Karta główna

# Wstęp

Jednym z najważniejszych aspektów w dobie wszechobecnego kryzysu jest optymalizacja procesów produkcyjnych. Odpowiedni harmonogram prac to często być albo nie być dla wielu przedsiębiorstw. Tworząc taki harmonogram należy zwrócić uwagę na możliwość optymalnego wykorzystanie zasobów, tak aby specjalistyczne maszyny i urządzenia nie stały bezczynnie, a wykorzystywały swój potencjał w jak największym czasie. Tego typu problemy pozwalają rozwiązać algorytmy szeregowania zadań.

# Problem szeregowania zadań

W przypadku planowania wykonania pewnych czynności (zadań) w momencie korzystania z ograniczonej liczby zasobów (procesorów) można stwierdzić, że kolejność wykonanych zadań, ich uszeregowanie, pozwala na osiągnięcie lepszych lub gorszych czasów realizacji całego przedsięwzięcia. Przykładowymi sytuacjami w których takie szeregowanie ma istotne znaczenie są:

* określenie kolejności zadań, obliczeń w wykonywanym algorytmie programu komputerowego. W tym wypadku zadaniem jest krok algorytmu, a zasobem czas procesora
* stworzenie harmonogramu prac inwestycji budowlanej. Zasobami są ludzie, sprzęt budowlany, a szeregowaniu podlegają prace budowlane takie jak np. położenie powierzchni asfaltowej
* zdefiniowanie modelu produkcyjnego pewnych dóbr złożonych z wielu podzespołów. Zasobem może być tu praca ludzka, dostęp do urządzeń koniecznych do wykonania podzespołów, uszeregowaniu podlega tworzenie kolejnych podzespołów

W zależności od warunków istnieje wiele algorytmów, które pozwalają na znalezienie optymalnego rozdzielenia zadań na zasoby [1]. Pierwszego podziału można dokonać na dwa typy szeregowania: dynamiczne i statyczne. Z szeregowaniem dynamicznym najczęściej można się spotkać przy rozdzielaniu czasu procesora pomiędzy procesy, gdy nie są znane czasy zakończenia zadania przed jego uruchomieniem. Najczęściej stosowanymi algorytmami przy tym szeregowaniu są: FIFO. Z kolei szeregowanie statyczne pojawia się, gdy znany jest czas wykonania wszystkich zadań, znane są także zasoby i koszty przesyłania zadań. Poniższa praca skupia się na problemie statycznego szeregowania.

Następnego podziału można dokonać ze względu na liczbę zasobów potrzebnych do wykonania czynności. Ich liczba może być ograniczona lub nieskończona. Drugi przypadek jest rzadko spotykany w rzeczywistej sytuacji, jednak najistotniejsze w tym przypadku byłoby znalezienie najkrótszego czasu szeregowania biorąc pod uwagę wyłącznie zależności pomiędzy czynnościami oraz koszt transmisji danych pomiędzy zasobami. Praca rozważa problem ograniczonych zasobów, w którym wielokrotnie wiele zadań mogłoby być wykonanych równocześnie, jednak ze względu na brak wolnych zasobów nie jest to możliwe.

Stworzono algorytmy, które można wykorzystać w sytuacji, dla której czasy wykonania wszystkich procesów są stałe. Nie jest to często spotykane w rzeczywistych problemach, jednak wartym podkreślenia faktem jest możliwość rozwiązania takiego problemu przy dodatkowych warunkach jak problemu klasy P-zupełnego. W poniższej pracy czas wykonywania poszczególnych zadań nie jest obłożony tym ograniczeniem, istotne jest, że nie może być ujemny lub zerowy.

Kosztem przesyłania danych pomiędzy zasobami określa się czas jaki jest konieczny na transmisję danych, produktów wykonanego zadania do zasobu, na którym zostanie wykonane kolejne zadanie. Koszt musi zostać uwzględniony wyłącznie jeśli zadania są zależne. Częstym założeniem, istotnym w tej pracy, jest zerowy koszt w przypadku, gdy zadania były wykonywane na tym samym procesorze, zasobie. Z praktycznego punktu widzenia jeśli informacja może zostać zachowana w danym zasobie bez kosztu jej składowania nie ma potrzeby wykonywania dodatkowych czynności.

Rodzaje algorytmów szeregowania możemy podzielić również ze względu na uwzględnianie kosztu przy szeregowaniu (np. w systemach z pamięcią …) lub jego ignorowanie (np. w systemach z pamięcią współdzieloną). Także istotne jest czy przesyłanie może odbywać się równolegle czy musi być ono kolejkowane.

