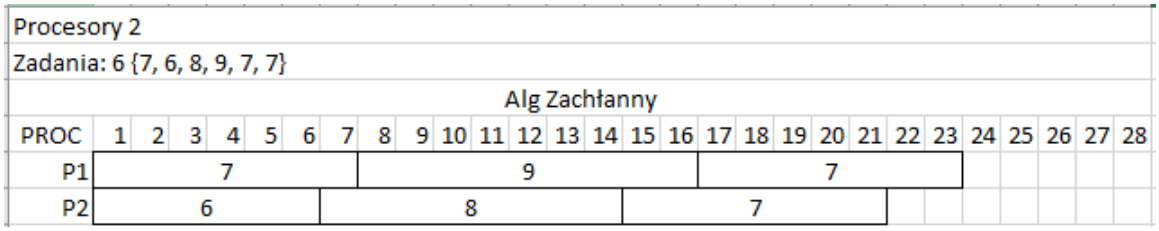
P||Cmax

Bartosz Dolata 148222
Wiktor Szymański 148084

Algorytm przeszukiwania Tabu

1. ## Inicjalizacja



Procesory 2																										
Zadania: 6 {7, 6, 8, 9, 7, 7}																										
												Alg Zachłanny														
PROC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
P1	7							9									7											
P2	6							8									7											

2. ## Opis Algorytmu

Algorytm przeszukiwania Tabu (Tabu Search) to metaheurystyka pozwalająca rozwiązywać problemy optymalizacyjne. W algorytmie wykorzystuje się sekwencję ruchów do znalezienia najlepszego sąsiedztwa. Dodatkowo wyposażony jest listę Tabu (Tabu List) w celu uniknięcia oscylowania metaheurystyki wokół lokalnego optimum. Tabu Search jest metodą bardzo ogólną. Różni się zależnie od rozwiązywanego problemu. Dla problemu P||Cmax zbędnym jest wykorzystanie takich elementów jak kryterium aspiracji, gdyż dane procesy nie posiadają wag, które mogłyby decydować o wyborze wyżej cenionego procesu. Algorytm sprawnie opuszcza lokalne optima, lecz wiąże się to z koniecznością przechowywania informacji na temat poprzednich rozwiązań

3. Pseudokod

Procedure TabuSearch
n <- wielkość listy sąsiedztwa
S <- rozwiązanie początkowe

while S jest różne od 100 ostatnich identycznych rozwiązań lub upłynęło mniej niż 5 min
while length(neighbourhood)<**n**
wykonaj **Ruch**(**S**)
wpisz rozwiązanie do sądzsiedztwa

for całe sąsiedztwo
wpisz czas sąsiedztwa do tablicy czasów

S <- rozwiązanie z najlepszym czasem z sąsiedztwa

Procedure Ruch

Znalezienie **best_move** \\ zadanie z najbardziej obciążonego procesora przeniesione do najmniejobciążonego lub robi swap między zadaniami w różnych procesorach

while best_move jest w liście tabu:
szukaj innego najlepszego ruchu:

dodaj best_move do listy tabu
return best_move

4. Przykład obrazujący działanie

1. W rozwiązaniu algorytmu zachłannego, w procesorze najbardziej obciążonym zadaniami wybieramy najkrótsze
zadanie.

Alg Zachłanny

PROC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
P1	7							9									7											
P2	6							8									7											

2. Przenosimy zadanie do procesora najmniejobciążonego

Krok 1

PROC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
P1	7							9																				
P2	6							8									7											

3. Ponownie wybieramy zadanie w procesorze najbardziej obciązonym i przenosimy do najmniej obciążonego

Krok 1																											
PROC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
P1	7							9																				
P2	6							8							7							7						

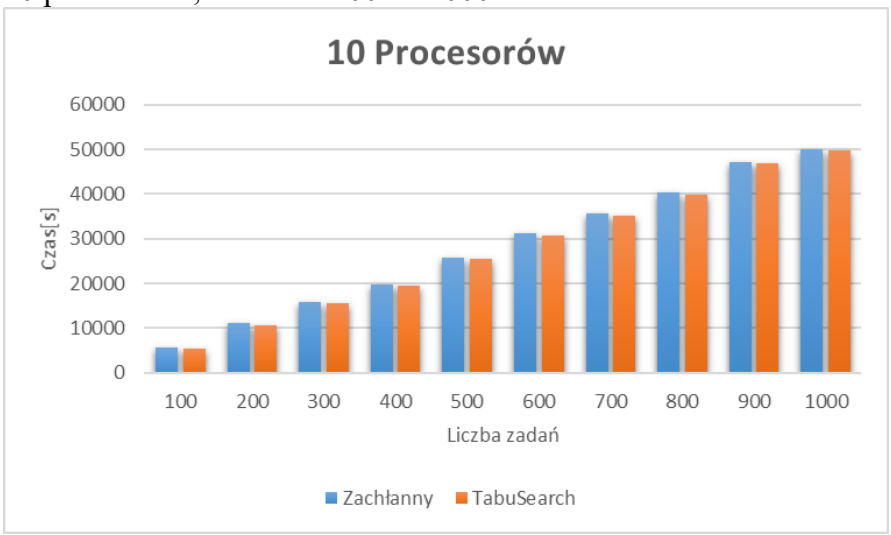
Krok 2																											
PROC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
P1	7							9									6											
P2	8							7							7													

4. **Finalizacja**

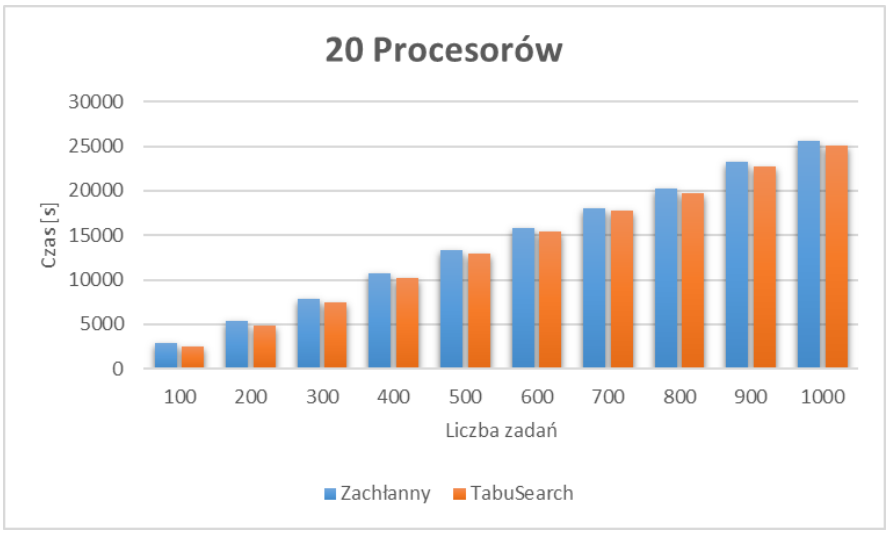
Krok 2																											
PROC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
P1	7							9									6											
P2	8							7							7													

Wykresy

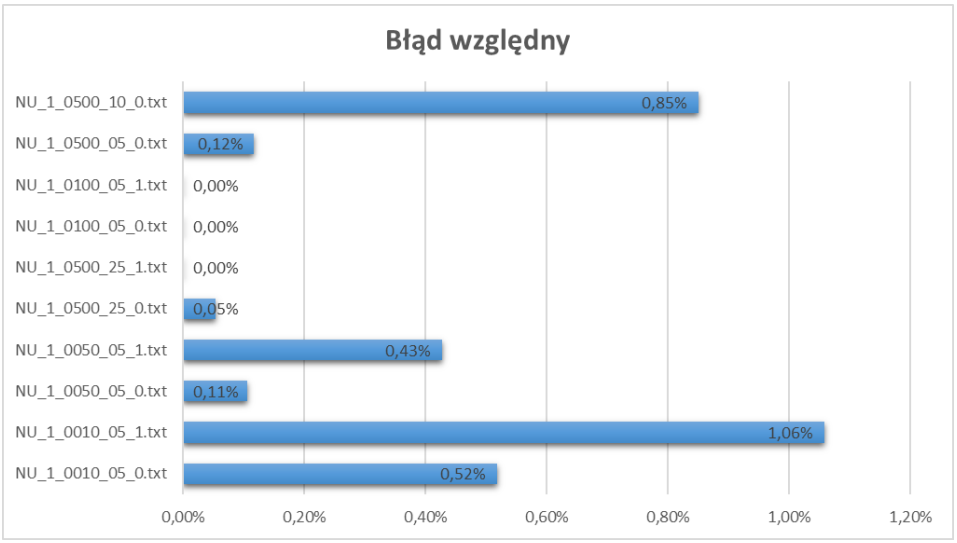
1. Porównanie TabuSearch z algorytmem zachłannym
    1. Wykresy przedstawia minimalny czas potrzebny do realizacji zadań na 10 lub 20 procesorach. Widzimy, że algorytm TabuSearch poprawił czas w stosunku do algorytmu zachłannego. Nie wielkie różnice wynikają z dość dobrej pracy algorytmu zachłannego. TabuSearch okazał się najlepszy w zakresie liczby zadań wynoszącym od 300 do 700.
    2. 10 procesorów, zadań od 100 do 1000



3. 20 procesorów zadań od 100 do 1000



2. Wartość błędu względnego wartości optymalnej w stosunku do uzyskanej z metaheurystyki TabuSearch



3. Wyniki instancji rankingowych

M25	**3470**
M50	**151**
M10N200	**11000**
M50N200	**1016**
M50N1000	**9763**