Laboratoria Systemy Inteligentne

Sprawozdanie z projektu

## Konrad Antczak 260326 Igor Lewandowski 260357

# Temat

Celem tego projektu było wyznaczenie jak najlepszego rozwiązania problemu make span z losowo określoną zajętością zadań. Problem ten polega na znalezieniu scenariusza rozdysponowania zadań pracownikom tak, aby pracownik, który pracuje najdłużej skończył jak najszybciej.

N- ilość pracowników

P-ciąg zadań, których ilość wynosi K (processing time)

- najkrótsza możliwa (shortest) zajętość k-tego zadania

- najdłuższa możliwa (longest) zajętość k-tego zadania

D-decyzja przypisania konkretnego zadania konkretnemu pracownikowi

-przypisanie zadania pierwszemu pracownikowi

- przypisanie zadania N-temu pracownikowi

Suma wartości wewnątrz tabeli przypisania konkretnego zadania wynosi 1. Nie można konkretnego zadania przypisywać częściowo, zadanie to może być wykonywane tylko przez jednego pracownika.

-scenariusz określający czasową zajętość każdego konkretnego zadania

-zajętość czasowa K-tego zadania w tym scenariuszu

Dla uproszczenia problemu zakładamy, że każdy z pracowników wykonuje to samo zadanie w takim samym czasie określonym przez jego zajętość czasową

# Proponowane przez nas rozwiązanie

Do rozwiązania tego problemu użyliśmy 3 opisanych poniżej heurystyk do wyznaczenia konkretnych decyzji oraz funkcji żalu, która spośród tychże decyzji wybierze według swoich kryteriów najlepszą.

## Pierwsza heurystyka -algorytm zachłanny

Pierwszą heurystyką, która zwraca nam decyzję zbliżoną do optymalnej, jest zachłanne przydzielanie posortowanych według zajętości czasowych zadań do najmniej zapracowanych pracowników.

Pseudokod tej heurystyki:

1. Zdefiniuj funkcję greedy przyjmującą 2 argumenty: scenariusz , ilość pracowników

2. Inicjalizuj pustą listę decyzja\_index

3. Inicjalizuj pustą listę li

4. Dla każdego elementu i w zakresie od 0 do długości scenariusza Dodaj parę [czas trwania zadania, indeks zadania] do listy li

6. Posortuj listę li rosnąco

7. Dla każdego elementu i w zakresie od 0 do długości scenariusza:

8. Jeśli decyzja\_index jest pusta:

9. Dodaj element li[i] oraz indeks pracownika [0] do listy decyzja\_index

10. W przeciwnym przypadku:

11. Dodaj element li[i] oraz indeks pracownika o aktualnie najmniejszej zajętości czasowej do listy decyzja\_index

12. Inicjalizuj pustą listę decyzja

13. Posortuj tablicę decyzja\_index rosnąco według drugiego elementu (indeksu zadania)

14. Dla każdego pracownika i w zakresie od 0 do długości decyzja\_index:

15. Dodaj indeks przypisanego zadania do listy decyzja

16. Zwróć listę decyzja

## Druga heurystyka-stan początkowy dla algorytmu symulowanego wyżarzania

Drugą heurystyką jest algorytm generowania początkowego stanu rozwiązania, który jest wykorzystywany do rozpoczęcia procesu optymalizacji za pomocą algorytmu symulowanego wyżarzania.

Pseudokod:

1. Zdefiniuj funkcję seed\_generation przyjmującą trzy argumenty: scenariusz , ilość pracowników, jakąś możliwą decyzję(w tym przypadku wynik pierwszej heurystyki)

2. Inicjalizuj PMS (previous makespan) na bardzo dużą wartość, na przykład 9999999999

3. Inicjalizuj MS (makespan) na 0

4. Dopóki MS < PMS:

5. Znajdź pracownika o najdłuższym czasie wykonywania zadań, zapisz jego indeks jako R

6. Oblicz MS jako makespan dla aktualnej decyzji

7. Przypisz PMS jako MS

8. Inicjalizuj pustą listę lzwR zadań pracownika o indeksie R

9. Dla każdego elementu i w zakresie od 0 do długości decyzja:

10. Skopiuj decyzja do d\_temp

11. Jeśli decyzja[i] != R:

12. Dla każdego elementu j w lzwR:

13. Zamień ze sobą zadania d\_temp[i] z d\_temp[j]

14. Oblicz new\_MS jako jako makespan po zamienieniu ze sobą zadań

15. Jeśli new\_MS < MS:

16. Przypisz d\_temp do decyzja

17. Przypisz new\_MS do MS

18. Zwróć listę decyzja

## Trzecia heurystyka-algorytm symulowanego wyżarzania

Trzecią heurystyką jest algorytm symulowanego wyżarzania.

