Projektowanie Efektywnych Algorytmów

Projekt

24/01/2024

264067 Mateusz Waszczuk

(7) ACO

1.Sformułowanie zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywności czasowej, pamięciowej oraz jakości rozwiązań w zależności od ilości wierzchołków w algorytmu ant colony optimization (ACO) rozwiązującego problem komiwojażera w wersji optymalizacyjnej. Algorytm ma wykorzystywać heurystykę visibility, schemat rozkładu feromonu DAS, a także działać z maksymalnym błędem:

- dla $n < 25$, 0%,
- dla $24 < n < 74$, 50%,
- dla $75 < n < 449$, 100%,
- dla $450 < n < 2500$, 150%.

Gdzie n jest liczbą wierzchołków w grafie.

Zaimplementowany program ma mierzyć czas wykonania algorytmu ACO dla każdej instancji oraz obliczać koszt znalezionego cyklu Hamiltona w celu obliczenia różnicy między rozwiązaniem optymalnym. Dane wejściowe dotyczące ilości powtórzeń dla każdej instancji zostaną umieszczone w pliku inicjującym pomiary.ini. Wyniki działania algorytmu mają zostać zwrócone w pliku pomiary.csv. Po zaimplementowaniu algorytmu ACO następnym zadaniem będzie przeprowadzanie testów poprawności jego działania oraz określenie liczby pomiarów dla wybranych instancji na podstawie czasu pojedynczego wykonania, co zamierzam zrobić jednym plikiem inicjującym wstępne_testy.ini. Ostatnią i najważniejszą częścią zadania jest zbadanie zależności między ilością wierzchołków w grafie, a czasem wykonania, dokładnością i zużyciem pamięci.

2.Metoda

Metoda Ant Colony Optimization (ACO), podobnie jak metoda Simulated Annealing (SA), jest jedną z technik heurystycznych używanych do rozwiązywania problemów optymalizacyjnych. W przeciwieństwie do podejść siłowych czy metod Branch and Bound, ACO nie dąży do znalezienia rozwiązania optymalnego, ale zamiast tego skupia się na dostarczeniu akceptowalnych rozwiązań w rozsądnym czasie. Pomimo tego, że nie gwarantuje optymalności, metoda ta ma zdolność do efektywnego rozwiązywania dużych instancji problemów optymalizacyjnych, czego nie można powiedzieć, o algorytmach brute force czy b&b, a dla niewielkich instacji przy odpowiednim dostrojeniu, zwrócone wyniki są bardzo często optymalne. Algorytm ACO działa na zasadzie symulacji zachowania mrówek, które poszukują najkrótszej ścieżki pomiędzy źródłem pokarmu a mrowiskiem, pozostawiając przy tym feromony na trasie. Prawdziwe feromony to substancje chemiczne tworzone i rozpoznawane przez mrówki stanowiące informacje o jakości ścieżki, które mrówki pozostawiają w środowisku. Algorytm odwzorowuje ten sposób działania i również używa feromony, a dokładniej ich wirtualny odpowiednik. W moim przypadku jest to dwuwymiarowy vector ze zmiennymi typu double informującymi o poziomie feromonu na krawędzi.
2.1 Hipoteza

Złożoność czasowa algorytmu ACO zależy od wielu czynników, w tym od liczby wierzchołków (n), liczby mrówek w symulacji (m), oraz liczby iteracji algorytmu. W najprostszym przypadku, można by ją przedstawić jako O(m*n^2). Można więc się spodziewać, że wykres funkcji czasu od ilości wierzchołków będzie jakąś funkcją kwadratową. Współczynnik C będzie tym większy im więcej potrzeba wstępnych operacji, jak inicjalizacja zmiennych. Z uwagi, że pisząc algorytm przywiązywałem uwagę do tego aspektu, nie spodziewam się aby był on istotny.

W kwestii błędu pomiaru istnieje bardzo wiele zmiennych które na to wpływają, z samą charakterystyką grafu włącznie, a badane były przeze mnie instancje symetryczne, asymetryczne a wartości w grafach znajdowały się w skrajnie odmiennych od siebie przedziałach.. Z uwagi na to, grafy rzędu kilkuset wierzchołków będą prawdopodobnie zbyt małe, żeby dokładnie określić jak kształtuje się zależność błędu od liczby wierzchołków. Można jednak wysunąć hipotezę, że mimo wszystko błąd będzie mieć tendencję wzrostową wraz ze wzrostem liczby wierzchołków.

Warto także wspomnieć o ostatniej badanej zależności-złożoności pamięciowej. Algorytm ACO wymaga przechowywania więcej informacji niż same grafy i pojedyncze zmienne, jak choćby informacji o ścieżkach feromonowych, co może wymagać zauważalnych ilości pamięci, która powinna rosnąć ze złożonością klasy O(n*n). Jednakże należy też bardzo mocno podkreślić, że struktury takie jak vectory, nawet wielowymiarowe w liczbie użytej w algorytmie nie spowodują złożoności, która dla badanych instancji będzie jakkolwiek istotna. Należy przez to rozumieć, że próbując rozwiązać instancje o rozmiarach które potencjalnie mogłyby to spowodować będą się wykonywać tak niewyobrażalnie długo, że w praktyce do wyczerpania pamięci nie dojdzie nigdy.

Teraz przejdę do omówienia sposobu implementacji istotnych elementów mojego algorytmu-heurystyki visibility, schematu rozkładu feromonu, widoczności, atrakcyjności i mechanizmu ruletki.

2.1 Heurystyka Visibility

Heurystyka visibility odnosi się do widoczności czyli inaczej odwrotności odległości między dwoma wierzchołkami grafu. W moim programie widoczność jest używana do późniejszego obliczenia atrakcyjności trasy mrówki. Im mniejsza odległość między dwoma wierzchołkami, tym większa widoczność, co z kolei zwiększa atrakcyjność trasy. Im bliżej są sąsiednie wierzchołki, tym większa jest atrakcyjność trasy, co zgodnie z zasadą algorytmu mrówkowego prowadzi do większego prawdopodobieństwa wyboru ścieżki przy losowaniu kolejnego wierzchołka w ścieżce mrówki.

2.2 Schemat Rozkładu Feromonu

W algorytmie ACO, feromony odgrywają kluczową rolę w przekazywaniu informacji o jakości tras mrówkom. Informują bowiem mrówki o jakości dostępnych dla nich tras. Istnieją 3 metody rozkładu feromonów na krawędziach są to, DAS (ant-density), QAS (ant-quantity) i CAS (ant-cycle). Z uwagi, że w użyłem w swoim programie schematu CAS, omówię po krótce tylko tą metodę. Jej podstawową cechą jest to, że każda mrówka posiada tą samą ilość feromonu która po podzieleniu przez długość znalezionej trasy, dodawana jest do każdej krawędzi przez którą mrówka szła. Dodawanie odbywa się po każdej iteracji algorytmu, stąd każda kolejna grupa mrówek posiada informacje zostawione po grupie poprzedniej. Jednocześnie wraz z aktualizacją ilości feromonów następuje ich parowanie.

