ALTERNATYWNE KODOWANIE W ALGORYTMACH – PROBLEM PRZEPŁYWOWY Z SUMĄ SPÓŹNIEŃ

Mariusz MAKUCHOWSKI

Streszczenie: W pracy rozważa się permutacyjny problem przepływowy z kryterium będącym sumą spóźnień realizacji zadań. Dla tego zagadnienia proponuje się cztery algorytmy popraw. Pierwszy z nich bazuje na metodzie symulowanego wyżarzania, drugi natomiast jest algorytmem genetycznym. Następnie proponuje się zupełnie nową metodę kodowania rozwiązania (permutacji). Dokonuje się modyfikacji przedstawionych algorytmów tworząc dwa kolejne wykorzystujące proponowane kodowanie. Skonstruowane algorytmy poddaje się badaniom numerycznym, na znanych z literatury, przykładach testowych. Na podstawie uzyskanych wyników poszczególnych wersji algorytmów, zostają opisane wnioski dotyczące wpływu wyboru kodowania rozwiązania na ich efektywność.

Słowa kluczowe: problem przepływowy, suma spóźnień, kodowanie permutacji, algorytm symulowanego wyżarzania, algorytm genetyczny.

1. Wprowadzenie

Optymalizacja jest zagadnieniem rozpatrywanym zarówno z praktycznego jak i teoretycznego punktu widzenia. Dysponując skutecznymi metodami optymalizacyjnymi możemy usprawniać wszelkiego rodzaju działania, zaczynając od lepszego planowania dnia codziennego, a kończąc na ulepszaniu harmonogramów w dużych przedsiębiorstwach. Zdecydowanie większa część praktycznych zagadnień optymalizacji dyskretnej, należy do klasy problemów NP-trudnych. Dla problemów tej klasy nie udało się opracować efektywnych czasowo (działających w czasie rosnącym wielomianowo wraz ze wzrostem rozmiaru instancji) algorytmów dokładnych i o ile klasa problemów P jest różna od klasy problemów NP nigdy nie da się ich stworzyć. Z tego względu tworzone są wszelkiego rodzaju algorytmy przybliżone działające w akceptowalnym czasie. Niektóre metody optymalizacji dyskretnej są na tyle ogólne, iż na ich bazie można tworzyć algorytmy dedykowane różnym problemom optymalizacyjnym. Takich ogólnych metod znanych jest obecnie kilkadziesiąt [1]. Co więcej, nie precyzują one w jaki sposób należy wykonywać poszczególne fragmenty metody, a jedynie nakreślają ogólną ideę ich działania. Dlatego nawet dla tego samego problemu można stworzyć wiele różnych algorytmów bazujących na tej samej metodzie.

W pracy bada się zastosowanie nowej metody kodowania rozwiązania za pomocą rangowego (wagowego) kodowania permutacji [2]. Z nowym, alternatywnym kodowaniem związany jest nie tylko format rozwiązania przechowywanego w pamięci komputera, ale co ważniejsze, nowe operatory. Wspomniane nowe operatory tworzą nowe sposoby zaburzania takiego rozwiązania jak i sposoby tworzenia rozwiązań potomnych (w algorytmach genetycznych) oraz rozwiązań sąsiednich (w algorytmach symulowanego wyżarzania i poszukiwania z zabronieniami). W szczególności badania dotyczą zastosowania alternatywnej metody kodowania wraz z odpowiadającymi jej operatorami

w algorytmach popraw typu symulowane wyżarzanie oraz algorytmach genetycznych. Przykładowy, wybrany na potrzeby niniejszej pracy, problem optymalizacyjny jest permutacyjnym problemem przepływowym z kryterium będącym sumą spóźnień realizacji zadań. Dla rozważanego problemu znany jest duży zestaw przykładów testowych oraz znane są rozwiązania uzyskane zestawem 17 wzorcowych algorytmów popraw [3]. Zagadnienie to jest stosunkowo trudne, z praktycznego punktu widzenia, a istniejące wyspecjalizowane algorytmy dostarczają mocno zróżnicowane jakościowo rozwiązania. W pracy próbuje się odpowiedzieć na pytanie, jaki wpływ na efektywność algorytmów ma wybór sposobu kodowania rozwiązania wraz z odpowiadającymi operatorami.

