AUTOMATYKA • 2007 • Tom 11 • Zeszyt 1–2

Józef Grabowski*, Jarosław Pempera*

Metody dywersyfikacji procesu przeszukiwań algorytmów popraw dla problemu przepływowego z kryterium sumacyjnym

1. Wprowadzenie

Zagadnienia szeregowania zadań produkcyjnych z sumacyjnymi kryteriami optymalizacyjnymi należą do jednych z najtrudniejszych zagadnień optymalizacji kombinatorycznej. Już najprostsze dwumaszynowe systemy produkcyjne, generują NP-trudne problemy optymalizacyjne, np. dwumaszynowy problem przepływowy z kryterium minimalizacji sumy czasów zakończenia realizacji zadań [1]. Konstrukcja efektywnych algorytmów (dostarczających dobrych rozwiązań w rozsądnym czasie) utrudniona jest głównie ze względu na długi czas obliczeń wartości funkcji celu oraz brak własności pozwalających na aprioryczne (bez wyznaczenia wartości funkcji celu) wyeliminowanie pewnych rozwiązań.

Efektywne algorytmy dedykowane takim problemom z reguły oparte są na szeroko rozumianych metodach przeszukiwań lokalnych. Co więcej, często ich konstrukcja opiera się na kilku takich metodach. Dobrym przykładem takiego algorytmu jest algorytm genetyczny [2], który oprócz mechanizmów genetycznych zawiera elementy przeszukiwania z zabronieniami, oraz symulowanego wyżarzania. Dla rozważanego w pracy problemu, w literaturze można znaleźć również bardzo szybkie algorytmy konstrukcyjne [3], jednakże jakość generowanych przez nie rozwiązań jest znacznie gorsza.

W pracy przedstawiamy opis matematyczny problemu przepływowego, opis podstawowych sąsiedztw wykorzystywanych w algorytmach lokalnego przeszukiwania dla problemów permutacyjnych, proponujemy algorytm tabu search oraz różne metody zwiększające dywersyfikację procesu przeszukiwań takie, jak: multiruchy, dynamiczna długość listy zabronień oraz zmienne otoczenie. W ostatniej sekcji przedstawione są wyniki eksperymentu komputerowego przeprowadzonego na danych testowych zaproponowanych przez Taillarda.

2. Sformułowanie problemu

W permutacyjnym problemie przepływowym należy wykonać n zadań ze zbioru $J = \{1, \ldots, n\}$ na m maszynach ze zbioru $M = \{1, \ldots, m\}$. Każde wykonywane jest kolejno

* Instytut Automatyki, Informatyki i Robotyki, Politechnika Wrocławska

na każdej maszynie w kolejności zgodnej z numeracją maszyn. Zadanie $j \in J$ wykonuje się na maszynie k w czasie $p_{kj} > 0$, $k = 1, \ldots, m$. Wykonywanie zadań nie może być przerywane. Maszyna $i \in M$ może w danej chwili wykonywać tylko jedno zadanie oraz kolejność zadań na maszynach jest identyczna. Zatem kolejność wykonywania zadań możemy opisać za pomocą permutacji elementów zbioru $\{1, \ldots, n\}$.

Niech Π oznacza zbiór wszystkich permutacji określonych na zbiorze $\{1, \ldots, n\}$. Dla ustalonej permutacji $\pi \in \Pi$, niech $C_{k, \pi(j)}$ oznacza moment zakończenia wykonywania zadania $\pi(j)$ na maszynie k, $j \in J$, $k = 1, \ldots, m$. Harmonogram wykonywania zadań określony przez momenty zakończenia ich wykonywania na poszczególnych maszynach musi spełniać opisane wyżej ograniczenia, które można precyzyjnie opisać za pomocą nierówności matematycznych:

$$C_{1, \pi(1)} \geq p_{1,j} \tag{1}$$

$$C_{k,j} \geq C_{k-1,j} + p_{k,j}, \quad k = 2, \ldots m, j = 1, \ldots, n \tag{2}$$

$$C_{k,\pi(j)} \geq C_{k,\pi(j-1)} + p_{k,\pi(j)}, \quad k = 1, \ldots, m, j = 2, \ldots, n \tag{3}$$

Harmonogram spełniający powyższe ograniczenia można wyznaczyć, stosując następującą formułę rekurencyjną:

$$C_{k,\pi(j)} = \max\{C_{k,\pi(j-1)}, C_{k-1,\pi(j)}\} + p_{k,\pi(j)}, \; k = 1, \ldots, m, \; j = 1, \ldots, n \tag{4}$$

gdzie:

$\pi(0) = 0$, $C_{k,0} = 0$, $k = 1, \ldots, m$,

$C_{0,j} = 0$, $j = 1, \ldots, n$.