Zadania, które podlegają szeregowania, można podzielić na przerywalne i nieprzerywalne. Pierwsze z wymienionych dotyczy przypadku, gdy wykonywaną czynność można przerwać w dowolnym momencie, następnie można ją wznowić na innym procesorze. Z teoretycznego punktu widzenia powinno to pozwolić osiągnąć lepsze czasy szeregowania, jednak należy uwzględnić dodatkowe koszty związane z przesłaniem podzielonego zadania, które mogą być niewspółmierne do zysku związanego ze wcześniejszym zakończeniem innych zadań. Koszty nie mogą być również obliczone na starcie szeregowania, a dopiero w momencie przenoszenia podzielonego zadania na inny zasób. Pamiętać należy także o tym, iż nie w każdych warunkach jest możliwe podzielenie zadania na części. Jest to także bardzo trudny przypadek do implementacji.

Udowodniono, że problem szeregowania jest problemem NP-zupełnym, z wyjątkiem trzech sytuacji:

* przy drzewiastej strukturze grafu zależności zadań z arbitralną wartością procesorów
* graf procesów z jednakowym czasem wykonania zadań na dwóch procesorach
* graf typu interval-order (*interval-ordered*) z jednakowym czasem wykonania zadań i stałą liczbą procesorów.

Oznacza to, że nie jest możliwe szybkie znalezienie najlepszego rozwiązania i głównie używa się algorytmów heurystycznych, które próbują znaleźć optimum we względnie krótkim czasie, nie gwarantując jednak jego odnalezienia.

## Graf programu równoległego

W przypadku szeregowania statycznego z uwzględnieniem kosztów przesłania danych najefektywniejszym i najczytelniejszym sposobem prezentacji zależności pomiędzy zadaniami jest użycie acyklicznego grafu skierowanego. Przykład takiego grafu jest widoczny na Rys. 2.1. Węzeł grafu symbolizuje proces, zaś krawędź zależność pomiędzy zadaniami. W węźle widoczne są dwie wartości: numer procesu oraz czas wykonania (wyróżniany w komórce z szarym tłem). Zadanie numer 0 wykona się w czasie 3 jednostek, z kolei 2-gie zadanie w czasie 4 jednostek. Krawędzie są skierowane i krawędź prowadząca od węzła nr 0 do węzła nr 2 oznacza, że możliwe jest wykonanie zadania nr 2 dopiero, gdy zakończy się zadanie 0. Wartość przy krawędzi oznacza koszt przesłania danych pomiędzy modułami. Należy zwrócić uwagę, że koszt przesłania liczymy według wzoru:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

gdzie: tkl – czas komunikacji między modułami k i l

dkl – minimalna odległość pomiędzy zasobami wykorzystywanymi przez moduły k i l, odległość definiuje się według grafu procesów opisanego w rozdziale X

akl – waga krawędzi pomiędzy węzłami k i l

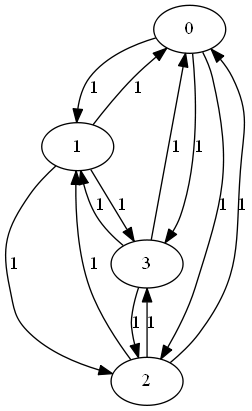
Jeśli procesy są alokowane na tym samym procesorze, odległość dkl wynosi 0 i koszt także jest równy 0. Z kolei, gdy procesory nie są połączone bezpośrednio należy zsumować odległość pomiędzy nimi uwzględniając koszt komunikacji (może być różny pomiędzy procesorami). W pracy założono, że czas wykonywania poszczególnych modułów na wszystkich procesorach jest identyczny.



Rys. . Przykładowy graf procesów rozpatrywany w ramach problemu szeregowania

## Graf procesorów

Zasoby można podobnie jak zadania przedstawić w formie grafu. Rys. 2.2 pokazuje przykładowy graf cykliczny dla 4 procesorów. Wszystkie procesory są ze sobą połączone co oznacza, że można wymieniać dane pomiędzy każdym z nich. Koszt w każdym wypadku wynosi 1 (wartość przy krawędzie).



Rys. . Graf przedstawiający powiązanie procesorów wraz z kosztem zmiany procesora

Liczba procesorów jest nieograniczona, przy czym minimalną logiczną wartością są 2 procesory. Także koszty komunikacji mogą być różne i zależne od przypadku. Na potrzebny poniższej pracy skorzystano z 2, 4 i 8 procesorów z założeniem, że komunikacji może odbywać się na zasadzie każdy z każdym, a koszt transmisji danych wynosi 1.

