

Mariusz Makuchowski*

Algorytm genetyczny dla problemu gniazdowego z ograniczeniem bez czekania**

1. Wprowadzenie

W pracy rozważa się problem gniazdowy (*job shop problem*) z dodatkowym ograniczeniem bez czekania (*no-wait*). Optymalizowaną funkcją celu jest termin zakończenia wykonywania wszystkich zadań (*makespan*). Problem ten różni się od swojego klasycznego odpowiednika wymogiem rozpoczęcia wykonywania operacji dokładnie w chwili zakończenia wykonywania jej poprzednika technologicznego (bezwzględnie). Ograniczenie to jest typowe dla wielu rzeczywistych procesów produkcyjnych, w których przetwarzany produkt zmienia swoje własności fizyczno-chemiczne, przykładowo podczas wyrobu stali [1], produkcji żywności [2] czy wyrobu elementów betonowych [3].

Z punktu widzenia złożoności obliczeniowej, już klasyczny problem gniazdowy jest silnie *NP*-trudny, co zostało pokazane w pracy [4]. Analizowało go wielu badaczy, zaowocowało to opracowaniem szeregu algorytmów (różnego rodzaju). Dla niewielkich instancji dedykowane są algorytmy dokładne, bazujące na technice „dziel i ograniczaj” (*branch and bound*) [5], a dla pozostałych algorytmy przybliżone, najczęściej algorytmy popraw [6, 7]. Problem gniazdowy z ograniczeniem bez czekania jest także problemem silnie *NP*-trudnym [4], nawet w przypadku zredukowanym do dwóch maszyn [8]. Choć teoretyczna złożoność obu wymienionych wariantów problemu gniazdowego jest identyczna, to wielu badaczy uważa, z praktycznego punktu widzenia, problem z ograniczeniem bez czekania za znacznie trudniejszy. Przykładowo przedstawiony w pracy [9] algorytm dokładny, bazujący na metodzie podziału i ograniczeń, potrafi rozwiązać w sensownym czasie instancje o liczbie zadań nie większej niż 15 przy 5 maszynach i nie większej niż 10 przy 10 maszynach. Ponadto rozważany problem doczekał się wielu algorytmów przybliżonych bazujących na różnorodnych technikach. Przykładowo w pracy [10] przedstawiono algorytm oparty na technice poszukiwania z zabronieniami, w pracy [11] algorytm oparty na technice symulowanego wyżarzania, a w pracy [12] algorytm genetyczny z elementami symulowanego wyżarzania.

* Instytut Cybernetyki Technicznej, Politechnika Wrocławska

** Praca była częściowo finansowana ze środków KBN grant T11A01624

Dalszy układ pracy jest następujący. W rozdziale drugim podaję sformułowanie matematyczne rozważanego problemu oraz wprowadzam najpotrzebniejsze oznaczenia wykorzystywane w dalszej części pracy. W rozdziale trzecim definiuję dwie klasy rozwiązań, tzn. rozwiązania superaktywne oraz rozwiązania pseudoaktywne. Następnie przeprowadzam eksperyment numeryczny mający na celu praktyczną ocenę najlepszych rozwiązań występujących w danej klasie. W rozdziale czwartym opisuję zaproponowane algorytmy genetyczne, bazujące na wcześniej opisanych rozwiązaniach superaktywnych oraz pseudoaktywnych. Rozdział ten kończę badaniami numerycznymi z wykorzystaniem znanych w literaturze przykładów testowych. Uzyskane rezultaty porównuję do algorytmu literaturowego GASA [12], uwzględniając zarówno czas pracy algorytmu, jak i jakość otrzymanych rozwiązań. Ostatni piąty rozdział zawiera ogólne uwagi autora, zarówno na temat zaproponowanych klas rozwiązań superaktywnych i pseudoaktywnych, jak i zastosowania podejścia ewolucyjnego do rozwiązywania instancji rozważanego problemu.