W rozważanym problemie, należy znaleźć permutację $\pi^* \in \Pi$, taką że

$$C_{sum}(\pi^*) = \min_{\pi \in \Pi} C_{sum}(\pi) \tag{5}$$

gdzie $C_{sum}(\pi) = \sum_{j=1}^{n} C_{m,\pi(j)}$.

3. Algorytm tabu search (TS)

Technika tabu search od wielu lat stosowana jest z powodzeniem do konstruowania efektywnych algorytmów heurystycznych dla problemów szeregowania zadań. W każdej iteracji algorytmu opartego na tej technice przeglądane są wszystkie rozwiązania sąsiednie pewnego rozwiązania nazywanego bazowym celem znalezienia rozwiązania najlepszego, tj. o najmniejszej (największej) wartości funkcji celu. Znalezione rozwiązanie zastępuje rozwiązanie bazowe w następnej iteracji. W sąsiedztwie pewnym rozwiązaniom przyporządkowuje się status **zabronione**. Rozwiązania te nie są brane pod uwagę podczas poszukiwania rozwiązania najlepszego w sąsiedztwie. Są to w szczególności rozwiązania, które

zostały już wcześniej wygenerowane. Z rozwiązań lepszych od najlepszych dotychczas znalezionych przez algorytm zdejmuje się status **zabronione**.

Mechanizm zabronień jest elementem charakterystycznym dla techniki tabu i jest najczęściej realizowany w postaci listy, na której umieszczane są pewne informacje pochodzące od rozwiązania bazowego i/lub nowego rozwiązania bazowego wybranego podczas przeszukiwań. Informacje te będziemy nazywali atrybutami. Lista zabronień ma ograniczoną długość i jest obsługiwana zgodnie z regułą FIFO, tj. w chwili gdy nowy element dodawany jest do listy, najstarszy jest z niej usuwany.

Algorytm kończy działanie, gdy zostanie wykonana określona liczba iteracji bez poprawy bieżącego najlepszego rozwiązania, gdy zostanie wykonana określona liczba iteracji algorytmu (*Maxiter*), gdy zostanie przekroczony zadany czas obliczeń lub gdy zostanie znalezione rozwiązanie o satysfakcjonującej wartości funkcji celu itd.

Efektywność algorytmów tabu search można zasadniczo poprawić, stosując różne metody intensyfikcji i dywersyfikacji procesu przeszukiwań. Jedną z najbardziej znanych metod intensyfikacji jest metoda skoku powrotnego [4]. W pracy skupiamy się na metodach dywersyfikacji (rozproszenia) procesu przeszukiwań.

Opisane zostaną trzy techniki:

1) ruchy złożone (multiruchy),
2) dynamiczna lista zabronień,
3) zmienne otoczenie.

3.1. Ruchy i sąsiedztwo

Podstawowym elementem każdego algorytmu opartego na przeszukiwaniu lokalnym jest definicja zbioru ruchów generujących otoczenie rozwiązania bazowego.

W permutacyjnych problemach szeregowania zadań najczęściej stosuje się trzy typy ruchów:

1) wstaw INS (*insert*),
2) zamień ICH (*interchange*),
3) zamień sąsiednie SWAP (*interchange adjacent or swap*).

