Wojciech BOŻEJKO, Mariusz UCHROŃSKI, Mieczysław WODECKI
Politechnika Wrocławska

BLOKI DLA DWUMASZYNOWEGO SUMOKOSZTOWEGO PROBLEMU PRZEPŁYWOWEGO

Streszczenie. W pracy jest rozpatrywany dwumaszynowy problem przepływowy z minimalizacją sumy kosztów spóźnień, będący uogólnieniem popularnego NP-trudnego problemu jednomaszynowego z tym kryterium. Proponujemy wprowadzenie nowych eliminacyjnych własności blokowych pozwalających na przyspieszenie działania przybliżonych algorytmów lokalnych poszukiwań rozwiązujących ten problem oraz poprawę jakości wyznaczanych przez nie rozwiązań.

BLOCKS FOR TOTAL WEIGHTED TARDINESS FLOW SHOP PROBLEM WITH TWO MACHINES

Summary. In the work we consider a two-machine flow shop problem with minimization of the sum of tardiness costs, being a generalization of the popular NP-hard single-machine scheduling problem with this criterion. We propose the new elimination block criteria allowing for acceleration of the approximate local search algorithms solving this problem and improvement of the quality of solutions determined by them.

1. Wprowadzenie

W pracy rozważamy problem szeregowania zadań na dwóch maszynach pracujących w systemie przepływowym i przy założeniu, ze kolejność wykonywania zadań przez maszyny jest identyczna (tzw. problem permutacyjny). Kryterium jest ważona suma spóźnień zadań w stosunku do zakładanych maksymalnych momentów zakończenia (*due dates*). Schaller [11] rozważa dwumaszynowy problem przepływowy z kryterium sumy spóźnień (*total tardiness*, $F2||\sum T_i$ wg. notacji Grahama [3] – tj. bez wag) podając 5 własności eliminacyjnych typu: jeśli zachodzą pewne zależności pomiędzy parametrami zadań i i k, to w rozwiązaniu optymalnym zadanie i jest przed zadaniem k. Podaje też, za pracą Koulamus [6] ideę dowodu silnej NP-trudności tego problemu, a więc także rozważanego w niniejszej pracy problemu $F2||\sum w_iT_i$. Z kolei Lee i Kim [7] rozważają dwumaszynowy problem przepływowy $F2||\sum T_i$ z dodatkowymi ograniczeniami na momenty dostępności zadań na pierwszej maszynie.

Przeglądając wyniki literaturowe dotyczące metaheurystyk zaprojektowanych do rozwiązywania rozpatrywanego problemu sumokosztowego wymienić można pracę Liao et al. [8]. Autorzy proponują do rozwiązania problemu wyjorzystanie algorytmu przeszukiwania za zabronieniami (*tabu search*). Ruiz i Stützle [10] proponują wykorzy-

stanie algorytmu zachłannego. Z kolei Khalili i Tavakkoli-Moghaddam [5] rozpatrują dla rozważanego problemu sumokosztowego model optymalizacji wielokryterialnej z dwoma funkcjami celu: maksymalnym czasem zakończenia wykonywania zadań (*makespan*) oraz ważoną sumą kosztów spóźnień (*total weighted tardiness*) proponując wykorzystanie algorytmu elektromagnetycznego.

W dalszej części pracy przedstawiamy dokładny opis rozpatrywanego problemu $F2||\sum w_iT_i$ oraz jego model matematyczny. Dowodzimy specyficznych własności (tzw. własności eliminacyjnych bloków [1]), które znacznie poprawiają efektywność algorytmów rozwiązywania wielomaszynowych problemów sumokosztowych. W przypadku ich zastosowania w konstrukcjach algorytmów opartych na metodzie lokalnych poszukiwań pozwalających na znaczne zmniejszenie, generowanego w każdej iteracji, otoczenia. Problem przepływowy dla 2 maszyn z kryterium $C_{\max}$ (tj. minimalizacją momentu zakończenia wykonywania wszystkich zadań) jest problemem wielomianowym, natomiast z rozważanym w niniejszej pracy kryterium sumokosztowym $\sum w_iT_i$ należy już do klasy problemów NP-trudnych.

