left=3cm, bottom=3cm, top=3cm

# Flow-Shop Scheduling
# Laboratorium Optymalizacji Kombinatorycznej

17 stycznia 2014

Szymon Gramza
szymon.gramza@student.put.poznan.pl

Maciej Sobkowski
maciej.sobkowski@student.put.poznan.pl

Prowadzący: Marcin Radom

## 1 Wprowadzenie

### 1.1 Opis problemu

Problem Flow-Shop jest problemem optymalizacji kobminatorycznej, który polega na szeregowaniu zadań przy wykorzystaniu ustalonej liczby maszyn. Każde zadanie składa się z listy operacji, które muszą zostać wykonane w określonej kolejności na określonej maszynie, a każdą operację cechuje czas wykonywania. Zadania pojawiają się w różnych momentach pracy systemu, a maszyny na których są wykonywane posiadają wymuszone przestoje. Do rozwiązania powyższego problemu wykorzystaliśmy zainspirowaną zachowaniem się mrówek poszukujących pożywienia metaheurystykę ACO. Algorytm mrówkowy będziemy porównywać z rozwiązaniem uzyskanym za pomocą algorytmów losowego oraz rozwiązaniem optymalnym obliczonym za pomocą wzoru:

$$opt = \frac{\text{suma czasów wszystkich operacji}}{\text{liczba maszyn}} \tag{1}$$

### 1.2 Program testujący

#### 1.2.1 Opis

Aplikacja została napisana w języku C i wykorzystuje biblioteki dostępne w systemach z rodziny GNU/Linux.

### 1.2.2 Kompilacja

Kompilacja została zautomatyzowana przy pomocy programu GNU Make. Aby skompilować program należy przejść do katalogu *scheduler* i z poziomu konsoli wydać polecenie *make*. Prawidłowe zakończenie procesu kompilacji zostanie zasygnalizowane komunikatem *Zakończono linkowanie!*. W ten sam sposób należy przebiega kompilacja programu *generator*.

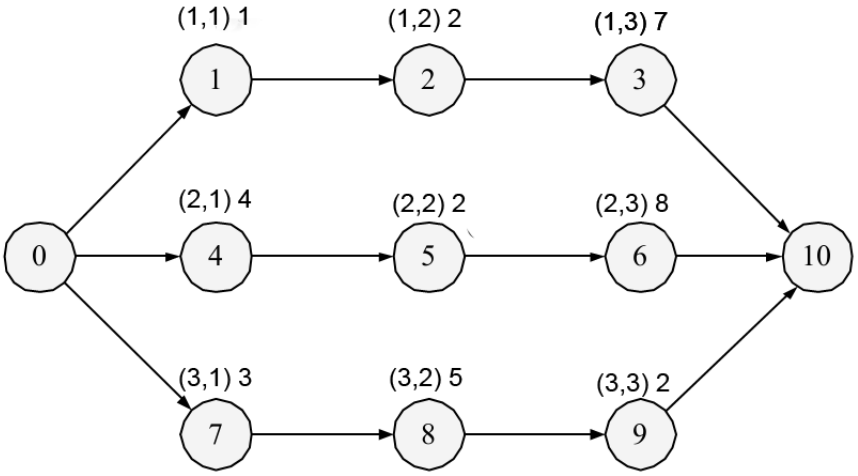
### 1.2.3 Uruchomienie

Praca aplikacji jest sterowana przez przełączniki powszechnie stosowane w oprogramowaniu linuksowym. Przykładowe wywołanie *scheduler -i inst_name -o file_name.csv* uruchomi program dla instancji zapisanej w pliku inst_name i wykona algorytm ACO oraz losowy. Wynik szeregowania zostanie zapisany do pliku o nazwie file_name.csv. Dodatkowo, aby uruchomić program generujący instancje należy użyć polecenia *./generator*.
Przykładowe wywołanie: *./generator -i input_name -o inst_name* spowoduje wygenerowanie pliku instancji inst_name przyjmując parametry zawarte w pliku input_name zgodnie ze schematem, że każda linia składa się z 5 liczb całkowitych, które odpowiednio charakteryzują: liczbę zadań, początek i koniec przedziału czasu wykonania operacji, początek i koniec przedziału czasu gotowości zadania.
Generator losuje czasy wykonania operacji i czasy gotowości dla zdefiniowanej liczby zadań.

# 2 Algorytm Ant Colony Optimization

## 2.1 Opis algorytmu

Zastosowana przez nas metaheurystyka - Ant Colony Optimization - opiera się na naturalnym zachowaniu kolonii mrówek poszukujących pożywienia. Algorytm korzysta ze struktur utrzymujących informację o drodze od źródła do celu, przebytej przez każdą mrówkę danego pokolenia. Algorytm został zaprojektowany do problemów znajdowania ścieżki w grafie o najmniejszym koszcie (problem TSP). Przystosowanie algorytmu do problemu szeregowania zadań sprowadza się w naszym przypadku do przedstawienia operacji danej instancji w postaci grafu, którego przykład umieszczono poniżej:



Każdemu wierzchołkowi grafu przyporządkowana jest jedna operacja, każda operacja zaś posiada informacje o maszynie i numerze zadania. Zależności pomiędzy kolejnością wykonania operacji reprezentują łuki między nimi, ponadto graf posiado dwa dodatkowe wierzchołki - początkowy i końcowy - które pozwalają przypisać pierwszej i ostatniej operacji feromon. Dzięki takiej reprezentacji problemu możemy korzystać z wzorów przeznaczonych dla algorytmu TSP z drobnymi modyfikacjami.

