym większa rola heurystyki w wyborze dalszej drogi):

- Sprawdzono wpływ wartość parametru *beta* jako: 2, 3, 5. (Wartości zalecane przez Dorigo: <2, 5>).

5.3. Parametr ρ :

- Dopuszczalne wartości: (0, 1>,
- Testowane wartości: 0.3, 0.5 (zalecane przez Dorigo), 1.0.

5.4. Liczba mrówek *m*:

- Liczba mrówek równa jest liczbie miast.

5.5. Parametr τ_0:

- $\tau_0 = m/C^{nn}$, gdzie C^{nn} to szacowana długość trasy, tutaj średnia kosztu trzech losowych kombinacji wierzchołków.

5.6. Sposób rozkładu feromonu:

- DAS – gęstościowy algorytm rozkładu (stała ilość feromonu na jednostkę długości, aktualizacja feromonu po przejściu krawędzi),
- QAS – ilościowy algorytm rozkładu (stała ilość feromonu dzielona przez długość krawędzi, aktualizacja feromonu po przejściu krawędzi),
- CAS – cykliczny algorytm rozkładu (stała ilość feromonu dzielona przez długość trasy, aktualizacja feromonu po przejściu cyklu).

5.7. Heurystyka wyboru:

- Visability: $\eta = 1/d_{ij}$ (d_{ij} – odległość pomiędzy miastami),
- $\eta = 1 \,/\, d_{ij}^2$. Zainspirowana pracą naukową z Politechniki śląskiej: praca-dr-rutczynska-wdowiak.

5.8. Liczba mrówek:

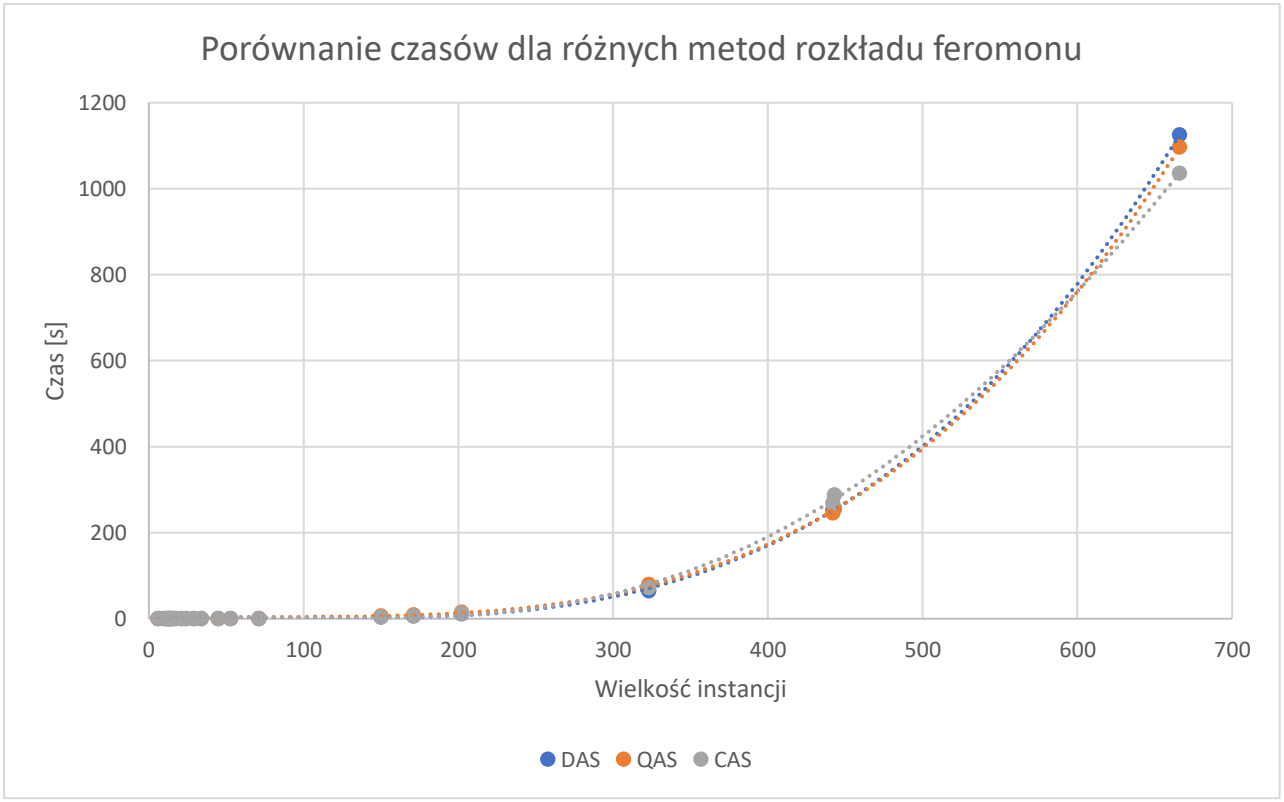
- Równa liczbie miast, *m = n*.

5.9. Warunek końca:

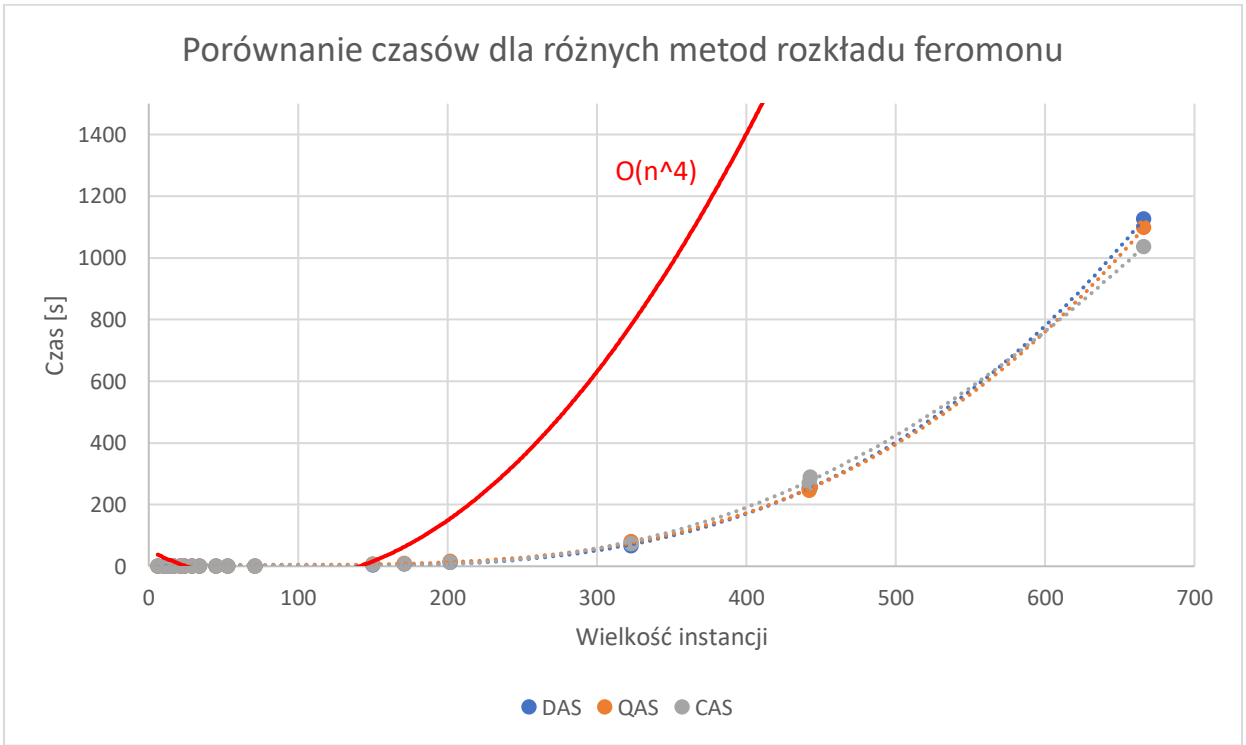
- Wykonano zadaną ilość iteracji (sprawdzano *iteracje = 1, n,* $¼ * n$),
- W kolejnej iteracji każda mrówka wybrała dokładnie taki sam cykl.