Niech $v = (x, y)$ będzie parą pozycji w permutacji π, $x, y \in \{1, 2, ..., n\}$, $x \neq y$. Para $v = (x, y)$ definiuje ruch **wstaw**, który usuwa zadanie $\pi(x)$ z jego oryginalnej pozycji x, a następnie wstawia je na pozycję y w π. Ruch ten generuje nową permutację π_v z π w następujący sposób:

$$\pi_v = (\pi(1), ..., \pi(x-1), \pi(x+1), ..., \pi(y), \pi(x), \pi(y+1), ..., \pi(n)), \text{ jeżeli } x < y \quad (6)$$

$$\pi_v = (\pi(1), ..., \pi(y-1), \pi(x), \pi(y), ..., \pi(x-1), \pi(x+1), ..., \pi(n)), \text{ jeżeli } x > y \quad (7)$$

Otoczenie rozwiązania bazowego π składa się ze zbioru permutacji otrzymanych przez wykonanie ruchów ze zbioru $Z = \{(x, y) \mid x, y \in \{1, 2, ..., n\}, y \notin \{x, x-1\}\}$. Otoczenie to będziemy oznaczać symbolem $N(Z, \pi) = \{\pi_v \mid v \in Z\}$. Zawiera ono $(n-1)^2$ rozwiązań.

Niech $v = (x, y)$ będzie parą pozycji w permutacji π, $x, y \in \{1, 2, ..., n\}$, $x < y$. Para $v = (x, y)$ definiuje ruch **zamień**, który usuwa zadanie $\pi(x)$ z pozycji x oraz usuwa zadanie $\pi(y)$ z pozycji y, a następnie wstawia zadanie $\pi(y)$ na pozycję x oraz $\pi(x)$ na pozycję y w π.

Ruch ten generuje nową permutację π_v z π w następujący sposób:

$$\pi_v = (\pi(1), ..., \pi(x-1), \pi(y), \pi(x+1), ..., \pi(y-1), \pi(x), \pi(y+1), ..., \pi(n)) \quad (8)$$

Otoczenie rozwiązania bazowego π składa się z zbioru permutacji otrzymanych przez wykonanie ruchów ze zbioru $U = \{(x, y) \mid x, y \in \{1, 2, ..., n\}, x < y\}$. Otoczenie to będziemy oznaczać symbolem $N(U, \pi) = \{\pi_v \mid v \in U\}$. Zawiera ono $n(n-1)/2$ rozwiązań.

Zbiór ruchów typu zamień sąsiednie jest podzbiorem ruchów typu zamień $V = \{(x, y) \mid x, y \in \{1, 2, ..., n\}, y = x + 1\}$. Sąsiedztwo generowane przez zbiór ruchów V zawiera $(n-1)$ rozwiązań.

3.2. Ruchy złożone (multiruchy)

W zbiorze ruchów generujących rozwiązania sąsiednie może znajdować się wiele ruchów poprawiających. W pracy [5], dla problemu przepływowego z ograniczeniem bez czekania przedstawiono warunki, jakie musi spełniać podzbiór ruchów poprawiających, tak aby ich jednocześnie wykonanie spowodowało wygenerowanie rozwiązania, w którym poprawy generowane przez każdy ruch sumowały się. Tego typu podzbiór nazwany został zbiorem ruchów niezależnych oraz multiruchem. Testy komputerowe pokazały, że zastosowanie multiruchów spowodowało duże poprawy złych rozwiązań oraz w połączeniu z odpowiednią strategią ich użycia, dobre ich właściwości dywersyfikacyjne. Dla rozważanego problemu proponujemy nową definicję multiruchu bazującą na pojęciu ruchów *k*-odseparowanych.

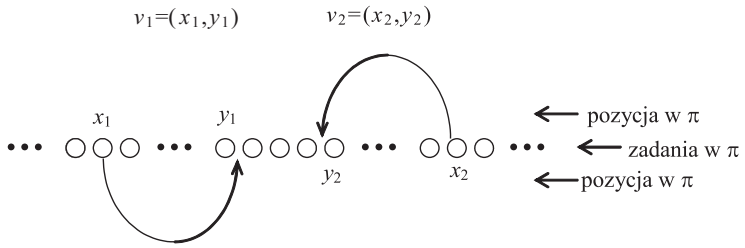
Niech $PZ = \{v \in Z \mid C_{sum}(\pi_v) < C_{sum}(\pi)\}$ będzie zbiorem ruchów poprawiających bieżące rozwiązanie bazowe. Dwa ruchy $v_1 = (x_1, y_1) \in PZ$ i $v_2 = (x_2, y_2) \in PZ$ w π będziemy nazywali *k*-odseparowanymi, jeżeli sekwencje $(\min\{x_1, y_1\}, \max\{x_1, y_1\})$ oraz $(\min\{x_2, y_2\}, \max\{x_2, y_2\})$ oddzielone są co najmniej k zadaniami w π, precyzyjniej spełniony jest następujący warunek:

$$\max(x_1, y_1) + k < \min(x_2, y_2) \text{ lub } \max(x_2, y_2) + k < \min(x_1, y_1) \quad (9)$$

