Projektowanie Efektywnych Algorytmów
Projekt

19/01/2023

259193 Kacper Wróblewski

(7) Algorytm mrówkowy

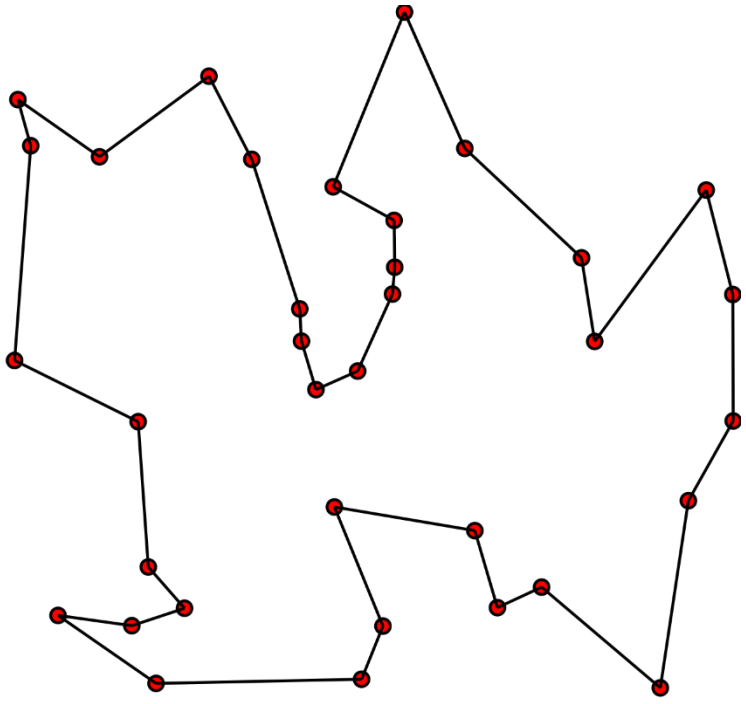
1. Sformułowanie zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywności algorytmu mrówkowego rozwiązującego problem komiwojażera w wersji optymalizacyjnej. Algorytm był realizowany na gotowym grafie stworzonym z danych do opracowania. Ze względu na specyfikę metody wśród danych do opracowania zamieszczono również dokładny wynik najkrótszej drogi dla każdej instancji jako punkt orientacyjny dla wyliczania wartości błędu.

Należało zbadać zależność czasową od wielkości instancji (jak w zadaniu 1. i 2.), pamięciową (jak w zadaniu 2.) oraz od parametrów algorytmu.

Problem komiwojażera, czyli TSP (*travelling salesman problem*) polega na znalezieniu cyklu Hamiltona w grafie, który ma najmniejszy koszt. Sprowadza się to do wyznaczenia najkrótszej ścieżki pomiędzy wierzchołkami przedstawianymi jako miasta, stąd problem podróżującego sprzedawcy. Dana jest określona ilość miast i odległość albo cena podróży pomiędzy nimi. Podróżujący musi odwiedzić wszystkie z nich płacąc jak najmniej lub podróżując jak najmniejszą odległość. Jak przedstawiono na Rys. 1., rozwiązaniem jest cykl w grafie zupełnym w postaci listy kolejnych wierzchołków oraz całkowity koszt przebytej drogi.

Problem TSP należy do klasy problemów NP-trudnych, co znaczy, że rozwiązanie nie zawsze jest jasne lub zajmuje zwyczajnie zbyt długo, aby brać je pod uwagę przez co wymagane jest częste zawężanie kryteriów lub kontemplacja jakości algorytmu w danym kontekście.



https://pl.wikipedia.org/wiki/Problem_komiwojażera#/media/Plik:GLPK_solution_of_a_travelling_salesman_problem.svg

Rys. 1. Przykładowe rozwiązanie problemu TSP

2. Metoda

Algorytm mrówkowy (ang. *ant colony optimization, ACO*) podobnie jak algorytm symulowanego wyżarzania swoją ideę czerpie z przyrody, a konkretnie z zachowania mrówek szukających pożywienia dla swojej kolonii. Został on zaproponowany przez Marco Dorigo w 1991 roku. Jest on heurystyczną techniką szacowania drogi w grafie, co oznacza, że wyniki jego działania należy badać ze świadomością niedokładności metody.

Mrówki w przyrodzie poruszają się w sposób losowy w poszukiwaniu pożywienia, kiedy jednak je znajdą, wracają do swojej kolonii zostawiając ślad feromonów (zjawisko to nazywa się stygmergia), który inne mrówki mogą wykorzystywać by optymalizować swoje poszukiwania. Feromony po pewnym czasie wyparowują, jest to jednak zjawisko pożądane, gdyż ślady krótszych ścieżek dzięki temu posiadają większe stężenie feromonów na sobie sugerując mrówkom, mającym wyśmienity zmysł węchu, że ta właśnie trasa zapewnia szybszy dostęp do pożywienia. W efekcie, ścieżka o nieoptymalnym koszcie zanika, a ta uczęszczana (najkrótsza) pozostaje, gdyż ślad feromonów jest najbardziej intensywny. Dzięki temu, że mrówek w kolonii jest nieprzeliczalnie dużo ten sposób zachowania nazywa się inteligencją zbiorową.

W algorytmie, symulowane „mrówki" zachowują się podobnie. Na początku losowana jest ścieżka na grafie dla każdej z nich, a po przejściu po niej aktualizują ślad feromonów w chwili *t.* Określany jest jako $\tau_{ij}(t)$, który na początku jest taki sam dla każdej trasy. Aktualizowany jest wg wzoru na rysunku 2.

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t, t+1),$$

gdzie:
ρ jest współczynnikiem z przedziału $< 0,1 >$, określającym ilość wyparowującego feromonu w jednostce czasu (0 - wyparowuje cały, 1 - wyparowuje nic).

$$\Delta\tau_{ij}(t, t+1) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t, t+1),$$

$\Delta\tau_{ij}^{k}(t, t+1)$ jest ilością feromonu odkładaną na jednostkę długości krawędzi (i, j) rozkładanego przez k-tą mrówkę w jednostce czasu.

Rys. 2. Wzór opisujący aktualizację stężenia feromonu
(Źródło: slajd z wykładu)

Sztuczne mrówki mogą podejmować decyzje o wyborze drogi w oparciu o kryterium, jeśli ścieżki mają takie samo stężenie feromonów (rys. 3.). Dodatkowe kryterium jest wymagane, gdyż bez niego decyzja prowadziłaby do wyboru zachłannego.

α określa wpływ doświadczeń poprzednich pokoleń, czyli jak mocny wpływ poprzednie przejścia po danym wierzchołku pomagają „zadecydować". β natomiast określa siłę wyboru zachłannego drogi na podstawie odległości między miastami (heurystyka *visibility,* stopień widoczności miasta *j* z *i*).

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

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}.

