Projektowanie Efektywnych Algorytmow
Projekt
22/12/2020

248820 Przemysław Rychter

(4) Ant Colony Optimization

1. Sformułowanie zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywności algorytmu opartego o metode kolonizacji mrówkowej w problemie Komiwojażera. Problem komiwojażera (eng. *Travelling salesman problem, TSP)* to zagadnienie polegające na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Algorytm mrówkowy posiada kilka parametrów, ich dobór ma kluczowe znaczenie, dla działania algorytmu.

Wielkość błędu bezwzględnego, w zależności od wielkości instancji, nie może przekraczać:
- dla $n < 25$, 0%,
- dla $24 < n < 350$, 50%,
- dla $75 < n < 2500$, 150%.

2. Metoda

W 1989 roku Deneubourg odkrył i wykazał, że mrówki potrafią odszukać najkrótszą drogę do pożywienia. Posiadają tzw. inteligencję zbiorową (stadną), która oparta jest na wykorzystaniu wspomnianego systemu komunikacji. Inteligencja stadna jest techniką sztucznej inteligencji bazująca na studiach nad kolektywnym wzorcem zachowań indywiduów w samoorganizujących się systemach. W celu znalezienia najkrótszej drogi do pożywienia mrówki komunikują się ze sobą zostawiając feromony w otoczeniu. Ta forma komunikacji nazywana jest stygmergią. Pojedyncza mrówka porusza się losowo, lecz gdy napotka na swej drodze ślad feromonowy, jest to wielce prawdopodobne, że mrówka będzie podążać za tym śladem. Mrówka furażując, pozostawia ślad feromonowy na swej drodze. Gdy znajdzie źródło pożywienia, wraca do mrowiska wzmacniając drogę powrotu śladem feromonowym. Z czasem ślad feromonowy zanika.

Metoda Optymalizacyjna zaproponowana przez Marco Dorigo na początku lat 90

- Każda wirtualna mrówka jesta probabilistycznym mechanizmem, który konstruuje rozwiązanie problemu wykorzystując:
  - Pozostawianie sztucznego feromonu
  - Informację Heurystyczną: ślady feromonowe, pamięć odwiedzonych miast i widoczność

Implementacja algorytmu mrówkowego dla problemu komiwojażera wygląda następująco:

- Algorytm uruchamia określoną ilość iteracji(może one bazować na danym warunku zatrzymania)
- Podczas każdej iteracji określona ilość mrówek szuka cyklu Hamiltona, podejmując probabilistyczną decyzję podczas wyboru następnego wierzchołka tworzącego cykl
- Po każdej iteracji ilość feromonów jest uaktualniana w określony sposób

Mrówki mogą być rozmieszczane losowo, lub każda w innym wierzchołku, w implementacji liczba wierzchołków instancji to liczba mrówek.

Początkowa ilość feromonów na każdej krawędzi wynosi:

$$\tau_0 = \frac{m}{C^{nn}} \qquad (1)$$

m - liczba mrówek

C^{nn} - szacowana długość trasy (średnia z 10000 próbek losowo znalezionych tras)

Każda mrówka podczas konstruowania trasy wybór następnego wierzchołka podejmuje w oparciu o prawdopodobieństwo wyliczone na podstawie wcześniej podanej informacji heurystycznej:

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

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

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: Slajd z wykładu opisujący prawdopodobieństwo wyboru następnego wierzchołka

Na powyższym rysunku jest błąd w mianowniku zamiast τ drugi raz powinno być η

W implementacji przyjęto:

$\alpha = 1$

$\beta = 3$

τ_{ij} - ilość feromonów między wierzchołkami i oraz j

Istnieją trzy typy algorytmu: gęstościowy, ilościowy oraz cykliczny. W implementacji zastosowałem cykliczny, w którym ilość feromonów uaktualniamy po wykonaniu iteracji czyli znalezieniu m tras przez m mrówek. Każda mrówka posiada tą samą ilość feromonu która po podzieleniu przez długość znalezionej trasy, dodawana jest do każdej krawędzi przez którą mrówka szła.

