ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2002
Seria: AUTOMATYKA z. 134 Nr kol. 1554

Józef GRABOWSKI, Jarosław PEMPERA
Politechnika Wrocławska

METODY DYWERSYFIKACJI PROCESU PRZESZUKIWAŃ W ALGORYTMACH POPRAW DLA PRZEPŁYWOWEGO PROBLEMU KOLEJNOŚCIOWEGO

Streszczenie. W pracy przedstawiono klasyczne przepływowe zagadnienie szeregowania z kryterium minimalizacji terminu zakończenia wykonywania wszystkich zadań. Przedstawiono nowe metody dywersyfikacji (tzw. perturbacje) polegające na jednoczesnym przesunięciu kilku zadań w danej permutacji. Zaprezentowano także tablicę tabu o zmiennej długości. W niniejszej pracy elementy te zastosowano do algorytmu bazującego na technice tabu search. Przeprowadzono eksperymenty obliczeniowe, a uzyskane rezultaty porównano z wynikami aktualnie najlepszych na świecie algorytmów prezentowanych w literaturze.

SOME METHODS OF DIVERSIFICATION IN LOCAL SEARCH ALGORITHMS FOR THE FLOW-SHOP PROBLEM

Summary. The paper deals with the classic flow-shop scheduling problem with the makespan criterion. There are presented and discussed some original methods of diversification (so-called perturbations) associated with the blocks by using of which a few jobs are moved simultaneously in a given permutation, and a tabu list with dynamic length. The algorithm based on tabu search approach is presented. Computational experiments are provided and compared with the results given by the best algorithms proposed in the literature.

1. Wprowadzenie

Problem przepływowy należy do jednych z najbardziej intensywnie badanych problemów szeregowania zadań i jest często traktowany przez wielu badaczy jako "poligon doświadczalny" do testowania nowo powstających metod szeregowania algorytmów popraw [2]. Pomimo prostoty sformułowania problemu, zagadnienia optymalizacji

w systemach przepływowych należą do problemów silnie NP-trudnych. Ogranicza to zakres stosowania algorytmów dokładnych do przykładów o małej liczbie zadań i/lub maszyn. Stąd też dla przykładów z większą liczbą maszyn i zadań proponuje się różnego rodzaju algorytmy przybliżone.

2. Sformułowanie problemu przepływowego

W klasycznym problemie przepływowym mamy m maszyn, na których należy wykonać n zadań produkcyjnych; niech $M=\{M_1,...,M_m\}$ oraz $J=\{J_1,...,J_n\}$ będą (odpowiednio) zbiorami maszyn oraz zadań. Zadanie $J_j \in J$ składa się z m operacji ze zbioru $J_j=(O_{j1},...,O_{jm})$ wykonywanych kolejno na wszystkich maszynach w systemie. Operacja O_{jk} wykonywana jest bez przerw na maszynie M_k w czasie $p_{jk}>0$. Każda maszyna w danej chwili może wykonywać co najwyżej jedną operację. Kolejność wykonywania zadań na wszystkich maszynach jest taka sama i określona przez permutację $\pi=(\pi(1), ...,\pi(n))$. Niech Π oznacza zbiór wszystkich takich permutacji. Chcemy znaleźć permutację $\pi^* \in \Pi$, taką że $C_{max}(\pi^*) = \min_{\pi\in\Pi} C_{max}(\pi)$, gdzie $C_{max}(\pi)$ oznacza termin zakończenia realizacji wszystkich zadań.

3. Algorytmy popraw

Dla rozważanego problemu algorytmy konstrukcyjne oparte na metodach relaksacyjnych i szeregowania priorytetowego stosuje się do rozwiązywania przykładów o bardzo dużych rozmiarach. Natomiast dla przykładów o rozmiarach średnich i dużych znacznie lepsze rezultaty można uzyskać (w rozsądnym czasie) stosując algorytmy konstrukcyjne bazujące na metodzie wstawień, a następnie poprawiać algorytmami popraw [6]. Algorytmy lokalnego przeszukiwania, nazywane niekiedy algorytmami popraw, startują z pewnego rozwiązania początkowego (bazowego), otrzymanego najczęściej przez algorytmy konstrukcyjne, a następnie w kolejnych iteracjach poprawiają to rozwiązanie. Z reguły algorytmy tego rodzaju produkują rozwiązania bliższe optymalnym, ale niestety kosztem zwiększonego czasu obliczeń. W algorytmach tych z rozwiązania bazowego jest generowany zbiór rozwiązań dopuszczalnych, nazywany otoczeniem. Następnie w otoczeniu tym wybiera się rozwiązanie najlepsze, które staje się nowym rozwiązaniem bazowym dla