Rys. 3. Prawdopodobieństwo wyboru miasta j przez k*-tą mrówkę w mieście* i

(Źródło: slajd z wykładu)

W przypadku wymyślonej heurystyki *frequency factor* liczba η_{ij} została zamieniona na:

$$\frac{1}{d_{ij} \cdot f_i}$$

Gdzie:

f_i – częstość odwiedzania miasta *i* przez mrówki na danym etapie losowania drogi

Każda z mrówek posiada wiedzę (swoistą listę tabu) o poprzednio odwiedzonych wierzchołkach, dzięki czemu nie wraca się do odwiedzonych już miast. Po każdej iteracji (ilość których ustala użytkownik) zapamiętywana jest najkrótsza trasa, a „pamięć" mrówek jest zerowana.

Istnieje również przypadek zwany *uni-path behawior,* czyli wylosowanie takiej samej trasy dla każdej z mrówek, jednak nie jest on tutaj badany.

Według Dorigo (twórcy metody), wyróżniamy trzy rodzaje algorytmów mrówkowych:

- Algorytm Gestosciowy (*ang. ant-density DAS*)
- Algorytm Ilosciowy (*ang. ant-quantity QAS*)
- Algorytm Cykliczny (*ang. ant-cycle CAS*)

Różnią się one głównie sposobem aktualizowania feromonów oraz momentem aktualizacji.

W metodzie *DAS* przy przejściu po krawędzi *(i, j)* na każdą jednostkę długości rozkład feromonu jest stały, zależny od zmiennej *Q*. W metodzie *QAS* stała ilość feromonu *Q* dzielona jest przez długość krawędzi d_{ij}. W metodzie *CAS* natomiast, *Q* dzielone jest przez długość trasy L_k znalezioną przez *k*-tą mrówkę. W przeciwnych wypadkach przypisywane jest 0.

Według literatury złożoność algorytmu wynosi:

$$O(CC \cdot n^2 \cdot m)$$

Gdzie:

CC – liczba iteracji algorytmu,

n^2 – liczba miast,

m – liczba mrówek

Jednak twórcy algorytmu znaleźli liniową zależność pomiędzy liczbą miast, a liczbą mrówek, zatem można przyjąć, że złożoność obliczeniowa wynosi:

$$O(CC \cdot n^3)$$

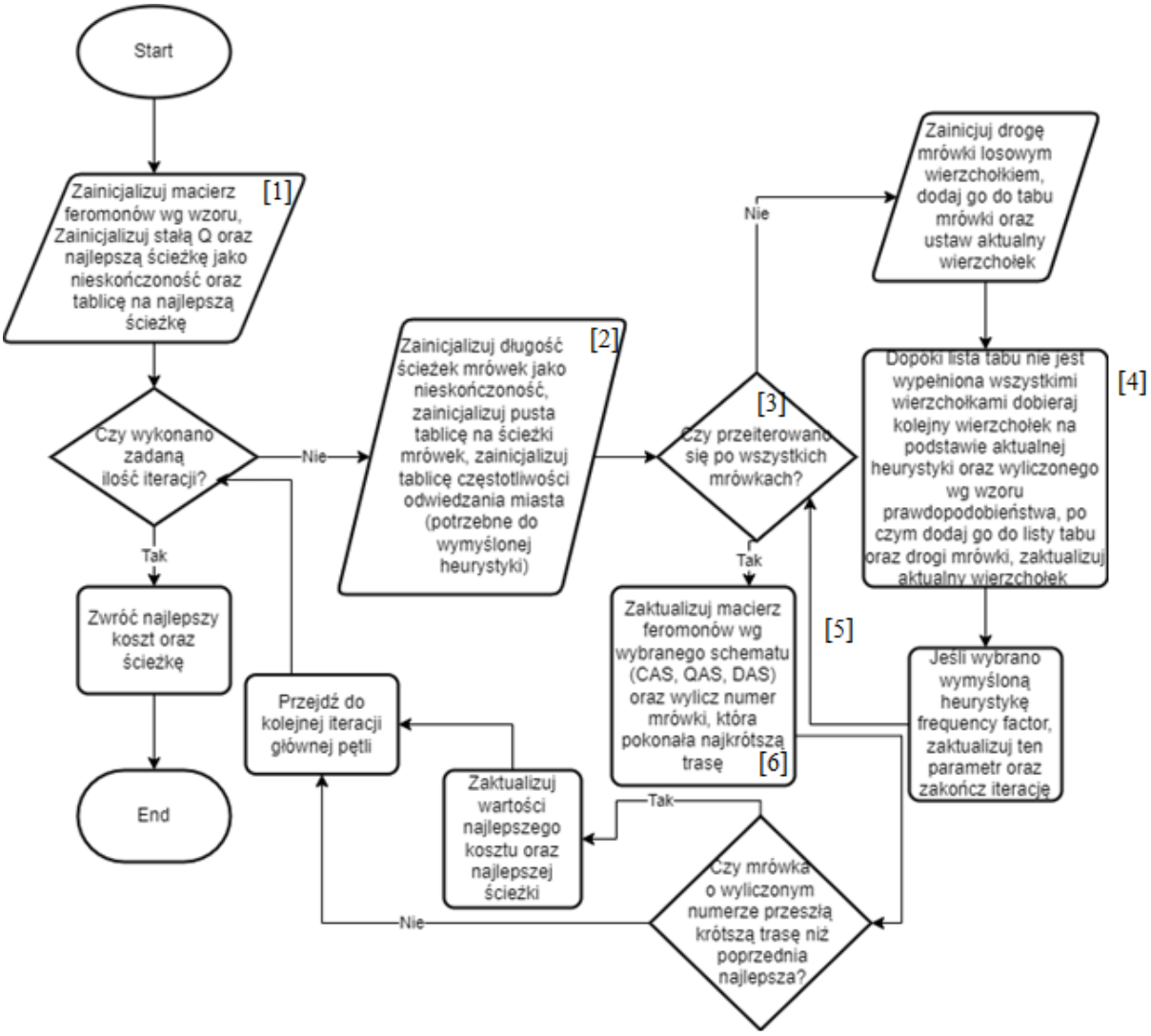
Na podstawie badań zaleca się przyjęcie następujących wartości parametrów:

- $\alpha = 1$,
- $\beta \in [2; 5]$,
- $\rho = 0.5$,
- $m = n$,
- $\tau_0 = \frac{m}{C^{nn}}$, gdzie C^{nn} jest szacowaną długością trasy.

3. Algorytm

Rys. 4. Schemat blokowy całego programu

Program, jak przedstawiono na Rys. 2., rozpoczyna się od wczytania danych zawartych w pliku konfiguracyjnym. Potem należy rozpatrzeć przypadek rozszerzenia pliku, gdyż część z badanych zbiorów ma inny format niż *.txt*, który wymaga nieco innej metody odczytu (pliki *.tsp/.astp* można łatwo odczytać za pomocą biblioteki *tsplib95* w pythonie). Następnie program wchodzi w pętlę realizującą zaimplementowany algorytm. Liczba iteracji tej pętli określona jest w pliku konfiguracyjnym. Gdy program zakończy iterację, zapisuje zwrócone dane do pliku, którego ścieżka i nazwa również znajdują się w pliku sterującym, czym powinien zakończyć się program. Opcjonalnie, na końcu programu można wypisać w konsoli interesujące programistę dane, w celu szybszej weryfikacji poprawności metody.