W CAS stała ilość feromonu Q_{Cycl} dzielona jest przez długość trasy L znalezionej przez k-tą mrówkę - L^k.

$$\Delta\tau_{ij}^{k}(t, t+1) = \begin{cases} \frac{Q_{Cycl}}{L^k} & \text{jeżeli } k\text{-ta mrówka przeszła z } i \text{ do } j \text{ na swojej trasie} \\ 0 & \text{w przeciwnym przypadku} \end{cases}$$

Rysunek 2: Sposób dodawania feromonów w algorytmie typu CAS

W implementacji zostało przyjęte $Q_{Cycl} = 100$

Po wykonaniu iteracji ilość feromonu na każdej krawędzi jest aktualizowana:

Wartość feromonu aktualizowana jest po n krokach wg wzoru:

$$\tau_{ij}(t+n) = \rho_1 \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t, t+n),$$

gdzie

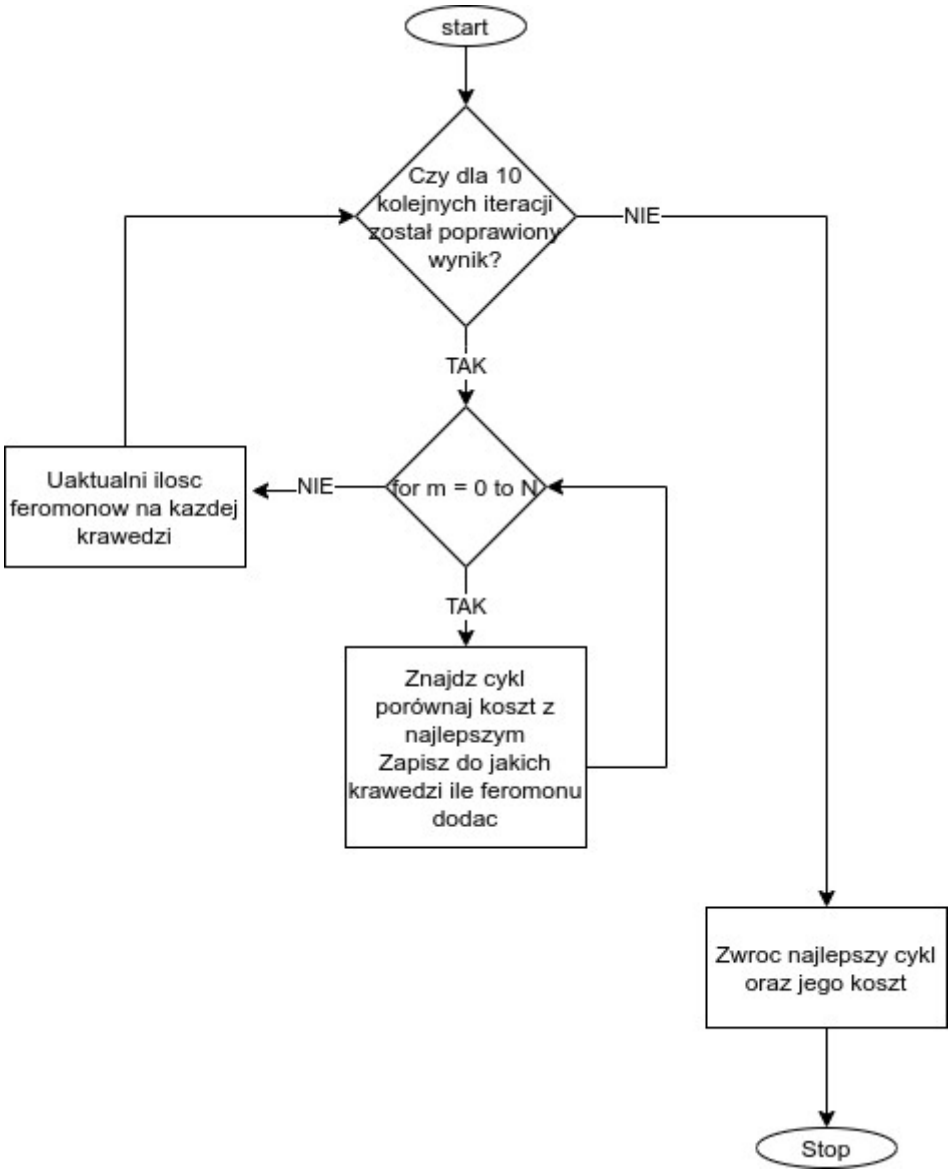
$$\Delta\tau(t, t+n) = \sum_{m}^{k=1} \Delta\tau_{ij}^{k}(t, t+n)$$