2. Opis problemu

Problem przepływowy (ang. flow shop) jest sztandarowym zagadnieniem związanym z szeregowaniem zadań, i jest od wielu lat analizowany przez naukowców z całego świata. Modeluje on wiele rzeczywistych procesów przemysłowych, w szczególności linie produkcyjne. Rzeczywiste procesy montażowe często używają taśm produkcyjnych, w których elementy dostarczane są w tej samej kolejności na poszczególne maszyny. Każda maszyna odpowiedzialna jest za wykonanie jednego etapu produkcji. W realnych produkcjach występuje często dodatkowe ograniczenie związanie z brakiem możliwości wyprzedzania jednego zadania przez inne. Oznacza to, iż kolejność wykonywania zadań na wszystkich maszynach jest jednakowa. Fakt ten powoduje szczególny nacisk na analizę i rozwiązanie problemu z uwzględnieniem powyższego ograniczenia. Przypadek ten znany jest jako permutacyjny problem przepływowy (ang. permutation flow shop).

Ogólnie harmonogramowanie zadań, polega na wybraniu maszyn dla każdej z operacji oraz wyznaczeniu momentów rozpoczęcia i zakończenia wykonywania operacji na wybranych wcześniej urządzeniach. W analizowanym zagadnieniu maszyny dla wszystkich operacji są ustalone i należy określić tylko moment rozpoczęcia wykonywania każdej z operacji. Problem polega na wyznaczeniu harmonogramu minimalizującego wskazane kryterium. Najczęściej spotykanym w literaturze kryterium jest minimalizacja czasu trwania całej produkcji. Często jednak, w praktyce oceniając jakiś harmonogram uwzględnia się przede wszystkim jego terminowość. Proces przemysłowy, który trwa dłużej, ale wykonuje wszystkie zadania w ustalonym terminie, jest bardziej pożądany niż, produkcja o krótszym czasie trwania, ale niedotrzymująca ustalonych terminów. Dlatego przyjętym w pracy kryterium oceny jest suma spóźnień realizacji zadań.

Analizowany w pracy permutacyjny problem przepływowy z kryterium będącym sumą spóźnień realizacji zadań należy do klasy problemów NP-trudnych [4]. Choć teoretycznie zagadnienie to należy do tej samej klasy problemów co permutacyjny problem przepływowy z kryterium będącym momentem zakończenia wszystkich zadań, to przez praktyków uważany jest on za znacznie trudniejszy. Wynika to z faktu, iż nie znaleziono dla niego własności pozwalających na efektywne wyeliminowanie słabych rozwiązań bez konieczności obliczania wartości funkcji celu. Także znane w literaturze akceleracje obliczeń, stosowane przy wyznaczaniu długości harmonogramu, nie mają zastosowania w przypadku wyliczania wartości kryterium będącego sumą spóźnień realizacji zadań.

2.1. Model matematyczny

Permutacyjny problem przepływowy z kryterium będącym sumą spóźnień realizacji zadań oznaczany jest w notacji Grahama [5,1] poprzez $F^*||T_{sum}$ i można go opisać w poniższy sposób.

Dany jest zbiór n zadań $J = \{1, \dots, n\}$ oraz zbiór m maszyn $M = \{1, \dots, m\}$. Zadanie $j \in J$ ma być wykonane kolejno na maszynach $1, 2, \dots, m$. Czynność polegająca na wykonaniu zadania j na maszynie k nazywamy operacją i notujemy jako parę (j, k). Operacja (j, k) realizowana jest bez przerw w czasie $p_{j,k} \geq 0$. Dla każdego zadania $j \in J$ zdefiniowany jest pożądany termin zakończenia $d_j \geq 0$. Ponadto zakłada się, że: (i) maszyna $k \in M$ może wykonywać co najwyżej jedną operację w danej chwili, (ii) nie można jednocześnie wykonywać więcej niż jednej operacji danego zadania, oraz (iii) każda maszyna wykonuje zadania w tej samej kolejności. Uszeregowanie dopuszczalne definiowane jest przez momenty zakończeń wykonywania poszczególnych operacji takich, że wszystkie powyższe ograniczenia są spełnione, oraz momenty rozpoczęcia wykonywania operacji są nieujemne.