Rys. 5. Schemat blokowy implementacji algorytmu mrówkowego

Algorytm zaczyna się inicjalizacją macierzy feromonów (na samym początku ma wszędzie takie same wartości), stałej *Q* potrzebnej do wyliczania feromonów zostawionych przez mrówkę na jej drodze oraz zmiennych do przetrzymywania wyników najlepszej mrówki dla danej iteracji (rys. 5. [1]).

Następnie algorytm wkracza w główną pętlę, której liczba iteracji zdefiniowana jest przez użytkownika programu. Jest w niej inicjalizowana tablica długości ścieżki dla każdej mrówki (na początku są to nieskończoności), tablica ścieżek każdej z mrówek oraz opcjonalnie wyliczany *frequency factor* używany w wymyślonej heurystyce (rys. 5. [2]).

W kolejnym kroku, uruchamiana jest pętla iterująca się po każdej mrówce zgodnie z jej numerem (rys. 5. [3]). Dla każdej mrówki losowany jest pierwszy wierzchołek (dodawany tym samym do listy tabu mrówki) oraz ten wierzchołek ustawiany jest jako aktualnie rozpatrywany. Następnie program wkracza

w pętlę *while*, której warunkiem zatrzymania jest wypełnienie listy tabu dla *k*-tej mrówki (rys. 5. [4]). W ciele tej pętli losowane są kolejne wierzchołki, które ma pokonać ta mrówka w oparciu o macierz feromonów oraz wybraną metodę (heurystyka *visibility* lub *frequency factor*) i dodawane są do listy tabu gwarantującej brak powtórzeń miast w ścieżce. Na koniec każdej iteracji aktualizowany jest również rozpatrywany wierzchołek tym wybranym w oparciu o kryterium. Po skończeniu trasy, mrówka wraca do początkowego miasta, po czym liczona jest trasa przebyta przez rozpatrywaną mrówkę wraz z dodaniem jej do listy ścieżek (rys. 5. [5]). Opcjonalnie, jeśli wybrana jest heurystyka *frequency factor*, aktualizowany jest ten parametr. Gdy program rozpatrzy wszystkie mrówki w danej iteracji, aktualizuje macierz feromonów na podstawie wybranej metody (*QAS. DAS, CAS)* oraz wylicza numer mrówki, która pokonała najkrótszą trasę (rys. 5. [6]). Jeżeli ta mrówka pokonała trasę o mniejszym koszcie niż poprzedni najlepszy, wartości najlepszej ścieżki oraz kosztu są aktualizowane.

Po wykonaniu określonej przez użytkownika liczby iteracji algorytm zwraca najlepsze wyliczone wartości kosztu podróży oraz kolejności odwiedzonych wierzchołków.

4. Dane testowe

Do sprawdzenia poprawności działania algorytmu i przeprowadzenia badań wybrano następujący zestaw instancji (wyniki zebrano w punkcie 6. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/):

Plik:	*Optymalne wartości (koszt):*
1. burma14.tsp	*3323*
2. gr17.tsp	*2085*
3. gr21.tsp	*2707*
4. gr24.tsp	*1272*
5. bays29.tsp	*2020*
6. ftv33.atsp	*1286*
7. ftv44.astp	*1613*
8. ft53.atsp	*6905*
9. ch150.tsp	*6528*
10. ftv170.atsp	*2755*
11. gr202.tsp	*40160*
12. gr666.tsp	*294358*
13. rbg323.atsp	*1326*
14. pcb442.tsp	*50778*
15. rbg443.atsp	*2720*
16. pr1002.tsp	*259045*
17. pr2392.tsp	*378032*

Do sczytywania optymalnej drogi wykorzystano plik pomocniczy, z którego zgodnie z nazwą instancji program odczytuje optymalną wartość ścieżki.

Do programu został dołączony plik *config.ini*, aby sterować parametrami programu w sposób zastępujący:

Plik wejściowy:
./input/ftv170.tsp *//wybór instancji po nazwie z folderu input*
Plik wyjściowy:
./output/data.csv *//wybór pliku wyjściowego*
Ilość powtórzeń:
1 *//ilość iteracji pętli z algorytmem*
Alfa: *//poniżej dobór parametrów algorytmu*
1.0 *// mrówkowego*
Beta:
5.0
Rho:
0.5
Ilość mrówek: (dla 0 - automatyczna = ilość wierzchołków)
170
Ilość iteracji:
10
Aktualizacja feromonu: (DAS, QAS, CAS) *//sposób aktualizacji rozkładu feromonu*
QAS
Heurystyka wyboru: (visibility, frequency factor) *//wybór heurystyki* visibility *(podana na*
visibility *// wykładzie) lub* frequency factor *(wymyślona)*

5. Procedura badawcza

Należało zbadać zależność czasu rozwiązania problemu i złożoność pamięciową od wielkości instancji dla algorytmu mrówkowego. W przypadku tego algorytmu w przestrzeni rozwiązań dopuszczalnych występowały parametry programu, które mogły mieć wpływ na czas i jakość uzyskanego wyniku. W związku z tym procedura badawcza polegała na uruchomieniu programu sterowanego plikiem inicjującym *.INI*, wraz z podaniem żądanych parametrów metody (jego struktura została opisana na poprzedniej stronie).

Dla instancji mniejszych niż 24 błąd musiał wynosić 0%, dla przedziału [24, 350) błąd mógł mieścić się w granicach 50%, natomiast dla wyższych instancji mógł wynosić maksymalnie 150%.

Oprócz tego, zbadano wpływ parametrów α, β, ρ i sposobu rozkładu feromonu *(QAS, DAS, CAS)* na błąd oraz czas wykonywania algorytmu. Porównano również heurystykę wyboru *visibility* do wymyślonej, o nazwie *frequency factor*, która obliczana jest na podstawie częstości odwiedzania miast przez mrówki na danym etapie generacji drogi oraz zbadano jej wpływ na jakość rozwiązania.

Instancje badano 10 razy, po czym wyciągano średni wynik.

Podczas badań, żadna z instancji nie wykroczyła poza zakres badań długością wykonywania większą niż 10 minut. Dla instancji mniejszych niż 100, aby osiągnąć żądany zakres błędu, należało dobierać parametry, natomiast dla większych instancji dobór parametrów programu wpływał na jakość wyniku, lecz wystarczyło uruchomić program dla wartości domyślnych, aby uzyskać zadowalające wyniki, mieszczące się w żądanym zakresie błędu.

Podczas gdy badano poszczególne parametry, resztę ustawiano na wartości domyśle (te ustalone przez Dorigo, opisane w punkcie 2.), aby na przestrzeni badań jedyną zmienną był aktualnie badany parametr.

Domyślą metodą podczas badań był rozkład feromonu *QAS* oraz heurystyka *visibility.*

Ponadto, mierzono średnie zużycie pamięci procesu dla tego algorytmu, celem porównania złożoności pamięciowej z programowaniem dynamicznym (które znane jest z wysokiej złożoności pamięciowej).