2.3 Feromony i widoczność, a atrakcyjność

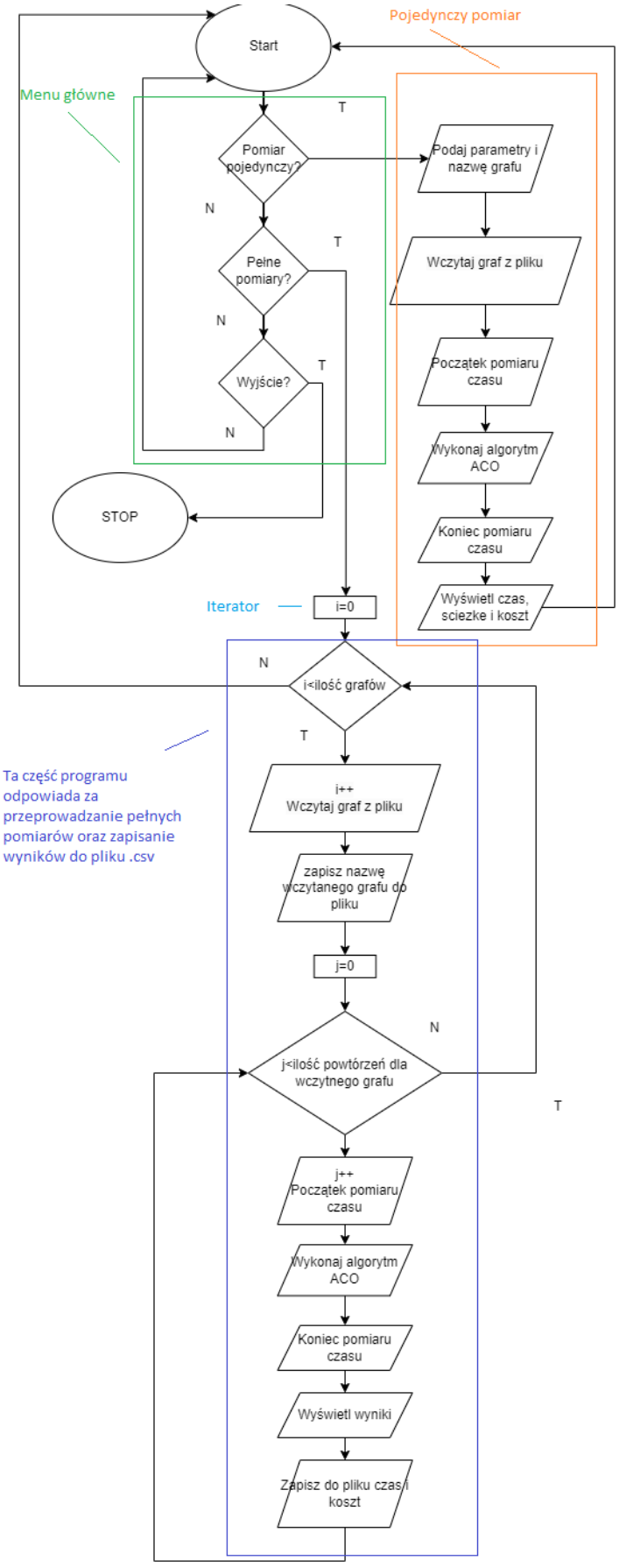
Wspomniane wcześniej feromony oraz widoczność (visibility) są (wraz z parametrami alfa i beta) składowymi wyliczanej następnie dla wierzchołków atrakcyjności. Atrakcyjność jest wyrażana następującym wzorem: $atrakcyjność = feromon^{alfa} \times visibility^{beta}$

2.4 Mechanizm ruletki

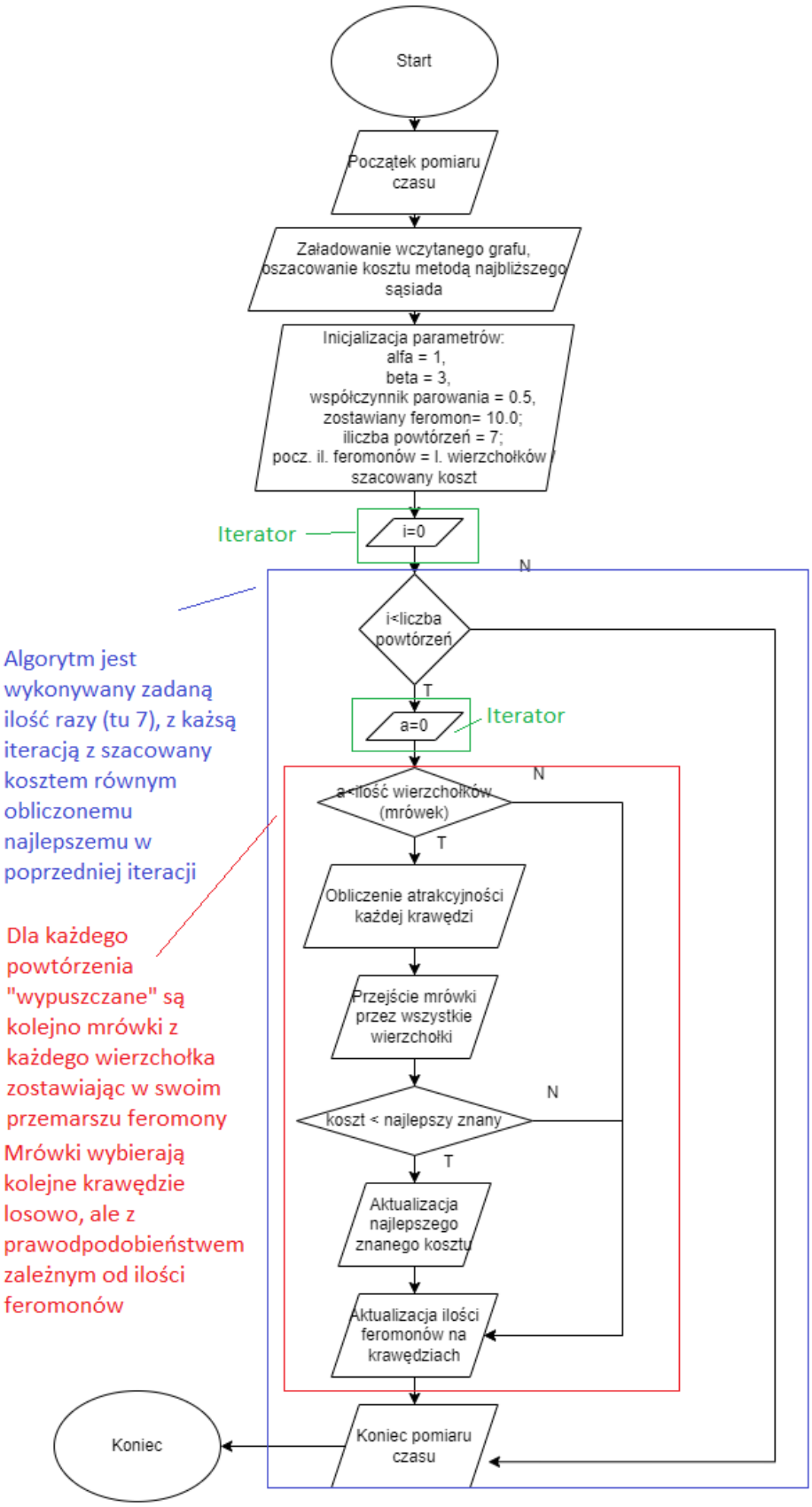
Algorytm mrówkowy w swoich założeniach narzuca pogodzenie dwóch różnych sposobów wyboru kolejnych wierzchołków-losowania oraz wyboru wierzchołków na podstawie ich atrakcyjności. Aby to wykonać w swojej implementacji użyłem przybliżonego programowego odwzorowania mechanizmu gry jaką jest rosyjska ruletka, z dość istotną modyfikacją. Działanie mojego algorytmu można przyrównać do takiej gry, w sytuacji, gdy bęben rewolweru jest nierównomiernie obciążony. Wówczas, to która komora bębna rewolweru zostanie wybrana jest losowe, ale z uwagi na obciążenie prawdopodobieństwo wyboru nie jest jednakowe dla wszystkich komór. Zaimplementowany przeze mnie mechanizm działa w podobny sposób. Następny wybrany, z dostępnych wierzchołków jest losowany, ale prawdopodobieństwo dla każdego z nich jest zależne od jego atrakcyjności. Częściej są więc wybierane wierzchołki bardziej atrakcyjne, ale nie są także wykluczone wierzchołki mniej atrakcyjne.

