Projektowanie Efektywnych Algorytmów
Projekt
23/01/2024

263974 Szymon Kluska

(4) Algorytm mrówkowy

Spis treści	strona

1. Sformułowanie zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywności algorytmu opartego o metodę algorytmu mrówkowego rozwiązującego problem komiwojażera.
Problem komiwojażera (eng. travelling salesman problem, TSP) to zagadnienie polegające na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Cykl Hamiltona to cykl w grafie, w którym każdy wierzchołek grafu jest odwiedzany dokładnie raz oprócz wierzchołka startowego. Rozwiązanie polega na znalezieniu w grafie ścieżki o najmniejszym koszcie.

Hipotezy
Metoda algorytmu mrówkowego jest metodą heurystyczną co oznacza, że dąży do znalezienia rozwiązania o wartości funkcji celu jak najbliższej wartości optymalnej, ale nie gwarantuje znalezienia optymalnego rozwiązania. Jakość wyników może się różnić w zależności od wybranych schematów rozkładu feromonów, doboru wielkości parametrów oraz od instancji problemu.
Metoda algorytmu mrówkowego wykorzystuje tablice do zapamiętywania ilości feromonu na poszczególnej ścieżce o wielkości N x N, gdzie N jest wielkością instancji problemu, co oznacza, że wraz z wielkością wykorzystanie pamięci powinno wzrastać.

2. Metoda

Algorytm mrówkowy (ang. Ant Colony Optimization) to metaheurystyka oparta na analogii zaczerpniętej z funkcjonowania mrówek.
Mrówki są właściwie ślepe, ale mają bardzo dobry zmysł węchu. Podczas przemieszczania się do źródła pokarmu pozostawiają po sobie feromony. Ilość feromonów, które dana mrówka pozostawia są zależne od czynników takich jak długość trasy czy bezpieczeństwo. Służy to do komunikacji między mrówkami, które później sugerują się intensywnością zapachu podczas wytyczania trasy. Feromony z czasem wyparowują.
Metoda algorytmu mrówkowego naśladuje zachowania kolonii mrówek. Symuluje się zachowanie mrówki, która wybiera losowo początkowe miasto [5]. Następnie na podstawie funkcji prawdopodobieństwa wybiera kolejne miasta, które odwiedzi. Funkcja prawdopodobieństwa jest zależna od parametrów α i β, którymi steruje się, aby unikać ekstremów lokalnych.

Rodzaje algorytmu mrówkowego według aktualizacji feromonów

- CAS (Cycle Ant System)
  W algorytmie cyklicznym stała ilość feromonu Q jest dzielona przez długość drogi przebytej przez mrówkę i rozkładana na wszystkich krawędziach tej drogi po skonstruowaniu zadania.
- DAS (Density Ant System)
  W algorytmie gęstościowym stała ilość feromonu Q jest rozkładana na wszystkich krawędziach tej drogi bezpośrednio po przejściu krawędzi.

Parametry

- α – parametr odpowiadający za współczynnik wpływu ilości feromonu na prawdopodobieństwo wyboru wierzchołka
- β – parametr odpowiadający za współczynnik ważności heurystyki wyboru lokalnego
- ρ – parametr odpowiadający za współczynnik parowania feromonu
- m – liczba mrówek
- τ_0 – wartość początkowa ilości feromonów na każdej ścieżce

Dobór wartości początkowej ilości feromonów

Wartość τ_0 oblicza się ze wzoru $\tau_0 = \frac{m}{C^{nn}}$, gdzie m to liczba mrówek, a C^{nn} to szacowana długość trasy. W implementacji algorytmu do szacowania długości trasy wykorzystywany jest algorytm najbliższego sąsiada.

Wyparowywanie feromonów

Wyparowywanie feromonów służy do zapobiegnięcia nadmiernemu kumulowaniu się feromonów, które może wpłynąć na funkcje doboru kolejnego elementu rozwiązania (miasta).
Wzór na wyparowywanie feromonów [4]:
$\tau_{ij} = (1 - \rho)\tau_{ij}$, gdzie $i, j \in [0, n)$, a n to liczba instancji

Wybór kolejnego miasta

Z początkowego (wylosowanego) miasta mrówki dobierają kolejne miasta na podstawie funkcji prawdopodobieństwa [5]

$$p_{ij} = \frac{\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}{\sum_{k \in S} \tau_{ik}^{\alpha} * \eta_{ik}^{\beta}},$$

gdzie

- i – miasto, w którym obecnie znajduje się mrówka
- j - potencjalnie kolejne miasto
- S – zbiór miast, które pozostały do wyboru
- τ_{ij} - ilość feromonów na ścieżce z miasta i do j
- η_{ij} - heurystyka wyboru lokalnego dla ścieżki z i do j

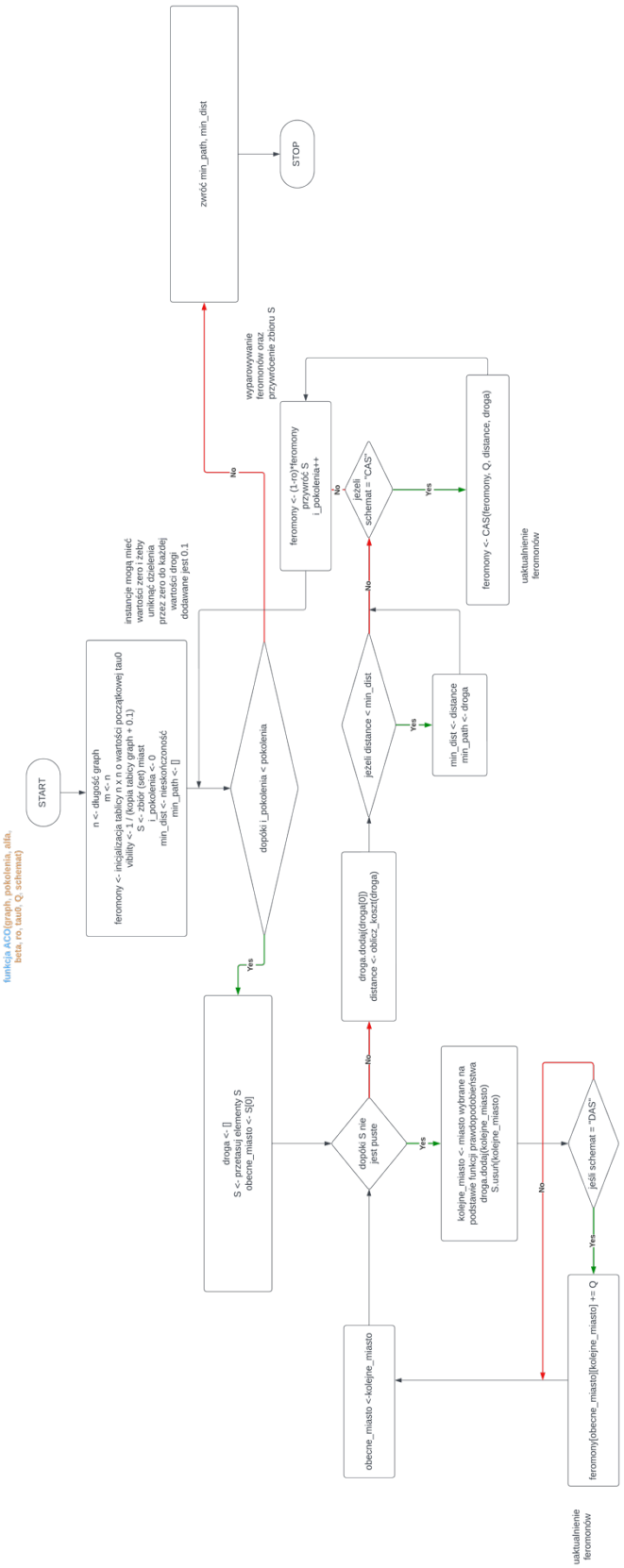
Kryterium zakończenia

Kryterium zakończenia może być:

- przekroczenie ustalonego limitu czasowego
- brak poprawy wyniku przez określoną liczbę iteracji
- przekroczenie ustalonej liczby iteracji wykonywania algorytmu.