Wyniki opracowane zostały w programie MS Excel.

Badania pokazały, że instancje o rozmiarach kolejno 33, 170 oraz 443 dają nieprzewidywalne wyniki, nie brano ich zatem pod uwagę w podsumowaniu.

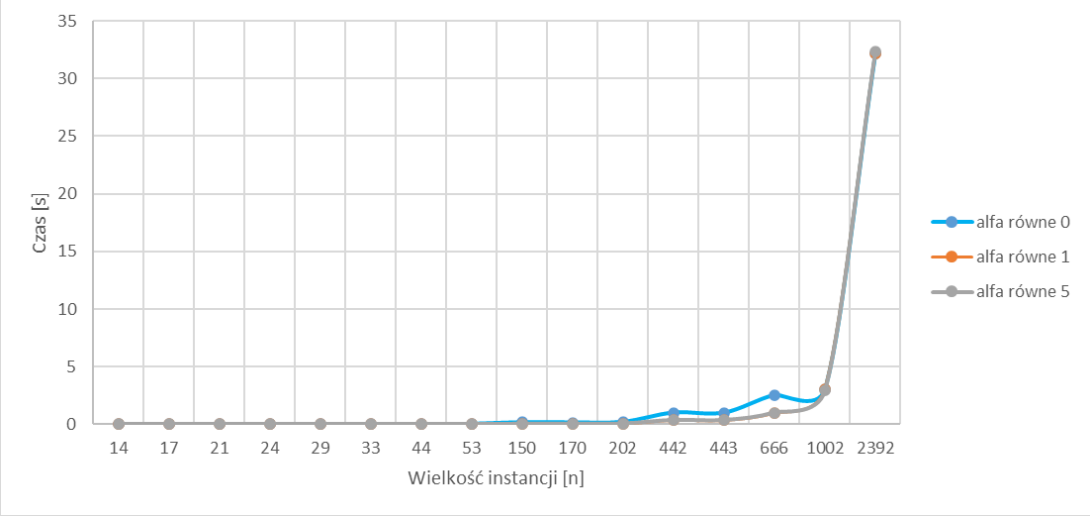
Do pomiarów złożoności pamięciowej został wykorzystany menedżer zadań Windows, który pokazuje aktualną ilość pamięci zajmowanej przez proces programu.

6. Wyniki

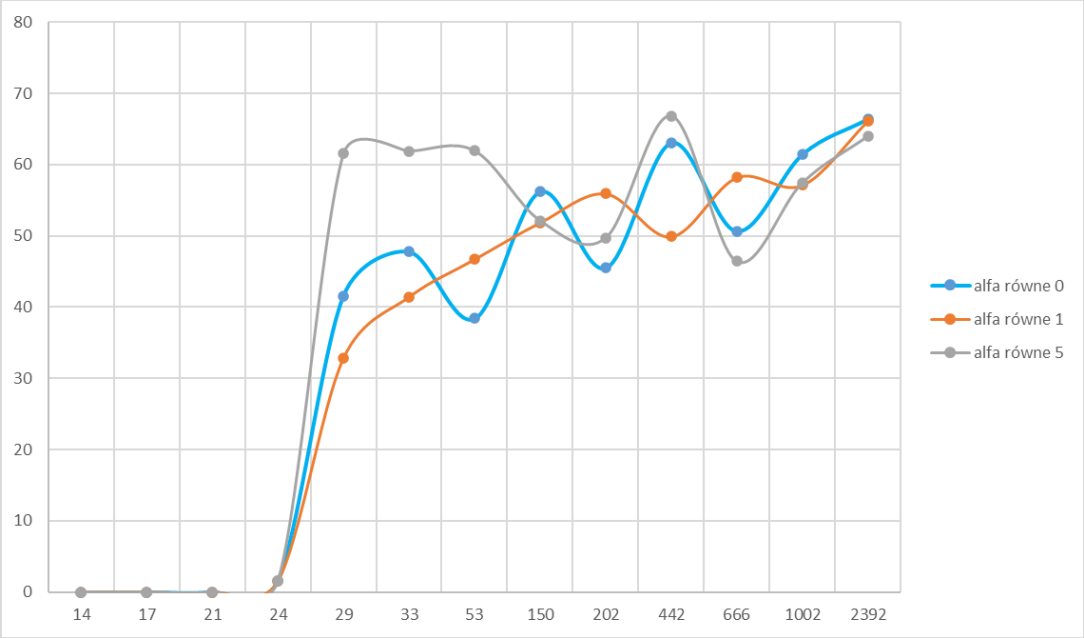
Wyniki zgromadzone zostały w plikach *.csv* w katalogu *output*. Przedstawione zostały w postaci wykresów zależności czasu uzyskania rozwiązania problemu od wielkości instancji (jak na wykresie 1.) lub jako wykresy zależności wartości błędu od wielkości instancji (wykres 2.).

6.1. Badanie zależności parametrów od czasu wykonania programu i wartości błędu dla zadanych instancji:

Parametr α:

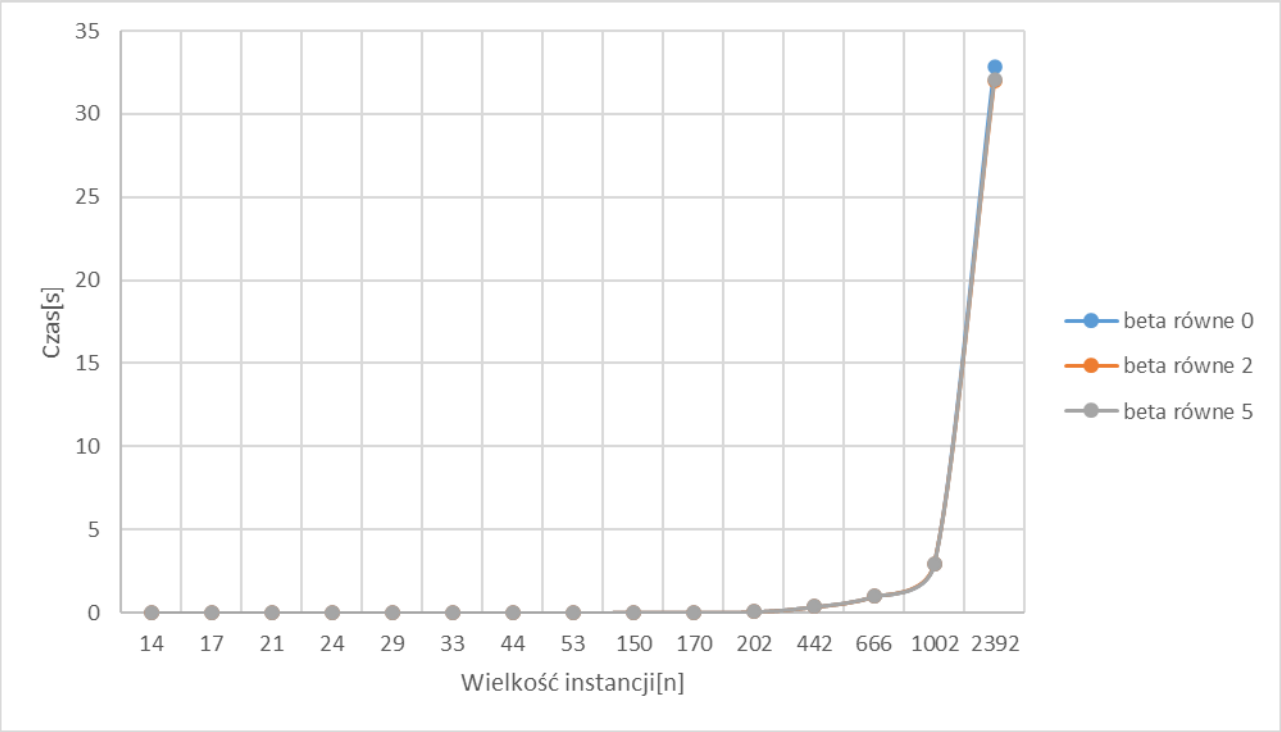


Wykres 1. Wielkość instancji od czasu wykonania programu dla różnych wartości parametru alfa

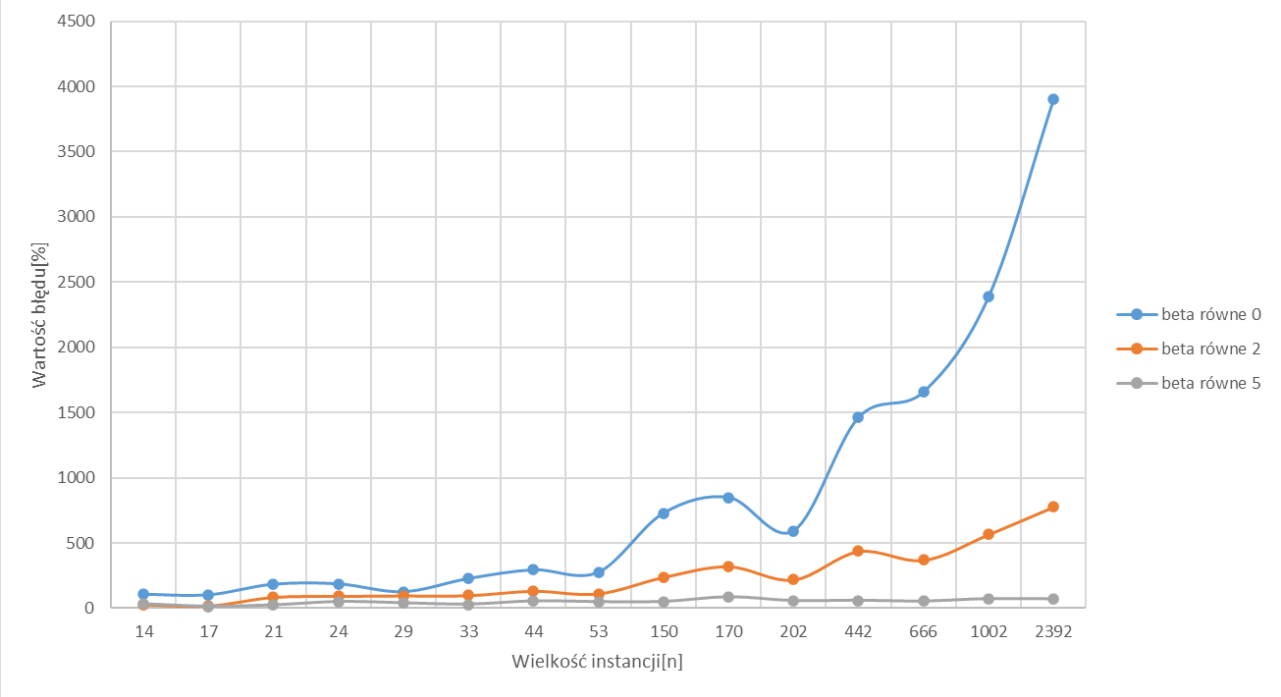


Wykres 2. Wielkość instancji od wartości błędu dla różnych wartości parametru alfa

Parametr β:

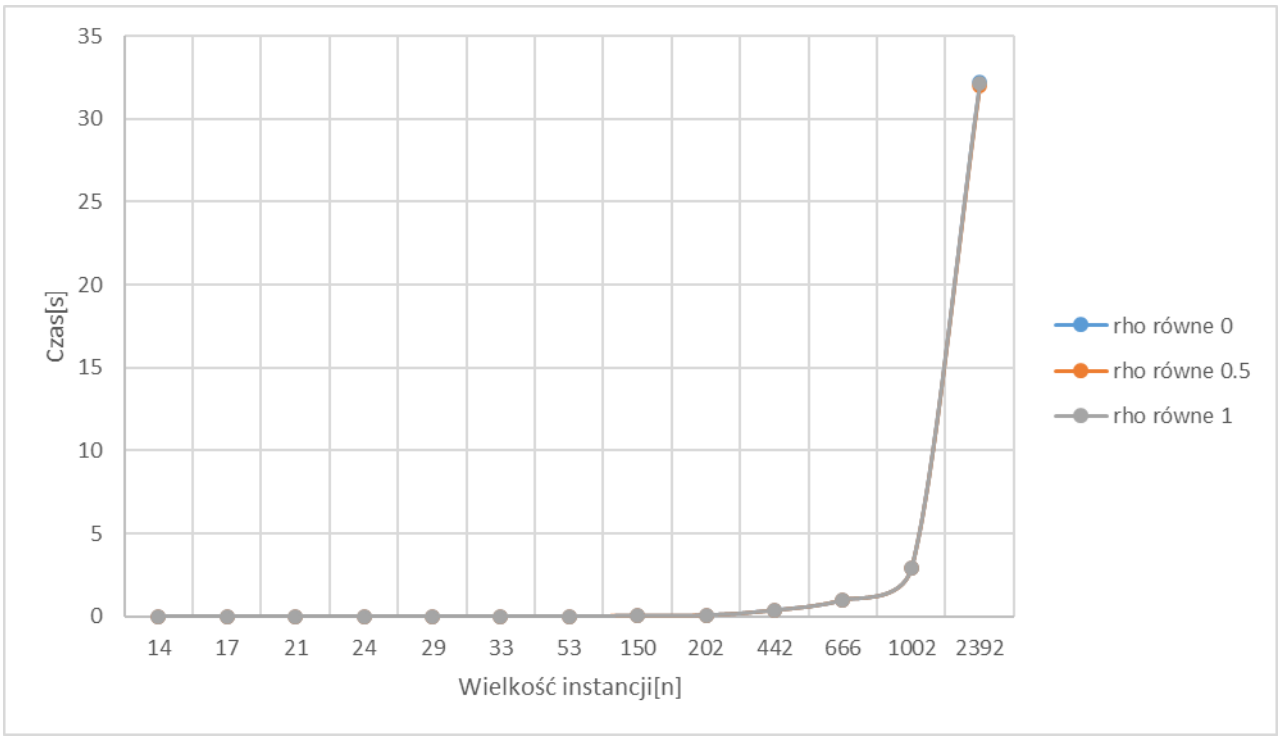


Wykres 3. Wielkość instancji od czasu wykonania dla różnych wartości parametru beta

