PTSZ - Zadanie 1 - Problem $1|r_i|\Sigma w_i U_i$

Dariusz Max Adamski 136674 (grupa I9, godzina 8:00)

dariusz.adamski@student.put.poznan.pl

Data oddania: 28 lutego 2021

Wstęp

W tym sprawozdaniu opisane jest podejście rozwiązujące problem szeregowania zadań na jednej maszynie, z czasami startowymi i minimalizacją ważonej liczby zadań spóźnionych.

1 Generator instancji

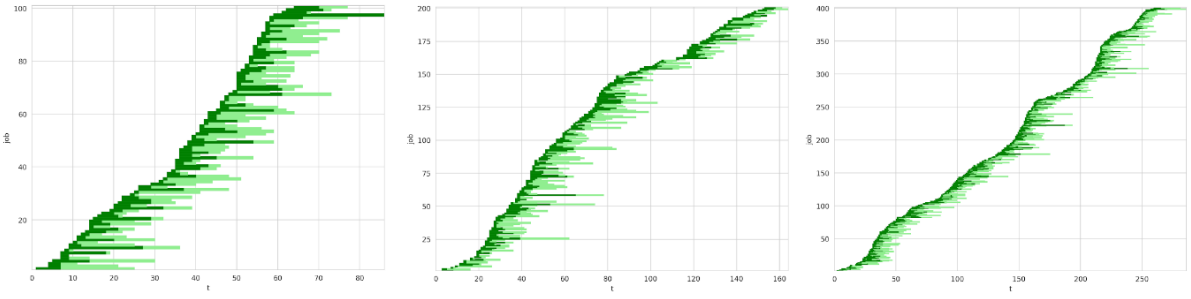
Algorithm 1: Algorytm generatora instancji dla problemu $1|r_i|\Sigma w_i U_i$

$\mu_p = 5; \mu_s = 7; \sigma_w = 2$
$t = \mu_p$
for $i \in 1 \dots n$ **do**
  $r_i := \max\{0, \lfloor Normal(t, \mu_p) \rfloor\}$
  $p_i := 1 + NegativeBinomial(\mu_p, 2)$
  $d_i := r_i + p_i + NegativeBinomial(\mu_s, 2)$
  $w_i := \max\{1, \lfloor Normal(p(\frac{p}{d-r} + 0.5), \sigma_w) \rfloor\}$
  if $Bernoulli(step) = 1$ **then**
    $step := \max\{0.05, Beta(2, 10)\}$
    $t := t + Binomial(\mu_p + \mu_s, 0.5)$
Potasuj losowo zadania

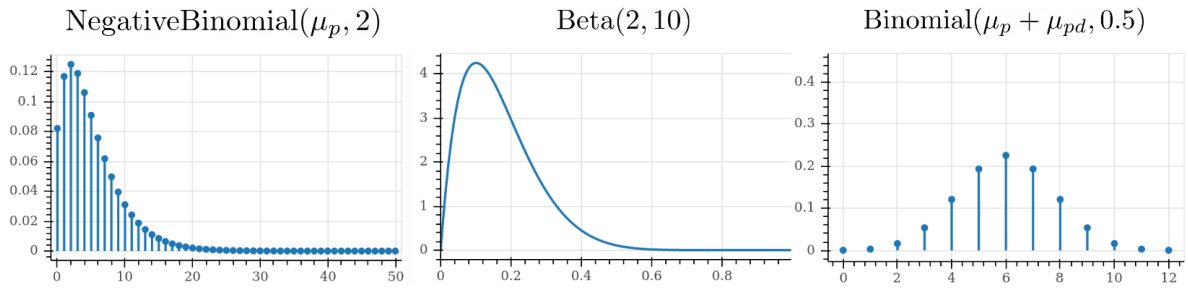
1.1 Opis algorytmu

Dla każdego zadania czas gotowości r_i jest losowany z rozkładu normalnego wyśrodkowanego na aktualnym czasie symulacji, z odchyleniem standardowym równym średniemu czasowi przetwarzania μ_p, przy czym czas gotowości nie może być mniejszy od 0, a wartość jest zaokrąglana

Rysunek 1: Wizualizacje wygenerowanych instancji o wielkości 100, 200 i 400



Rysunek 2: Rozkłady prawdopodobieństwa użyte w algorytmie generatora



funkcją podłoga. W efekcie, nawet jeśli symulacja jest w czasie t, to czas gotowości może przypaść na wcześniejszy moment, przez co zadanie może kolidować z innym wcześniejszym.

Czas przetwarzania p_i jest losowany z dyskretnego rozkładu Pascala ze średnią wartością μ_p i eksperymantalnie dobranym parametrem φ, równym 2. Do tej wartości dodawane jest jeszcze 1, żeby minimalną wartością czasu przetwarzania było 1, a nie 0. Podobnie losowany jest oczekiwany czas zakończenia d_i, przy czym minimalną wartością jest $r_i + p_i$, a do tego jest dodawany luz, który jest losowany z rozkładu Pascala ze średnią wartością μ_s i φ równym 2. Pierwszy z rozkładów jest zwizualizowany na rysunku 2. Wynika z niego, że po dodaniu 1, najczęstszym czasem przetwarzania jest 3, często są też zadania o długości 1 i 2. Większość zadań ma czas przetwarzania mniejszy od 10, ale mogą zdarzyć się nawet zadania o długości 20 lub nawet większej, chociaż prawdopodobieństwo tego jest niskie.