3.Algorytm

Na rysunku 1 znajduje się ogólny schemat blokowy całego programu który realizuje pomiary, zaś na rysunku 2 znajduje się schemat blokowy części programu odpowiedzialnej za samą logikę algorytmu ACO. Algorytm do znalezienia rozwiązania początkowego jak już wspomniano używa algorytmu najbliższego sąsiada. Funkcja heurystycznyKosztPoczatkowy która znajduje rozwiązanie początkowe wyżej wspomnianym sposobem jest na rysunku 2 jedynie wywoływana w odpowiednim miejscu, natomiast z uwagi że algorytm najbliższego sąsiada nie jest przedmiotem niniejszego opracowania nie posiad dokładnego schematu blokowego. Sam algorytm ACO jest też stosunkowo obszerny w porównaniu z poprzednimi implementacjami, stąd też schemat blokowy nie jest dokładnym schematem blokowym kodu, lecz jego przybliżonym odwzorowaniem skupiającym się na istotnych z punktu widzenia logiki działania algorytmu elementach, przy jednoczesnym pominięciu elementów koniecznych do poprawnego działania programu, lecz nie mających istotnego znaczenia w samej logice algorytmu.



Rysunek 1: Ogólny schemat blokowy całego programu



Rysunek 2: Schemat blokowy algorytmu SA

4.Dane testowe

Do sprawdzenia poprawności działania algorytmu oraz do określenia zestawu instancji do pomiarów użyto następujący zestaw instancji:

-tsp_10.txt, min koszt: 212

-tsp_14.txt, min koszt: 282

-gr21.tsp, min koszt: 2707

-bayg29.tsp, min koszt: 1610

-ftv33.atsp, min koszt: 1286

-ftv44.atsp, min koszt: 1613

-ft53.atsp, min koszt: 6905

-ftv70.atsp, min koszt: 1950

-rbg323.atsp, min koszt: 1326

-pcb442.tsp, min koszt: 50778

-rbg443.atsp, min koszt: 2720

-pr1002.tsp, min koszt: 259045

-pr2392.tsp, min koszt: 378032

https://github.com/PrzemekRychter/PEA

Po przetestowaniu programu za pomocą wszystkich podanych instancji koszty były zbliżone do podanych, choć poza instancjami do 25 wierzchołków wyższe niż najlepsze znane. Jest to całkowicie zgodne z oczekiwaniami wobec tego algorytmu. Nie udało się wykonać algorytmu dla największej z instancji (2392 wierzchołków) w założonym dopuszczalnym czasie 10 minut. Została więc wykluczona przy ustalaniu zestawu do właściwych pomiarów. Z uwagi na stosunkowo duży w porównaniu do algorytmów B&B i BF przedział badanych instancji, zdecydowano o rozszerzeniu zestawu używanego do testowania poprawności działania algorytmu i dodano jeszcze dodatkowe instancje, celem uzyskania lepszej jakości pomiarów i dalszych badań. Do przeprowadzenia właściwych pomiarów wybrano zatem następujący zestaw instancji:

-tsp_10.txt, min koszt: 212

-tsp_14.txt, min koszt: 282

-gr21.tsp, min koszt: 2707

-bayg29.tsp, min koszt: 1610

-ftv33.atsp, min koszt: 1286

-ftv44.atsp, min koszt: 1613

-ft53.atsp, min koszt: 6905

-ftv70.atsp, min koszt: 1950

-pr107.tsp, min koszt: 44303

-pr136.tsp, min koszt: 96772

-pr144.tsp, min koszt: 58537

-pr152.tsp, min koszt: 73682

-rat 195.tsp, min koszt: 2323

-gr202.tsp, min koszt: 40160

-gr229.tsp, min koszt: 134602

-gil262.tsp, min koszt: 2378

-a280.tsp, min koszt: 2579

-rbg323.atsp, min koszt: 1326

-pcb442.tsp, min koszt: 50778

-rbg443.atsp, min koszt: 2720

-pr1002.tsp, min koszt: 259045

-pr2392.tsp, min koszt: 378032

5.Procedura badawcza

Należało zbadać wpływ ilości wierzchołków na czas wykonania, błąd oraz zużycie pamięci. W przypadku algorytmu realizującego algorytm ACO dla przestrzeni rozwiązań dopuszczalnych występowały parametry programu, które mogły mieć wpływ na czas i jakość uzyskanego wyniku. Parametry alfa, beta, współczynnik parowania, liczba mrówek oraz sposób obliczania początkowej ilości feromonów zostały zainicjalizowane zgodnie z narzuconymi wartościami:

$\alpha = 1.0$

$\beta = 3.0$

$\rho = 0.5$

$liczba\ mrówek = liczba\ wierzchołków$

$\tau_0 = \frac{m}{C^{nn}}, gdzie\ C^{nn} jest\ szacowaną\ długością\ trasy$

Ilość powtórzeń algorytmu została eksperymentalnie ustalona na 7. Była to pierwsza wartość dla której dla wszystkich badanych instancji do 25 wierzchołków błąd był zerowy, co było również jednym z wymagań. W związku z tym procedura badawcza polegała na uruchomieniu programu sterowanego plikiem inicjującym pomiary.ini: (format pliku : nazwa_pliku_z_grafem, ilość_powtórzeń). Wyniki były zapisywane w pliku pomiary.csv. Poniżej przedstawiono fragment zawartości jednego z plików inicjujących (wstępne_testy.ini):

tsp_10.txt 1

tsp_14.txt 1

gr21.tsp 1

bayg29.tsp 1

ftv33.atsp 1

ftv44.atsp 1

Każda instancji rozwiązywana była zgodnie z liczbą jej wykonań, w tym przypadku każda raz. Do pliku wyjściowego pomiary.csv zapisywany był czas wykonania pomiarów, błąd (procentowo) jak bardzo obliczony koszt odbiega od najlepszego znanego oraz nazwa instancji której te pomiary dotyczą. Plik wyjściowy zapisywany był w formacie csv. Dane znajdują się w następującym formacie: Nazwa instancji; Czas[ms]; Błąd[%];. Dla każdej instancji dodatkowo przed pierwszym pomiarem zapisywane są nazwy kolumn zgodnie z wyżej podanym formatem. Poniżej przedstawiono fragment zawartości pliku wyjściowego:

ftv33.atsp;Czas [ms];Blad [%]

;15;1,399689

ftv44.atsp;Czas [ms];Blad [%]

;46;10,291383

ft53.atsp;Czas [ms];Blad [%]

;78;24,663287

Pomiary zużycia pamięci zostały przeprowadzone przy użyciu menedżera zadań systemu Windows 10. Z tego powodu każdy pomiar był przeprowadzany poprzez wczytanie pliku inicjującego nakazującego programowi wykonanie algorytmu dla danej instancji 100000 razy dla zadanych parametrów. Zadeklarowanie dużej liczby powtórzeń miało na celu zapewnienie czasu na wykonanie ręcznego odczytu używanej przez proces pamięci RAM. Poniżej przykład zawartości pliku inicjującego dla pr152.tsp do pomiarów zużycia pamięci:

gr21.tsp 100000

Wyniki zostały opracowane w MS Excel.

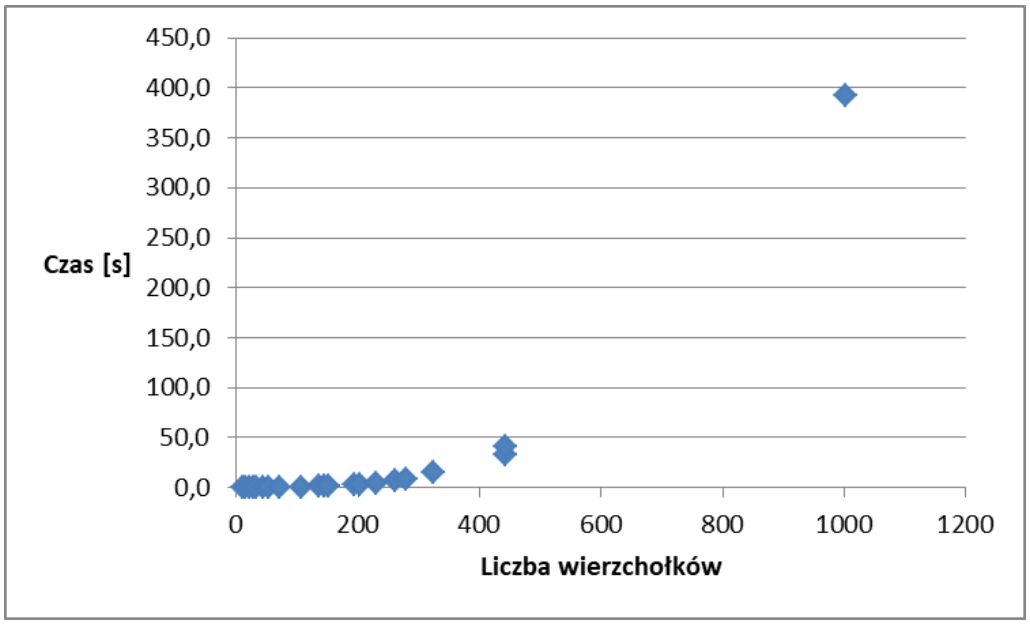
6.Wyniki

Wyniki zgromadzone zostały w plikach: pomiary.xlsx, pamiec.xlsx Wszystkie pomiary z ww. plików (uśrednione) znajdują się w tabeli 1, tabeli 2, tabeli 3 oraz te same pomiary (uśrednione oraz dla każdej instancji osobno) znajdują na dysku Google pod adresem: https://drive.google.com/drive/folders/1WV49rtOCrEiHnzpUSO6HhYhHfetYnGID?usp=sharing. Na podstawie wspomnianych wyżej pomiarów sporządzono rysunki 3-6. Dla każdego z punktów 6.1-6.3 sporządzono po 2 wykresy, każdorazowo dla wszystkich instancji oraz tych do 500 wierzchołków. Spowodowane jest to faktem, że ostatni pomiar (dla grafu o 1002 wierzchołkach) był tak silnie oddalony od przedostatniego pod względem liczby wierzchołków, że istotnie wpływało to na widoczność linii trendu oraz utrudniało dostrzeżenie w jaki sposób zmieniają się pomiary wraz ze wzrostem liczby wierzchołków. Przykładowo dla rysunku 3 nie sposób dostrzec w jaki sposób rosną wartości. Po sporządzeniu rysunku 4 widać jednak wyraźnie, że linia trendu jest funkcją kwadratową, a nie liniową z lokalnym zaburzeniem, co nie byłoby wcale takie jasne bazując jedynie na rysunku 3.

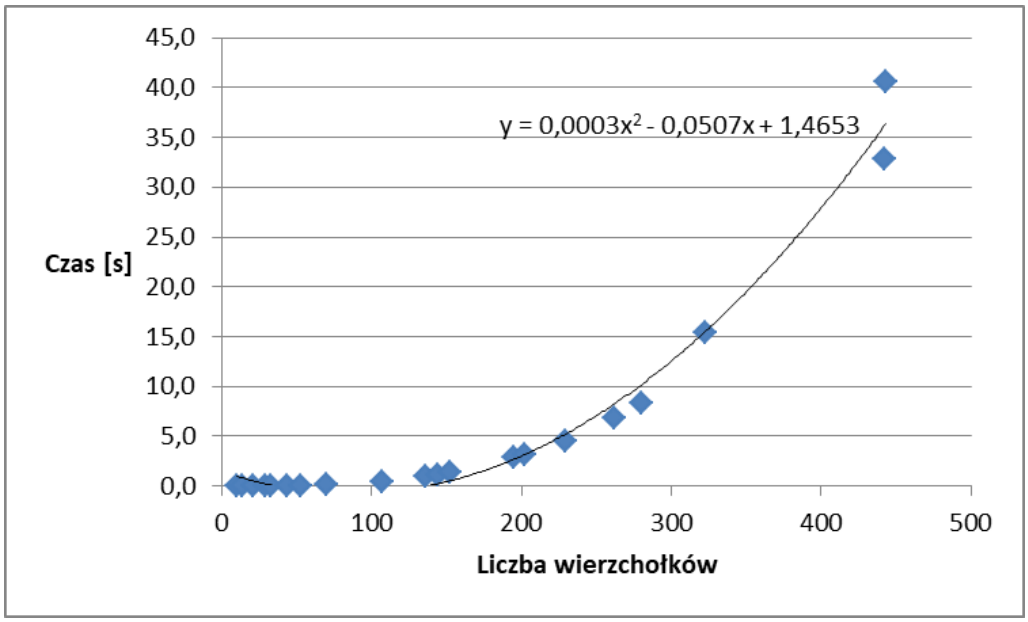
Tabela 1: Wyniki pomiarów czasu, pamięci i błędu dla algorytmu ACO