6. Wyniki

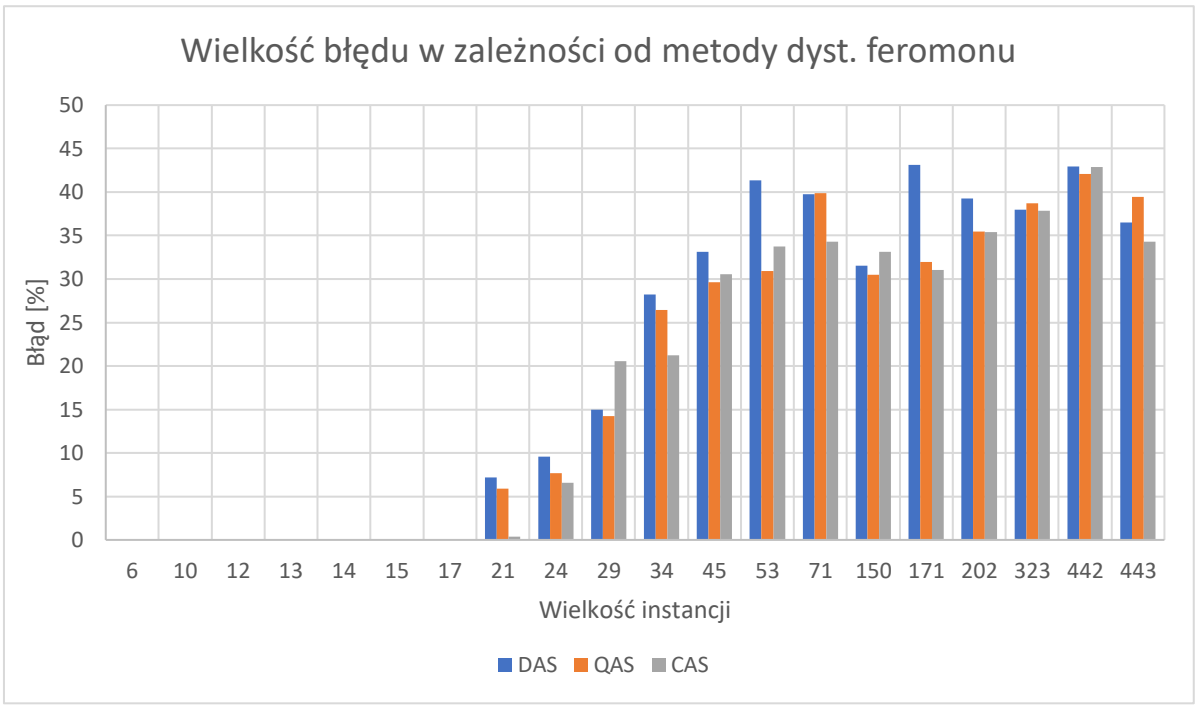
Porównanie dla trzech różnych metod rozkładu feromonu:



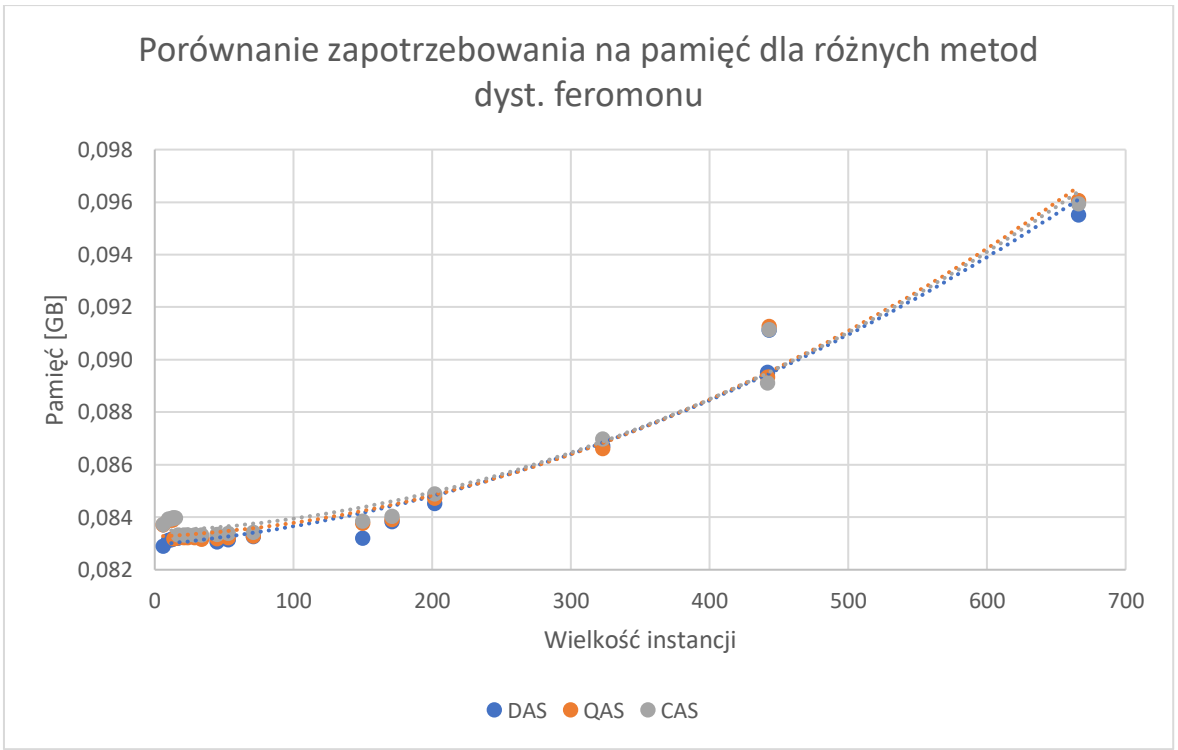
Rysunek 7. Porównanie czasu działania w zależności od sposobu rozkładu feromonu. $\alpha = 1,\ \beta = 5,\ \rho = 0.5, heurystyka = visability.$