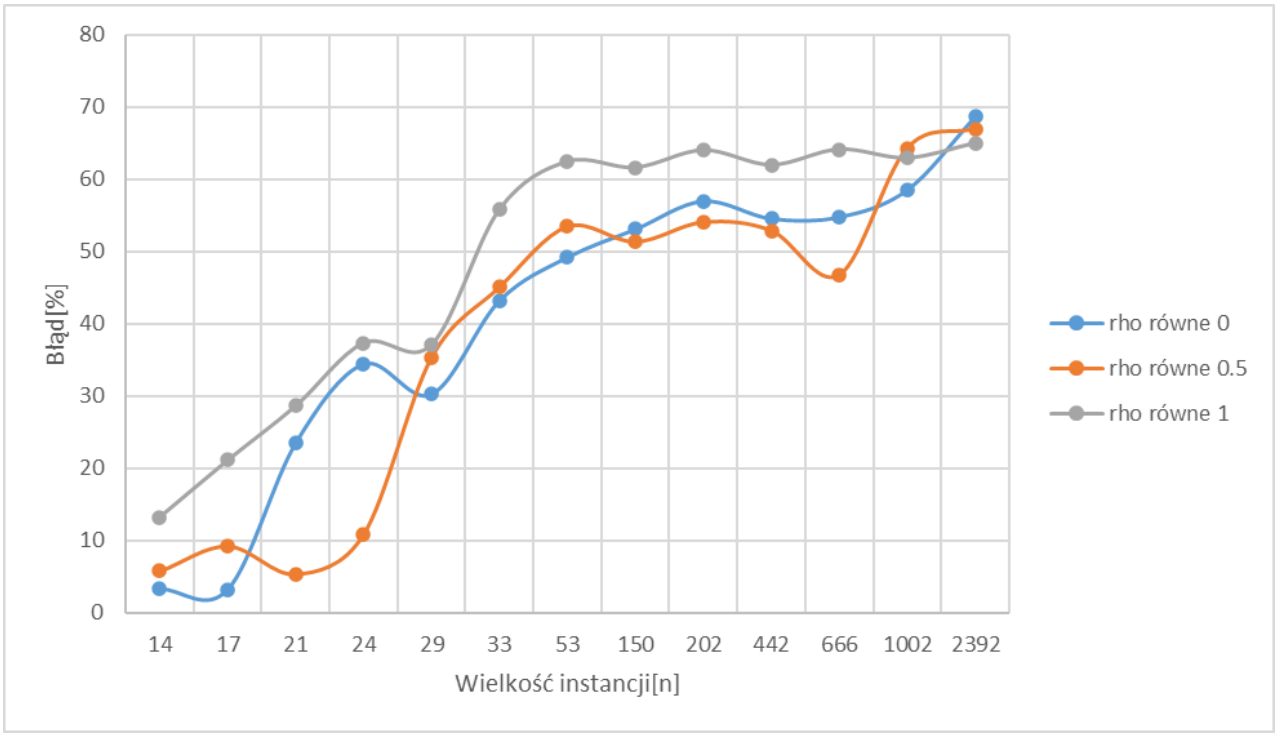


Wykres 4. Wielkość instancji od wartości błędu dla różnych wartości parametru beta

Parametr ρ:

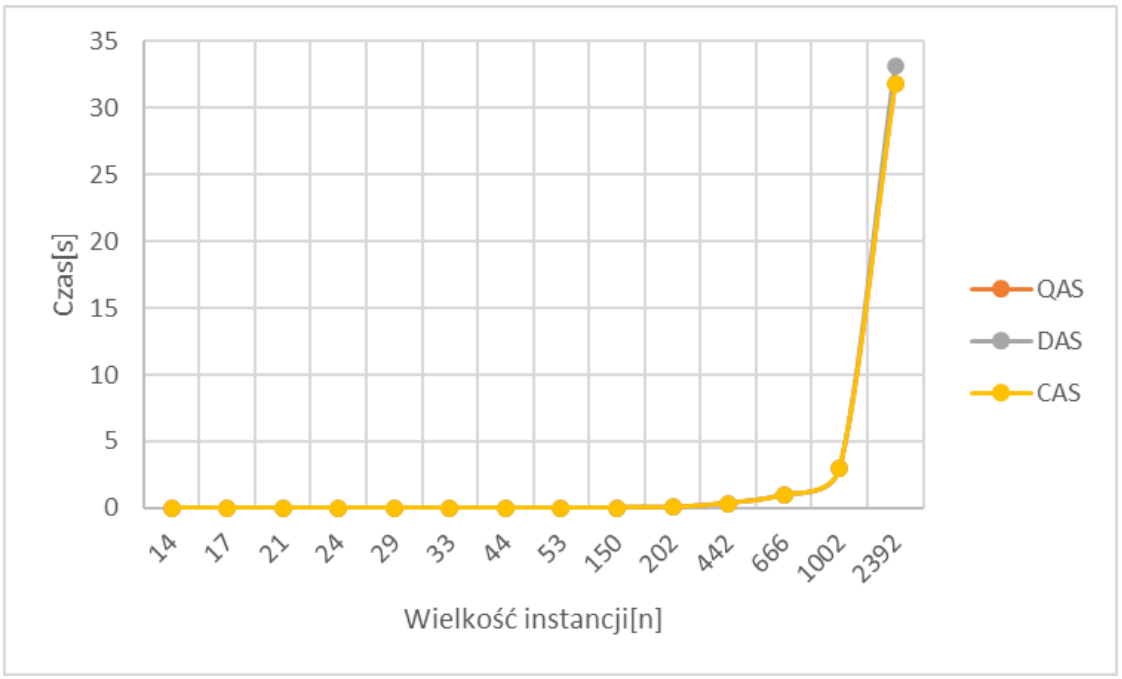


Wykres 5. Wielkość instancji od czasu wykonania programu dla różnych wartości parametru rho

