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

Adam JANIAK, Marcin MAREK
Politechnika Wrocławska

PORÓWNANIE TECHNIK APROKSYMACYJNYCH: TABU SEARCH, EWOLUCYJNEJ I SIECI NEURONOWEJ NA PRZYKŁADZIE PROBLEMU SZEREGOWANIA ZADAŃ Z ZASOBAMI NA POJEDYNCZEJ MASZYNIE

Streszczenie. W pracy porównano trzy algorytmy metaheurystyczne: tabu search, genetyczny, sieć neuronową dla problemu szeregowania zadań na jednej maszynie z zadanymi terminami dostępności i czasami realizacji zależnymi od ilości przydzielonego zasobu. Przyjętym kryterium jest minimalizacja maksymalnej nieterminowości. Podano wyniki przeprowadzonych eksperymentów numerycznych.

THE COMPARISON OF THE APPROXIMATION ALGORITHMS: TABU SEARCH, GENETIC, NEURAL NETWORK FOR THE SCHEDULING PROBLEM WITH RESOURCES ON SINGLE MACHINE

Summary. The paper deals with a single machine scheduling problem with given release dates and processing times dependent on resources. Considered criterion is the maximum lateness minimization. To solve the problem three metaheuristic algorithms are presented and compared.

1. Wstęp

W wielu dyskretnych lub dyskretno-ciągłych procesach produkcyjnych mamy do czynienia z sytuacją, w której z technologicznego punktu widzenia tylko jedna maszyna jest krytyczna, a wszystkie pozostałe maszyny występujące w ciągu produkcyjnym, przed i po tej maszynie, można traktować jako maszyny o nieograniczonej przepustowości, gdyż nie tworzą one wąskiego gardła w procesie technologicznym. Niniejsza praca dotyczy jednomaszynowego problemu harmonogramowania zadań z zadanymi terminami gotowości i pożądanymi terminami zakończenia wykonania. Czasy wykonywania zadań zależą od ilości przydzielonego zasobu.

W rozdziale 2 pracy dokonano sformułowania problemu. Do rozwiązania postawionego problemu w rozdziale 3, 4 i 5 zaproponowano odpowiednio algorytm tabu search, algorytm ewolucyjny oraz sieć neuronową typu Hopfielda. W rozdziale 6 opisano sposób, w jaki

przeprowadzono eksperyment i zaprezentowano jego wyniki, a następnie w rozdziale 7 dokonano analizy uzyskanych rezultatów.

2. Sformułowanie problemu

Dany jest zbiór $J=\{J_1,...,J_n\}$ n niezależnych i niepodzielnych zadań, które mają być wykonane na pojedynczej maszynie. W każdej chwili maszyna może wykonywać co najwyżej jedno z nich. Dla każdego zadania J_j dany jest termin jego dostępności r_j, pożądany termin zakończenia jego wykonania d_j oraz czas wykonywania p_j, dany w postaci liniowej funkcji zależnej od ilości przydzielonego zasobu u_j:

$$p_j=b_j-a_ju_j,$$

gdzie $a_j\geq 0$, $b_j>0$, przy czym $\alpha_j\leq u_j\leq\beta_j$, gdzie α_j oraz $\beta_j\leq b_j/a_j$ są zadanymi ograniczeniami technologicznymi na ilość zasobu przydzielonego do wykonania zadania j. Zadana jest globalna ilość całkowitego zasobu $\hat{R}$ dysponowanego do wykonania wszystkich zadań: $\sum_{j\in J} u_j \leq \hat{R}$.

Niech Π oznacza zbiór wszystkich możliwych uszeregowań zadań wykonywanych na pojedynczej maszynie, a U niech będzie zbiorem wszystkich możliwych rozdziałów zasobów do zadań. Niech $\pi\in\Pi$ oznacza permutację zadań wykonywanych na maszynie, a $u\in U$ przydział zasobów do zadań.