2. Opis problemu oraz jego model matematyczny

Dwumaszynowy permutacyjny problem przepływowy z minimalizacją sumy kosztów spóźnień można sformułować następująco:

Problem: Dany jest zbiór zadań $\mathcal{J} = \{1, 2, \ldots, n\}$ oraz zbiór maszyn $\mathcal{M} = \{1, 2\}$. Zadanie $i \in \mathcal{J}$ składa się z dwóch operacji operacji O_{i1}, O_{i2}. Operacja O_{ik} odpowiada czynności wykonywania zadania i na maszynie $k \in \mathcal{M}$. Dla zadania $i \in \mathcal{J}$, niech d_i będzie *żądanym terminem zakończenia*, w_i *współczynnikiem funkcji kary*, a p_{ik} *czasem wykonywania* operacji O_{ik}. Należy wykonać zadania na maszynach przy czym, muszą być spełnione następujące ograniczenia:

(*a*) każde zadanie należy wykonać na pierwszej, a następnie na drugiej maszynie,
(*b*) wykonywanie zadania nie może być przerwane,
(*c*) zadanie może być wykonywane jednocześnie tylko na jednej maszynie,
(*d*) maszyna nie może wykonywać jednocześnie więcej niż jedno zadanie,
(*e*) kolejność wykonywania zadań, na obu maszynach, musi być taka sama.

Dla ustalonej kolejności wykonywania zadań na maszynach, niech S_{ij} będzie momentem rozpoczęcia wykonywania operacji O_{ij} $(i \in \mathcal{J},\ j = 1, 2)$. Z ograniczeń (b) i (c) wynika, że $C_{ij} = S_{ij} + p_{ij}$ jest momentem zakończenia operacji O_{ij}. Momenty te można wyznaczyć z następujących zależności rekurencyjnych:

$$C_{\pi(i),j} = \max\{C_{\pi(i-1),j},\ C_{\pi(i),j-1}\} + p_{\pi(i),j}, \quad i = 1, 2,\ \ldots .n, \quad j = 1, 2, \tag{1}$$

z warunkami początkowymi:

$$C_{\pi(0),j} = 0, \quad j = 1, 2 \text{ oraz } C_{\pi(i),0} = 0, \quad i = 1, 2,\ \ldots, n. \tag{2}$$

Przez $\mathcal{C}_i = C_{i2}$ oznaczamy termin zakończenia zadania i, tj. operacji O_{i2}. Wówczas

$$T_i = \max\{0, \mathcal{C}_i - d_i\} \tag{3}$$

jest *spóźnieniem* wykonania zadania i, $f_i = w_i \cdot T_i$ *karą* za spóźnienie (inaczej - *kosztem wykonania zadania*). Jeżeli $T_i = 0$, to zadanie nazywamy *terminowym*, a w przeciwnym przypadku - *spóźnionym*.

Każde rozwiązanie, tj. kolejność wykonywania zadań (taka sama na obu maszynach) może być reprezentowana przez permutację zadań ze zbioru $\mathcal{J}$. Niech Π będzie zbiorem wszystkich takich permutacji. Dla dowolnej permutacji

$$\pi = (\pi(1), \ldots, \pi(n))$$

kara za spóźnienia wykonania zadań (w skrócie *koszt rozwiązania*) jest określona jako

$$F(\pi) = \sum_{i=1}^{n} w_{\pi(i)} \cdot T_{\pi(i)}. \tag{4}$$

W rozpatrywanym w pracy problemie należy wyznaczyć kolejność wykonywania zadań minimalizującą karę za spóźnienia zadań, tj. permutację optymalną $\pi^* \in \Pi$, dla której

$$F(\pi^*) = \min\{F(\pi) : \ \pi \in \Pi\}. \tag{5}$$

Dwumaszynowy problem przepływowy z kryterium $C_{\max}$ należy do klasy $\mathcal{P}$. Do jego rozwiązywania jest stosowany algorytm Johnsona [4] o złożoności $O(n \log n)$. W dalszej części pracy będziemy stosowali ten algorytm do wyznaczania optymalnej, ze względu na kryterium $C_{\max}$, kolejności pewnych zadań. Tak wyznaczone podsekwencje zadań będą posiadały cechę klasycznych bloków – każda zmiana kolejności wewnątrz nie poprawi wartości funkcji kryterialnej.