2. Sformułowanie problemu

Problem gniazdowy z ograniczeniem bez czekania i funkcją kryterialną, będącą terminem zakończenia wykonywania wszystkich zadań, oznaczany jest w notacji Grahama przez: $J|no - wait|C_{\max}$, [13]. Może on być sformułowany zgodnie z notacją zaproponowaną w pracy [12] w następujący sposób:

Dany jest zbiór maszyn $M = \{1, 2, \dots, m\}$ oraz zbiór zadań $J = \{1, 2, \dots, n\}$ wykonywanych na tych maszynach. Dla każdego zadania $k \in J$ dany jest ciąg $O_k = (o_1^k, \dots, o_{r_k}^k)$ zawierający r_k operacji produkcyjnych. Operacja o_l^k składa się z pary (m_l^k, p_l^k) określającej kolejno wykorzystywaną maszynę oraz czas trwania operacji.

Ponadto w rozważanym problemie obowiązują następujące ograniczenia:

- *kolejnościowe*: operacje zadania k należy wykonać w kolejności O_k ;
- *synchroniczne*: każda maszyna może wykonywać w danej chwili co najwyżej jedną operację oraz nie można w tym samym czasie wykonywać więcej niż jednej operacji danego zadania;
- *bez czekania*: każda nie pierwsza operacja danego zadania musi rozpocząć się dokładnie w momencie zakończenia wykonywania operacji wcześniejszej tego samego zadania.

Zauważmy, że termin t_k rozpoczęcia wykonywania zadania k (moment rozpoczęcia wykonywania operacji o_1^k) określa precyzyjnie terminy rozpoczęć oraz zakończeń wykonywania wszystkich operacji $o \in O_k$ tego zadania. Dlatego jako rozwiązanie (uszeregowanie) problemu przyjmuję wektor $T = (t_1, \dots, t_n)$ terminów rozpoczęcia wykonywania wszystkich n zadań. Rozwiązanie spełniające wszystkie powyższe ograniczenia nazywamy rozwiązaniem dopuszczalnym. Długością $C_{\max}(T)$ uszeregowania T nazywamy termin wykonania wszystkich zadań

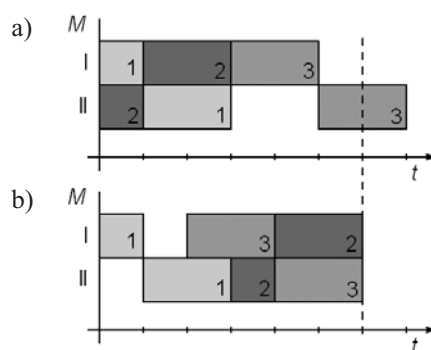
$$C_{\max}(T) = \max_{k \in J} \left(t_k + \sum_{l=1}^{r_k} p_l^k \right) \quad (1)$$

Problem polega na znalezieniu rozwiązania dopuszczalnego T^* o najmniejszej długości.

3. Uszeregowania superaktywne i pseudoaktywne

Uszeregowania superaktywne rozważane w pracy, zdefiniowane są precyzyjnie przez daną permutację zadań $\pi = (\pi(1), \dots, (n))$ (permutacja ładująca) oraz pewną procedurę przekształcającą tę permutację w rozwiązanie. Procedura tworząca uszeregowanie superaktywne z permutacji π składa się z n identycznych kroków. W i -tym kroku do częściowego rozwiązania dokładane jest zadanie $\pi(i)$, tworząc nowe uszeregowanie częściowe. Procedura zaczyna działanie z pustego uszeregowania, a po zakończeniu działania budowane uszeregowanie częściowe staje się ostatecznym rozwiązaniem. Dodanie zadania $\pi(i)$ w i -tym kroku procedury polega na wyznaczeniu minimalnego terminu $t_{\pi(i)} \geq 0$, spełniającego wszystkie ograniczenia względem $\pi(1), l = 1, \dots, i - 1$, uszeregowanych już wcześniej zadań. Zauważmy, że każda permutacja π zbioru J definiuje dokładnie jedno uszeregowanie, co więcej, uszeregowanie to jest zawsze dopuszczalne, co wynika bezpośrednio ze sposobu jego konstrukcji. Zaimplementowana procedura posiada złożoność obliczeniową $O(n^2 N^2)$, gdzie $N = \max_{k \in J} \{r_k\}$ oznacza największą liczbę operacji w szeregowanych zadaniach.