Należy znaleźć takie uszeregowanie zadań π^* i taki rozdział zasobu do zadań u^*, które zminimalizują kryterium maksymalnej nieterminowości $\min L_{\max}(\pi,u)=\min\{\max_{1\leq j\leq n} L_j(\pi,u)\}=\min\{\max_{1\leq j\leq n}(C_j(\pi,u)-d_j)\}$, gdzie C_j jest terminem zakończenia wykonywania zadania j.

W [8] wykazano, że rozpatrywany tutaj problem jest *silnie NP-trudny*.

Przykład zastosowania problemu można znaleźć w hutnictwie, gdzie czas nagrzewania wlewków, które następnie są walcowane przez walcarkę-zgniatacz, zależy od ilości zużytego gazu. Innym przykładem zastosowania jest informatyka, gdzie czas wykonania programu zależy od ilości dostępnej pamięci.

3. Zastosowanie algorytmu typu tabu do rozwiązania badanego problemu

Technika poszukiwania z zabronionymi ruchami została zaprezentowana po raz pierwszy w pracach [3], [4]. Metoda ta używa tzw. pamięci obliczeń do sterowania bieżącym procesem poszukiwania rozwiązania. Wymaga ona określenia szeregu elementów składowych takich, jak: ruch, sąsiedztwo, funkcja oceny rozwiązań, funkcja aspiracji, pamięci obliczeń. Metoda ta

nie doczekała się jeszcze pełnego uporządkowania swoich podstaw teoretycznych i ciągle się rozwija.

W celu poprawnego zdefiniowania elementów składowych techniki tabu search rozpatrzmy zadanie optymalizacji dyskretnej, polegające na minimalizacji wartości funkcji celu, tzn. $\min\{f(\pi) : \pi \in \Pi\}$, gdzie Π jest zbiorem dyskretnych rozwiązań dopuszczalnych, a f jest funkcją celu odwzorowującą Π w zbiór liczb całkowitych.

Ruchem v określamy funkcję transformującą elementy zbioru $\Pi(v)$ w Π, gdzie $\Pi(v) \subset \Pi$. Należy zatem rozumieć, że $\Pi(v)$ jest zbiorem tych rozwiązań dopuszczalnych, w stosunku do których możliwe jest zastosowanie ruchu v i ruch ten nie spowoduje wyprowadzenia nowo otrzymanego rozwiązania poza zbiór rozwiązań dopuszczalnych. Zazwyczaj zachodzi równość $\Pi(v) = \Pi$.

Otoczenie funkcyjne O(π) rozwiązania π jest zbiorem wszystkich tych ruchów, które zastosowane w stosunku do rozwiązania π nie spowodują wyprowadzenia nowo otrzymanych rozwiązań poza zbiór rozwiązań dopuszczalnych $O(\pi) = \{v \in V : \pi \in \Pi(v)\}$, gdzie V jest zbiorem wszystkich możliwych ruchów.

Otoczenie (sąsiedztwo) O(π) rozwiązania π jest zbiorem rozwiązań, które powstały w wyniku wykonania na rozwiązaniu $\square$ ruchów należących do otoczenia funkcyjnego $O(\pi)$ rozwiązania π, tzn. $S(\pi) = \{v(\pi) : v \in O(\pi)\}$.