Informacje o ilości feromonów utrzymywane są w macierzy NxN (N - liczba operacji). Rozwiązaniem generowanym przez mrówkę jest łańcuch wszystkich wierzchołków zachowujący zależnosci między operacjami w zadaniu. (sortowanie topologiczne) Przykładowy łańcuch: [0 1 4 2 7 5 8 3 6 9 10].

Przed uruchomieniam właściwego algorytmu, początkowa ilość feromonu na każdej ścieżce przyjmuje bardzo małą wartość, w naszej implementacji przyjęliśmy $\tau_0 = 0.001$

### 2.1.1 State Transition Rule

Każda mrówka wybierając drogę stosuję regułę zmiany stanu (State Transition Rule), którą obrazuje poniższy wzór:

$$s = \begin{cases} arg\ max_{u \in J(r)}\{[\tau(r,u)] \cdot [\eta(r,u)^{\beta}]\}, & \text{jeśli } q \leq q_0 \\ S, & \text{w przeciwnym razie} \end{cases} \tag{2}$$

gdzie:

- $(r, u)$ - łuk od wierzchołka r do u
- $J(r)$ - zbiór wierzchołków dostępnych w punkcie decyzyjnym r
- $\tau(r, u)$ - ilość feromonu na łuku (r,u)
- $\eta(r, u)$ - atrakcyjność łuku (r,u), która w przypadku naszego problemu zdefiniowana jest jako odwrotność długości operacji u
- $\beta$ - parametr kontrolujący wpływ atrakcyjności
- $q$ - losowa liczba z przedziału $\langle 0; 1\rangle$
- $q_0$ - zdefiniowany parametr przyjmujący wartości z przedziału $\langle 0; 1\rangle$
- $S$ - zmienna losowa wybierana zgodnia z rozkładem prawdopodobieństwa opisanym wzorem (2).

$$P(r,s) = \begin{cases} \frac{[\tau(r,s)]\cdot[\eta(r,s)^{\beta}]}{\sum_{u \in J(r)}[\tau(r,u)]\cdot[\eta(r,u)^{\beta}]}, & \text{jeśli } s \in J(r) \\ 0, & \text{w przeciwnym razie} \end{cases} \tag{3}$$

Powyższa metoda wyboru nosi nazwę pseudoruletki, której zasada działania opiera się na przydzieleniu wartości procentowej koła ruletki, odpowiadającej iloczynowi ilości feromonu i atrakcyjności. Po zakręceniu kołem, które jest równoważne wylosowaniu liczby, wybierany jest następny wierzchołek na drodze mrówki.

### 2.1.2 Aktualizacja wartości feromonów

W algorytmie stosujemy dwie strategie aktualizacji feromonów:

1. lokalna (local updating rule) - w czasie konstruowania ścieżki mrówka modyfikuje ilosć feromonu stosując się do poniższego wzoru:

$$\tau(r,s) \leftarrow (1-\rho)\cdot\tau(r,s) + \rho\cdot\tau_0 \tag{4}$$

   gdzie:

   - $\rho$ - współczynnik parowania feromonu (z przedziału $(0;1)$)

2. globalna (global updating rule) - gdy wszystkie mrówki danej populacji dotrą do celu (osiągną końcowy wierzchołek grafu) wartość feromonu jest aktualizowana zgodnie ze wzorem:

$$\tau(r,s) \leftarrow (1-\alpha)\cdot\tau(r,s) + \alpha\cdot\Delta\tau(r,s) \tag{5}$$

   gdzie:

   - $\alpha$ - parametr zaniku feromonu
   - $\Delta\tau(r,s)$ - przedstawia się wzorem:

$$\Delta\tau(r,s) = \begin{cases} (L_{gb})^{-1}, & \text{jeśli } (r,s) \in \text{najlepsza ścieżka} \\ 0, & \text{w przeciwnym razie} \end{cases} \tag{6}$$

     gdzie:

   - $L_{gb}$ - długość najlepszej ścieżki ($C_{max}$)

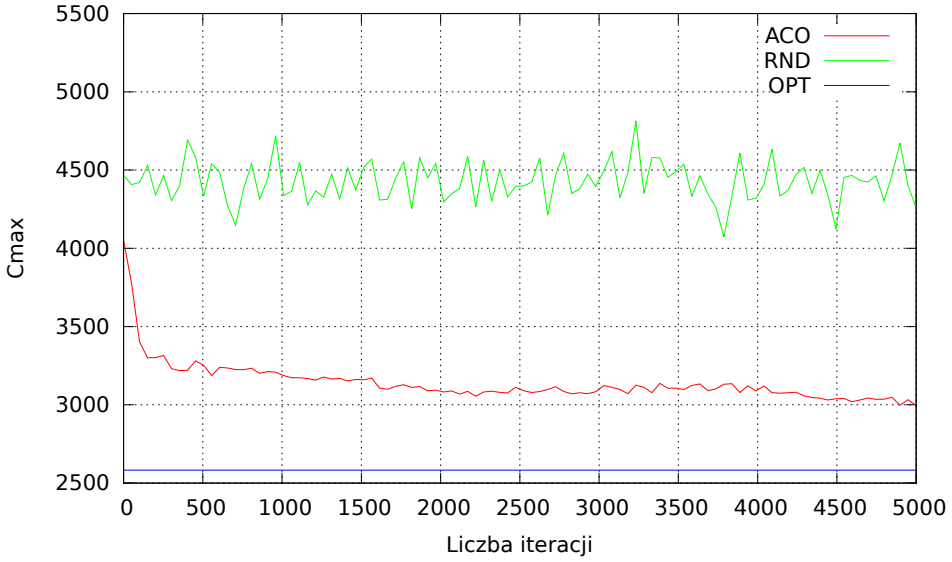
## 2.2 Instancje testowe

Każda znajdująca się poniżej instancja sterowana jest następującymi parametrami: $\rho, \alpha, q_0, \tau_0, \beta$. Klasy instancji podzieliliśmy na 4 grupy w zależności od rodzaju danych wejściowych: losowe, rosnące, malejące, v-kształtne.

