ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2000
Seria: AUTOMATYKA z. 129 Nr kol. 1474

Józef GRABOWSKI, Jarosław PEMPERA
Politechnika Wrocławska

SZEREGOWANIE ZADAŃ NA DWÓCH STANOWISKACH Z MASZYNAMI RÓWNOLEGŁYMI ZE SPECYFICZNYMI WYMAGANIAMI

Streszczenie. Praca poświęcona jest zagadnieniu szeregowania zadań w systemie przepływowym z ograniczeniami „bez czekania", który składa się z dwóch stanowisk zawierających po kilka maszyn w każdym stanowisku. W pracy przedstawia się algorytmy popraw, bazujące na technice przeszukiwania genetycznego, z zabronieniami oraz symulowanego wyżarzania. Przedstawiono wyniki obliczeniowe algorytmów oraz analizę porównawczą.

SEQUENCING OF JOBS IN CONSTRAINED TWO STAGES FLOW-SHOP PROBLEM WITH PARALLEL MACHINES

Summary. In the paper two stages flow-shop problem with the "no-wait" requirements and parallel machines is considered. Some approximation algorithms, computational results and discussion of the performance of algorithms are presented.

1. Wstęp

W rozważanym problemie park maszynowy podzielony jest na dwa stanowiska obróbcze $Z=(Z_1,Z_2)$. Każde ze stanowisk $Z_i=\{M_{i1},...,M_{im_i}\}$, $i=1,2$ zawiera $m_i\geq 1$ identycznych maszyn. Zadania ze zbioru $J=\{1,2,...,n\}$ wykonywane są kolejno w każdym stanowisku, tzn. zadanie $j\in J$ najpierw wykonywane jest w stanowisku Z_1 w czasie p_{j1}, następnie w Z_2 w czasie p_{j2}. Zatem każde zadanie składa się z dwóch operacji $j=(o_{j1},o_{j2})$, gdzie o_{ji} oznacza operację j-tego zadania wykonywaną na i-tym stanowisku. Operacja o_{ji} może być wykonywana na jednej z m_i identycznych maszyn równoległych, należących do i-tego stanowiska. Ograniczenie „bez czekania" oznacza natychmiastowe rozpoczęcie wykonywania zadania w stanowisku 2 po zakończeniu obróbki w stanowisku 1. Zadanie optymalizacji polega na przyporządkowaniu zadań do maszyn oraz ustaleniu harmonogramu (terminów rozpoczęcia lub/i zakończenia) wykonywania zadań na stanowiskach tak, aby czas wykonania wszystkich zadań był możliwie najmniejszy. W literaturze rozważany problem jest oznaczany symbolicznie w następujący sposób : **F2| bez czekania,** $m_1,m_2\geq 2|C_{max}$.

Problem przepływowy z ograniczeniami „bez czekania" oraz identycznymi maszynami równoległymi w każdym stanowisku należy do problemów NP-zupełnych, ponieważ problem dla dwóch stanowisk z $m_1=1$ oraz $m_2=2$ (patrz przegląd [2]) należy do tej klasy. Szczególny przypadek, w którym $m_1=m_2=1$ (Gilmore, Gomory [3]), jest rozwiązywalny w czasie $O(n \ln n)$.

2. Model matematyczny

Niech S_{ji} oraz $C_{ji}=S_{ji}+p_{ji}$ oznaczają odpowiednio termin rozpoczęcia oraz zakończenia wykonywania j-tego zadania w i-tym stanowisku, μ_{ji} oznacza numer maszyny przyporządkowanej do wykonywania j-tego zadania w i-tym stanowisku. Para $<\mu_{j1},\mu_{j2}>$ określa *sposób* wykonania zadania j, oczywiście liczba wszystkich możliwych sposobów wynosi m_1m_2. Dla każdego stanowiska Z_i, $i=1,2$ zbiór zadań J można podzielić na rozłączne podzbiory zadań J_{ik} wykonywanych na maszynach należących do tego stanowiska, gdzie $J=\bigcup_{k=1}^{m_i} J_{ik}$, $J_{ik}\cap J_{il}=\emptyset$, $k\neq l$ $k,l=1,\ldots,m_i$. Przez π_{ik} będziemy oznaczać kolejność wykonywania zadań ze zbioru J_{ik}, a przez $n_{ik}=|J_{ik}|$ liczność tego zbioru. Kolejność wykonywania zadań na wszystkich maszynach oznaczamy $\pi=(\pi_{11},\ldots,\pi_{1m_1}, \pi_{21},\ldots,\pi_{2m_2})$. Oczywiście, podanie π jednoznacznie określa kolejność wykonywania zadań na maszynach oraz przydział zadań do maszyn. Opisane ograniczenia można sformułować w następujący sposób:

$$C_{\pi_{ik}(j)k} \leq S_{\pi_{ik}(j+1)k} \qquad i=1,2,\ k=1,\ldots,m_i, j=1,\ldots,n_{ik}-1 \tag{1}$$

$$C_{j1} \leq S_{j2} \qquad j=1,\ldots,n, \tag{2}$$

$$C_{j1}=S_{j2} \qquad j=1,\ldots,n, \tag{3}$$

przy czym ograniczenia (1) i (2) zabraniają, odpowiednio, wykonywania kilku zadań jednocześnie przez tę samą maszynę oraz wykonania zadania na kilku maszynach jednocześnie. Natomiast (3) odpowiada ograniczeniu „bez czekania".

Rozpatrywane zagadnienie można łatwo zamodelować w postaci grafu $G(\pi)=(N,E(\pi)\cup E\cup E_c)$, gdzie $N=\{1,\ldots,n\}\times\{1,2\}$ oznacza zbiór węzłów reprezentujących operacje obciążonych wagą p_{ji}. Zbiory $E(\pi)$, E, E_c są zdefiniowane następująco: $E(\pi)=\bigcup_{i=1}^{2}\bigcup_{k=1}^{m_i}\bigcup_{j=1}^{n_{ik}-1}\{((\pi_{ik}(j),i),(\pi_{ik}(j+1),i))\}$ z wagą równą zeru; $E=\bigcup_{j=1}^{n}\{((j,1),(j,2))\}$ z wagą równą zeru; $E_c=\bigcup_{j=1}^{n}\{((j,2),(j,1))\}$ z wagą równą $-p_{j1}-p_{j2}$. Dla grafu $G(\pi)$ przez $C_{\max}(\pi)$ oznaczymy długość najdłuższej drogi w tym grafie (drogi krytycznej). Oczywiście, najwcześniejszy termin zakończenia operacji o_{ji}, $j=1,\ldots,n$, $i=1,2$ równa się długości najdłuższej drogi „dochodzącej" do odpowiadającego jej węzła w grafie $G(\pi)$, natomiast najwcześniejszy

termin zakończenia wszystkich zadań równa się $C_{max}(\pi)$. Teraz nasz problem optymalizacyjny można sformułować jako znalezienie permutacji π^*, dla której graf $G(\pi^*)$ nie zawiera cykli o dodatniej wartości oraz posiada najmniejszą długość ścieżki krytycznej. Termin zakończenia wszystkich zadań można wyznaczyć w czasie $O(n)$ wykorzystując niżej przedstawione własności zbioru dopuszczalnych rozwiązań.

Definicja 1. Kolejność π jest *wewnętrznie zgodna*, jeżeli dla każdej pary zadań i oraz j wykonywanych przy użyciu tej samej maszyny w stanowisku 1 (tzn. $\mu_{i1}=\mu_{j1}$), a także wykonywanych przy użyciu tej samej maszyny w stanowisku 2 (tzn. $\mu_{i2}=\mu_{j2}$); kolejność wykonywania tych zadań jest taka sama na obu tych maszynach (tzn. zadanie i jest wykonywane przed j lub odwrotnie).

Można udowodnić, że dla rozważanego problemu zbiór rozwiązań dopuszczalnych, spełniających ograniczenia (1)-(3), składa się z kolejności wewnętrznie zgodnych.

Definicja 2. Permutację n-elementową wszystkich zadań α nazywamy *sekwencją ładującą* [11] dla kolejności wewnętrznie zgodnej π, jeżeli dla każdej pary zadań i oraz j, takiej że zadanie i jest wykonywane przed zadaniem j w π na pewnej maszynie, to w permutacji α zadanie i występuje przed zadaniem j.