Jeżeli poszukiwany harmonogram jest aktywny (czynności rozpoczynają się bez zbędnych opóźnień), to wygodnie jest zastosować tzw. permutacyjne sformułowanie badanego problemu, w którym zmienną decyzyjną jest permutacja π zbioru zadań J. Zbiór wszystkich możliwych takich permutacji oznaczać będziemy przez Π. Permutacja $\pi \in \Pi$ jednoznacznie wyznacza momenty zakończenia wykonywania operacji (j, k):

$$C(\pi, \pi(j), k) = \max\{C(\pi, \pi(j-1), k), C(\pi, \pi(j), k-1)\} + p_{j,k} \text{ dla } j \in J, k \in M, \quad (1)$$

gdzie $\pi(0) = 0$; $C(\pi, 0, k) = 0$, $k \in M$; $C(\pi, j, 0) = 0$, $j \in J$. W każdym dopuszczalnym uszeregowaniu $\pi \in \Pi$ dla danego zadania $j \in J$ można wyznaczyć zarówno moment jego zakończenia $C(\pi, j) = C(\pi, j, m)$, jak i odpowiadające mu spóźnienie:

$$T(\pi, j) = \max\{C(\pi, j) - d_j, 0\}. \quad (2)$$

Kryterium optymalizacji jest suma spóźnień wykonania wszystkich zadań:

$$T(\pi) = \sum_{j \in J} T(\pi, j). \quad (3)$$

Problem optymalizacji polega na znalezieniu permutacji minimalizującej sumę spóźnień w odpowiadającym jej harmonogramie:

$$\pi^* \in \arg\min T(\pi), \ \pi \in \Pi. \quad (4)$$

3. Kodowanie permutacji

W wielu problemach optymalizacji dyskretnej rozwiązaniem lub częścią rozwiązania jest pewna permutacja (np. problem komiwojażera, problem przepływowy, problem wyznaczenia cyklu Hamiltona). Klasyczne kodowanie permutacji jest intuicyjne i powszechnie znane. Jednakże, ze względu na porównanie do proponowanego alternatywnego kodowania, poniżej poddane analizie zostaną oba (tzn. klasyczne i alternatywne) podejścia. W obu przypadkach kodowaniu poddana zostanie permutacja

n elementowego zbioru $J = \{1,2,\dots,n\}$ indeksów bytów odpowiednich do problemu (odwiedzanych miast, wykonywanych operacji, odwiedzanych wierzchołków w grafie, itd.).

3.1. Kodowanie klasyczne

W klasycznych modelach [1] permutacja n elementowa wyznaczająca kolejność elementów ze zbioru J, kodowana jest w postaci sekwencji indeksów;

$$\pi = \big(\pi(1), \pi(2), \dots, \pi(n)\big) \tag{5}$$

gdzie $\pi(i) \in J$, $i = 1, \dots, n$ jest numerem obiektu na i-tej pozycji w permutacji π. Zbiór wszystkich rozwiązań oznaczyliśmy poprzez Π. Ponieważ każdej permutacji odpowiada dokładnie jedno kodowanie, liczność zbioru wszystkich możliwych kodowań odpowiada liczności wszystkich permutacji n elementowego zbioru J; $|\Pi| = n!$.

3.2. Kodowanie rangowe (wagowe)

W rangowym kodowaniu [2], sekwencja $\big(v(1), v(2), \dots, v(n)\big)$, n elementowego zbioru J, kodowana jest w postaci wektora n liczb rzeczywistych;

$$v = \big[v[1], v[2], \dots, v[n]\big] \tag{6}$$

gdzie $v[i] \in (0,1), i \in J$ oznacza rangę elementu o numerze i. Zbiór wszystkich wektorów kodujących oznaczamy V. Niech $poz(v,a), a \in J, v \in V$ oznacza rzeczywistą pozycję elementu a w kodowanej sekwencji; $v\big(poz(v,a)\big) = a$. Wektor v koduje sekwencję

$$(v) = \big(v(1), v(2), \dots, v(n)\big) \tag{7}$$

w niemalejącej kolejności rang elementów, a w przypadku tych samych wartości, kodowana kolejność wynika z indeksów elementów;

$$poz(v,a) < poz(v,b) \Leftrightarrow (v[a] < v[b]) \cup \big((v[a] = [b]) \cap (a < b)\big) \quad \text{dla} a, b \in J, v \in V. \tag{8}$$

