

Algorytm dokładny dla $P3||L_{max}$

3 kwietnia 2006 roku

1 Sformułowanie problemu

Problem uszeregowania zadań, którego dotyczy sprawozdanie precyzyjnie opisywany, jako

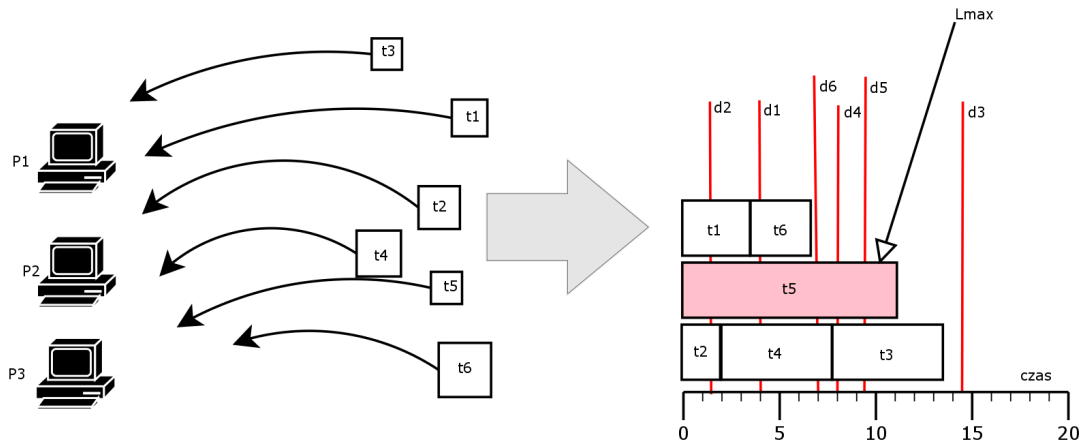
$$P3||L_{max} \quad (1)$$

polega na przydzieleniu zadań t trzem jednakowym procesorom P i uporządkowaniu ich w taki sposób, by maksymalne opóźnienie (L_{max}) dowolnego zadania było jak najmniejsze.

By móc określić opóźnienie l , każde zadanie charakteryzuje się czasem c , jaki jest potrzebny do jego wykonania i terminem d , na kiedy ma zostać wykonane.

Procesory są jednakowe i pracują równolegle, bez przerwy. Nie ma więc różnicy, na którym procesorze będzie wykonane dane zadanie. Również zadania nie są powiązane ze sobą zależnościami, a więc mogą być wykonane w dowolnej kolejności. Po rozpoczęciu zadanie nie może być przerwane.

Tak problem został opisany w poprzednim sprawozdaniu, omawiającym algorytm przybliżony, o złożoności wykładniczej, rozwiązujący ten problem.



2 Opis algorytmu

Algorytm bazuje na dotychczasowych dokonaniach. Jednym z narzędzi wykorzystanych w poprzednim zadaniu był program **wynik**, który znajdował rozwiązanie optymalne poprzez przeszukanie całej przestrzeni rozwiązań. Na nim się oparliśmy tworząc algorytm dokładny szeregowania **szereguj_dokl**.

Program w najgorszym wypadku tworzy wszystkie permutacje zadań i przydziela je kolejno do procesorów. Wykorzystano algorytm branch and bound, dzięki któremu nie są sprawdzane niektóre obszary przestrzeni rozwiązań, co do których jesteśmy pewni, że nie zawierają rozwiązania optymalnego. Wprowadzając kolejno Upper bound(UB) i Lower bound(LB), oraz poprawiając je, obserwowaliśmy poprawy szybkości w kolejnych wersjach.

Ogólnie algorytm działa tak, że funkcja permutująca otrzymuje dotychczasowe uszeregowanie części zadań i zbiór zadań, które nie zostały jeszcze uszeregowane. Każdy element ze zbioru zadań nie uszeregowanych, jest kolejno dołączany do uszeregowania. Dla każdego nowego uszeregowania (części, lub wszystkich elementów) obliczany jest L_{max} .

2.1 Wersja 1. B&B: Upper bound

Górne ograniczenie dotyczy wartości L_{max} . Dodatkowo przez cały czas pamiętany jest wynik najlepszego dotychczasowego uszeregowania, oznaczany $best_ever_L_{max}$. Dla najszybszego osiągnięcia najlepszych wyników potrzebne jest możliwie najlepsze początkowe uporządkowanie. Nie wykorzystano tutaj algorytmu przybliżonego (składającego się z sortowania i TabuSearch), a tylko jego pierwszy etap - szybkie sortowanie. Zrobiliśmy tak dlatego, że już samo szybkie sortowanie daje bardzo dobre, czasem optymalne, wyniki, a jest przy tym proste.