następnej iteracji. Algorytm kończy działanie w przypadku spełnienia jednego z warunków stopu. Zwykle warunkami tego rodzaju są: przekroczenia limitu czasowego, limitu iteracji, limitu iteracji bez poprawy; osiągnięcie zadowalającej wartości funkcji celu, znalezienie rozwiązania dokładnego itp. Skuteczność działania algorytmów popraw można zwiększyć przez intensyfikację oraz dywersyfikację procesu poszukiwań. Intensyfikacja polega na bardziej intensywnym przeszukiwaniu obszarów (podzbiorów rozwiązań) potencjalnie mogących zawierać dobre rozwiązania. Będą to w szczególności obszary, w których podczas przeszukiwania znaleziono najlepsze rozwiązania. Z kolei proces dywersyfikacji polega na skierowaniu procesu poszukiwań do innych obszarów. W dalszej części pracy przedstawimy wybrane metody i intensyfikacji dywersyfikacji procesu poszukiwań.

Metoda skoku powrotnego. Metoda skoku powrotnego po raz pierwszy została zaproponowana w pracy [7]. Jest ona uważana za jedno z najbardziej skutecznych podejść przy konstrukcji algorytmów tabu search. W metodzie tej podczas działania algorytmu tworzy się listę zawierającą szereg najlepszych rozwiązań znalezionych podczas przeszukiwań, "reprezentujących" korzystne obszary. Po wykonaniu pewnej liczby iteracji bez poprawy następuje wznowienie procesu przeszukiwań od pewnego rozwiązania znajdującego się na tej liście. Metoda ta poprzez powrót do korzystnych obszarów zapewnia intensyfikację przeszukiwań w tych regionach, natomiast przeszukiwania po innej trajektorii powodują przejście do innych obszarów przeszukiwań.

Przeszukiwanie mrówkowe. Zaproponowana przez Dorigo i innych [4] metoda bazuje na „inteligentnym'' zachowaniu się mrówek podczas poszukiwania pokarmu. Zasadniczą rolę w tym procesie odgrywają feromony transportowe. Każda mrówka oznacza nimi drogę od mrowiska do źródła pokarmu. Wielokrotne przejście mrówek wzdłuż tej drogi zwiększa intensywność feromonu. Mrówka poszukując pokarmu podąża oznaczonymi feromonami ścieżkami wybierając je z prawdopodobieństwem proporcjonalnym do intensywności feromonu. Ścieżki oznaczone dużą ilością feromonu są zatem częściej wybierane przez mrówki. W ten sposób zwiększa się intensywność przeszukiwania pewnych regionów, natomiast chaotyczne błądzenie wykonywane przez mrówkę w ostatnim etapie wędrówki zapewnia przejście do nowych obszarów przeszukiwań.

Symulowane podskakiwanie. Proces przeszukiwania przez symulowane podskakiwanie [1] sterowany jest za pomocą wielu parametrów, z których najbardziej istotna jest temperatura. W każdej iteracji, z otoczenia rozwiązania bazowego, wybiera się w sposób losowy (lub według ustalonego porządku) pewne rozwiązanie. Znalezione rozwiązanie staje

się rozwiązaniem bazowym dla następnej iteracji algorytmu, tzn. jest zaakceptowane, gdy posiada lepszą wartość funkcji, w przeciwnym przypadku akceptowane jest z prawdopodobieństwem wzrastającym wraz z malejącą różnicą między wartościami funkcji celu obu rozwiązań i malejącym wraz z malejącą temperaturą. W przypadku zaakceptowania rozwiązania następuje zmniejszenie temperatury (schłodzenie). Zapewnia to intensywniejszą penetrację przeszukiwanego regionu. W przeciwnym przypadku następuje zwiększenie temperatury (podgrzewanie), co zapewnia zaakceptowanie gorszych rozwiązań i tym samym przejście do innych obszarów przeszukiwań.

Przeszukiwanie wzdłuż ścieżki (Path relinking). Skuteczność tej metody bazuje na spostrzeżeniu, że większość przestrzeni rozwiązań problemów optymalizacji dyskretnej ma strukturę tzw. wielkiej doliny, gdzie lokalne mimima znajdują się stosunkowo blisko siebie oraz blisko minimum globalnego. W takim przypadku należy przeszukiwać dokładniej obszary wyznaczone przez znalezione minima lokalne. W metodzie tej ciąg rozwiązań bazowych generowanych podczas przeszukiwania tworzy ścieżkę łączącą dwa wybrane rozwiązania będące minimami lokalnymi, przy czym w kolejnych krokach dba się o to, aby kolejne rozwiązania były coraz bliższe (w sensie pewnej miary) rozwiązania docelowego. W ten sposób realizowane przeszukiwanie daje szansę znalezienia rozwiązania lepszego od tych dwóch rozwiązań oraz globalnego minimum. Metoda przeszukiwania wzdłuż ścieżki została efektywnie wykorzystana w algorytmie genetycznym dla problemu przepływowego w pracy [8]. W swoim algorytmie autorzy wykorzystali tę metodę do konstrukcji operatora krzyżowania mającego na celu intensyfikację przeszukiwań tych korzystnych obszarów. Ponadto wykorzystali oni operator mutacji do dywersyfikacji procesu przeszukiwań.

Perturbacje. Zasadniczą rzeczą w algorytmach popraw jest sposób generowania otoczenia. Z licznych badań dostępnych w literaturze można zauważyć, że pewne sposoby generowania otoczenia są znacznie lepsze od innych. Niemniej w przypadku konkretnej instancji problemu sposób generowania otoczenia może być poważnym ograniczeniem. Co więcej, pewne ograniczenia na proces poszukiwania może przynieść sama metoda optymalizacyjna, np. ograniczenia wynikające z zawartości listy tabu w metodzie tabu search. Główna idea metody perturbacyjnej oparta jest na spostrzeżeniu, że pewne metody optymalizacyjne generują ścieżkę rozwiązań bazowych, która w pewnych momentach znacznie oddala się od lokalnych minimów. Zatem tego rodzaju sytuacje można wykorzystać do szybkiego "zejścia" do minimum za pomocą innego typu ruchów lub do przejścia do

innych regionów przeszukiwań dokonując stosunkowo dużych modyfikacji aktualnego rozwiązania bazowego.

4. Algorytm przeszukiwania z zabronieniami i perturbacjami

W tej części pracy przedstawimy sposób wykorzystania perturbacji w algorytmie Bożejki i innych [3] bazującym na metodzie tabu search (TS). Algorytm TS dla każdego rozwiązania bazowego π konstruuje sąsiedztwo $N(V,\pi)$, $\pi \notin N(V, \pi)$ generowane przez zbiór ruchów V. Z sąsiedztwa tego wybierane jest najlepsze rozwiązanie π^*, które staje się rozwiązaniem bazowym w następnej iteracji. Rozwiązanie π^* porównywane jest (w sensie wartości funkcji celu) z najlepszym rozwiązaniem π^{TS} znalezionym w poprzednich iteracjach oraz jest zapamiętywane w przypadku, gdy $C_{max}(\pi^*) < C_{max}(\pi^{TS})$, tj. $\pi^{TS} = \pi^*$. W celu zapobieżenia powrotom do rozwiązań już przebadanych wprowadza się mechanizm zabronień (tabu). W jego wyniku pewne rozwiązania z sąsiedztwa są uważane za zabronione. Rozwiązania te są odrzucane podczas wyznaczania rozwiązania π^{TS}. Niekiedy rozwiązania zabronione uważa się za perspektywiczne i traktuje jak nie zabronione.

Na jakość działania tego algorytmu znaczący wpływ ma wybór ruchu i/lub sąsiedztwa, formy realizującej mechanizm tabu, strategii poszukiwań.

Ruch i otoczenie

Niech $v=(a,b)$ będzie parą liczb określających pozycje zadań w permutacji π, $a,b \in \{1,2,...,n\}$, $a \neq b$. Dla rozpatrywanego zagadnienia para ta definiuje *ruch* w π, który oznacza usunięcie zadania $\pi(a)$ z pozycji a i wstawienie na pozycję b w π, generując w ten sposób nową permutację π_v:

$$\pi_v = \begin{cases} (\pi(1),...,\pi(a-1),\pi(a+1),...,\pi(b),\pi(a),\pi(b+1),...,\pi(n)), & dla \quad a < b, \\ (\pi(1),...,\pi(b-1),\pi(a),\pi(b),...,\pi(a-1),\pi(a+1),...,\pi(n)), & dla \quad a > b. \end{cases}$$

Dla danej permutacji π termin zakończenia wszystkich zadań może być wyznaczony przy użyciu zależności:

$$C_{max}(\pi) = \max_{1 \le j_1 \le j_2 \le ... \le j_{m-1} \le n} \left(\sum_{j=1}^{j_1} p_{\pi(j)1} + \sum_{j=j_1}^{j_2} p_{\pi(j)2} + ... + \sum_{j=j_{m-1}}^{n} p_{\pi(j)m} \right).$$

Ciąg liczb całkowitych $(j_1,j_2,...,j_{m-1})$ spełniających warunek $1 \le j_1 \le j_2 \le ... \le j_{m-1} \le n$ będziemy nazywać ***drogą*** w π. Drogę $(u_1,u_2,...,u_{m-1})$, taką że

$$C_{max}(\pi)=\sum_{j=1}^{u_1} p_{\pi(j)1}+\sum_{j=u_1}^{u_2} p_{\pi(j)2}+...+\sum_{j=u_{m-1}}^{n} p_{\pi(j)m}$$

będziemy nazywać ***drogą krytyczną*** w π. Ciąg zadań

$$B_k=(\pi(u_{k-1}),\pi(u_{k-1}+1),...,\pi(u_k)) \qquad k=1,2,...,m,\ u_0 \hat{=} 1, u_m \hat{=} n.$$

nazywamy k-tym blokiem w π. Zadania $\pi(u_{k-1})$ oraz $\pi(u_k)$ będziemy nazywać odpowiednio ***zadaniem pierwszym*** oraz ***ostatnim*** w k-tym bloku w π. Z określeń tych wynika, że zadanie $\pi(u_k)$ jest ostatnim zadaniem w k-tym bloku oraz pierwszym w bloku k+1, k=1,2,...,m-1. Ciąg zadań

$$B_k^* = \begin{cases} B_k \setminus \{\pi(u_1)\} & \text{gdy} \quad k=1 \\ B_k \setminus \{\pi(u_{k-1}),\pi(u_k)\} & \text{gdy} \quad 1<k<m \\ B_k \setminus \{\pi(u_{1m-1})\} & \text{gdy} \quad k=m \end{cases}$$

będziemy nazywać k-tym ***blokiem wewnętrznym*** w π.

W pracy [3] pokazano, że bardzo dobre rezultaty można osiągnąć przesuwając jedynie zewnętrzne zadania z bloków i niektóre zadania należące do wnętrza bloków. Będą to w szczególności zadania, których przesunięcie rokuje największą poprawę, tj.

$$\Delta_k(j)=p_{kj}-p_{lj}, \quad J_j \in J$$

gdzie $k \neq l$, l=1,2,...,m, k - określa maszynę, do której bloku wewnętrznego należy zadanie J_j.

Formalizując, niech RZ będzie zbiorem zadań, których przesunięcie rokuje poprawę. Zbiór RZ składa się z zadań pierwszych (lub ostatnich) ze wszystkich bloków oraz jest uzupełniany o najbardziej rokujące zadania z bloków wewnętrznych do liczby LZ zadań. Dalej, niech $ps(j)$ oznacza pozycję w zadania J_j w π. Dla każdego zadania $J_j \in RZ$ definiujemy zbiór ruchów następująco:

$V_j(\pi)=\{(ps(j),b) \mid b \neq ps(j),\ b=1,...,n\}$, jeżeli J_j jest pierwszym lub ostatnim zadaniem z bloku,

$V_j(\pi)=\{(ps(j),b) \mid b \in \{1,...,u_{k-1}\} \cup \{u_{k-1}...,n\}\}$, jeżeli J_j należy do k-tego bloku wewnętrznego. W konsekwencji otrzymujemy zbiór ruchów

$$V(\pi)=\bigcup_{J_j \in RZ} V_j(\pi).$$

Mechanizm zabronień

W naszym algorytmie do realizacji mechanizmu tabu użyliśmy cyklicznej listy T o długości LT. W przypadku wykonania ruchu $v=(a,b)$ w π, podobnie jak [6], do listy dodawany jest element $(\pi(a),\pi(a+1))$ dla $a<b$ i $(\pi(a-1),\pi(a))$ w przeciwnym przypadku. Przed dodaniem nowego ruchu do T „najstarszy" element jest z niej usuwany. Ruch $v=(a,b)$ w

π jest zabroniony, jeżeli istnieje w T element $(\pi(j),\pi(a))$, $j=a+1,...,b$, dla $a<b$, lub $(\pi(a),\pi(j))$, $j=b,...,a-1$, dla $a>b$.

Perturbacje

Niech uporządkowana lista L zawiera pary (j,b), gdzie j określa zadanie, natomiast b „najlepszą" pozycję wstawienia, określające ruchy generujące rozwiązania "lepsze" niż π. Elementy listy L uporządkowane są leksykograficznie względem pozycji wstawiania i pozycji zadania w permutacji bazowej π. Wykonanie perturbacji polega na tym, że po kolei dla każdej pary (j, b) z listy usuwane jest zadanie J_j z permutacji π i wstawiane na pozycję b tej permutacji. Oznacza to, że jednocześnie przemieszczanych jest kilka zadań w permutacji π. Zauważmy, że tego rodzaju modyfikacja może spowodować znaczne zmiany w permutacji bazowej. Może to przynieść sumowanie się korzyści, jakie można by uzyskać z wykonania poszczególnych pojedynczych ruchów, i tym samym może spowodować znaczne zmniejszenie wartości funkcji celu.

Powyższe cechy przynoszą pozytywny efekt, jeżeli rozwiązania bazowe generowane przez algorytm oddalają się od minimum lokalnego. Odległość od minimum lokalnego będziemy wykrywać określając liczbę BP kolejnych iteracji, w których rozwiązanie w iteracji następnej nie jest lepsze od rozwiązania z iteracji poprzedniej. Perturbacja jest wykonywana, jeżeli liczba iteracji bez poprawy oraz liczba korzystnych ruchów, tj. *długość*(L), będą odpowiednio duże.

Algorytm TSN+P

Wielkości oznaczone symbolem (*) oznaczają "najlepsze" wartości znalezione w trakcie przebiegu algorytmu, zerem (^{0}) wartości początkowe, natomiast bez indeksu oznaczają wartości bieżące. Algorytm startuje z permutacji początkowej π^0, która może być otrzymana przy użyciu dowolnego algorytmu. Algorytm zatrzymuje się, jeżeli całkowita liczba iteracji jest większa niż *Maxiter*.

INICJALIZACJA

Podstaw $\pi := \pi^0$, $\pi^* := \pi^0$, $C^* := C_{max}(\pi^*)$, *iter* :=0, $T:=\emptyset$.

PRZESZUKIWANIE

Ze zbioru $NK(\pi) \subseteq V(\pi)$ zawierającego wszystkie ruchy niezabronione i zabronione, ale korzystne (tzn. takie, że $C_{max}(\pi_w)<C^*$), wybierz ruch v taki, że

$$C_{max}(\pi_v) = \min_{w \in NK(\pi)} C_{max}(\pi_w).$$

Jeżeli $C_{max}(\pi_v)<C^*$, podstaw $C^*=C_{max}(\pi_v)$, $\pi^* := \pi_v$.

Jeżeli $BP \geq BP_{min}$, wyznacz listę L. Jeżeli *długość*$(L) \geq L_{min}$, wykonaj perturbację.

WYKONAJ RUCH

Odpowiednio zmodyfikuj listę tabu oraz podstaw $\pi := \pi_v$.

KRYTERIUM ZATRZYMANIA

Jeżeli $iter < Maxiter$, to idź do PRZESZUKIWANIE.

STOP

Parametry metody: LT – długość listy zabronień, BP_{min}-minimalna liczba kolejnych iteracji bez poprawy, L_{min}-minimalna liczba elementów listy L.

5. Wyniki obliczeniowe

Algorytm TSN+P został przetestowany na zestawie przykładów testujących zaproponowanych przez Taillarda [9]. W zestawie tym dla każdej pary $n \times m$: 20×5, 20×10, 20×20, 50×5, 50×10, 50×20, 100×5, 100×10, 100×20, 200×10, 200×20, 500×20, znajduje się 10 trudnych przykładów testujących. W celu wyznaczenia najlepszych parametrów algorytmu przeprowadzono szereg testów algorytmu z różnymi wartościami długości listy tabu, parametrów perturbacji, tj. minimalnej liczby iteracji bez poprawy BP_{min} oraz minimalnej liczby korzystnych ruchów L_{min}. W wyniku tego eksperymentu obliczeniowego najlepsze rezultaty zostały osiągnięte dla następujących wartości parametrów:

L_{min} =2, BP_{min} =5 dla m=20;

L_{min} =2, BP_{min} =2 dla n/m=20;

L_{min} =4, BP_{min} =3 dla n/m=10;

L_{min} =2, BP_{min} =4 dla n/m ≤ 5.

Następnie przeprowadzono szereg testów algorytmu ze skokowo zmieniającą się długością listy tabu (dalej będziemy go oznaczali TSN+PT). Precyzyjniej, niech liczby $P1$ oraz $P2$ będą pewnymi parametrami algorytmu. Modyfikacja algorytmu TSN+P wykorzystująca skokowo zmieniającą się długość listy tabu polega na tym, że algorytm wykonuje na przemian $P1$ iteracji z długością listy tabu LT oraz $P2$ iteracji z długością listy tabu $LT + \Delta LT$. W wyniku przeprowadzonych testów zauważono, że najlepsze rezultaty osiąga się przy parametrach: $\Delta LT = 5$, $P1 = m(n+m)/200$, $P2 = LT + \Delta LT + 15n/100$.

Algorytm TSN+P został zaprogramowany w języku C++ i był testowany na komputerze IBM RISC System 6000, 200 MHz. Do wyznaczenia permutacji początkowej został użyty algorytm NEH [5][6]. Dla każdego przykładu wyliczono następujące wielkości:

C^X – wartość funkcji celu otrzymana algorytmem X, X∈{TSAB, TSGP, TSN, TSN+P, TSN+PT}.

Time – czas obliczeń CPU.

Na podstawie tych wielkości obliczono wartości:

PRD(X) – 100% $(C^X-C^T)/C^T$ średnia procentowa różnica wartości funkcji celu uzyskanej algorytmem X względem wartości C^T uzyskanej algorytmem Taillarda [9],

CPU – średni czas obliczeń w sekundach.

W tabeli 1 porównano rezultaty otrzymane proponowanymi algorytmami TSN+P, TSN+PT z wynikami uzyskanymi przez algorytm TSAB [6], TSGP [5] oraz TSN [3] dla 1000 iteracji.

Z przeprowadzonych badań testujących wynika, że zastosowanie w algorytmie TSN zaproponowanych perturbacji poprawia skuteczność jego działania. Poprawa ta jest największa dla problemów o dużej liczbie maszyn i małym współczynniku *n/m*. Dla problemów z 10 i 20 maszynami dodatkowe korzyści można osiągnąć stosując skokowo zmieniającą się długość listy zabronień. Średni czas obliczeń dla 1000 iteracji algorytmu TSN z proponowanymi modyfikacjami porównywalny jest z czasem obliczeń algorytmu TSAB i znacznie mniejszy od czasu obliczeń algorytmu TSGP.