Liczbę iteracji wykonywania algorytmu określa się jako liczba pokoleń mrówek. W implementacji algorytmu jest wykorzystywane kryterium zakończenia po ustalonej liczbie pokoleń mrówek.

3. Algorytm



Rysunek 1 Schemat blokowy algorytmu dla metody algorytmu mrówkowego

4. Dane testowe

Do sprawdzenia poprawności działania algorytmu wybrano następujący zestaw danych instancji:

1. tsp_6_1.txt 20 132 [0, 1, 2, 3, 4, 5, 0]
2. tsp_6_2.txt 20 80 [0, 5, 1, 2, 3, 4, 0]
3. tsp_10.txt 20 212 [0, 3, 4, 2, 8, 7, 6, 9, 1, 5, 0]
4. tsp_12.txt 20 264 [0, 1, 8, 4, 6, 2, 11, 9, 7, 5, 3, 10, 0]

http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/

Do wykonania badań wybrano następujący zestaw instancji:

1. gr17.txt 20 2085
2. gr21.txt 20 2707
3. ftv33.txt 20 1286
4. ft53.txt 20 6905
5. ftv70.txt 20 1950
6. gr96.txt 20 55209
7. ftv170.txt 20 2755
8. gr202.txt 20 40160
9. rbg323.txt 20 1326
10. pcb442.txt 20 50778
11. rbg443.txt 20 2720

http://softlib.rice.edu/pub/tsplib/tsp/

Dla powyższych danych z tsplib na stronie podane są tylko optymalne koszty (bez ścieżek).
http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf

5. Procedura badawcza

Należało zbadać wpływ schematu rozprzestrzeniania feromonu (metodą CAS lub DAS) oraz wpływu ustawienia parametrów alfa i beta.
Zestaw badanych konfiguracji parametrów:

- Wg Dorigo
  - Alfa = 1
  - Beta = 3.5
  - ro = 0.5
- Alfa > Beta
  - Alfa = 5
  - Beta = 1
  - ro = 0.5
- Beta > Alfa
  - Alfa = 1
  - Beta = 5
  - ro = 0.5

Dla każdej konfiguracji były badane schematy rozkładu feromonów CAS i DAS. Ilość pokoleń została ustawiona na 40.
Procedura badawcza polegała na uruchomieniu programu sterowanego plikiem konfiguracyjnym test_atsp.ini. Dla każdego pliku testowego dla każdej konfiguracji parametrów metoda algorytmu mrówkowego została wykonana 20 razy, a dla instancji większych niż n > 300 została wykonana 10 razy, tak aby czas dla pojedynczej konfiguracji nie przekraczał ustalonego czasu (około godziny).
Program został napisany w języku Python 3.10, a generowanie wyników procedury badawczej zostało przeprowadzone na MacBooku Pro o procesorze Apple M1, 16GB RAM i systemie macOS Sonoma 14.1.
Do pomiaru czasu została użyta biblioteka time [2], a do pomiaru zużycia pamięci biblioteka memory_profiler [3].

```python
def perform_method_aco(file_name: str, method: (), parametry):
    from time import perf_counter
    from AntColony.ACO import nearest_neighbour
    graph = file_to_graph_aco(file_name)
    start = perf_counter()
    est_path, est_cost = nearest_neighbour(graph)
    tau0 = len(graph) / est_cost
    memory_profiler.profile()
    # parametry = [float(alfa), float(beta), str(schemat)]
    path, dist = method(graph, 40, parametry[0], parametry[1], 0.5, tau0, est_cost, parametry[2])
    mem_usage = memory_profiler.memory_usage()
    stop = perf_counter()
    diff = stop - start
    return path, dist, diff, mem_usage
```

Rysunek 2 Fragment kodu funkcji perform_method_aco odpowiadający za pomiar czasu i pamięci w module pea_utils.py

Do pliku wyjściowego test_atsp_out.csv najpierw zapisywane były: nazwa pliku zestawu instancji, ilość powtórzeń i wybrane parametry. Następnie program zapisywał otrzymywane wyniki: ścieżkę, koszt wykonania ścieżki, czas wykonania funkcji algorytmu [w sekundach], zużycie pamięci [w MiB], długość epoki, temperaturę początkową i temperaturę końcową. Plik wyjściowy był zapisywany w formacie csv.

Poniżej przedstawiono fragment zawartości przykładowego pliku wyjściowego.

dane/gr17.txt	20	2085	[1.0, 5.0, 'CAS']
[9, 10, 2, 14, 13, 16, 5, 7, 6, 12, 3, 0, 15, 11, 8, 4, 1, 9]	2085	0.2963314169901423	[46.34375]
[4, 1, 9, 10, 2, 13, 14, 5, 7, 16, 6, 12, 3, 0, 15, 11, 8, 4]	2134	0.2677189579990227	[46.359375]
[14, 2, 13, 16, 7, 5, 6, 12, 3, 0, 15, 11, 8, 9, 1, 4, 10, 14]	2231	0.26930924999760464	[46.359375]
[4, 10, 2, 14, 13, 5, 16, 6, 7, 12, 3, 0, 15, 11, 8, 9, 1, 4]	2174	0.27038766600890085	[46.359375]