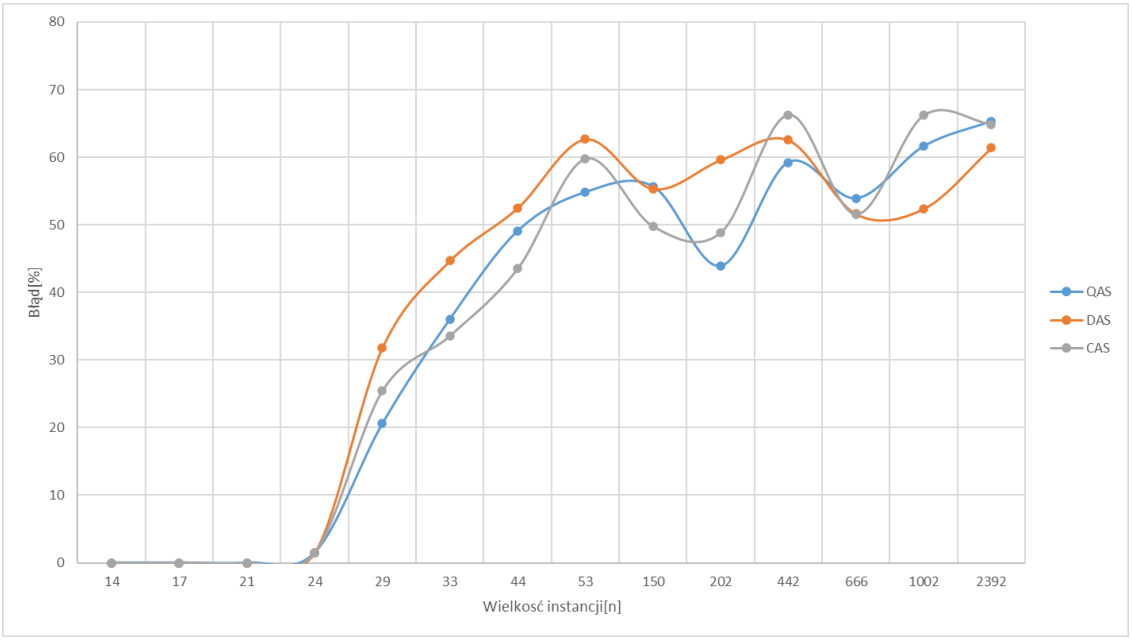


Wykres 6. Wielkość instancji od wartości błędu dla różnych wartości parametru rho

6.2. Badanie zależności metody rozkładu feromonu od czasu oraz wartości błędu:

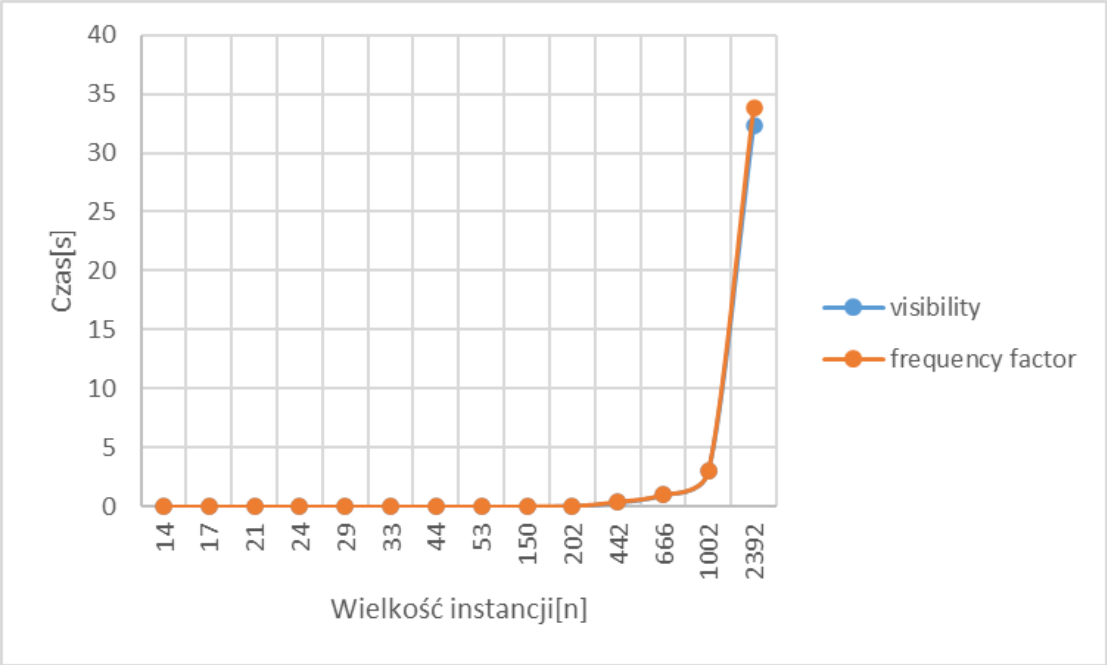


Wykres 7. Wielkość instancji od czasu wykonania programu dla różnych metod rozkładu feromonu

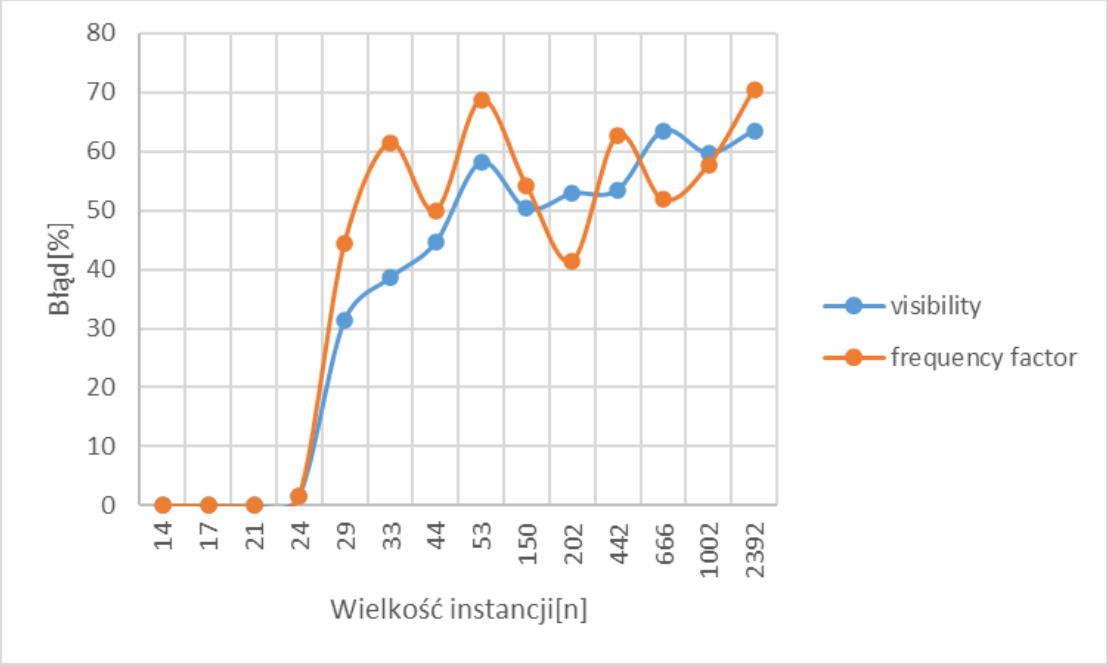


Wykres 8. Wielkość instancji od wartości błędu dla różnych metod rozkładu feromonu

