Algorytm symulowanego wyżarzania dla szeregowania zadań równoległych z przyległymi przydziałami procesorów

Paweł Kierkosz 155995
Kamil Bryła 155988
E-mail do kontaktu – pawel.kierkosz@student.put.poznan.pl

1. Opis problemu

Podobnie, jak to miało miejsce w przypadku poprzedniego sprawozdania, rozważany tu problem polega na ustaleniu kolejności dostępu zadań do procesorów, w taki sposób, aby suma czasów zakończenia wszystkich zadań była możliwie najmniejsza.

Przyjęte oznaczenia:

n – liczba zadań,

m – liczba procesorów,

$T = \{T_1, T_2, \dots, T_n\}$ – zbiorem przydzielanych zadań,

$P = \{P_1, P_2, \dots, P_m\}$ – zbiór równoległych procesorów,

r_j – moment gotowości zadania T_j, tj. najszybszy możliwy czas jego rozpoczęcia,

p_j – czas trwania zadania T_j (długość zadania),

$size_j$ lub s_j – liczba procesorów przydzielonych do zadania T_j (rozmiar lub szerokość zadania).

Zakładamy również, że procesory są identyczne (jednakowo szybkie), a zadania przydzielane są do procesorów w sposób przyległy (indeksy procesorów przypisanych do każdego z zadań muszą być kolejnymi liczbami naturalnymi) oraz ciągły (zadanie już rozpoczęte musi być realizowane w sposób nieprzerwany przez p_j jednostek czasu na takim samym zestawie procesorów).

Każdy sposób przydziału zadań do procesorów spełniający wymagane warunki będziemy nazywać harmonogramem lub uszeregowaniem zadań. Jako funkcję celu, czyli kryterium optymalizacji, przyjmiemy sumę czasów zakończenia wszystkich zadań, którą będziemy oznaczać przez $SumC_j$. Kryterium to będzie podlegać minimalizacji.

2. Opis algorytmu

Opracowany przez nas algorytm bazuje na heurystycznej strategii BF (best-fit) przydzielania kolejnych zadań do procesorów opisanej w poprzednim sprawozdaniu, dotyczącym algorytmu zachłannego. Jest to zmodyfikowany wariant algorytmu zaproponowanego w pracy [1] i nazywanego często w literaturze algorytmem „skyline”.

Do przetwarzania kolejnych rozwiązań nasz algorytm wykorzystuje znaną z literatury metodę (metaherustykę) symulowanego wyżarzania (np.: [2, 3, 4]). Jest to iteracyjna technika polegająca na tym, że najpierw losowane jest pewne rozwiązanie rozpatrywanego problemu,

a następnie jest ono w kolejnych iteracjach modyfikowane. Jeżeli nowo otrzymane rozwiązanie jest lepsze od poprzedniego, to jest ono zawsze akceptowane, jeżeli natomiast jest rozwiązaniem gorszym, to jest ono akceptowane z pewnym prawdopodobieństwem, które to prawdopodobieństwo zależy między innymi od wartości parametru sterującego algorytmem zwanego temperaturą. W trakcie wykonywania algorytmu temperatura jest stopniowo obniżana, zgodnie z przyjętym schematem chłodzenia. Sposób działania algorytmu przypomina znane z metalurgii zjawisko wyżarzania. Stąd też wzięła się nazwa tej metaheurystyki.

Poniżej przedstawiony został ogólny schemat algorytmu symulowanego wyżarzania. W schemacie tym przyjęto następujące oznaczenia:

T_0 – temperatura początkowa,
t, l – zmienne kontrolujące pętle: zewnętrzną (t) i wewnętrzną (l),
L – liczba iteracji wewnętrznych (liczba iteracji w jednym cyklu temperaturowym),
f – funkcja celu (minimalizowana),
$g(T, t)$– przyjęty schemat schładzania.

Algorytm symulowanego wyżarzania

1: inicjalizuj $T \coloneqq T_0$
2: przyjmij $t \coloneqq 0$
3: wygeneruj rozwiązanie startowe x
4: oceń rozwiązanie startowe, tj. oblicz $f(x)$
5: **repeat**
6: **for** $l \coloneqq 1$ **to** L **do**
7: wybierz nowe rozwiązanie x' z sąsiedztwa rozwiązania x
8: oblicz $f(x')$
9: **if** $f(x') < f(x)$ **then**
10: $x \coloneqq x'$
11: **else**
12: **if** $\text{random}[0, 1) < \exp\left(-\frac{f(x\prime)-f(x)}{T}\right)$ **then**
13: $x \coloneqq x'$
14: $T \coloneqq g(T, t)$
15: $t \coloneqq t + 1$
16: **until** (kryterium stopu)
17: **return** najlepsze znalezione rozwiązanie

Omówimy teraz ważniejsze punkty naszego algorytmu w odniesieniu do kroków powyższego schematu:

***Krok* 3 i *Krok* 4:** wygeneruj rozwiązanie startowe x oraz dokonaj jego oceny

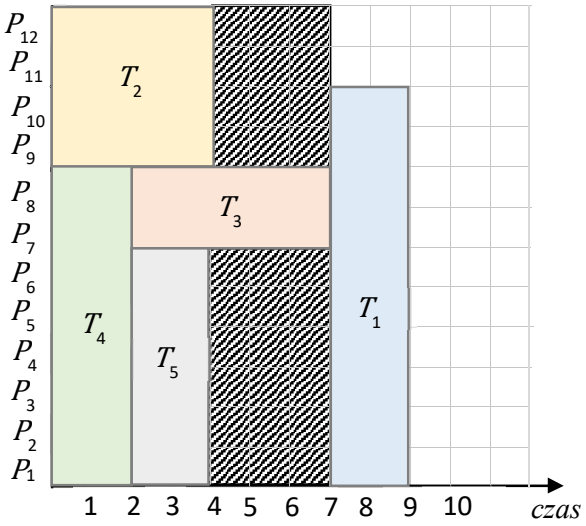
W przypadku naszego algorytmu testowaliśmy dwa sposoby tworzenia rozwiązań startowych. W pierwszej metodzie w sposób losowy generowana jest permutacja *n* liczb naturalnych od 0 do *n* – 1, która stanowi zakodowaną postać rozwiązania danego problemu, czyli pewnego

losowego harmonogramu. Następnie permutacja ta jest dekodowana z wykorzystaniem heurystycznej procedury przydziału zadań do procesorów opisanej w poprzednim sprawozdaniu. W procedurze tej w każdej iteracji wyznaczany jest najpierw punkt zaczepienia, a następnie ze zbioru wszystkich elementów, które mogą być zaczepione w danym punkcie wybiera się jedno zadanie. O ile w poprzednio opisywanym algorytmie zachłannym wybierany jest element o najmniejszej długości (zadanie o najkrótszym czasie wykonania), o tyle w algorytmie symulowanego wyżarzania wybierany jest pierwszy element z listy możliwych do umieszczenia elementów, gdzie kolejność elementów na tej liście określona jest przez daną permutację. Jako ilustracja niech posłuży podany w poprzednim sprawozdaniu przykład:

Przykład 1. Mamy 12 identycznych procesorów i na nich ma zostać wykonanych 5 zadań. Parametry tych zadań oraz kodujące wartości indeksów zadań podane są w poniższej tabelce.

indeks zadania	wartość kodowa w permutacji	r_j	p_j	$size_j$
1	0	0	2	10
2	1	0	4	4
3	2	0	5	2
4	3	0	2	8
5	4	1	2	6

Załóżmy, że losowo została wygenerowana następująca permutacja: {3, 1, 0, 4, 2}. Permutacji tej odpowiadają zadania o indeksach: {4, 2, 1, 5, 3}. Znajdujemy najpierw pierwszy punkt zaczepienia: (0, 0) oraz wyznaczamy zbiór elementów (zadań), które mogą zostać zaczepione w danym punkcie: {4, 2, 1, 3}, a których kolejność określona jest przez kolejność elementów permutacji. Wybieramy pierwszy element z tak otrzymanej listy (zadanie o indeksie 4) i umieszczamy go w harmonogramie.



Rys. 1. Harmonogram otrzymany do przykładu 1

Następnym punktem zaczepienia będzie punkt (0, 8). Uwzględniając kolejność określoną przez daną permutację, w punkcie tym mogą zostać zaczepione zadania: {2, 3}. Wybieramy pierwsze

zadanie z listy, tj. zadanie nr 2 i umieszczamy go w harmonogramie. W następnym punkcie zaczepienia (2, 0) można umieścić zadania: {5, 3}. Zatem wybieramy zadanie o indeksie 5. Stosując taką samą zasadę do pozostałych elementów otrzymalibyśmy harmonogram przedstawiony na rysunku 1. Kolejność rozmieszczanych elementów byłaby następująca: {4, 2, 5, 3, 1}. Dla tego harmonogramu $SumC_j = 26$. Należy tutaj zauważyć, że różnym permutacjom po ich zdekodowaniu mogą odpowiadać takie same harmonogramy. Przykładowo taki sam harmonogram, jak ten przedstawiony na rysunku 1, uzyskalibyśmy dla permutacji: {4, 3, 1, 0, 2}. Zdecydowaliśmy się jednak na taki pośredni sposób kodowania harmonogramów, ponieważ ułatwia on przetwarzanie i modyfikowanie rozwiązań. Każdą permutację można w ten sposób zdekodować na dopuszczalne rozwiązanie.