…

test_beta_cas.csv			

Wyniki zostały opracowane przy użyciu biblioteki pandas [6] i programu Microsoft Excel. Wielkość błędu względem optimum jest wyliczana ze wzoru

$$błąd = \frac{otrzymany\ wynik - optimum}{optimum} * 100\%$$

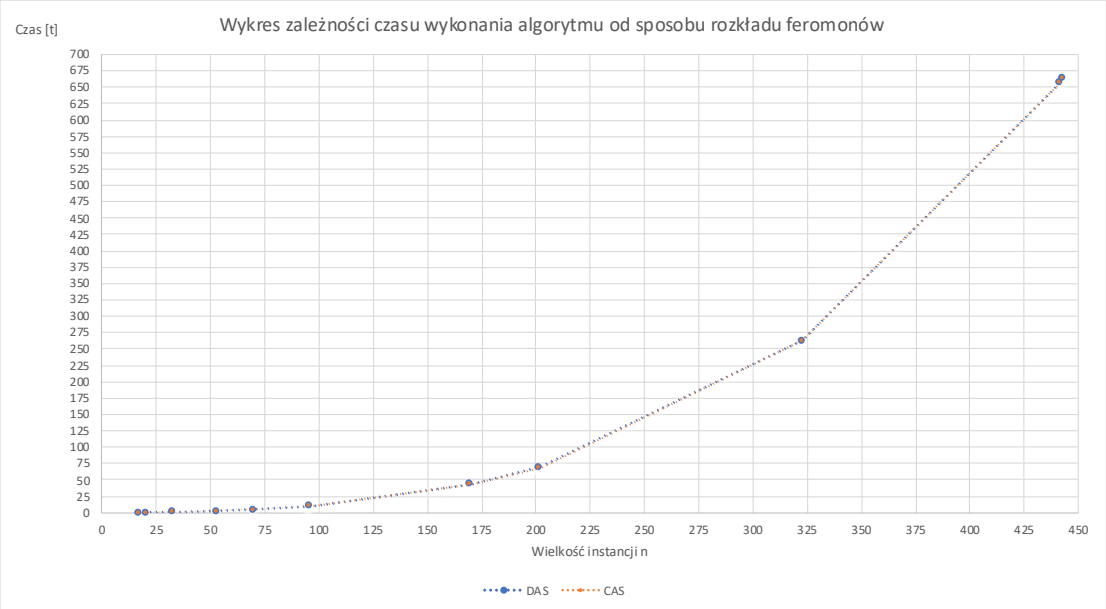
6. Wyniki

Wyniki przedstawione zostały w postaci wykresu zależności czasu uzyskania rozwiązania problemu od wielkości instancji i tabeli z dokładnymi wartościami.

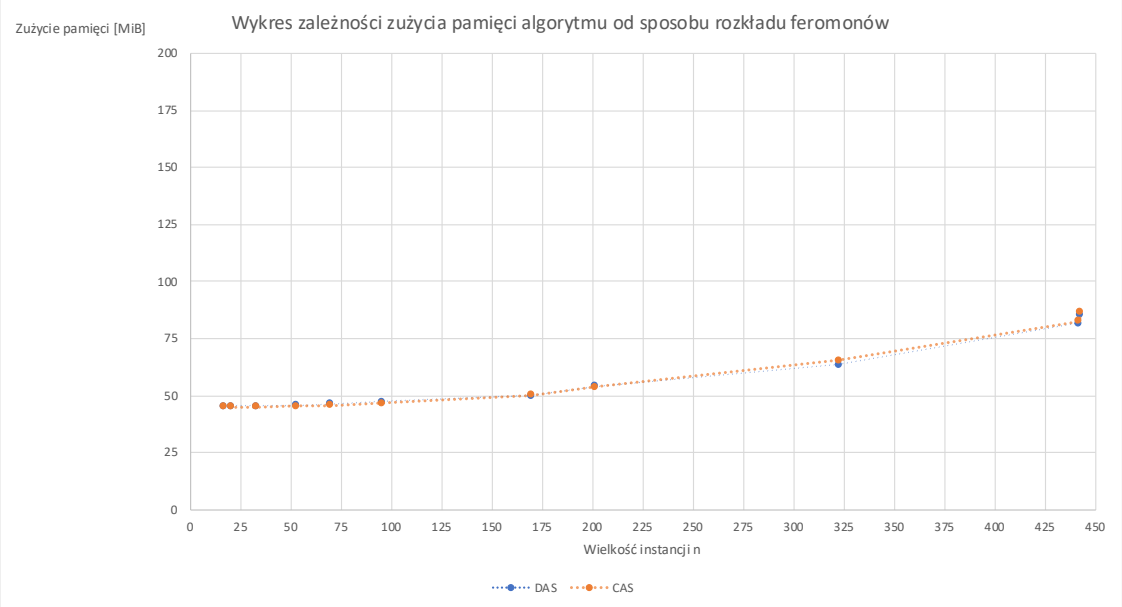
Wyniki wyboru różnych schematów rozkładu feromonów

	DAS			CAS		
wielkość instancji	średni czas [s]	średnie zużycie pami	średnia jakość rozwiązania [%]	średni czas [s]	średnie zużycie pamię	średnia jakość rozwiązania [%]
17	0,2742	45,29	3,19	0,2681	44,96	2,65
21	0,3602	45,32	8,18	0,3540	44,98	6,94
33	0,8380	45,51	21,30	0,8205	45,09	21,80
53	2,2071	45,78	37,61	2,1601	45,42	37,33
70	4,4448	46,19	43,37	4,3808	45,74	42,48
96	9,5139	47,24	35,60	9,4012	46,65	35,74
170	43,9358	50,00	72,50	43,6608	50,33	74,24
202	69,3194	54,09	48,73	68,8478	53,69	48,29
323	263,0960	63,52	39,79	262,3623	65,35	40,60
442	657,9583	81,52	77,50	657,8560	82,68	78,24
443	663,2612	85,33	25,78	663,7630	86,40	25,62

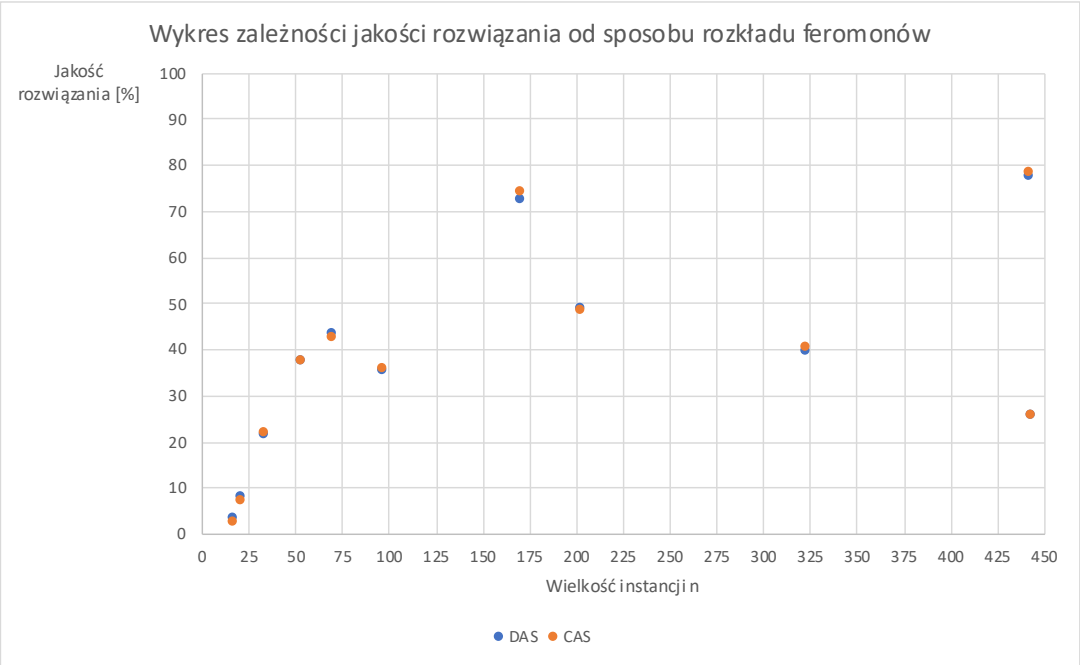
Tabela 1 Wyniki badanego czasu, zużycia pamięci i średniej wysokości błędu dla schematów rozkładu feromonów CAS i DAS