3. Własności problemu

Do rozwiązywania rozpatrywanego, *NP*-trudnego problemu szeregowania będziemy stosowali algorytm bazujący na metodzie lokalnych poszukiwań. Zasadniczym elementem tej metody, mającym decydujący wpływ na czas obliczeń oraz wartości wyznaczanych rozwiązań, jest procedura generowania oraz przeszukiwania otoczenia.

W najlepszych algorytmach metaheurystycznych rozwiązywania problemu przepływowego z minimalizacja terminu zakończenia wykonywania zadań (tj. $F||C_{\max}$) są stosowane otoczenia generowane przez ruchy typu wstaw (ang. *insert*), w skrócie zwane *i-ruchami*. Jeżeli $\pi \in \Pi$, to *i-ruch* i_l^k $(1 \leqslant k, l \leqslant n)$ generuje z π nową permutację $i_l^k(\pi) = \pi_l^k$ przez przestawienie elementu $\pi(k)$ na pozycję l. Otoczenie generowane przez te ruchy ma $n(n-1)$ elementów. Ruchy te oraz generowane przez nie otoczenia są opisane w pracy Bożejko i in. [1].

W wielu algorytmach metaheurystycznych rozwiązywania problemu $F||C_{\max}$ są stosowane tzw. „własności eliminacyjne bloków". Dzięki nim, w procedurze przeszukiwania otoczenia, można pominąć wiele gorszych rozwiązań. Jak wykazały przeprowadzone eksperymenty obliczeniowe, dzięki temu nie tylko skraca się czas przeszukiwania otoczenia, ale także uzyskuje się lepsze rozwiązania. W dalszej części tego rozdziału udowodnimy podobne własności pozwalające na eliminację, poprze przegląd pośredni, wiele rozwiązań. Osłabiając lub pomijając pewne ograniczenia w definicji klasycznych bloków, wprowadzamy nowe pojęcie „słabszego" bloku, tzw. *semi-bloku*.

Ciąg bezpośrednio po sobie występujących elementów w ciągu wartości permutacji π będziemy nazywali *subpermutacją*. Jeżeli

$$\eta = (\pi(u), \pi(u+1), \ldots, \pi(v)), \ 1 \leqslant u \leqslant v \leqslant n,$$

jest subpermutacją permutacji π, to koszt wykonywania zadań z η wynosi

$$F_\pi(\eta) = \sum_{i=u}^{v} (w_{\eta(i)}(C_{\eta(i)} - d_{\eta(i)})), \tag{6}$$

gdzie $C_{\eta(i)}$ jest *terminem zakończenia* wykonywania zadania $\eta(i)$ w permutacji π. Niech

$$\alpha = (1, 2, \ldots, a-1),\ \beta = (a, a+1, \ldots, b-1, b),\ \gamma = b+1, \ldots, n), \tag{7}$$

gdzie $1 < a < b \leqslant n$ będą pewnymi subpermutacjami w π, tj.