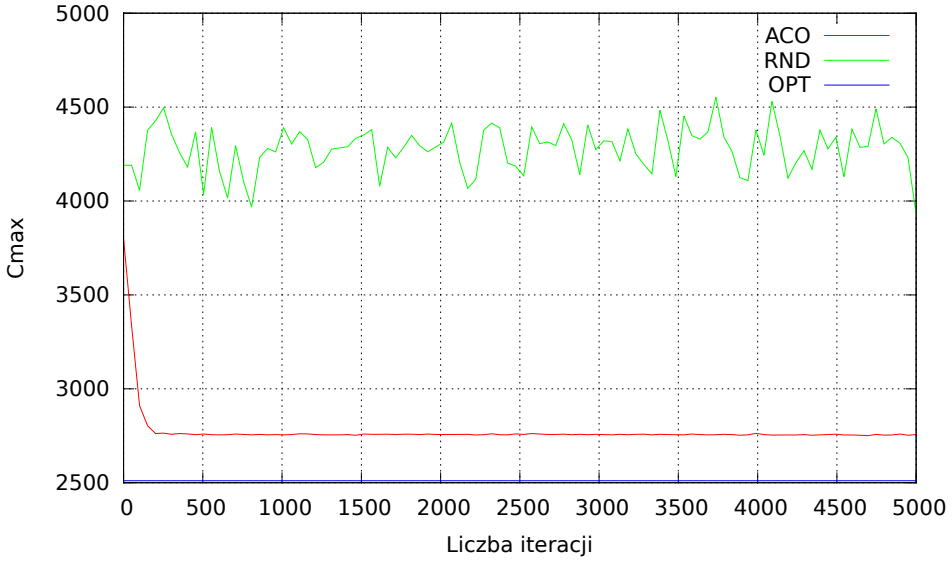
left=3cm,top=15mm, bottom=2cm

### 2.2.1 Różne klasy instancji danych wejściowych.

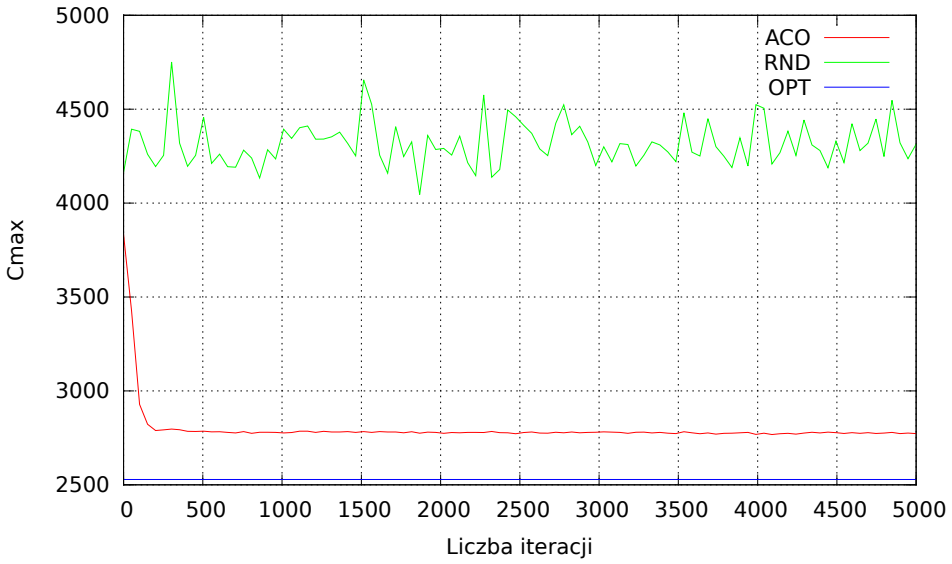
| | |
|---|---|
| liczba zadań | 50 |
| czasy operacji | $\langle 5; 100 \rangle$ |
| liczba mrówek | 10 |
| liczba pokoleń | 5000 |
| $\rho$ | 0.01 |
| $\alpha$ | 0.1 |
| $q_0$ | 0.8 |
| $\tau_0$ | 0.001 |
| $\beta$ | 2 |