Tak więc na początku zadania są sortowane wg. $d-c$ (termin zakończenia - długość zadania), a wynik uszeregowania zapamiętywany jako najlepszy. W dalszej części algorytm wybiera kolejne elementy do nowej permutacji i za każdym razem oblicza L_{max} dla już wybranych elementów. Jeśli to L_{max} jest większe, lub równe $best_ever_L_{max}$, to znaczy, że tak budowane uszeregowanie nie będzie już lepsze od dotychczasowego najlepszego i wywoływany jest nawrót.

Nowe $best_ever_L_{max}$ zapamiętywane jest wtedy, gdy stworzone zostanie uszeregowanie wszystkich zadań z L_{max} mniejszym od dotychczasowego najlepszego.

Na tym etapie nie robiono jeszcze dokładnych testów efektywności algorytmu, bo do pełnego działania potrzebne jest jeszcze dolne ograniczenie. Sprawdzano jedynie poprawność wyników, poprzez porównanie z programem sprawdzającym wszystkie przypadki.

2.2 Wersja 2. B&B: Lower bound

Analiza poprawności górnego ograniczenia wykazała, że większość nawrotów wykonywana jest, gdy uszeregowano już prawie wszystkie elementy, a więc jest to stosunkowo mało skuteczne. Potrzebne było ograniczenie, które już dla sporego zbioru zadań było w stanie odpowiedzieć na pytanie, czy dodanie któregoś z zadań jeszcze nie uszeregowanych spowoduje przekroczenie L_{max} . Dolne ograniczenie działa więc tak, że dla zadań jeszcze nie uszeregowanych, obliczane jest opóźnienie z założeniem, że dane zadanie zostanie wykonane jako pierwsze. Wybieramy najgorsze z tych opóźnień i jeżeli jest ono większe od $best_ever_L_{max}$, to znaczy, że nie ma sensu permutować pozostałego zbioru nieuszeregowanych zadań. Zadanie, które miało najgorsze opóźnienie l byłoby w tych permutacjach napewno nie wcześniej, a więc mogłoby mieć jeszcze większe opóźnienie.

2.3 Wersja 3. B&B: Poprawa lower bound

Po zrobieniu części testów szybkości zauważono, że wywołując nawroty już nie tylko, gdy największe opóźnienie spośród pozostałych zadań jest większe od $best_ever_L_{max}$, ale gdy opóźnienia te są sobie równe, odrzucanych jest wiele przypadków, gdy różne uszeregowania zwracały ten sam L_{max} .

3 Wyniki

Dla każdej wersji poprawność wyników była testowana tak, że znajdowano rozwiązanie nową metodą dokładną i porównywano z rozwiązaniem podanym przez algorytm przybliżony. Jeśli wynik przybliżony był lepszy od dokładnego, to świadczyło o błędach nowego algorytmu. Dodatkowo sprawdzono także, kiedy wyniki algorytmu dokładnego są lepsze niż algorytmu przybliżonego – czyli kiedy warto używać algorytmu dokładnego:

```
tasks=8 c=50 cr=5
tasks=8 c=50 cr=10
tasks=8 c=50 cr=16
tasks=8 c=50 cr=50
tasks=8 c=100 cr=10
tasks=8 c=100 cr=20
tasks=8 c=100 cr=33
tasks=8 c=100 cr=100
tasks=8 c=200 cr=20
tasks=8 c=200 cr=40
tasks=8 c=200 cr=66
tasks=8 c=200 cr=200
tasks=8 c=500 cr=50
tasks=8 c=500 cr=100
tasks=8 c=500 cr=166
tasks=8 c=500 cr=500
tasks=9 c=50 cr=5
tasks=9 c=50 cr=10
tasks=9 c=50 cr=16
tasks=9 c=50 cr=50
tasks=9 c=100 cr=10
tasks=9 c=100 cr=20
tasks=9 c=100 cr=33
tasks=9 c=100 cr=100
tasks=9 c=200 cr=20
tasks=9 c=200 cr=40
tasks=9 c=200 cr=66
tasks=9 c=200 cr=200
tasks=9 c=500 cr=50
tasks=9 c=500 cr=100
tasks=9 c=500 cr=166
tasks=9 c=500 cr=500
```

Po lewej stronie tego wykresu przedstawione są parametry, dla jakich robiono testy, a więc liczba zadań (l_{tasks}), średnia długość zadania (c), oraz o ile maksymalnie długości zadań mogą się różnić od wartości średniej (cr). Na wykresie kropka oznacza wynik jednakowy dla algorytmu dokładnego i przybliżonego, a gwiazdka - wynik algorytmu przybliżonego jest gorszy od dokładnego. Z powyższego wykresu wynika także, że dla każdej konfiguracji parametrów wykonano 100 testów.

