Flow-Shop Scheduling
Laboratorium Optymalizacji Kombinatorycznej

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

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

Prowadzący: Marcin Radom

15 stycznia 2014

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:

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

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*.

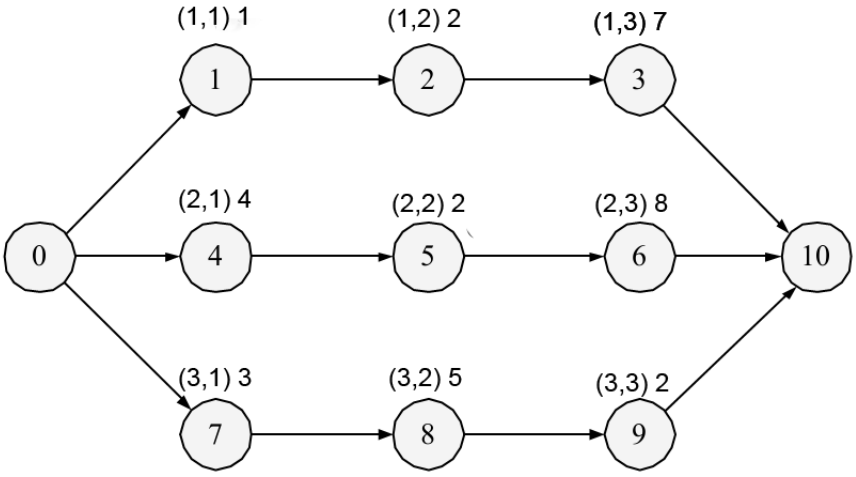
1.2.3 Uruchomienie

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:

$$s = \begin{cases} arg\ max_{u \in J(r)}\{[\tau(r,u)] \cdot [\eta(r,u)^{\beta}]\}, & \text{jeśli } q \leq q_0 \\ S, & \text{w przeciwnym razie} \end{cases} \tag{1}$$

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
- β - 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).

$$P(r,s) = \begin{cases} \frac{[\tau(r,s)] \cdot [\eta(r,s)^{\beta}]}{\sum_{u \in J(r)} [\tau(r,u)] \cdot [\eta(r,u)^{\beta}]}, & \text{jeśli } s \in J(r) \\ 0, & \text{w przeciwnym razie} \end{cases} \tag{2}$$

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:

$$\tau(r,s) \leftarrow (1-\rho)\cdot\tau(r,s) + \rho\cdot\tau_0$$

   gdzie:

   - ρ - 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:

$$\tau(r,s) \leftarrow (1-\alpha)\cdot\tau(r,s) + \alpha\cdot\Delta\tau(r,s)$$

   gdzie:

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

$$\Delta\tau(r,s) = \begin{cases} (L_{gb})^{-1}, & \text{jeśli } (r,s) \in \text{najlepsza ścieżka} \\ 0, & \text{w przeciwnym razie} \end{cases} \tag{3}$$

   gdzie:

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

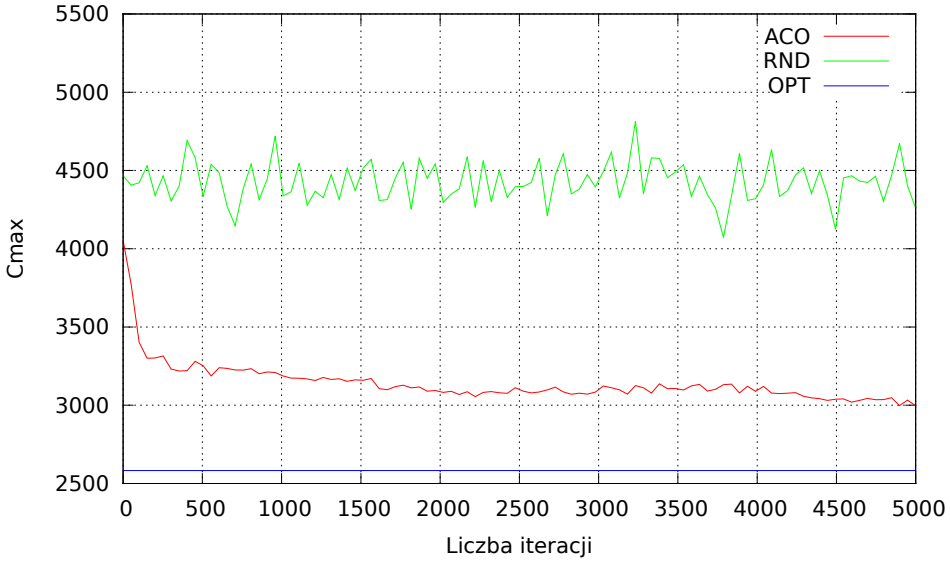
2.2 Instancje testowe

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.