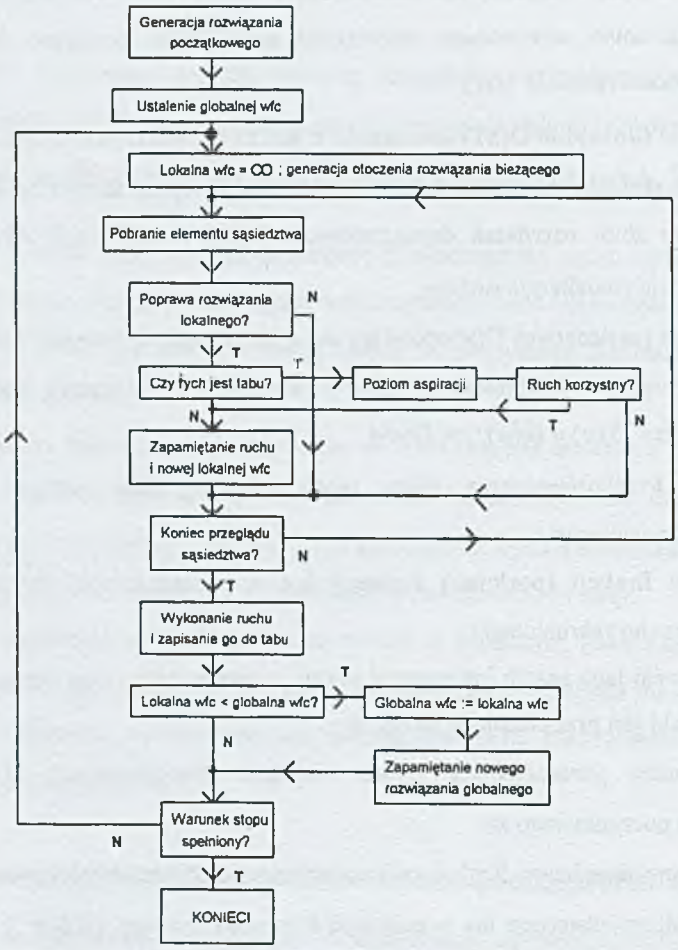
Pamięć krótkoterminowa (lista tabu) określa zbiór ruchów zabronionych, niemożliwych do wykonania.

Zadaniem **funkcji (poziomu) aspiracji** jest w uzasadnionych przypadkach zdjęcie statusu tabu z ruchu zabronionego.

Idea techniki tabu search jest prosta i można ją ująć w kilku w następujących punktach (schemat techniki jest przedstawiony na rys. 1).

1. Rozpoczynamy przeszukiwanie zbioru rozwiązań dopuszczalnych Π od pewnego rozwiązania początkowego π_p.
2. Przeszukujemy sąsiedztwo $S(\pi_p)$ w celu znalezienia nie zabronionego wpisem na listę tabu ruchu v i odpowiadającego mu rozwiązania lokalnego $\pi = v(\pi_p) \in S(\pi_p)$ o najmniejszej (lokalnej) wartości funkcji celu.

3. Wykonujemy ruch v niezależnie od relacji między wartościami funkcji celu $f(\pi_p)$ i $f(\pi)$ (dopuszczając pogorszenie wartości funkcji celu nowego rozwiązania unikamy pułapki lokalnego optimum).
4. Rozwiązanie π jest przyjmowane jako początkowe dla kolejnego kroku iteracyjnego, a ruch v wpisywany jest na listę tabu i posiada status tabu przez określoną liczbę iteracji (zapobiega to powstawaniu nieefektywnych iteracji pracy algorytmu).
5. Przeszukiwanie zbioru rozwiązań zostaje przerwane określonym warunkiem stopu i rozwiązanie odpowiadające globalnej wfc (rys. 1) jest poszukiwanym rozwiązaniem.



Rys.1. Ogólny schemat algorytmu typu tabu search (wfc oznacza wartość funkcji celu)
Fig.1. Schematic diagram for tabu search algorithm

Omówiony algorytm zaimplementowano do rozpatrywanego problemu w sposób następujący. Na początku dokonuje on wstępnego przydziału zasobów do zadań, przydzielając maksymalną, dopuszczalną ilość zasobów począwszy od zadań o największym współczynniku kierunkowym modelu a_i, aż do wyczerpania całkowitej ilości zasobów.

W ten sposób uzyskujemy ustalone czasy wykonywania zadań p_i. Powyższego rozdziału nie zmieniamy w trakcie działania algorytmu. Dopiero po uzyskaniu końcowej permutacji dokonujemy wtórnego, tym razem optymalnego, rozdziału zasobów za pomocą wspomnianego już zmodyfikowanego algorytmu Hamachera i Tufekciego [5].

