Metody optymalizacji
Lista 2

Jakub Kołakowski
221457

Zadanie 1.

Dane liczbowe o m cechach populacji, znajdują się na n różnych serwerach. Należy wyznaczyć serwery, z których pobranie danych zajmie najmniej czasu.

Rozwiązanie:

m - liczba cech populacji
n - liczba serwerów z cechami
Available - macierz mówiąca czy dana cecha i dostępna jest na serverze j
AccessTime - wektor czasów dostępu na każdy z serverów
x – wektor zmiennych decyzyjnych określający czy następuje pobieranie z danego serwera

funkcja celu: minimalizacja czasu dostępu do cech populacji znajdujących się na serwerach
min AccessTime * x

ograniczenie:
Każda cecha musi i musi zostać pobrana z serwera, na którym jest ona dostępna

$$\forall i \left(\sum_{j=1}^{n} x[j] * Available[i, j] = 1 \right)$$

Wyniki:

przykładowe dane:

AccessTime = [3; 2; 4]
Available = [1 0 1;
0 1 1]

serwery do podłączenia: [0, 0, 1]

Widać, więc że warto podłączyć się jedynie do serwera trzeciego, gdyż zawiera on obie cechy populacji, a czas dostępu jest mniejszy niż suma czasów połączeń do dwóch innych serwerów.

AccessTime = [3; 2; 6]
Available = [1 0 1;
0 1 1]

serwery do podłączenia: [1, 1, 0]

Teraz z kolei opłaca się podłączyć do dwóch różnych serwerów z krótszym czasem dostępu, niż do jednego z dłuższym czasem.

Zadanie 2.

Należy ułożyć program P, który obliczy pewien zbiór funkcji I, korzystając z podprogramów $P_{i,j}$, z których każdy zajmuje $r_{i,,j}$ komórek pamięci i jego czas wykonania wynosi $t_{i,\ j}$, aby program P nie zajmował łącznie więcej niż M komórek pamięci oraz czas jego wykonania był minimalny.

Rozwiązanie:

m - liczba mozliwych funkcji do obliczenia
n - liczba podprogramów
ProgramMemory - macierz określająca zużycie pamieci każdego z programów
ProgramExecTime - macierz określająca czas potrzebny na wykonanie kazdego z programow
FuncToCompute - wektor mówiący które funkcje chcemy obliczyć
MaxMemory - maksymalna dostepna ilosc pamieci na obliczenie wszystkich zadanych funkcji

funkcja celu: minimalizacja czasu wykonania wszystkich podprogramów
min ProgramExecTime * x

ograniczenia:

wybrane podprogramy nie mogą przekroczyć możliwej dostępnej pamięci

$$(\sum_{i=1}^{m}\sum_{j=1}^{n} x[i,j] * ProgramMemory[i,j]) \leq MaxMemory$$

wybieramy podprogramy tylko dla wybranych funkcji

$$\forall i (\sum_{j=1}^{n} x[i,j]) = FuncToCompute[i]$$

Wyniki:

Przykładowe dane:
ProgramMemory = [5 6 4;
1 2 3;
10 5 16]
ProgramExecTime = [11 2 15;
3 4 5;
5 10 4]
FuncToCompute = [1; 0; 1]
MaxMemory = 10

wybrane programy: [1 0 0;
0 0 0;
0 1 0]

Wybrany został pierwszy podprogram dla pierwszej funkcji oraz drugi podprogram dla ostatniej funkcji. Łącznie zajmują one 10 komórek pamięci, czyli maksymalna dopuszczalna wartość. Gdyby wybrany został trzeci podprogram dla pierwszej funkcji, zostałoby użyte 9 komórek pamięci jednak czas wykonania programu wzrósłby o 4 jednostki czasu.

Zadanie 3.

Zadania muszą zostać wykonane na trzech procesorach P1, P2, P3. Muszą zostać spełnione warunki:

1) każdy procesor może wykonywać w danym momencie tylko jedno zadanie
2) każde zadanie musi być wykonane najpierw na procesorze P1, potem P2, a potem P3.
3) kolejność wykonywania zadań na wszystkich procesorach jest taka sama.

Rozwiązanie:

m - liczba procesorów
n - liczba zadań
programExecTime - macierz określająca czas potrzebny na wykonanie każdego z zadań na każdym z procesorów
permutation - macierz określająca permutacje
endTaskTime - macierz określająca czas zakończenia zadania na każdej z maszyn

funkcja celu:
minimalizacja maksymalnego czasu zakończenia na ostatnim procesorze
$C_{max} = C_{permutation} \rightarrow \min$

ograniczenia:

stworzenie permutacji, w każdym rzędzie i kolumnie może znajdować się tylko jedna ‘1’. Pozycja j ‘1’ w rzędzie i oznacza że i-tą liczbą w permutacji jest j.

$$\forall i(\sum_{j=1}^{n} permutation[i, j])=1$$

$$\forall i(\sum_{j=1}^{n} permutation[j, i])=1$$

czas końca pierwszego zadania na pierwszym procesorze to czas wykonywania tego zadania

$$\sum_{j=1}^{n} permutation[1, j]*programExecTime[1, j]=endTaskTime[1,1]$$

czas zakończenia pozostałych zadań to czas zakończenia poprzedniego zadania plus czas wykonywania aktualnego zadania

$$(\forall i>1)endTaskTime[1, i-1]+\sum_{j=1}^{n} permutation[i, j]*programExecTime[1, j]=endTaskTime[1, i]$$

czas zakończenia pierwszego zadania na kolejnych procesorach to czas zakończenia pierwszego zadania na wcześniejszym procesorze plus czas wykonywania danego zadania na aktualnym procesorze

$$(\forall p>1)endTaskTime[p-1,1]+\sum_{j=1}^{n} permutation[1, j]*programExecTime[p, j]=endTaskTime[p,1]$$

czas zakończenia kolejnych zadań na kolejnych procesorach jest większy, bądź równy czasowi zakończenia zadania na wcześniejszym procesorze lub czasowi zakończenia poprzedniego zadania na aktualnym procesorze, plus czas wykonywania aktualnego zadania

$$(\forall p>1 \forall i>1)endTaskTime[p-1, i]+\sum_{j=1}^{n} permutation[i, j]*programExecTime[p, j]=endTaskTime[p, i]$$

$$(\forall p>1 \forall i>1)endTaskTime[p, i-1]+\sum_{j=1}^{n} permutation[i, j]*programExecTime[p, j]=endTaskTime[p, i]$$

maksymalny czas zakończenia na ostatnim procesorze:

$$(\forall p>1)endTaskTime[p, n]\leq cost$$

Wyniki:

przykładowe dane:

programExecTime = [2 3 4 1;
3 2 2 3;
4 1 2 3]

4222113333
x44422111xx33
xxxx4442x111133

Widać, że zadanie 1 musi czekać chwilę na trzecim procesorze, aż zostanie wykonane w całości na drugim procesorze.

programExecTime = [3 3 4 2 3 2;
3 3 1 2 2 2;
4 3 2 3 1 3]

66333355544111222
xx66xx3xx5544x111x222
xxxx666xxx335444x1111222

Widać tutaj, że nie jest to optymalne ułożenie dla drugiego procesora, gdyż ostatnie oczekiwanie na rozpoczęcie zadania 2 nie ma sensu, jednak jest to akceptowalne, gdyż minimalizowany jest całkowity ostateczny czas na trzecim procesorze.