Flow-Shop Scheduling
Laboratorium Optymalizacji Kombinatorycznej

Szymon Gramza
109785
szymon.gramza@student.put.poznan.pl

Maciej Sobkowski
112711
maciej.sobkowski@student.put.poznan.pl

Prowadzący: dr inż. Marcin Radom
Wydział Informatyki Politechniki Poznańskiej

14 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 z poziomu konsoli wydać polecenie *make*. Prawidłowe zakończenie procesu kompilacji zostanie zasygnalizowane komunikatem *Zakończono linkowanie!*.

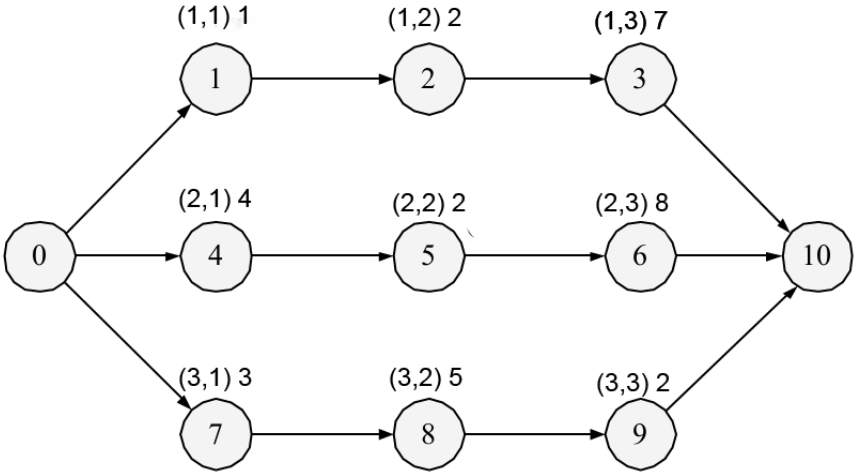
1.2.3 Uruchomienie

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 zwrócony 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:



Informacje o ilości feromonów utrzymywane są w macierzach NxN (N - liczba zadań), odpowiadających poszczególnym maszynom. Każda mrówka wyboru drogi, dokonuje losowo, a wpływ pozostawionych feromonów uzyskaliśmy stosując psuedo-ruletkę, która zwiększa prawdopodobieństwo wylosowania danej drogi przydzielając "pasmo dostępności" danej drogi w losowanym zakresie w zależności od aktualnej wartości feromonu. W rozwiązaniu przyjęto także element parowania pozostawionych feromonów, którą można wygodnie regulować za pomocą współczyniika parowania.

Poszukiwanie rozwiązania opiera się na pętli z warunkiem wykonania wszystkich zleceń, która wykonuje następujące kroki:

1. dla każdej z maszyn: jeżeli maszyna jest bezczynna to na podstawie tablicy iteratorów znajdź zadanie przeznaczone dla tej maszyny (i zapisz czas jego rozpoczęcia) lub pozostań w stanie bezczynności,
2. znajdź czas zadania, które kończy się najwcześniej i zapisz w zmiennej t,
3. wykonaj $T+ = t$ i dla każdego aktualnie przetwarzanego zadania zmniejsz pozostały czas wykonania o t,
4. jeżeli istnieje zadanie o pozostałym czasie wykonania równym 0, to usuń je z maszyny i przesuń iterator zlecenia o 1 w prawo.

Po zakończeniu powyższej pętli długość uszeregowania znajduje się w zmiennej T.

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

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

2.2 Instancja testowa

Do porównania wyników wygenerowano różne rodzaje instancji testowych, które regulowano następującymi parametrami:

Algorytm został przetestowany przy pomocy dostarczonego oprogramowania. Dla wszystkich rozwiązań oprogramowanie sprawdzające zwróciło wynik pozytywny.

7		
0	2	4
0	2	6
0	4	6

Tabela 2: wynik uszeregowania algorytmem zachłannym dla instancji zaprezentowanej w tabeli 1

Rysunek 1: wykres Gantta dla uszeregowania podanego w tabeli 2

3 Ocena efektywności

Dla instancji tai20 — tai25 dokonano pomiaru czasu wykonywania i jakości uszeregowania, co przedstawiono na poniższych wykresach.

Wykres 1: długość uszeregowania w porównaniu do wartości najlepszych granic dla instancji Taillarda

Wykres 2: czas wykonywania algorytmu dla zadanych instancji

Dla instancji tai25 przeprowadzono pomiar czasu wykonywania algorytmów dla $n = 1 \ldots 20$. Wyniki zaprezentowano na poniższym wykresie.

Wykres 3: czas wykonywania algorytmu dla instancji tai25 przy zmiennej liczbie wczytanych zleceń

Przypadek pesymistyczny (kiedy w każdej sprawdzanej chwili czasu kończy się tylko jedno zadanie) ma złożoność obliczeniową rzędu $O(mn^2)$, gdzie: m — liczba maszyn, n — liczba zleceń. W przypadku optymistycznym (wszystkie aktualnie przetwarzane zadania kończą się w tej samej chwili) uzyskujemy złożoność $O(mn)$.

4 Wnioski

Zaproponowana heurystyka zachłanna generuje rozwiązania dopuszczalne, ale dalekie od optimum. Metoda ta nie korzysta z pełnej informacji odczytanej z pliku i nie przewiduje przyszłego wykorzystania maszyn celem zoptymalizowania obciążenia. Zaimplementowany algorytm jest symulacją pracy maszyn, gdzie kolejne zadania zgłaszają żądanie zasobów po zakończeniu przetwarzania ich poprzednika.
Otrzymane w ten sposób rozwiązanie może posłużyć jako baza dla optymalizacji przy pomocy bardziej zaawansowanych algorytmów heurystycznych.