$$\pi = (1, 2, \ldots, a-1, a, a+1, \ldots, b-1, b, b+1, \ldots, n). \tag{8}$$

Wówczas permutacja $\pi = (\alpha, \beta, \gamma)$ jest ciągiem (konkatenacją) trzech subpermutacji, a jej koszt wynosi

$$F(\pi) = F_\pi(\alpha) + F_\pi(\beta) + F_\pi(\gamma). \tag{9}$$

3.1. Bloki zadań terminowych

Niech permutacja $\pi \in \Pi$ będzie ciągiem trzech subpermutacji, tj. $\pi = (\alpha, \beta, \gamma)$. Do zbioru zadań z subpermutacji β stosujemy algorytm Johnsona (patrz rozdział 2.). W ten sposób wyznaczamy nową kolejność zadań

$$\beta' = (a', a'+1, \ldots, b'-1, b'). \tag{10}$$

Subpermutację β' będziemy nazywać *Johnson optymalną*, w skrócie *J-opt*. Jest to bowiem optymalna kolejność ze względu na minimalizację terminu zakończenia wykonania wszystkich zadań z β.

Niech permutacja

$$\pi' = (\alpha, \beta', \gamma), \tag{11}$$

gdzie β' jest subpermutacją *J-opt*.

Twierdzenie 1. Jeżeli permutacja $\delta = (\alpha, \beta", \gamma)$ została wygenerowana z π' (11) przez zmianę kolejności zadań w *J-opt* subpermutacji β', to moment zakończenia dowolnego zadania z γ (w permutacji δ) nie jest mniejszy niż moment zakończenia tego zadania w permutacji π'.

Dowód. Dowód wynika bezpośrednio z faktu, że β' *J-opt* subpermutacją, a więc dowolna zmiana kolejności jej elementu nie zmniejszy momentu zakończenia wykonywania ostatniego zadania $C_{b"}$.

Definicja 1. Niech permutacja $\pi' = (\alpha, \beta', \gamma)$ oraz β' będzie subpermutacją *J-opt*. Jeżeli wszystkie zadania w β' są terminowe, to β' nazywamy *blokiem zadań terminowych* (w skrócie *T-blokiem*).

Twierdzenie 2. (Eliminacyjna własność *T-bloku*) Jeżeli permutacja δ została wygenerowana z $\pi' \in \Pi$ przez zmianę kolejności zadań w pewnym *T-bloku*, to $F(\delta) \geqslant F(\pi')$.

Dowód jest oparty na idei zamieszczonej w pracy [1].

Wniosek 1. Generując z π nowe permutacje można pominąć te z nich, które powstały prze zamianę kolejności zadań w dowolnym *T-bloku*. Nie dają one bowiem poprawy wartości funkcji celu (4).

Niech $\pi = (\alpha, \beta, \gamma)$, $\pi \in \Pi$ oraz $\pi' = (\alpha, \beta', \gamma)$ gdzie β' jest supermutacją *J-opt*. Załóżmy, że nie wszystkie zadania w β' są terminowe. Wobec tego β' nie jest *T-blokiem*. Rozpatrujemy następujące warunki:

(A) $(F(\pi') - F(\pi))/F(\pi') < \varepsilon$;
(B) $F(\beta') < \lambda$
(C) $(C_b - C_{b'})/C_b < \varrho$,

gdzie $\varepsilon, \lambda, \varrho$ są pewnymi parametrami (nieujemnymi liczbami rzeczywistymi).
Definicja 2. Jeżeli *J-opt* subpermutacja β' nie jest *T-blokiem*, a jednocześnie spełnia jeden z warunków (A) lub (B) lub (C), to nazywamy ją *semi T-blokiem* (w skrócie *sT-blokiem*).

Niestety, *sT-blok* nie ma własności *T-bloku* zawartej we Wniosku 1. Zamiana kolejności elementów w *sT-bloku* może prowadzić do poprawy wartości funkcji celu. Jednak wielkość ewentualnej poprawy możemy kontrolować poprzez odpowiednie dobranie parametrów. Dzięki temu możemy ewentualnie odrzucić jedynie niewiele lepsze rozwiązania.

Generując otoczenia w algorytmach lokalnych popraw, do eliminowania pewnych rozwiązań, będziemy zasadniczo stosowali *T-bloki*. Jeżeli okaże się, że liczba zadań w *T-blokach* jest niewielka wówczas skorzystamy z *sT-bloków*. Eksperymentalnie ustalimy, którą z ograniczeń (A), (B) lub (C) zastosować.