Waga zadania w_i jest losowana z rozkładu normalnego o średniej $p(0.5+p/(d-r))$ z odchyleniem standardowym σ_w. Chciałem, żeby waga była proporcjonalna do czasu przetwarzania, ale żeby zadania, których nie można „przesuwać" (te dla których $p/(d-r)$ jest bliżej 1) miały większą wagę (odwrotnie proporcjonalnie do luzów).

Ciekawą według mnie częścią algorytmu jest to, że czas symulacji nie jest zwiększany jednolicie, ale z pewnym prawdopodobieństwem *step*. Po każdym zadaniu, jeśli wartość próby Bernoulliego wyniesie 1, to czas symulacji jest zwiększany o krok losowany z rozkładu dwumianowego, ze średnią wartością wartością równą $\mu_p + \mu_s$ (trzeci wykres na rysunku 1). Następnie ponownie losowane jest prawdopodobieństwo zwiększenia czasu symulacji z rozkładu Beta o parametrach 2 i 10, przy czym minimalne prawdopodobieństwo ma wynosić 5%. Rozkład Beta został dobrany eksperymentalnie tak, żeby większość prawdopodobieństw była mniejsza od 0.3, ale żeby czasami mogły być też większe. Takie zachowanie skutkuje wygięciami instancji (widocznymi na rysunku 1), które mają symulować momenty, w których na maszynę nie wpływa wiele nowych zadań.

2 Algorytm szeregowania

2.1 Oznaczenia

Zbiór T początkowo zawiera wszystkie zadania $T_1 \dots T_n$. Gdy zadanie jest dodane do uszeregowania, usuwane jest z tego zbioru.

Zmienna *assigned* zawiera liczbę zaplanowanych zadań w danej iteracji. Jeśli pod koniec iteracji ta zmienna jest równa zero, to z algorytmu wynika, że nie można już zaplanować więcej zadań, tak żeby wykonały się przed „due date".

Zmienna t oznacza aktualny czas symulacji.

Zbiór T_{valid} zawiera zadania T_j, które mogą być przypisane w chwili t, lub późniejszej, tak aby

Algorithm 2: Algorytm dla problemu $1|r_i|\Sigma w_i U_i$

```
while T ≠ ∅ do
    assigned := 0
    t := 0
    while t < max d do
        T_valid := {T_j ∈ T | d_j − t ≥ p_j ∧ ¬scheduled(max{t, r_j}, p_j)}
        if T_valid = ∅ then
            t := max{t + 1, min_{T_j ∈ T} r_j}
            continue
        T_best := T_valid [argmax_{T_j ∈ T_valid} w_j / (d_j − t)]
        t_1 := max{t, r_best}
        t_2 := t_1 + p_best
        Zaplanuj zadanie T_best w czasie t_1 ... t_2
        T := T \ {T_best}
        t := t_2
        assigned := assigned + 1
    if assigned = 0 then
        break
Zaplanuj pozostałe zadania z T w dowolnej kolejności
```

zakończyły się przed due date d_j i nie kolidowały z innymi zaplanowanymi zadaniami.

Predykat scheduled(t, p) przyjmuje wartość prawdziwą, jeśli jest już zaplanowane jakieś zadanie w przedziale $t \dots t + p$ (lub jego podzbiorze). W przeciwnym wypadku predykat przyjmuje wartość fałszywą.

2.2 Opis algorytmu

Działanie algorytmu szeregowania jest bardzo proste. Dla danego punktu w czasie, według heurystyki wybierane jest „najlepsze" zadanie, które nie koliduje z innymi już zaplanowanymi zadaniami i może się zakończyć przed due date d_i. Jeśli $t < r_i$, to czas jest ustawiany na r_i, a następnie robiony jest krok czasowy równy czasowi wykonania tego zadania p_i. Gdy czas symulacji jest równy lub większy od największego due date, algorytm jest powtarzany, żeby zapełnić wolne miejsca w uszeregowaniu, które powstały w wyniku planowania zadań z czasem gotowości r_i większym od aktualnego czasu symulacji t.

Według wybranej heurystyki najlepsze zadanie ma największą wartość wyrażenia $w_i/(d_i - t)$, co oznacza, że w pierwszej kolejności wybierane są zadania z największą wagą i najkrótszym czasem do due date od aktualnego czasu symulacji.

2.3 Analiza złożoności

Złożoność algorytmu to $O(n^2)$, gdzie n to wielkość instacji - liczba zadań. Złożoność wynika z tego, że zachłannie planujemy n zadań, a przy planowaniu każdego zadania rozważamy maksymalnie n zadań kandydatów, aby skonstruować zbiór T_{best}, plus dla każdego kandydata obliczamy w n krokach heurystykę (co nie wpływa na złożoność). Jeśli zbiór kandydatów jest pusty, to ustawiamy czas na aktualny $t + 1$, albo najbliższy czas gotowości r_i, w zależności co jest większe, dzięki czemu złożoność algorytmu nie zależy od czasów r_i, d_i, p_i.