Rysunek 8. Górne ograniczenie funkcji czasu wykonywania algorytmu. $\boldsymbol{O(CC * n^3)}$*. Gdzie CC to liczba iteracji, a n liczba miast. W tym badaniu liczba iteracji CC była równa liczbie miast, zatem CC = n, .* $\boldsymbol{O(n^4)}$*.*

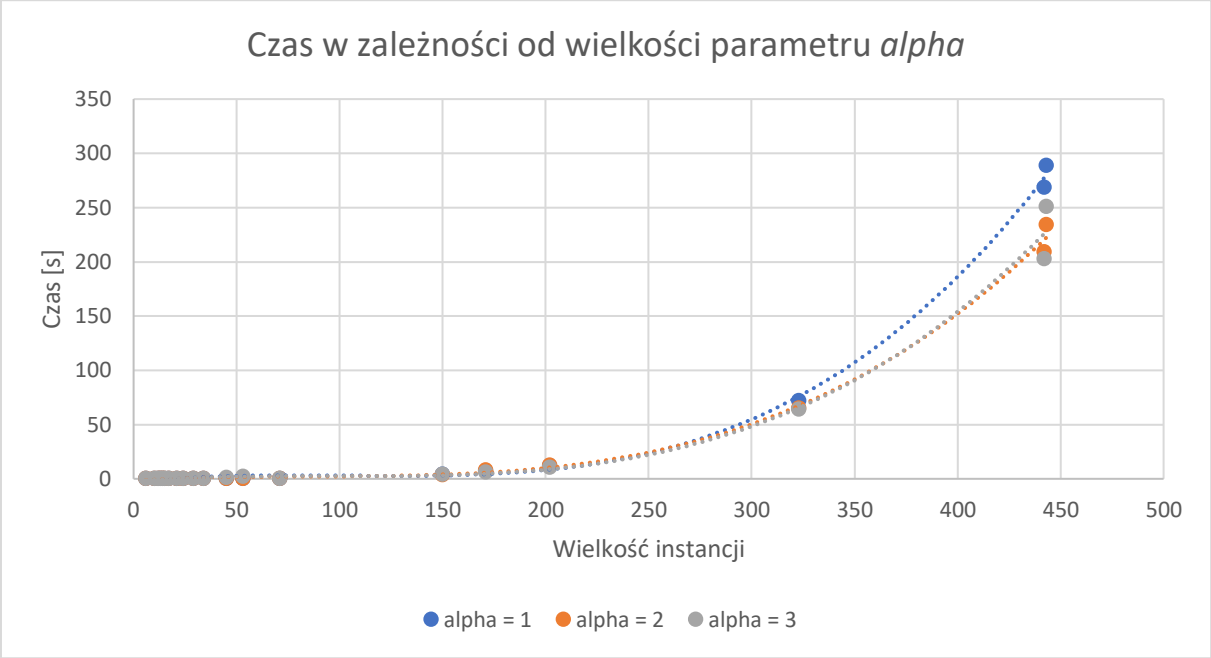


Rysunek 9. Porównanie błędu w zależności od sposobu rozkładu feromonu. $\alpha = 1,\ \beta = 5,\ \rho = 0.5, heurystyka = visability.$

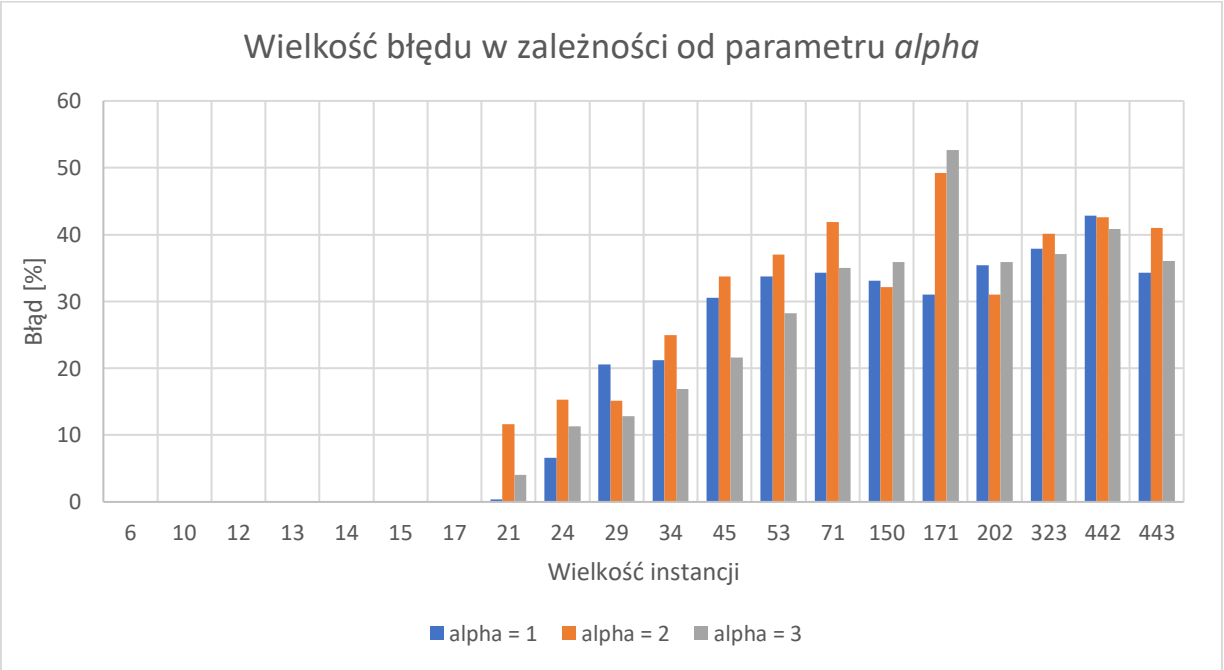


Rysunek 10. Porównanie zapotrzebowania na pamięć dla różnych metod rozkładu feromonu. . $\alpha = 1,\ \beta = 5,\ \rho = 0.5, heurystyka = visability.$

Porównanie dla różnych wartości parametru α:

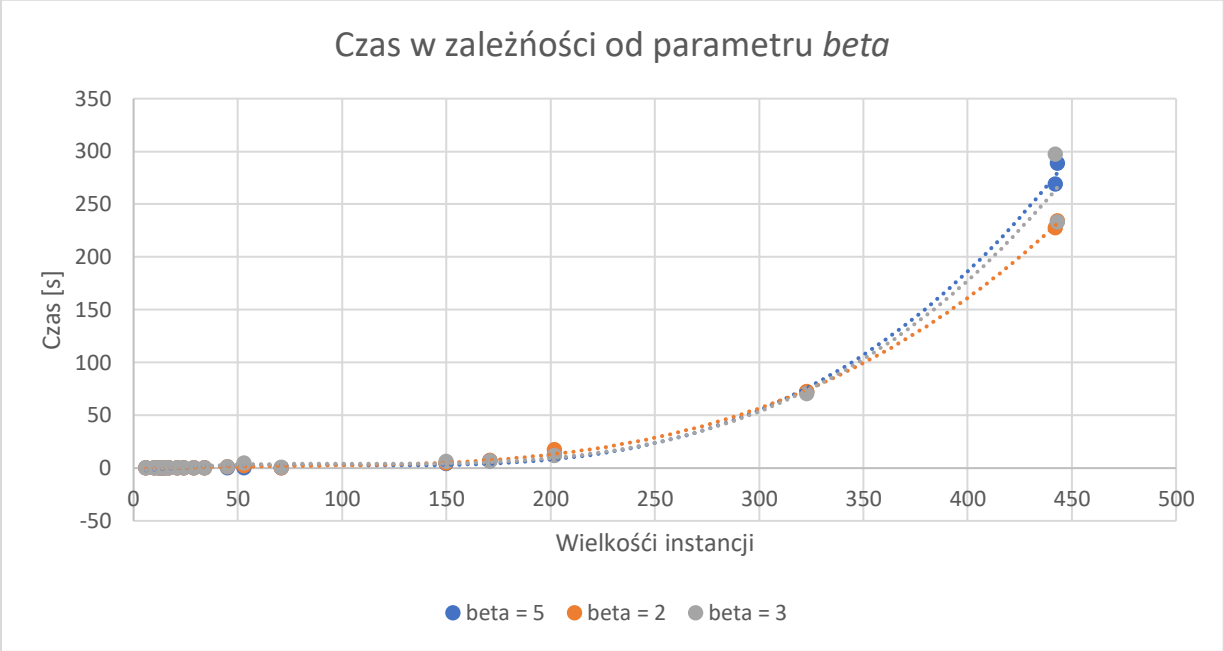


Rysunek 11. Porównanie czasu działania w zależności od wielkości alpha. Rozkład feromonu: CAS, $\beta = 5$, $\rho = 0.5$, heurystyka = visability.

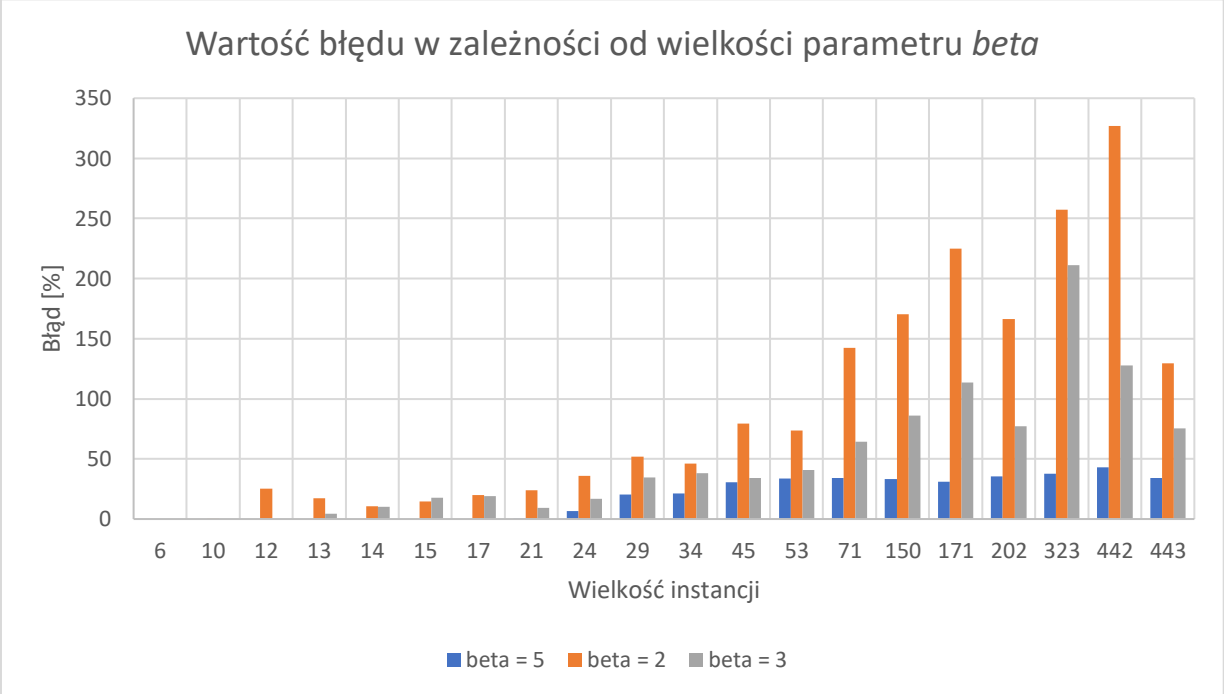


Rysunek 12. Porównanie błędu w zależności od wielkości alpha. Rozkład feromonu: CAS, $\beta = 5$, $\rho = 0.5$, heurystyka = visability.

Porównanie dla różnych wartości parametru β:

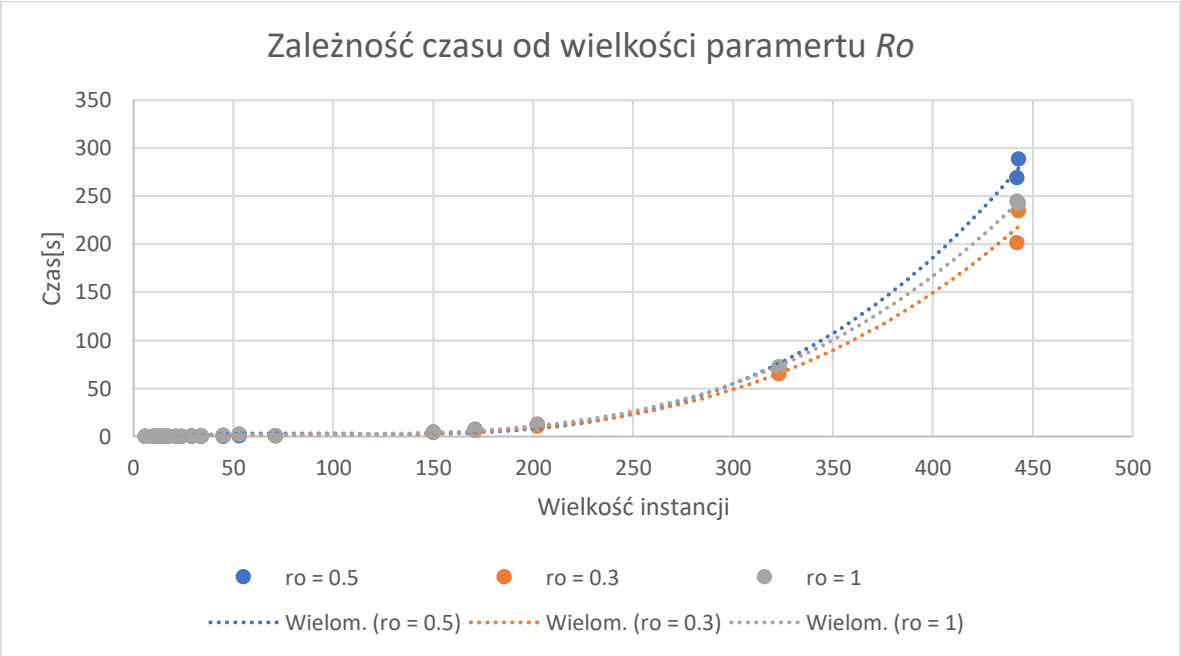


Rysunek 13. Porównanie czasu działania w zależności od wielkości beta. Rozkład feromonu: CAS, α = 1, ρ = 0.5, heurystyka = visability.