Rysunek 3: Sposób aktualizowania feromonu

Na powyższym zdjęciu czynnik Δ to po prostu sumą feromonu które zostawiły wszystkie mrówki na danej krawędzi, natomiast ρ_1 to liczba informująca ile % feromonów wyparowało. W implementacji przyjęte zostało: $\rho_1 = 0.5$

W implementacji ilość iteracji wyznaczona jest warunkiem zatrzymania który zakończy algorytm jeżeli w 100 kolejnych iteracjach najlepszy wynik nie zostanie poprawiony.

3. Algorytm



Rysunek 4: Schemat blokowy algorytmu ACO

Na powyższym rysunku przedstawiony jest konceptualny schemat blokowy algorytmu, na podstawie którego go napisałem, poza materiałami z wykładu inspirowałem się tutorialem na youtube: https://www.youtube.com/watch?v=783ZtAF4j5g&ab_channel=AliMirjalili stroną http://155.158.112.25/~algorytmyewolucyjne/materialy/inteligencja_stadna.pdf?fbclid=IwAR3sSYjw7eWPe_GqagjLeEqfOam0jI0n2Kv5Qul3b9QKGCmDI0V2W4kFTio oraz załączonym plikiem pdf.

4. Dane testowe

Do sprawdzenia poprawności działania algorytmu wybrano następujący zestaw instancji:

- tsp_6_1.txt
- tsp_10.txt
- tsp_12.txt
- tsp_13.txt
- tsp_14.txt
- tsp_15.txt

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

Algorytm dla powyższych instancji, zwrócił scieżki o koszcie optymalnym

http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/pea.php#instances

Do wykonania ostatecznych badań wybrano następujący zestaw instancji:

- Instancje symetryczne
  - gr17.tsp
  - gr21.tsp
  - bayg29.tsp 2
  - dantzig42.tsp
  - berlin52.tsp
  - brazil58.tsp
  - st70.tsp
  - eil76.tsp
  - gr96.tsp
  - kroB100.tsp
  - pr107.tsp
  - gr120.tsp
  - bier127.tsp
  - pr136.tsp
  - pr144.tsp
  - pr152.tsp
  - brg180.tsp
  - rat195.tsp
  - gr202.tsp
  - gr229.tsp
  - gil262.tsp
  - a280.tsp
  - pr299.tsp
  - linhp318.tsp
  - rd400.tsp
  - fl417.tsp
- Instancje asymetryczne
  - m9.atsp
  - br17.atsp
  - ftv33.atsp
  - ftv38.atsp
  - p43.atsp
  - ft53.atsp
  - ftv64.atsp
  - ftv70.atsp
  - ftv170.atsp
  - rbg323.atsp
  - rbg358.atsp
  - rbg403.atsp
  - rbg443.atsp

http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/pea.php#instances

5. Procedura badawcza

Zbadałem czas rozwiązania problemu oraz bładu względem rozwiązania optymalnego w zależności od wielkości instancji.