W rozważanym problemie nie ma żadnej gwarancji, że klasa rozwiązań superaktywnych zawiera poszukiwane rozwiązanie optymalne. Przykład takiej instancji (możliwie najmniejszy pod względem liczby maszyn i zadań) wraz z najlepszym rozwiązaniem superaktywnym oraz rozwiązaniem optymalnym znajduje się na rysunku 1.



Rys. 1. Harmonogram zadań: a) najlepsze rozwiązanie superaktywne; b) rozwiązanie optymalne

Chcąc zmniejszyć częstotliwość występowania niepożądaney, wspomnianej wyżej własności, zdecydowałem się na rozszerzenie rozwiązań superaktywnych o ich symetryczne odpowiedniki. W konsekwencji procedura generująca rozwiązania rozszerzona została o dodatkowy komplementarny moduł. W skrócie, moduł ten pracuje na „odbiciach lustrzanych” oryginalnych danych, generując harmonogram superaktywny, po czym zwraca rozwiązanie odpowiadające „lustrzanemu odbiciu” wygenerowanego harmonogramu. Ostatecznie jako rozwiązanie dla danej sekwencji ładującej przyjmuję harmonogram lepszy w sensie wartości funkcji celu. Uzyskane w ten sposób rozwiązania będę nazywał dalej rozwiązaniami pseudoaktywnymi (wszystkie zadania na wykresie Gantta dosunięte są najdalej jak to możliwe do lewej lub prawej strony).

3.1. Ocena jakości rozwiązań superaktywnych i pseudoaktywnych

W celu praktycznej oceny jakości rozwiązań superaktywnych oraz pseudoaktywnych, zaprojektowałem algorytm dokładny wyznaczający najlepszą (w sensie wartości funkcji celu) permutację ładującą. Badania przeprowadziłem dla przykładów, w których liczba zadań była nie większa niż 10 (wartości rozwiązań optymalnych podano w pracy [9, 12]). Otrzymane rezultaty zamieściłem w tabeli 1.

Tabela 1
Błędy względne najlepszych rozwiązań danej klasy
względem rozwiązań optymalnych

Grupa przykładów	Błędy względne [%] najlepszych rozwiązań klasy:	
	superaktywnej	pseudoaktywnej
La01-La05	0,70	0,00
La16-La20	0,19	0,00
Orb01-Orb05	0,07	0,03
Orb06-Orb10	0,46	0,00
Ft06, Ft10	0,00	0,00
Średnia	0,32	0,01

Z tabeli 1 wynika, że pogorszenie jakości rozwiązań związane z ograniczeniem przeglądania rozwiązań tylko superaktywnych czy pseudoaktywnych jest stosunkowo małe. Połączenie tego faktu z łatwością kodowania tych rozwiązań (jedna ładująca permutacja zadań) oraz stosunkowo prostą metodą ich konstrukcji sugeruje, że omawiane klasy są dobrą bazą dla projektowania efektywnych algorytmów przybliżonych. Oczywiście jest, że przeglądanie rozwiązań pseudoaktywnych umożliwia znalezienie lepszych rozwiązań, niż w przypadku przeglądania rozwiązań superaktywnych, jednakże obciążone jest to dwukrotnie dłuższym czasem wyznaczenia wartości funkcji celu.