3. Algorytmy konstrukcyjne

W pracy [7] zaproponowano dwa algorytmy konstrukcyjne dla problemu $F2|$ bez czekania, $m_1=m_2=m\geq 2|C_{max}$, będące kombinacją szeregowania listowego na maszynach równoległych oraz algorytmu Gilmore-Gomory.

Algorytm H_a

Krok 1. Podziel zbiór maszyn na podzbiory zawierające dokładnie jedną maszynę z każdego stanowiska $\{M_{11},M_{21}\}$, $\{M_{12},M_{22}\},\ldots,\{M_{1m},M_{2m}\}$.

Krok 2. Traktując parę $\{M_{1i},M_{2i}\}$ jako pojedynczą maszynę M_i oraz sumę $T_j=p_{j1}+p_{j2}$ jako czas wykonania zadania j ($j=1,\ldots,n$) na maszynie M_i przy użyciu algorytmu szeregowania listowego przydziel każdemu zadaniu parę maszyn do jego wykonania.

Krok 3. Dla każdej pary maszyn $\{M_{1k},M_{2k}\}$ $k=1,\ldots,m$ wyznacz optymalną kolejność wykonywania zadań algorytmem Gilmore-Gomory.

Po wykonaniu wszystkich czynności kroku 2 oraz kroku 3 uzyskujemy kolejność wykonania zadań na wszystkich maszynach.

Algorytm H_b

Krok 1. Taki sam jak H_a

Krok 2. Taki sam jak H_a, przy czym lista jest sortowana wg nierosnącej wartości T_j (LPT).

Krok 3. Taki sam jak H_a.

Złożoność obliczeniowa wyżej opisanych algorytmów wynosi $O(mn+n \ln n)$.

Zmodyfikowany algorytm NEH (ZNEH)

Algorytm NEH[5] jest algorytmem opartym na technice wstawiania. Badania eksperymentalne dla problemów permutacyjnych wskazują na jego wysoką efektywność. Przy stosowaniu tego algorytmu dla rozważanego problemu wprowadziliśmy modyfikację polegającą na tym, że w każdym kroku wstawiania analizowane są wszystkie możliwe przyporządkowania zadania do maszyn. Zadanie wstawiane jest na wszystkie możliwe pozycje w częściowej sekwencji ładującej.

4. Algorytmy popraw

4.1. Algorytm tabu seach (TS)

Technika TS jest aktualnie jednym z najlepszych podejść do konstrukcji algorytmów heurystycznych do rozwiązywania zagadnień optymalizacji dyskretnej, w tym również zagadnień szeregowania [6], [8]. Algorytm TS dla każdego rozwiązania bazowego generuje zbiór rozwiązań *sąsiednich* (sąsiedztwo). Sąsiedzi generowani są przez modyfikacje rozwiązania bazowego zwane *ruchami*. Z sąsiedztwa wybierane jest najlepsze rozwiązanie, które staje się rozwiązaniem bazowym w następnej iteracji. W celu zapobieżenia powrotu do rozwiązań już przebadanych wprowadza się mechanizm zabronień (tabu). Algorytm rozpoczyna działanie od rozwiązania bazowego otrzymanego najczęściej algorytmem konstrukcyjnym, a kończy po spełnieniu warunku zatrzymania.

Ruchy i sąsiedzi

Modyfikacja (ruch) rozwiązania bazowego powinna uwzględniać zarówno zmianę przydziału zadań do maszyn, jak również zmianę kolejności na maszynach. Zatem rozważając reprezentację permutacyjną problemu oznaczmy, podobnie jak w [6], przez $a = (\mu^{\pi}_{j1}, \mu^{\pi}_{j2})$ - parę indeksów maszyn przydzielonych do wykonania zadania j w π, na których wykonywane jest zadanie j odpowiednio na pozycjach $y = (ps^{\pi}_{j1}, ps^{\pi}_{j2})$. Ruch typu *włóż* $v=(a,y,b,z)$ oznacza przeniesienie zadania j znajdującego się na maszynach określonych przez a na pozycjach y oraz

wstawieniu go na maszyny określone przez parę indeksów maszyn b na pozycje z. Zbiór wszystkich możliwych ruchów $V(\pi)$ można zdefiniować następująco:

$$V(\pi) = \{v=(a,y,b,z): [G(\pi_v) \text{ nie zawiera cyklu dodatniego}, \ a=(\mu^{\pi}_{j1},\mu^{\pi}_{j2}), \ y=(ps^{\pi}_{j1},ps^{\pi}_{j2})] \wedge [(1\le z_i\le n_{ib_i}, b_i\in a) \vee (1\le z_i\le n_{ib_i}+1, b_i\notin a)] \wedge [M_{b_i}\in Z_i, i\in\{1,2\}, j\in J]\}.$$

Liczbę elementów tego zbioru można zmniejszyć eliminując ruchy *redundantne* oraz ruchy, o których wiemy apriorycznie, że nie przyniosą poprawy wartości funkcji celu, a w szczególności ruchy dotyczące zadań nie należących do drogi krytycznej.

Mechanizm Tabu

Do realizacji mechanizmu tabu została użyta cykliczna lista T o długości L. W przypadku wykonania ruchu $v=(a,y,b,z)$ w π (podobnie jak [6]) do listy dodawane jest :

$(k,\pi_k(y_s), \pi_k(y_s+1))$, gdy zadanie przesuwane jest na maszynie M_k w prawo,

$(k,\pi_k(y_s-1), \pi_k(y_s))$, gdy zadanie przesuwane jest na maszynie M_k w lewo,

$(k,\pi_k(y_s-1), \pi_k(y_s))$, $(k,\pi_k(y_s), \pi_k(y_s+1))$, gdy zadanie usuwane jest z maszyny M_k,

$k\in a$, $y_s\in y$.

Ruch $v=(a,y,b,z)$ zadania j w π jest *zabroniony*, jeżeli istnieje w T element $(k,j,\pi_k(x))$ dla $z_i\le x\le n_{ib_i}$ lub element $(k,\pi_k(x),j)$ dla $1\le x<z_i$, $k\in b$, $z_i\in z$, $i\in\{1,2\}$.

Strategia przeszukiwania

W naszym algorytmie wybiera się zadania, których przynajmniej jedna operacja należy do ścieżki krytycznej. Dla każdego wybranego zadania wyznacza się nowe najlepsze położenie analizując wszystkie możliwe ruchy typu włóż, nie powodujące powstania cyklu dodatniego. W celu przyspieszenia procesu obliczeń wstępnie eliminuje się pewne ruchy powodujące wygenerowanie kolejności niedopuszczalnej (tzn. nie posiadającej cechy wewnętrznej zgodności). Następnie wyznacza się szybką metodą wartość funkcji celu (dla grafu nie zawierającego cyklu dodatniego) i jeżeli jest ona mniejsza od najmniejszej do tej pory znalezionej, wówczas sprawdza się czy graf $G(\pi_v)$ nie zawiera cyklu dodatniego. Jeżeli spełniony jest ten warunek, wówczas rozwiązanie takie jest zapamiętywane jako reprezentant wszystkich ruchów, które można wykonać rozpatrywanym zadaniem. Oznaczmy przez $MR(\pi)$ zbiór wszystkich reprezentantów.

W przedstawionym poniżej algorytmie wartości oznaczone górnym indeksem w postaci (*) oznaczają dotychczas znalezione najlepsze wartości, zerem (°) - wartości początkowe, natomiast bez indeksu oznaczają wartości bieżące. Algorytm startuje od rozwiązania

początkowego π^*, otrzymanego algorytmem ZNEH, natomiast kończy działanie po wykonaniu *limititer* iteracji.

Algorytm TS

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

Krok 1. Podstaw $iter := iter + 1$.
Spośród ruchów nie zabronionych $w \in MR(\pi)$ (lub zabronionych ale korzystnych tzn. takich, że $C_{\max}(\pi_w) < C^*$) wybierz ruch v taki, że $C_{\max}(\pi_v) = \min_{w \in MR(\pi)} C_{\max}(\pi_w)$.

Krok 2. Jeżeli $C_{\max}(\pi_v) < C^*$, to podstaw $C^* = C_{\max}(\pi_v)$, $\pi^* := \pi_v$.
Odpowiednio zmodyfikuj listę tabu oraz podstaw $\pi := \pi_v$.