Niech $v = (i, j)$ będzie parą pozycji w permutacji

$$\pi \triangleq (\pi(1), \pi(2), ..., \pi(i)\pi(i+1), ..., \pi(j-1)\pi(j), ..., \pi(n)), i < j .$$

Dla permutacji π para $v = (i, j)$ definiuje ruch polegający na odwróceniu kolejności wykonywania zadań od pozycji i do pozycji j. Zatem ruch v generuje permutację π_v na podstawie π w następujący sposób: $\pi_v = (\pi(1), ..., \pi(y), \pi(y-1), ..., \pi(x+1), \pi(x), ..., \pi(n))$. W naszej implementacji $i := 1, 2, ..., n-1$, a $j := i+1, ..., n$.

Lista tabu (pamięć krótkoterminowa) definiowana jest jako skończony zbiór T ruchów. Jeżeli ruch $v = (i,j)$ jest wykonany na permutacji π, to dodajemy go do T (początkowo T jest puste). Jeżeli liczność zbioru $|T| = dlt$, gdzie dlt jest długością listy tabu, to przed dodaniem nowego ruchu do T najstarszy musi być usunięty z listy. Ruch v ma status tabu dopóki znajduje się na liście tabu.

Z otoczenia funkcyjnego O rozwiązania π wybierany jest ruch v generujący rozwiązanie $\pi_{v(O)}$ o najniższej wartości funkcji celu $f(\pi_{v(O)}) = \min_{v \in O} f(\pi_v)$. Ruch $v(O)$ musi być: nie zabroniony lub zabroniony i korzystny. Ruch $v(O)$ jest zabroniony i korzystny, jeżeli ma status tabu i spełniony jest warunek $f(\pi_{v(O)}) < A$, gdzie A jest poziomem aspiracji, zdefiniowanym tutaj jako aktualnie najmniejsza wartość funkcji celu otrzymana w procesie poszukiwań.

W naszym algorytmie, prócz wspomnianej już wcześniej pamięci krótkoterminowej, wykorzystuje się również pamięć długoterminową PDT o długości $dldt$ z elementami w postaci zbiorów (π, T), gdzie π jest permutacją, zaś T jest listą tabu dla π. W każdej iteracji algorytmu, w której aktualnie najlepsze rozwiązanie zostaje poprawione, permutacja π poprzedzająca najlepsze rozwiązanie, łącznie ze związaną z nią listą T, dodawane są do PDT. Jeżeli PDT jest pełna, to przed dodaniem nowego elementu najstarszy element jest z niej usuwany. Przy pobieraniu elementu z pamięci PDT bierzemy zawsze element najmłodszy.

Przy założeniu, że została wykonana *maxiter* liczba kolejnych iteracji bez poprawy globalnie najlepszej wartości funkcji celu w procesie poszukiwań cofamy się do ostatniej permutacji π, po której nastąpiła poprawa. Oznacza to, że proces poszukiwań rozpoczyna się od najmłodszego elementu zbioru *PDT*, który jest z niej następnie usuwany. Algorytm kończy pracę, gdy pamięć długoterminowa *PDT* jest pusta.

Zaproponowany algorytm przedstawia się następująco:

Krok 0. Podstaw $\pi := \pi^*$ {π^* jest losowo wybraną permutacją początkową},
$f^* = f(\pi)$, $T := \emptyset$, $PDT := \emptyset$, $iter := 0$.

Krok 1. Podstaw $iter := iter + 1$ oraz generuj otoczenie permutacji π.

Krok 2. Znajdź w otoczeniu π najlepszy ruch v nie zabroniony lub zabroniony korzystny (dający rozwiązanie π_v).

Krok 3. Usuń najstarszy element z listy T oraz dodaj do T ruch v.