Rzeczywista pozycja $poz(v,a)$ elementu a w kodowanej przez wektor v sekwencji (v) wynosi:

$$poz(v,a) = |\{b: (v[b] < v[a]), b \in J\} \cap \{c: (v[c] = v[a]) \cap (c \le a), c \in J\}| \quad \text{dla} a \in J, v \in V. \tag{9}$$

W celu ułatwienia analizy, praktycznie nie zmniejszając ogólności, załóżmy, iż wartości rang w wektorze permutacji v są parami różne; $v[a] \neq v[b]$ dla $a \neq b$. Ostatecznie, przy założeniu, $v[a] \neq v[b]$dla $a \neq b$, wzory (8) i (9) upraszczają się do:

$$poz(v,a) < poz(v,b) \Leftrightarrow v[a] < v[b] \text{ dla} a, b \in J, v \in V, \tag{10}$$

$$poz(v,a) = |\{b: v[b] < v[a], b \in J\}| + 1 \text{ dla} a \in J, v \in V. \tag{11}$$

Dla każdej permutacji zbioru J istnieje nieskończony, nieprzeliczalny zbiór wektorów $V(n) \subset V, n = |J|$, z których każdy koduje tę samą sekwencję.

W pracy [2] pokazano, iż dla obu sposobów kodowania dostępne są klasyczne operatory zmieniające kodowaną sekwencję (operator typy zamień, typu wstaw). Ponadto, w tej samej pracy pokazano, iż w alternatywnym kodowaniu zapamiętana jest nie tylko sama sekwencja, ale dodatkowo jej unikatowe cechy wewnętrzne związane rozkładem prawdopodobieństwa zaburzanych fragmentów sekwencji. Dodatkowo pokazano nowe operatory krzyżowania dedykowane omawianej metodzie kodowania oraz pokazano szereg własności dotyczących dziedziczenia, czyli przechodzenia relacji kolejnościowych z rodziców na potomstwo.

4. Badane algorytmy

W celu praktycznego zweryfikowania wpływu metody kodowania permutacji na sprawność algorytmów zaprojektowane i przebadane zostały 4 algorytmy popraw. Algorytmy te zostały bardzo dobrze dostrojone, o czym świadczy jakość generowanych rozwiązań będąca zdecydowanie lepsza niż dla znanych algorytmów literaturowych, patrz praca [3] oraz tabela 1.

Pierwszy algorytm nazwany SA, bazuje na metodzie symulowanego wyżarzania [6]. Startuje on, tak jak pozostałe omawiane algorytmy, z rozwiązania dostarczonego algorytmem NEH_{EDD} [7]. Zaburza on bieżące rozwiązanie wykonując ruch typu wstaw, polegający na przeniesieniu losowego elementu permutacji π w nowe losowe miejsce. Prawdopodobieństwo zaakceptowania nowego rozwiązania jest zgodne z powszechnie stosowaną funkcją akceptacji wynikającą z rozkładu Boltzmana [8]. W algorytmie zastosowano geometryczny schemat chłodzenia z temperaturą zmniejszającą się od 1000 do 10. Ilość wykonywanych iteracji wynosiła 1 milion.

Drugi prezentowany algorytm XSA, jest modyfikacją pierwszego. Różnica w stosunku do pierwszego algorytmu polega na zmianie metody kodowania permutacji z klasycznego na kodowanie rangowe. Startowe rangi wyznaczone zostały w ten sposób aby generowały sekwencję zwracaną przez algorytm NEH_{EDD}. Sposób zaburzania rozwiązania polegał na wpisaniu w losową pozycję wektora rang losowej wartości z zadanego przedziału.