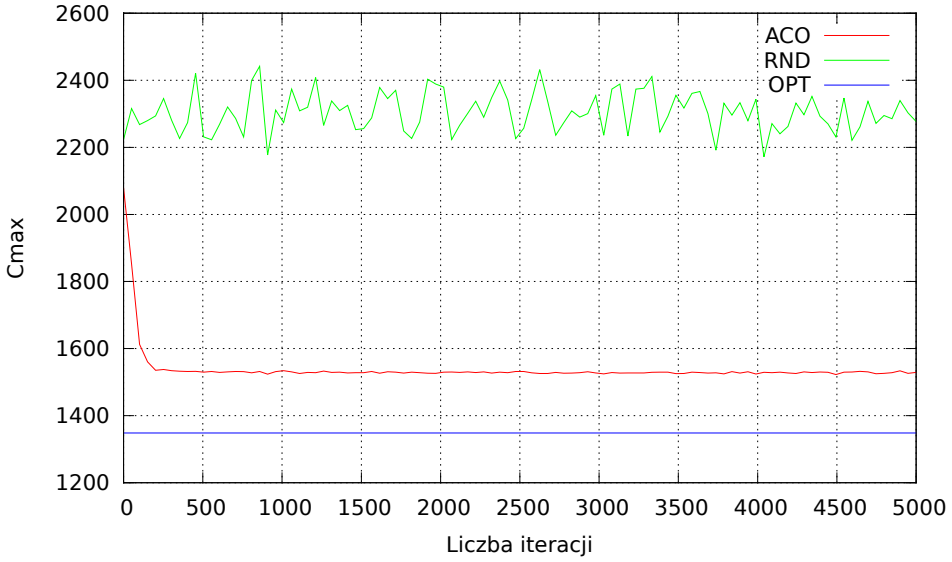

losowy




rosnący




malejące




v-kształtne

Algorytm losowy, niezależnie od rodzaju danych wejściowych oraz ilości iteracji, osiąga średnio podobne wyniki (względna odległość od optimum). Jest to spowodowane brakiem parametrów optymalizujących algorytm losowy. Natomiast ACO w kolejnych iteracjach zwraca wynik dążący do optimum. Z analizowanych klas instancji wyróżniają się dane losowe dla których po 5000 iteracji algorytm nie osiąga wyraźnej stabilizacji. Dla pozostałych zestawów danych po około 200 pokoleniach dochodzimy do stabilizacji algorytmu. Wynika to z uporządkowania pozostałych zestawów danych (rosnące/malejące długości kolejnych operacji).

Liczbowe przedstawienie klas instancji

| losowe dane | | | | |
|---|---|---|---|---|
| Iteracja | ACO | RND | opt/ACO(%) | opt/RND(%) |
| 50 | 3,796.33 | 4,294.33 | 68.03 | 60.14 |
| 1000 | 3,192.15 | 4,419.17 | 80.91 | 58.44 |
| 2000 | 3,089.85 | 4,388.67 | 83.59 | 58.85 |
| 3000 | 3,101.63 | 4,331.83 | 83.27 | 59.62 |
| 4000 | 3,077.53 | 4,458.83 | 83.92 | 57.92 |
| 5000 | 3,016.63 | 4,238.50 | 85.61 | 60.93 |

| rosnące dane | | | | |
|---|---|---|---|---|
| Iteracja | ACO | RND | opt/ACO(%) | opt/RND(%) |
| 50 | 3,397.37 | 4,300.33 | 73.89 | 58.38 |
| 1000 | 2,756.32 | 4,252.50 | 91.08 | 59.03 |
| 2000 | 2,757.57 | 4,480.17 | 91.03 | 56.03 |
| 3000 | 2,757.55 | 4,448.83 | 91.03 | 56.43 |
| 4000 | 2,757.02 | 4,457.83 | 91.05 | 56.31 |
| 5000 | 2,755.25 | 4,126.83 | 91.11 | 60.83 |

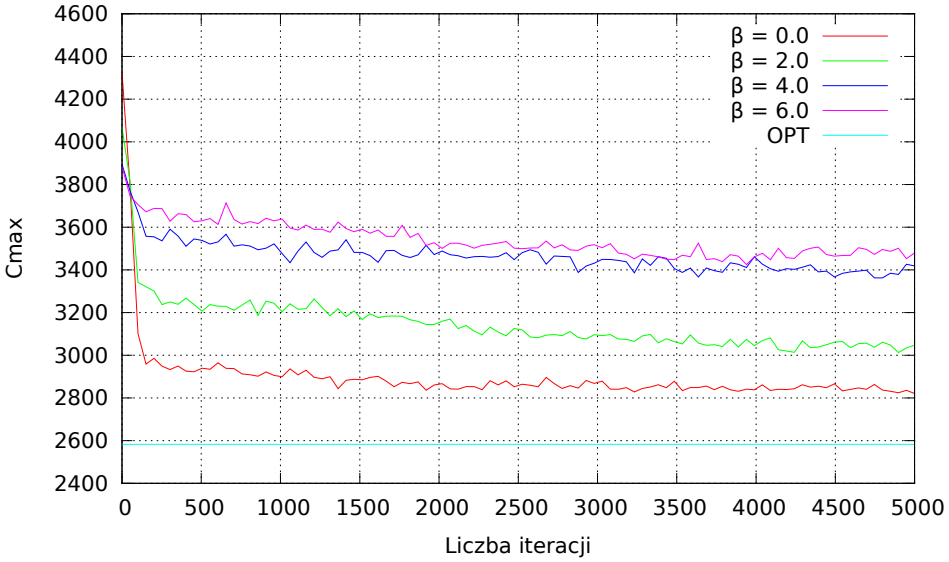
| malejące dane | | | | |
|---|---|---|---|---|
| Iteracja | ACO | RND | opt/ACO(%) | opt/RND(%) |
| 50 | 3,418.53 | 4,367.67 | 73.97 | 57.90 |
| 1000 | 2,778.82 | 4,341.67 | 91.00 | 58.24 |
| 2000 | 2,780.03 | 4,394.00 | 90.96 | 57.55 |
| 3000 | 2,777.32 | 4,295.50 | 91.05 | 58.87 |
| 4000 | 2,770.00 | 4,411.33 | 91.29 | 57.32 |
| 5000 | 2,774.88 | 4,333.50 | 91.13 | 58.35 |

| v-kształtne dane | | | | |
|---|---|---|---|---|
| Iteracja | ACO | RND | opt/ACO(%) | opt/RND(%) |
| 50 | 1,859.73 | 2,282.00 | 72.50 | 59.09 |
| 1000 | 1,530.27 | 2,277.67 | 88.11 | 59.20 |
| 2000 | 1,529.47 | 2,341.00 | 88.16 | 57.60 |
| 3000 | 1,528.95 | 2,304.33 | 88.19 | 58.51 |
| 4000 | 1,529.55 | 2,340.17 | 88.15 | 57.62 |
| 5000 | 1,527.00 | 2,268.83 | 88.30 | 59.43 |