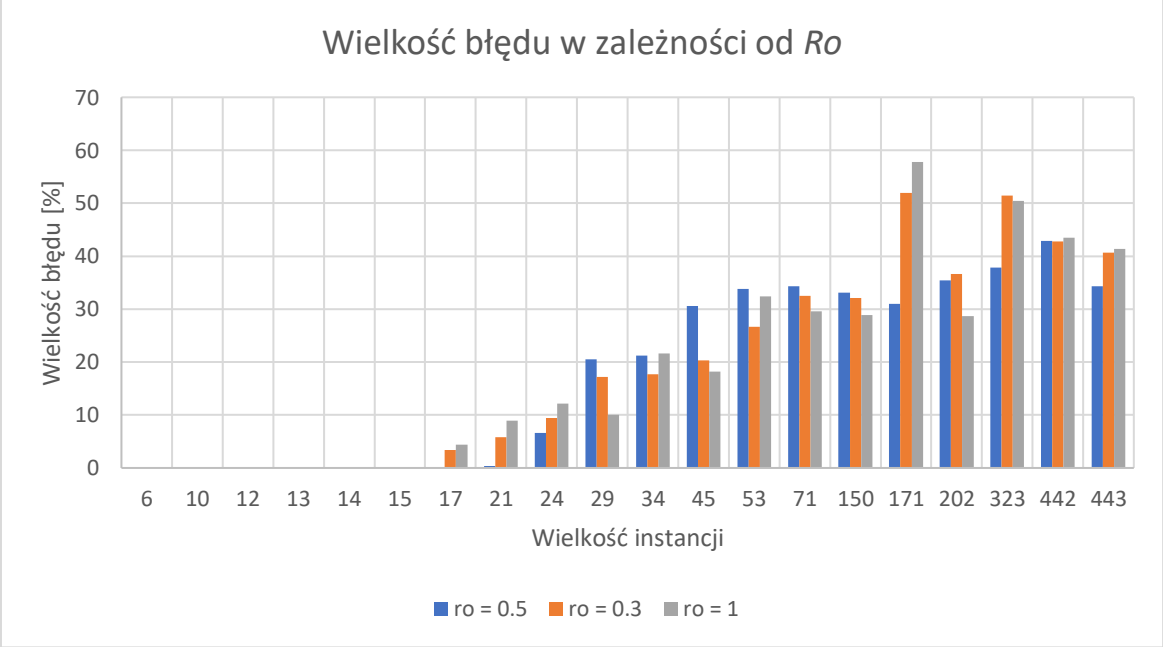


Rysunek 14. Porównanie wielkości błędu w zależności od wielkości beta. Rozkład feromonu: CAS, α = 1, ρ = 0.5, heurystyka = visability.

Porównanie dla różnych wartości ρ:

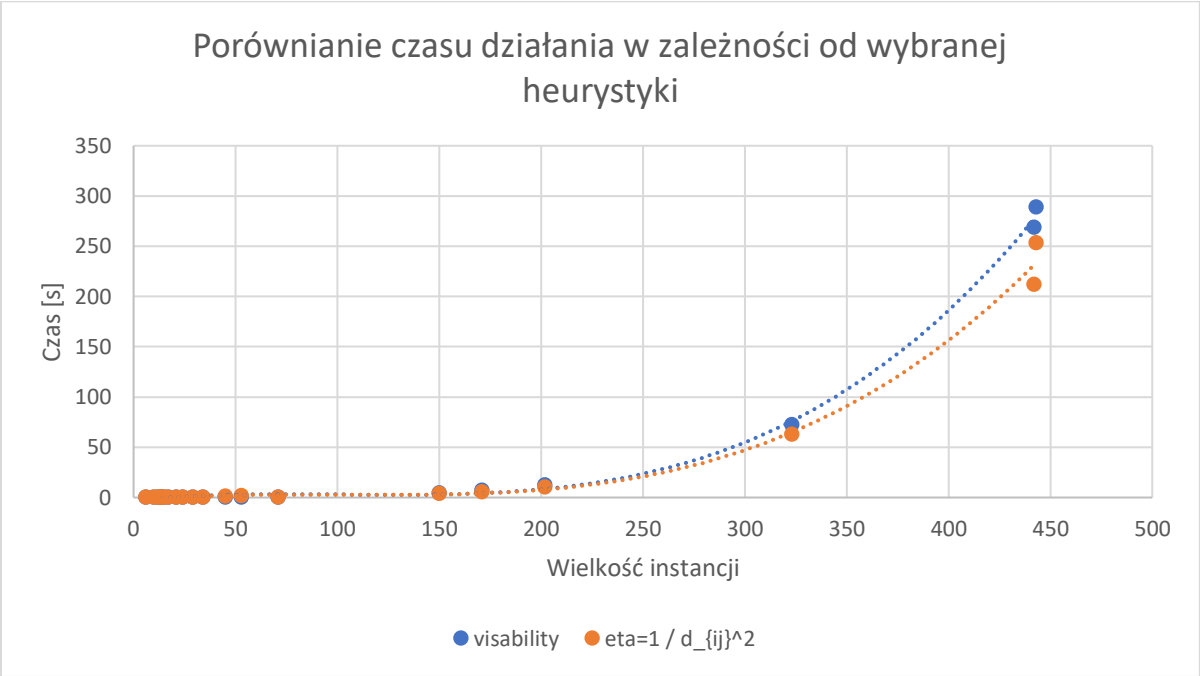


Rysunek 15. Porównanie czasu działania w zależności od wielkości Ro. Rozkład feromonu: CAS, α = 1, β = 5, heurystyka = visability.