Liczba wierzcholkow	10	14	21	29	33	44	53	70	107	136	144	152	195	202	229	262	280	323	442	443	1002
Czas [s]	0,0	0,0	0,0	0,0	0,0	0,0	0,1	0,1	0,5	1,0	1,2	1,4	2,9	3,2	4,6	6,8	8,3	15,5	32,8	40,6	392,6
Błąd[%]	0,0	0,0	0,0	3,6	1,4	10,3	24,7	20,9	5,4	24,8	5,3	14,4	16,4	20,2	15,9	28,3	22,4	30,8	22,1	44,2	27,8
Zużycie pamięci [MB]	0,6	0,6	0,6	0,6	0,6	0,6	0,7	0,7	0,9	1,1	1,1	1,2	1,5	1,7	1,9	2,1	2,4	3,2	4,7	4,7	21

6.1 Wpływ ilości wierzchołków na czas pomiaru

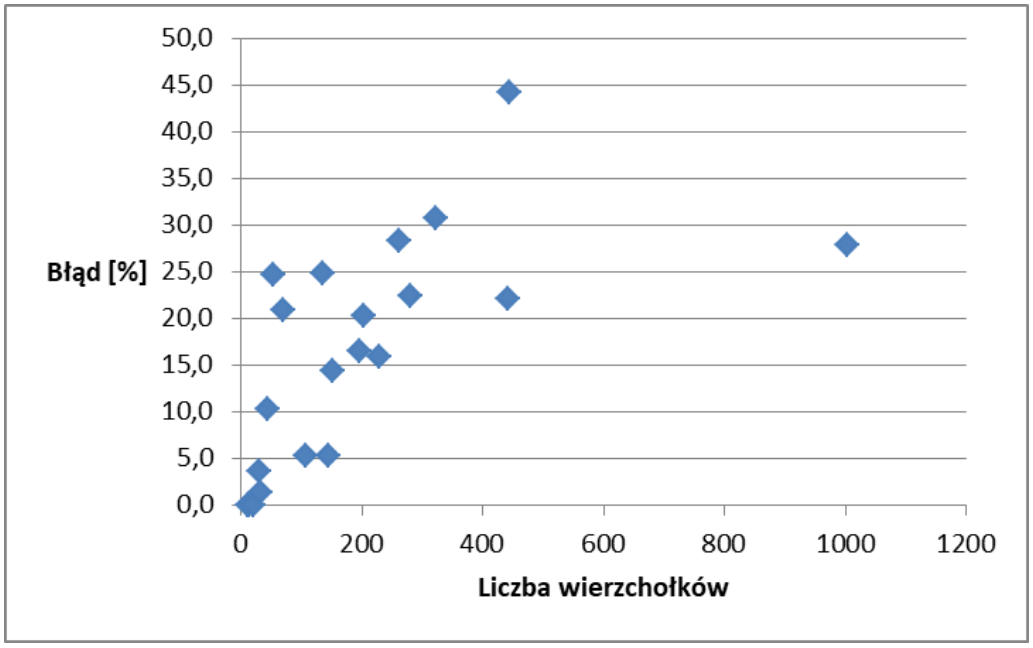


Rysunek 3: Wpływ liczby wierzchołków na czas wykonania

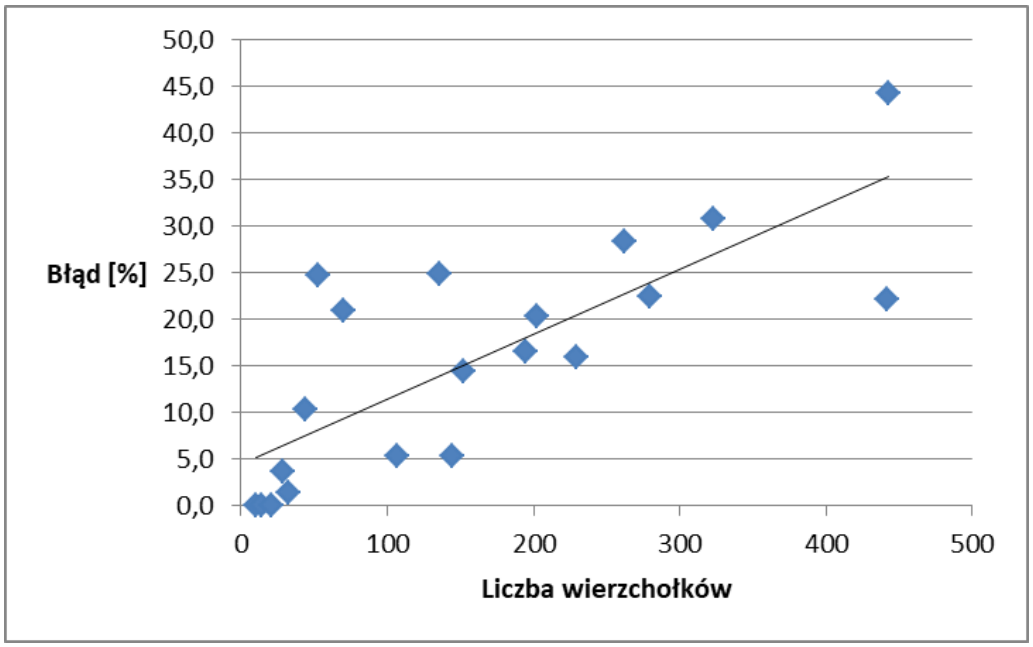


Rysunek 4: Wpływ liczby wierzchołów (do 500) na czas pomiaru

6.2 Wpływ ilości wierzchołków na błąd pomiaru

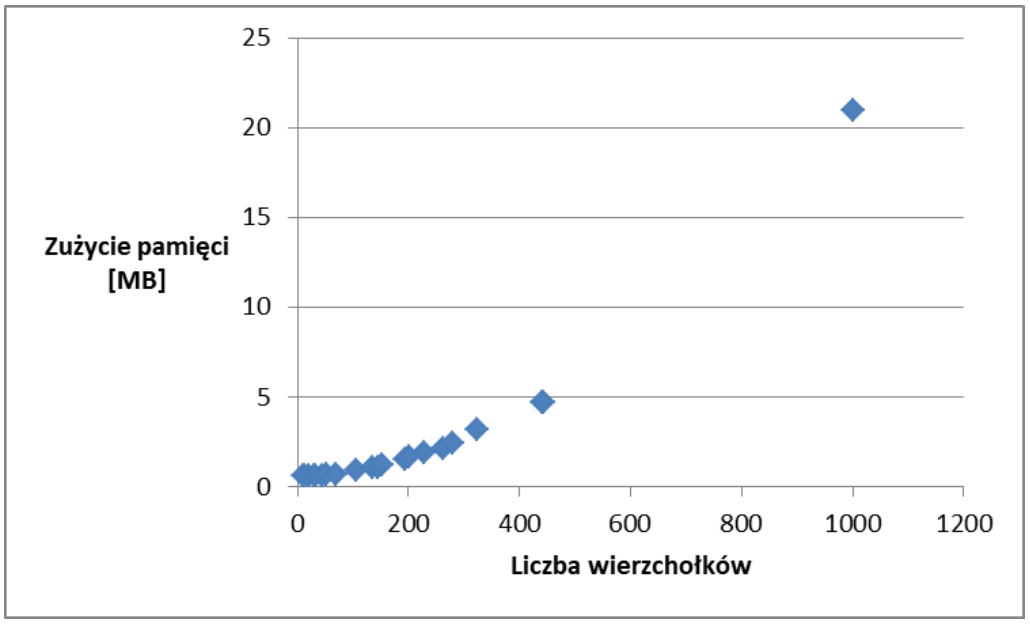


Rysunek 5: Wpływ liczby wierzchołków na błąd pomiaru

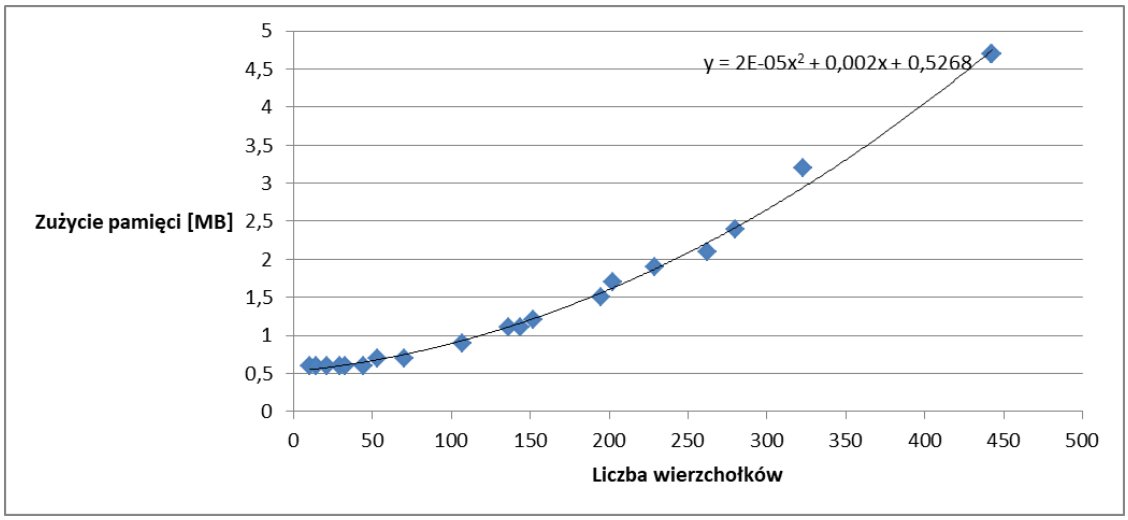


Rysunek 6: Wpływ liczby wierzchołków (do 500) na błąd pomiaru

6.3 Wpływ ilości wierzchołków na zużycie pamięci



Rysunek 7 : Wpływ liczby wierzchołków na zużycie pamięci



Rysunek 8: Wpływ liczby wierzchołków (do 500) na zużycie pamięci

6.4 Platforma sprzętowa

Pomiary zostały przeprowadzone na sprzęcie o następujących parametrach:

- Procesor Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz 2.50 GHz
- RAM 8GB
- System operacyjny Windows 10 64-bit, procesor x64

Czas został zmierzony za pomocą biblioteki std::chrono, natomiast zużycie pamięci za pomocą menedżera zadań systemu Windows 10.

7.Analiza wyników i wnioski dot. algorytmu SA

Wszystkie otrzymane wyniki nie wykazują cech mogących wskazywać na błędy w algorytmie lub instancjach. Nie stwierdzono podstaw aby wykluczyć jakiekolwiek z pomiarów i wszystkie wykonane przedstawiono na odpowiednich wykresach. Jedynie ostatnia z badanych instancji (1002 wierzchołków) została przedstawiona na nie wszystkich wykresach, z uwagi na duże oddalenie od pozostałych i istotny wpływ na czytelność.