Wyraźnie zauważalne jest, że im dłuższe zadania (coraz większe c) oraz im bardziej różnorodne (coraz większe cr), tym większe prawdopodobieństwo, że rozwiązanie przybliżone nie będzie optymalne. Wtedy właśnie najlepiej stosować algorytm dokładny, pod warunkiem że będzie on oczywiście w miarę szybki.

3.1 Ilość odcięć

Sprawą kluczową dla szybkości działania algorytmu jest to, ile i na jak wysokim poziomie rozwiązań będzie odrzucanych przez górne i dolne ograniczenia. Całkowita liczba permutacji to $n!$, natomiast liczba permutacji do określonego poziomu k to $n(n-1)\dots(n-k)$. A więc odrzucając błędne rozwiązania, na etapie, gdy nie znamy już tylko jednego elementu uszeregowania, możnaby się pozbyć łącznie połowy permutacji. Z drugiej strony, jeśli znając już pierwsze uszeregowane zadanie, możnaby stwierdzić, czy w pozostałych jest L_{max} gorszy od $best_ever_L_{max}$, zbiór rozwiązań do przeszukania zmniejszałby się za każdym razem o $\frac{1}{n}$. Czas jednak przedstawić wyniki doświadczalne.

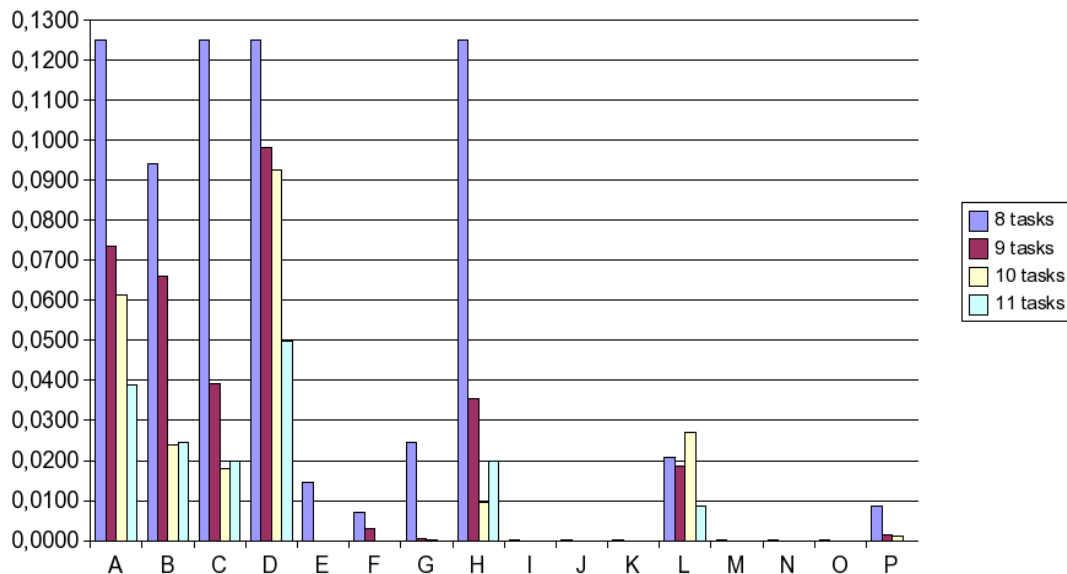
Wykonano próby A-P, o następujących parametrach $c \in [100, 200, 500, 1000]$, $cr \in [\frac{1}{10}c, \frac{1}{5}c, \frac{1}{3}c, 1c]$:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
c	100	100	100	100	200	200	200	200	500	500	500	500	1000	1000	1000	1000
cr	10	20	33	100	20	40	66	200	50	100	166	500	100	200	333	1000

Przez liczbę zejść rozumieliśmy liczbę permutacji wszystkich elementów, jakie wyświetlił program w trakcie poszukiwania rozwiązania optymalnego. Dla każdego przypadku wykonano 100 testów, a poniżej przedstawione są najgorsze wyniki, czyli maksymalna oraz średnia liczba permutacji, jakie wykonał program. Wartości minimalnej nie podawano, gdyż w wersji 3, jeśli algorytm znajdował optymalne L_{max} za pierwszym razem, to nie wyświetlał więcej permutacji, a w wersji 2 wyświetlał zbiór różnych uszeregowień o najlepszym L_{max} . Liczba permutacji na wykresach jest z przedziału $[0, 1]$, gdzie $1 = l_{tasks}!$.

