Sprawozdanie:

1. Opis problemu

Dana jest pojedyncza maszyna/procesor oraz zbiór *J*={1,…,*j*,…*n*} *n* niepodzielnych zadań. Każde z zadań j charakteryzuje się czasem wykonywania *pj*, terminem dostępności *rj* oraz czasem stygnięcia *qj*. Należy tak uszeregować zadania, aby czas zakończenia wykonywania wszystkich zadań był jak najkrótszy (minimalizujemy tzw. długość uszeregowania – makespan).

1. Wybrane algorytmy
   1. Algorytmy heurystyczne oparte o sortowanie (po rj i qj)

Stworzono dwa takie algorytmy: jeden sortuje zadania po terminie dostępności rj rosnąco a drugi po czasie stygnięcia qj malejąco. Pseudokod: sortowanie bąbelkowe

* 1. Przegląd zupełny

Algorytm oblicza kryterium dla wszystkich przypadków i wybiera rozwiązanie optymalne. Pseudokod:

n := liczba\_zadań\_w\_problemie;

std::vector<int> perm(n) := {0,n}

std::vector<int> optimum(n);

minimalnyCmax = INT\_MAX;

do {

rozwiązanie := perm;

cmax := countCzasWykonania(rozwiązanie);

if(cmax < minimalnyCmax){

minimalnyCmax := cmax;

optimum := perm;

}

} while (std::next\_permutation(perm.begin(), perm.end()));

Rozwiązanie := optimum

Final\_Cmax := minimalnyCmax

* 1. Algorytm konstrukcyjny własnego autorstwa

Algorytm sortuje zadania najpierw malejąco po czasie stygnięcia qj. Następnie zadania wykonywane są zgodnie z tą kolejnością. Jeśli wybrane zadanie nie jest jeszcze dostępne, brane jest kolejne. Po ukończeniu każdego zadania sprawdzana jest dostępność wszystkich zadań od początku.

Pseudokod:

p.sortuj\_po\_q();

czas := 0;

std::vector<int> kolejnosc;

std::vector<bool> wykonane(liczba\_zadan, false);

while ( kolejnosc.size() < liczba\_zadan) {

znalezione := false;

for (i in liczba\_zadan) {

if (NOT wykonane[i] AND czas >= p(i).TerminDostepnosci() ) {

kolejnosc.push\_back(i);

czas += p(i).CzasWykonania();

wykonane[i] := true;

znalezione := true;

break;

}

}

if (NOT znalezione) czas += 1;

}

Rozwiązanie := kolejnosc

Final\_Cmax := countCzasWykonania(rozwiązanie)

* 1. Algorytm Schrage

Algorytm szereguje zadania od najmniejszego terminu dostępności *rj* a następnie od najdłuższego czasu stygnięcia *qj*. Pseudokod:

1. Opis przeprowadzonego eksperymentu numerycznego: procesor, pamięć

Tabela . Wartość kryterium, w nawiasach kwadratowych błąd względny (w %) oraz czas działania zaimplementowanych algorytmów.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rozmiar instancji | Wartość kryterium  Sort r\_j | Czas dział. alg. Sort r\_j [s] | Wartość kryterium  Sort q\_j | Czas dział. alg. Sort q\_j [s] | Wartość kryterium  Konstr. | Czas dział. alg.  Konstr. [s] | Wartość kryterium  Schrage | Czas dział. alg. Schrage |
| 5 | 35 [50%] | 0.01 |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  | 687 | 0 |  |  |
| 11 |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Uwaga, Tabelę dla większej liczby algorytmów sensowniej zorientować tabelę poziomo. W kolumnie „Rozmiar instancji problemu” umieszczamy liczbę zadań, która jest w wybranej instancji. Proszę pamiętać też o kolumnach dla przeglądu zupełnego. Dla alg. Schrage z podziałem zadań nie liczymy błędu, ten algorytm rozwiązuje inny problem.

Jak liczymy błędy:

* Gdy znamy wartość kryterium dla rozwiązania optymalnego, wówczas liczymy błąd względny względem rozwiązania optymalnego
* Gdy nie znamy wartości kryterium dla rozwiązania optymalnego, wówczas liczymy błąd względny względem najlepszego znalezionego rozwiązania

Sensowne jest wygenerowanie np. 50 instancji o danym rozmiarze i wyznaczenie średniego i maksymalnego błędu. W tym przypadku, ma to sens tylko dla rozmiarów instancji 5-12, dla większych nie mamy tylu benchmarków, a tu jesteśmy w stanie porównać efektywność algorytmów z przeglądem zupełnym.

Po angielsku przegląd zupełny nazywa się exhaustive search albo brute search.

1. Wnioski, uwagi, co zauważył\_ś, co było problemem.