Krok 4. Jeżeli $f(\pi_v) < f^*$, wtedy podstaw $\pi^* := \pi_v$, $f^* = f(\pi^*)$, $iter := 0$ oraz usuń najstarszy element z *PDT* i dodaj do *PDT* parę (π, T).

Krok 5. Podstaw $\pi := \pi_v$.

Krok 6. Jeżeli $iter \le maxiter$, to idź do kroku 1.

Krok 7. Jeżeli $PDT \neq \emptyset$, to pobierz najmłodszy element (π, T) z *PDT* ($PDT := PDT \setminus \{\pi, T\}$), podstaw $iter := 0$ i wróć do kroku 1.
W przeciwnym razie STOP - π^* jest poszukiwanym rozwiązaniem.

Podczas eksperymentu obliczeniowego przyjęto następujące wartości parametrów sterujących: $dlt = 10$, $maxiter = 20$, $dldt = 5$.

Na złożoność obliczeniową jednego kroku iteracyjnego tego algorytmu składa się:

- złożoność obliczeniowa generacji otoczenia, która w tym przypadku wynosi $O\left(\frac{1}{2}n(n-2)\right)$ [2],
- złożoność obliczeniowa wyznaczenia wartości funkcji celu dla każdego elementu otoczenia, która w tym przypadku wynosi O(n).

Zatem złożoność obliczeniowa jednego kroku iteracyjnego algorytmu tabu wynosi $O\left(\frac{1}{2}n^3\right)$.

4. Algorytm genetyczny

Zastosowanie algorytmów genetycznych do przeszukiwania dyskretnego zbioru rozwiązań umożliwia prowadzenie dynamicznych i wielokierunkowych poszukiwań z

jednoczesnym unikaniem pułapek lokalnego ekstremum. Aby zminimalizować wartość funkcji celu na zadanym zbiorze dopuszczalnych rozwiązań, tworzona jest pewna początkowa populacja rozwiązań, która poddawana jest później następującym prawom ewolucji: krzyżowaniu, mutowaniu oraz selekcji rozwiązań do następnej generacji. Ogólny opis i schemat algorytmu genetycznego został przedstawiony w [1], [10]. Omówienie szczegółowe poszczególnych faz zaimplementowanego algorytmu genetycznego do rozwiązania rozpatrywanego problemu można znaleźć w pracy [9].

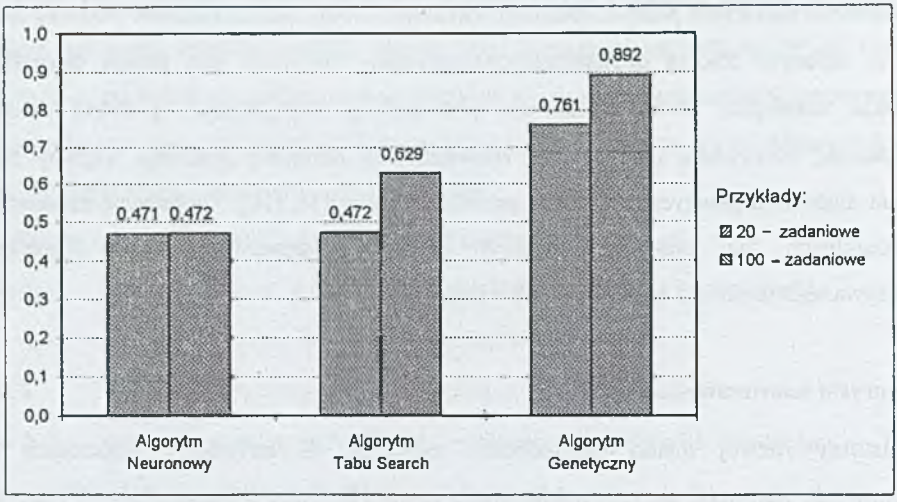
5. Algorytm neuronowy

Istotny rozwój metod optymalizacji, opartych na technikach sztucznych sieci neuronowych, rozpoczął się od opublikowania przez Hopfielda i Tanka ([6], [7]) artykułu, gdzie zaproponowano sieć neuronową nowego typu (w literaturze określa się ją mianem sieci Hopfielda) i zastosowano ją do rozwiązywania silnie NP-trudnego problemu komiwojażera.