Procedura badawcza polegała na uruchomieniu programu sterowanego plikiem inicjującym .ini format pliku:

nazwa_instancji liczba_wykonań rozwiązanie_optymalne
...
nazwa_instancji liczba_wykonań rozwiązanie_optymalne
nazwa_pliku_wyjściowego

Instancje testowe pochodziły ze stron:

- http://elib.zib.de/pub/mp-testdata/tsp/tsplib/atsp/index.html (**ATSP**)
- http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html (**TSP)**
- http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/

Instancje z dwóch pierwszych powyższych adresów zostały pobrane ze strony:

- http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/pea.php

Poniżej przykładowa treść plik „conf.ini" (dla badań mających na celu dostrojenie algorytmu).

```
m9.atsp 10 215
br17.atsp 10 39
ftv33.atsp 10 1286
ftv38.atsp 10 1530
p43.atsp 10 5620
ft53.atsp 10 6905
ftv64.atsp 10 1839
ftv70.atsp 10 1950
gr17.tsp 10 2085
gr21.tsp 10 2707
bayg29.tsp 20 1610
dantzig42.tsp 15 699
berlin52.tsp 10 7542
brazil58.tsp 10 25395
st70.tsp 10 675
eil76.tsp 10 538
Opis_jak_dostrojone.csv
```

Każda instancji rozwiązywana była zgodnie z liczbą jej wykonań, np. ftv64.atsp wykonana została 10 razy. Do pliku wyjściowego Opis_jak_dostrojone.csv zapisywane były informacje o instancji: jej nazwa, liczba wykonań algorytmu oraz ilość wierzchołków. Następnie zapisywane były czasy wykonań algorytmu dla tej instancji. Plik wyjściowy zapisywany był w formacie csv. Poniżej przedstawiono fragment zawartości przykładowego pliku wyjściowego.

```
ft53.atsp Reps: 10 Nodes:  53
6604922
5205713
6375854
6400286
7179492
5108533
6247084
5229969
7029841
6305316
ftv64.atsp Reps: 10 Nodes:  65
```

Na standardowe wyjście dla każdego powtórzenia algorytmu zwracana była wartość błędu, znalezionego rozwiązania względem optymalnego podanego w pliku „conf.ini" oraz średnia wartość tego błedu dla każdej z instancji. Na koniec wyjścia zwracane było średnie wartości błedu

w kolenności odpowiadającej badanym instancją. Poniżej fragment pliku do którego stdandardowe wyjście było przekierowywane w celu zapisania danych o błędach i ich analizie.

```
_____________________________________________
eil76.tsp               nodes: 76
blad: 9.47955 %
blad: 9.29368 %
blad: 9.8513 %
blad: 10.5948 %
blad: 9.8513 %
blad: 7.0632 %
blad: 14.4981 %
blad: 9.8513 %
blad: 11.1524 %
blad: 6.3197 %
sredni blad (w %) dla aktualnej instancji ponizej
9.79554
_____________________________________________
SREDNIE BLEDY w % DLA KOLEJNO WSZYSTKICH INSTACJI Z conf.ini
0
0.339858
17.5091
19.3692
21.3436
5.45652
11.9014
9.79554
```

Pomiary zostały wykonane na platofrmie sprzętowej:

- procesor: Intel® Core™ i5-8250U CPU 1.60GHz × 8
- pamięć operacyjna: 31,2 GB
- system operacyjny: z rodziny Linux - Ubuntu 20.04.1 LTS 64-bit

Pomiary czasu zostały wykonane za pomocą biblioteki std::chrono [6].

```
Aco aco = Aco();

auto start = std::chrono::high_resolution_clock::now();

solution_cost = aco.calculate(vertices, distances);

auto stop = std::chrono::high_resolution_clock::now();
aco.~Aco();
```