4. Algorytm SGA i PGA

Ogólna idea algorytmu ewolucyjnego przedstawiona jest po raz pierwszy w pracy [14]. Algorytmy genetyczne są programami symulującymi dobór naturalny, ewolucję i dziedziczenie, występujące w przyrodzie. Pojedyncze osobniki symulowanej populacji utożsamiane są z rozwiązaniami danego problemu. Zazwyczaj jedna iteracja takiego algorytmu odpowiada życiu jednego pokolenia, w którym najsłabsze osobniki giną, a najlepiej przystosowane stają się rodzicami nowego pokolenia. Jakość przystosowania osobnika oceniana jest na podstawie wartości funkcji kryterialnej odpowiadającego mu rozwiązania. W celu nadania ewolucji tendencji do generowania coraz lepszych rozwiązań, nowo powstałe osobniki

dziedziczą geny (pewne atrybuty rozwiązań) swoich rodziców. Natomiast w celu uniknięcia sytuacji, w której wszystkie nowo generowane osobniki są bardzo do siebie podobne, stosuje się niewielką ich mutację.

Z powyższego opisu wynika, że podstawowymi operacjami występującymi w każdej iteracji algorytmu genetycznego są:

- 1) selekcja (wybór rodziców z całego pokolenia),
- 2) krzyżowanie (generowanie nowych osobników na podstawie materiału genetycznego rodziców),
- 3) mutacja (wprowadzenie niewielkich zmian w materiale genetycznym nowo powstałych osobników).

Szczegółowa implementacja poszczególnych elementów algorytmu (wariantu algorytmu o najwyższej jakości osiąganych rozwiązań z pośród kilkudziesięciu testowanych) opisana jest poniżej.

- *Selekcja*: w algorytmie zastosowano najprostszy rodzaj selekcji, zwaną selekcją twardą, polegającą na wyborze *pareN* najlepszych osobników.
- *Krzyżowanie*: do krzyżowania stosuje dwa operatory bazujące na bazie operatora PMX. Oba z nich losują po dwa punkty przecięcia się permutacji, dzieląc permutację rodzica A na trzy części. Pierwszy z nich, operator LRX, polega na skopiowaniu 1 i 3 części rodzica A, natomiast brakujące środkowe elementy uzupełniane są zgodnie z kolejnością występowania w rodzicu B. Drugi, operator MX, jest analogiczny, z tą różnicą, że kopiowaniu podlega środkowa część permutacji, a początkowa i końcowa jest uzupełniana.
- *Mutacja*: w algorytmie zastosowano mutację typu „wstaw”. Polega ona na przełożeniu jednego losowego elementu permutacji ładującej na nową wylosowaną pozycję.
- *Dodatkowa dywersyfikacja*: każdy osobnik o wartości funkcji celu występującej już u innego osobnika w pokoleniu poddawany jest dodatkowej mutacji. Ta dodatkowa mutacja wykonuje się tak długo, aż nastąpi zmiana wartości funkcji celu mutowanego rozwiązania.

Ostatecznie parametrami sterującymi pracą algorytmu są kolejno:

- *timeN*: maksymalny czas pracy algorytmu podany w sekundach;
- *iterN*: maksymalna liczba pokoleń symulowana przez algorytm;
- *idleN*: liczba pokoleń bez poprawy, po której następuje restart (dla *idleN*>0) lub zatrzymanie algorytmu (dla *idleN*<0);
- *childN*: liczba osobników w pokoleniu;
- *pareN*: liczba osobników będących rodzicami dla następnego pokolenia;
- *crossOver*: zbiór operatorów krzyżowania; operatory z tego zbioru używane są cyklicznie;
- *insertP*: prawdopodobieństwo mutacji;
- *N*: krotność powtórzenia algorytmu.

Opisany algorytm bazujący na rozwiązaniach superaktywnych nazywany jest algorytmem SGA, natomiast algorytm bazujący na rozwiązaniach pseudoaktywnych, algorytmem PGA.