Aby zastosować sieć Hopfielda do rozwiązania zadania minimalizacji pewnej funkcji celu, należy dokonać odpowiedniej transformacji tego zadania (ze względu na ograniczenia objętościowe niniejszej pracy nie będziemy omawiali tego zagadnienia). Idee zastosowania sieci neuronowych do rozwiązywania problemów szeregowania stosowano w pracach [11], [12], gdzie analizowano odpowiednio problem gniazdowy z kryterium będącym sumą czasów zakończenia wykonania zadań oraz jednomaszynowy problem minimalizacji średniego opóźnienia z zerowymi czasami dostępności. Do rozwiązywania zadanego problemu szeregowania zadań zastosowano sieć neuronową, której szczegółowy opis można znaleźć w pracy [9].

6. Eksperyment obliczeniowy

Przy wyborze parametrów przykładów, na których przeprowadzono eksperymenty numeryczne, porównujące zaproponowane tutaj algorytmy metaheurystyczne, korzystano ze schematu przewidującego generację szerokiego spektrum instancji odzwierciedlających praktyczne zastosowania, którego szczegółowy opis można znaleźć w pracy [9].



Rys. 2. Średni wskaźnik redukcji wartości funkcji celu uzyskany przez algorytmy metacheurystyczne

Fig. 2. The average reduction coefficient of the criterion value after working metaheuristic algorithms

Ponieważ minimalna wartość funkcji celu nie była znana, obserwowano, w jakim stopniu badane algorytmy zmniejszały początkową wartość funkcji celu. W tym celu zdefiniowano wskaźnik względnej redukcji $R=f(\pi_2)/f(\pi_1)$, gdzie $f(\pi_1)$ jest średnią wartością funkcji celu z 10^4 losowo wygenerowanych permutacji, natomiast $f(\pi_2)$ jest wartością funkcji celu najlepszej znalezionej permutacji dla poszczególnych badanych algorytmów. Algorytmy testowano na komputerze klasy PC 586 100MHz. Średni wskaźnik względnej redukcji dla poszczególnych algorytmów pokazano na rys. 2. Natomiast średnie czasy obliczeń dla poszczególnych algorytmów zestawiono w tablicy 1.

Tablica 1

Średnie czasy obliczeń dla badanych algorytmów

Liczba zadań	Algorytm Neuronowy	Algorytm Tabu Search	Algorytm Genetyczny
20	2.87 s	0.38 s	1.60 s
100	71.08 s	73.13 s	27.27 s

7. Podsumowanie

Otrzymane wyniki z przeprowadzonej analizy eksperymentalnej pokazują, że techniki metaheurystyczne w istotny sposób poprawiają losowe rozwiązania początkowe.

Najlepsze własności minimalizujące dla badanego problemu wykazuje zaproponowana sieć neuronowa Hopfielda.

Znacznie lepsze wyniki dostarczają zaproponowane algorytmy typu tabu i sieć Hopfielda w porównaniu z algorytmem genetycznym, mimo iż, w przeciwieństwie do algorytmu genetycznego, nie korzystają one z optymalnego rozdziału zasobów w każdym kroku iteracyjnym. Algorytm tabu i sieć neuronowa szeregują zadania bazując na heurystycznym rozdziale zasobu do zadań. Dopiero w końcowej fazie ich pracy następuje optymalny rozdział zasobu dla ostatecznego uszeregowania zadań. Optymalny rozdział zasobu w każdym kroku iteracyjnym, w algorytmie tabu search, zbyt mocno podniósłby jego nakład obliczeniowy. Natomiast w zaproponowanej sieci neuronowej uwzględnianie na bieżąco optymalnego rozdziału zasobu wymagałoby zastosowania bardziej skomplikowanej funkcji energetycznej.