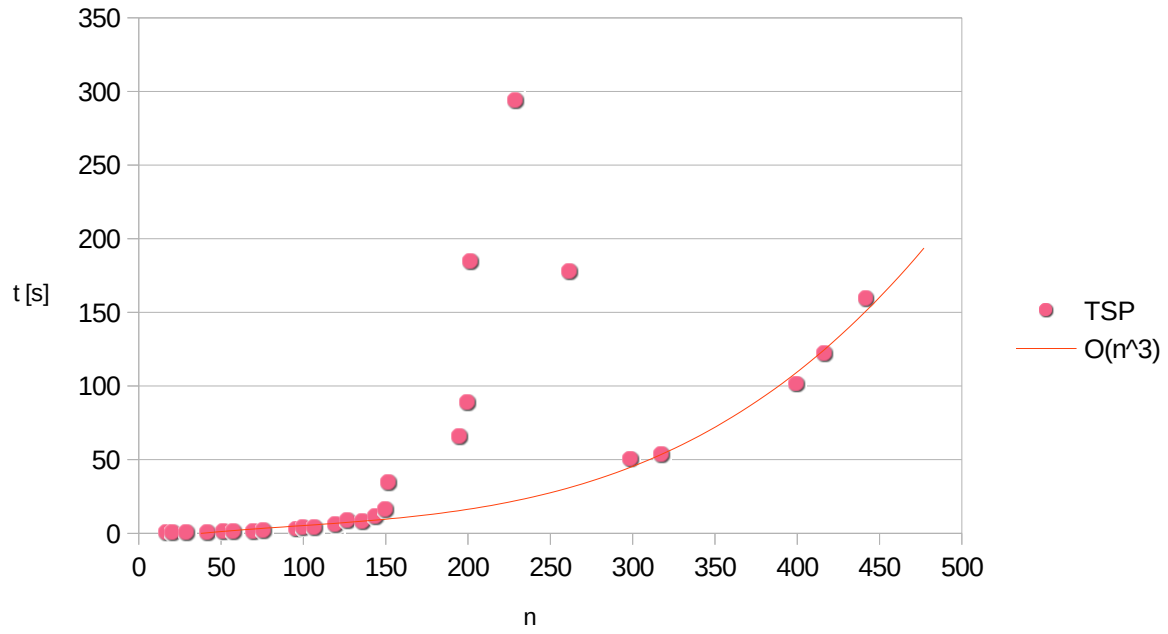
Rysunek 5: Fragment kodu przedstawiający sposób pomiaru czasu wykonania algorytmu

Wyniki zostały opracowane w programie LibreOffice Calc.