4.1. Analiza eksperymentalna

Wszystkie badania przedstawione w tej pracy, zostały przeprowadzone na komputerze klasy PC z procesorem Athlon XP 2000+ taktowanym zegarem 1666 MHz. Prezentowane algorytmy zostały zaprogramowane w języku C++ i uruchamiane były w środowisku Windows XP.

Efektywność proponowanych algorytmów rozumiana zarówno jako szybkość działania, jak i jakość dostarczanych rozwiązań, przetestowana została na znanych w literaturze przykładach testowych. Przykłady te są zróżnicowane zarówno pod względem ilości maszyn (od 5 do 20), jak i ilości zadań (od 6 do 30). Dla lepszej interpretacji uzyskanych rezultatów, porównane zostały one względem wyników otrzymanych algorytmem hybrydowym GASA [12], łączącym w sobie algorytm genetyczny z elementami algorytmu symulowanego wyżarzania.

Tabela 2
Wyniki działania algorytmów GASA i PGA dla przykładów „łatwych”

Przykład	$m \times n$	C_{\max}^{OPT}	C_{\max}^{GASA}	t^{GASA} [s]	ρ^{GASA} [%]	C_{\max}^{PGA}	t^{PGA} [s]	ρ^{PGA} [%]
La01	5 x 10	971	1037	23	6,80	971	11	0,00
La02	5 x 10	937	990	24	5,66	937	10	0,00
La03	5 x 10	820	832	24	1,46	820	10	0,00
La04	5 x 10	887	889	25	0,23	887	9	0,00
La05	5 x 10	777	817	24	5,15	777	9	0,00
La16	10 x 10	1575	1637	39	3,94	1575	32	0,00
La17	10 x 10	1371	1430	42	4,30	1371	29	0,00
La18	10 x 10	1417	1555	42	9,74	1417	33	0,00
La19	10 x 10	1482	1610	40	8,64	1482	37	0,00
La20	10 x 10	1526	1693	45	10,94	1526	33	0,00
Orb01	10 x 10	1615	1663	39	2,97	1615	33	0,00
Orb02	10 x 10	1485	1555	40	4,71	1485	27	0,00
Orb03	10 x 10	1599	1603	41	0,25	1599	33	0,00
Orb04	10 x 10	1653	1653	43	0,00	1653	31	0,00
Orb05	10 x 10	1365	1415	44	3,66	1367	34	0,15
Orb06	10 x 10	1555	1555	42	0,00	1555	34	0,00
Orb07	10 x 10	689	706	43	2,47	689	31	0,00
Orb08	10 x 10	1319	1319	41	0,00	1319	31	0,00
Orb09	10 x 10	1445	1535	40	6,23	1445	39	0,00
Orb10	10 x 10	1557	1618	46	3,92	1557	35	0,00
Ft06	6 x 6	73	73	6	0,00	73	3	0,00
Ft10	10 x 10	1607	1607	41	0,00	1607	29	0,00

Właściwy test składa się z dwóch części. W pierwszej z nich bada się efektywność algorytmu na „łatwych”, a w drugiej na „trudnych” przykładach testowych. Przykłady „łatwe” to te (tab. 2), które udało się rozwiązać dokładnie stosując algorytm typu „dziel i ograniczaj” opisany w pracy [9], natomiast pozostałe noszą miano przykładów „trudnych”. Pewnego komentarza wymagają przykłady La06-La10, które udało się rozwiązać dokładnie, a zakwalifikowane zostały do przykładów trudnych. Decyzja o takim przydziale wynika bezpośrednio z faktu, iż pozostałe przykłady o liczbie zadań wynoszącej 15 są przykładami trudnymi. Ponadto praktyczna trudność ich rozwiązania (tab. 3 i tab. 5) jest na poziomie cechującym przykłady trudne.