# Algorytm uogólnionej ekstremalnej optymalizacji

W celu rozwiązania problemów NP-zupełnych często stosuje się algorytmy heurystyczne i metaheurystyczne. Jednym z algorytmów metaheurystycznych jest algorytm uogólnione ekstremalnej optymalizacji (*generalized extremal optimization*), który wykorzystano w poniższej pracy. Ma on swoje zastosowania w problemach optymalizacji, pozwalając na dobranie najlepszych parametrów optymalizowanej funkcji.

# Wyniki eksperymentów

## Konfiguracje procesorów

Eksperymenty zostały przeprowadzone na trzech konfiguracjach procesorów. Każdą z nich można przedstawić za pomocą grafu pełnego, gdzie każdy węzeł łączy się z pozostałymi. Waga każdej z krawędzi ma wartość 1, co oznacza na podstawie wzoru (1), że dane z jednego procesora można przenieść na inny z kosztem równym akl. Taka konfiguracja powoduje, że połączenia procesorów nie mają wpływu na wyniki działania algorytmu szeregowania.



Rys. . Graf procesorów FULL2



Rys. . Graf procesorów FULL4

## Rozkład prawdopodobieństwa mutacji w przypadkach testowych

Przeprowadzając testy uruchomiono 1000 razy algorytm GEO do szeregowania na grafie intree15 w następujących konfiguracjach parametru prawdopodobieństwa τ: 0.1, 0.2, 0.5, 0.8, 1.0, 1.5, 2.0, 5.0 i 8.0. Wszystkie uruchomienia wykonały 100 iteracji.

Tabela 1 Prawdopodobieństwo wyboru konfiguracji do dalszej analizy w zależności od parametru prawdopodobieństwa

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0,1** | **0,2** | **0,5** | **0,8** | **1** | **1,5** | **2** | **5** | **8** |
| **1** | 1,0000 | 1,0000 | 1,0000 | 1,0000 | 1,0000 | 1,0000 | 1,0000 | 1,0000 | 1,0000 |
| **2** | 0,9330 | 0,8706 | 0,7071 | 0,5743 | 0,5000 | 0,3536 | 0,2500 | 0,0313 | 0,0039 |
| **3** | 0,8960 | 0,8027 | 0,5774 | 0,4152 | 0,3333 | 0,1925 | 0,1111 | 0,0041 | 0,0002 |
| **4** | 0,8706 | 0,7579 | 0,5000 | 0,3299 | 0,2500 | 0,1250 | 0,0625 | 0,0010 | 0,0000 |
| **5** | 0,8513 | 0,7248 | 0,4472 | 0,2759 | 0,2000 | 0,0894 | 0,0400 | 0,0003 | 0,0000 |
| **6** | 0,8360 | 0,6988 | 0,4082 | 0,2385 | 0,1667 | 0,0680 | 0,0278 | 0,0001 | 0,0000 |
| **7** | 0,8232 | 0,6776 | 0,3780 | 0,2108 | 0,1429 | 0,0540 | 0,0204 | 0,0001 | 0,0000 |
| **8** | 0,8123 | 0,6598 | 0,3536 | 0,1895 | 0,1250 | 0,0442 | 0,0156 | 0,0000 | 0,0000 |
| **9** | 0,8027 | 0,6444 | 0,3333 | 0,1724 | 0,1111 | 0,0370 | 0,0123 | 0,0000 | 0,0000 |
| **10** | 0,7943 | 0,6310 | 0,3162 | 0,1585 | 0,1000 | 0,0316 | 0,0100 | 0,0000 | 0,0000 |
| **11** | 0,7868 | 0,6190 | 0,3015 | 0,1469 | 0,0909 | 0,0274 | 0,0083 | 0,0000 | 0,0000 |
| **12** | 0,7800 | 0,6084 | 0,2887 | 0,1370 | 0,0833 | 0,0241 | 0,0069 | 0,0000 | 0,0000 |
| **13** | 0,7738 | 0,5987 | 0,2774 | 0,1285 | 0,0769 | 0,0213 | 0,0059 | 0,0000 | 0,0000 |
| **14** | 0,7680 | 0,5899 | 0,2673 | 0,1211 | 0,0714 | 0,0191 | 0,0051 | 0,0000 | 0,0000 |
| **15** | 0,7628 | 0,5818 | 0,2582 | 0,1146 | 0,0667 | 0,0172 | 0,0044 | 0,0000 | 0,0000 |

W Tabeli 1 przedstawiono prawdopodobieństwo z jakim może zostać wybrana konfiguracja podziału zadań na procesorach do następnej iteracji jako wartość bazowa. Oznacza to również szansę, że wybrana konfiguracja zostanie wyselekcjonowana do porównania z aktualnie najlepszym ustawieniem i może zastąpić je w przypadku, gdy czas szeregowania będzie lepszy.