Wykres 1 Wykres czasu wykonania algorytmu dla schematów rozkładu feromonów CAS i DAS



Wykres 2 Wykres złożoności czasowej algorytmu dla schematów rozkładu feromonów CAS i DAS

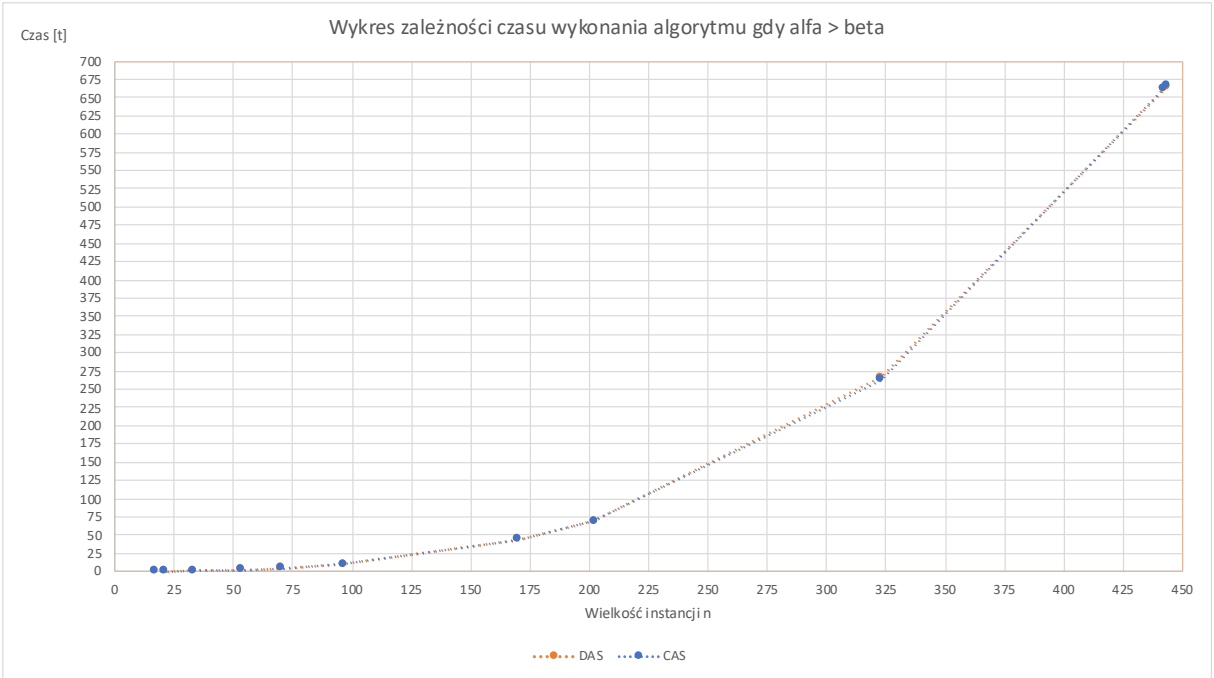


Wykres 3 Wykres średniej wielkości błędu dla schematów rozkładu feromonów CAS i DAS

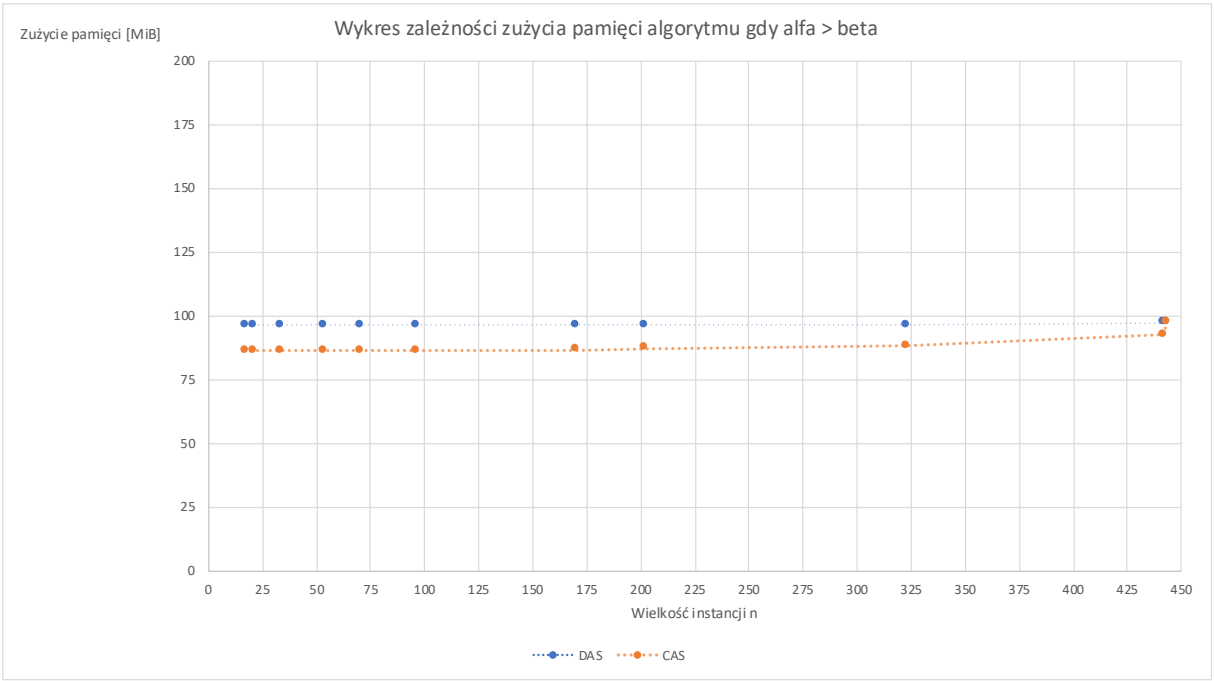
Wyniki dla konfiguracji parametrów alfa > beta

Tabela 2 Wyniki badanego czasu, zużycia pamięci i średniej wysokości błędu dla konfiguracji parametrów alfa > beta