Pseudokod:

1. Zdefiniuj funkcję annealing przyjmującą pięć argumentów: scenariusz , ilość pracowników, temperatura T=60, współczynnik odpowiadający za zmianę temperatury r=0.85, delta=0.01:

2. Inicjalizuj S0 jako wynik drugiej heurystyki

3. Dopóki T > delta:

4. Inicjalizuj pustą listę p\_list

5. Dla każdego pracownika i w zakresie od 0 do pracownicy:

6. Dodaj i do listy p\_list

7. Wygeneruj R jako liczbę losową z przedziału (0, 0.99)

8. Jeśli R <= 0.49:

9. losowo zamień zadania z dwóch różnych maszyn, zapisz jako S1

10. W przeciwnym razie:

11. dokonaj transferu losowego zadania z jednej maszyny do innej losowo wybranej,, zapisz jako S1

12. Oblicz d jako makespan(S0) – makespan(S1)

13. Jeśli d > 0:

14. Przypisz S0 jako S1

15. W przeciwnym razie:

16. Wygeneruj R jako liczbę losową z przedziału (0, 0.99)

17. Jeśli R < :

18. Przypisz S0 jako S1

19. Przypisz T jako r \* T

20. Jeśli T=<delta

21. Znajdź końcowe optimum lokalne za pomocą drugiej heurystyki z decyzji S0

22. Zwróć decyzję

## Funkcja żalu

Funkcja ta ma na wyjściu podać o ile najlepsza decyzja jest lepsza od wybranej przez nas decyzji w najgorszym scenariuszu dla tej decyzji. Jako najlepszą decyzję wybraliśmy dokładnie równe rozłożenie prac dla pracowników, niezależnie od tego, czy jest to możliwe. Mianowicie zsumowaliśmy czas wszystkich zadań w tym scenariuszu i podzieliliśmy go przez ilość pracowników. To jest wynik decyzji, która nie zawsze jest możliwa, ale daje najlepszy wynik, który zapewni nam, że funkcja żalu nie zwróci nam ujemnego żalu. (Nie da się zadań podzielić lepiej niż idealnie po równo)

Przyjęliśmy również kolejną modyfikację. Mianowicie nie sprawdzamy każdego możliwego scenariusza dla naszej decyzji by stwierdzić, który jest najgorszy poprzez przegląd zupełny. Z góry wiemy, że potencjalnie najgorszymi scenariuszami są takie, kiedy jeden pracownik jest maksymalnie obciążony (zajętości czasowe przypisanych mu zadań mają wartości maksymalne), a reszta pracowników minimalnie (zajętości czasowe przypisanych im zadań mają wartości minimalne). Dlatego przeszukujemy tylko tyle scenariuszy, ile jest pracowników, co znacznie przyspiesza działanie kodu.

## Rozwiązanie

Jako rozwiązanie problemu przyjmujemy decyzję, dla której funkcja żalu ma wartość minimalną. Czyli dla tej decyzji w najgorszym scenariuszu najlepsza decyzja jest minimalnie od niego lepsza.

# Wyniki i wnioski

Prezentacja wyników

Przeprowadziliśmy symulację działania naszej metody podejmowania decyzji na 10 i 50 losowo wygenerowanych zadań, z których każde ma losowo określoną zajętość czasową. Dla każdej heurystyki znaleźliśmy decyzje podjęte na podstawie trzech scenariuszy:

- w przypadku gdy wszystkie zadania przyjmują najmniejszą możliwą zajętość czasową,

- w przypadku gdy wszystkie zadania przyjmują największą możliwą zajętość czasową,

- w przypadku gdy wszystkie zadania przyjmują zajętość czasową dokładnie pomiędzy najmniejszą i największą.

Tak przedstawia się wykres wartości funkcji żalu od podjętej za pomocą heurystyki decyzji dla 10 losowo wygenerowanych zadań:

Obraz zawierający tekst, zrzut ekranu, Czcionka, Wykres

Opis wygenerowany automatycznie

Decyzja o najmniejszej wartości funkcji żalu:

(7.666666666666668, [0, 0, 1, 1, 2, 0, 1, 2, 2, 0])

Tak przedstawia się wykres wartości funkcji żalu od podjętej za pomocą heurystyki decyzji dla 50 losowo wygenerowanych zadań:

Obraz zawierający tekst, zrzut ekranu, Prostokąt, Czcionka

Opis wygenerowany automatycznie Decyzja o najmniejszej wartości funkcji żalu:

(26.33333333333333, [2.0, 0.0, 0.0, 1.0, 2.0, 0.0, 1.0, 0.0, 1.0, 2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 2.0, 2.0, 1.0, 0.0, 1.0, 0.0, 2.0, 2.0, 1.0, 2.0, 1.0, 0.0, 0.0, 2.0, 2.0, 1.0, 0.0, 0.0, 2.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 2.0, 1.0, 1.0, 1.0, 2.0, 2.0])

Wnioski

Na podstawie przeprowadzonych przez nas badań możemy stwierdzić, że dal małej liczby zadań algorytm zachłanny osiąga najlepsze wyniki (oczywiście na wyniki ma wpływ losowość, jednak dla małej liczby zadań uruchamialiśmy program kilka razy i za każdym razem metoda zachłanna działała najlepiej). W przypadku większej liczby zadań heurystyki bazowane na symulowanym wyżarzaniu dorównywały, a często przebijały wyniki uzyskane za pomocą algorytmu zachłannego.

Kod źródłowy:

<https://github.com/IgorLewandowski/projekt_systemy_inteligentne/blob/main/kod.ipynb>