Krok 3. Jeżeli $iter \geq limititer$, to STOP. W przeciwnym przypadku idź do kroku 1.

4.2. Algorytm genetyczny (GA)

Algorytm genetyczny w każdej iteracji t generuje podzbiór $P(t) = \{x_1, \ldots, x_p\}$ zbioru wszystkich rozwiązań zwany *populacją* (gdzie p oznacza rozmiar populacji). Każde z rozwiązań nazywamy *osobnikiem*, natomiast funkcję oceniającą osobnika *funkcją dopasowania* (FD). Nową populację tworzy się przez selekcję osobników najlepiej dopasowanych i dokonaniu na nich pewnych zmian za pomocą operatorów genetycznych. Wybrane osobniki nazywa się *rodzicami*, natomiast wygenerowane osobniki *potomkami*. Część lub wszyscy potomkowie mogą zastąpić najgorzej dostosowane osobniki w populacji. Dla rozpatrywanego (zagadnienia podobnie jak w [10]) osobnika można przedstawić za pomocą sekwencji ładującej oraz przydziału zadań do maszyn, natomiast funkcja dopasowania jest wprost funkcją celu optymalizacji. W badanym przez nas algorytmie proces modyfikacji genetycznych dotyczył sekwencji ładującej, natomiast przydział do maszyn był ustalany za pomocą losowego algorytmu zstępującego.

Krok 1. Wybierz r rodziców z $P(t)$. Poddaj wszystkich rodziców operacjom genetycznym (krzyżowaniu, mutacji, inwersji z prawdopodobieństwem odpowiednio p_k, p_m, p_i).

Krok 2. Wybierz potomków, którzy przejdą do następnej populacji.

Krok 3. Wybierz osobniki, które zostaną zastąpione przez wybranych potomków.

Do krzyżowania została użyta metoda PMX, a do mutacji użyto operatora przesunięcia (więcej szczegółów w [4],[9]). Rodzice wybierani są z prawdopodobieństwem proporcjonalnym do różnicy wartości FD najgorszego w populacji osobnika oraz wartości FD rozpatrywanego osobnika. Osobniki do odrzucenia są wybierane z prawdopodobieństwem proporcjonalnym do FD osobnika, a potomkowie wybierani są tylko tacy, którzy mają inną niż dla innych osobników wartość FD. Populacja początkowa została wygenerowana przez

mutację rozwiązania otrzymanego algorytmem ZNEH. Algorytm kończy działanie po wykonaniu *limititer* iteracji.

Losowy algorytm zstępujący

Wstaw wszystkie zadania do listy L.
Dopóki lista L nie jest pusta, wykonuj:
Wylosuj i usuń zadanie j z listy. Ustal najlepszy przydział zadania j do maszyn.
Jeżeli nastąpiła poprawa, zachowaj nowy przydział zadania j do maszyn oraz wstaw usunięte zadania do listy.

4.3. Algorytm symulowanego wyżarzania (SA)

W badanym przez nas algorytmie opartym na technice symulowanego wyżarzania dla pewnego rozwiązania bazowego π rozwiązanie sąsiednie generuje się przez wylosowanie zadania należącego do ścieżki krytycznej oraz wstawienie go na wylosowaną (ze wszystkich możliwych) pozycję w sekwencji ładującej. Po wstawieniu zadania przydziela się dla niego „najlepszy" zestaw maszyn. Dla każdego wygenerowanego rozwiązania sąsiedniego π_v obliczana jest wartość $C_{max}(\pi_v)$, która jest porównywana z wartością $C_{max}(\pi)$. Jeżeli $C_{max}(\pi_v) \leq C_{max}(\pi)$, wówczas π_v staje się nowym rozwiązaniem bazowym, tj. $\pi=\pi_v$, w przeciwnym przypadku rozwiązanie π_v może stać się rozwiązaniem bazowym z prawdopodobieństwem p, zależnym od wartości różnicy $\Delta=C_{max}(\pi_v)-C_{max}(\pi)$ oraz wartości parametru T_t, zwanego temperaturą i wyznaczanym ze wzoru $p=\exp(-\Delta/T_t)$. Temperatura zmienia się co n^2 iteracji wg zależności $T_t=\lambda T_{t-1}$. Algorytm kończy działanie, gdy temperatura spadnie poniżej wartości T_k. Początkowe rozwiązanie bazowe π otrzymane jest algorytmem ZNEH, natomiast wartość początkowa temperatury T_0 oraz λ są parametrami algorytmu.