3.2. Bloki zadań spóźnionych

Niech permutacja zadań $\pi = (1, 2, \ldots, a-1, a, a+1, \ldots, b-1, b, b+1, \ldots, n) = (\alpha, \beta, \gamma)$, gdzie $\alpha = (1, 2, \ldots, a-1)$, $\beta = (a, a+1, \ldots, b-1, b)$, $\gamma = b+1, \ldots, n)$. Zakładamy, że w subpermutacji β wszystkie zadania są spóźnione, a ponadto

$$\forall i \in \beta,\ d_i < C_{a-1} + p_{i,2}. \tag{12}$$

Jeżeli a=1, to przyjmujemy $C_{a-1} = 0$.

Z warunku (12) wynika, że dowolne zadanie z β wstawione na pierwszą pozycję, w β tj. na pozycję a, jest spóźnione. Załóżmy, że w subpermutacji β wszystkie zadania są spóźnione, tj. spełnione są ograniczenia (12). Z β generujemy dwie nowe subpermutacje:

(a) *J-opt* subpermutację β', tj. ustalamy kolejność elementów stosując algorytm Johnsona
(b) subpermutację β" ustawiając zadania według nierosnących wartości ilorazów $w_i/(p_{i,1} + p_{i,2})$.

Uwaga 1. W problemie jednomaszynowym z kryterium $\sum w_i T_i$ subpermutacja β" (punkt (b)) jest optymalna ze względu na sumę kosztów spóźnień (Smith [12]).
Definicja 3. Subpermutację β" nazywamy *semi D-blokiem* (w skrócie *sD-blokiem*), jeżeli

$$(C_{b"} - C_{b'})/C_{b"} \leqslant \varphi,$$

gdzie φ jest parametrem, a $C_{b"}$ i $C_{b'}$ są terminami zakończenia wykonywania ostatniego zadania odpowiednio w β' oraz β".

Definiując otoczenie będziemy pomijali rozwiązania generowane przez zmianę kolejności zadań w dowolnym *sD-bloku*.
Twierdzenie 3. Dowolną permutację $\pi \in \Pi$ można przedstawić w postaci ciągu $\pi = [B^1, B^2, \ldots, B^t]$, gdzie B^i $(i = 1, 2, \ldots, t)$ jest blokiem lub semi blokiem.
W dalszej części pracy będziemy rozpatrywać bloki oraz semi bloki, które:

(i) maja co najmniej 4 elementy,

(ii) są maksymalne ze względu na zawieranie, tj. nie można dodać na początek lub na koniec elementu tak, aby spełniona była definicja bloku lub semi-bloku.

4. Algorytm poszukiwania z zabronieniami

Algorytm poszukiwania z zabronieniami (w skrócie TS - *tabu search*) jest metodą lokalnych poszukiwań zaproponowaną przez Glovera [2]. W celu dywersyfikacji procesu poszukiwań wprowadzany jest mechanizm skoków powrotnych (*ang. backtrack jump*), który polega na wznawianiu procesu poszukiwań od zapamiętanych obiecujących rozwiązań. Mechanizm ten jest realizowany poprzez wprowadzenie tzw. pamięci długoterminowej (*ang. long term memory*) LT. Ma ona postać listy o ustalonej długości na której pamiętane są pary (y, T). Jeśli $F(y) < F(x^*)$ to do listy LT dodawana jest najlepszy element z otoczenia $\mathcal{N}(x)$ oraz lista zabronień T z dodanymi atrybutami rozwiązania y. Skok powrotny wykonywany jest w przypadku, gdy w procesie poszukiwań wystąpi cykl o ustalonej wcześniej długości. W trakcie wykonywania kolejnych iteracji algorytmu pamiętane jest najlepsze znalezione rozwiązanie x^*. Algorytm kończy działanie, gdy spełniony zostanie warunek stopu (np. ustalony czas obliczeń lub liczba iteracji). W zaimplementowanym algorytmie wykorzystano *T*- oraz *D-bloki* z uwzględnieniem *Johnson optymalności*.