Istotne jest, że szansa wybrania danej konfiguracji pojawia się dopiero w sytuacji gdy wcześniej podana konfiguracja zostanie wylosowana. Rozkład prawdopodobieństwa jest w tym wypadku stały i wynosi *1/k*, gdzie *k* jest ilością konfiguracji. Oznacza to, że przykładowo gdy wartość parametru *τ*=1, a ilość konfiguracji wynosi 15, to szansa że do następnej konfiguracji przejdzie konfiguracja numer 1 wynosi 1/15 \* 1, z kolei szansa wyboru konfiguracji numer 10 wynosi 1/15 \* 1/10 = 1/150. Istnieje także niezerowa szansa, że w losowaniu nie zostanie wybrana konfiguracji i będzie należało ponowić wybór. Taka sytuacja będzie miała miejsce w znacznej większości przypadków gdy τ=8.

Bardzo ważną zależnością widoczną w tabeli jest fakt zmniejszania szansy na wybór gorszych konfiguracji (im wyższy numer tym gorsze ustawienie) wraz ze wzrostem wartości parametru τ. W przypadku τ=8 szansa, że zostanie wybrane ustawienie inne od najlepszego wynosi < 0.1% i w przypadku wykonania 100 iteracji prawdopodobnie nie zostanie wybrana inna konfiguracja niż najlepsza do dalszego przetwarzania. Można to zaobserwować na wykresie 1. Z kolei na wykresie 2 widoczna jest skokowość przebiegu algorytmu i częsty wybór gorszych konfiguracji.

## Proste grafy drzewiaste

Grafy procesów tree15 (Rys. 4.4) oraz intree15 (Rys. 4.3) są jednymi z najprostszych do uszeregowania ze względu na swoją budowę, brak złożonych zależności, ten sam czas szeregowania wszystkich procesów i równy koszt zmiany procesora. Oba grafy mają najlepszy czas szeregowania *T* = 9 w przypadku użycia dwóch procesorów w konfiguracji FULL2. Wykonując testy na obu grafach wykonano 1000 uruchomień algorytmu dla każdego z grafów. Limit iteracji był równy 100 co oznaczało, że liczba ewolucji, wykonania funkcji przystosowania (*fitness*), wyniosła 1500 (100 iteracji \* 15 mutujących konfiguracji na iterację).



Rys. . Graf procesów intree15

W Tabeli 1 przestawiono dane statystyczne po wykonaniu eksperymentów na grafie intree15. Dane pokazują zależność pomiędzy ilością iteracji potrzebną do znalezienia optymalnego rozwiązania a parametrem prawdopodobieństwa τ. Jak pokazuje kolumna średnia *T* i min *T*, które oznaczają odpowiednio średni i najlepszy czas szeregowania, w obu przypadkach wartość wynosi 9 i jest to najlepszy czas uszeregowania dla tego grafu. Spoglądając na średnią i wariancję liczby iteracji, potrzebnych do znalezienia optymalnego rozwiązania, można zauważyć że najlepsze wyniki otrzymano, gdy wartość τ wynosiła 5. Przy tej wartości parametru średni wyniosła 1.474 i jest znacznie lepsza od średniej dla wartości parametru równej 0.1 i wynoszącej 4.322. W tym przypadku także wariancja potwierdza, że różnica między potrzebnymi iteracjami jest nieznaczna. Wartość mediany, kwartylu górnego oraz maksymalnej liczby iteracji potwierdzają te obserwacje. Porównując wartości τ równego 0.1 i 5.0 mamy odpowiednio:

* medianę równą 3 i 1, co oznacza, że w 50% uruchomień potrzebne były co najmniej 3 iteracje w przypadku τ=0.1 i tylko 1 w przypadku τ=5.0
* kwartyle górne równe 6 i 2, jest to duża różnica jeśli chodzi o ilość potrzebnych iteracji
* maksymalną liczbę iteracji równą 42 i 6.