5. Wyniki badań testujących

Algorytmy zaproponowane w niniejszej pracy zostały zakodowane w języku C++ i były testowane na komputerze IBM RISC System/6000, 200 MHz. Badania testowe przeprowadzono na przykładach, dla których wartości p_{j1}, p_{j2} zostały wygenerowane według rozkładu równomiernego ze zbioru $\{1,2,...,100\}$. Przeprowadzono badania dla liczby maszyn $m=m_1=m_2=\{2,3,4\}$ oraz liczby zadań $n=\{10,20,50,100\}$. Dla każdej kombinacji wartości n, m wygenerowano 30 przykładów testujących, otrzymując w sumie 360 problemów.

Dla badanych algorytmów przyjęto następujące wartości parametrów: TS: $L=10$, *limintiter*=1000; GA: $r=10$, $p=50$, $p_k=0.9$, $p_m=0.05$, $p_i=0.05$, *limintiter*=1000; SA: $T_0=1000$, $T_k=1$, $\lambda=0{,}91$ (w przybliżeniu 100 zmian temperatury).

Dla każdego przykładu wyznaczono następujące wielkości :

C^H - wartość funkcji celu otrzymana algorytmem H, $H=\{H_a,H_b,ZNEH,TS,GA,SA\}$,
$C^m = \min_H(C^H)$ - najmniejsza wartość funkcji celu otrzymana jednym z algorytmów.

Na podstawie tych wielkości obliczono:
$r^H = 100\%(C^H - C^m) / C^m$ - względny błąd rozwiązania otrzymanego algorytmem H w stosunku do najlepszego rozwiązania otrzymanego jednym z algorytmów.
p^H- procent przykładów, dla których $C^H = C^m$.
$r_{śr}^H$ - średni względny błąd dla algorytmu H.
CJI – czas jednej iteracji w ms.

Tablica 1

	n=10			n=20			n=50			n=100		
Algorytm	$r_{śr}^H$	p^H	CJI	$r_{śr}^H$	p^H	CJI	$r_{śr}^H$	p^H	CJI	$r_{śr}^H$	p^H	CJI
$m_1=m_2=2$												
H_a	31.4	0.0	-	30.2	0.0	-	27.2	0.0	-	26.4	0.0	-
H_b	31.1	0.0	-	30.1	0.0	-	25.9	0.0	-	26.2	0.0	-
$ZNEH$	5.98	0.0	-	4.53	0.0	-	2.59	0.0	-	1.33	0.0	-
TS	1.02	46.7	1	0.06	83.3	5	0.04	86.7	40	0.04	90.0	189
GA	0.28	70.0	34	1.15	26.7	69	1.52	3.3	160	1.03	0.0	311
SA	3.35	3.3	9	3.74	0.0	44	1.88	10.0	275	0.93	10.0	1099
$m_1=m_2=3$												
H_a	38.4	0.0	-	36.5	0.0	-	29.9	0.0	-	28.1	0.0	-
H_b	36.5	0.0	-	36.8	0.0	-	29.2	0.0	-	27.4	0.0	-
$ZNEH$	7.06	0.0	-	4.98	0.0	-	3.44	0.0	-	2.15	0.0	-
TS	1.88	16.7	1	0.19	73.3	6	0.00	100	52	0.00	1000	252
GA	0.04	93.3	19	1.03	36.7	44	2.24	0.0	105	1.77	0.0	205
SA	3.35	6.7	9	3.55	0.0	45	2.85	3.3	299	1.93	0.0	1201
$m_1=m_2=4$												
H_a	38.2	0.0	-	42.8	0.0	-	30.3	0.0	-	29.5	0.0	-
H_b	29.7	0.0	-	40.0	0.0	-	29.6	0.0	-	28.6	0.0	-
$ZNEH$	3.73	36.7	-	4.83	0.0	-	3.56	0.0	-	2.57	0.0	-
TS	0.94	66.7	1	0.32	70.0	12	0.00	100	51	0.00	100	252
GA	0.05	93.3	12	1.11	36.7	34	2.20	0.0	89	2.27	0.0	159
SA	1.94	43.3	7	3.73	0.0	35	3.00	0.0	294	2.49	0.0	1145