Na rysunku 1 przedstawiono dwa ruchy typu wstaw 3-odseparowane. W celu wyznaczenia podzbioru ruchów *k*-odseparowanych zastosowano przedstawiony na rysunku 2 algorytm zachłanny. Zbiór w ten sposób otrzymanych *k*-odseparowanych ruchów będziemy nazywali multiruchem. Zauważmy, że każdy z ruchów składowych multiruchu operuje na oddzielnej frakcji permutacji bazowej.

W rozważanym problemie wykonywanie multiruchu nie gwarantuje otrzymania rozwiązania, w którym poprawa rozwiązania bazowego będzie równa sumie poprawy generowanych przez każdy ruch składowy. Co więcej, nowe rozwiązanie bazowe może być znacznie gorsze od rozwiązania bazowego. Oczywiście w przypadku gdy multiruch składa się tylko z jednego ruchu składowego, otrzymamy nowe rozwiązanie lepsze od bazowego. Z tego też powodu multiruch należy wykonywać na rozwiązaniach znacznie gorszych od najlepszych dotychczas znalezionych. W celu wykrywania takich rozwiązań proponujemy prosty mechanizm oparty na zliczaniu iteracji algorytmu bez poprawy rozwiązania bazowego. W przypadku przekroczenia wartości progowej g wykonywany jest multiruch.

W kontekście wykorzystania multiruchu w algorytmach opartych na metodzie tabu, należy nadmienić, że multiruch może być generowany na bazie ruchów niezabronionych lub wszystkich ruchów. W drugim przypadku wykonane multiruchu może spowodować wygenerowania rozwiązania, w którym niektóre zakazy nie są spełnione.



Rys. 1. Ruchy v_1 i v_2 3-odseparowane

Dane: k, π.
Szukane: $\overline{v}$ – zbiór ruchów k-odseparowanych
Krok 1. Wyznacz zbiór PZ dla permutacji π, podstaw $\overline{v}=\emptyset$.
Krok 2. Dopóki $PZ \neq \emptyset$ wykonaj:
znajdź najlepszy ruch u taki, że $C_{\text{sum}}(\pi_u) = \min_{v \in PZ} C_{\text{sum}}(\pi_v)$,
usuń go ze zbioru PZ oraz wstaw do $\overline{v}$, tzn. podstaw $PZ:=PZ - \{u\}$
oraz $\overline{v} := \overline{v} \cup \{u\}$.
Dla każdego $v \in PZ$ wykonaj:
jeżeli dla ruchów v oraz u nie jest spełniony warunek (9), to
usuń v z PZ, tj. $PZ = PZ - \{v\}$.

Rys. 2. Algorytm wyznaczania multiruchu $\overline{v}$

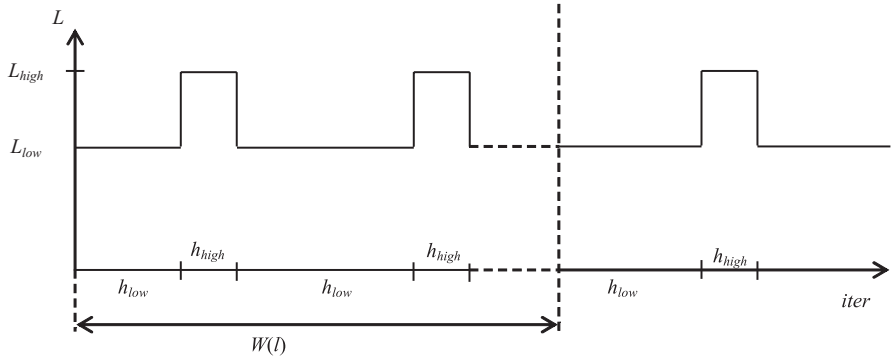
3.3. Lista tabu, status ruchów, dynamiczna zmiana długości