Powyższe tabelki obrazują stosunek wartości optimum do wartości $C_{max}$ w otoczeniach wybranych puntów $\langle numer\ iteracji - 50; numer\ iteracji + 50\rangle$, potwierdzając brak optymalizacji rozwiązania w przypadku algorytmu losowego, niezależnie od rodzaju danych wejścciowych. Zastosowana metaheurystyka natomiast w przypadku danych malejących, rosnących i v-kształtnych ma bardzo podobne osiągi bliskie 90% wartości optymalnej. Dane losowe dają gorsze wyniki, osiągając ok. 85% wartości optymalnej, co jest spowodowane naturą losowego rozkładu danych.

### 2.2.2 Różne wartości parametru $\beta$.

| | |
|---|---|
| klasa instancji | losowa |
| liczba zadań | 50 |
| czasy operacji | $\langle 5; 100 \rangle$ |
| liczba mrówek | 10 |
| liczba pokoleń | 5000 |
| $\rho$ | 0.01 |
| $\alpha$ | 0.1 |
| $q_0$ | 0.8 |
| $\tau_0$ | 0.001 |

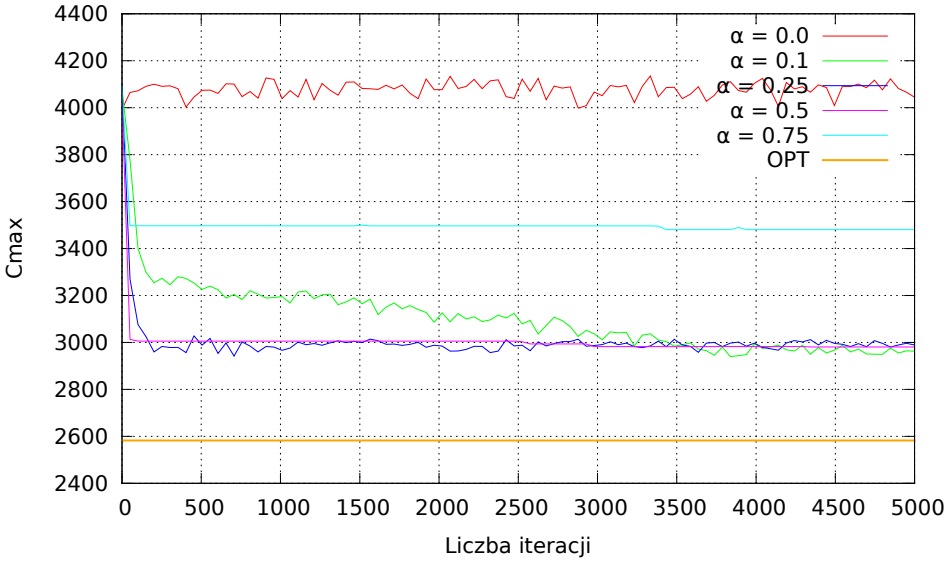



Rysunek 1: $\beta = \{0, 2, 4, 6\}$

Badany parametr $\beta$, wpływa na wagę atrakcyjności operacji ($\eta$), podczas wyboru następnego wierzchołka. W przeprowadzonym teście algorytm zwraca najlepsze osiągi dla wartości $\beta = 0$. Wraz ze wzrostem wartości tego parametru wyniki pogarszają się. Oznacza to, że nie branie pod uwagę atrakcyjności poprawia efektywność algorytmu. Jest to spowodowane tym, że algorytm ACO został zaprojektowany dla problemu TSP w którym uzyskanie rozwiązania nie wymusza odwiedzenia wszystkich krawędzi grafu, zatem wybieranie tych o małym koszcie sprzyja lepszym rozwiązaniom. W problemie FSP wyklucza się możliwość pominięcia jakiejkolwiek operacji, dlatego wybieranie najpierw krótszych, nie sprzyja lepszym rozwiązaniom, a jedynie utrudnia dobre uszeregowanie zadań.

### 2.2.3 Różne wartości parametru $\alpha$.

| | |
|---|---|
| klasa instancji | losowa |
| liczba zadań | 50 |
| czasy operacji | $\langle 5; 100 \rangle$ |
| liczba mrówek | 10 |
| liczba pokoleń | 5000 |
| $\rho$ | 0.01 |
| $\beta$ | 2 |
| $q_0$ | 0.8 |
| $\tau_0$ | 0.001 |