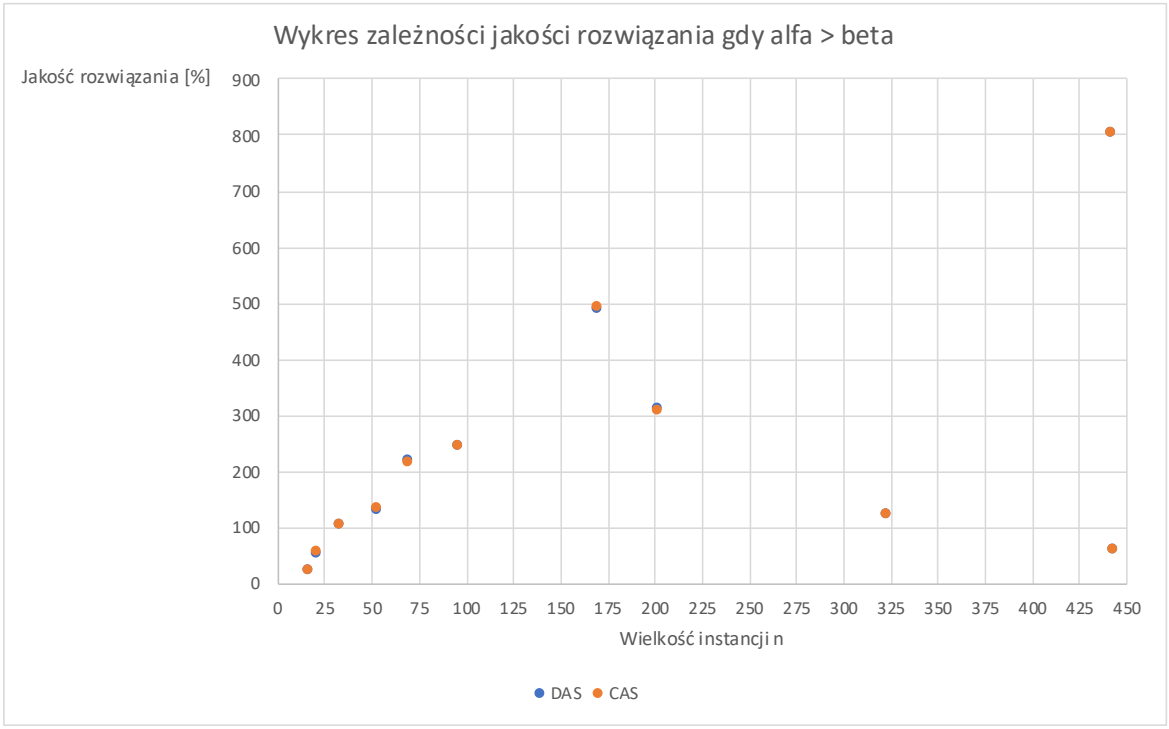
	DAS			CAS		
wielkość instancji	średni czas [s]	średnie zużycie pamięci [MiB]	średnia jakość rozwiązania [%]	średni czas [s]	średnie zużycie pamięci [MiB]	średnia jakość rozwiązania [%]
17	0,2765	96,52	22,76	0,2754	86,48	22,63
21	0,3649	96,52	52,84	0,3612	86,48	54,60
33	0,8461	96,52	102,70	0,8355	86,48	103,17
53	2,2104	96,52	130,60	2,1861	86,48	132,58
70	4,4432	96,52	216,99	4,4039	86,53	215,54
96	9,4891	96,52	242,12	9,4105	86,58	242,17
170	43,9870	96,52	489,18	43,7159	86,83	492,76
202	69,4380	96,52	308,94	69,0390	87,35	307,35
323	266,0681	96,52	122,09	263,0077	88,38	122,16
442	661,0174	97,53	800,05	662,2007	92,56	801,57
443	664,9928	97,53	59,62	666,2360	97,52	60,17



Wykres 4 Wykres czasu wykonania algorytmu dla konfiguracji parametrów alfa > beta



Wykres 5 Wykres złożoności czasowej algorytmu dla konfiguracji parametrów alfa > beta