Najmniej odcieć, a więc najgorsze wyniki są dla uszeregowień 8 zadań (próby A,B,C,D). Dla większych zbiorów algorytm osiąga coraz lepsze wyniki. Z tym, że najtrudniejsze obliczeniowo są zbiory o bardzo zróżnicowanych długościach zadań (próby D,H,L,P).

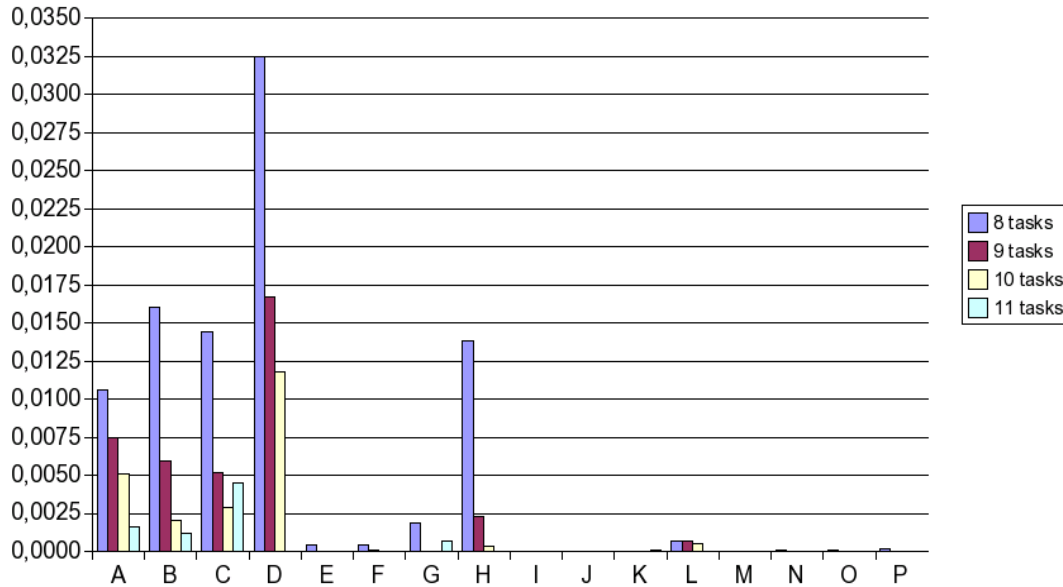
B&B wer.3 Maksymalna liczba zejść



Dla najgorszych rozwiązań interesująca jest także ta wartość maksymalna (powyżej 0,12), osiągnięta w kilku kolumnach wykresu dla 8 zadań. Przeglądając dokładne dane liczbowe, sprawdzono, że 0,12 odpowiada 5041 permutacji. Przy czym $5041 = 7! + 1$. A więc w najgorszym przypadku, algorytm nie jest w stanie zastosować dolnego ograniczenia, a opiera się jedynie na górnym. Posiadając 7 uszeregowanych zadań jest w stanie określić, czy dla pozostałego 1

zadania, uszeregowanie będzie optymalne, czy nie. W najgorszym przypadku konieczne było sprawdzenie wszystkich $7!$ takich przypadków, oraz 1 przypadek, kiedy faktycznie znalezione zostało rozwiązanie optymalne. Co ciekawe, dla dłuższych zadań (seria L i P) przypadek ten nie miał już miejsca.

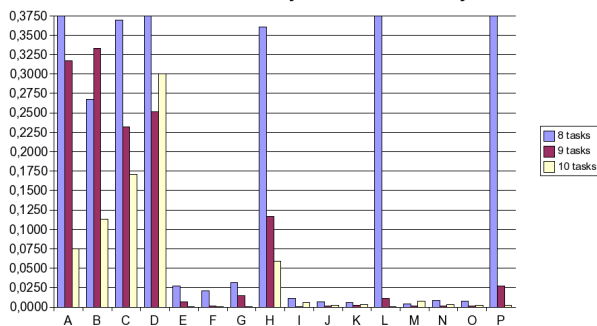
B&B wer.3 Średnia liczba zejść



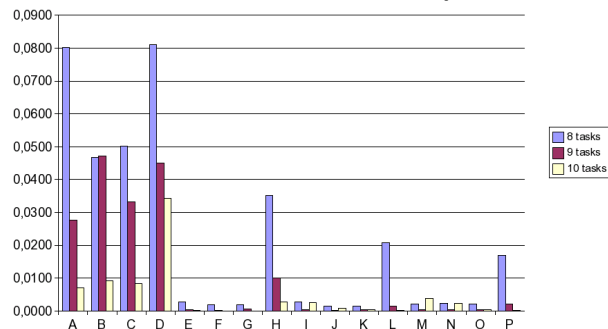
Poniżej zamieszczone zostały dla porównania wyniki działania dla algorytmu w wersji 2. Bardzo wyraźnie widać, jak duże są różnice między jednym i drugim rozwiązaniem, mimo że jest to tak naprawdę tylko zmiana znaku w jednym z warunków. Tutaj równie wyraźnie widoczna jest pesymistyczna złożoność - dla prób A,D,L i P, dla 8 zadań.

Dla wersji 2 algorytmu nie ma serii z 11 zadaniami, bo nie doczekano się wyników w sensownym czasie.

B&B wer.2 Maksymalna liczba zejść



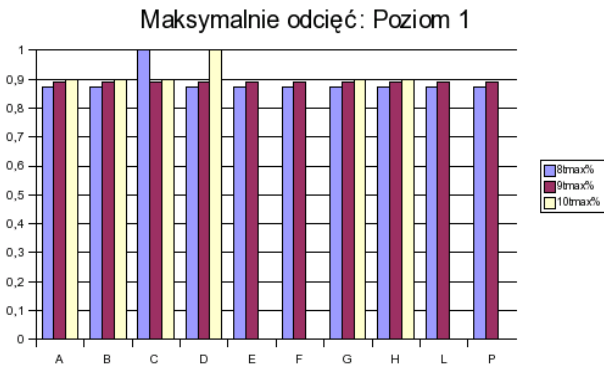
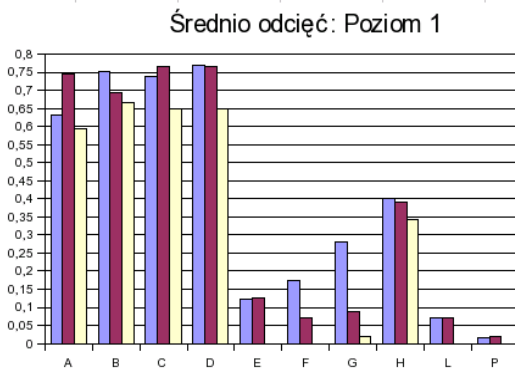
B&B wer.2 Średnia liczba zejść



3.2 Jakość odcięć

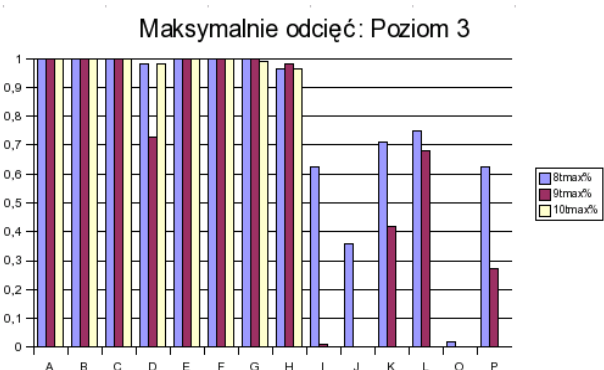
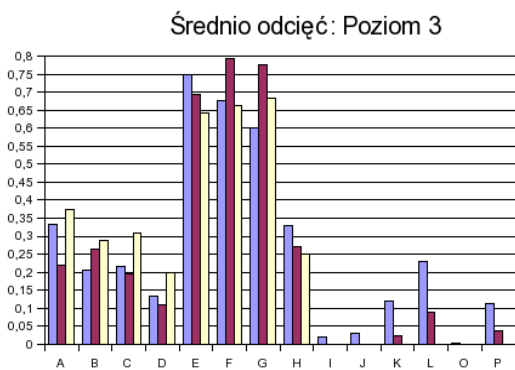
Należy pamiętać, że aby utworzyć permutację wszystkich elementów, algorytm po kolei składa kolejne zadania. Wypisywane są dopiero ciągi, gdy wszystkie zadania zostały uszeregowane. Jeśli więc algorytm robił wszystkie uszeregowania, a odcięcia dopiero na przy ostatnim zadaniu (jak przypadek omówiony powyżej dla 8 zadań, w próbach A,C,D i H - o najbardziej różnorodnych zadaniach). Na drugim etapie testów sprawdzono, jak to się dzieje, że dla większych zbiorów zadań wyniki są lepsze, oraz prześledzono na jakich poziomach są odcięcia.

Poniżej przedstawiono wykresy liczby odcięć w zależności od ilości zadań (8-10) i rodzaju próby (A-P), dla każdego poziomu informacje zostały zebrane na wykresach z wartościami średnimi i maksymalnymi. Znowu ilości odcięć są w przedziale $[0, 1]$. Jeśli dla jakiegoś rodzaju próby, w żadnej serii nie było odcięć, to pomijano takie przypadki na wykresach.

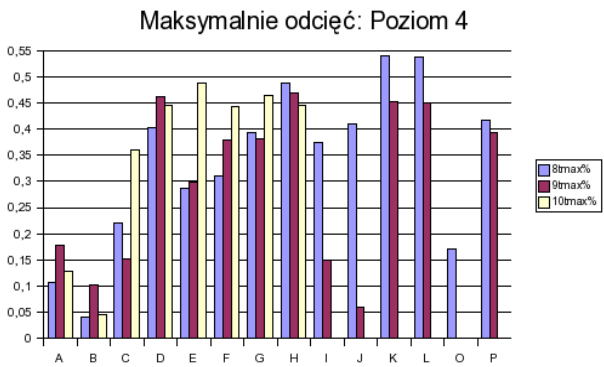
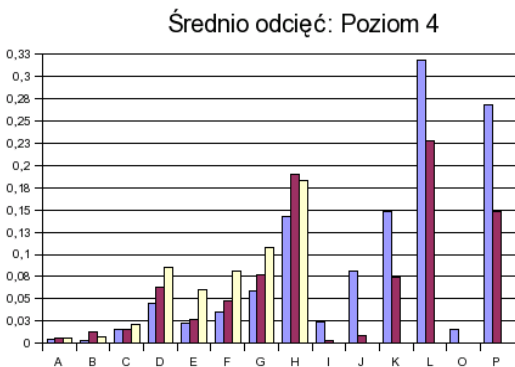


Na odcięciach na 1 poziomie najbardziej nam zależy, ponieważ eliminują $\frac{1}{n}$ rozwiązań. W dwóch przypadkach pierwsze w ogóle uszeregowanie było optymalne (wykres maksimum odcięć, wartości równe 1). Uwidacznia się też zależność między rodzajem danych a jakością odcięć. Dla zbiorów krótkich zadań (A,B,C,D - $c = [90, 110]$), niezależnie od liczby zadań, algorytm przy wybieraniu pierwszego elementu optymalnego uszeregowania odrzuca 60-75% złych zadań. Więc np. dla zbioru $n = 8$, odrzuca 6-7 zadań. Niestety im większa różnorodność zadań, tym decyzja staje się trudniejsza. Kluczowe znaczenie ma początkowe uporządkowanie zadań (u nas posortowanie wg. $d - c$) oraz początkowe L_{max} .

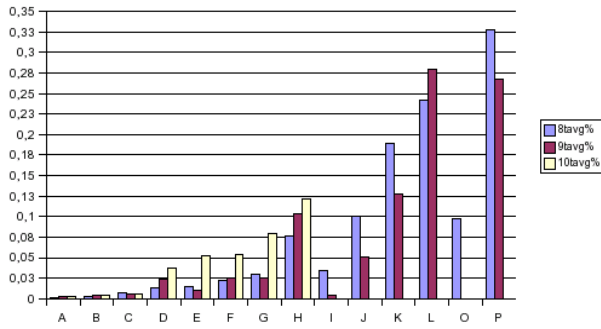
Dla odcięć na 2 poziomie nie przygotowano wykresów, ponieważ nie występowały takie. Dokładnie mówiąc jedne odcięcia na 2 poziomie miały miejsce dla 10 zadań o długości $[0, 200]$ (próba D), w jednym przypadku było 7 odcięć. Według nas jest to skutek tego, że mamy trzy procesory i zarówno 1 jak i 2 zadanie, startują od 0. Pierwsze zadanie ma krótszy termin (wpływ uszeregowania), więc drugie zadanie zawsze ma mniejsze l .



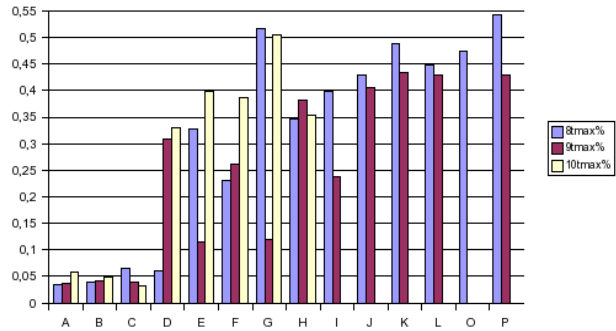
Na trzecim poziomie największa liczba odcięć przenosi się na zadania trochę dłuższe. Na dalszych etapach Średnia odcięć będzie się przesunąć coraz bardziej w prawą stronę wykresów. Znaczy to, że dla coraz dłuższych zadań, odcięcia występują na coraz niższym poziomie.



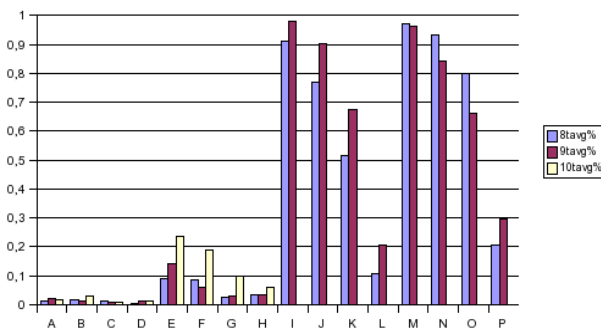
Średnio odcięć: Poziom 5



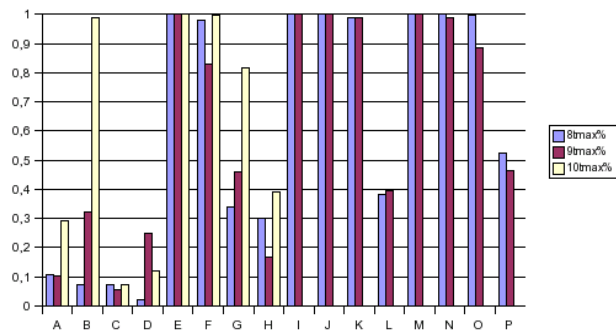
Maksymalnie odcięć: Poziom 5



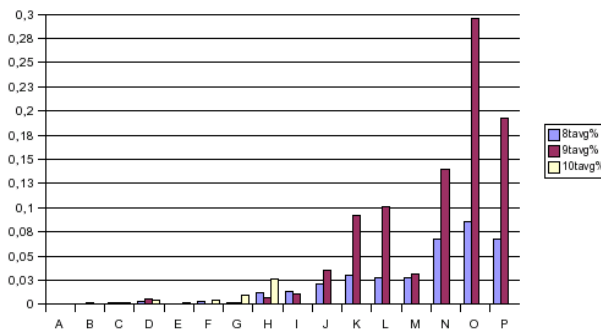
Średnio odcięć: Poziom 6



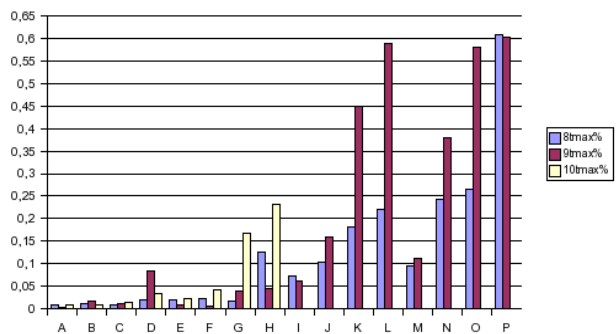
Maksymalnie odcięć: Poziom 6



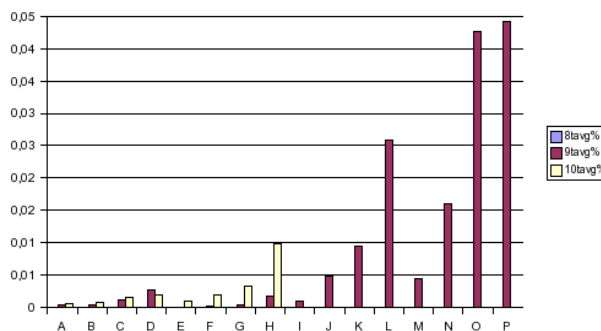
Średnio odcięć: Poziom 7



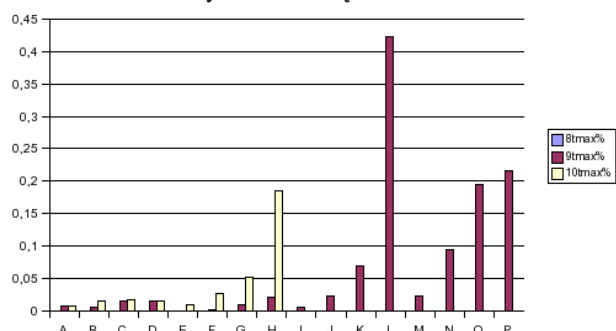
Maksymalnie odcięć: Poziom 7



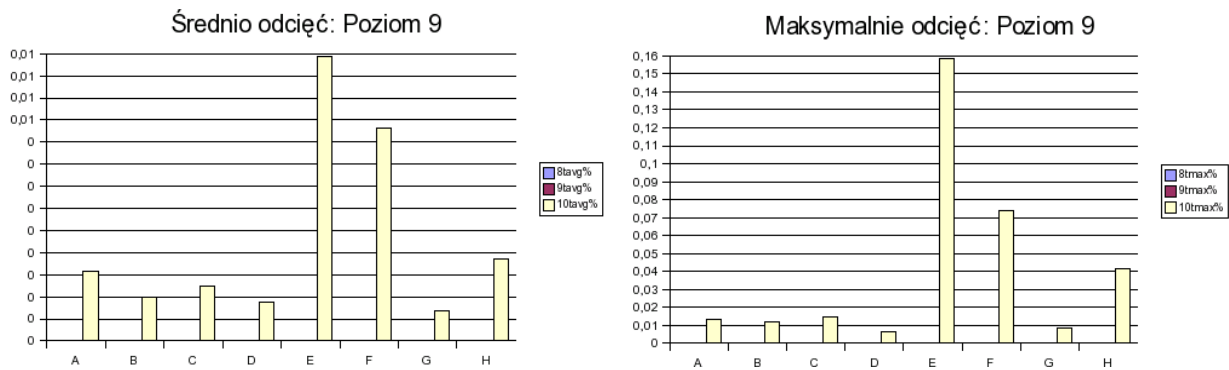
Średnio odcięć: Poziom 8



Maksymalnie odcięć: Poziom 8



Na poziomie 8, nie ma już odcięć dla zbiorów 8 zadań, bo wszystkie ostateczne odcięcia zostały wykonane do poziomu 7. Analogicznie dla zbiorów 9 zadań na kolejnym wykresie.



Podsumowując ten podpunkt, mamy nadzieję, że dokładnie udało się udokumentować działanie zastosowanych odcieć dla zbiorów zadań o różnych charakterystykach. Najbardziej interesujące wg. nas uwagi z tego i wcześniejszych podpunktów zostały także zebrane we wnioskach.

4 Wnioski

- Obserwując, gdzie są wykonywane odcięcia, doszliśmy do wniosku, że nawet przy dostatecznie dobrym ograniczeniu górnym, w postaci *best_ever Lmax*, dużo czasu tracone jest na sprawdzaniu permutacji “w środku” drzewa rozwiązań, czyli gdy pozostało jeszcze sporo zadań do uszeregowania.
- Jakość odcieć, chociaż ma bardzo duży wpływ na szybkość algorytmu i tak nie jest w stanie “przełamać” wykładniczej złożoności. A więc, niezależnie od nich, z coraz większymi zbiorami zadań do uszeregowania, potrzebne może być coraz więcej czasu.
- Rodzaj danych ma decydujące znaczenie dla szybkości działania algorytmu.
- Im dłuższe są zadania, tym odcięcia występują na niższym poziomie - później.
- Im zadania są bardziej różnorodne, tym więcej trzeba wykonać permutacji. Jednak w zadaniach bardziej różnorodnych jest więcej odcieć niż w zadaniach o zbliżonych długościach na tych samych poziomach.
- Obydwa ograniczenia są istotne, jednak dzięki efektywności dolnego ograniczenia, górne prawie nigdy nie jest nawet sprawdzane.