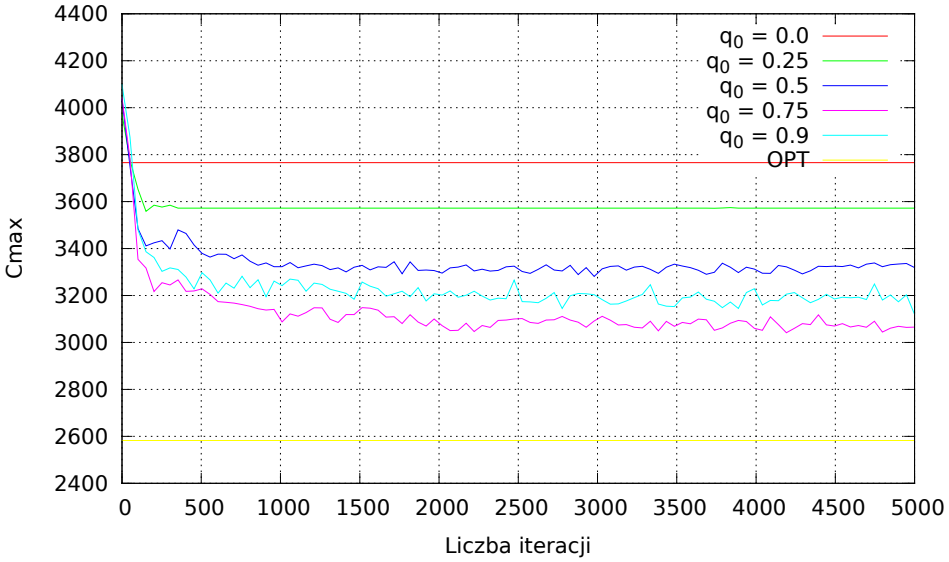



Rysunek 2: $\alpha = \{0.0, 0.1, 0.25, 0.5, 0.75\}$

Parametr $\alpha$ wpływa na zanik feromonów na wszystkich ścieżkach oraz przyrost na najlepszej ścieżce w globalnej aktualizacji (wzór 5). Dla wartości $\alpha = 0$ algorytm zachowuje się jak losowy, ponieważ nie są przyznawane ńagrodyźa najlepszą ścieżkę. Wzrost wartości parametru zwiększa szybkość stabilizacji (nie optymalizacji). Parametr wielkośći $\alpha = 0.5$ daje w pierwszych kilkudziesięciu iteracjach wartość stablilną przy dobrym wyniku. Jednak zbytnie zwiększenie tego parametru ogranicza dalsze przeszukiwanie ścieżek, gdyż za bardzo zwiększa feromony na wcześnie odkrytych, dobrych ścieżkach. Przy parametrze $\alpha = 0.1, 0, 25$ algorytm zwraca wartość zbliżoną do wartości przy $\alpha = 0.5$, jednak czas jej osiągnięcia jest dłuższy i algorytm nie osiąga pełnej stabilności.

### 2.2.4 Różne wartości parametru $q_0$.

| | |
|---|---|
| klasa instancji | losowa |
| liczba zadań | 50 |
| czasy operacji | $\langle 5; 100 \rangle$ |
| liczba mrówek | 10 |
| liczba pokoleń | 5000 |
| $\rho$ | 0.01 |
| $\beta$ | 2 |
| $\alpha$ | 0.1 |
| $\tau_0$ | 0.001 |

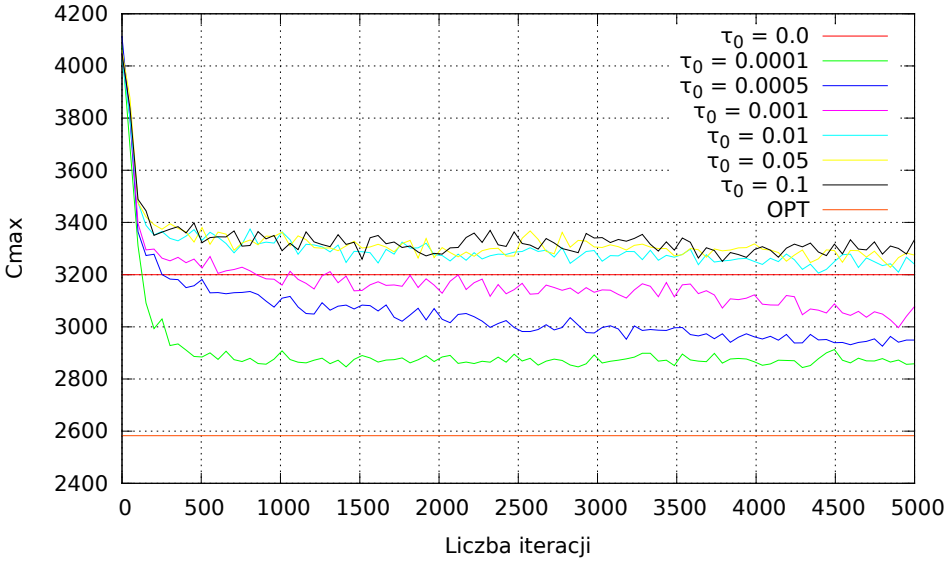



Rysunek 3: $q_0 = \{0.0, 0.25, 0.5, 0.75\}$

Parametr $q_0$ wpływa na ruletkę - większa wartość oznacza częstsze korzystanie z ruletki. Przykładowa wartość $q_0 = 0.8$ oznacza użycie ruletki w średnio 80% przypadków zgodnie ze wzorem 2. Przy zerowej wartości parametru nie ma poprawy jakości rozwiazań algorytmu, gdyż zawsze wybiera tę samą ścieżkę. Poprawa jakości rozwiązań rośnie wraz ze wzrostem wartości $q_0$, jednak zbyt duży wpływ ruletki powoduje spadek jakości rozwiązań. Potwierdzają to przeprowadzone testy, które wykazują lepsze wyniki dla parametru $q_0 = 0.75$ niż wartości $q_0 = 0.9$.