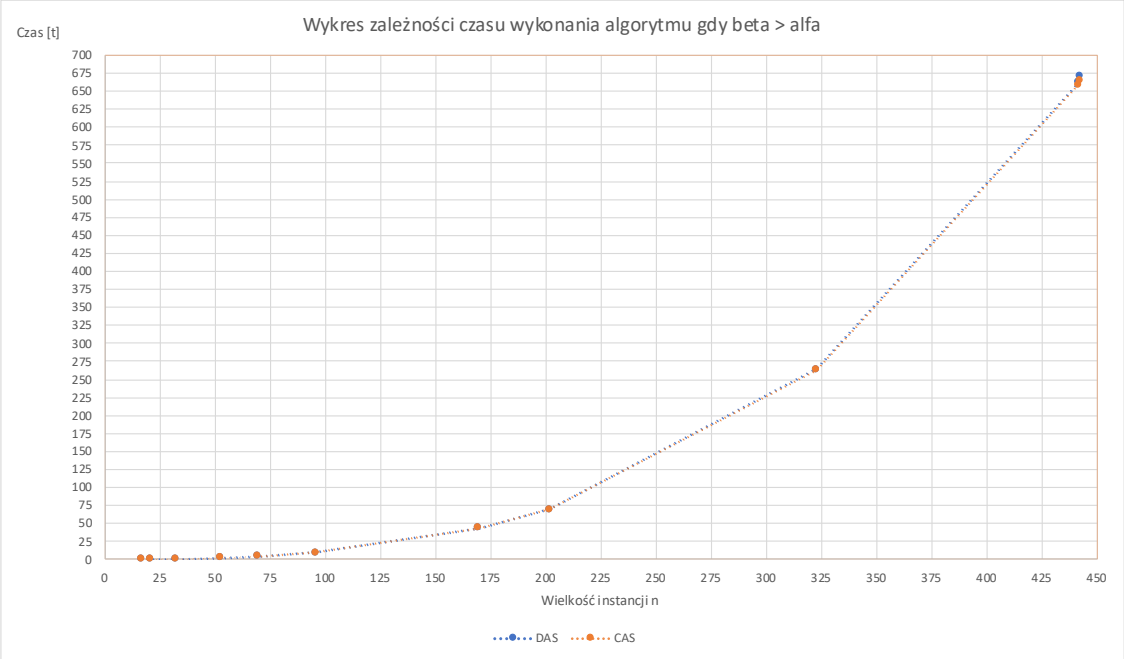


Wykres 6 Wykres średniej wielkości błędu dla konfiguracji parametrów alfa > beta

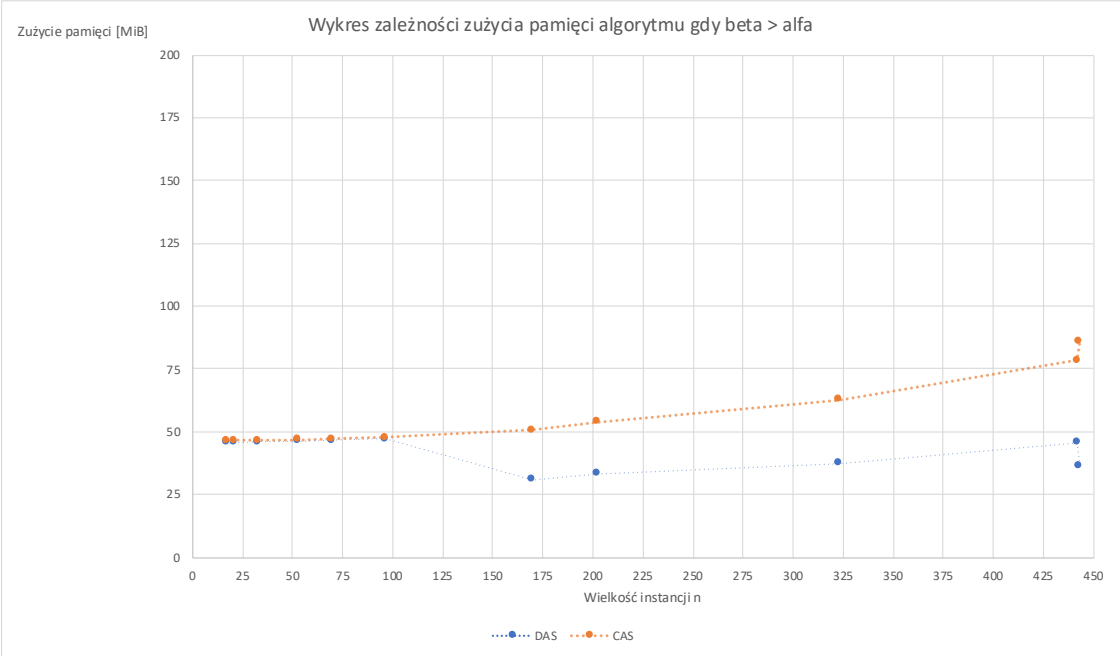
Wyniki dla konfiguracji parametrów beta > alfa

Tabela 3 Wyniki badanego czasu, zużycia pamięci i średniej wysokości błędu dla konfiguracji parametrów beta > alfa

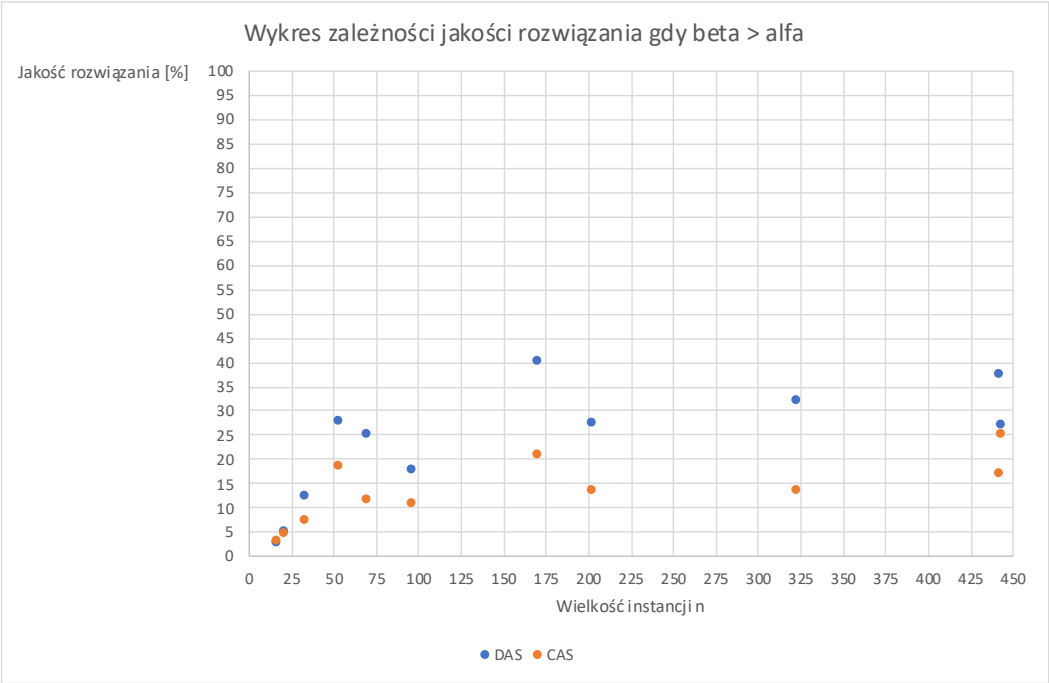
	DAS			CAS		
wielkość instancji	średni czas [s]	średnie zużycie pamięci [MiB]	średnia jakość rozwiązania [%]	średni czas [s]	średnie zużycie pamięci [MiB]	średnia jakość rozwiązania [%]
17	0,2812	45,70	2,62	0,2705	46,38	3,30
21	0,3580	45,72	5,22	0,3545	46,39	4,87
33	0,8462	45,86	12,51	0,8289	46,47	7,60
53	2,2257	46,11	27,89	2,1843	46,72	18,71
70	4,4869	46,49	25,32	4,4180	46,99	11,80
96	9,5372	47,23	17,68	9,4754	47,72	10,75
170	44,1362	30,97	40,17	43,8770	50,56	20,80
202	69,6419	33,29	27,44	69,2242	53,92	13,65
323	264,1938	37,35	32,18	263,3890	62,71	13,70
442	661,9700	45,71	37,53	658,7151	78,26	17,14
443	670,6586	36,60	27,22	664,6028	85,68	25,02



Wykres 7 Wykres czasu wykonania algorytmu dla konfiguracji parametrów beta > alfa