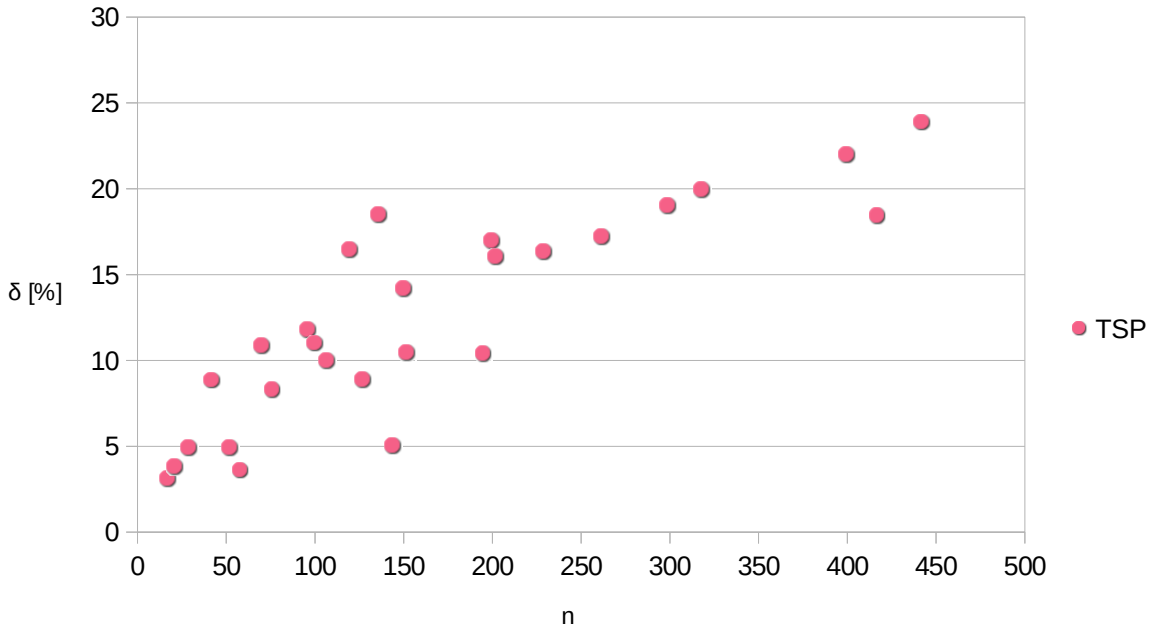
6. Wyniki

Wyniki zgromadzone zostały w plikach csv :

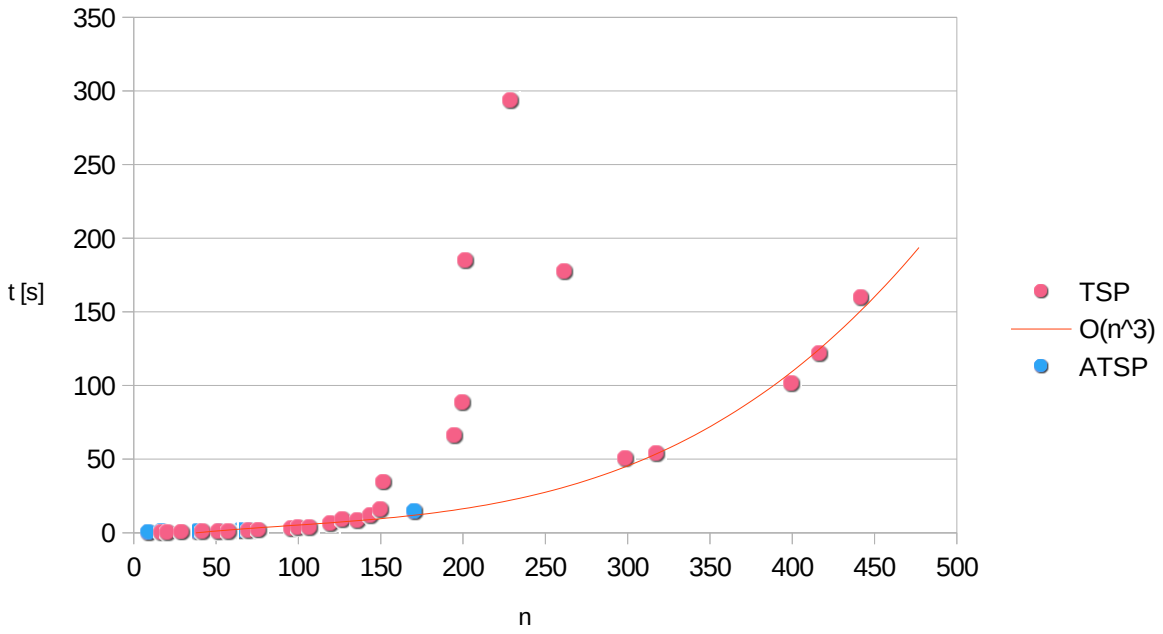
Pliki został dołączone do raportu i znajdują się na dysku Google pod adresem https://drive.google.com/drive/folders/1yHS-PS9DVc7rIv4o933_UdkAuBjO8zVU.



Rysunek 6: Wpływ wielkości instancji n na czas rozwiązania TSP algorytmem ACO



Rysunek 7: Wpływ wielkości instancji n na błąd względny δ uzyskanego rozwiązania algorytmem ACO



Rysunek 8: Wpływ wielkości instancji n na czas rozwiązania TSP i ATSP algorytmem ACO