### 2.2.5 Różne wartości parametru $\tau_0$.

| | |
|---|---|
| klasa instancji | losowa |
| liczba zadań | 50 |
| czasy operacji | $\langle 5; 100 \rangle$ |
| liczba mrówek | 10 |
| liczba pokoleń | 5000 |
| $\rho$ | 0.01 |
| $\beta$ | 2 |
| $\alpha$ | 0.1 |
| $q_0$ | 0.8 |

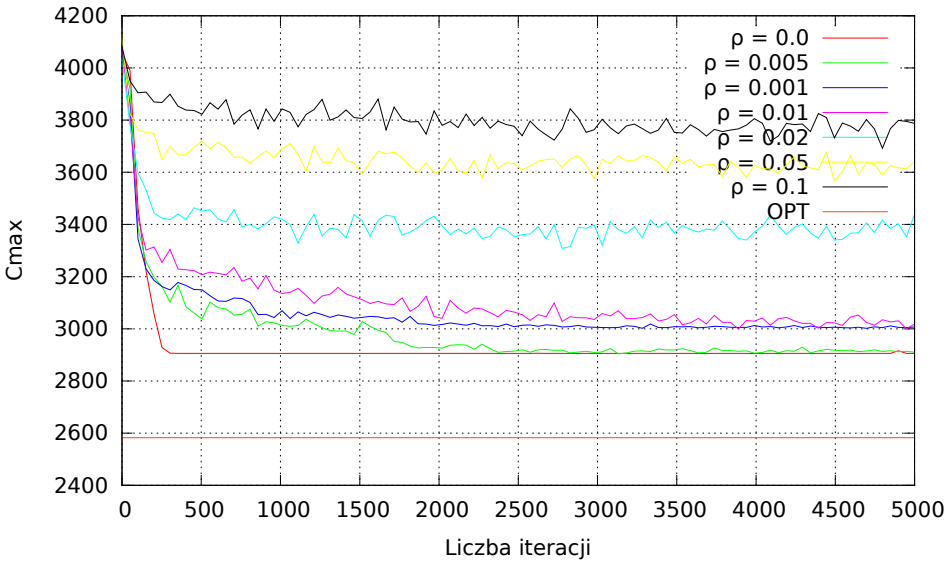



Rysunek 4: $\tau_0 = \{0.0, 0.0001, 0.0005, 0.001, 0.01, 0.05, 0.1\}$

Kolejna wartość sterująca programem - parametr $\tau_0$ - oznacza początkową wartość feromonu pomiędzy każdą parą wierzchołków grafu. Gdy $\tau_0 = 0$ wówczas, zgodnie ze wzorem 4,5 i 6 feromon aktualizowany jest jedynie na najlepszej ścieżce. Dlatego długość uszeregowania dla wszystkich operacji jest stała. Jakość rozwiązań rośnie wraz ze spadkiem współczynnika $\tau_0$. Na podstawie badań przeprowadzonych przez Marco Dorigo - twórcę ACO - najlepsze osiągi uzyskuje się przy $\tau_0 = \frac{1}{n \cdot L_{nn}}$ gdzie n to liczba operacji, a $L_{nn}$ to $C_{max}$ obliczony przez pokrewną metaheurystykę.

### 2.2.6 Różne wartości parametru $\rho$.

| | |
|---|---|
| klasa instancji | losowa |
| liczba zadań | 50 |
| czasy operacji | $\langle 5; 100 \rangle$ |
| liczba mrówek | 10 |
| liczba pokoleń | 5000 |
| $\tau_0$ | 0.001 |
| $\beta$ | 2 |
| $\alpha$ | 0.1 |
| $q_0$ | 0.8 |

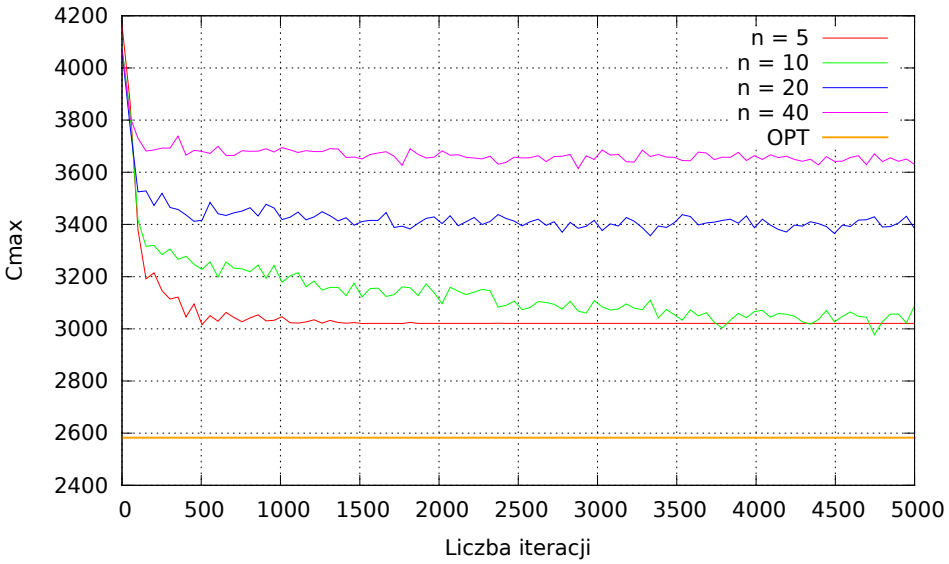



Rysunek 5: $\rho = \{0.0, 0.005, 0.01, 0.02, 0.05, 0.1\}$

Współczynnik $\rho$ odpowiada za parowanie feromonu w akutalizacji lokalnej (wzór 4). Im mniejsza wartość parametru, tym algorytm zwraca korzystniejszy wynik, co spowodowane jest doborem zbyt dużego współczynnika $\tau_0$. Zbyt duża wartość współczynnika parowania, obniża jakość rozwiązania. W badanym przypadku wartość $\rho \leq 0.01$ zwraca zadowalające wyniki.