Przeprowadziliśmy również testy, w których zbiór przydzielanych zadań zostaje najpierw posortowany rosnąco względem czasów gotowości, a następnie, jako wektor startowy wybierana jest permutacja odpowiadająca takiemu uporządkowanemu zbiorowi zadań. Permutacja ta dekodowana i oceniana jest zgodnie z opisaną powyżej procedurą. Okazało się, że w niektórych przypadkach takie wstępne posortowanie zadań poprawia skuteczność całego algorytmu, dlatego w dalszych testach efektywnościowych używaliśmy tego wariantu algorytmu.

***Krok* 7:** wybierz nowe rozwiązanie x' z sąsiedztwa rozwiązania x

Nowe rozwiązanie tworzone jest z poprzedniego poprzez zamianę miejscami dwóch losowo wybranych elementów w permutacji będącej zakodowaną postacią danego rozwiązania. Przykładowo dla permutacji: {3, 2, 1, 0, 4}, której po zdekodowaniu odpowiada harmonogram: {4, 3, 2, 5, 1}, po zamianie miejscami elementów 2 i 0 (zmodyfikowana permutacja: {3, 0, 1, 2, 4}) odpowiadałby harmonogram: {4, 2, 3, 5, 1}. Przeprowadziliśmy również testy, w których przestawialiśmy sąsiadujące ze sobą elementy, ale średnia z otrzymywanych wyników była wówczas gorsza.

***Krok* 9 i *Krok* 10: if** $f(x') < f(x)$ **then** $x := x'$

Jeżeli nowo utworzone rozwiązanie jest lepsze od poprzedniego, to jest ono akceptowane i przyjmowane, jako rozwiązanie bieżące.

***Kroki* 11 - 13: else if** $\text{random}[0\,,1) < \exp\left(-\frac{f(x\prime)-f(x)}{T}\right)$ **then** $x := x'$

Jeżeli nowo wygenerowane rozwiązanie nie jest lepsze od poprzedniego, to ze wzoru:

$$p = \exp\left(-\frac{f(x') - f(x)}{T}\right)$$

obliczane jest prawdopodobieństwo akceptacji tego rozwiązania i jeżeli jest ono większe niż losowo wygenerowana liczba z przedziału $[0\,,1)$, to nowe rozwiązanie jest akceptowane. Zauważmy, że prawdopodobieństwo akceptacji p nowego rozwiązania zależy od dwóch wartości. Po pierwsze, od różnicy wartości funkcji celu nowego i starego rozwiązania – im ta różnica jest mniejsza, tym nowe rozwiązanie ma większą szansę być zaakceptowanym jako rozwiązanie bieżące i być przetwarzane w kolejnej iteracji algorytmu. Po drugie, od aktualnej wartości temperatury T – im temperatura jest większa, tym większe jest prawdopodobieństwo przyjęcia nowego rozwiązania, jako rozwiązanie aktualne. Temperatura

ta jest stopniowo obniżana w kolejnych iteracjach głównej pętli algorytmu. Zatem w początkowej fazie algorytmu, gdy temperatura jest wysoka, gorsze rozwiązania mają większą szansę akceptacji, a następnie szansa ta spada wraz z obniżaniem się temperatury. Przy dobrze dostrojonym algorytmie, w końcowych iteracjach szansa akceptacji gorszego rozwiązania niż bieżące jest już bardzo niskie.

***Krok* 14:** $T \coloneqq g(T, t)$

W tym kroku obniżana jest temperatura T zgodnie z przyjętym schematem chłodzenia. My w swoim algorytmie zmieniamy temperaturę według następującego wzoru:

$$T_{t+1} = \alpha\, T_t,$$

gdzie α jest pewną stałą mniejszą od 1 (na ogół od 0,8 do 0,999). Jest to tzw. geometryczny schemat chłodzenia.

***Krok* 16: until** (kryterium stopu)

Warunki zatrzymania algorytmu mogą być różne. My zdecydowaliśmy się zastosować trzy alternatywne warunki:

1) spadek temperatury T poniżej poziomu temperatury końcowej $T_k = 0{,}0001$,
2) brak poprawy wartości funkcji celu przez 400 iteracji (włączając to również pętlę wewnętrzną),
3) przekroczony czas działania algorytmu (3 lub 5 minut dla prezentacji działania algorytmu i 30 minut do testów).

Algorytm kończy działanie po spełnieniu któregokolwiek z tych warunków.

3. Dobór parametrów sterujących algorytmem

Podobnie jak inne metaherystyki, algorytm symulowanego wyżarzania wymaga dostrojenia, czyli ustalenia wartości pewnych parametrów sterujących jego działaniem. Wpływają one na wydajność i skuteczność algorytmu. W przypadku opracowanego przez nas algorytmu parametrami tymi były: temperatura początkowa T_0, liczba iteracji wewnętrznych L (wykonywanych w jednym cyklu temperaturowym) oraz współczynnik redukcji temperatury α. Przeprowadziliśmy szereg testów, w których zmienialiśmy wartości tych parametrów. W każdym teście sprawdziliśmy wszystkie możliwe układy parametrów dla:

$$T_0 \in \{1;\ 10;\ 100;\ 1000\},\quad L \in \{10;\ 50;\ 100\},\quad \alpha \in \{0{,}8;\ 0{,}85;\ 0{,}9;\ 0{,}95;\ 0{,}99\}.$$

Dla każdego zestawy parametrów algorytm był uruchamiana 10 razy i obliczana była średnia arytmetyczna zyskanych wyników. W tabelce 1 umieściliśmy przykładowe wyniki (zaokrąglone do jedności) ilustrujące, jak zmieniała się wartość funkcji celu w zależności od parametru α, przy pewnych wybranych wartościach parametrów: T_0 i L. Test przeprowadzono na pliku „LANL-CM5-1994-4.1-cln.swf" z biblioteki Parallel Workloads Archive (https://www.cs.huji.ac.il/labs/parallel/workload/) dla *n* = 1000. Można zauważyć, że w zależności od wybranych wartości parametrów T_0 i L możemy mieć do czynienia z różną tendencją jeżeli chodzi o wpływ współczynnika obniżania temperatury na uzyskiwaną wartość funkcji celu. Po przeprowadzeniu testów wstępnych ostatecznie zdecydowaliśmy się przyjąć

następujące domyślne wartości parametrów sterujących działaniem opracowanego przez nas algorytmu symulowanego wyżarzania: $T_0 = 10$, $L = 10$, $\alpha = 0{,}99$.

Wartości parametrów T_0 i L	Wartości parametru α				
	0,8	0,85	0,9	0,95	0,99
T_0=1, L=10	128075392	128055720	127904144	127623248	127139368
T_0=1, L=50	127625072	127463320	127380624	127251280	127179120
T_0=1, L=100	127363608	127251688	127249984	127183632	127209960
T_0=10, L=10	128011824	127856064	127745664	127507496	127136384
T_0=10, L=50	127539696	127450448	127215888	127292720	127158464
T_0=10, L=100	127314024	127162928	127247696	127325224	127259760
T_0=100, L=10	127788928	127838976	127763240	127490096	127236440
T_0=100, L=50	127385688	127381568	127312984	127322832	127263920
T_0=100, L=100	127348112	127224912	127281128	127203712	127217880
T_0=1000, L=10	128103168	127923432	127815664	127458392	127442536
T_0=1000, L=50	127308944	127395176	127180080	127121024	127380456
T_0=1000, L=100	127285544	127296896	127302656	127388248	127389608

Tab. 1. Wartości $SumC_j$ uzyskane dla instancji „LANL-CM5-1994-4.1-cln.swf" w zależności od wartości parametrów sterujących

Zaznaczmy tutaj również, że trudno jest dobrać wartości parametrów sterujących, które byłyby najlepsze dla wszystkich instancji testowych. W przeprowadzonych testach okazało się, że na uzyskiwane wyniki duży wpływ miał chociażby rozmiar problemu, rozumiany jako liczba wczytywanych zadań. Dodatkowo wartości parametrów sterujących mocno wpływają na czas, w jakim algorytm symulowanego wyżarzania stabilizuje się osiągając niską temperaturę. Zatem wybór wartości parametrów algorytmu może być wynikiem kompromisu między efektywnością algorytmu, a czasem jego działania.

4. Wprowadzenie do przeprowadzonych badań

W celu sprawdzenia efektywności drugiego algorytmu przeprowadziliśmy badania, tak jak w poprzednim sprawozdaniu, na plikach obowiązkowych: *DAS2-fs0-2003-1.swf*, *LANL-CM5-1994-3.1-cln.swf* oraz *SDSC-SP2-1998-3.1-cln.swf*. Ponownie przeprowadziliśmy testy na dwóch dodatkowych plikach, w celu dokonania pogłębionej analizy, jak i również lepszego pokazania efektywności naszego algorytmu. Były to pliki: *CTC-SP2-1996-3.1-cln.swf* oraz *LANL-CM5-1994-4.1-cln.swf*. W odróżnieniu od pierwszych badań, wyniki aktualnie badanego algorytmu symulowanego wyżarzania zostały porównane z rezultatami uzyskanymi z wykorzystaniem poprzednio opracowanego algorytmu zachłannego. We wstępie warto również wspomnieć, że maksymalny dopuszczalny przez nas czas działania algorytmu symulowanego wyżarzania wynosił 30 minut (był to jeden z warunków zawartych w kryterium stopu).

Zostały wykonane następujące testy badające zależności:

- czasu działania algorytmu od liczby wczytanych elementów,
- wartość funkcji celu od liczby wczytanych elementów,
- wartości funkcji celu od temperatury w danej iteracji,
- wartości funkcji celu od czasu działania algorytmu.

Pierwszy test, pokazujący czas działania algorytmu w zależności od liczby wczytanych elementów został wykonany podobnie, jak w poprzednim algorytmie. Testy zostały przeprowadzone przez wyznaczenie średniej arytmetycznej czasu działania algorytmu. Dodatkowo wyznaczyliśmy odchylenie standardowe. Dla każdej instancji, w celu uśrednienia wyników, dokonaliśmy dziesięciu pomiarów.

Następne testy zostały wykonane w celu zbadania efektywności algorytmu symulowanego wyżarzania w porównaniu do algorytmu zachłannego. Jest to z pewnością najbardziej wartościowy test, pokazujący efektywność nowego algorytmu w porównaniu do drugiego.

Następnym badanym czynnikiem była temperatura. W porównaniu do poprzedniego, jednoprzebiegowego algorytmu, postanowiliśmy wykonać testy obrazujące przebieg procesu optymalizacji, tj. w jakim stopniu wyniki są poprawiane wraz ze spadkiem temperatury. W algorytmie symulowanego wyżarzania temperatura jest jednym z kluczowych czynników wpływających na jego efektywność.

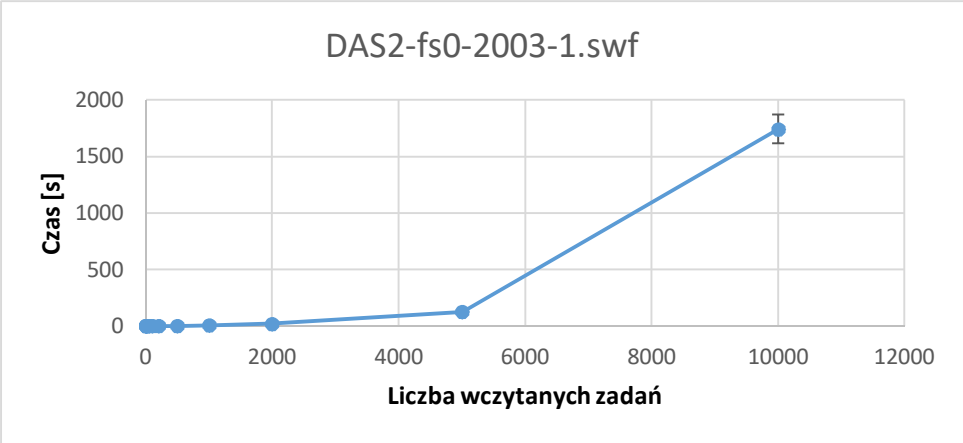
Ostatnie badanie dotyczyło zmienności jakości rozwiązania w zależności od czasu działania algorytmu. Jak dobrze wiemy, algorytm symulowanego wyżarzania będzie zmieniał wartość funkcji celu w czasie. Motywacją do wykonania tego testu było zobrazowanie efektywności naszego algorytmu oraz przebiegu optymalizacji.

Podobnie, jak to miało miejsce w testach mierzących czas działania algorytmu, wyniki dotyczące uzyskiwanych wartości funkcji celu (dla algorytmu symulowanego wyżarzania) były uśredniane z dziesięciu pomiarów. Zostało to wykonane w taki sposób, ponieważ element losowości tego algorytmu powoduje, że wyniki oparte na jednym przebiegu dla każdej instancji byłyby mało miarodajne. Jedynymi badaniami, które nie oparliśmy na wyznaczeniu średniej arytmetycznej były testy pokazujące zmienność wartości funkcji celu od czasu oraz temperatury. W tym przypadku chodziło nam o zaprezentowanie przykładowych przebiegów optymalizacji dla pojedynczych instancji. Przy każdym uruchomieniu algorytmu, z racji jego losowego charakteru, ten przebieg może wyglądać inaczej, oczywiście przy zachowaniu ogólnej tendencji.

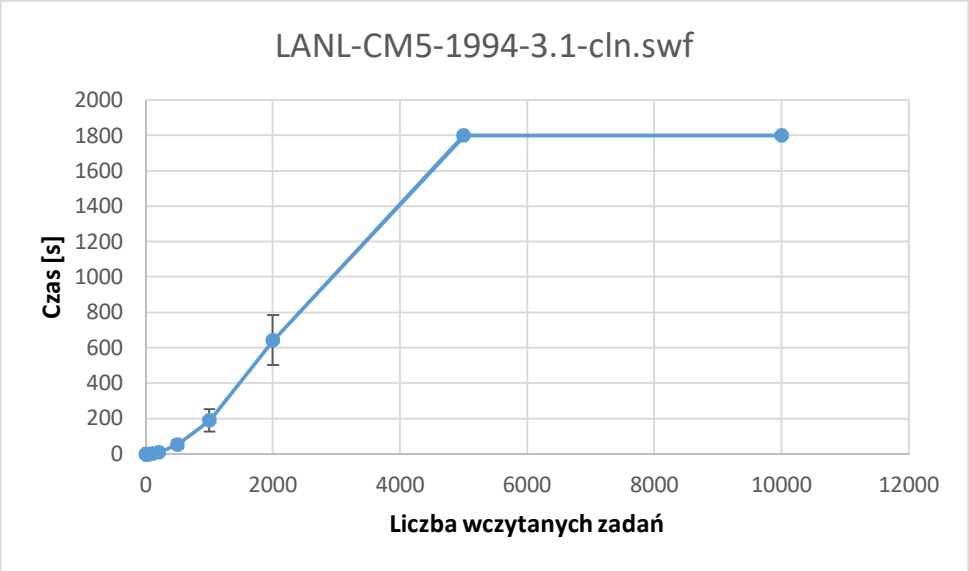
Podobnie, jak w przypadku badań nad algorytmem zachłannym, również w niniejszym sprawozdaniu postanowiliśmy zamieścić wykresy punktowe z prostymi liniami, aby lepiej zobrazować uzyskane wyniki.

5. Wyniki testów obliczeniowych

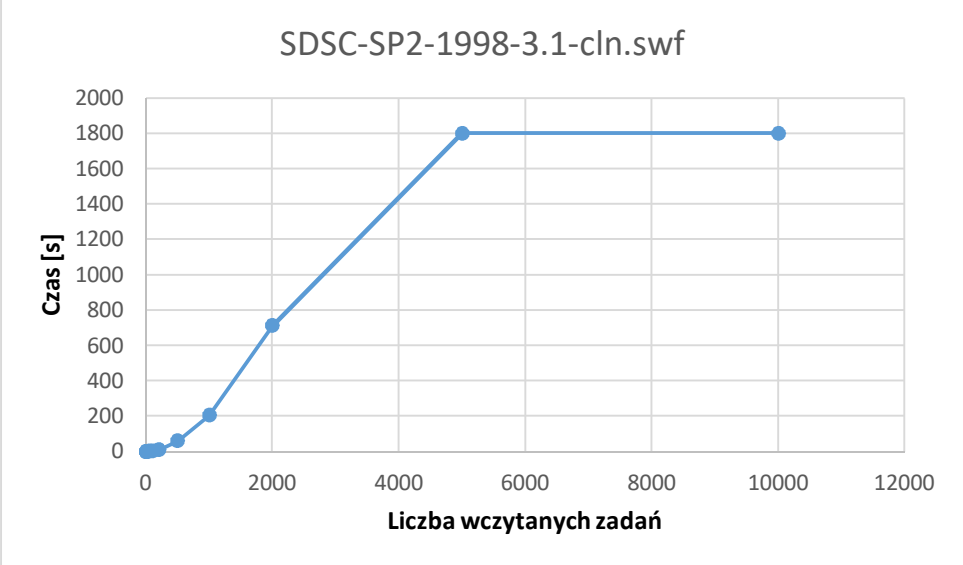
Jak zostało to wspomniane powyżej, pierwsze testy dotyczyły zbadania zależności czasu obliczeń od liczby wczytanych elementów. Na poniższych wykresach zaprezentowano uzyskane wyniki. Można zauważyć, że w większości testów, w przypadku dużych wartości *n* (liczby wczytywanych zadań) miało miejsce zatrzymanie algorytmu po przekroczeniu przyjętego maksymalnego czasu obliczeń równego 30 minut.



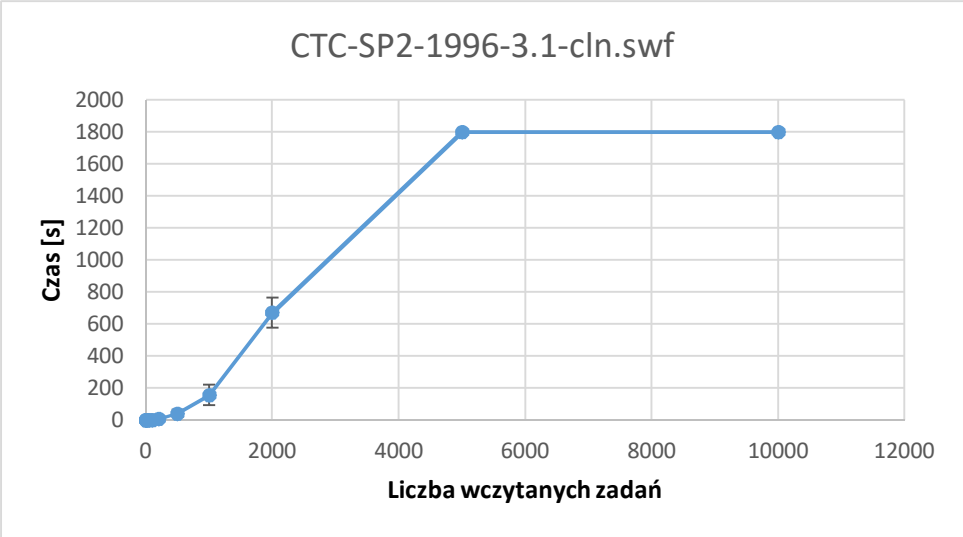
Wykres 1. Zależność czasu obliczeń od liczby wczytanych zadań dla pliku 1.



Wykres 2. Zależność czasu obliczeń od liczby wczytanych zadań dla pliku 2.

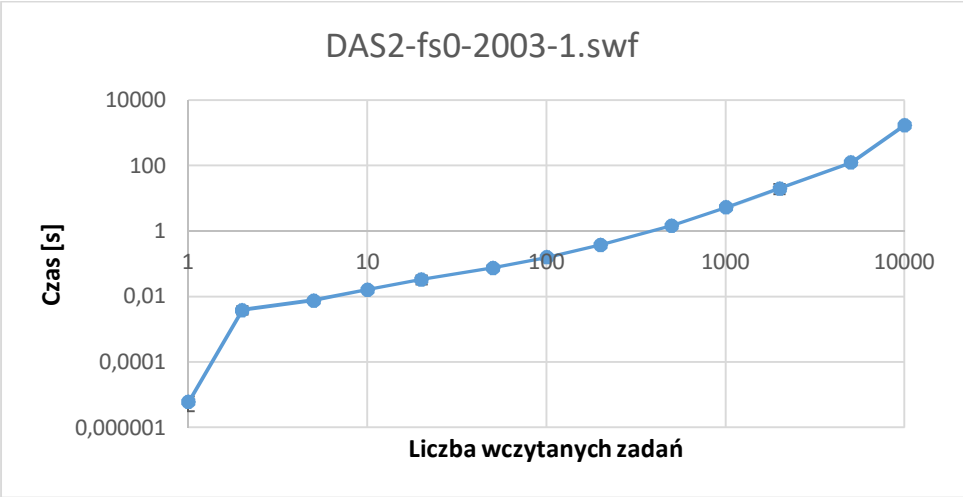


Wykres 3. Zależność czasu obliczeń od liczby wczytanych zadań dla pliku 3.

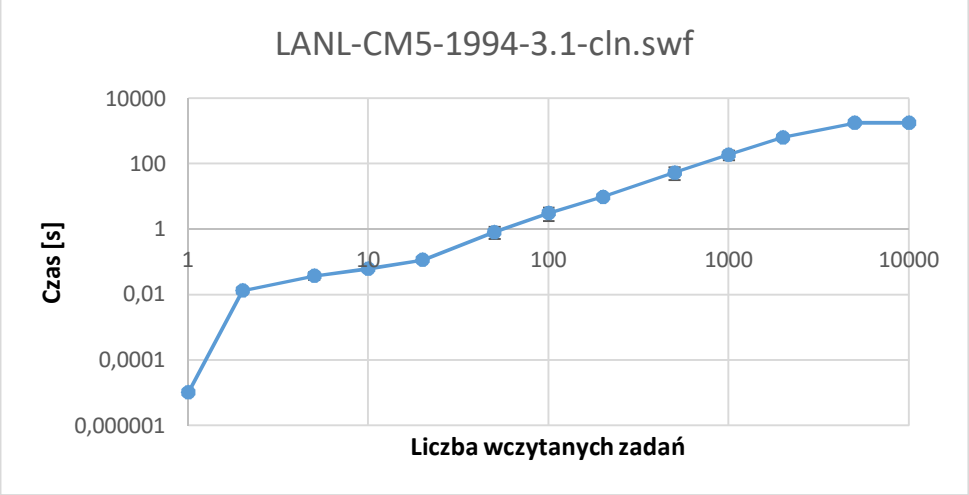


Wykres 4. Zależność czasu obliczeń od liczby wczytanych zadań dla pliku 4.

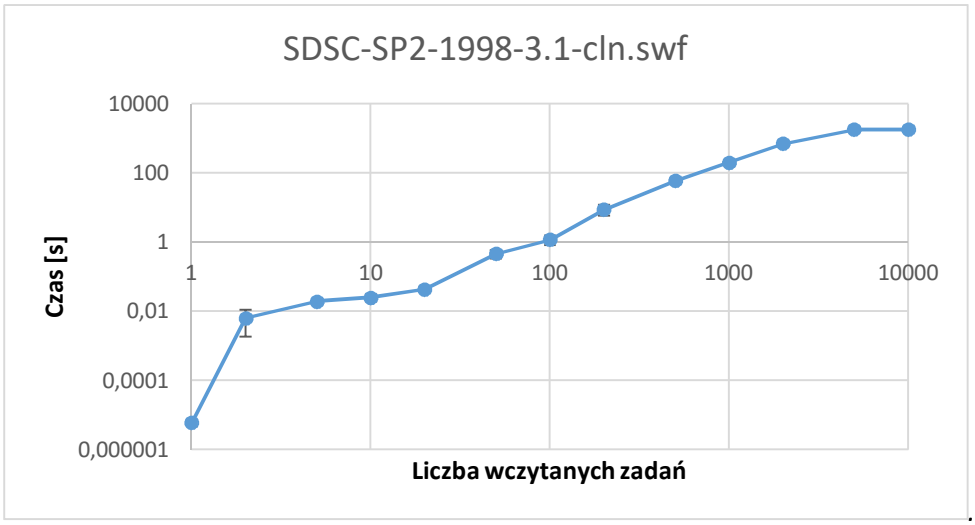
Aby uzyskać lepszy ogląd wyników dla niewielkiej liczby wczytanych zadań zdecydowaliśmy się zamieścić poniżej wykresy uzyskane na podstawie tych samych testów, ale zaprezentowane w skali logarytmicznej (o podstawie 10) dla obu osi.



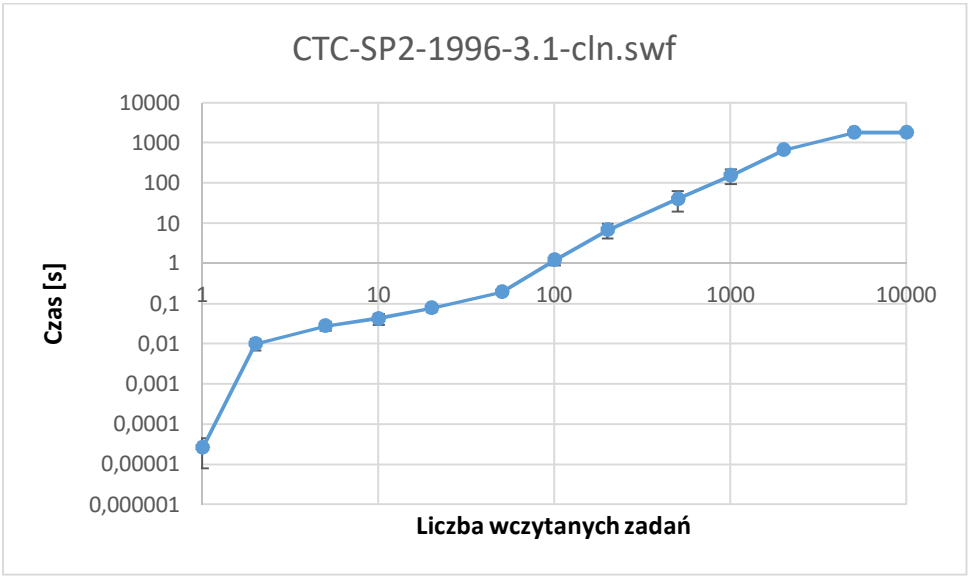
Wykres 5. Zależność czasu od liczby wczytanych zadań w skali logarytmicznej dla pliku 1.



Wykres 6. Zależność czasu od liczby wczytanych zadań w skali logarytmicznej dla pliku 2



Wykres 7. Zależność czasu od liczby wczytanych zadań w skali logarytmicznej dla pliku 3.



Wykres 8. Zależność czasu od liczby wczytanych zadań w skali logarytmicznej dla pliku 4.

W odróżnieniu do poprzedniego sprawozdania, nie dokonujemy tutaj porównania do algorytmu zachłannego. Nie miałoby to większego sensu, ponieważ był on algorytmem jednoprzebiegowym, więc jego czasy wykonania byłyby zdecydowanie krótsze. Podane wykresy dobrze nam pokazują, dla których instancji miało miejsce zatrzymanie algorytmu po przekroczeniu przyjętego maksymalnego czasu obliczeń równego 30 minut. Fakt ten będzie miał potem swoje odzwierciedlenie przy prezentowaniu jakości rozwiązań uzyskanych dla tych instancji w przypadku dużych wartości *n*, czyli liczby wczytywanych zadań.

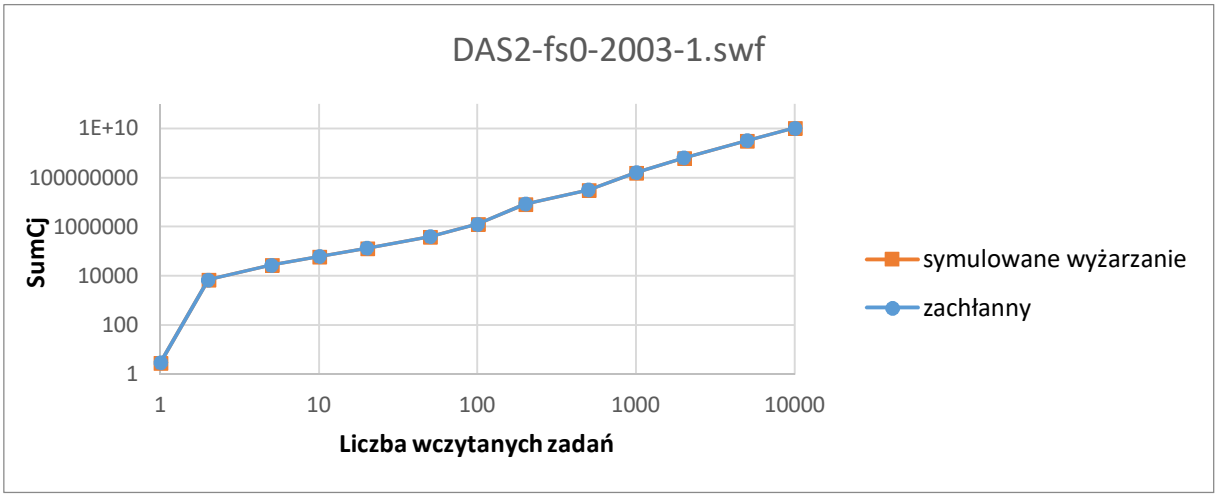
Na powyższych wykresach zostały uwzględnione odchylenia standardowe, jednak nie są one w każdym przypadku widoczne, dlatego zdecydowaliśmy się poniżej zamieścić tabelę pokazującą wartość odchylenia standardowego czasu działania algorytmu dla poszczególnych instancji testowych.

Liczba zadań	DAS2-fs0-2003-1.swf	LANL-CM5-1994-3.1-cln.swf	SDSC-SP2-1998-3.1-cln.swf	CTC-SP2-1996-3.1-cln.swf
1	0,000003	0,000001	0,000001	0,000019
2	0,00114	0,002831	0,00456	0,003235
5	0,001156	0,009433	0,002247	0,007239
10	0,002139	0,003689	0,005714	0,012856
20	0,009907	0,003903	0,007246	0,018074
50	0,01256	0,329818	0,123144	0,019492
100	0,007317	1,344118	0,364997	0,32489
200	0,009802	1,305146	3,135278	2,731348
500	0,05888	23,58013	7,159236	21,55869
1000	0,697471	63,436419	2,472707	62,929781
2000	7,034798	141,808961	5,506907	94,855508
5000	11,28973	0	0	0
10000	128,412912	0	0	0

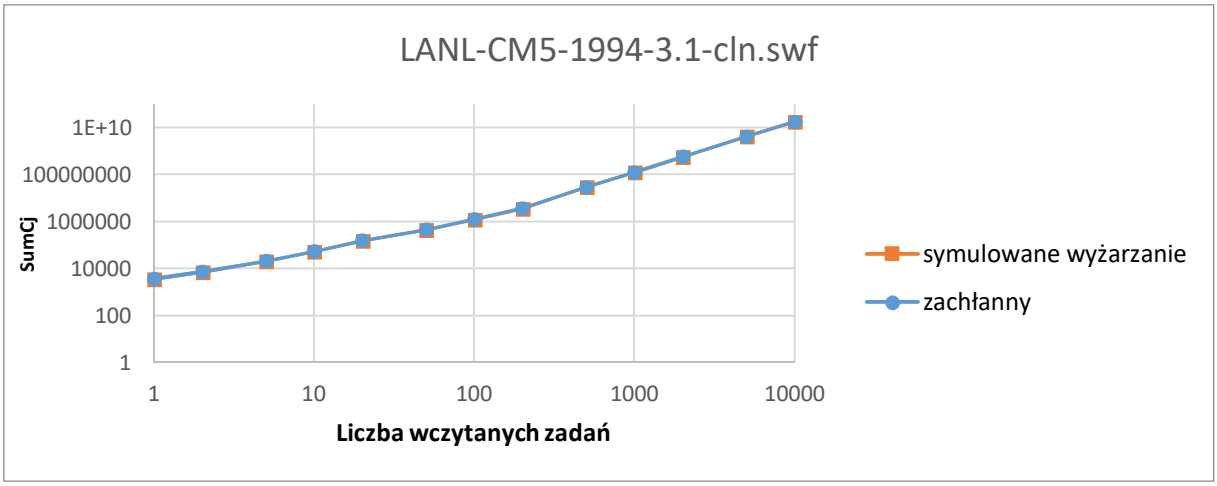
Tabela 2. Wartość odchylenia standardowego [s] odpowiednio dla plików: 1, 2, 3, 4

W plikach, w których wartość odchylenia standardowego wynosi 0, został przekroczony czas 30 minut.

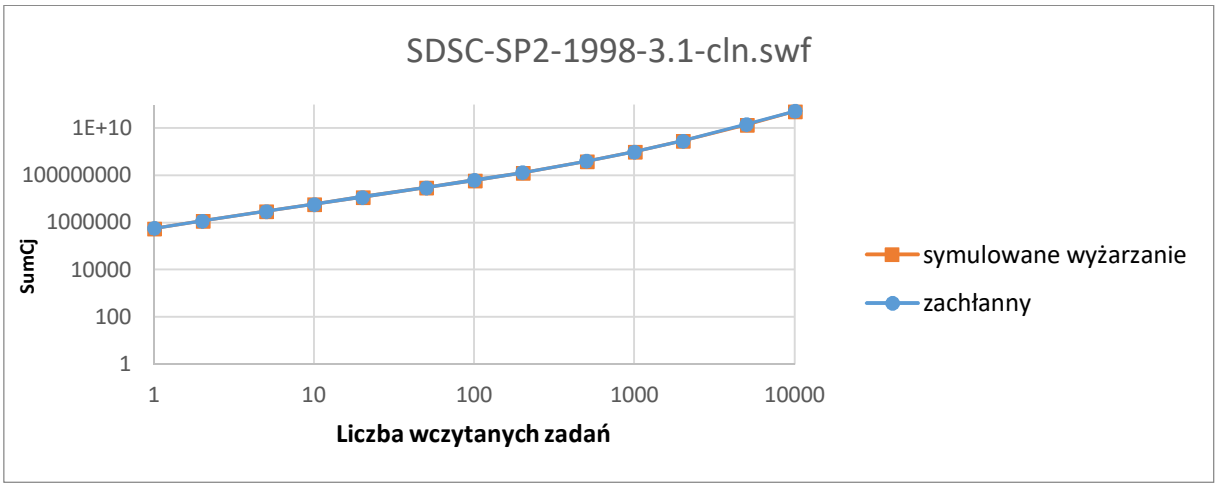
Kolejne wykresy, ilustrują związek między uzyskiwaną wartością funkcji celu, a liczbą wczytanych zadań. W celu pokazania wartości dla każdej z instancji, wykresy zostały wykonane w skali logarytmicznej.



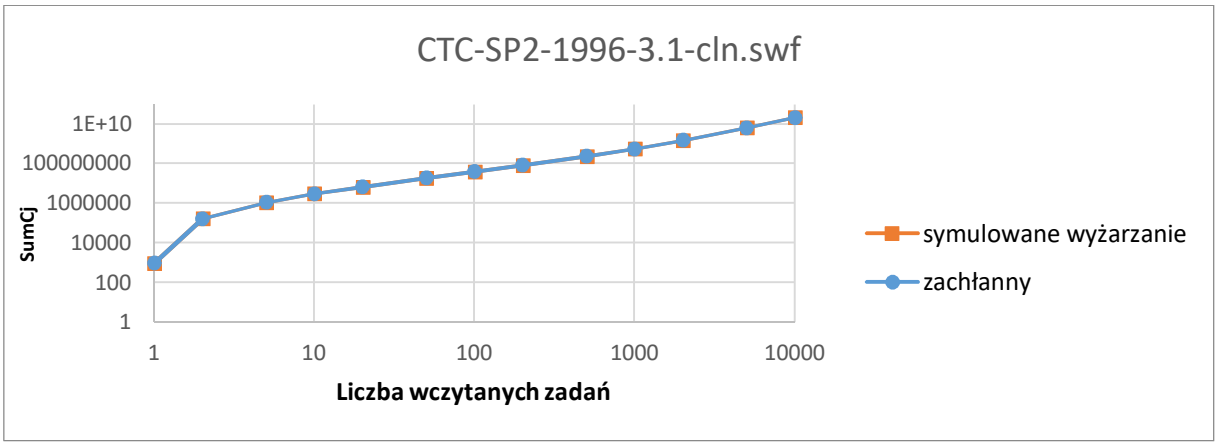
Wykres 9. Zależność wartości funkcji celu od liczby wczytanych zadań w skali logarytmicznej dla pliku 1.



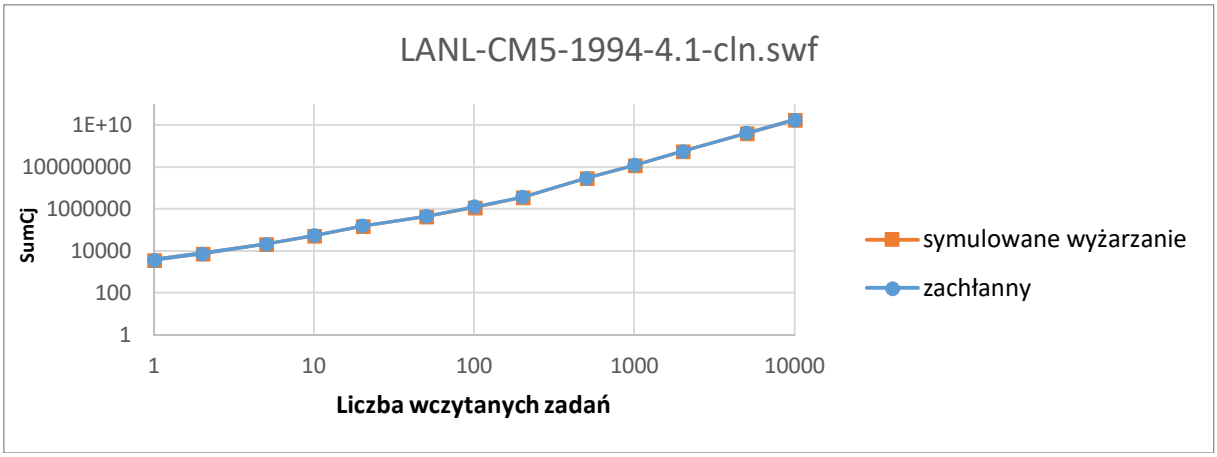
Wykres 10. Zależność wartości funkcji celu od liczby wczytanych zadań w skali logarytmicznej dla pliku 2.



Wykres 11. Zależność wartości funkcji celu od liczby wczytanych zadań w skali logarytmicznej dla pliku 3.

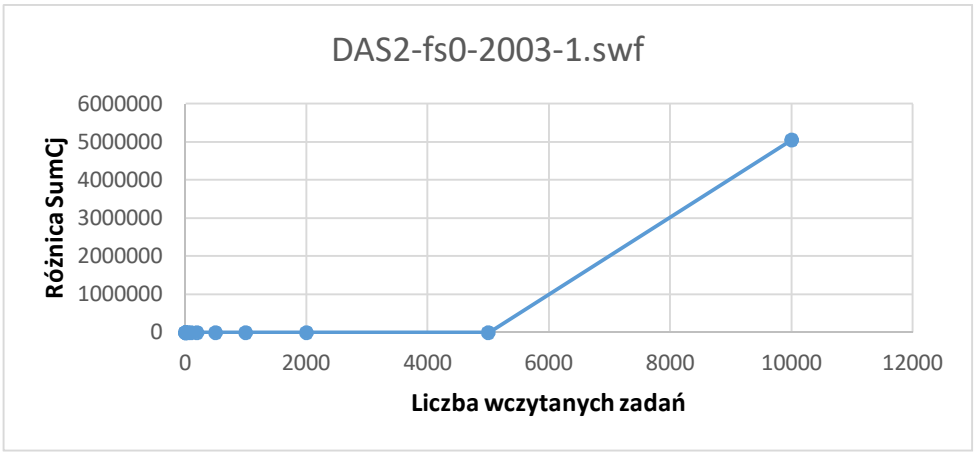


Wykres 12. Zależność wartości funkcji celu od liczby wczytanych zadań w skali logarytmicznej dla pliku 4.

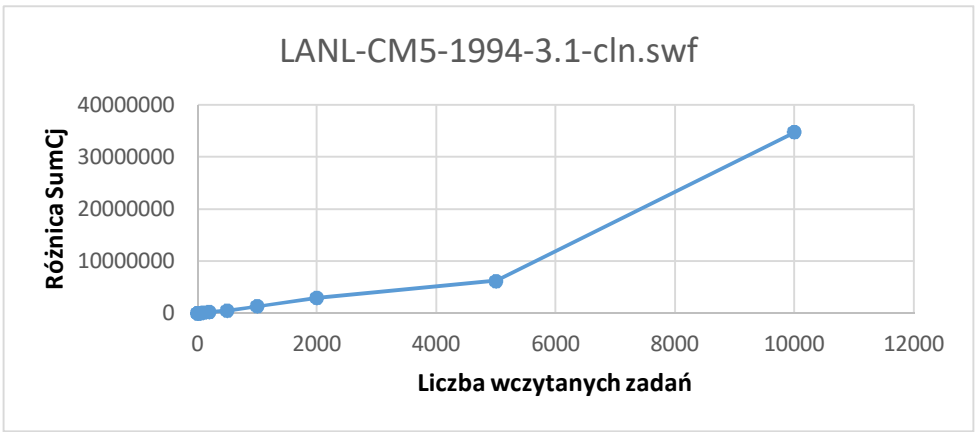


Wykres 13. Zależność wartości funkcji celu od liczby wczytanych zadań w skali logarytmicznej dla pliku 5.

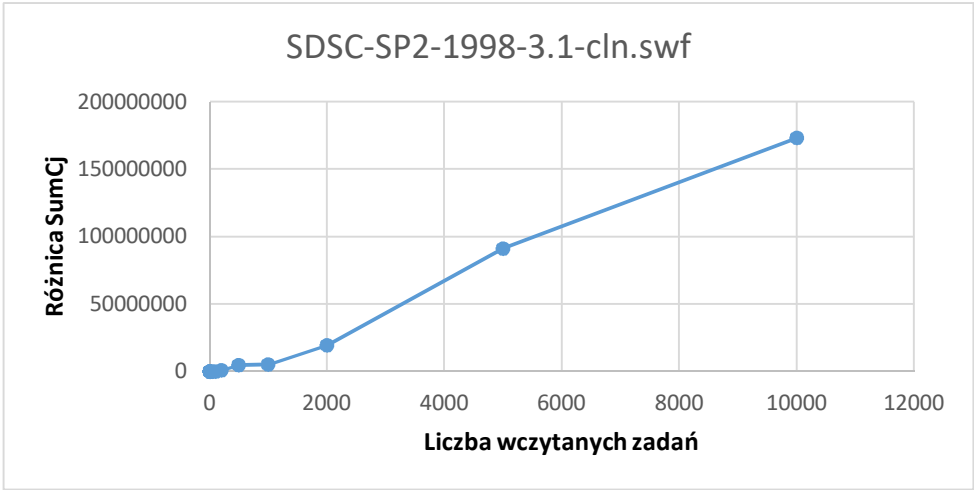
Podobnie jak w poprzednich testach, podczas porównywania algorytmu zachłannego oraz losowego, zaprezentowane powyżej wyniki mogłyby sugerować brak jakiejkolwiek poprawy. Dlatego również i w tym przypadku zdecydowaliśmy się zamieścić wykresy ilustrujące zależność różnicy wyników uzyskiwanych dla algorytmu zachłannego oraz symulowanego wyżarzania od liczby wczytanych zadań. Spodziewamy się wyników powyżej osi OX, które będą oznaczały większą efektywność (w odniesieniu do wartości funkcji celu) nowego algorytmu w porównaniu do poprzednio prezentowanego.



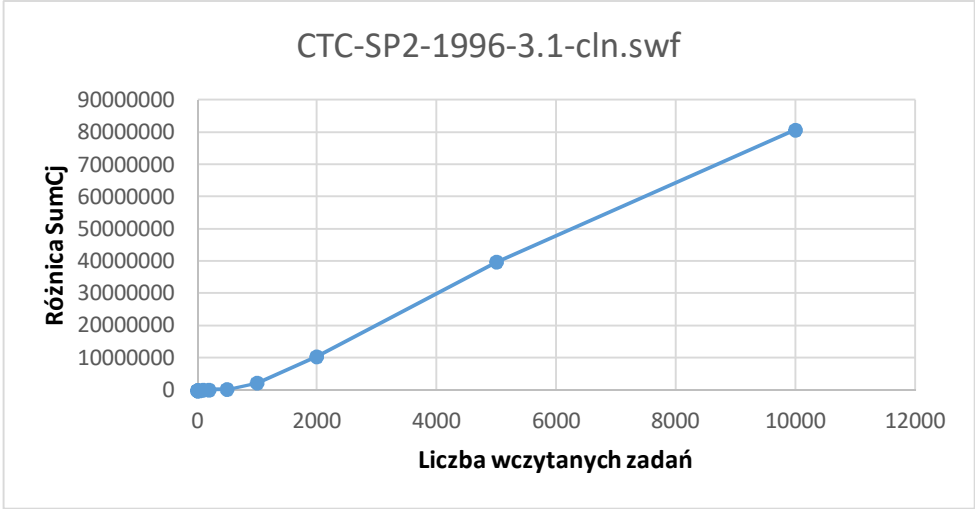
Wykres 14. Zależność różnicy wartości funkcji celu (zachłanny - symulowane wyżarzanie) od liczby wczytanych zadań dla pliku 1.



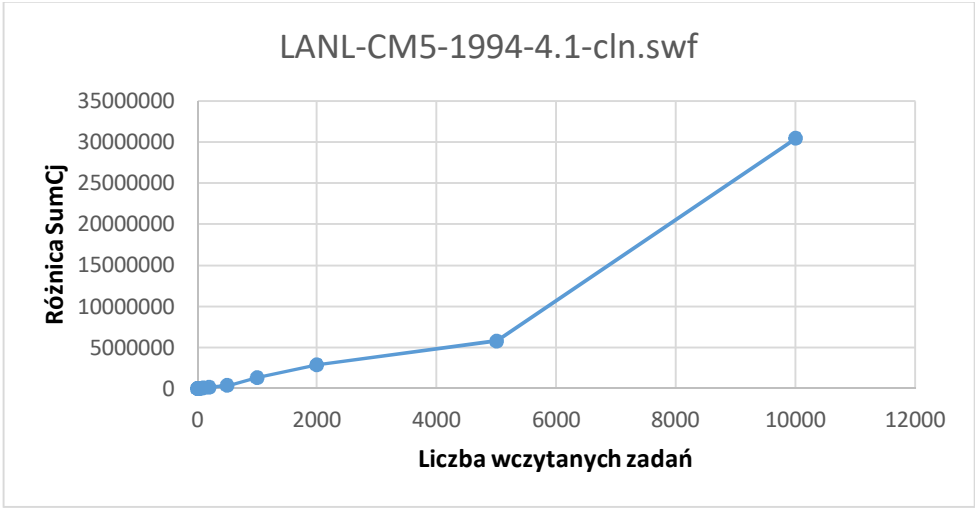
Wykres 15. Zależność różnicy wartości funkcji celu (zachłanny - symulowane wyżarzanie) od liczby wczytanych zadań dla pliku 2.



Wykres 16. Zależność różnicy wartości funkcji celu (zachłanny - symulowane wyżarzanie) od liczby wczytanych zadań dla pliku 3.



Wykres 17. Zależność różnicy wartości funkcji celu (zachłanny - symulowane wyżarzanie) od liczby wczytanych zadań dla pliku 4.



Wykres 18. Zależność różnicy wartości funkcji celu (zachłanny - symulowane wyżarzanie) od liczby wczytanych zadań dla pliku 5.

Dla lepszego zobrazowania wyników posłużyliśmy się tutaj również wszystkimi, wcześniej wspomnianymi plikami dodatkowymi. Jak można zauważyć, rezultaty są satysfakcjonujące, wykazując poprawę jakości uzyskiwanych wyników w przypadku zastosowania algorytmu symulowanego wyżarzania.

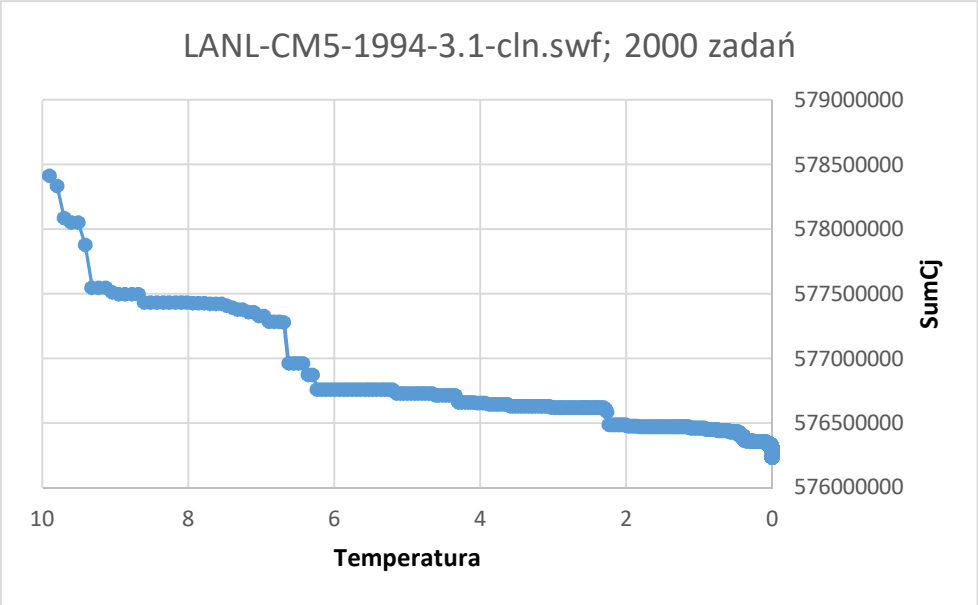
Nie wykonaliśmy tutaj wykresów w skali logarytmicznej, ponieważ dla małych instancji algorytm nie wykazuje poprawy. Wartości różnicy są równe zero. Podejrzewamy, że w przypadku użytych zadań testowych i małej liczby wczytywanych zadań oba algorytmy generują rozwiązania optymalne, jednak z powodu braku informacji o optymalnych wartościach funkcji celu jest to tylko hipoteza, która wymagałaby dalszego potwierdzenia, np. poprzez wykorzystanie algorytmów dokładnych. Sugeruje nam to, że dla podanych problemów (plików) i niewielkiej liczby wczytanych zadań lepiej jest użyć, wcześniej przez nas opracowanego, algorytmu zachłannego. Będzie on lepszym wyborem, nie ze względu na poprawę wartości funkcji celu, lecz z powodu czasu jego działania. Algorytm zachłanny wykonuje się zdecydowanie szybciej, ze względu na swoją jednoprzebiegowość.

Jeszcze jedną rzeczą, na którą chcielibyśmy zwrócić uwagę w tym teście jest wartość odchylenia standardowego dla wyników uzyskiwanych z wykorzystaniem algorytmu symulowanego wyżarzania. Obserwowany rozrzut wyników spowodowany jest elementami losowości w opracowanym przez nas algorytmie.

Liczba zadań	DAS2-fs0-2003-1.swf	LANL-CM5-1994-3.1-cln.swf	SDSC-SP2-1998-3.1-cln.swf	CTC-SP2-1996-3.1-cln.swf	LANL-CM5-1994-4.1-cln.swf
1	0	0	0	0	0
2	0	0	0	0	0
5	0	0	0	0	0
10	0	0	0	0	0
20	0	0	0	0	0
50	0	314,5811501	0	0	325,8869436
100	0	4222,648482	2389,028526	2389,028526	2027,725524
200	0	2532,053179	40426,67184	40426,67184	9897,852914
500	0	37346,41333	232081,1869	232081,1869	54775,88756
1000	0	56545,25935	959458,8426	959458,8426	36214,08439
2000	14,31083506	323086,3228	537116,0946	537116,0946	376430,5559
5000	116,7227484	208406,5139	2049982,95	2049982,95	588976,4734
10000	8434324,76	2632539,552	5482137,478	5482137,478	2709311,881

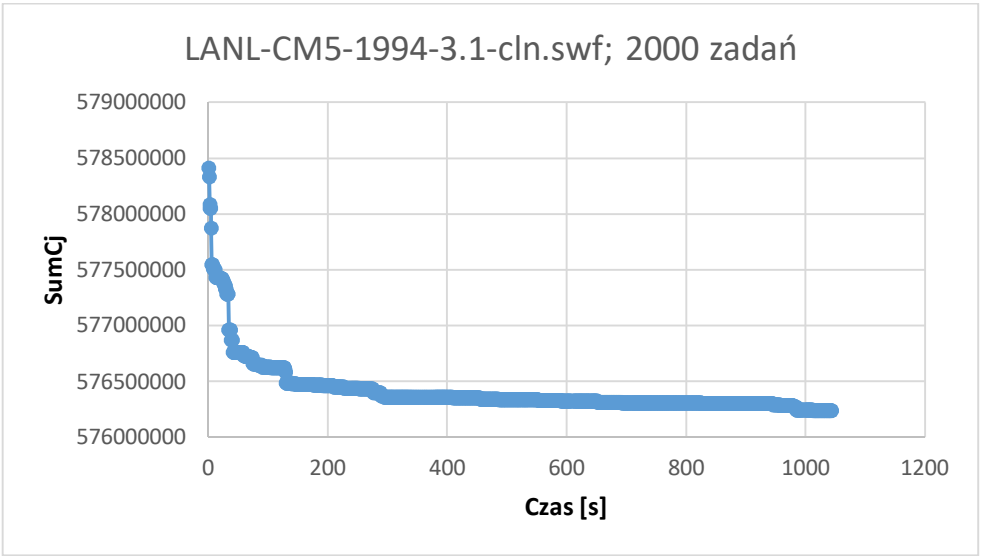
Tabela 3. Wartość odchylenia standardowego [s] odpowiednio dla plików: 1, 2, 3, 4, 5

Dodatkowo postanowiliśmy jeszcze sprawdzić, jak przebiega proces optymalizacji. Wykonaliśmy w tym celu dwa testy. W pierwszym badaliśmy, jak zmiana temperatury w kolejnych iteracjach algorytmu wpływa na uzyskiwane wartości minimalizowanej funkcji celu. Jak wiemy, temperatura jest jednym dwóch czynników, od których zależy prawdopodobieństwo akceptacji rozwiązania gorszego od poprzednio wyznaczonego. Jest to więc jeden z kluczowych parametrów, wpływających na działanie całego algorytmu. Test przeprowadziliśmy na podstawie pliku obowiązkowego *LANL-CM5-1994-3.1-cln.swfl* przy 2000 wczytywanych zadań.



Wykres 19. Zależność wartości funkcji celu od temperatury

Jak możemy zauważyć, algorytm wraz z obniżaniem się temperatury generuje coraz to lepsze wyniki. Podobną zależność można zaobserwować na wykresie będącą ilustracją ostatniego z przeprowadzonych testów, w którym badaliśmy, jak w zależności od czasu działania algorytmu zmienia się wartość funkcji celu sukcesywnie wyznaczanych coraz lepszych rozwiązań.



Wykres 20. Zależność wartości funkcji celu od czasu dla każdej iteracji

Widzimy, że w początkowych fazach działania algorytmu dość szybko znajdowane są rozwiązania o coraz niższych wartościach funkcji celu oraz, że te spadki są stosunkowo duże. Natomiast potem następuje stabilizacja jego działania, w czasie której mamy już do czynienia z lokalną optymalizacją i próbą poprawy znalezionego wcześniej rozwiązania. Zaobserwowane w dwóch ostatnich testach zależności są typowe dla algorytmu symulowanego wyżarzanie i wynikają ze sposobu jego działania.

6. Wnioski z przeprowadzonych badań

W ramach prowadzonych badań poddano analizie efektywność algorytmu symulowanego wyżarzania w porównaniu do algorytmu zachłannego. Eksperymenty obliczeniowe zostały wykonane na trzech plikach obowiązkowych oraz dwóch dodatkowych, a ich celem było porównanie wyników obu algorytmów w odniesieniu do czasu ich działania oraz uzyskiwanych wartości funkcji celu. Dodatkowo zbadano wpływ temperatury oraz czasu działania algorytmu na przebieg procesu optymalizacji.

Pierwszym krokiem było przetestowanie czasu działania algorytmu w zależności od liczby wczytanych elementów. Wyniki zostały przedstawione na wykresach, zarówno w skali liniowej, jak i logarytmicznej. Na ich podstawie można stwierdzić, że w przypadku zadań o dużym rozmiarze algorytm symulowanego wyżarzania wymaga długiego czasu działania, aby uzyskać satysfakcjonujące wyniki.

Kolejnym aspektem było porównanie „jakości" uzyskanych wyników, czyli wartości funkcji celu w zależności od liczby wczytanych zadań. Przeprowadzone testy pozwoliły stwierdzić, że dla instancji testowych o niewielkim rozmiarze oba algorytmy generują jakościowo takie same rozwiązania, natomiast wraz ze wzrostem liczby wczytywanych zadań algorytm symulowanego wyżarzania uwidaczniał już swoją przewagę nad algorytmem zachłannym.

Dodatkowo przeprowadzono testy, których celem było prześledzenie i zobrazowanie przebiegu poszukiwania rozwiązania w kolejnych iteracjach algorytmu, w zależności od temperatury oraz czasu działania. Jak można było przypuszczać wraz z upływem czasu działania algorytmu (i jednoczesnym obniżaniem się temperatury) algorytm znajduje coraz lepsze rozwiązania, przy czym na początku jest to dość dynamiczny proces, a następnie algorytm się stabilizuje. Związane jest to z faktem, że w początkowych fazach temperatura jest wysoka przez co z większym prawdopodobieństwem akceptowane są gorsze rozwiązania, co z kolei umożliwia eksplorację przestrzeni rozwiązań w poszukiwaniu obiecujących obszarów. W miarę postępu algorytmu temperatura (a co za tym idzie prawdopodobieństwo akceptacji gorzej ocenianych rozwiązań) stopniowo maleje, co sprzyja dokładniejszemu przeszukiwaniu lokalnych obszarów.

Podsumowując przeprowadzone badania nad porównaniem efektywności algorytmu zachłannego w stosunku do algorytmu symulowanego wyżarzania, możemy stwierdzić, że w sytuacjach, w których kluczowy jest czas działania algorytmu, lepszym wyborem będzie algorytm zachłanny. Działa on dużo szybciej niż algorytm symulowanego wyżarzania, generując przy tym stosunkowo dobre uszeregowania. Jak wykazały jednak testy opisane w poprzednim sprawozdaniu należy starannie dobrać kolejność, w jakiej wybierane są zadania do umieszczenia w harmonogramie, w zależności od przyjętego kryterium optymalizacji. Jeżeli natomiast priorytetem jest jakość rozwiązania, rozumiana jako wartość funkcji celu, to z dwóch przetestowanych metod optymalizacji powinniśmy zdecydować się na algorytm symulowanego wyżarzania. Szczególnie w problemach o dużym rozmiarze jego przewaga nad algorytmem zachłannym jest widoczna. Należy jednak mieć na uwadze fakt, że im większa jest liczba zadań do uszeregowania, tym czas na jakim uruchamiamy algorytm symulowanego wyżarzania powinien być dłuższy.

Literatura:

[1] Burke, E.K., Kendall, G., Whitwell, G. (2004). A new placement heuristic for the orthogonal stock-cutting problem. Operations Research, 52 (4), 655–671.

[2] Burke, E.K., Kendall, G., Whitwell, G. (2009). A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock cutting problem. INFORMS Journal on Computing 21 (3), 505–516.

[3] Kirkpatrick, S., Gelatt, Jr C. D., Vecchi, M. P. (1983), Optimization by simulated annealing, Science, 220 (4598), 671–680.

[4] Leung, S. C.H., Zhang, D., Sim, K. M. (2011). A two-stage intelligent search algorithm for the two-dimensional strip packing problem. European Journal of Operational Research, 215(1), 57-69.