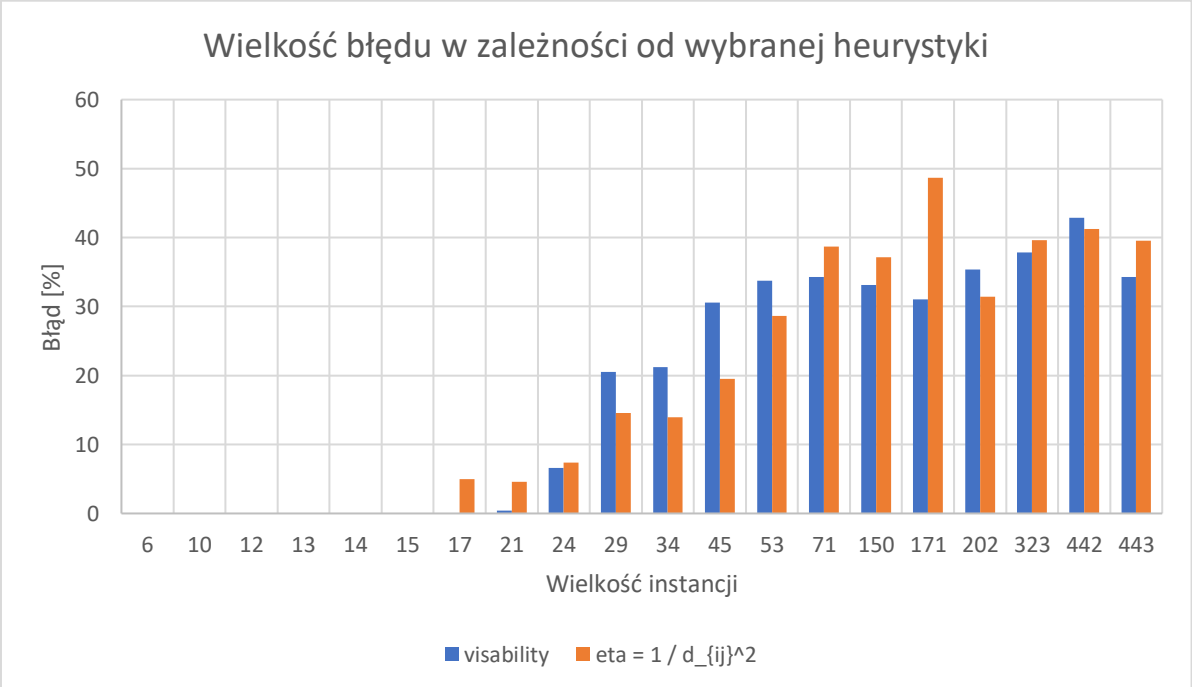


Rysunek 16. Porównanie wielkości błędów w zależności od wielkości Ro. Rozkład feromonu: CAS, α = 1, β = 5, heurystyka = visability.

Porównanie dla różnych heurystyk:



Rysunek 17. Porównanie czasu działania dla różnych heurystyk wyboru. Rozkład feromonu: CAS, $\alpha = 1$, $\beta = 5$, $\rho = 0.5$.



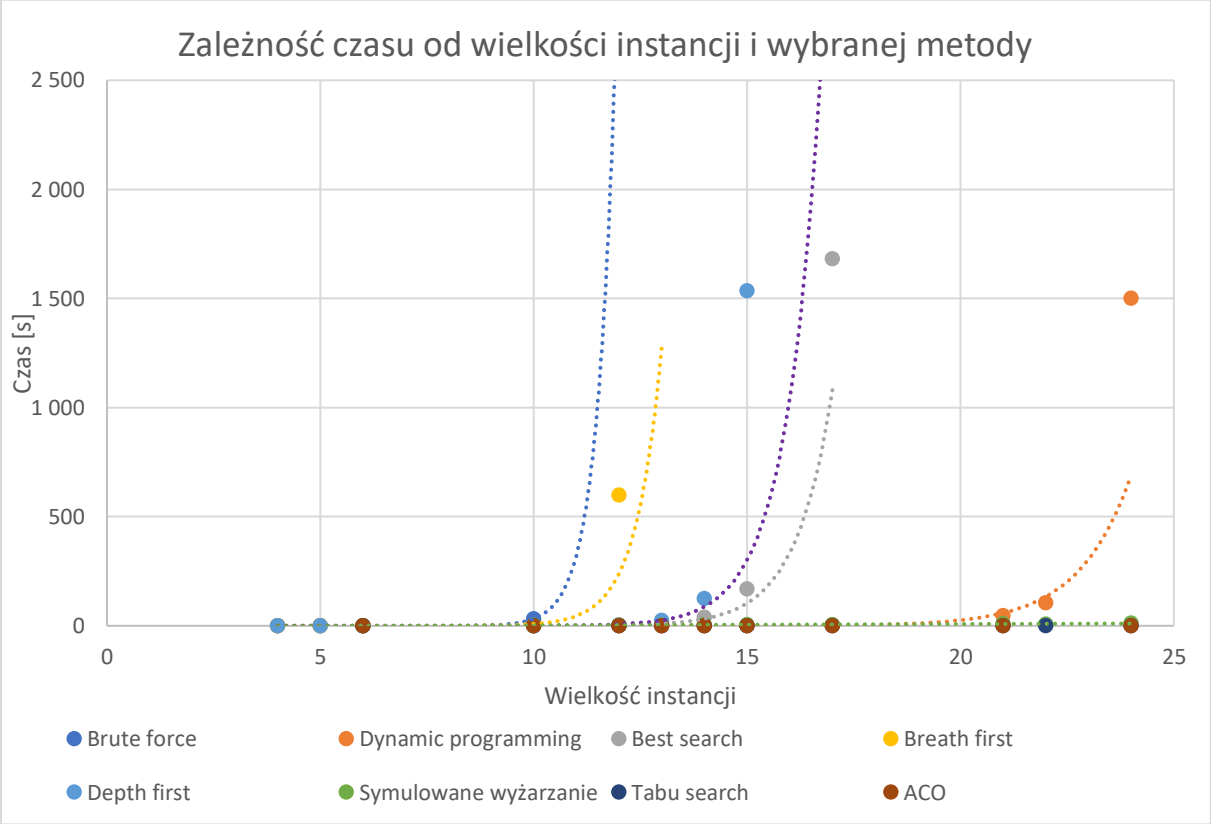
Rysunek 18. Porównanie błędu dla różnych heurystyk wyboru. Rozkład feromonu: CAS, $\alpha = 1$, $\beta = 5$, $\rho = 0.5$.

Porównanie działania algorytmu opartego o metodę algorytmów mrówkowych z innymi:

Porównywane są najlepsze uzyskane wyniki algorytmu mrówkowego.

