Adam Nowacki inf155838 adam.nowacki@student.put.poznan.pl

Paweł Kolec inf155873 [pawel.kolec@student.put.poznan.pl](mailto:pawel.kolec@student.put.poznan.pl)

Data: 21.11.2023r.

**Temat II: Job-Shop**

**Algorytm Losowy**

**Uruchomienie programu**

Nasz algorytm kompiluje się w systemach Linux za pomocą następującej komendy:

**g++ Job\_shop.cpp -o Job\_shop**

Przy uruchamianiu programu należy podać 3 argumenty są to: plik wejściowy, plik wyjściowy(wyniikowy), znak formatu (B lub T w zależności od formatu instancji w pliku wejściowym).

Aby uruchomić program w zależności od formatu instancji będących w pliku wejściowych należy:

1. W przypadku formatu Beasleya przykładowe uruchomienie to: **./Job\_shop input.txt output.txt B**
2. W przypadku formatu Taillarda przykładowe uruchomienie to: **./Job\_shop input.txt output.txt T**

**Ogólny opis problemu**

Problem Job-Shop to, gdzie celem jest zoptymalizowanie kolejności wykonywania operacji dla zestawu zadań na dostępnych maszynach. Każde zadanie składa się z sekwencji operacji, z których każda musi być wykonana na określonej maszynie, w określonej kolejności, a operacje mają różne czasy trwania. Istnieje określony zestaw maszyn, które mogą obsługiwać różne operacje. Kolejność operacji ma kluczowe znaczenie, a celem jest minimalizacja długości uszeregowania, czyli czasu trwania całego procesu od rozpoczęcia pierwszej operacji do zakończenia ostatniej. W praktyce, problem ten stanowi wyzwanie w zarządzaniu produkcją, gdzie konieczne jest efektywne planowanie, aby zminimalizować czas i zasoby potrzebne do ukończenia wszystkich zadań.

Problem Job-Shop należy do klasy problemów trudnych obliczeniowo (NP-trudnych), co oznacza, że znalezienie optymalnego rozwiązania może być czasochłonne, szczególnie dla większych instancji problemu.

**Opis metody rozwiązania problemu**

Algorytm, który zastosowaliśmy do rozwiązania problemu szeregowania Job Shop to algorytm losowy. Algorytm jest prostą heurystyką losową, która eksploruje przestrzeń rozwiązań poprzez losowanie kolejności zadań i przypisywanie operacji do maszyn. Może to prowadzić do różnych rozwiązań w zależności od wyników losowania.

Główną ideą algorytmu jest losowanie w każdej jednostce czasu kolejności zadań, które mają zostać przypisane do maszyn. Jeśli operacja wylosowanego zadania, nie może zostać przypisana do maszyny, ponieważ jest ona zajęta, to jest pomijana w tej jednostce czasu. Algorytm wykonuje losowania aż do momentu, gdy wszystkie operacje zostaną zakończone. Również jeśli dana operacja ma czas równy 0, jest pomijana, a jej moment rozpoczęcia jest wypisany w wynikowym rozwiązaniu.

Algorytm działa przez określony czas lub przez określoną liczbę iteracji. Obie wartości podane są wewnątrz kodu źródłowego. Przykładowo jeśli użytkownik ustawi liczbę iteracji na 1000, to algorytm wybierze najlepsze rozwiązanie ze wszystkich, które wykonał. W naszym przypadku najlepszym rozwiązaniem będzie najkrótsza uzyskana przez algorytm długość uszeregowania. Również w przypadku, w którym algorytm przekroczy czas podany w pliku przez użytkownika, to program zakończy się i wypisze najlepsze rozwiązanie.

Algorytm posiada również warunki, które sprawdzają w każdej jednostce czasu czasy trwania operacji. Jeśli wszystkie operacje mają pozostały czas trwania większy od 1, nazwany w kodzie jako left\_time, to wtedy pozostały czas trwania wszystkich operacji jest zmniejszony o najmniejszy czas trwania operacji. Dzięki temu algorytm wykonuje skok w jednostkach czasu o kilka lub więcej w zależności od przypadku, więc nie musi wykonywać się w każdej jednostce czasu. Przyśpiesza to znacznie czas wykonania algorytmu.

Program jest przygotowany z myślą o dwóch typach instancji wejściowych w formacie Beasleya lub Taillarda. Gdy uruchamiamy program z poziomu konsoli, jako trzeci argument po pliku wejściowym i wyściowym należy podać typ formatu. Literka B oznacza format Beasleya, a literka T format Taillarda. Program posiada dwie funkcje, które uruchamiają się w zależności o formatu podanego przez użytkownika. Są to funkcje void Beasley() oraz void Taillard(). Przyjmują one argumenty podane przez użytkownika oraz int x, czyli liczbę iteracji jakie ma wykonać algorytm. Funkcja Taillard różni się od funkcji Beasley jedynie sposobem wczytywania danych oraz w niektórych miejscach numery maszyn są zmniejszone o 1, ponieważ w formacie Taillarda maszyny numerują się od 1 w przeciwieństwie do formatu Beasleya, w którym numeracja maszyn zaczyna się od 0. Algorytm działa również bezproblemowo, jeśli zmniejszymy podaną w pliku ilość zadań, ponieważ posiada pętlę, która nie wczytuje nadmiarowych w takim przypadku czasów wykonywania z pliku.

**Badania efektywnościowe**

Do wykonania testów uzyskanych rozwiązań przyjęliśmy 1 minutę czasu wykonywania dla każdej instancji (pliku wejściowego).

1. Wykres przedstawiający medianę czasu uszeregowania instancji oraz maksymalny oraz minimalny czas uszeregowania danej instancji.
2. Wykres przedstawiający jakość uzyskanych rozwiązań w zależności od instancji w odniesieniu do lower bound znajdującego się w plikach.
3. Wykres przedstawiający uzyskane długości uszeregowania w zależności od instancji.
4. Wykres przedstawiający czas szeregowania dla tai25.txt w zależności od liczby zadań.

**Wnioski**

- Algorytm losowy jest prosty do zaimplementowania, co sprawia, że może być szybko wdrożony. Brak skomplikowanych struktur danych i algorytmów sprawia, że jest to podejście łatwe do zrozumienia.

- Algorytm losowy nie stosuje żadnych heurystyk ani zaawansowanych strategii planowania. Jego działanie opiera się na czystym losowaniu permutacji, co może prowadzić do niestabilnych i nieskutecznych rozwiązań.

- Brak gwarancji co do optymalności rozwiązania. Wynik działania algorytmu losowego może być suboptymalny, a efektywność zależy od liczby iteracji i przypadkowości generowanych permutacji.

- Algorytm losowy może działać szybko, co jest korzystne w przypadku dużych instancji problemu Job Shop. Jednakże, szybkość działania może być uzyskiwana kosztem jakości rozwiązania.

- Algorytm losowy może być użyteczny jako narzędzie do generowania początkowych rozwiązań, które później są doskonalone przez bardziej zaawansowane algorytmy optymalizacyjne.