Przeprowadzony eksperyment po raz kolejny pokazał siłę technik metaheurystycznych w rozwiązywaniu bardzo trudnych problemów z algorytmicznego punktu widzenia. Potwierdza to fakt, że praktyczne zastosowanie tych technik jest obecnie jedną z najbardziej skutecznych metod sterowania w złożonych, rzeczywistych procesach produkcyjnych. Trudno jednak określić, która metoda jest najlepsza, ponieważ w zależności od specyfiki rozwiązywanego problemu krytycznym wskaźnikiem jakości algorytmu może być czas uzyskiwania wyniku, jakość wyniku lub stopień komplikacji realizacji algorytmu.

LITERATURA

1. Chudzik K., Janiak A.: Algorytm genetyczny dla jednomaszynowego problemu szeregowania zadań z zasobami. Zeszyty Naukowe Politechniki Śląskiej, s.Automatyka, z.117, Gliwice 1996.
2. Deo N., Kowalik J., Sysło M.: Algorytmy optymalizacji dyskretnej. PWN, Warszawa 1993.
3. Glover F.: Taboo Search. Part I, ORSA Journal of Computing 1, 1989.
4. Glover F.: Taboo Search. Part II, ORSA Journal of Computing 2, 1990.
5. Hamacher H.W., Tufekci S.: Algebraic flows and time-cost tradeoff problems, Annals of Discrete Mathematics, vol. 19, 1984.
6. Hopfield J. J., Tank D. W.: "Neural" Computation of Decision in Optimization Problems, Biological Cybernetics, 1985, vol. 52, pp. 141-152.
7. Hopfield J.J., Tank D. W.: Computing with Neural Circuits: A Model, Science, vol. 233, pp. 625-633, 1986.
8. Janiak A.: Dokładne i przybliżone algorytmy szeregowania zadań i rozdziału zasobów w dyskretnych procesach przemysłowych. s. Monografie, Wydawnictwo Politechniki Wrocławskiej, 1991.
9. Janiak A., Krzyżanowski T., Marek M.: Jednomaszynowy problem sekwencyjny z zasobami – wybrane algorytmy metaheurystyczne. Zeszyty Naukowe AGH, s. Automatyka, tom 3, zeszyt nr 1, Kraków 1999.

10. Michalewicz Z., Genetic Algorithms + Data structures = Evolution Programs, Springer-Verlag, 1992.
11. Sabuncuoglu I., Gurgun B., A neural network model for scheduling problems, European Journal of Operational Research, vol. 93, 1996, 288-299.
12. Zohu D. N., Cherkassky V., Baldwin T. R., Olson D. E., A neural approach to job shop scheduling, IEEE Transactions on Neural networks, vol. 2, 1991, pp. 175-179.

Recenzent: Prof.dr hab.inż. E.Toczyłowski

Abstract

In the paper the effective approximation algorithms for the single machine scheduling problem are presented. The aim is to minimize the maximum lateness with job processing times dependent on resources and given job release dates.

The job processing times are given as a linear functions dependent on resources:

$$p_j(u_j) = b_j - a_j u_j,$$

with restrictions: $a_j \geq 0$, $b_j \geq 0$, $\alpha_j \geq u_j \geq \beta_j$,

where:

a_j – the resource consumption ratio;
b_j – the constant part of job processing time;
p_j – the job processing time;
u_j – the amount of resource assigned to the job;
α_j, β_j - the technological constraints on minimum and maximum amount of resource;
$\hat{R}$ - the total amount of resource assigned to all the jobs.

The problem is NP-hard, so three approximation algorithms (tabu search, genetic algorithm, neural network) are used to solve it. The results of the experimental analysis of the applied methods are presented.