2.2.1 Grupa instancji 1

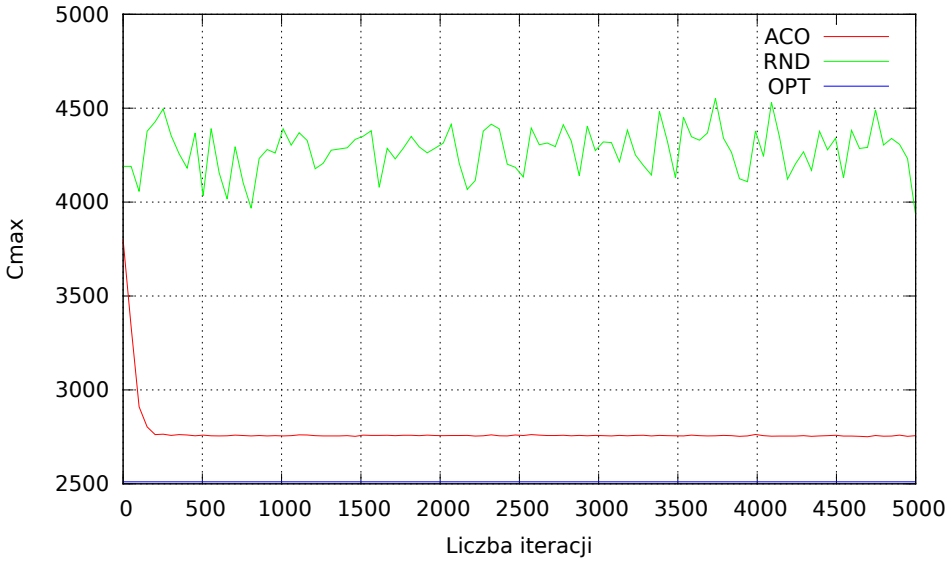
Różne klasy instancji danych wejściowych.

liczba zadań	50
czasy operacji	$\langle 5; 100 \rangle$
liczba mrówek	10
liczba pokoleń	5000
ρ	0,01
α	0,1
q_0	0,8
τ_0	0,001
β	2

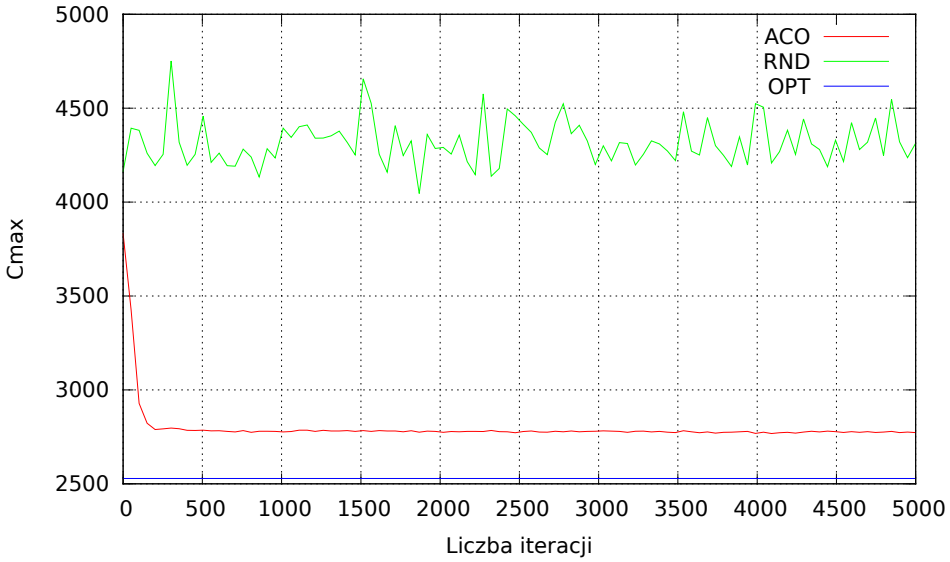


Rysunek 1: Losowe

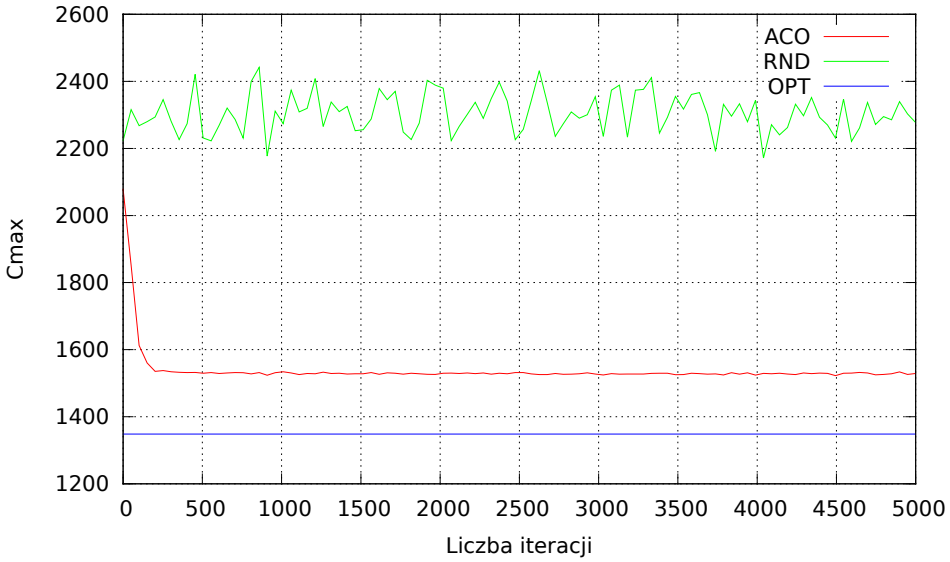
Wyczerpujący opis wykresu



Rysunek 2: Rosnące



Rysunek 3: Malejące



Rysunek 4: V-kształtne

2.2.2 Grupa instancji 2

Różne wartości parametru β.

klasa instancji	losowa
liczba zadań	50
czasy operacji	$\langle 5; 100 \rangle$
liczba mrówek	10
liczba pokoleń	5000
ρ	0,01
α	0,1
q_0	0,8
τ_0	0,001

Rysunek 5: $\beta = \{0, 2, 4, 6\}$

2.2.3 Grupa instancji 3

Różne wartości parametru α.