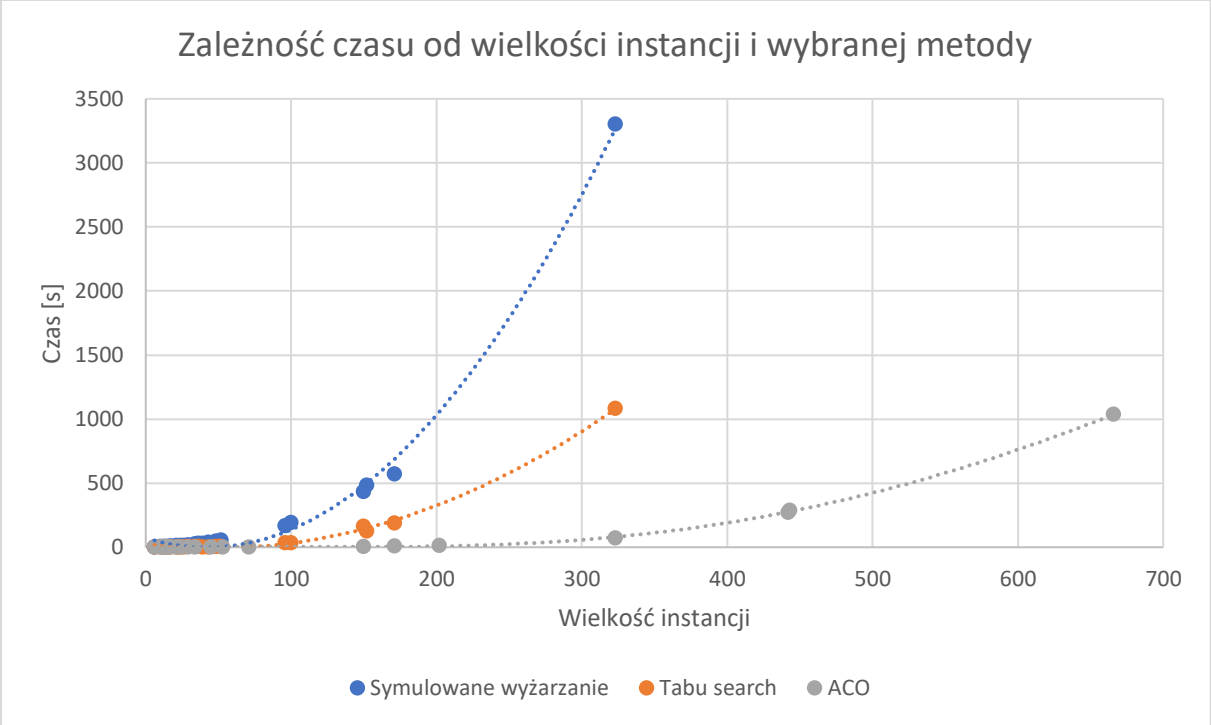
Takie wyniki uzyskano dla parametrów:

- *Alpha*: 1*,*
- *Beta*: 5,
- *Ro*: 0.5,
- *Sposób rozkładu feromonu:* CAS.

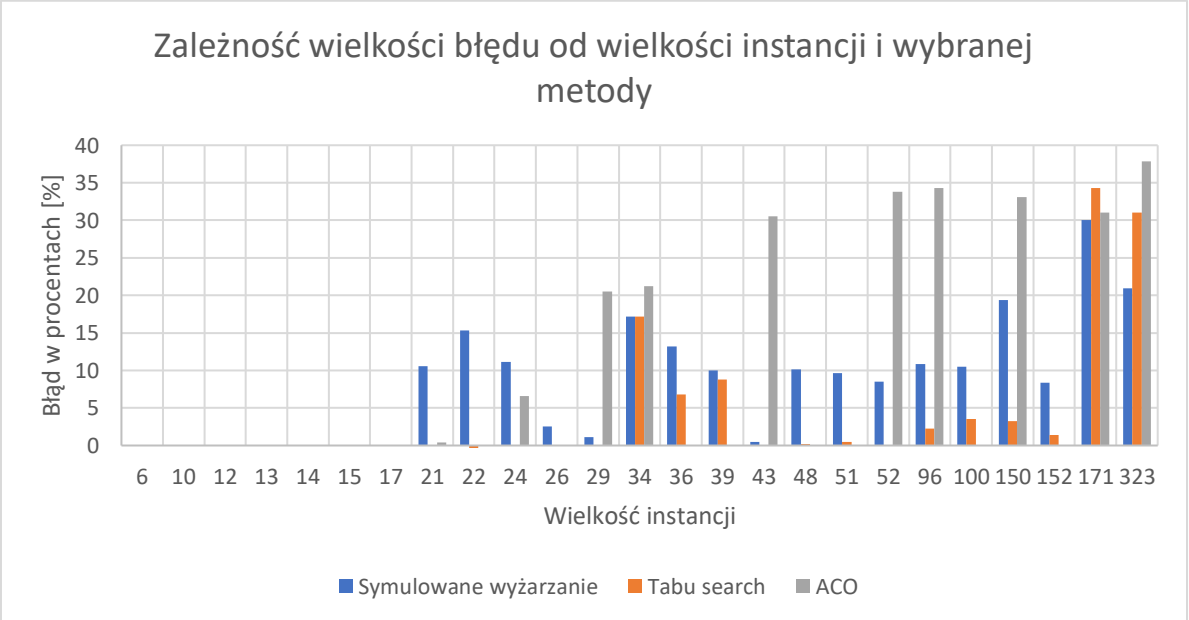


Rysunek 19. Porównanie czasowe ACO z innymi metodami.

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

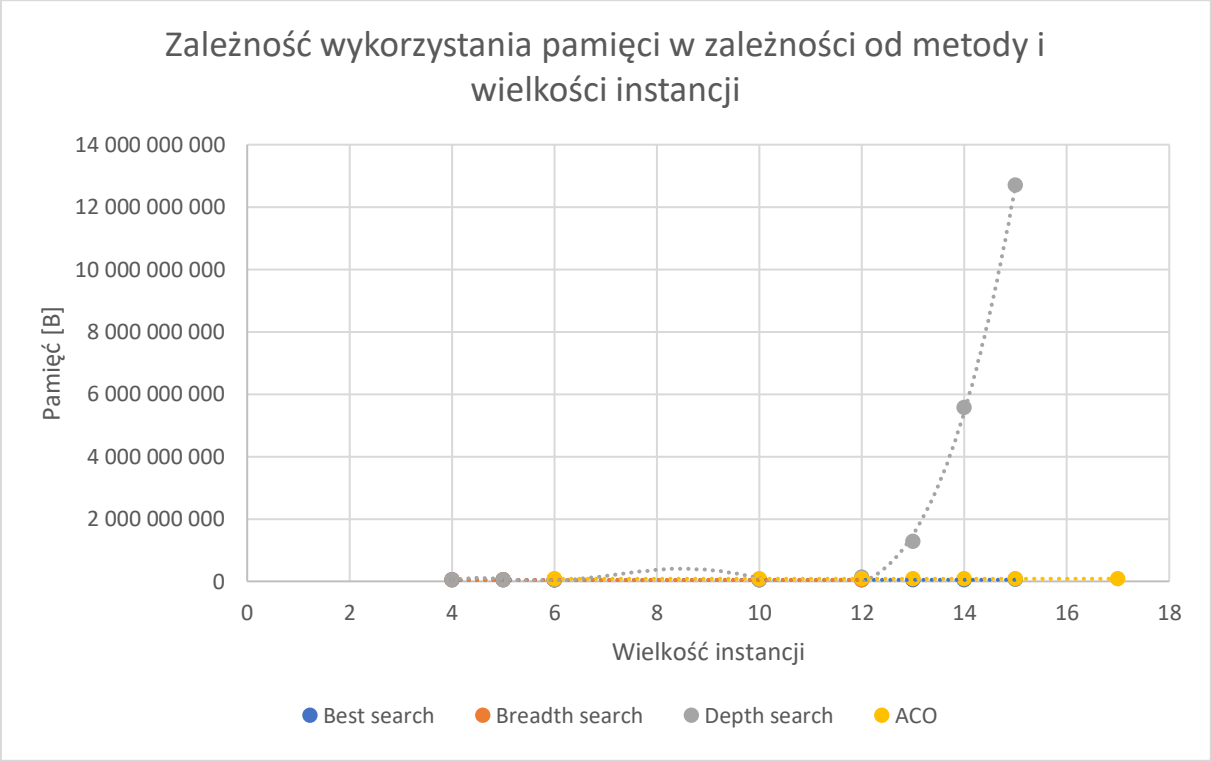


Rysunek 20. Porównanie czasu działania algorytmu opartego o metodę kolonii mrówek do tabu search oraz symulowanego wyżarzania. Liczba iteracji dla algorytmu mrówkowego wynosiła odpowiednio: iteracje = n jeśli n <= 50, iteracje = n / 3 jeśli n > 50 i n <= 80, iteracje = 3 jeśli n > 80.



Rysunek 21. Porównanie błędu algorytmu opartego o metodę kolonii mrówek do tabu search oraz symulowanego wyżarzania. Liczba iteracji dla algorytmu mrówkowego wynosiła odpowiednio: iteracje = n jeśli n <= 50, iteracje = n / 3 jeśli n > 50 i n <= 80, iteracje = 3 jeśli n > 80.

Porównanie zużycia pamięci:



Rysunek 22. Porównanie zużycia pamięci algorytmu ACO do pozostałych.

7. Analiza wyników i wnioski

Porównując wyniki eksperymentu dla różnych metod rozkładu feromonu (*Rysunek 7., 8., 9. 10., strona 10, 11*), można dostrzec, że największe błędy generowane są przez metodę *DAS*. Również porównując tendencję wzrostu czasu, *DAS* wykonuje się wolniej od pozostałych. Metody *QAS* i *CAS* otrzymują podobne wyniki, zarówno czasów jak i błędów. Nieznaczną przewagę czasową można przesądzić na korzyść *CAS.* Wynika to najprawdopodobniej z faktu, że ilość feromonu jest w tym rozkładzie aktualizowana rzadziej. Dopiero gdy wszystkie mrówki wykonają pełny cykl to feromon jest rozkładany (dzieje się to raz na iterację, a nie jak w przypadku pozostałych, co przejście jednej mrówki po jednej krawędzi). Do dalszych badań wybrano więc *CAS.* Jako sposób rozkładu feromonu.

Sprawdzono wpływ różnych wartości parametru α (reguluje on wpływ wartości feromonu na prawdopodobieństwo wyboru krawędzi) na wynik symulacji (*Rysunek 11., 12. Strona 12*.). Zalecaną przez Dorigo wartością parametru α jest 1. Taka też wartość dała najlepsze wyniki. Dla wzrostu istotności feromonu ($\alpha = 2\ lub\ 3$), generowane były większe błędy. Sztuczne mrówki sugerowały się bardziej śladem pozostawionym przez pozostałe mrówki, niż długością krawędzi.

Zbadano trzy wartości parametru β (odpowiedzialnego za wpływ heurystyki wyboru na prawdopodobieństwo wyboru krawędzi) (*Rysunek 13., 14. Strona 13.).* Wszystkie badane wartości (2, 3, 5), znajdują się w przedziale sugerowanym przez Dorigo. Na wskazanych wykresach widać tym razem znaczną różnicę. Błąd jest tym większy im mniejsza jest wartość β. Jest to wynik spodziewany, mrówki poruszają się bardziej losowo i długość krawędzi ma mniejsze znaczenie podczas wyboru (a algorytm powinien szukać najkrótszej trasy). Najlepsza okazała się wartość 5.

Wartość współczynnika parowania ρ ma istotny wpływ na sposób pracy algorytmu.(Rysunek 15., 16. Strona 14.). Jego wartość mieści się w przedziale (0, 1>, co sugeruje wyparowywany procent feromonu. W zależności od wybranego sposobu dystrybucji feromonu, różni się także moment wyparowywania. Jeśli wartość współczynnika parowania zbliża się do 1, to feromon ma duży wpływ na kolejne decyzje mrówek w kolejnych przejściach, ponieważ nie odparowuje w ogóle. Jeśli współczynnik parowania zbliża się do 0, mrówki będą poruszać się bardziej losowo. Feromon nie będzie miał dużego wpływu na ich decyzję w kolejnych przejściach, ponieważ przy każdym wyparowaniu będzie usuwał się cały feromon z krawędzi. Najbardziej rozsądną wartością wydaje się $\rho = 0.5$ (zaproponowane zresztą przez Dorigo).

Zbadano dwie heurystyki wyboru (*Rysunek 17., 18. Strona 15*): visability, oraz jej lekką modyfikację, opisaną w punkcie *parametry* (*Strona 9.*), która za wartość lokalnej funkcji kryterium przyjmuje odwrotność kwadratu długości krawędzi, a nie samą odwrotność długości krawędzi. Okazało się, że druga z nich daje bardziej dokładne wyniki dla instancji mniejszych niż ok. 53 miasta, a pierwsza dla instancji większych. Nie są to jednak różnice wskazujące jednoznacznie, która z heurystyk jest lepsza.

W zależności sposobu wyznaczania liczby iteracji (opisane pod wykresami) algorytm mrówkowy jest w stanie znajdować wyniki zadawalające (czyli : dla n < 25, 0%, dla 24 < n < 350, 50%, dla 75 < n < 2500, 150%.) w krótszym czasie niż pozostałe badane, w szczególności tabu search oraz algorytm symulowanego wyżarzania (*Rysunek 19., 20. Strona 16., 17*.). Nie oznacza to jednak, że znajduje wyniki bardziej dokładne, a jedynie że mieści się w podanych zakresach. W praktyce, manipulując parametrami, a w szczególności liczba iteracji nie udało mi się osiągnąć lepszych wyników błędu niż te przedstawione na wykresie *Rysunek 21. Strona 17.*. Tak więc algorytm *ACO* jest mniej dokładny dla instancji wielkości > 29 i < od ok. 300. Dla największych instancji widać jednak, że trend może się odwrócić. Nie można tego jednoznacznie stwierdzić, ponieważ dla poprzednich algorytmów nie zbadano ich działania dla instancji większych niż 323.

Porównując zużycie pamięci dla algorytmu ACO (*Rysunek 22. Strona 18.*) oraz pozostałych zbadanych algorytmów (w szczególności Best search, Breadth search, Depth Search, ponieważ ich słabą stroną było właśnie zużycie pamięci), można zauważyć, że ACO wymaga znacząco mniej pamięci. Wyjaśnia to dlaczego jest w stanie liczyć wyniki dla dużo większych instancji, dysponując takimi samymi zasobami.

Z eksperymentu wynika, że wszystkie wartości parametrów zaproponowane przez Dorigo, faktycznie okazały się być najbardziej odpowiednie (przynajmniej z tych przebadanych).