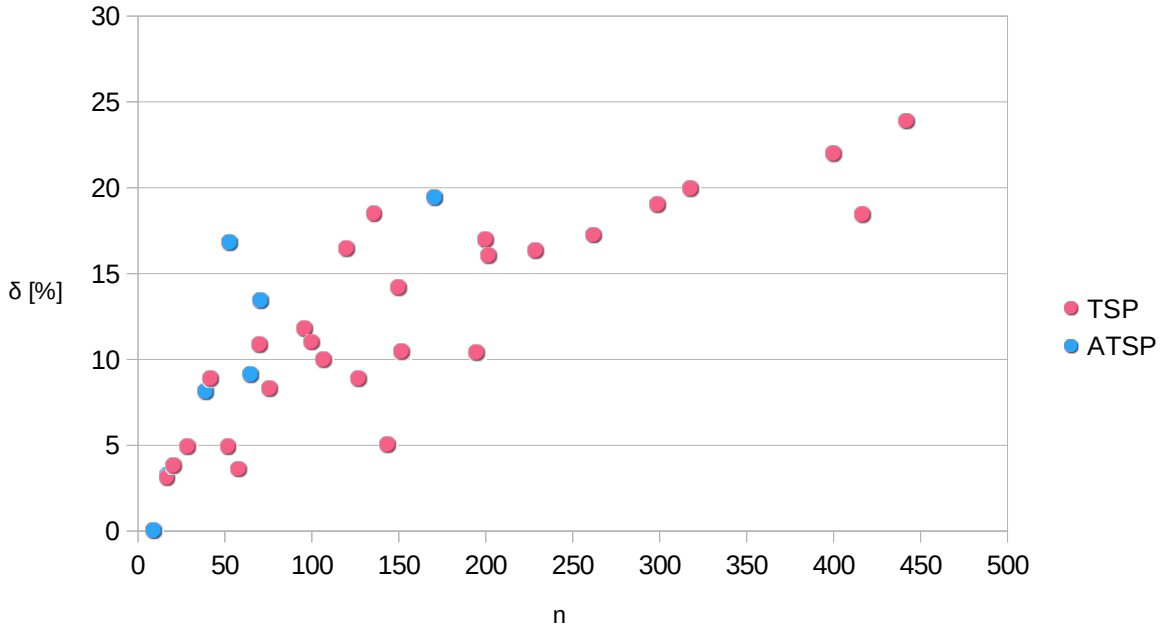
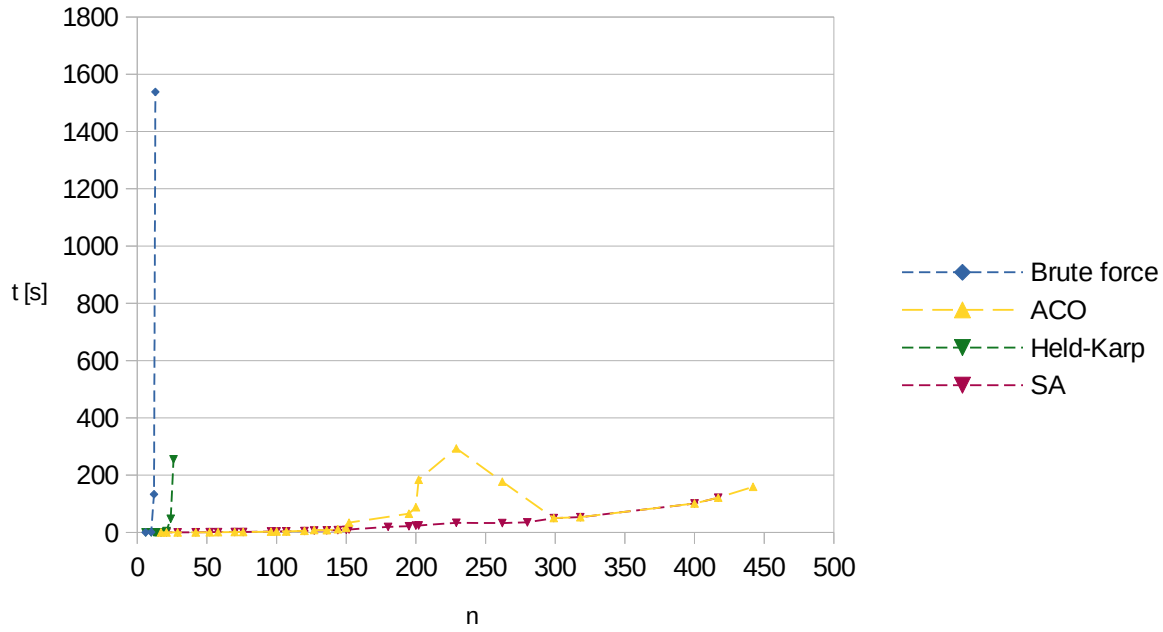