Tabela 1

Rezultaty porównania algorytmów TSAB, TSGP i TSN

	PRD[%] 1000 iteracji					CPU[s] 1000 iteracji				
n/m	TSAB	TSGP	TSN	TSN+P	TSN+PT	TSAB	TSGP	TSN	TSN+P	TSN+PT
20|5	0,00	0,00	0,00	0	0	1,5	0,5	0,4	0,3	0,2
20|10	0,26	0,25	0,14	0,14	0,09	3,3	0,6	1,0	1,1	1,1
20|20	0,28	0,18	0,21	0,09	0,09	5,0	1,8	2,3	2,2	2,2
50|5	0,00	-0,02	0,01	0,01	0,01	4,1	0,5	1,0	0,8	0,8
50|10	0,11	-0,31	-0,37	-0,36	-0,36	8,2	2,0	2,8	2,7	2,6
50|20	0,78	0,39	0,53	0,50	0,49	19,8	6,2	6,4	6,0	6,0
100|5	-0,01	-0,03	0,02	0,03	0,02	7,4	1,0	1,7	1,2	1,2
100|10	-0,01	-0,10	-0,06	-0,07	-0,09	17,5	2,9	4,5	4,0	4,1
100|20	0,59	-0,35	-0,15	-0,16	-0,18	38,7	14,6	11,4	11,4	11,4
200|10	-0,16	-0,15	0,05	-0,02	-0,02	32,7	6,4	9,0	7,0	7,2
200|20	0,15	-0,67	-0,30	-0,31	-0,33	88,7	34,6	24,0	22,0	22,1
500|20	-0,04	-0,45	-0,25	-0,22	-0,22	240,9	117,5	48,0	54,0	55,5
Średnio	0,16	-0,11	-0,02	-0,03	-0,04					

LITERATURA

1. Amin S.: Zusing Adaptive Temperature Control for Solving Optimisation Problems, Baltzer Journals, 1996.
2. Aarts E.H.I., Lenstra J.K.: Local search in Combinatorial Optimization. John Wiley and Sons Ltd, Chichester 1997, England.
3. Bożejko W., Grabowski J., Pempera J.: Nowy algorytm lokalnej optymalizacji dla zagadnienia kolejnościowego przepływowego. Automatyka 2001, 77-86.
4. Dorigo M., Maniezzo V., Colorni A.: Ant System: Optimization by a Colony of Cooperating Agents. Iee Transactions on Systems, Man, and Cybernetics, 1996, 26 29-41.
5. Grabowski J., Pempera J.: New block properties for the permutation flow-shop problem with application in TS, Journal of Operational Research Society 52, 2001, 210-220.
6. Nowicki E., Smutnicki C.: A fast tabu search algorithm for the permutation flow-shop problem, European Journal of Operational Research 91 (1996), 160-175.
7. Nowicki E., Smutnicki C.: A fast tabu search algorithm for the job-shop problem. Management Science 42 (1996), 797-813.
8. Reeves C. R., Yamada T.: Genetic Algorithms, Path Relinking and the Fowshop Sequencing Problem, Evolutionary Computation Journal (MIT press)
9. Taillard E.: Some efficient heuristic methods for flow-shop sequencing, European Journal of Operational Research 47 (1990) 65-74.

Recenzent: Prof. dr hab. inż. Jacek Błażewicz

Abstract

This paper deals with the classic flow-shop problem. This problem can formulated as follows. There is the set of jobs $J=\{J_1,J_2..,J_n\}$, each of n jobs has to be processed on machines $M_1,M_2,...,M_m$ in that order. A machine can process only one job at a time and preemption of a job is not permitted. The purpose of the optimization is to find such a schedule of jobs on machines that maximum completion time of jobs is minimized. The problem belongs to class NP-hard problems what justifies searching for heuristic algorithms. In the paper we propose some original methods of diversification (so-called perturbations) associated with the blocks by using of which a few jobs are moved simultaneously in a given permutation. Also, we propose a tabu list with dynamic length which is changed cyclically, as the current iteration number of algorithm increases. The algorithm based on tabu search approach is presented. Computational experiments are given and compared with the results yielded by the best algorithms discussed in the literature.