Złożoność czasowa algorytmu SA dla badanych instancji jest zgodna z oczekiwaną i jej linia trendu tak jak można było przypuszczać jest funkcją kwadratową (Rysunek 4). Jednocześnie warto wskazać, iż współczynnik a linii trendu wynosi 0,003. Oznacza to, że mimo złożoności klasy O(n*n) wzrost czasu wraz ze wzrostem liczby wierzchołków jest bardzo niski i mimo takiej złożoności umożliwia wykonywanie algorytmu komiwojażera w akceptowalnym czasie nawet dla ponad 1000 wierzchołków. Warto też zaznaczyć, że jest to podstawowa wersja algorytmu, nie były prowadzone działania mające prowadzić do optymalizacji czasu, co jest kolejnym argumentem przemawiającym za wyjątkowo dobrą złożonością czasową. Jednocześnie zwrócone rozwiązania mają błąd nieprzekraczający 50% nawet dla najtrudniejszych pod tym względem instancji. Należy zaznaczyć, że największy uzyskany błąd wcale nie został uzyskany przy największej z instancji, dla największej, czyli 1002 wierzchołkowego grafu było to niecałe 30%. Można wywnioskować z tego, że przy odpowiednim dostrojeniu algorytmu pod konkretne problemy możliwe jest wykonanie również zwracających największy błąd instancji tak, aby rozwiązanie było dużo lepszej jakości. W porównaniu z poprzednimi badanymi algorytmami, ACO ma najwięcej parametrów którymi można operować, aby algorytm działał jak najlepiej dla specyficznych problemów.

Na koniec warto wspomnień o złożoności pamięciowej. Algorytm ACO wykazuje złożoność pamięciową klasy O(n*n), tak jak oczekiwano. Jednocześnie można zauważyć, że współczynnik a linii trendu jest niezwykle niewielki, a zużywana pamięć w najgorszym pod tym względem badanym przypadku to zaledwie niespełna 3 promile całości. Hipoteza mówiąca, że zużycie pamięci będzie można pominąć okazała się być trafna.