5. Eksperymenty obliczeniowe

Na potrzeby przeprowadzenia eksperymentów obliczeniowych wygenerowanych zostało dziewięć różnych zbiorów instancji testowych. Każdy ze zbiorów zawiera 70 instancji testowych. Czasy wykonywania zadań na poszczególnych maszynach zostały wylosowane z zakresu od 1 do 99. Wartości żądanych czasów zakończenia zadań wygenerowane zostały na podstawie *współczynnika spóźnień* T oraz *zakresu terminowości* R z rozkładem jednostajnym z zakresy od $P(1 - T - R/2)$ do $P(1 - T + r/2)$ z rozkładem jednostajnym wg. mechanizmu opisanego w [9]. Parametr P to wartość dolnego ograniczenia dla kryterium C_{max} zaproponowanego w [13]. Instancje testowe zostały wygenerowane dla wartości parametrów $T = \{0.2, 0.4, 0.6\}$ oraz $R = \{0.2, 0.6, 1.0\}$ co daje dziewięć kombinacji. W niektórych przypadkach, szczególnie dla małych wartości T oraz dużych wartości R wartości żądanych czasów zakończenia zadań mogą być ujemne. W takich sytuacjach wartości ujemne zostały zastąpione zerami. Liczba zadań dla których wygenerowane zostały instancje testowe to $n = \{10, 20, 50, 100, 200, 500, 1000\}$. Dla każdej wartości n wygenerowanych zostało 10 instancji co w sumie dało 70 instancji dla każdej kombinacji T i R. W sumie wygenerowano 630 instancji.

Algorytm poszukiwania z zabronieniami dla dwumaszynowego problemy przepływowego z ważoną sumą spóźnień został zaimplementowany w języku C++. Eksperymenty obliczeniowe zostały przeprowadzone na klastrze Bem we Wrocławskim Centrum Sieciowo–Superkomputerowym pracującym pod kontrolą 64–bitowego systemu operacyjnego Scientific Linux 6.7 (Carbon) wyposażonym w procesory Intel Xeon E5-2670 (2.30GHz).

W tabeli 1 zostały przedstawione wyniki eksperymentów obliczeniowych w postaci wartości procentowego błędu względnego (PRD) względem rozwiązań referencyj-

nych. Wartości PRD zostały wyznaczone z zależności

$$PRD = \frac{F_{ref} - F_{alg}}{F_{ref}} \cdot 100\%, \tag{13}$$

gdzie F_{ref} jest wartością funkcji celu dla rozwiązania referencyjnego, a F_{alg} jest wartością funkcji celu dla rozwiązania uzyskanego przez badany algorytm. Jako rozwiązania referencyjne zostały wykorzystane wyniki uzyskane za pomocą algorytmu poszukiwania z zabronieniami bez własności blokowych (TS). Poszczególne kolumny w tabeli 1 oznaczają:

- TS_B – algorytm poszukiwań z zabronieniami wykorzystujący pamięć długoterminową oraz własności blokowe - T-bloki i D-bloki.
- TS_{BJ} – algorytm poszukiwań z zabronieniami wykorzystujący pamięć długoterminową oraz własności blokowe - T-bloki *J-opt* oraz D-bloki.

Tabela 1

Wartości PRD dla instancji testowych dla różnych parametrów T i R.

parametry	n	TS_B	TS_{BJ}
$T = 0,2$, $R = 0,2$	10	-2,40	5,22
$T = 0,2$, $R = 0,6$	20	-10,26	-62,58
$T = 0,2$, $R = 1,0$	50	-10,11	-74,45
$T = 0,4$, $R = 0,2$	100	0,15	-5,52
$T = 0,4$, $R = 0,6$	200	-4,18	-33,92
$T = 0,4$, $R = 1,0$	500	6,05	-77,88
$T = 0,6$, $R = 0,2$	100	-0,06	-7,20
$T = 0,6$, $R = 0,6$	200	-1,14	-24,22
$T = 0,6$, $R = 1,0$	500	-0,14	-43,16
Średnia		-2,45	-35,97