### 2.2.7 Różne wartości parametru liczby mrówek.

| | |
|---|---|
| klasa instancji | losowa |
| liczba zadań | 50 |
| czasy operacji | $\langle 5; 100 \rangle$ |
| liczba pokoleń | 5000 |
| $\rho$ | 0.01 |
| $\tau_0$ | 0.001 |
| $\beta$ | 2 |
| $\alpha$ | 0.1 |
| $q_0$ | 0.8 |



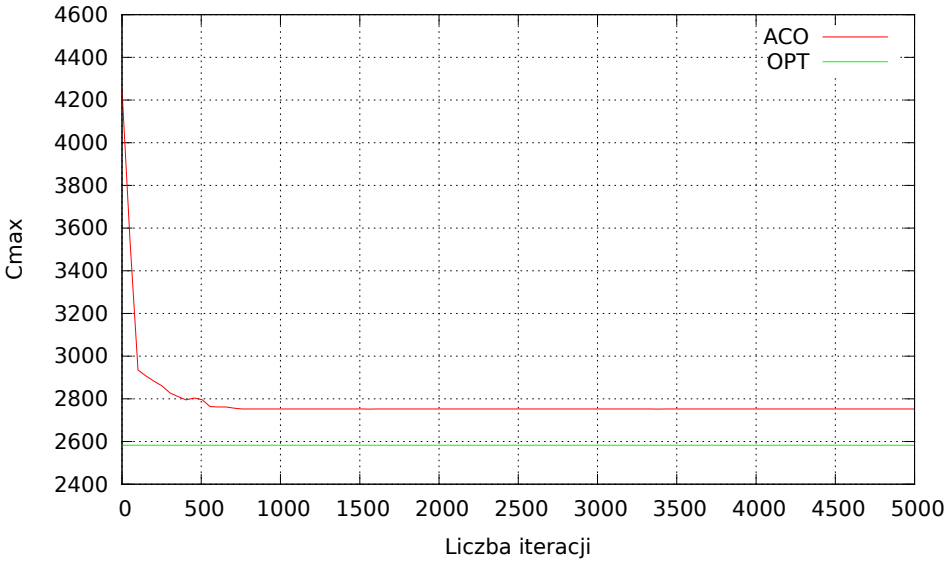

Rysunek 6: liczba mrówek = $\{5, 10, 20, 40\}$

W badanym przypadku, algorytm najlepiej zachowuje się dla liczby mrówek nie przekraczającej 20% liczby zadań. Większa liczba mrówek zbyt szybko hamuję poprawę wyników. 5 mrówek (10% liczby zadań) osiąga najlepszą stabilizację, jednak 10 mrówek zwraca podobne wartości, jednak potrzebuje na to więcej iteracji, ponadto nie osiągając wyraźnej stabilizacji.

# 3 Ocena efektywności

Jednoznaczna ocena efektywności z punktu widzenia parametrów jest bardzo trudna, ponieważ zależą one od siebie. Na podstawie wcześniejszych testów i analizy wygenerowanych wykresów, wykonaliśmy zestawienie wartości optymalnych, każdego z parametrów, następnie wprowadziliśmy te paramtry do algorytmu i zgodnie z oczekiwaniami, uzyskaliśmy najlepszą dotychczasową jakość wyniku.

| | |
|---|---|
| klasa instancji | losowa |
| liczba zadań | 50 |
| czasy operacji | $\langle 5; 100 \rangle$ |
| liczba mrówek | 10 |
| liczba pokoleń | 5000 |
| $\rho$ | 0.005 |
| $\tau_0$ | 0.0001 |
| $\beta$ | 0 |
| $\alpha$ | 0.1 |
| $q_0$ | 0.75 |




Rysunek 7: optymalne wartości parametrów

Stosunek optimum do uzyskanej wartości wynosi $\frac{2582.667}{2752.333} \approx 93.84\%$
Zważywszy na szybkość osiągania stabilizacji oraz małą różnicę w stosunku do nierealnego optimum, wynik jest bardzo zadowalający.

# 4 Wnioski

Algorytm ACO po odpowiednich modyfikacjach nadaje się do rozwiązywania problemu FSP (Flow-shop Scheduling Problem), a po dobraniu odpowiednich parametrów zwraca zadowalające (ok. 95% wartości optymalnej) wyniki. Algorytm cechuje duża szybkość znajdowania optymalnego rozwiązania. Wadą algorytmu AMH jest wysoki stopień sparametryzowania, który powoduje, e wymaga on czasochłonnego procesu „dostrajania".

# 5 Bibliografia


1. Marco Dorigo, Thomas Stutzle, Ant Colony Optimization, 2004 Massachusetts Institute of Technology
2. Thomas Stützle, An Ant Approach to the Flow Shop Problem
3. Adam Kostrubiec, Zastosowanie algorytmu mrówkowego w harmonogramowaniu produkcji przepływowej
4. Jun Zhang, Xiaomin Hu, X. Tan, J.H. Zhong and Q. Huang, Implementation of an Ant Colony Optimization technique for job shop scheduling problem
5. http://www.scholarpedia.org/article/Ant_colony_optimization
6. http://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms