Optymalizacja kombinatoryczna 2017 – projekt

Grupa III (Inf) : ŚRODA, tydzień nieparzysty **13:30-15:00**

Witold Kupś , index 127088, INF sem III dzienne

witold.kups@student.put.poznan.pl

Problem 6

Flowshop, liczba maszyn m=2, liczba zadań n,
operacje niewznawiane,
dla pierwszej maszyny k okresów przestoju o losowym czasie rozpoczęcia i trwania (określonym przez generator instancji problemu), k >= n/4,
czas gotowości dla każdej operacji nr 1 każdego zadania, nieprzekraczający połowy sumy czasów wszystkich operacji dla maszyny I
minimalizacja sumy czasów zakończenia wszystkich operacji

Wybrana metaheurystyka – ANT COLONY OPTIMIZATION.

Język – Java 8, projekt Gradle.

Wstęp

Chciałbym tutaj usystematyzować słownictwo występujące w sprawozdaniu.

W parametrach problemu występuje liczba maszyn równa 2, w związku z tym każde **zadanie** posiada **dwie operacje** wykonywane stosownie, pierwsza na **maszynie pierwszej**, oraz druga na **maszynie drugiej**, co wynika z zadanego problemu.

Jest on typu flowshop, w związku z czym operacja pierwsza każdego zadania musi wykonać się w całości na maszynie pierwszej, zanim operacja drugą będzie mogła być rozpoczęta. Ponadto, na maszynie pierwszej występują **maintenance**(lub zamiennie **przerwy techniczne**), a każda jej operacja ma parametr **Ready Time** opisany w dalszej części sprawozdania.

W tym również miejscu chciałbym zaznaczyć, że mówiąc o zadaniu ‘długie’, ‘średnie’ itd. mam na myśli *przedział czasu trwania każdej z operacji* danego zadania.

OPIS ALGORYTMU

Standardowe parametry:

It – Liczba iteracji (warunek końca) <implementacja posiada również możliwość zadania całkowitego czasu wykonania>
An – populacja mrowiska
Ev – parametr odparowania macierzy feromonów.
Sp – parametr odpowiadający ilości utrwalanych po każdej iteracji w macierzy feromonów rozwiązań.
Parametry mojej implementacji :
PER – ilość iteracji dzielących moment lokalnego wyszukiwania.
$\mathbf{P_{min}}$, $\mathbf{P_{max}}$ - minimalna i maksymalna wartość feromonów w macierzy(parametr często występujący również w innych popularnych implementacjach, jak np. Rank AS; Bullnheimer, Hartl and Strauss, 1997
Ir – szansa na zignorowanie macierzy feromonów przez mrówkę.

Ts – wielkość listy Tabu dla wyszukiwania lokalnego.

Algorytm oparty jest na idei algorytmu kolonii mrówek, w którym najlepsze uporządkowanie kształtuje się w czasie na podstawie drogi wybranej przez kolonię. W pierwszej iteracji każda mrówka zaczyna od własnego losowego rozwiązania, jednak przez następne iteracje kieruje się ona doświadczeniem zapisanym w macierzy feromonów. Na końcu każdej iteracji wybierane jest Sp (parametr opisany w dalszej części) osobliwych najlepszych dróg, które następnie są zapisywane do macierzy feromonów wedle wzoru:

$$PM[i][j]_+ = PM[i][j] + \frac{Q * I}{q}$$

gdzie:

PM - macierz feromonów lub punkt wejścia (wtedy występuje jedynie index ***j***, gdyż żadne zadanie nie było wcześniej wykonywane)

Q – Cmax pierwszego rozwiązania losowego (najlepsze rozwiązanie z pierwszej iteracji)

q – Cmax utrwalanego rozwiązania

i – id obecnie zakończonego/wykonywanego zadania

I – stała mnożenia, skalowana według rozmiaru listy zadań

j – id następnego zadania

Po każdej iteracji, jednak przed wpisaniem wynikających z niej rozwiązań, wszystkie feromony znajdujące się w macierzy są odparowywane zależnie od zadanego parametru ***evaporation rate (er)*** według następującego wzoru :

$$PM[i][j]_+ = PM[i][j] * (1 - Ev)$$

W literaturze przejawia się skłonność do stopniowania wartości parametru odparowania macierzy feromonów, jednak ze względu na ograniczoną ilość zasobów czasowych oraz dużego nakładu związanych z tym podejściem testów, postanowiłem przyjąć stałą wartość.

Następnym ważnym parametrem jest ilość feromonów zapisywanych do macierzy wraz z każdą iteracją (ilość najlepszych ścieżek) oraz populacja mrówek kolonii. Ostatni z nich wiąże się ze znacznym obciążeniem czasowym, dlatego wielkość kolonii jest ściśle powiązana z ilością iteracji. W standardowej implementacji ACO, jak i np. Min-Max AS (Stützle & Hoos, 1997) do macierzy feromonów zapisywane jest jedynie najlepsze rozwiązanie. Przeprowadziłem jednak testy dla wartości z przedziału <1,30>, które wykazały tendencję do polepszania rozwiązań dla mniejszej ilości zapisywanych w macierzy feromonów dróg, jednak większej niż 1.

By algorytm nie stał się algorytmem losowym, konieczne jest korzystanie z ww. macierzy – w mojej implementacji algorytmu macierz feromonów składa się z dwóch części – wektora punktów wejścia (**entry points**) oraz samej macierzy (**pheromones path**) symbolizującej, jako numer wiersza obecne zadanie, natomiast kolejne kolumny w danym wierszu symbolizują liczbę feromonów dla każdego zadania, które ma zostać wykonane w następnej kolejności. Aby zapobiec skupieniu się wartości wokół jednej ścieżki, w macierzy został wybrany górny oraz dolny poziom feromonów, zależny od jej rozmiarów

$$(\boldsymbol{Pmin} = {}^{rozmiar}/_{10000}, \boldsymbol{Pmax} = {}^{rozmiar}/_{10})$$

Każda komórka macierzy, której index wiersza nie pokrywa się z indexem kolumny, jest również inicjalizowana wartością

$$\boldsymbol{I} = rozmiar \,/\, 100,$$

natomiast do komórek, w których index wiersza jest równy indexowi kolumny, wpisywane jest 0 – nie są one również brane pod uwagę w trakcie wyboru ścieżki czy odparowania macierzy.

Droga mrówki tworzona jest na podstawie losowej liczby z zakresu sumy wszystkich wartości feromonów dla zadań dotychczas niewykonanych, w wierszu odpowiadającym obecnie wykonywanemu zadaniu. Jeżeli mrówka dopiero zaczyna drogę, algorytm ten jest wykonywany dla wektora punktów wejścia. Złożoność tego procesu wynosi $O(n^2)$.

Ponadto, dla zwiększenia spektrum wyszukiwania, zaimplementowany został algrytm wyszukiwania lokalnego, opartego na liście Tabu – co wyznaczoną ilość iteracji, każda mrówka bierze najlepsze dotychczas znalezione rozwiązanie, by zamienić kolejnością dwa losowe zadania. Jeśli znalezione rozwiązanie znajduje się w liście Tabu, algorytm jest powtarzany do momentu znalezienia unikatowego rozwiązania. Z powodów wydajnościowych Lista jest czyszczona po przekroczeniu zadanej wielkości zależnej od ilości iteracji, jednak sama jej implementacja pozwoliła na poprawienie rezultatów w porównaniu z początkowym losowym rozwiązaniem z poziomu 10% do blisko 30%. Pomimo znacznego wzrostu jakości, algorytm ten nie może być wykonany zbyt często (co zostało wykazane w testach) ze względu na ograniczone spektrum rozwiązań (oraz narzut pamięciowy przechowywania listy + sam czas sprawdzania, czy rozwiązanie istnieje). Również złożoność tego procesu może być nie wielomianowa (im więcej zadań w liście, tym większa szansa na odrzucenie rozwiązania, a tym samym ponowne wykonanie algorytmu), a w przypadku zbyt małej ilości zadań oraz zbyt dużej populacji mrówek, może on wpaść w nieskończoną pętle (wszystkie rozwiązania zostaną znalezione, jednak on dalej będzie szukał). Rozwiązaniem może być skalowanie listy Tabu wedle długości zadań, jednak powoduje to kolejny problem – ograniczenia sprzętowe oraz wydajnościowe, a także sam charakter jej skalowania ze względu na silnie narastającą ilość możliwych rozwiązań. Kompromisem mogłoby być wyznaczenie maksymalnego rozmiaru listy - wtedy jednak przestałaby ona być parametryzowana, lub prosty warunek stopu.

Również, dla większego spektrum rozwiązań, każda mrówka ma parametryzowaną szansę na wybór utworzenia rozwiązania losowego. Jest to argumentowane zmniejszeniem szansy na optimum lokalne, ponieważ pozwala ono na ignorowanie doświadczenia zapisanego w macierzy feromonów. Takie podejście pozwoliło na niewielką poprawę jakości rozwiązań końcowych.

Jedną z wad algorytmu mrówkowego, wspólną dla większości metaheurystyk, jest wpadanie w optimum lokalne. Za próg jego powstania przyjąłem nie polepszenie się wyniku o co najmniej 0,05% przez 100 iteracji z rzędu, pod warunkiem że do wykonania zostało więcej niż 10% iteracji – jest to sensowna wartość dla sensownej ilości iteracji (np. <2000), ponieważ przez tak niewielką ilość iteracji szansa na znalezienie lepszego od obecnie najlepszego rozwiązania dąży do zera – lepszym rozwiązaniem jest próba polepszenia obecnego rozwiązania.

Za każdym razem gdy takie optimum wystąpi, obecne rozwiązanie porównywane jest z dotychczas najlepszym – jeśli jest lepsze, staje się najlepszym. Ponadto, najlepsze dotychczas znalezione rozwiązanie jest wpisywane do macierzy feromonów - nie powoduje to ponownego wpadania w optimum lokalne (przynajmniej nie natychmiast), dzięki niezerowej wartości początkowej feromonów w macierzy, które są ponownie ustanawiane po resecie macierzy. Tym sposobem, następne iteracje wykorzystują w jakiejś części dotychczasowe doświadczenie.

Powierzchowna analiza złożoności algorytmu pozwala na wydzielenie jego najbardziej wpływających na złożoność części, w szczególności:

- tworzenie losowego rozwiązania przez każdą mrówkę, podczas którego wykorzystany jest algorytm **Fisher–Yates shuffle** o złożoności O(n). Jest on wykonywany na kopi oryginalnej listy zadań.

- tworzenie ścieżki mrówki na podstawie macierzy feromonów – złożoność $O(n^2)$

- kolekcjonowanie rozwiązań zebranych przez kolonię mrówek w trakcie iteracji - O(log(n)), oraz wybieranie z kolekcji Sp najlepszych rozwiązań O(Sp log(n)) (wykorzystanie drzewa czerwono-czarnego)

- obliczania rozwiązania na podstawie ustalonej drogi – O(mn), gdzie m jest liczbą maszyn, natomiast n liczą zadań.

Opis generatora Instancji

Generator instancji testowych jest częścią programu zawierającego algorytm – użytkownik przed rozpoczęciem pracy z algorytmem ma możliwość utworzenia nowej instancji, lub skorzystanie z jednej z dotychczas istniejących.

Generator przyjmuje za parametr kolejno długość najdłuższego zadania (długość zadań będzie liczbą całkowitą z zakresu <Min, Max> gdzie Min i Max są wartościami odpowiednio minimalnej oraz maksymalnej długości czasu trwania zadania), ilość zadań do wykonania oraz ilość maintenance'ów. W opisie problemu wystąpił parametr *Ready Time*, w związku z czym jest on częścią każdego zadania wykonywanego na pierwszej maszynie, a jego wartość nie przekracza sumy czasów trwania wszystkich operacji maszyny pierwszej.

Długość każdej przerwy jest losowana z zakresu <1, M> gdzie M = 0,05 * [suma z wartości Max(długość operacji pierwszej, długość operacji drugiej) każdego zadania] lub wartość podana przez użytkownika. Dla racjonalnej proporcji liczby przerw technicznych do całkowitej liczby zadań (1/4, 1/3 …) całkowita długość maintenance'ów nie powinna przekraczać 20% sumy czasów wszystkich operacji. Same instancje utrwalane są w dwóch formatach – JSON oraz czytelnej. Są one równoważne. Algorytm interpretuje jedynie instancje utrwalone w postaci JSON, natomiast forma czytelna jest dla ułatwienia interpretacji przez człowieka

Opis testów

Format nazw instancji oraz odpowiadających im rozwiązań :
<id>[opcjonalne]<nazwa>T<ilość zadań>tt<min. długość zadania>_<max. długość zadania>M<ilość maintenance'ów>mt<max. czas trwania maintenance lub D dla czasu ustawionego przez generator>

Testy zostały podzielone na dwie fazy:

1) Testy parametrów algorytmu – dla każdej badanej konfiguracji parametru po 20 prób, z których następnie jako wynik przyjmuję średnią wartość końcowego Cmax. Aby mieć możliwość porównania wyników oraz większą ich dokładność, postanowiłem wykorzystywać instancję składającą się ze 100 zdań, których maksymalna długość wynosiła 100, oraz 26 maintenance'ów, o całkowitym czasie trwania nie dłuższym niż 10% całkowitego czasu wszystkich operacji, oraz minimalnym odstępie czasowym równym długości trwania najdłuższej operacji na maszynie 1. Zastosowanie jedynie jednej instancji wynikało z kompromisu pomiędzy rozpoznaniem instancji (jej najlepszej wartości) a postępem i wynikami algorytmu – wiązało się to z ograniczonymi zasobami czasowymi oraz chęcią jak najprecyzyjniejszego określenia możliwości zadanej konfiguracji – dzięki temu mogłem wykonać bardzo dużą ilość testów oraz precyzyjnie określić średni rezultat dla danej konfiguracji. Wynik to ponad 5 tysięcy testów parametrów dla danej instancji, z których 3260 zostało zaprezentowanych w sprawozdaniu. Najlepszą osiągniętą wartością Cmax dla danej instancji było 680775 przy początkowym uporządkowaniu 1026961 – stanowiło to dla mnie swojego rodzaju odwołanie (znajduje się ono z danymi do sprawozdania). Sama specyfika instancji, która była dosyć ogólna (nie popadała w żadną skrajność) umożliwiała takie rozwiązanie. Jestem świadom zagrożenia wynikającego z tak ograniczonego spektrum, nie jest jednak możliwe utworzenie statycznego algorytmu działającego z tą samą precyzją dla różnych rodzajów problemów.

Z uwagi na bardzo duża ilość testów, dla ustalenia parametru odparowania macierzy oraz najlepszego stosunku iteracji do populacji mrówek, postanowiłem nie zachowywać rezultatów algorytmu, a jedynie ich całkowity przebieg próbkowany co 10 iteracji oraz uśrednione wartości Cmax dla każdej z konfiguracji testowych. Dla pozostałych parametrów końcowe rozwiązania zostały również zachowane.

Testom podlegały następujące parametry:

- stosunek liczby iteracji do populacji mrówek (It/An)

- odparowanie macierzy (Ev)

- ilość zachowywanych w macierzy feromonów po każdej iteracji rozwiązań (Sp)

- częstość procesu lokalnego wyszukiwania (PER)

- szansa na niezależność mrówki w poszukiwaniu rozwiązania (Ir)

2) Testy dla instancji

Zadania jak i przerwy techniczne podzieliłem na 3 klasy :

Długie	100-150
Średnie	50-100
Krótkie	1-50

Testy zostały przeprowadzone dla kombinacji zadań długich, średnich, krótkich, a także mieszanych w postaci krótkie-średnie, średnie-długie oraz w całym zakresie, tzn. <1,150>

Ponadto, przeprowadziłem testy porównujące dwie instancje o takich samych parametrach zadań, jednak innych parametrach przerw (dla wielu zadań o krótkim czasie trwania).

Zbadany został również wpływ wielkości instancji testowej (ogólnie ilości zadań) na procentową poprawę jakości rozwiązania w porównaniu do rozwiązania powstałego w pierwszej iteracji (które jest losowe).

Ostatni przeprowadzony test dotyczy porównania dwóch skrajnych przypadków – wielu zadań krótkich oraz nie dużej ilości zadań długich.

Wyniki dla poszczególnych testów wraz z dotyczącymi ich instancjami zostaną zapisane w formie wymaganej przez prowadzącego w oddzielnych folderach sygnowanych numerem wykresu oraz krótkim opisem.

Testy Parametrów Algorytmu

W pierwszej kolejności zostały przeprowadzone testy na jakość rozwiązania dla bardzo małych wartości iteracji/populacji mrówek w zależności od współczynnika odparowania macierzy, oraz dla rzeczywistych konfiguracji testowych. Równoważność parametrów kwalifikowana była podobnym czasem wykonania, który dla próby szybkiej przyjąłem 10 sec, natomiast dla próby długiej około 3 minuty. Dla każdej konfiguracji zostało przeprowadzonych 20 prób, z których następnie została wyciągnięta średnia wartość. Oprócz niej w tabelach znajdują się również średnie czasy wykonania (mTime) liczone w sekundach, najlepszy pojedynczy wynik dla zadanej konfiguracji It/An oraz średnią wartość Cmax dla danej konfiguracji It/An zależnej od parametru Ev.

Inne parametry przyjęte za stałe w teście (przynajmniej na potrzeby innych testów dopóki same nie zostaną przetestowane)

Ts - 30000	**Sp – 1(ustalone)**
Ir – 0%	**PER – 10**
Pmin, Pmax – jak wyżej (ustalone)	

Pierwsze wykonane testy dotyczyły proporcji iteracji do populacji mrówek oraz odpowiadającej im wartości współczynnika odparowania macierzy feromonów.

Próba szybka – około 10 sec wykonania

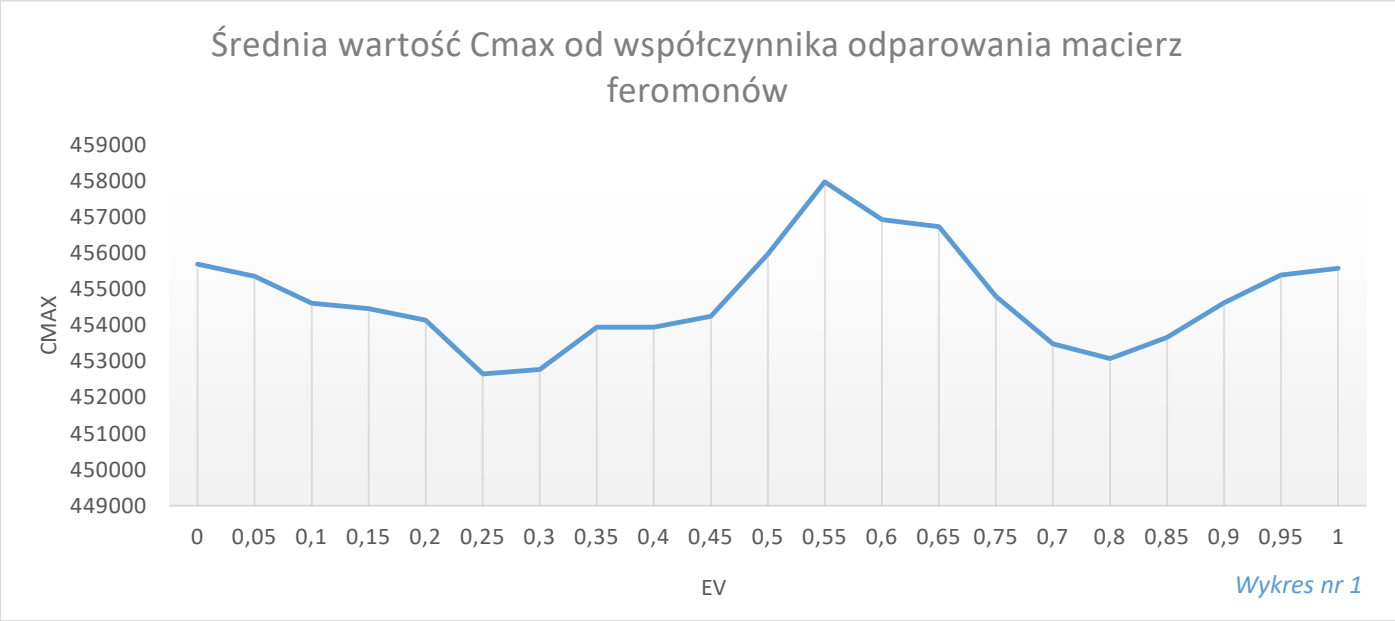
params	**0,4**	**0,35**	**0,3**	**0,25**	**0,2**	**0,15**	**0,1**	**0,05**	**mTime [s]**	**best**	**min**
It10 An1000	974426,2	972196,5	970478,7	972668,3	976253,4	982659,1	972888,4	963353,5	7	906793	963353,5
It20 An500	871461	879507	864401,6	873567,1	877245,9	866128,5	866738,4	874790,6	8,875	818170	864401,6
It50 An200	851095,2	848790,8	843676,5	850163,3	850443,3	853119,7	853959,1	847704,6	10	776941	843676,5
It100 An100	847798,8	839675,4	854091,6	836922,1	849936,8	843188	845653,2	843268,3	11,3	779042	836922,1
It200 An50	840974,8	838341,4	839739,6	847933,4	840602,2	850483,4	850013,7	846121,5	11,75	789972	838341,4
It500 An20	841672,4	848060,3	840871,9	842148,2	848872,1	842324,9	854794,5	847342,7	12,8	770978	840871,9
It1000 An10	841103	834110	854127,5	842349,2	849205,7	842728,6	844600,8	849944,9	12,6	782091	834110
min	840974,8	834110	839739,6	836922,1	840602,2	842324,9	844600,8	843268,3			

Próba długa – czas około 3 minuty

params	**0,4**	**0,35**	**0,3**	**0,25**	**0,2**	**0,15**	**0,1**	**0,05**	**mTime [s]**	**best**	**min**
It2000 An500	704491,4	703746,9	704011,5	702887,1	702139,7	705369,2	704315	705319	171	684008	702139,7
It1000 An1000	704398,8	702738,9	703076,7	703791,4	701767,2	703720,4	701321,3	705309,2	162,5	688169	701321,3
It100 An10000	710233,8	708662,3	706803	705675,8	708102,3	708036,1	705803,9	709452,3	161,1	684009	705675,8
It500 An2000	705402,7	704593	703846,6	707656	704183,7	710743,4	705116,1	704564,1	160,7	686863	703846,6
It5000 An200	708599,6	707796,2	706122,3	705845,5	707827,4	707473,6	707387,7	707791,5	162,7	692876	705845,5
min	704398,8	702738,9	703076,7	702887,1	701767,2	703720,4	701321,3	704564,1			

Jak widać, różnice na poziomie jednej konfiguracji iteracja/populacja mrówek dla różnych współczynników odparowania są znikome, i wynoszą w najgorszym przypadku dla długiej próby maksymalnie 1% (dla It500An2000), w większości około 0,5%, natomiast dla krótkiej maksymalnie 2,3% (It1000An10), zazwyczaj 1,7%. Ze względu na najlepszą średnią wyników najbardziej obiecującą konfiguracją wydaje się być ta dla równoważąca populację mrowiska z ilością iteracji, co jest wspólne zarówno dla próby długiej, jak i krótkiej. Jednak badając inne rozwiązania, większa ilość iteracji przekłada się na rozwiązania lepsze niż dla odwrotnej konfiguracji, dlatego też w testach instancyjnych postanowiłem przeprowadzać standardowo 2000 iteracji dla populacji mrówek wynoszącej 1000.

W celu szerszego spojrzenia na problem postanowiłem przeprowadzić również dodatkowe testy dla 10 różnych instancji jednakowego typu (100 zadań, maksymalna długość 100, 26 maintenance o maksymalnym czasie trwania 50), a następnie przedstawić końcowe wartości funkcji celu dla najbardziej obiecującej konfiguracji It1000An1000. Uśrednione wyniki znajdują się poniżej.



Wykres nr 1

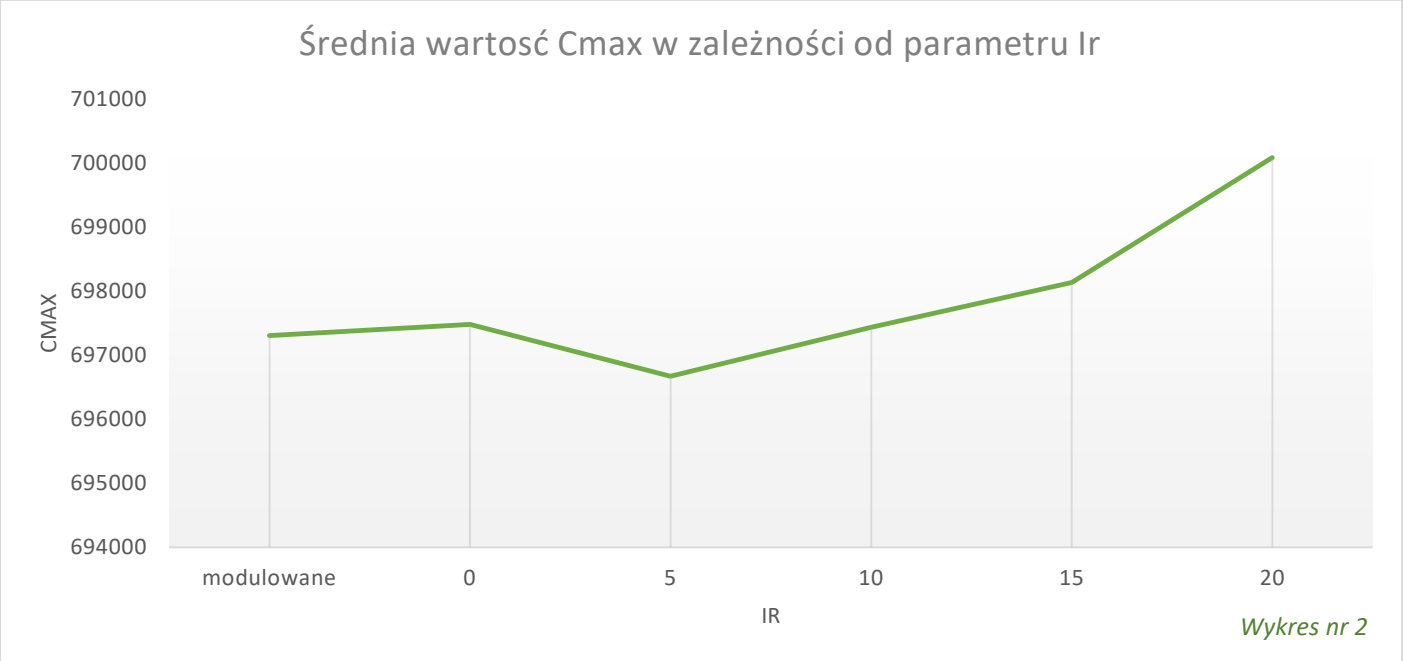
Widać, że tendencja zauważona dla danych w tabeli pokrywa się z zachowaniem wykresu. Zastanawiający może być jednak charakter krzywej dla wartości odparowania macierzy 0.8, która również stanowi minimum lokalne. Ważne jest jednak to, że większa wartość współczynnika odparowania macierzy pociąga za sobą mniejsza pamięć o dotychczasowych rozwiązaniach, a tym samym zwiększa szansa na rozwiązanie rosowe. Wtedy jakość rozwiązania możemy zawdzięczać jedynie przypadkowi oraz temu, że przechowywane jest najlepsze dotychczas znalezione rozwiązanie, które jest również systematycznie polepszane przez algorytm lokalnego przeszukiwania – może on również zaburzać odczyt, jednak jego wpływ na algorytm zostanie zaprezentowany później.

Przeprowadziłem również testy dla szansy na niezależność mrówki (tj. że nie skorzysta ona z macierzy feromonów) dla wartości modulowanej wartości parametru IR, których uśrednione wyniki znajdują się poniżej:

params	**0,4**	**0,35**	**0,3**	**0,25**	**0,2**	**0,15**	**0,1**	**0,05**	**mTime [s]**	**best**	**min**
It2000 An500	706315,8	703332,05	702889,1	700951,2	702441,1	705982,5	705915,9	703664	376	683525	700951,2
It1000 An1000	705884,6	703626,6	702610,1	701893,3	705940,85	703165,6	702894,1	705223	390	685044	701893,3
It100 An10000	706369,9	710164,6	709499,3	703877,3	708347,2	709209,7	709719,6	707195,1	375	685606	703877,3
It500 An2000	705929,05	705298	700791,9	702052,8	706128,55	704691	703620,4	703950,85	385	688248	700791,9
It5000 An200	703429,6	705930,6	706033,4	706164,9	706852,3	706133,9	702475,9	705998,6	381,5	688919	702475,9
min	705884,6	703626,6	700791,9	700951,2	702441,1	703165,6	702894,1	703950,85			

Widać poprawę rezultatów dla większości konfiguracji, ponadto końcowa wartość funkcji celu przesunęła się nieznacznie w stronę większych wartości odparowania macierzy. Jest to prawdopodobnie powiązane z większym wpływem losowości cząstkowego rozwiązania na końcową wartość funkcji celu, lub mówiąc krótko – zmniejszenie wpływu tego parametru na przebieg algorytmu.

Poniżej znajduje się również zestawienie innych wartości parametru Ir oraz wykres prezentujący zależność:

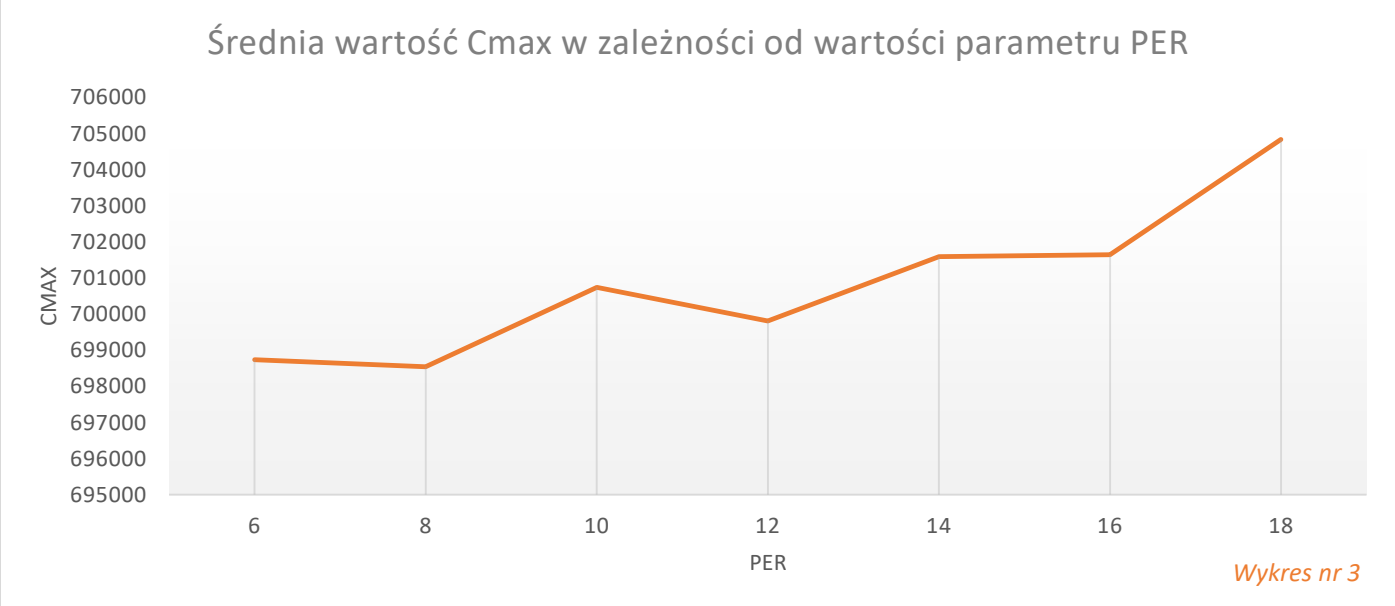


Wykres nr 2

	modulowane	**0**	**5**	**10**	**15**	**20**
Średnie Cmax	697303,1	697480	696671,4	697436,6	698133,2	700081,9
Średni czas wykonania	538,95	563,45	556,2	550,15	543,05	540,4
Najlepszy rezultat	682977	688310	681187	687696	689592	689558

Za modulowanie parametru przyjąłem zmienianie się wartości parametru Ir na przestrzeni 100 iteracji poczynając od 50% (z wyjątkiem pierwszego cyklu dla którego wartość ta wynosi 70%) gwałtownie malejąc do 0, powtarzane okresowo. Przyniosło to jednak niewielki wzrost wydajności oraz jakości rozwiązania końcowego. Z tego powodu najlepszą wartością Ir jest 5, co objawia się również w najlepszym znalezionym rozwiązaniu.

Następnie dla konfiguracji It500An2000Ev0,25 przeprowadziłem testy na parametr odpowiadający za częstość lokalnego wyszukiwania (PER):



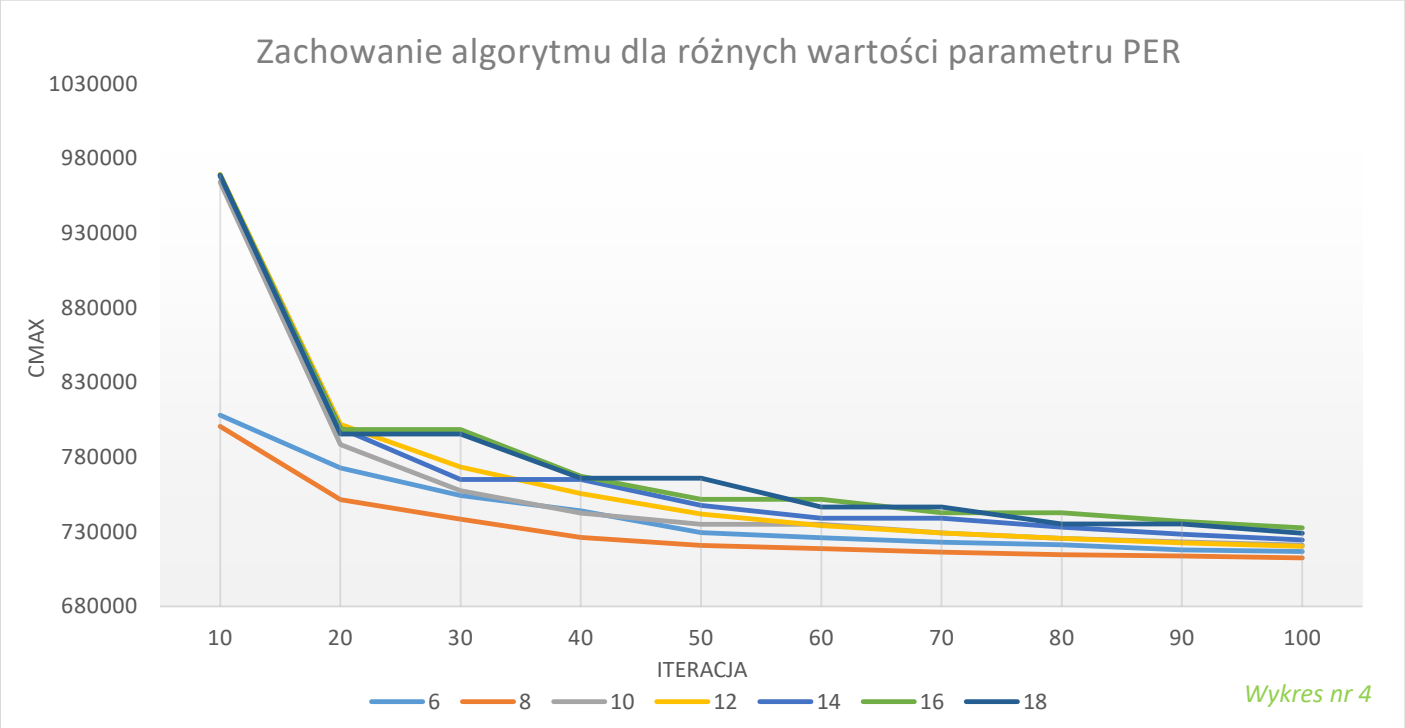
Wykres nr 3

It500An2000Ev0,25	6	8	10	12	14	16	18
Średnie Cmax	698740,2	698543,6	700743,5	699809,1	701588,8	701644,1	704837,5
Średni czas wykonania	575	567,4	562,35	556,95	557,45	557,25	551,8
Najlepszy rezultat	682614	686508	683674	688473	695233	691678	691274

Najlepszą ilością iteracji dzielącą wykonania algorytmu lokalnego przeszukiwania okazała się wartość 8.

W tabeli umieściłem również średni czas wykonania (zawyżony ze względu na wykonanie na wielu wątkach, a tym samym konieczność dzielenia pamięci RAM przez JVM, co również wiązało się z częstszymi cyklami garbage collectora – proporcje jednak zostają podobne).

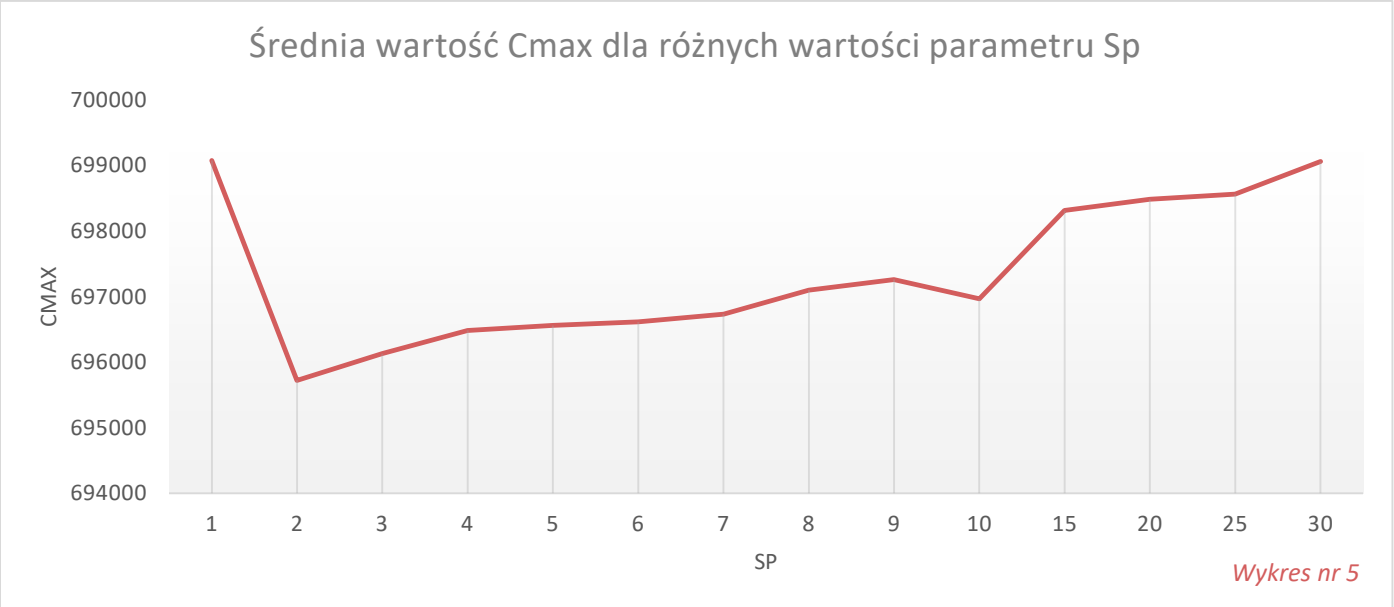
Poniżej przedstawiony jest uśredniony początkowy przebieg algorytmu dla zadanych parametrów, gdzie zmienną jest parametr PER.



Wykres nr 4

Wykres celowo został ograniczony do 100 iteracji, aby możliwe było łatwiejsze zauważenie wpływu parametru na działanie algorytmu. Widzimy na nim wpływ lokalnego poszukiwania na wynik algorytmu – dla parametru PER == 16 czy 18 można zauważyć, że jakość ulega znacznej poprawie po iteracji 20 oraz 40 (próbkowanie co 10 iteracji). To samo można stwierdzić dla parametru PER == 12 lub 14 po 30 iteracji. Najlepszy wynik jednak utrzymał się dla wartości parametru równej 8, co również odzwierciedlone jest w tabeli.

W następnej kolejności zostały przeprowadzone testy na ilość zapisywanych do macierzy feromonów rozwiązań, których wynik jest następujacy:



Wykres nr 5

	1	**2**	**3**	**4**	**5**	**6**	**7**
Średnie Cmax	699078,6	695723,7	696130,3	696485,5	696561,5	696613,9	696733,4
Najlepszy rezultat	692678	683132	682757	689415	684688	687995	684403

	8	**9**	**10**	**15**	**20**	**25**	**30**
Średnie Cmax	697100,6	697262,6	696966,9	698314,8	698484,7	698565,2	699060,6
Najlepszy rezultat	685244	687628	687767	688362	690005	691199	689634

Widać, że najlepsza jakość rozwiązania przypada dla parametru Sp równego 2. Najmniejszy wzrost wartości Cmax dla rozwiązań tworzonych w konfiguracji z parametrem Sp == 1 można uzasadnić wąskim spektrum poszukiwań, przy czym warto jednak wspomnieć o wysokim parametrze odparowania macierzy. Testy bowiem zostały wykonane dla konfiguracji It2000 An1000 **Ev0,75** PER8 IR5 – pomimo, że testy na ten współczynnik były wykonywane przy założeniu wartości parametru Sp == 1, w celu głębszej analizy należałoby je ponowić dla jego nowej wartości. Zaskakująca jest jednak różnica pomiędzy średnimi jakościami rozwiązań dla parametrów Sp == 1 oraz 2, choć można to argumentować stosunkowo małym spektrum rozwiązań utrwalonych w macierzy (szczególnie biorąc pod uwagę współczynnik odparowania) dla parametru o wartości 1, oraz ich dwukrotne zwiększenie (a tym samym szansę na zdwojenie wartości feromonu dla zadania zajmującego takie samo miejsce w uporządkowaniu dla obu rezultatów) po jego zmianie, nie powodujące jednak uogólnienia ścieżki czy zbytniego premiowania najlepszego rozwiązania .

Testy dla Instancji.

Testy zostały przeprowadzone dla wcześniej wymienionych rodzajów instancji w różnych konfiguracjach(każdy test składa się na 10 różnych instancji o tych samych parametrach wejściowych generatora).

Parametry algorytmu są następujące:

It (ilość iteracji) – 2000, chyba że ich ilość jest wyraźnie podkreślona dla danego testu

An (populacja mrowiska) – 1000

PER (częstość lokalnego wyszukiwania) – 8

Ts (maksymalny rozmiar listy Tabu) – min(max(3% * It * An, An), 30000)

Ir (stopień niezależności mrówki) – 5%.

Ev (odparowanie macierzy) – 0.25

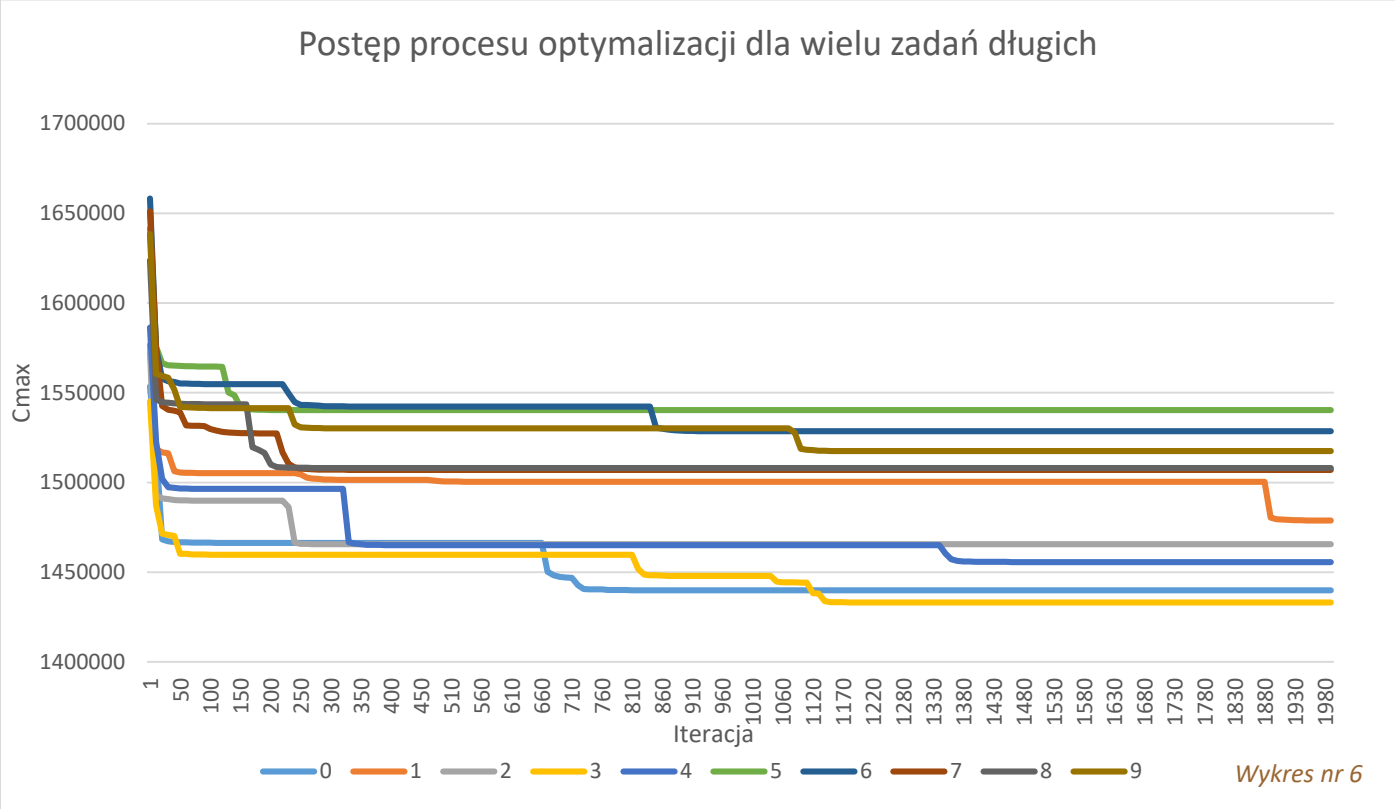
Sp (ilośc utrwalanych w macierzy rozwiązań) – 2

$\mathbf{P_{min}}$, $\mathbf{P_{max}}$ - zależne od ilości zadań

W pierwszej kolejności zostało przetestowane zachowanie algorytmu dla różnych długości zadań, następnie wpływ różnego rodzaju przerw technicznych na wynik końcowy oraz optymalizacje na przestrzeni iteracji. Ostatnie wykonane testy dotyczyły ścisłego wpływu ilości zadań w instancji na końcowy procentowy wzrost wartości Cmax w porównaniu do rozwiązania znalezionego w pierwszej iteracji. Porównane zostały również rezultaty dla instancji o zbliżonych sumach czasów operacji, jednak o skrajnie innych charakterach – instancje składające się z dużej ilości krótkich zadań, oraz instancje złożone z niewielu długich.

1) Zachowanie algorytmu dla zadań o różnej długości (dla 100 zadań oraz 25 maintenance'ów o maksymalnej długości 100)

a) *Zadania długie*

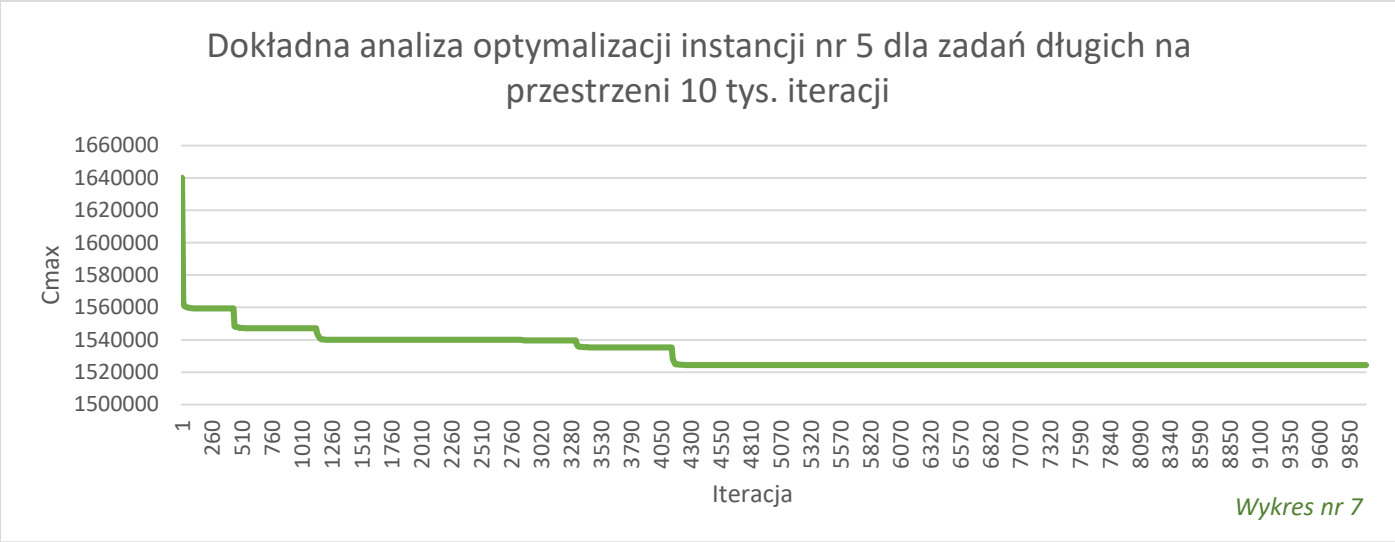


Wykres nr 6

	min	max
Numer instancji	5	4
Optymalizacja uszeregowania	6,17%	8,24%
Średnia	7,31%	
Mediana	7,29%	

Z wykresu jasno wynika, że największe polepszenie jakości następuje przez pierwsze 50 iteracji, po czym do około 350 ulega poprawie. Przez następne iteracje większość rozwiązań znajduje się w stagnacji, co wynika z napotkania optimum lokalnego. Rozwiązaniem może być modulowanie warunku jego rozpoznawania, lub inne zachowanie w momencie jego wystąpienia (np. mutację najlepszego rozwiązania na szeroką skalę oraz skalowanie macierzy feromonów). Ciekawym jest natomiast gwałtowny wzrost jakości rozwiązania pod koniec działania algorytmu dla instancji nr 1.

Instancja nr 5 cechowała się najmniejszą poprawą wartości Cmax, dlatego też postanowiłem dla niej ponowić test, tym razem dla liczby iteracji równej 10000. Jego wynik znajduje się poniżej:



Wykres nr 7

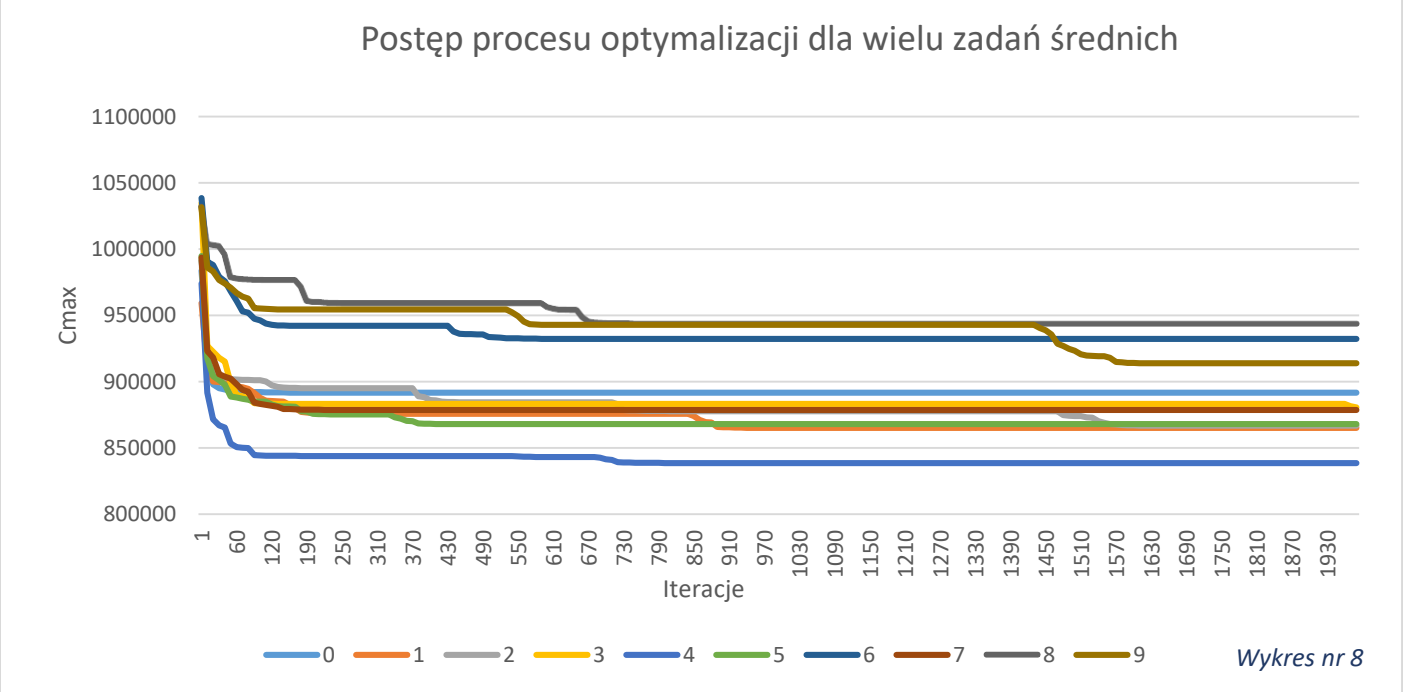
	It2000	**It10000**
Jakość uporządkowania początkowego	1641621	1640054
Jakość końcowa	1540279	1524460
Ilość Idle i ich długość (M1)	25, 460	25, 403
Ilość Idle i ich długość (M2)	50, 1886	53, 1833

Jak widać, dla tysięcznej iteracji wynik jest zbliżony do wyniku z testu masowego, jednak rozwiązanie w ciągu dalszym podlega optymalizacji – ustaje ono dopiero po 4100 iteracji z wynikiem poprawionym o 1.03% w porównaniu do rezultatu z próby dla 2000 iteracji.

Dla badanej powyżej Instancji wystąpiło 25 maintenance'ów (czyli jak w każdej) o łącznym czasie trwania 1289.

Po analizie innych rozwiązań, dla których łączny czas idle na maszynie nr 1 wynosił < 100 wnioskuję, że powyższa sytuacja wynika ze specyficznego czasu trwania zadań w danej instancji.

b) *Zadania średnie*

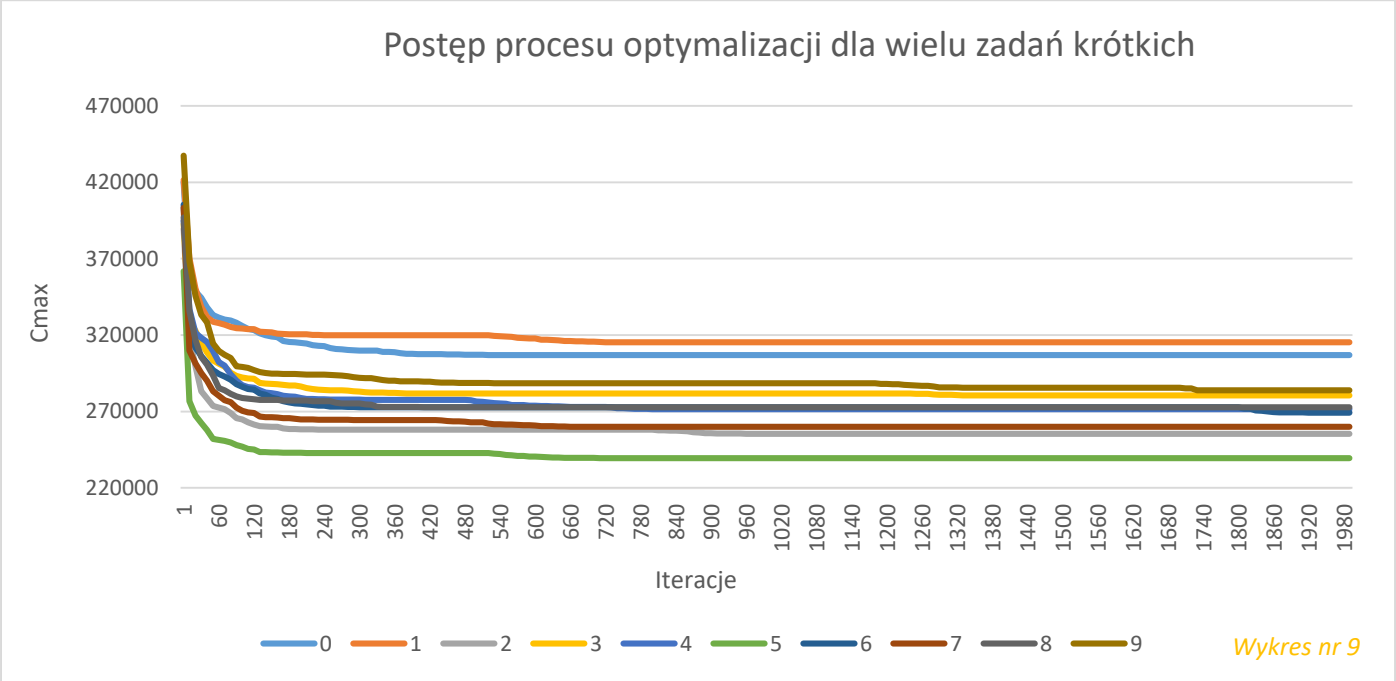


Wykres nr 8

	min	max
Numer instancji	8	3
Optymalizacja uszeregowania	8,54%	14,89%
Średnia	11,50%	
Mediana	11,49%	

Charakter wykresu nie zmienił się w znacznym stopniu, zaobserwować z pewnością można mniejszą agresję spadku wykresu przez pierwsze 50 iteracji – wynika to ze zmiany zakresu czasu trwania zadania. Możemy również zauważyć znaczną poprawę jakości uszeregowania końcowego w stosunku do jego początkowej wartości, jednak i większy rozstrzał tych wartości.

c) *Zadania krótkie*

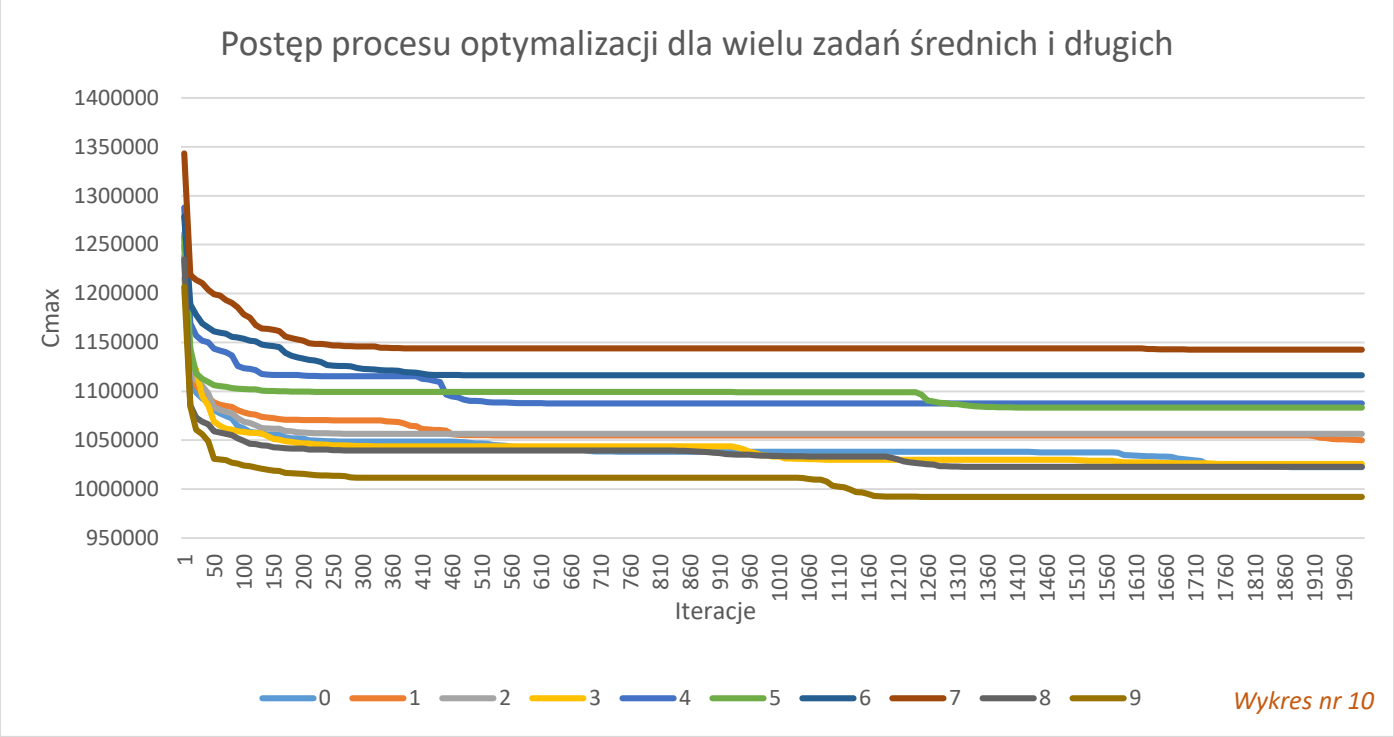


Wykres nr 9

	min	max
Numer instancji	1	7
Optymalizacja uszeregowania	25,21%	35,52%
Średnia	31,56%	
Mediana	31,56%	

Dla krótszych zadań algorytm przez pierwsze 100 iteracji dokonuje znacznej poprawy, by przez kolejne jedynie nieznacznie polepszać rozwiązanie – szczególnie widzimy to na przykładzie instancji nr 6 czy nr 2, które do samego końca ulegają polepszeniu. Warto jednak zauważyć, że po wyjściu z okresu stagnacji, jakości rozwiązania ulega jedynie nieznacznej poprawie. Ponownie następuje większy wzrost jakości końcowego uszeregowania w stosunku do jego początkowej wartości, w porównaniu do zadań o krótszym czasie trwania. Konsekwencją tego jest również zwiększenie dyspersji.

d) *Zadania składające się z operacji o średnim i długim czasie trwania*

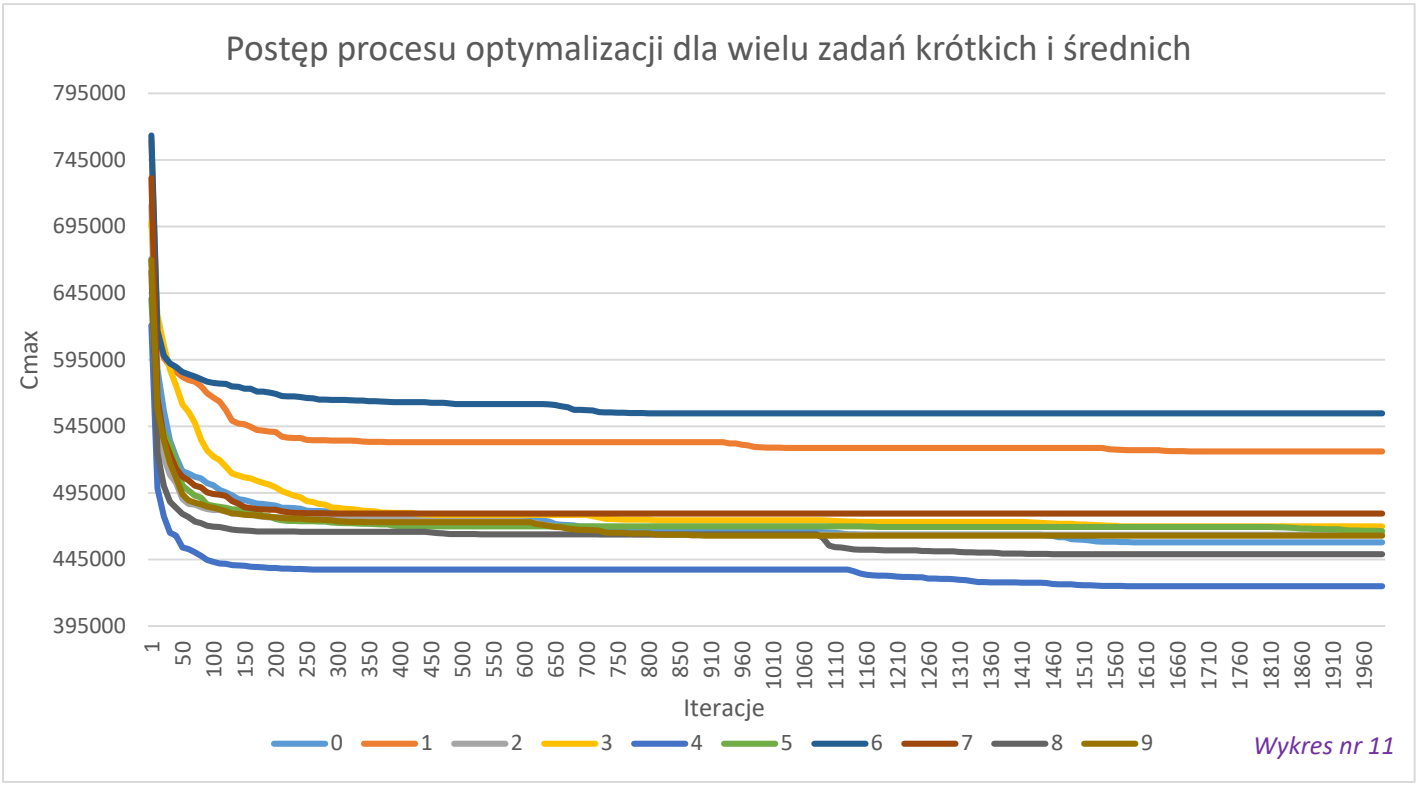


Wykres nr 10

	min	max
Numer instancji	6	0
Optymalizacja uszeregowania	12,69%	18,11%
Średnia	15,72%	
Mediana	15,58%	

Wykres ten różni się od wcześniejszych jednym interesującym aspektem – momenty stagnacji są rzadsze, a polepszanie rozwiązania zachodzi przez większą ilość iteracji z rzędu, skutkując również lepszym efektem (przykładowo instancja nr 9 czy 0). Warto również zauważyć, że pomimo zadań o długości trwania operacji z zakresu <50,150> procentowa poprawa rozwiązań wynosi znacznie więcej niż dla instancji składających się jedynie z zadań średnich czy długich.

e) *Zadania o krótkim oraz średnim czasie trwania*

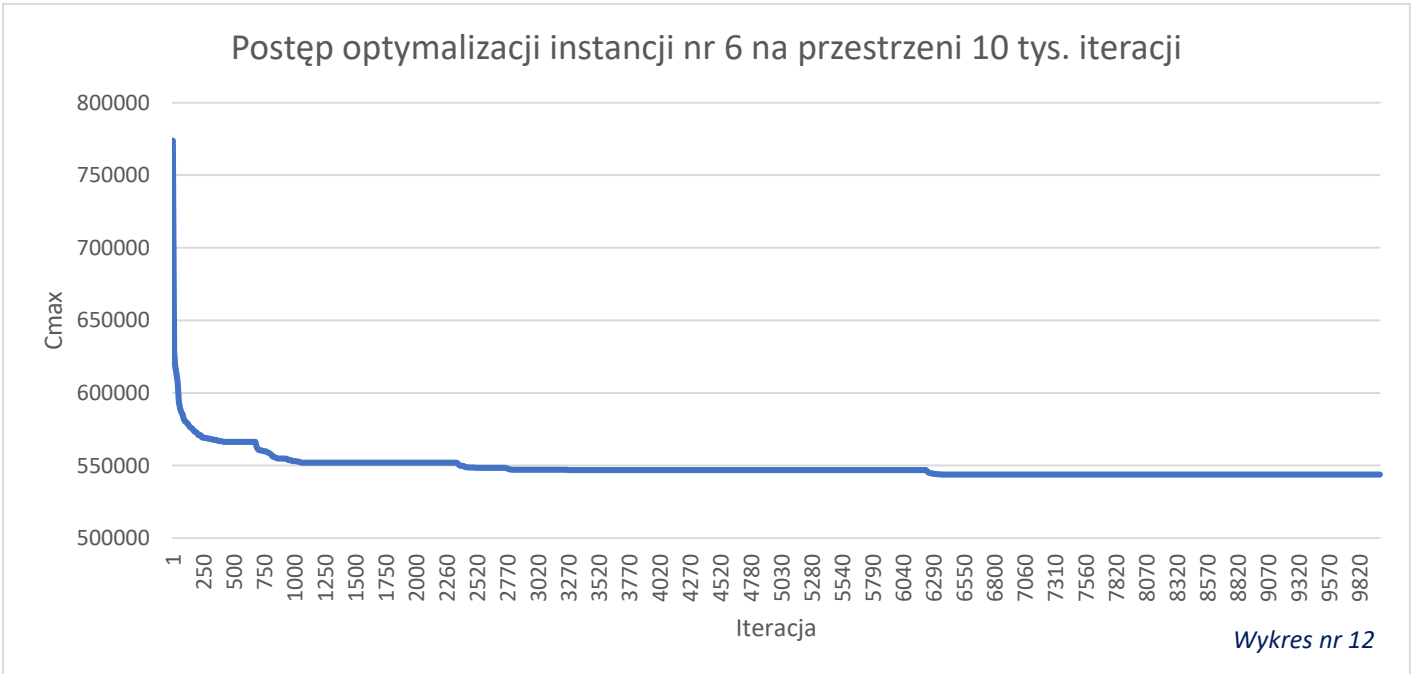


Wykres nr 11

	min	max
Numer instancji	5	2
Optymalizacja uszeregowania	27,21%	34,77%
Średnia	31,36%	
Mediana	31,67%	

Wykres ten wyróżnia się od pozostałych skumulowaniem wartości Cmax w okolicy punktu 480000. Do samej wartości nie należy się przywiązywać, jednak tendencja może być znacząca – być może jest to droga, jaką należy podążać w celu wyznaczenia najlepszej instancji testowej do badania parametrów algorytmu, nie zatracając przy tym jej różnorodności. Jest to argumentowane dużym wzrostem jakości uporządkowania zadań powiązanym z małym odchyleniem standardowym.

Od każdej reguły istnieje wyjątek – w tym przypadku jest nim Instancja numer 6, która w znacznej mierze odstaje jakością funkcji celu od pozostałych rozwiązań, dlatego też postanowiłem przeprowadzić dla niej rozszerzone testy:



Wykres nr 12

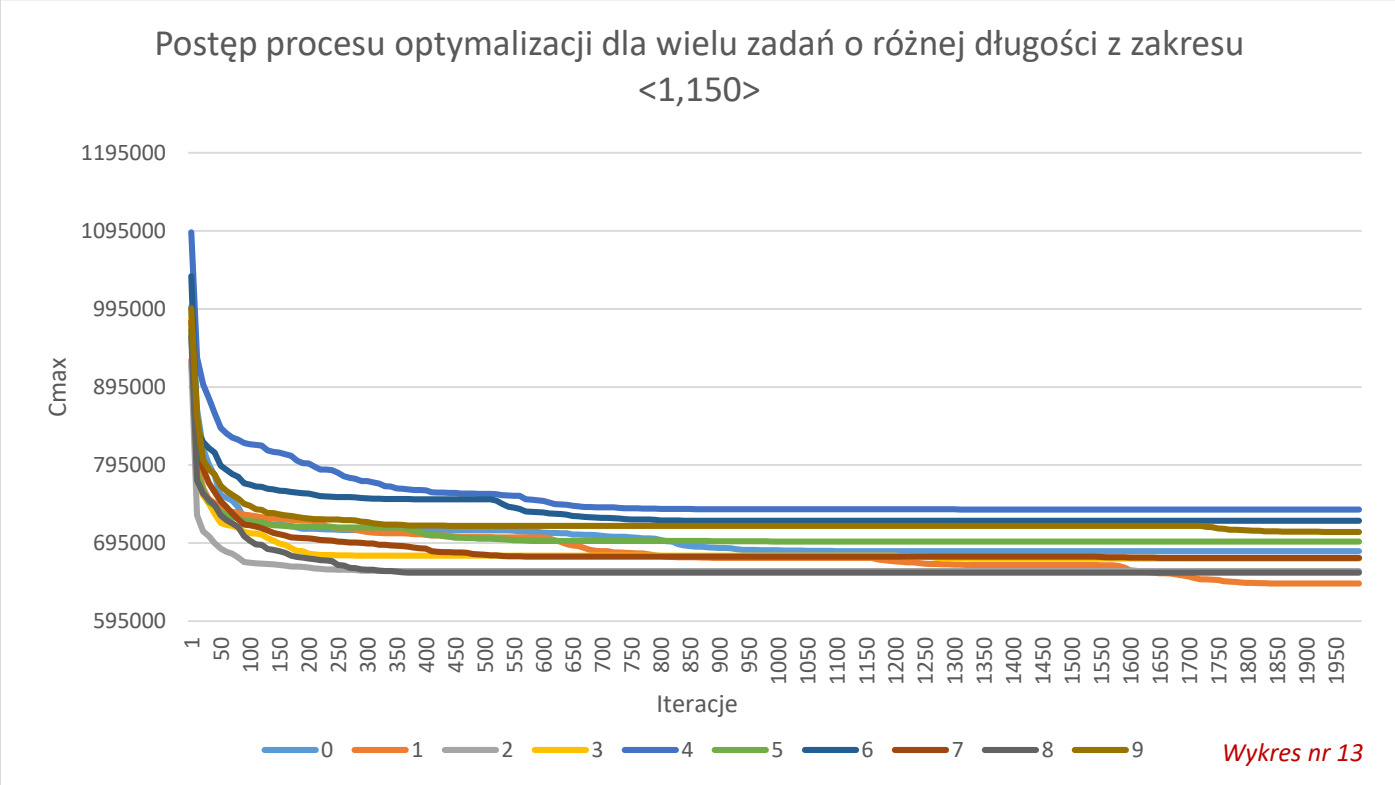
	It2000	It10000
Jakość uporządkowania początkowego	763491	*773705*
Jakość końcowa	554704	*543895*
Ilość Idle i ich dlługość (M1)	22, 119	22, 120
Ilość Idle i ich dlługość (M2)	58, 1769	59, 1735

Dla powyższej instancji wystąpiło 25 przerw technicznych o łącznym czasie trwania 1167

Jak widać, optymalizacja rozwiązania postępuje przez następne iteracje, jednak z nieakceptowalną poprawą rezultatu biorąc pod uwagę nakład czasowy potrzebny na jej uzyskanie. Porównując jednak dokładny test dla instancji nr 5 należącej do kategorii zadań długich, gdzie optymalizacja uzyskana dzięki próbie długiej była o 1,03% lepsza od wyniku dla próby o 2 tys. iteracji, wynik dla powyższej długiej próby jest o 2% lepszy niż dla próby liczącej 2 tysiące iteracji.

Poprawa wartości funkcji celu wzrasta o 29,3%, co pomimo dużego nakładu obliczeniowego, jest wynikiem poniżej średniej reprezentowanej dla danej kategorii instancji.

f) *Zadania o czasie trwania w zakresie <1,150>*



Wykres nr 13

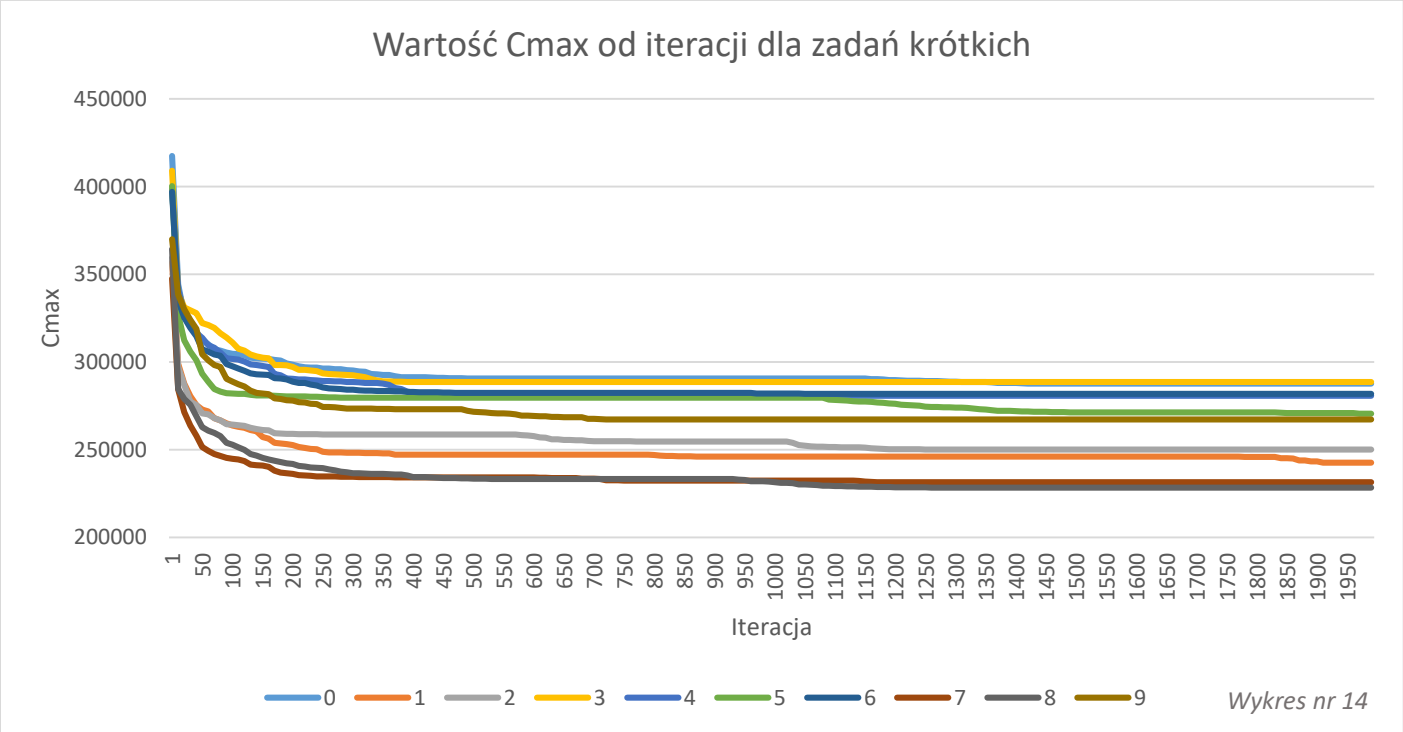
	min	max
Numer instancji	5	4
Optymalizacja uszeregowania	27,98%	32,48%
Średnia	30,44%	
Mediana	30,88%	

Widać, że większa różnorodność zadań składowych instancji skutkuje mniejszym odchyleniem procentu optymalizacji od średniej wartości. Na powyższym wykresie różnica w wartości Cmax dla instancji nr 1 (najlepsza) wynosi 643200, natomiast dla instancji nr 4 (najgorsza) wynosi 737930, widzimy więc różnice na poziomie 13% przy 31% polepszeniu jakości funkcji celu instancji nr 1 oraz 32,5% polepszeniu Cmax dla instancji nr 4. W przypadku zadań długich było to 7% różnicy przy polepszeniu w granicach od 6 do 8%. Dla instancji zadań długich i średnich, największa różnica w jakości rozwiązań wyniosła 13,21%, natomiast optymalizacja jakości mieściła się w przedziale 12,5 - 18%. Widzimy więc, że dla większej różnorodności zadań następuje lepsza optymalizacja, co między innymi wynika z tworzenia przerw technicznych przez generator instancji, których odstęp jest równy co najmniej długości najdłuższego zadania. Z drugiej strony, powoduje

to również większą różnorodność wartości funkcji celu dla poszczególnych instancji (a tym samym większe odchylenie od średniej), co skutkuje trudnościami w ocenie optymalnego uszeregowania oraz praktycznie uniemożliwia dokładne oszacowanie jakości znalezionego przez algorytm rozwiązania na podstawie ogólnego charakteru testowanej instancji problemu.

2) Zachowanie algorytmu z uwagi na stosunek ilości przerw technicznych do liczby zadań oraz ich długość.

a) Wykres dla instancji złożonych ze 100 krótkich zadań oraz 50 przerwach technicznych, których długość nie przekracza 50 jednostek czasu.

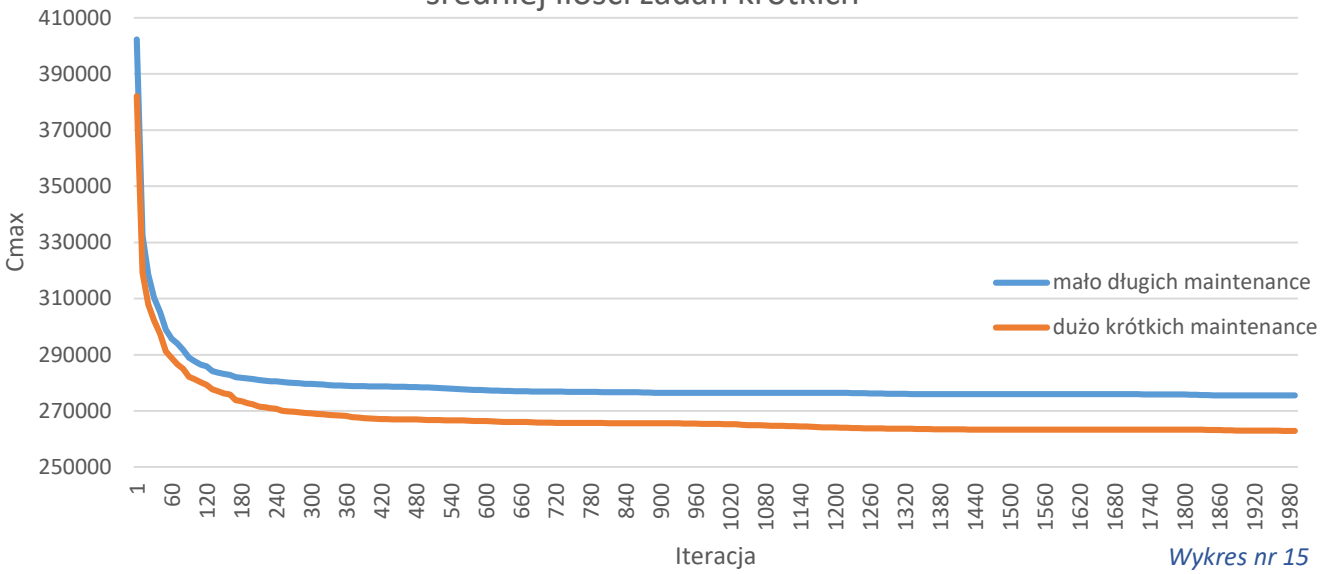


Wykres nr 14

	min	max
Numer instancji	9	8
Optymalizacja uszeregowania	27,80%	37,32%
Średnia	31,25%	
Mediana	30,75%	

Porównując rezultat powyższego testu do testu dla instancji różniącej się dwukrotnie mniejszą ilością przerw technicznych, jednak o dwukrotnie większym czasie ich trwania (wykres nr 9) widzimy przesunięcie się skrajnych wartości procentu optymalizacji uszeregowania o 2% - nie pociąga to jednak za sobą wzrostu średniej wartości, a co więcej – jej spadek. Warto również zauważyć, że pogorszeniu uległa wartość mediany. Analizując charakter wykresu można stwierdzić znaczne fluktuacje na przestrzeni pierwszych 350 iteracji, co wyróżnia ten wykres z pośród innych. Dalsze wnioski dotyczące wpływu przerw technicznych na jakość rozwiązania obrazuje następny wykres, oraz zamieszczone pod nim porównanie wyników.

b) Porównanie poprawy rozwiązania pomiędzy instancjami o krótkich zadaniach, jednak różniącymi się maintenance : dużo krótkich oraz mało długich

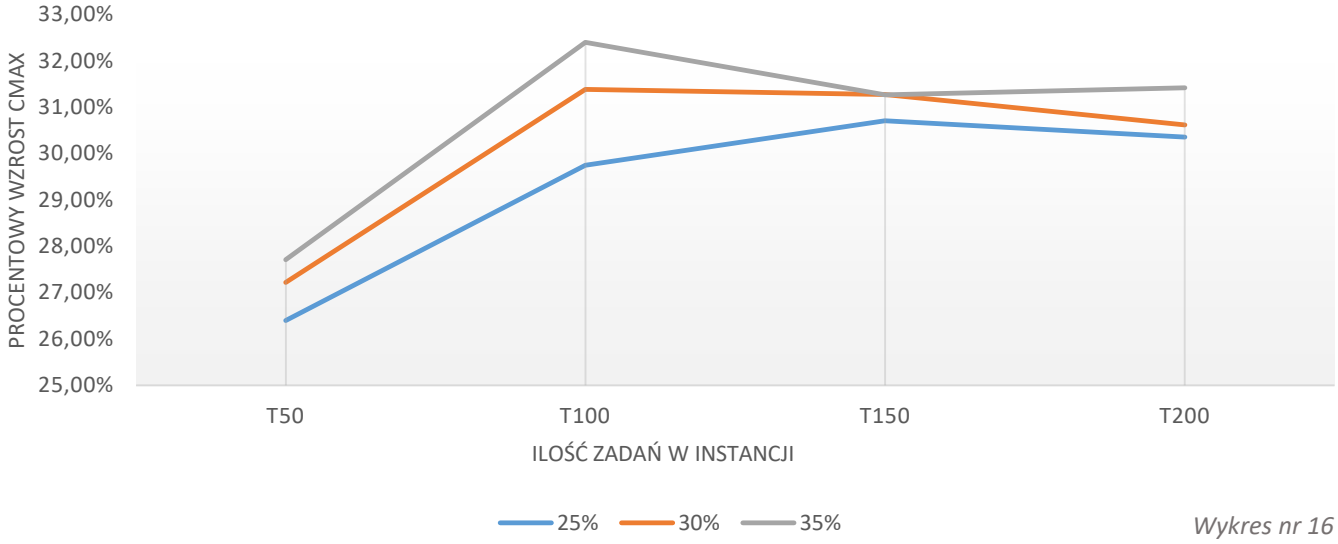


Wykres nr 15

	mało długich maintenance	**dużo krótkich maintenance**
Optymalizacja uszeregowania	31,56%	31,25%

Powyższy wykres jest porównaniem wartości funkcji celu (każda krzywa jest średnią z 10 instancji tego samego typu) w zależności od skrajnie różnych długości przerw dla tej samej ilości zadań, cechujących się długościami operacji należącymi do tego samego zakresu. Po załączonej do niego tabeli widać, że sama długość przerw technicznych nie wpływa w znaczny sposób na procentowe polepszenie jakości uporządkowania (poza zmianą wartości funkcji celu, co jest akurat logiczne) w porównaniu do początkowego wyniku.

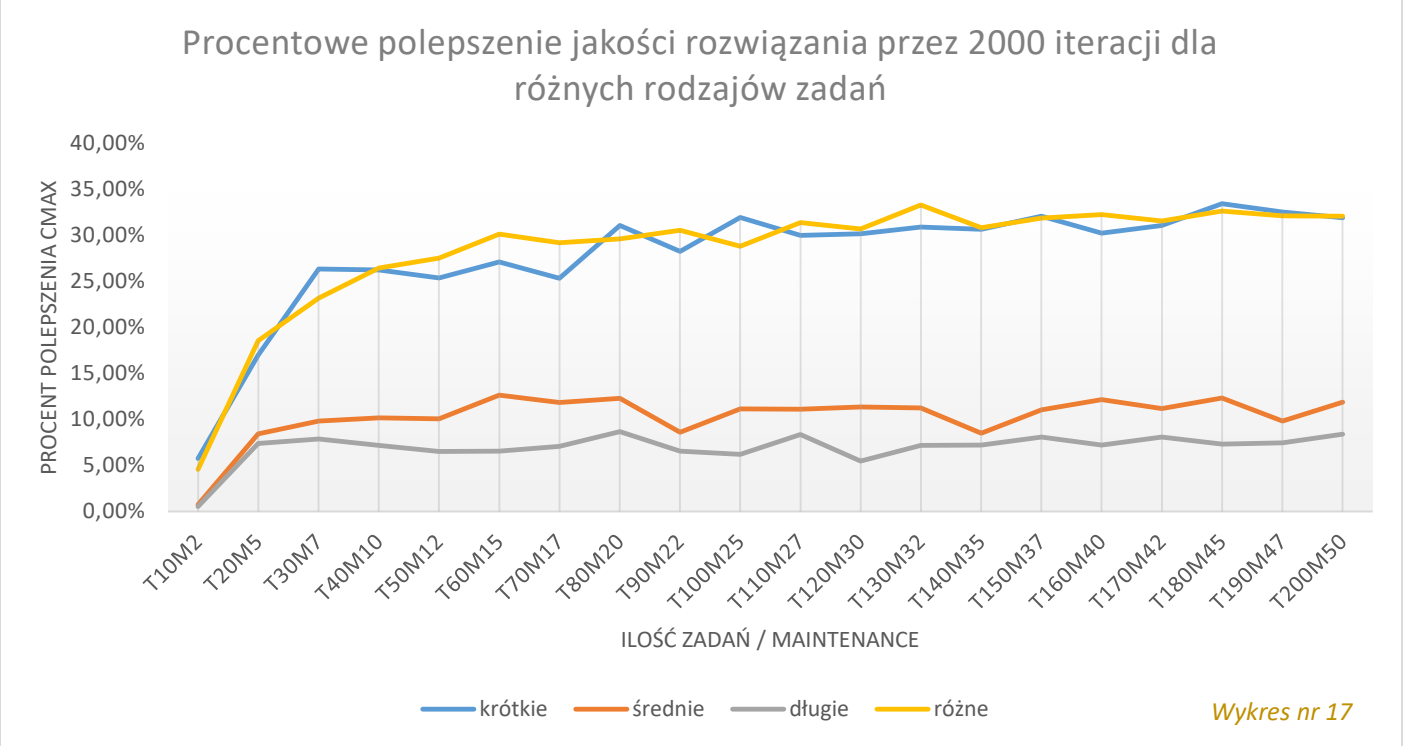
c) Wpływ ilości przerw technicznych na optymalizację uszeregowania



Wykres nr 16

Analizując powyższy wykres można stwierdzić, że dla większego udziału przerw technicznych (nie dłuższych niż 150) w instancji problemu, algorytm daje lepsze efekty. Myślę, że jest to dobry objaw ze względu na to, że pierwsze rozwiązanie jest rozwiązaniem losowym. Przerwy techniczne stanowią utrudnienie w uszeregowaniu, przez co staje się ono trudniejsze, a od algorytmu wymaga się większej inteligencji. Gdyby zwiększyć ilość iteracji, wyniki dla większych instancji mogłyby cechować się większym zróżnicowaniem niż te zaprezentowane, na podobnej zasadzie jak te dla instancji złożonych z 50 oraz 100 zadań, gdzie widać tendencję do wzrostu jakości wraz ze wzrostem ilości przerw technicznych.

3) Postęp optymalizacji dla różnych ilości zadań oraz przerw technicznych stanowiących 25% ilości zadań



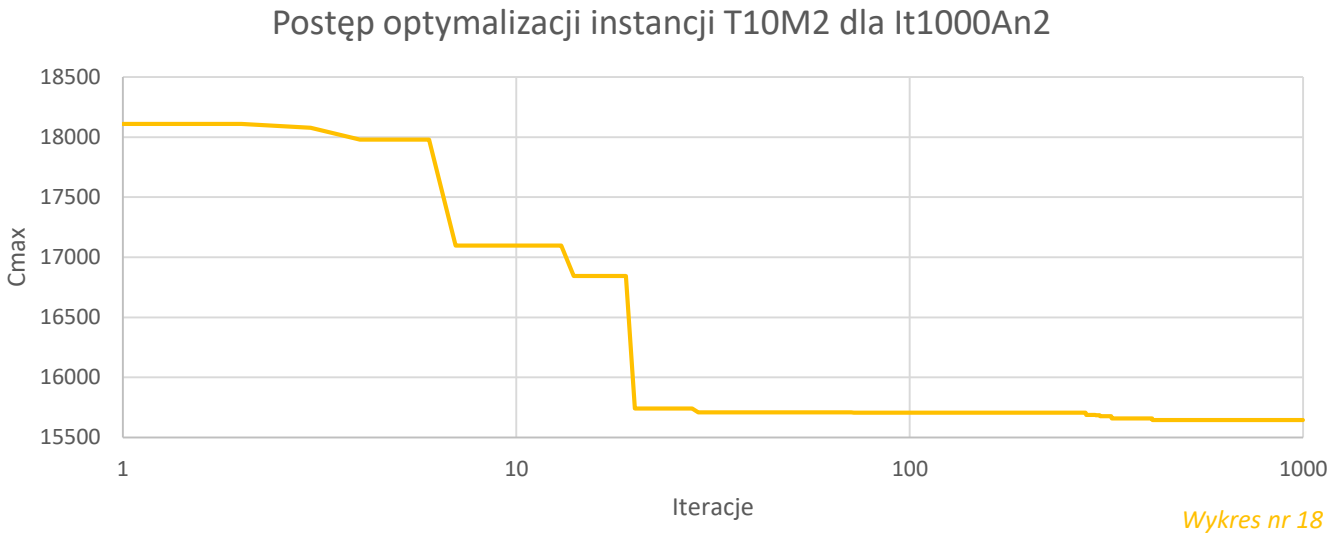
Wykres nr 17

	krótkie	średnie	długie	różne
T10M2	5,76%	0,76%	0,55%	4,57%
T20M5	16,98%	8,44%	7,40%	18,55%
T30M7	26,33%	9,81%	7,88%	23,16%
T40M10	26,24%	10,16%	7,18%	26,44%
T50M12	25,34%	10,06%	6,51%	27,50%
T60M15	27,09%	12,62%	6,57%	30,11%
T70M17	25,32%	11,83%	7,07%	29,17%
T80M20	31,05%	12,29%	8,69%	29,60%
T90M22	28,24%	8,59%	6,56%	30,53%
T100M25	31,91%	11,15%	6,22%	28,80%
T110M27	29,96%	11,11%	8,38%	31,37%
T120M30	30,16%	11,34%	5,48%	30,68%
T130M32	30,90%	11,26%	7,18%	33,29%
T140M35	30,64%	8,51%	7,23%	30,81%
T150M37	32,06%	11,04%	8,10%	31,87%
T160M40	30,23%	12,17%	7,20%	32,25%
T170M42	31,05%	11,18%	8,09%	31,55%
T180M45	33,42%	12,33%	7,33%	32,61%
T190M47	32,52%	9,82%	7,47%	32,09%
T200M50	31,90%	11,86%	8,41%	32,05%

Przeprowadzono 4 testy dla różnych konfiguracji długości zadań, dla których ilość przerw technicznych stanowiła 25% liczby zadań, natomiast ich długość maksymalna była równa górnej wartości progowej dla długości zadania.

Z wykresu jasno wynika(średnia z 6 prób), że algorytm najlepiej optymalizuje zadania, których czas trwania jest niedługi, lub których długość zadań mieści się w szerokim zakresie. Wyniki te pokrywają się również z zależnościami przedstawionymi we wcześniejszej części sprawozdania (np. dla instancji T100M25 długich zadań, gdzie stopień polepszenia rozwiązania [tam 6-8%]mieści się w przedziale prezentowanym przez wykres).

Dla Instancji testowej T10M2 w próbie dla zadań średnich oraz długich najgorsze wyniki wyniosły odpowiednio około 0,13% oraz 0,88% - jest to spowodowane bardzo dużą ilością mrówek (1000), przez co już w pierwszej iteracji znajdowane rozwiązanie jest bardzo dobre, natomiast później jedynie niewiele lepsze; dowód dla It1000An2 przy próbie dla zadań długich.



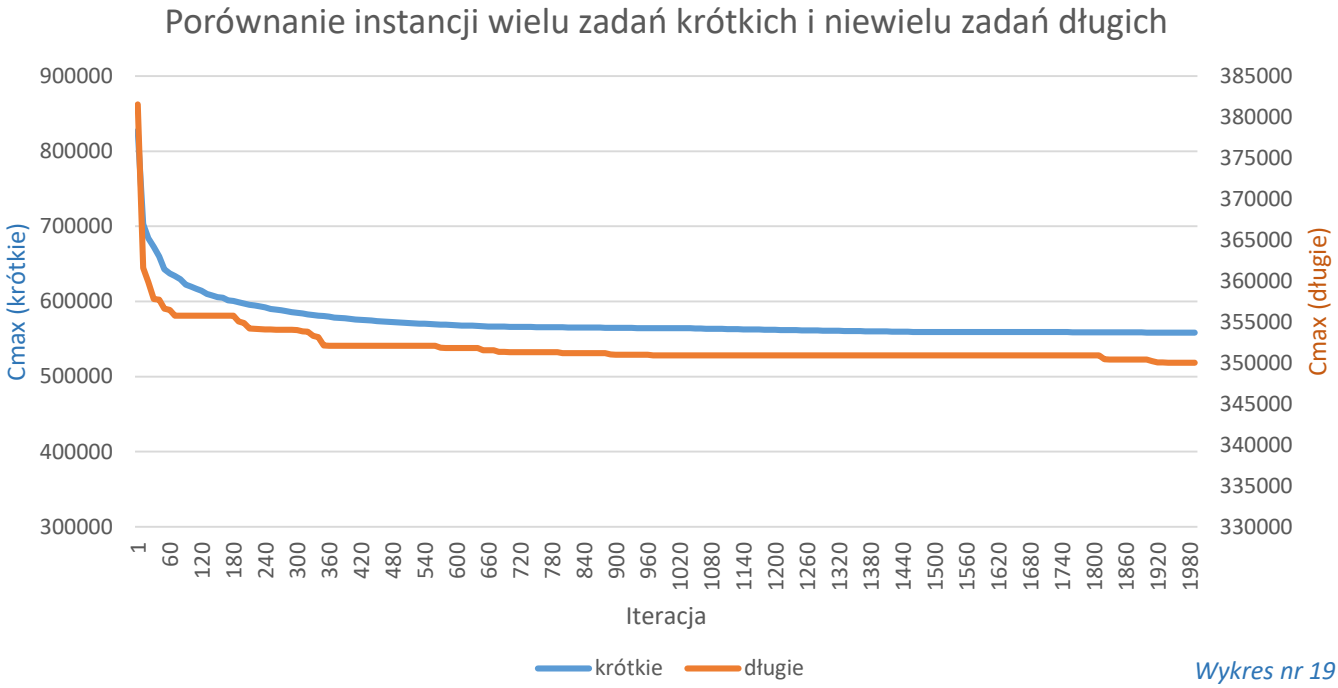
Wykres nr 18

W powyższym wykresie zastosowano skalę logarytmiczną, widać jednak, że rozwiązanie jest kształtowane przez pierwsze ~30 iteracji, by w okolicy 300 iteracji poprawić rozwiązanie o niewielki stopień – dla większej dokładności próbkowanie danych następuje z każdą iteracją (widać również znaczną poprawę po iteracji 8, 16 oraz 24[PER]) . Poniżej porównanie wyników dla It1000An2 oraz It2000An1000.

	It1000An2	**It2000An1000**
Wartość początkowa	18111	15665
Jakość końcowa	15646	15645
Ilość Idle i ich dlługość (M1)	2, 90	2, 94
Ilość Idle i ich dlługość (M2)	8, 221	6, 186

Łączny czas trwania maintenance'ów (2) wynosi 56

4) Analiza skrajnych przypadków instancji (dużo zadań krótkich oraz mało długich)



Wykres nr 19

Pod enigmatyczną nazwą ‘wielu’ oraz ‘niewielu’ zadań kryją się wartości 150 oraz 50, co w porównaniu z zastosowanymi przedziałami czasowymi operacji (dla przypomnienia – zadania krótkie składają się z operacji o długości z przedziału <1,50>, natomiast czas trwania długich oscyluje w przedziale <100,150>) powinno dawać zbliżone wartości sumaryczne (jeśli zsumować ich wartość).

Sam postęp optymalizacji instancji zadań długich jest jednak niewielki, co stanowi logiczną konsekwencję ograniczonego spektrum rozwiązań (im mniej, tym łatwiej o losowe rozwiązanie zbliżone do optymalnego). Nie jest również zaskoczeniem charakter wykresów, który dla zadań krótkich jest łagodniejszy, natomiast dla zadań długich (pomimo uśrednionego przebiegu z 10 instancji), obfituje w gwałtowne i znaczące załamania przez pierwsze 400 iteracji. Ponadto, dla zadanej funkcji celu, większa ilość zadań obfituje większą ilością składowych rozwiązania, co w przypadku porównywania samych jej wartości, okazałoby się bezcelowe.

	krótkie	długie
optymalizacja uszeregowania	32,48%	8,26%

Wnioski / rozważania

Podsumowując rezultaty optymalizacji uszeregowania uzyskanej za sprawą algorytmu widzimy, że radzi on sobie najlepiej dla zróżnicowanych instancji. Myślę jednak, że nie należy wysoko cenić tego spostrzeżenia – jest ono raczej ogólne dla każdego typu instancji – większa różnorodność skutkuje łatwiejszym dopasowaniem, jakie można wykonać. W niektórych przypadkach nie istnieje szansa na znalezienie lepszego uporządkowania – przykładowo dla identycznych zadań.

Znaczący może być jednak fakt, że wraz ze wzrostem stosunku przerw technicznych do ilości zadań w instancji, algorytm znajduje lepsze rozwiązania, co jak wcześniej wspomniałem, może być dowodem na zawierającą się w nim inteligencję działania. Oczywiście może to być również spowodowane masą jaką samą w sobie jest analogia mrowiska, jednak czym jest masa, jeśli źle ukierunkowana..?

Z pewnością pod rozwagę należy wziąć parametr odparowania macierzy oraz jego progresywną zmianę w czasie. Podobnie, można zastanowić się nad lepszym parametryzowaniem wartości minimalnych i maksymalnych w macierzy a także rozmiarowi listy tabu.

Kolejnym ważnym parametrem, który należy brać pod rozwagę jest ilość utrwalanych rozwiązań – można ją zmieniać lub uzależnić od dyspersji od rozwiązania najlepszego.

Biorąc pod uwagę inne implementacje przedstawionego algorytmu, w których ilość iteracji przeważnie jest znacznie mniejsza (<500) należy podjąć próbę zwiększenia wydajności algorytmu. Co prawda, źródła z których korzystałem, praktycznie zawsze oprócz zmutowanej metaheurystyki ACO, zawierały również implementację innych algorytmów, w szczególności Taboo Search (z czego również w niewielkim stopniu postanowiłem skorzystać). Sama jakość algorytmu prawdopodobnie również mogłaby być lepsza – jest mi to jednak trudno ocenić ze względu na krótki okres czasu jaki minął od jego powstania i ogólnej egzystencji, oraz brak spostrzeżeń ze strony innej osoby, za które byłbym bardzo wdzięczny.

Niestety, z powodu bardzo dużej ilości testów, nie wszystkie udało mi się utrwalić – cześć została utracona ze względu na awarię komputera, cześć samemu omylnie usunąłem. Niestety muszę przyznać, że wśród takich znajdują się testy dla parametru PER – na szczęście udało mi się sporządzić wykres oraz przeanalizować dane, które jednak znajdowały się w osobnej lokalizacji.

Mówiąc o testach oraz ogólnej wydajności, chciałbym wspomnieć o braku testu na progresywną zmianę populacji mrowiska, oraz samą jej statyczną wartość. Przyznam, że przyjęte początkowe wartości były dosyć intuicyjne i skutkowały wprowadzaniem jako parametry wejściowe kolejnych potęg liczby 10, co być może nie było najlepszym pomysłem. Może się więc okazać, że zmniejszenie populacji mrówek o 50% skutkowałoby pogorszeniem jakości o 1-2%, przy znacznym przyśpieszeniu procesu, co w niekiedy mogłoby okazać się ważniejsze. Mimo to, algorytm w wybranej konfiguracji znajduje rozwiązanie o – subiektywnie oceniając – dobrej jakości w czasie nie większym niż 3 minuty, co wydaje się być akceptowalne.

Referencje

Ant colony optimization combined with taboo search for the job shop scheduling problem Kuo-Ling Huang, Ching-Jong Liao(2006): https://svn-d1.mpi-inf.mpg.de/AG1/MultiCoreLab/papers/HuangLiao06%20-%20ACO%20for%20the%20job%20shop%20scheduling%20problem.pdf

- An Improved Ant Colony Optimization Algorithm for Solving TSP - Yimeng Yue, Xin Wang (2015) : http://www.sersc.org/journals/IJMUE/vol10_no12_2015/16.pdf

An improved ant colony algorithm for the Job Shop Scheduling Problem - Guillermo Leguizamon´, Martin Schutz : http://sedici.unlp.edu.ar/bitstream/handle/10915/23004/Documento_completo.pdf?sequence=1

- AN ANT COLONY OPTIMIZATION ALGORITHM FOR JOB SHOP SCHEDULING PROBLEM Edson Flórez1 , Wilfredo Gómez2 and MSc. Lola Bautista3 (2013): https://arxiv.org/ftp/arxiv/papers/1309/1309.5110.pdf

-Ant colony Optimization Algorithms : Introduction and Beyond Anirudh Shekhawat Pratik Poddar Dinesh Boswal(2009) - http://mat.uab.cat/~alseda/MasterOpt/ACO_Intro.pdf

- Parameter Adaptation in Ant Colony Optimization, IRIDIA – Technical Report Series Technical Report No. TR/IRIDIA/2010-002 January (2010) - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.182.377&rep=rep1&type=pdf