Trzecim badanym algorytmem GA jest algorytm genetyczny [9,10]. Zrezygnowano w nim z stałej mutacji na korzyść mutacji automatycznej [11]. Dokonuje ona losowych zaburzeń typu wstaw (analogicznie jak w algorytmie SA) na osobnikach dublujących rozwiązania w danym pokoleniu. Operator krzyżowania polegał na skopiowaniu początkowych elementów permutacji z jednego rodzica oraz uzupełnieniu brakujących elementów w kolejności w jakiej występują u drugiego z rodziców. Liczba k kopiowanych elementów z pierwszego rodzica była dla każdego krzyżowania losowana z zakresu [0, n-1]. Liczność populacji jednego pokolenia ustalono na 100 osobników. Do generowania puli rodzicielskiej zastosowano dwie selekcje turniejowe tworzące kolejno listy A i B, zawierające odpowiednio 5 i 25 najlepszych osobników. Nowo powstające osobniki były potomkami osobników, z których jeden był losowany z puli A a drugi losowany z puli B. Populacja początkowa wygenerowana była poprzez stukrotne powielenie rozwiązania otrzymanego algorytmem NEH_{EDD}. Liczba generowanych pokoleń, 10 tysięcy, ustalona została w ten sposób by całkowita ilość przeglądanych rozwiązań była taka sama jak w poprzednich algorytmach, tj. 1 milion.

Czwartym z badanych algorytmów jest XGA. Jest on modyfikacją algorytmu GA, polegającą na zastąpieniu klasycznego kodowania osobników kodowaniem rangowym. Automatyczna mutacja zamieniała losową rangę na losową wartość wybieraną z ustalonego przedziału. Operacja krzyżowania polegała natomiast na utworzeniu nowych wartości rang będących sumą ważoną rang obydwóch rodziców odpowiednio z wagą U i $(1-U)$ gdzie wartość U była losowana z zakresu [-0.1,0.1].

Przedstawione algorytmy były także badane z różnymi innymi nieprzedstawionymi w pracy ustawieniami. Algorytmy symulowanego wyżarzania testowane były między innymi z różnymi wartości temperatury początkowej i końcowej, z sąsiedztwem bazującym na ruchach typu wymień. W algorytmach genetycznych testowana była także klasyczna metoda mutacji (oparta na losowych ruchach typu wstaw i typu zamień), różne metody selekcji puli rodzicielskiej, oraz różne sposoby krzyżowania osobników. Prezentowane w pracy ustawienia algorytmów wybrane zostały podczas badań wstępnych.

5. Badania numeryczne

Eksperymenty numeryczne przeprowadzone zostały na komputerze PC z procesorem Intel Core 2 Duo E6750 2.66GHz 2GB RAM działającym z pod 32 bitowym systemem operacyjnym Windows 7. Algorytmy oprogramowane zostały w języku C++ i skompilowane zostały przez kompilator TDM-GCC 4.7.1. Wszystkie napisane programy działały sekwencyjnie i wykonywane były na jednym rdzeniu procesora. Do badań wykorzystano 540 literaturowych przykładów testowych [3,12]. Przykłady te tworzą 12 grup po 45 instancji o jednakowym rozmiarze. Grupy nazwane zostały $n \times m$ gdzie n oznacza ilość zadań a m oznacza ilość maszyn w instancji. Instancje są mocno zróżnicowane zarówno pod względem rozmiaru, jak i ze względu na parametry generujące pożądane czasy zakończeń. Ilość zadań zmienia się od 50 do 350, ilość maszyn od 10 do 50, a ilość operacji od 500 do 17.500. Dokładna procedura generująca przykłady testowe opisana została w pracy [3], natomiast przygotowane instancje w postaci gotowych plików z danymi znajdują się na witrynie internetowej pod adresem [12].

W celu oceny jakości uszeregowania zastosowano zaproponowany w pracy [13] wskaźnik RDI (ang. Relative Deviation Index). Wartość wskaźnika RDI rozwiązania α^A generowanego przez algorytm A obliczany jest następująco:

$$RDI(\alpha^A) = \frac{T(\alpha^A) - T(\alpha^*)}{T(\alpha^\times) - T(\alpha^*)} \cdot 100\%, \tag{12}$$

gdzie α^* oznacza najlepsze a $\alpha^\times$ najgorsze rozwiązanie uzyskane zestawem algorytmów. Wartości najlepszych i najsłabszych rozwiązań dla testowych przykładów pobrano z witryny internetowej [12]. Wartość wskaźnika RDI w pobliżu 0% oznacza, iż badany algorytm uzyskał rozwiązanie tak dobre jak najlepsze rozwiązanie uzyskane zestawem wzorcowych algorytmów, a wartość 100% oznacza, iż uzyskany harmonogram jest na poziomie rozwiązania najsłabszego. Znalezienie lepszego niż najlepsze z referencyjnych rozwiązań objawia się ujemną wartością wskaźnika RDI.