Figure 9: Wpływ wielkości instancji n na bład względny δ uzyskanego rozwiązania algorytmem ACO

6.1 Porównanie z innymi algorytmami



Rysunek 10: Porównianie czasu działania badanych algorytmów w zależności od wielkości instancji

7. Analiza wyników i wnioski

Udało się z powodzeniem zaimplementować algorytm ACO o złożoności $O(n^3)$ (pseudowielomianowej). Dla intancji symetrycznych do 450 wierzchołków udało się zwracać rozwiązania z błedem do 30% w stosunku do rozwiązania optymalnego.

Na wykresie zależności czasu wykonania algorytmu od wielkości instancji (rysunek 6) nałożono krzywą $O(n^3)$, ponieważ pasuje ona do pomiarów uznałem, że algorytm posiada taką właśnie złożoność czasową. Według literatury złożoność ACO wynosi $O(CC*n^3)$ gdzie CC to liczba iteracji, taką złożoność nazywamy pseudowielomianową ponieważ, spodziewamy się że CC powinno rosnąć wykładniczo. Warunek zatrzymania, który spowoduje zakończenie algorytmu, jeżeli dla 10 kolejnych iteracji nie zostanie poprawiony najlepszy wynik, może powodować dłuższe czasy obliczeń, instancji które nie pasują do krzywej n^3 . Możliwe, że mają one specyficzną budowę (np. wiele krawędzi jest podobnej długości, podczas gdy wszystkie inne instacje posiadają bardziej zróżnicowaną przestrzeń rozwiązań).

Jak widać na wykresie błedów w zależności od wielkości instancji (rysunek 7) wydaje się, że punkty na początku tworzą funkcję liniową. Do dokładniejszego stwierdzenia jak zachowuje się bład względny, należałoby przeprowadzić badania z instacjami o większych rozmiarach. Być może błedy względne tworzą na wykresie funkcję logarytmiczną, albo nawet nie ma wzrostu błedu, i pozostaje on na podobny poziomie wraz ze wzrostem wielkości instancji.

Algorytm nie okazał się być lepszy od SA (błedy do 6 %), zapewne z powodu braku dostrojenia i większej ilości badań tak jak zrobiłem to dla SA. Wartości parametrów ACO zostały ustalone na podstawie danych z wykładu, które zostały wyznaczone na podstawie doświadczeń Marco Dorigo.

Powyższy wykres (rysunek 10) porównujący wszystkie zaimplementowane przez mnie algorytmy pokazuje przewagę metod probabilistycznych i zastosowań heurystyk, jeżeli chodzi o czas wykonywania algorytmu. Algorytm ACO wykonuje się w podobnym czasie co SA jednakże, SA był dobrze dostrojony, czego nie uczyniłem dla ACO. Błedy dla ACO są znacznie większe (około 5 razy) tak więc z tych implementacji najlepszy okazał się być algorytm symulowanego wyżarzania. Trzeba pamiętać, że metody probabilistyczne przeważnie zwracają wynik z pewnym błędem, jednak poprzez dostrajanie parametrów może być zredukowany do małych wartości.

Ze względu na sposób obliczania prawdopodobieństwa wyboru następnego wierzchołka przez mrówkę, trzeba było odrzucić instancje które posiadały krawędzie o długości 0(aby nie dzielić przez 0), dlatego zostało wykonano tak mało testów instancji asymetrycznych.