W przypadku dominanty sytuacja prawie we wszystkich przypadkach wygląda podobnie. Najczęściej potrzeba tylko jednej iteracji do znalezienia optymalnego rozwiązania. Odstępstwem jest uruchomienie algorytmu z parametrem τ=0.1. Tutaj większe prawdopodobieństwo ma znalezienie optimum przy losowaniu bazowej konfiguracji (wynosi 21.1%), niż po pierwszej iteracji (wynosi ono 17%). Dla τ=5.0 prawdopodobieństwo znalezienie najlepszego rozwiązania w 1 iteracji wynosi 38.1% a łącznie z szansą wylosowania w zerowej iteracji osiąga 56.5%, dla τ=8.0 wartości te to odpowiednio 40.5% oraz 58.2%. Mimo dużej rozbieżności w wynikach dla wszystkich konfiguracji udało się osiągnąć najlepszy rezultat w 100 iteracjach.

Analizując powyższe dane nasuwa się wniosek, że najlepszą wartością parametru τ dla grafu intree15 jest 5.0 i 8.0. Powodem jest to, ze graf intree15:

* posiada proste zależności procesów,
* koszt zmiany procesora jest niewielki i wynosi 1.

Te czynniki powodują, że istnieje znaczny procent rozwiązań, które dają najlepszy czas szeregowania. Przejście z gorszej do najlepszej konfiguracji wymaga wykonania tylko kilku kroków co oznacza konieczność zmieniania tylko 2-3 zadań na procesorze. Nasuwa się stwierdzenie, że w przypadku algorytmu GEO nie istnieje minimum lokalne wśród zbioru konfiguracji. To sprawia, że łatwiej jest znaleźć najlepsze rozwiązanie zawsze wybierając minimum w aktualnym zbiorze rozwiązań (τ=8.0), niż przeszukiwać wybierając losowo kolejny zbiór (τ=0.1). Druga ewentualność w przypadku tego grafu prowadzi do zbytniego skupienia się na ominięciu minimum lokalnego, które nie istnieje i próbie przeszukania rozwiązań wśród zbyt wielu konfiguracji.

Tabela Dane statystyczne testów dla grafu intree15 z 2 procesorami

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| τ | średnia iteracji | wariancja iteracji | min | Q1 | mediana | Q3 | max | moda | moda (%) | najlepsze wyniki (%) | śr. T | min T |
| 0.1 | 4,322 | 27,097 | 0 | 1 | 3 | 6 | 42 | 0 | 21,1 | 100 | 9 | 9 |
| 0.2 | 3,502 | 14,579 | 0 | 1 | 2 | 5 | 32 | 1 | 19,4 | 100 | 9 | 9 |
| 0.5 | 2,03 | 3,669 | 0 | 1 | 2 | 3 | 11 | 1 | 26,3 | 100 | 9 | 9 |
| 0.8 | 1,749 | 2,498 | 0 | 1 | 1 | 2 | 10 | 1 | 30,3 | 100 | 9 | 9 |
| 1.0 | 1,723 | 2,056 | 0 | 1 | 1 | 2 | 12 | 1 | 33,4 | 100 | 9 | 9 |
| 1.5 | 1,519 | 1,381 | 0 | 1 | 1 | 2 | 9 | 1 | 37 | 100 | 9 | 9 |
| 2.0 | 1,497 | 1,279 | 0 | 1 | 1 | 2 | 6 | 1 | 33,3 | 100 | 9 | 9 |
| 5.0 | 1,474 | 1,183 | 0 | 1 | 1 | 2 | 6 | 1 | 38,1 | 100 | 9 | 9 |
| 8.0 | 1,483 | 1,247 | 0 | 1 | 1 | 2 | 6 | 1 | 40,5 | 100 | 9 | 9 |

Na wykresach 1 oraz 2 przedstawiono przebiegi algorytmu w zależności od parametry prawdopodobieństwa. W przypadku, gdy jego wartość jest mała (np. 0.1 na wykresie 2) można zauważyć, że nie zawsze wybierana jest do następnej iteracji najlepsza konfiguracja procesów. Jest to różnica między linią niebieską obrazującą najlepszy czas szeregowania w danej iteracji, a czerwoną – wartość czasu szeregowania dla wybranej (wylosowanej) konfiguracji. Druga zauważalna różnica, to odejście od najlepszej konfiguracji do innej, skrajnie najgorszej. Widoczne jest to przy odchyleniach linii czerwonej od zielonej. Taki przebieg pozwala na wyjście z minimum lokalnego, co może prowadzić do znalezienia najlepszej konfiguracji w następnych iteracjach.