Elementami listy zabronień są pary numerów zadań. Rozwiązanie sąsiednie α rozwiązania bazowego π jest zabronione (posiada status zabronione), jeżeli dla każdej pary (a, b) znajdującej się na liście $ps(a) < ps(b)$, gdzie $ps(x)$ oznacza pozycję zadania x w permutacji α. Zauważmy, że zaproponowany mechanizm zabronień możemy zastosować do przeszukiwania otoczenia typu wstaw jak i otoczenia typu zamień.

W przypadku wykonania ruchu $v = (a,b)$ typu wstaw, do listy zabronień wstawiana jest para $(\pi(a), \pi(a + 1))$, gdy $b > a$, albo para $(\pi(a – 1), \pi(a))$ gdy $b < a$. Natomiast w przypadku wykonania ruchu $v = (a, b)$ typu zamień, do listy zabronień wstawiane są dwie pary: $(\pi(a), \pi(a + 1))$ oraz $(\pi(b – 1), \pi(b))$. W przypadku wykonania multiruchu, zawartość listy zabronień modyfikowana jest na podstawie najlepszego ruchu składowego.

Proces dywersyfikacji przeszukiwań w algorytmach tabu można zwiększyć, zmieniając długość listy zabronień podczas działania algorytmu.

Technika ta została po raz pierwszy wykorzystana w algorytmie TS dla klasycznych problemów szeregowania zadań takich, jak problem przepływowy permutacyjny [6] i problem gniazdowy [7].



Rys. 3. Dynamiczna lista zabronień

Technika dynamicznej zmiany długości listy zabronień polega na cyklicznym, skokowym zwiększaniu i skracaniu długości listy zabronień. Podczas zwiększania długości listy dodawane są elementy puste, natomiast podczas skracania usuwane są elementy najstarsze. Długość listy zabronień L zmienia się zgodnie z następującą formułą:

$$\mathrm{L} = \begin{cases} L_{low}, & \text{dla} \quad W(l) < iter \le W(l) + h_{low} \\ L_{high}, & \text{dla} \quad W(l) + h_{low} < iter \le W(l) + h_{low} + h_{high} \end{cases} \tag{10}$$

gdzie:

$l = 1, 2, \ldots$ – numer cyklu,

$W(l) = \sum_{s=1}^{l} (l-1)(h_{low} + h_{high})$, L_{low} i L_{high} ($L_{low} < L_{high}$) – odpowiednio długość skróconej i wydłużonej listy tabu,

h_{low} i h_{high} – liczba iteracji jaką algorytm wykonuje odpowiednio ze skróconą i wydłużoną listą tabu (patrz rys. 3).

3.4. Zmienne otoczenie

Innym sposobem rozproszenia procesu przeszukiwań jest zmiana otoczenia rozwiązania bazowego. W pracy [8], dla problemu przepływowego z ograniczeniami bez czekania, zaproponowano algorytm zstępujący ze zmiennym otoczeniem, w którym zmiana otoczenia następowała po znalezieniu minimum lokalnego dla bieżącego otoczenia. Kolejne otoczenia generowane były w oparciu o pojęcie ruchu typu *k*-wstaw. Wykonanie ruchu *k*-wstaw polega na usunięciu *k* kolejnych zadań z π i ich powtórnym wstawieniu na inne pozycje w π w tej samej kolejności w jakiej występowały w π.

Dla rozważanego problemu proponujemy zmianę otoczenia z typu wstaw na typu zamień i odwrotnie za każdym razem, gdy wykonywany jest multiruch.

4. Badania testujące

Celem przeprowadzonego eksperymentu komputerowego była ocena wpływu zaprezentowanych mechanizmów dywersyfikacyjnych na efektywność algorytmu tabu TS. W tym celu zaimplementowano w języku C++ pięć algorytmów TS, TSD+M1, TSD+M2, TSVD+M1 oraz TSVD+M2, przy czym D w nazwie oznacza, że wykorzystano dynamiczną listę tabu, M1 – zastosowano multiruch konstruowany na bazie ruchów niezabronionych, M2 – zastosowano multiruch konstruowany na bazie wszystkich ruchów, natomiast V oznacza, że zastosowano zmienne otoczenie. Każdy z algorytmów TS, TSD+M1, TSD+M2 zaimplementowano w trzech wersjach, różniących się rodzajem otoczenia, tj. z otoczeniem **wstaw** (INS), z otoczeniem **zamień** (INCH) oraz z otoczeniem będącym sumą otoczenia wstaw i zamień (INS&INCH).