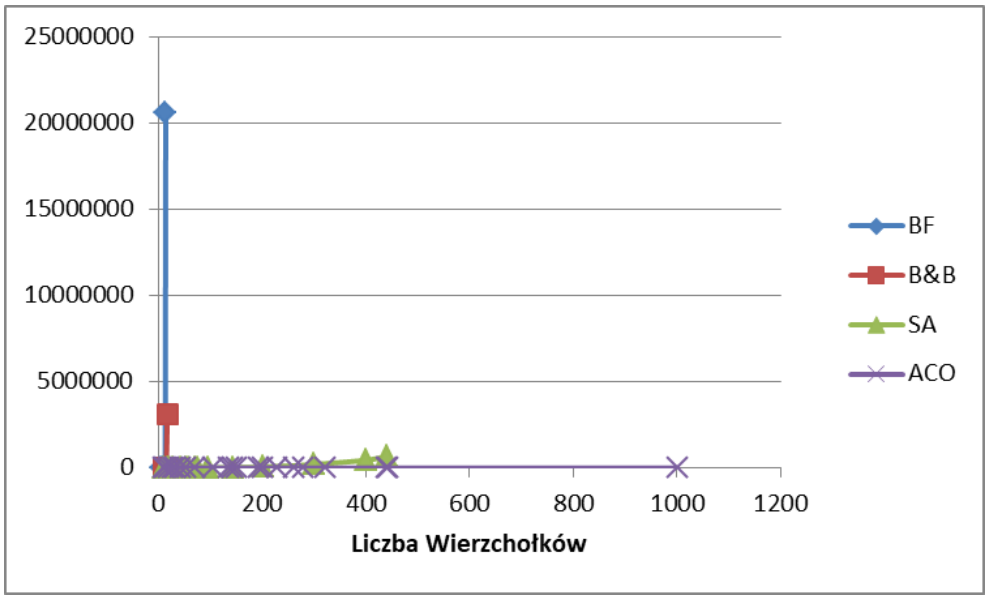
8.Porównanie algorytmów BF, B&B, SA i ACO

8.1 Pomiary

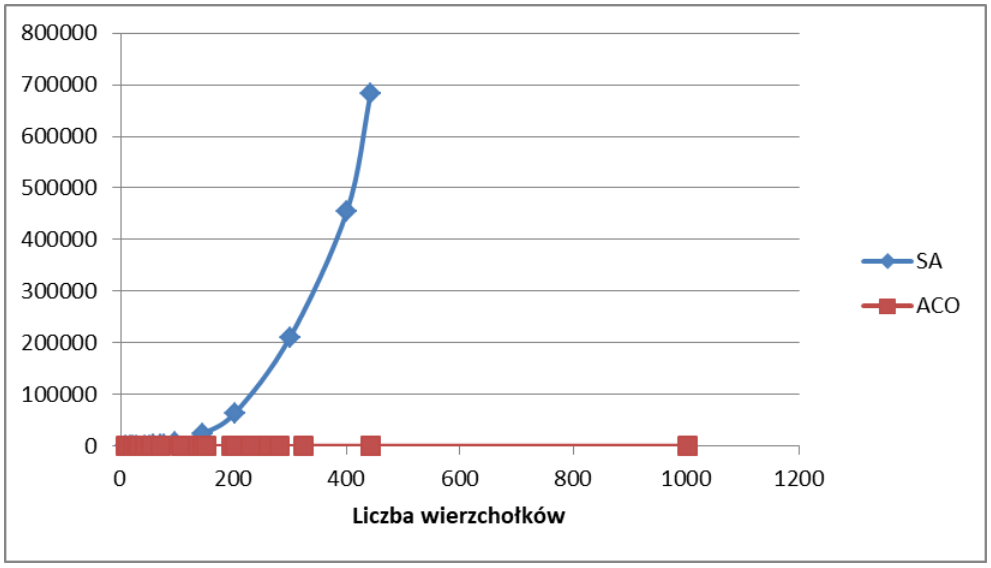
W celu porównania algorytmu ACO z dotychczas badanymi algorytmami BF, B&B i SA przedstawiono wykresy zużycia czasu w zależności od ilości wierzchołków (rysunki 9, 10) oraz zużycia pamięci (rysunek 11). W celu poprawienia czytelności wyżej wymienionych rysunków przedstawiono ponownie także rysunek 4. Dane służące do sporządzenia ww. wykresów znajdują się w tabelach 2 i 3.

Tabela 2: Porównanie czasu wykonania w zależności od liczby wierzchołków dla algorytmów BF, B&B, SA i ACO

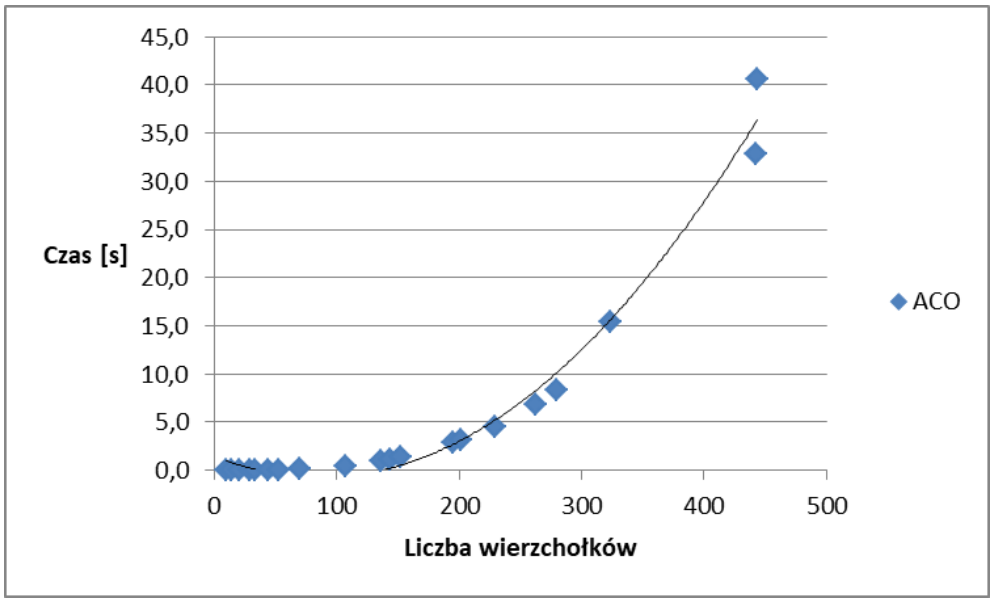
Brute Force		B&B		SA	
Liczba wierzchołków	Czas [ms]	Ilość wierzchołków	Czas [ms]	Liczba wierzchołków	Czas [ms]
6	0	10	6,02	10	12,7
10	688,74	12	188,34	17	51,5
12	104000	13	932,96	21	102,8
13	1410,3	14	5413,56	29	216,7
14	20552000	15	12000	42	512,1
-	-	17	3129400	52	1176,4
-	-	-	-	58	1797,7
-	-	-	-	70	1987,9
-	-	-	-	76	2394,2
-	-	-	-	96	7757,3
-	-	-	-	144	24323,2
-	-	-	-	202	64338,6
-	-	-	-	299	210305
-	-	-	-	400	455035,4
-	-	-	-	442	683557,2