6.3. Porównanie heurystyki *visibility* do wymyślonej *frequency factor*:



Wykres 9. Wielkość instancji od czasu wykonania programu dla różnych heurystyk



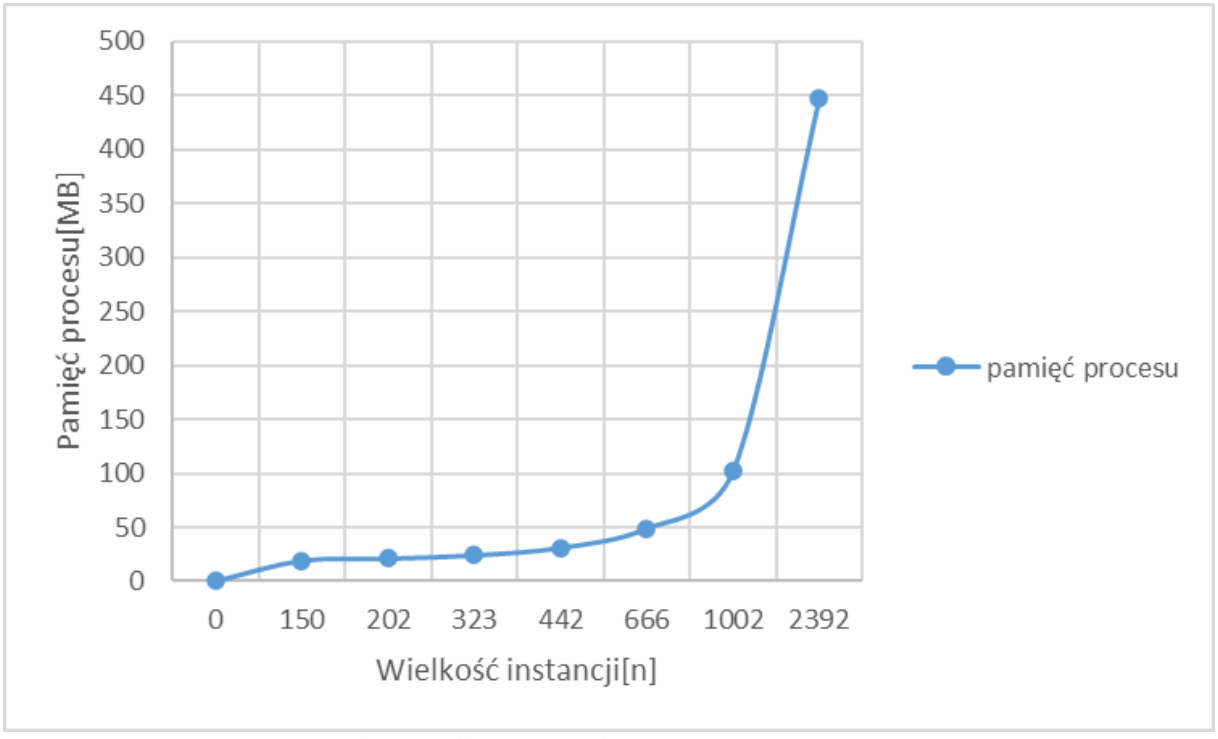
Wykres 10. Wielkość instancji od wartości błędu dla różnych heurystyk

6.4. Pomiar złożoności pamięciowej

Nazwa instancji	***Pamięć procesu***
Mniejsze	***<=18MB***
ch150.tsp	***19MB***
gr202.tsp	***21 MB***
rbg323.atsp	***24 MB***
pcb442.tsp	***31 MB***
gr666.tsp	***49 MB***
pr1002.tsp	***102 MB***
pr2392.tsp	***447 MB***

Tabela 1. Pomiary zajętości pamięciowej badanych instancji

Wyniki zostały zebrane po badaniu zajętości pamięciowej procesu. Złożoność większych instancji może nie być do końca zgoda ze stanem rzeczywistym, gdyż aby osiągnąć jak najlepsze czasy w stosunku do jakości, liczba mrówek oraz iteracji (od której złożoność pamięciowa głównie zależy) została ograniczona.



Wykres 9. Wielkość instancji od zajętości pamięciowej programu

7. Analiza wyników i wnioski

Z wykresów obrazujących zależności czasowe od wielkości instancji (wykresy 1., 3., 5., 7. oraz 9.) można zauważyć, że żaden z badanych parametrów ani wybór metody czy heurystyki nie wpływa znacząco na czas wykonywania algorytmu. Jest to zgodne z teorią, gdyż krzywe na wykresach przyjmują postać wykładniczą oraz jak podano w punkcie 2., złożoność czasowa algorytmu mrówkowego to w najlepszym wypadku $O(CC \cdot n^3)$, a w tym wzorze nie występuje żaden z badanych parametrów (α, β ani ρ).

Jak podaje Marco Dorigo, parametr α powinien być równy 1. Według wykresu 2. taka wartość parametru daje najbardziej stabilne wyniki (wykres najbardziej przypomina krzywą). Natomiast wraz z jej wzrostem wykres zaczyna być losowy, przypominać sinusoidę. Podobnie się dzieje, gdy alfa wynosi 0. Można z tego wnioskować, że jeśli alfa nie ma optymalnej wartości, mrówki częściej wybierają losowo, a nie według zadanego kryterium, co potwierdza tezę o tym, że α określa wpływ doświadczeń poprzednich pokoleń mrówek.

Dane zamieszczone na wykresie 4. obrazują, że wraz ze wzrostem parametru beta aż do optymalnej wartości równej 5, wyniki stają się coraz bardziej dokładne, wartość błędu maleje. Dla wielkich instancji, rozmiaru 2400, jest to różnica nawet ok. 4 tys. % względem wartości optymalnej. Potwierdza to tezę, że β określa siłę wyboru zachłannego drogi na podstawie odległości między miastami.

Wykres 6., prezentujący zależność współczynnika parowania feromonu ρ od błędu wykazuje, że gdy zbliża się on do wartości 1, algorytm daje najmniej dokładne wyniki co, podobnie jak wcześniej wspomniany współczynnik alfa, może powodować omijanie kryterium wyboru przez mrówki.

Rozkład feromonu *QAS* zdaje się dawać uśrednione wyniki spośród wszystkich metod, podczas gdy *DAS* najlepsze wyniki osiąga dla największych instancji (optymalnie od rozmiaru 666), a efektywność CAS spada wraz ze wzrostem rozmiaru instancji (wykres 8.).

Heurystyka *visibility* podana na wykładzie osiąga bardziej stabilne wyniki, podczas gdy wymyślona *frequency factor* losowe, co prezentuje wykres 10. Może to mieć związek z zaburzaniem kryterium wyboru mrówek przez wymyśloną metodę, znakiem czego, heurystyka *visibility* bardziej nadaje się do testowania algorytmu na większą skalę.

Złożoność pamięciowa algorytmu rośnie wykładniczo. Jest to powiązane z dużą ilością zagnieżdżonych pętli w programie wymaganych do obliczeń (punkt 3.).

Algorytm mrówkowy osiąga procent błędu ok. 65%, nawet w przypadku podania bardzo małych wartości liczby mrówek oraz iteracji dla instancji przekraczających rozmiar 350. Wobec tego może bardzo szybko znaleźć zadowalające rozwiązanie, lub jeśli wspomniane parametry zostaną znacznie zwiększone, rozwiązanie bliskie optymalnemu, kosztem dużo większego czasu obliczeń (zależność wykładnicza).