Badania testowe algorytmów były przeprowadzone na komputerze z procesorem Pentium 1,4 GHz na zbiorze 50 instancji należących do pierwszych pięciu grup zaproponowanych przez Taillarda [9], tj. grup o rozmiarach $n \times m$: 20×5, 20×10, 20×20, 50×5, 50×10. Rozwiązanie początkowe dla algorytmów TS generowane było przez algorytm NEH [10].

W wyniku wstępnych testów wybrano następujące wartości parametrów:

– dla algorytmu TS, $L = 7$ dla INS, oraz $L = 11$ dla INCH oraz INS&INCH,
– dla pozostałych algorytmów, parametry dynamicznej listy tabu: $L_{low} = 5$, $L_{high} = 16$, $h_{low} = 100$, $h_{high} = 200$, parametry multiruchu: $k = 2$, $g = 3$.

Dla każdej instancji problemu zostały wyznaczone następujące wartości:

– $PRD(A)=100\%(C_{\text{sum}}(\pi_A) - C^{Ref})/C^{Ref}$ – względny błąd algorytmu A, gdzie π_A jest rozwiązaniem wygenerowanym przez algorytm A, natomiast C^{Ref} jest wartością referencyjną z pracy [2];
– $CPU(A)$ – czas obliczeń algorytmu A.

Na podstawie tych danych, dla każdej grupy instancji, obliczono średni błąd względny $APRD(A)$ oraz średni czas obliczeń $ACPU$(A) algorytmu.

W tabeli 1 został przedstawiony średni błąd algorytmów TS, TSD+M1 oraz TSD+M2 operujących na otoczeniach INS, INCH, INS&INCH oraz wykonujących 1000 oraz 10 000 iteracji. Podano również średni czas obliczeń algorytmu uruchomionego na 1000 iteracji.

Z wyników badań przeprowadzonych dla klasycznego algorytmu TS wynika, że najlepsze rozwiązania (w sensie średnim) zostały wygenerowane dla otoczenia INS&INCH zarówno dla 1000, jak i dla 10 000 iteracji. Niestety lepsze rezultaty zostały okupione znacznie dłuższym czasem obliczeniowym, tj. blisko 1,5 razy niż dla algorytmu TS z otoczeniem INS oraz 3 razy niż dla algorytmu TS z otoczeniem INCH. Rozwiązania generowane przez algorytm TS z otoczeniem INS oraz otoczeniem INCH są znacznie gorsze nawet, jeżeli porównamy współczynniki APRD dla 10 000 iteracji w przypadku otoczeń INS lub INCH z współczynnikami APRD dla 1000 iteracji dla otoczenia INS&INCH. Analizując kolumny wyników dla 1000 i 10 000 iteracji, w przypadku otoczeń INS oraz INCH, możemy zauważyć tylko niewielką poprawę współczynnika APRD. Oznacza to, że obszar przeszukiwań algorytmu TS znacznie się ograniczył i/lub algorytm wpadł w cykl. Najprawdo-

podobniej spowodowane to jest zbyt małą liczbą „dobrych" rozwiązań w tych sąsiedztwach. W przypadku sąsiedztwa INS&INCH obserwuje się dalszą poprawę APRD, zapewne dzięki większej liczbie rozwiązań generowanych przez dwa różne typy ruchów.