Na podstawie otrzymanych wyników przedstawionych w tablicy 1 można stwierdzić, że rozwiązania otrzymane wyspecjalizowanymi algorytmami konstrukcyjnymi są znacznie poprawiane przez algorytmy popraw (błąd względny większy 30%). Podobnie jak dla innych zagadnień kolejnościowych algorytm oparty na technice wstawiania ZNEH generuje dość dobre rozwiązania początkowe (błąd względny ok. 7 %, zmniejszający się wraz ze wzrostem liczby zadań). Z testowanych przez nas algorytmów opartych na technikach tabu search, symulowanego wyżarzania oraz przeszukiwania genetycznego najlepszy okazał się

algorytm TS. Jego przewaga nad innymi jest szczególnie widoczna dla dużych wartości n. Dla małych wartości n najlepsze rezultaty daje algorytm GA.

LITERATURA

1. Aarts E.H.L., van Laarhoven P.J.M.: Simulated Annealing: A pedestrian review of the theory and some applications. In: Pattern Recognition and Applications, Eds. Diviijver P.A. and Kittler J., Springer, 1987, Berlin.
2. Gilmore, P. C., Gomory, R. E.: Sequencing a One State Variable Machine. A Solvable Case of the Travelling Salesman Problem. Operations Research, 1964, 12, 655-679.
3. Hall N. G., Sriskandarajanh C.: A survey of machine scheduling problems with blocking and no-wait in process. Operations Research, 1996, 44, 510-525.
4. Michalkiewicz Z.: Genetic Algorithms + Data Structures =Evolution Programs. Spinger-Verlag, Berlin Heidelberg 1992,1994,1996.
5. Nawaz M., Enscore Jr. E. E., Ham I.: A heuristic algorithm for m-machine, n-job flow-shop scheduling problem. OMEGA International Journal of Management Science, 1983, 11, 91-95.
6. Nowicki E.: Metoda Tabu w problemach szeregowania zadań produkcyjnych. Seria Monografie, Oficyna Wydawnicza PWr, Wrocław, 1999.
7. Sriskandarajanh C.: Performance of scheduling algorithms for no-wait flowshop with parallel machines. European Journal of Operational Research 1993, 70, 365-378.
8. Smutnicki C.: Optimization and control in Just-Time Manufacturing System. Seria Monografie, Oficyna Wydawnicza PWr, Wrocław, 1997.
9. Wala K., Chmiel W.: Operatory genetyczne dla permutacyjnych zagadnień optymalizacji. Zeszyty Naukowe Politechniki Śląskiej, s.Automatyka, z.114, Gliwice 1994, 272-287.
10. Wala K., Chmiel W.: Evolutionary Algorithm for the CIM Production Scheduling. Zeszyty Naukowe Politechniki Śląskiej, s.Automatyka, z.125, Gliwice 1998, 249-259.
11. Wittrock R. J.: An adaptable scheduling algorithm for flexible flow lines. Operations Research, 1988, 36, 445-453.

Recenzent: Prof.dr hab.inż. T.Sawik

Abstract

This paper deals with sequencing of jobs in the two stages flow-shop problem with "no-wait" requirements. This problem can formulated as follows. There is the set of jobs $J=\{1,2,...,n\}$ each of n jobs has to be processed on two stages Z_1, Z_2 in that order. Each stage Z_i contains m_i parallel machines. A machine can process only one job at a time and preemption of a job is not permitted. Furthermore, between stages Z_1 and Z_2 there are "no wait" requirements. The purpose of the optimization is to find such a schedule of jobs on machines that maximum completion time of jobs is minimized. Our problem belongs to the class of NP-hard problems what justifies searching for heuristic algorithms. In the paper we propose several approximation algorithms based on tabu search, simulated annealing and genetic approach. Finally, the computational results and discussion of the performance of algorithms are presented.