Wykres Przebieg algorytmu dla grafu intree15 i τ=8.0

Wykres 2: Przebieg algorytmu dla grafu intree15 i τ=0.1

Warto też zauważyć, że nie dochodzi w przypadku tego grafu do nagłych zmian wybranej konfiguracji. Przy przejściu od lepszego czasu do gorszego czasu szeregowania i odwrotnie nie następuje skok o kilka wartości lecz o 1 lub 2 jednostki czasu. Ma to związek z typem grafu. Procesy są tutaj powiązane w linii prostej, nie ma skomplikowanych zależności, a koszty zmiany procesora są niewielkie. To skutkuje tym, że generowanie nowych konfiguracji nie powoduje powstania dużo słabszych, a jedynie niewielkie odchylenia. Z kolei na wykresie dla dużego prawdopodobieństwa (np. 8.0) brak jest jakichkolwiek odchyleń, wszystkie linie nakładają się na siebie. Przyczyny są tutaj dwie:

* nie wybrano konfiguracji o rankingu niższym niż 2 (wykorzystano dwie najlepsze w kolejności),
* graf procesów jest prosty, koszty zmiany procesu są niskie (wynoszą 1) i istnieje wiele konfiguracji, które dają najlepszy czas szeregowania. Przez to zmiana procesora na którym wykonane zostanie jedno zadanie, a pozostawienie pozostałych bez zmian często nie spowoduje zmiany czasu szeregowania.



Rys. . Graf procesów tree15

Wyniki przeprowadzonych testów na grafie tree15 są podobne do wyników otrzymanych na drzewie intree15. Podstawową przyczyną jest podobieństwo grafów, intree jest odwróceniem grafu tree. Tutaj także najlepsze wyniki pojawiają się przy wprowadzeniu parametru prawdopodobieństwa równego 5.0 lub 8.0. Jak pokazuje Tabela 2 wszystkie uruchomienia zakończyły się sukcesem i zawsze otrzymano najlepszy wynik wynoszący 9.

Tabela Dane statystyczne testów na grafie tree15 z 2 procesorami

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| τ | średnia iteracji | wariancja iteracji | min | Q1 | mediana | Q3 | max | moda | moda (%) | najlepsze wyniki (%) | śr. T | min T |
| 0.1 | 4,114 | 21,388 | 0 | 1 | 3 | 6 | 34 | 0 | 21 | 100 | 9 | 9 |
| 0.2 | 3,073 | 11,363 | 0 | 1 | 2 | 5 | 22 | 0 | 22 | 100 | 9 | 9 |
| 0.5 | 2,152 | 3,781 | 0 | 1 | 2 | 3 | 16 | 1 | 27,8 | 100 | 9 | 9 |
| 0.8 | 1,642 | 2,03 | 0 | 1 | 1 | 2 | 8 | 1 | 33,4 | 100 | 9 | 9 |
| 1.0 | 1,673 | 1,744 | 0 | 1 | 1 | 2 | 8 | 1 | 33,8 | 100 | 9 | 9 |
| 1.5 | 1,443 | 1,43 | 0 | 1 | 1 | 2 | 6 | 1 | 36,3 | 100 | 9 | 9 |
| 2.0 | 1,433 | 1,209 | 0 | 1 | 1 | 2 | 5 | 1 | 35,4 | 100 | 9 | 9 |
| 5.0 | 1,418 | 1,192 | 0 | 1 | 1 | 2 | 6 | 1 | 41,8 | 100 | 9 | 9 |
| 8.0 | 1,394 | 1,138 | 0 | 1 | 1 | 2 | 6 | 1 | 38 | 100 | 9 | 9 |

# Różne sposoby wyboru najlepszej ścieżki – u mnie przy obliczaniu czasu szeregowania – polityka szeregowania There are various ways to determine the priorities of nodes, such as HLF (Highest Level First) [Coffman 1976]; LP (Longest Path) [Coffman 1976]; LPT (Longest Processing Time) [Friesen 1987; Gonzalez, Jr. 1977]; and CP (Critical Path) [Graham et al. 1979]. Opisać wybrany szczegółowo

# Bibliografia

1. **Kwok Y. i Ahmad I.** Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. *ACM Computing Surveys.* Grudzień 1999, Tom 31, 4, strony 407-471.