Tabela 1
Średni błąd algorytmów TS, TSD+M1, TSD+M2

Grupa	INS			INCH			INS&INCH		
	1000		10 000	1000		10 000	1000		10 000
	APRD	ACPU	APRD	APRD	ACPU	APRD	APRD	ACPU	APRD
TS									
20×5	0,39	1,7	0,39	0,50	0,8	0,46	0,23	2,5	0,11
20×10	0,41	3,0	0,41	0,73	1,5	0,72	0,07	4,4	0,00
20×20	0,45	5,4	0,44	0,80	2,7	0,78	0,26	8,0	0,11
50×5	3,32	25,6	3,30	1,55	12,7	1,51	1,16	40,5	0,95
50×10	3,27	46,5	3,27	2,82	22,4	2,74	1,83	69,5	1,44
Średnio	**1,57**		**1,56**	**1,28**		**1,24**	**0,71**		**0,52**
TSD+M1									
20×5	0,27	1,8	0,09	0,57	0,8	0,31	0,27	2,5	0,13
20×10	0,21	3,0	0,10	0,74	1,5	0,35	0,31	4,4	0,25
20×20	0,30	5,4	0,03	0,76	2,7	0,51	0,08	8,0	0,03
50×5	2,77	25,7	1,76	1,69	12,6	1,12	1,11	40,2	0,74
50×10	2,41	46,6	1,71	2,54	22,6	2,09	2,06	69,7	1,11
Średnio	**1,19**		**0,74**	**1,26**		**0,88**	**0,77**		**0,45**
TSD+M2									
20×5	0,21	1,7	0,01	0,54	0,8	0,25	0,09	2,5	0,01
20×10	0,14	3,0	0,00	0,61	1,5	0,42	0,07	4,5	0,00
20×20	0,28	5,4	0,02	0,65	2,7	0,37	0,02	8,0	0,00
50×5	2,74	25,5	2,51	1,66	12,6	1,21	1,06	41,3	0,52
50×10	2,39	46,6	1,83	2,41	22,7	1,75	1,74	70,2	0,97
Średnio	**1,15**		**0,87**	**1,17**		**0,80**	**0,60**		**0,30**

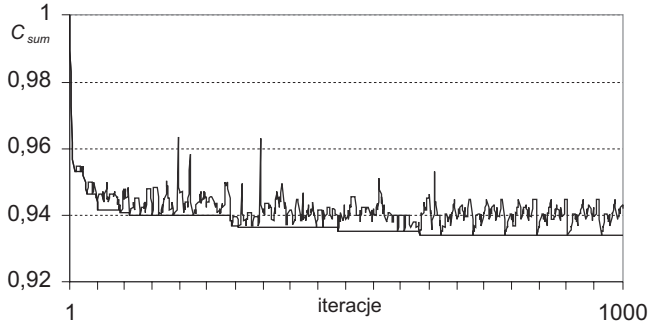
Zastosowanie dynamicznej listy zabronień oraz multiruchu ewidentnie zwiększyło efektywność algorytmu TS dla wszystkich otoczeń zarówno dla 1000, jak i 10 000 iteracji. Dla otoczeń INS i INCH obserwuje się znaczną poprawę współczynnika PRD dla 10 000 względem tej wartości dla 1000 iteracji. Świadczy to o efektywnym rozpraszaniu przeszukiwań przez zastosowane mechanizmy. Narzut czasowy związany z realizacją mechanizmów dywersyfikacyjnych jest pomijalnie mały. Świadczą o tym porównywalne z klasycznym TS czasy obliczeń (oczywiście dla tych samych otoczeń). Strategia wyboru ruchów podczas konstruowania multiruchu ma znaczący wpływ na jakość generowanych rozwiązań końcowych tylko w przypadku otoczenia INS&INCH.

W tabeli 2 został przedstawiony średni błąd algorytmów tabu ze zmiennym otoczeniem: TSVD+M1 oraz TSVD+M2. Z wyników prezentowanych w tabeli 2 wynika, że algorytm TS ze zmiennym otoczeniem, z dynamiczną listą zabronień oraz multiruchem (niezależnie od strategii jego konstruowania) generuje rozwiązania nieznacznie gorsze od najlepszego z testowanych algorytmów, algorytmu TSD+M2, w czasie blisko dwukrotnie krótszym.

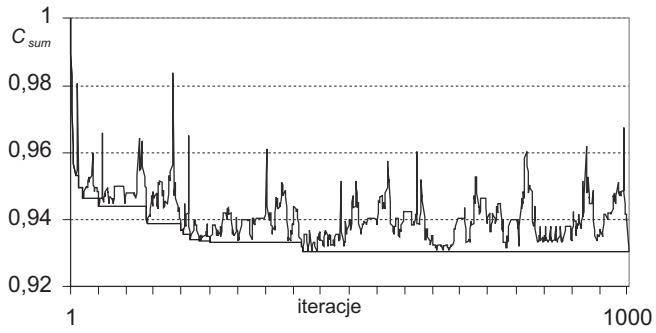
Na rysunkach 4 oraz 5 przedstawiono typowy przebieg obliczeń klasycznego algorytmu TS oraz algorytmu TS z zaprezentowanymi w pracy mechanizmami rozproszeniowymi.

Tabela 2
Średni błąd algorytmów tabu ze zmiennym otoczeniem

Grupa	TSVD+M1			TSVD+M2		
	1000		10 000	1000		10 000
	APRD	ACPU	APRD	APRD	ACPU	APRD
20×5	0,27	1,1	0,01	0,31	1,1	0,02
20×10	0,21	2,0	0,07	0,07	1,9	0,02
20×20	0,21	3,8	0,01	0,26	3,7	0,01
50×5	1,12	20,8	0,51	1,30	19,3	0,50
50×10	1,67	33,1	1,01	1,80	30,1	1,12
Średnio	**0,70**		**0,32**	**0,75**		**0,33**



Rys. 4. Typowy przebieg algorytmu TS z otoczeniem INS



Rys. 5. Typowy przebieg algorytmu TSVD+M1

Wartości funkcji celu rozwiązań bazowych prezentowane na wykresie zostały unormowane względem wartości funkcji celu rozwiązania początkowego wygenerowanego algorytmem NEH.

Na rysunku 4 wyraźnie widać gwałtowne skoki wartości funkcji celu odpowiadające rozwiązaniom generowanym przez multiruchu, w przypadku klasycznego algorytmu TS skoków takich jest znacznie mniej (spowodowanych mechanizmem zabronień), co więcej, można wyraźnie zauważyć, że algorytm wpadł w cykl.

5. Posumowanie

Zaprezentowane techniki dywersyfikacyjne ewidentnie poprawiają efektywność algorytmu tabu dla rozważanego problemu. Wyniki prezentowane w pracy porównywalne są z wynikami generowanymi przez najlepsze algorytmy prezentowane w literaturze.

Literatura

[1] Kohler W.H., Steiglitz K.: *Exact, approximate and guaranteed accuracy algorithms for the flow-shop problem n|2|F|F*. Journal of the ACM 1975, 22, 106–104

[2] Yamada T., Reeves C.R.: *Solving the Csum Permutation Flowshop scheduling problem by Genetic Local Search.* IEEE International Conference on Evolutionary Computation 1998, 230–234

[3] Wang C., Chu C., Proth J.: *Heuristic approaches for n/m/F/SCi scheduling problems.* European Journal of Operational Research 1997, 636–644

[4] Nowicki E., Smutnicki C.: *A fast tabu search algorithm for the permutation flow-shop problem*. European Journal of Operational Research 1996, 91, 160–175

[5] Grabowski J., Pempera J.: *Some local search algorithms for no-wait flow-shop problem with makespan criterion*. Computers & Operations Research 2005, 32, 2197–2212

[6] Grabowski J., Wodecki M.: *A very fast tabu search algorithm for the flow shop problem with makespan criterion*. Computers and Operations Research 2004, 11, 1891–1909

[7] Grabowski J., Wodecki M.: *A very fast tabu search algorithm for job shop problem*. Metaheuristic optimization via memory and evolution. Tabu search and scatter search. Ed. by Cesar Rego and Bahram Alidaee. Boston, Kluwer Academic Publ. 2005, 117–144

[8] Schuster C.J., J.M. Framinan J.M.: *Approximative procedures for no-wait job shop scheduling*. Operations Research Letters 2003, 31, 308–318
[9] M. Nawaz M., E. Enscore E., I. Ham I.: *A Heuristic algorithm for the m-machine, n-job flowshop sequencing problem*. OMEGA The International Journal of Management Science 1983, 11, 91–95
[10] Taillard E.: *Benchmarks for basic scheduling problems*. European Journal of Operational Research 1993, 64, 278–285