Rysunek 9: Porównanie czasu wykonania dla algorytmów BF, B&B, SA i ACO



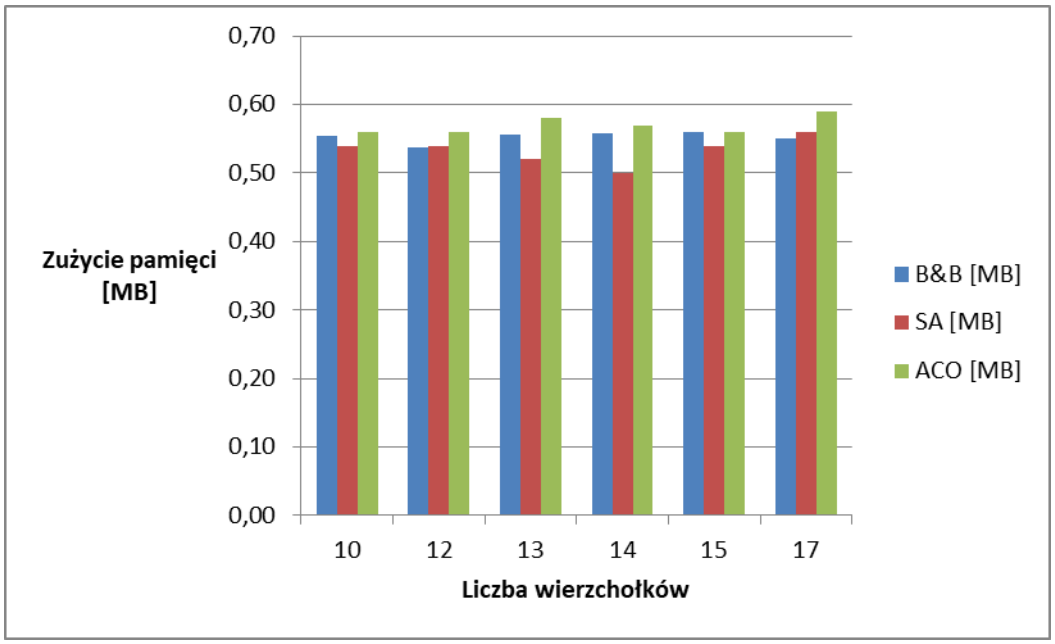
Rysunek 10: Porównanie czasu wykonania dla algorytmów SA i ACO



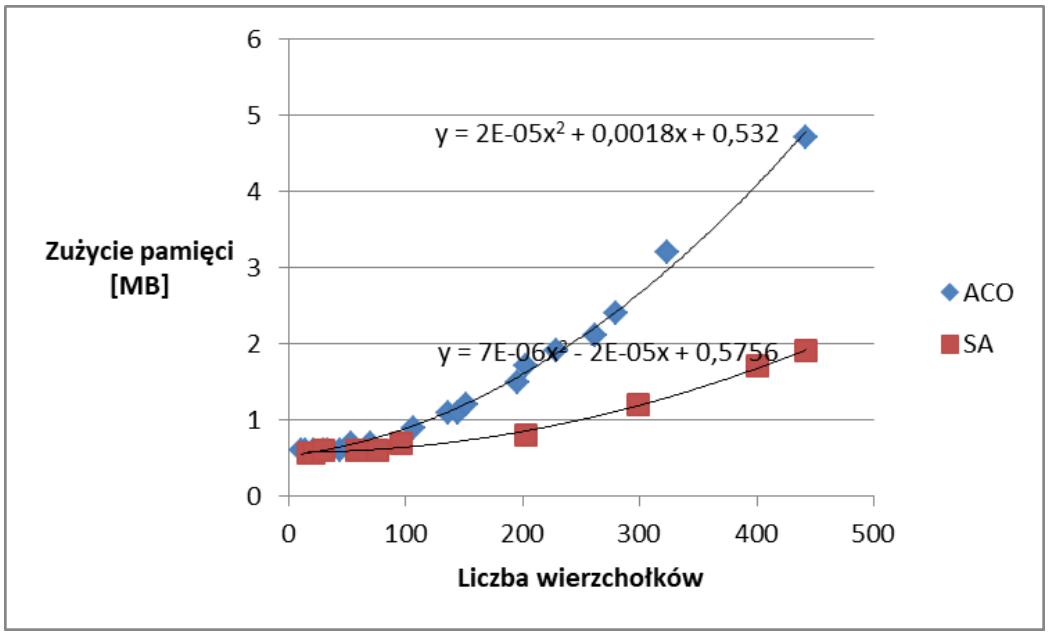
Rysunek 4: Wpływ liczby wierzchołów (do 500) na czas pomiaru dla algorytmu ACO

Tabela 3 Porównanie zużycia pamięci w zależności od liczby wierzchołków dla algorytmów B&B, SA i ACO

Instancja/l. wierzch	10	12	13	14	15	17
B&B [MB]	0,55	0,54	0,56	0,56	0,56	0,55
SA [MB]	0,54	0,54	0,52	0,50	0,54	0,56
ACO [MB]	0,56	0,56	0,58	0,57	0,56	0,59



Rysunek 11: Porównanie zużycia pamięci dla algorytmów B&B, SA i ACO



Rysunek 12: Porównanie złożoności pamięciowej algorytmów SA i ACO

8.2 Wnioski dotyczące porównania algorytmów B&B, BF, SA i ACO

Algorytm ACO okazał się wykazywać najlepszy stosunek czasu do jakości rozwiązania ze wszystkich badanych. Jego przewaga nad pozostałymi badanymi jest tak ogromna, że niezależnie jaki z badanych algorytmów zostanie przedstawiony na jednym wykresie wraz z algorytmem ACO na wykresie czasu od ilości wierzchołków w grafie, to wykres dla algorytmu ACO wydaje się być na równi z osią OX (rysunki 9, 10). W kwestii zużycia pamięci można go jedynie porównywać z algorytmem SA, ponieważ algorytmy B&B i BF nie wykonają się w akceptowalnym czasie dla liczby wierzchołków umożliwiających takie porównania (rysunek 11). Można zauważyć, że złożoność pamięciowa algorytmu ACO jest większa niż algorytmu SA, z uwagi na większą liczbę tymczasowych struktur, jak macierz feromonów, jednak gdy zwrócimy uwagę jakie są to wartości oraz jak mały jest współczynnik a linii trendu, można wysunąć wniosek, że dla badanych instancji zużycie pamięci można pominąć, tak samo jak dla algorytmu SA. Jest to zgodne z wysuniętą początkową hipotezą.