W tabeli 1 przedstawione są uśrednione wartości wskaźnika RDI oraz uśrednione czasy działania prezentowanych algorytmów. Czas działania proponowanych algorytmów popraw nie zawiera czasu pracy algorytmu NEH_{EDD} generującego rozwiązanie startowe.

Tab. 1. Wartość średnia wskaźnika RDI oraz czas działania t analizowanych algorytmów

Grupa	NEH_{EDD}		SA		XSA	
	$RDI[\%]$	$t[s]$	$RDI[\%]$	$t[s]$	$RDI[\%]$	$t[s]$
50 × 10	17.52	0.0	1.39	1.0	1.56	1.2
50 × 30	19.88	0.0	2.69	2.4	2.59	2.6
50 × 50	18.21	0.0	3.13	3.9	2.93	4.1
150 × 10	13.65	0.0	0.59	2.7	0.58	3.3
150 × 30	20.39	0.1	1.65	7.0	2.33	7.6
150 × 50	21.88	0.1	3.15	11.4	3.30	12.0
250 × 10	10.04	0.1	0.01	4.3	-0.10	5.5
250 × 30	17.44	0.2	0.13	11.6	0.28	12.8
250 × 50	20.10	0.4	1.44	19.2	1.87	20.4
350 × 10	9.06	0.2	-0.32	6.0	-0.23	7.6
350 × 30	15.53	0.7	-0.45	16.8	-0.38	18.5
350 × 50	17.43	1.1	-0.10	27.2	0.41	28.8
Wszystko	16.76	0.2	1.11	9.5	1.26	10.4

Grupa	GA		XGA	
	$RDI[\%]$	$t[s]$	$RDI[\%]$	$t[s]$
50 × 10	4.68	1.4	4.46	1.6
50 × 30	8.51	3.9	7.57	4.5
50 × 50	7.96	6.1	7.29	6.9
150 × 10	3.25	3.3	3.20	4.1
150 × 30	7.26	10.0	7.57	10.9
150 × 50	9.05	17.6	8.87	19.0
250 × 10	2.07	4.9	2.20	6.0
250 × 30	5.19	15.9	5.89	17.6
250 × 50	7.03	27.3	7.38	29.6
350 × 10	2.19	6.8	1.92	8.7
350 × 30	4.22	20.7	5.00	22.4
350 × 50	5.60	36.8	6.06	39.1
Wszystko	5.59	12.9	5.62	14.2

Pierwszą własnością proponowanych algorytmów jest ich wysoka efektywność. Jakość generowanych rozwiązań jest zdecydowanie lepsza niż w przypadku literaturowych algorytmów [3] działających w wielokrotnie dłuższym czasie. Najlepszy z 17

literaturowych algorytmów SRHdostarcza rozwiązania o wskaźniku $RDI(SRH) = 2.55\%$ w średnim czasie 270s, podczas gdy prezentowany algorytm SA, w czasie blisko 30 razy krótszym, dostarczył rozwiązania o wskaźniku$RDI(SA) = 1.11\%$. Literaturowe algorytmy działały na komputerze Pentium IV 3.0GHz, czyli o podobnej mocy obliczeniowej, co sprzęt wykorzystany w prezentowanych badaniach.