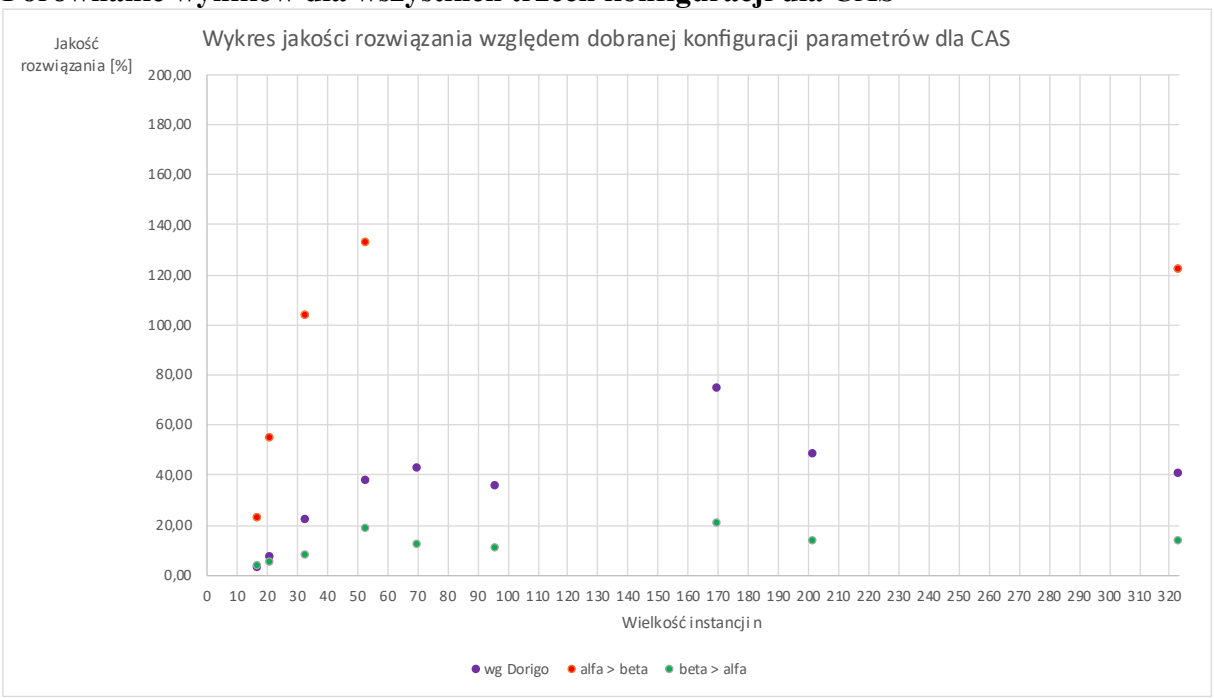


Wykres 8 Wykres zużycia pamięci dla konfiguracji parametrów beta > alfa



Wykres 9 Wykres średniej wielkości błędu dla konfiguracji parametrów beta > alfa

Porównanie wyników dla wszystkich trzech konfiguracji dla CAS



Wykres 10 Wykres średniej wielkości błędu dla wszystkich konfiguracji

7. Analiza wyników i wnioski

Analiza wyników

Analiza czasu wykonywania algorytmu

Tak jak można odczytać z Wykresów 1, 4 i 7 czas wykonywania algorytmu dla poszczególnych instancji problemu dla wszystkich trzech konfiguracji jest bardzo zbliżony. Różnice czasowe między schematami rozkładu feromonów CAS i DAS rzadko przekraczają 5 sekund.

Analiza zużycia pamięci

Z Wykresu 2 można odczytać, że zużycie pamięci rośnie im większy rozmiar instancji problemu. Na Wykresie 5 zużycie pamięci jest stałe. Na Wykresie 8 maleje przy większych instancjach.

Analiza wielkości błędu

Z Wykresów 3 i 6 można odczytać, że jakość rozwiązania jest zbliżona dla obu schematów rozkładu feromonów. Widoczne różnice są na Wykresie 9, gdzie przy parametrach alfa = 1 i beta = 5 algorytm CAS zwraca lepsze wyniki od DAS.

Wnioski

Dla metody algorytmu mrówkowego czas rozwiązywania jest zależny od wielkości instancji oraz ilości pokoleń (iteracji). Im więcej pokoleń tym wynik będzie dokładniejszy, lecz czas wykonywania algorytmu będzie większy.

Zużycie pamięci jest średnio między 30 a 100 MiB. Dla większych instancji powinno być większe ze względu na tworzeniu tablic o większym rozmiarze (odpowiedzialnych za poziom feromonów na poszczególnych trasach i za heurystykę wyboru lokalnego). W niektórych konfiguracjach parametrów pojawiają się anomalie, gdzie dla każdej badanej instancji zużycie pamięci jest stale wielkie (około 90MiB) lub stale małe (około 30MiB). Przebieg programu był testowany za pomocą tego samego narzędzia co poprzednie metody i w takich samych warunkach (na laptopie był uruchomiony tylko program wykonujący algorytm). Mimo tych dziwnych wyników, zużycie pamięci jest stosunkowo małe jak na dzisiejsze standardy technologiczne komputerów domowych.

Jakość rozwiązania jest najlepsza (najniższy średni błąd) dla konfiguracji parametrów alfa = 1 i beta = 5, a najgorsza (najwyższy średni błąd) dla konfiguracji alfa = 5 i beta = 1. Wynika to z tego, że gdy parametr alfa ma większe znaczenie to prowadzi do wchodzenia w ekstrema lokalne, które nie gwarantują osiągnięcia ekstremum globalnego.

8. Źródła

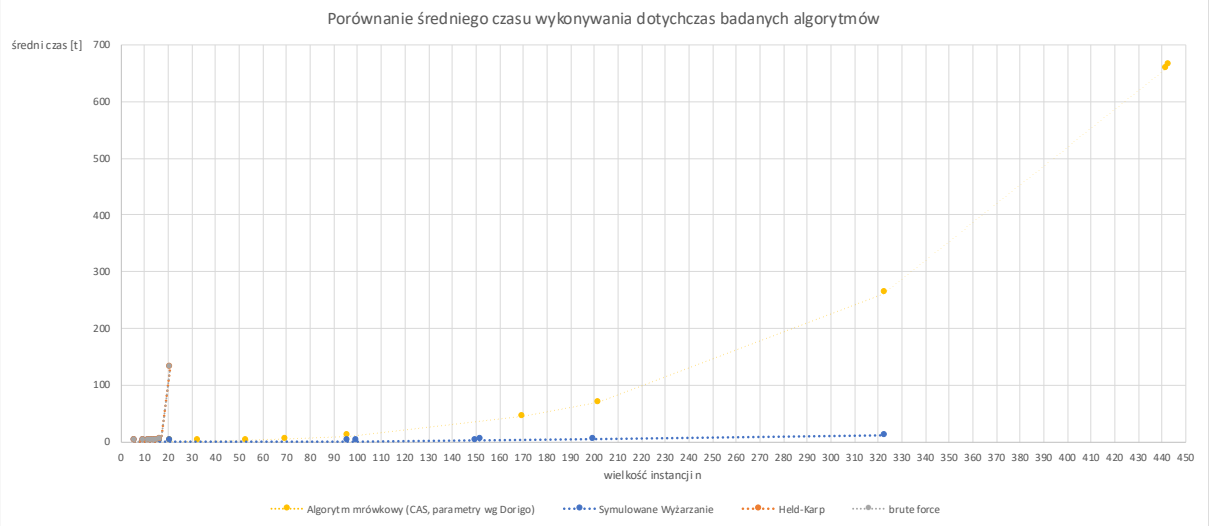
[1] J. Dreo, A. Petrowski, P. Siarry, E. Taillard – Metaheuristics for Hard Optimization
[2] https://docs.python.org/3/library/time.html
[3] https://pypi.org/project/memory-profiler/
[4] Sean Luke – Essentials of Metaheuristics
[5] El Ghazali Talbi – Metaheurstics. From design to implementation
[6] https://pandas.pydata.org

Dodatek A

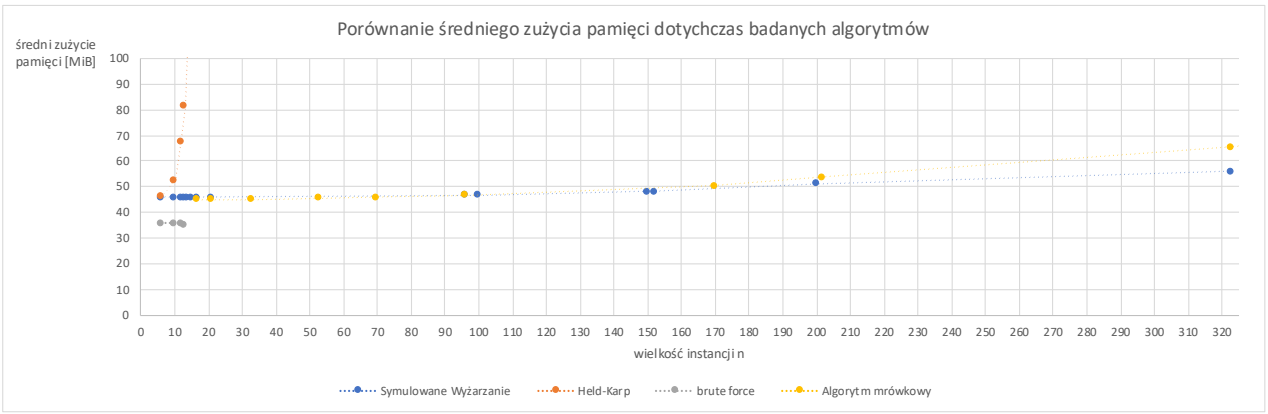
Porównanie algorytmów brute force, Helda-Karpa, symulowanego wyżarzania oraz algorytmu mrówkowego