Wszystkie testowane algorytmy uruchamiane były na 60 sekund każdy. Jak można zauważyć analizując Tabelę 1, najlepsze wyniki zostały uzyskane przez algorytm przeszukiwania z zabronieniami w którym wykorzystano mechanizm bloków wraz z subpermutacjami *Johnson optymalnymi* uzyskując bardzo znaczącą poprawę (aż o 35,97%) względem algorytmu bez mechanizmu bloków działającego przez ten sam czas.

6. Podsumowanie

W pracy jest rozpatrywany dwumaszynowy problem przepływowy z minimalizacją sumy kosztów spóźnień. Przedstawiamy opis rozpatrywanego problemu $F2||\sum w_i T_i$ oraz jego model matematyczny. Dowodzimy specyficznych własności (tzw. własności eliminacyjnych bloków), które znacznie poprawiają efektywność algorytmów rozwiązywania wielomaszynowych problemów sumokosztowych. W przypadku ich zastosowania w konstrukcjach algorytmów opartych na metodzie lokalnych poszukiwań pozwalają na znaczne zmniejszenie rozmiaru otoczenia generowanego w każdej iteracji algorytmu.

Praca powstała w wyniku realizacji projektu badawczego o nr DEC 2017/25/B/ST7 /02181 finansowanego ze środków Narodowego Centrum Nauki oraz w ramach grantu nr 96 we Wrocławskim Centrum Sieciowo-Superkomputerowym.

LITERATURA

1. Bożejko W., Grabowski J., Wodecki M.: Block approach tabu search algorithm for single machine total weighted tardiness problem. Computers & Industrial Engineering, 50(1-2), 2006. p. 1–14.
2. Glover F.: Tabu Search - Part I, ORSA Journal on Computing, 1(3), 1989, p. 190–206.
3. Graham, R. L., Lawler, E. L., Lenstra, J. K., Rinnooy-Kan, A. H. G.: Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics, 5, 1979, p. 287—326.
4. Johnson S.M.: Optimal two- and three-stage production schedules with setup times included. Naval Res. Logist. Quart. I, 1954, p. 61–68.
5. Khalili, M., Tavakkoli-Moghaddam, R.: A multi-objective electromagnetism algorithm for a bi-objective flowshop scheduling problem. Journal of Manufacturing Systems, 31(2), 2012, p. 232-–239.
6. Koulamus C.: The total tardiness problem: review and extensions. Operations Research 42, 1994, p. 1025—1041.
7. Lee J.-Y., Kim Y.-D.: Minimizing total tardiness in a two-machine flowshop scheduling problem with availability constraint on the first machine. Computers & Industrial Engineering, 114, 2017, p. 22–30.
8. Liao, C. J., Liao, L. M., Tseng, C. T.: A performance evaluation of permutation vs. non-permutation schedules in a flowshop. International Journal of Production Research, 44(20), 2006, p. 4297-–4309.
9. Potts C.N., Van Wassenhove L.N.: A decomposition algorithm for the single machine total tardiness problem. Operations Research Letters, 1, 1982, p. 177–181.
10. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research, 177(3), 2007, p. 2033-–2049.
11. Schaller J.: Note on minimizing total tardiness in a two-machine flowshop. Computers & Operations Research, 32(12), 2005, p. 3273–3281.
12. Smith W.E.: Various Optimizers for Single-Stage Production. Naval Research Logistics Quarterly, 3(1-2), 1956, p. 59–66.
13. Taillard E.:Benchmarks for basic scheduling problems. European Journal of Operational Research, 64, 1993, p. 278–285.
14. Uchroński M.: Benchmark files for 2FP$\|\sum$wiTi. 2018, http://mariusz.uchronski.staff.iiar.pwr.edu.pl/benchmarks/2fpTestData.zip