Tabela 3
Średni błąd oraz suma czasów działania algorytmów GASA i PGA dla grup „łatwych”

Grupy przykładów	t^{GASA} [s]	$\bar{\rho}^{GASA}$ [%]	t^{PGA} [s]	$\bar{\rho}^{PGA}$ [%]
La01-La05	120	3,86	49	0,00
La16-La20	208	7,51	164	0,00
Orb01-Orb05	207	2,32	158	0,03
Orb06-Orb10	212	2,52	170	0,00
Ft06, Ft10	47	0,00	32	0,00
Wszystkie	794	3,68	573	0,01

W pierwszej części testu istnieje duża szansa na osiągnięcie rozwiązań niewiele różniących się od rozwiązania optymalnego (w sensie wartości funkcji celu), dlatego zdecydowałem się na zastosowanie algorytmu PGA, opartego na rozwiązaniach pseudoaktywnych (najlepsze z rozwiązań klasy pseudoaktywnej są znacznie bliżej rozwiązania optymalnego niż analogiczne rozwiązania w klasie rozwiązań superaktywnych, patrz tab. 1). Algorytm PGA uruchamiany był z następującymi parametrami sterującymi: PGA(999,250,-100,50,20,(coMX,coLRX),0.005,10). Znaczenie wszystkich parametrów (w kolejności ich wstępowania) wypunktowałem i omówiłem wcześniej.

Dla każdego przykładu wyznaczyłem długość uszeregowania C_{\max}^{PGA} generowanego przez badany algorytm PGA w czasie t^{PGA} . Analogiczne dane C_{\max}^{GASA} i t^{GASA} dla algorytmu GASA otrzymano uruchamiając 30 razy algorytm GASA (na komputerze z procesorem Athlon 1400 Mhz). Następnie dla każdego przykładu obliczyłem ρ^A , względny błąd w stosunku do rozwiązania optymalnego C_{\max}^{OPT}

$$\rho^A = 100\% \cdot (C_{\max}^A - C_{\max}^{OPT}) / C_{\max}^{OPT}, \quad A \in \{GASA, PGA\} \quad (2)$$

Jak już wspominałem, rozwiązania C_{\max}^{OPT} zaczerpnięto z pracy [9], w której otrzymano je stosując algorytm dokładny typu dziel i ograniczaj. Wartości C_{\max}^{PGA} , t^{PGA} , C_{\max}^{GASA} , t^{GASA}

oraz C_{\max}^{OPT} dla każdego z łatwych przykładów zamieszczono w tabeli 2. Średnie wartości \bar{p}^A błędu p^A oraz sumę czasów działania t_{sum}^A algorytmu $A \in \{GASA, PGA\}$ dla poszczególnych grup przykładów łatwych zamieściłem w tabeli 3.

Dla drugiej części testu algorytm SGA (dwukrotnie szybszy względem algorytmu PGA) wydaje się być bardziej odpowiedni. Przykłady testowe są tym razem na tyle trudne, że spodziewana odległość względem rozwiązania optymalnego będzie dużo większa niż błąd wynikający z ograniczenia się do analizy rozwiązań superaktywnych, zamiast rozwiązań pseudoaktywnych. Tym razem algorytm SGA uruchamiany był z parametrami: SGA(999,250,250,50,20,(coMX,coLRX),0.000,20).

Tabela 4
Wyniki działania algorytmów GASA i SGA dla przykładów „trudnych”