	ALGORYTM HELDA-KARPA		ALGORYTM BRUTE FORCE		Algorytm oparty o SA		Algorytm mrówkowy (CAS, parametry wg Dorigo)	
n	średni czas [s]	średnie zużycie pamięci [MiB]	średni czas [s]	średnie zużycie pamięci [MiB]	średni czas [s]	średnie zużycie pamięci [MiB]	średni czas [s]	średnie zużycie pamięci [MiB]
6	0,1044	46,07	0,0001	35,61	0,1148	45,73		
10	0,1263	52,34	0,0001	35,62	0,1530	45,82		
12	0,1642	67,67	0,3232	35,37	0,1541	45,84		
13	0,2273	81,46	40,3472	35,13	0,1567	45,86		
14	0,3664	111,43	516,3495		0,1597	45,88		
15	0,7400	193,95	7098,5659		0,1573	45,88		
17	3,5391	561,01			0,1619	45,88		
21	130,5848	2903,25			0,1719	45,88		
96					0,9803	46,55		
100					0,8415	46,71		
150					2,1272	48,03		
152					2,5040	48,20		
200					3,8666	51,01		
323					10,3837	55,95		
17							0,2681	44,96
21							0,3540	44,98
33							0,8205	45,09
53							2,1601	45,42
70							4,3808	45,74
96							9,4012	46,65
170							43,6608	50,33
202							68,8478	53,69
323							262,3623	65,35
442							657,8560	82,68
443							663,7630	86,40

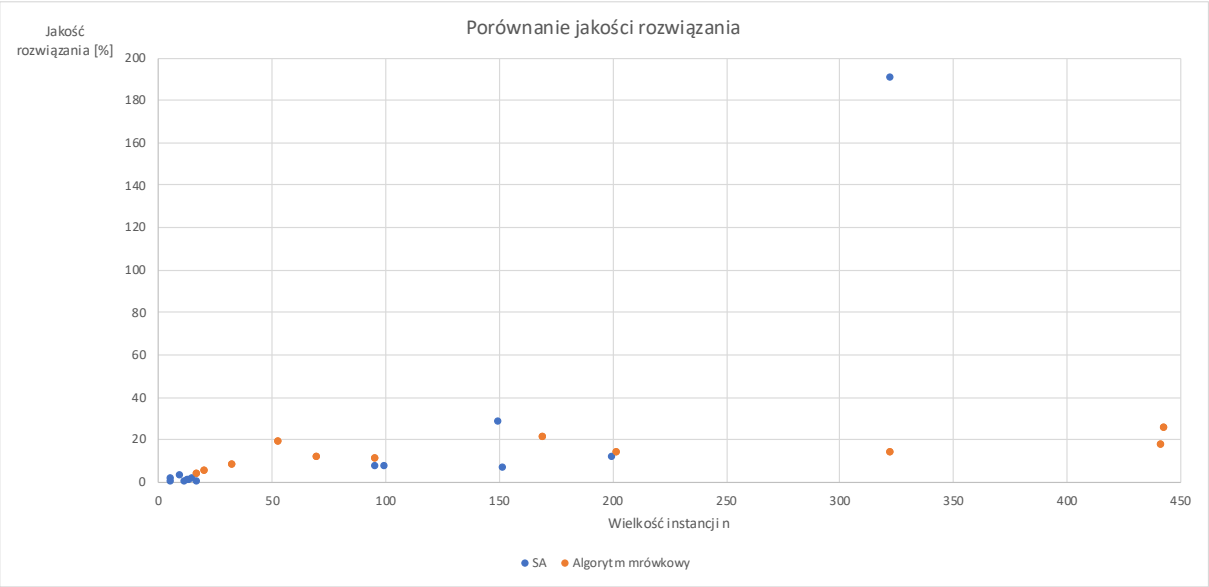
Tabela A.1 Porównanie średnich wyników dotychczas badanych algorytmów



Wykres A.1 Porównanie czasu wykonania dotychczas badanych algorytmów



Wykres A.2 Porównanie zużycia pamięci wykonania dotychczas badanych algorytmów



Wykres A.3 Porównanie jakości rozwiązania dla SA i ACO

Analiza i wnioski

Z Wykresu A.1 można odczytać, że algorytm oparty o metodę algorytmu mrówkowego wykonuje się o wiele dłużej niż algorytm oparty o metodę symulowanego wyżarzania.
Z Wykresu A.2 można odczytać, że algorytm oparty o metodę algorytmu mrówkowego przy instancjach n >= 200 zużywa więcej pamięci, jednak nie jest to duża różnica.
Z Wykresu A.2 można odczytać, że algorytm oparty o metodę algorytmu mrówkowego osiąga zbliżone lub gorsze rozwiązania od SA dla instancji n <= 200, lecz dla instancji n > 200 osiąga błędy mniejsze od 25%.
Przy metodach metaheurystycznych SA i ACO ważny jest dobór parametrów. Wybór odpowiednich parametrów w ACO wydaje się bardziej przewidywalny oraz nie zależy od złożoności i wielkości instancji. W SA dobór odpowiednich parametrów zależy od złożoności problemu (grafu) i wymaga testowania różnych konfiguracji w celu znalezienia rozwiązania zbliżonego do optimum. Metoda SA gorzej radzi sobie z instancjami powyżej n > 300.

Algorytm oparty o metodę symulowanego wyżarzania ma złożoność pamięciową stałą, niezależną od wielkości instancji i pozwala wygenerować wyniki dla instancji n < 20 tak samo szybko jak algorytm Helda-Karpa oraz metoda siłowa. Metoda SA generują wyniki dla instancji n > 20 w kilka sekund co jest nieosiągalne dla pozostałych dotychczas badanych algorytmów. Metoda algorytmu mrówkowego osiąga niskie czasy dla n < 100, lecz później czas znacznie wzrasta. Należy pamiętać, że metoda SA i ACO są metaheurystykami i zwracają wyniki dążące do globalnego optimum, a nie dokładny, optymalny wynik.
W przypadku, gdy poszukuje się optimum dla dużych instancji (n > 20) lub najważniejsza jest szybkość generowania i wynik może mieć niewielki (do 30% przy dobrze dobranych parametrach) błąd względem wyniku optymalnego należy wybrać algorytm oparty o SA.
Dla instancji n < 20 i przy dużych zasobach pamięciowych algorytm Helda-Karpa będzie najlepszym wyborem, bo w krótkim czasie zwraca optymalny (dokładny) wynik. Przy ograniczonych zasobach pamięciowych i instancjach n < 14 pozostaje wybór metody siłowej.