Podczas badań algorytm SA wygenerował 97, a algorytm XSA 88, lepszych rozwiązań niż najlepsze referencyjne. Przykładowo w grupie 350 × 30 algorytm SA znalazł aż 24 (grupa zawiera 45 instancji) lepsze niż referencyjne rozwiązania. Skutkowało to między innym tym, iż wartość wskaźnika RDI tych algorytmów dla niektórych grup (patrz tabela 1) osiągała wartości ujemne. Oznacza to, że dostarczane rozwiązania (dla grup o ujemnym wskaźniku RDI) są średnio lepsze niż najlepsze z prezentowanych w literaturze rozwiązań. Pozytywną własnością algorytmów SA i XSA jest ich rosnąca przewaga nad pozostałymi algorytmami (w sensie miary RDI) w miarę wzrostu liczby zadań rozwiązywanych instancji.

6. Podsumowanie

Dla badanego problemu opracowane algorytmy oparte na symulowanym wyżarzaniu są wydajniejsze niż algorytmy genetyczne. Wydajność w niniejszym przypadku oznacza generowanie lepszych rozwiązań w krótszym czasie. Zastosowanie alternatywnego kodowania wraz z odpowiadającymi mu operatorami nie poprawiło jakości dostarczanych rozwiązań. Wręcz przeciwnie zaobserwowano niewielkie około 0.1% pogorszenie się wskaźnika RDI, a dodatkowo zmiana kodowania wydłużyła czas obliczeń o około 10%. Pomimo dobrych teoretycznych własności [2] stosowanie rangowego kodowania permutacji dla analizowanego problemu nie jest zalecane. Niemniej, ze względu na jego bardzo łatwą implementację oraz otrzymywanie przy jego pomocy stosunkowo niewiele gorszych rozwiązań (w stosunku do tradycyjnego podejścia), można stosować go wstępnie, podczas fazy projektowania algorytmu, aby po wstępnych badaniach przejść na kodowanie klasyczne.

Literatura

1. Smutnicki Cz., Algorytmy szeregowania,. akademicka oficyna wydawnicza Exit, Warszawa 2002.
2. Makuchowski M., Wektorowe kodowanie permutacji. Nowe operatory genetyczne, Innowacje w Zarządzaniu i Inżynierii Produkcji (red. R. Knosala), Oficyna Wydawnicza Polskiego Towarzystwa Zarządzania Produkcją, Opole 2013, ISBN 978-83-930399-5-1, str. 1205-1213.
3. Vallada E, Ruiz R, Minella G, Minimisingtotaltardiness in the m-machineflow- shop problem: A review and evaluation of heuristics and metaheuristics. Computers and Operations Research 2008; 35:1350–1373.
4. Du J., Leung JYT., Minimizingtotaltardiness on one machineis NP-hard. Ope-rationsResearch (1990), 38, p. 22—36.
5. Graham R., Lawler E., Lenstra J., Rinnooy Kan A., Optimization and approximation in deterministicsequencing and scheduling: a survey, Annals of DiscreteMathematics, 1979, 5, 287.
6. Kirkpatrick S., Gelatt C.D., Vecchi M.P. Optimisation by simulatedannealing, Science 220, 1983, 671-680.

7. Kim YD. Heuristics for flowshopschedulingproblemsminimizingmeantardiness. Journal of OperationalResearchSociety 1993;44:19–28.
8. Makuchowski M., Symulowane wyżarzanie w problemie gniazdowym z operacjami wielomaszynowymi nierównocześnie wykorzystującymi maszyny, Komputerowo Zintegrowane Zarządzanie, Tom II, WNT Warszawa 2004, str. 71-78,
9. Holland J.H., GeneticAlgorithms, Scientific American, 1992, 267, 44.
10. Goldberg D.E., Algorytmy genetyczne i ich zastosowania, WNT Warszawa, 1995.
11. Bożejko W., Makuchowski M., Solving the no-waitjob shop problem by usinggeneticalgorithm with automatic adjustement, The International Journal of Advanced Manufacturing Technology: volume 57, issue 5, 2011, pp. 735-752,
12. http://soa.iti.es/files/Eva_Instances.zip.
13. Zemel E. Measuring the quality of approximatesolutions to zero–one programmingproblems. Mathematics of Operations Research 1981;6:319–32.

Dr inż. Mariusz Makuchowski
Instytut Informatyki, Automatyki i Robotyki
Politechnika Wrocławska
50-372 Wrocław, ul. Janiszewskiego 11/17
tel./fax: (0-71) 320-29-61
e-mail: mariusz.makuchowski@pwr.wroc.pl