Przykład	$m \times n$	C_{\max}^{REF}	C_{\max}^{GASA}	t_{sum}^{GASA} [s]	σ^{GASA} [%]	C_{\max}^{PAGA}	t_{sum}^{SGA} [s]	σ^{SGA} [%]
La06	5 x 15	1248	1339	80	7,29	1318	40	0,00
La07	5 x 15	1172	1240	70	5,80	1172	41	3,67
La08	5 x 15	1244	1296	72	4,18	1298	40	3,86
La09	5 x 15	1358	1447	83	6,55	1415	40	3,24
La10	5 x 15	1287	1338	70	3,96	1299	40	2,95
La11	5 x 20	1656	1825	170	10,21	1698	69	0,66
La12	5 x 20	1462	1631	164	11,56	1561	69	4,65
La13	5 x 20	1610	1766	183	9,69	1729	70	0,99
La14	5 x 20	1659	1805	176	8,80	1703	69	3,38
La15	5 x 20	1713	1829	167	6,77	1752	70	2,34
La21	10 x 15	2048	2182	147	6,54	2077	135	0,29
La22	10 x 15	1918	1965	135	2,45	1931	135	-0,42
La23	10 x 15	2075	2193	136	5,69	2132	134	0,29
La24	10 x 15	2044	2150	133	5,19	2044	136	1,42
La25	10 x 15	1946	2034	142	4,52	1959	133	1,34
La26	10 x 20	2627	2945	332	12,11	2732	252	2,40
La27	10 x 20	2763	3036	311	9,88	2848	253	-0,11
La28	10 x 20	2696	2902	324	7,64	2866	252	1,15
La29	10 x 20	2498	2617	311	4,76	2564	247	0,60
La30	10 x 20	2602	2892	346	11,15	2674	244	3,15
La31	10 x 30	3690	4298	957	16,48	3857	581	4,61
La32	10 x 30	4082	4686	869	14,80	4223	580	3,11
La33	10 x 30	3712	4214	860	13,52	3854	581	0,94
La34	10 x 30	3716	4401	968	18,43	3789	588	4,20
La35	10 x 30	3797	4299	897	13,22	3997	576	1,24
La36	15 x 15	2796	2949	203	5,47	2918	283	-3,72
La37	15 x 15	3025	3216	192	6,31	3105	287	0,89
La38	15 x 15	2612	2762	202	5,74	2734	280	0,19
La39	15 x 15	2760	2908	195	5,36	2730	285	-1,09
La40	15 x 15	2728	2950	214	8,14	2594	288	-4,91
Ft20	20 x 20	1587	1675	184	5,55	1580	70	1,39

Dalszy przebieg drugiej części testu przebiegał analogicznie do pierwszej części, z tą zasadniczą różnicą, że porównywane algorytmy $A \in \{GASA, PGA\}$ testowane były na przykładach trudnych, a błędy rozwiązań σ^A liczone są względem C_{\max}^{REF} wartości funkcji celu rozwiązań referencyjnych

$$\sigma^A = 100\% \cdot (C_{\max}^A - C_{\max}^{REF}) / C_{\max}^{REF}, \quad A \in \{GASA, SGA\} \quad (3)$$

Rozwiązania C_{\max}^{REF} zaczerpnięto z pracy [15], w której otrzymano je, stosując algorytm typu poszukiwania z zabronieniami oraz algorytm typu symulowanego wyżarzania. Wartości C_{\max}^{SGA} , t^{SGA} , C_{\max}^{GASA} , t^{GASA} oraz C_{\max}^{REF} przykładów trudnych zawarte są w tabeli 4, natomiast średnie wartości $\bar{\sigma}^A$ błędu σ^A oraz sumę czasów działania t_{sum}^A algorytmu $A \in \{GASA, PGA\}$ dla poszczególnych grup przykładów trudnych zamieściłem w tabeli 5.

Tabela 5
Średni błąd oraz suma czasów działania algorytmów GASA i SGA dla grup „trudnych”

Grupy przykładów	t^{GASA} [s]	$\bar{\rho}^{GASA}$ [%]	t^{SGA} [s]	$\bar{\sigma}^{SGA}$ [%]
La06-La10	375	5,56	201	2,74
La11-La15	860	9,41	347	2,40
La21-La25	693	4,88	673	0,58
La26-La30	1 624	9,11	1 248	1,44
La31-La35	4 551	15,29	2 906	2,82
La36-La40	1 006	6,21	1 423	– 1,73
Ft20	184	5,55	70	1,39
Wszystkie	9 292	8,32	6 868	1,38

5. Uwagi końcowe

Z badań wstępnych zamieszczonych w punkcie 3 wynika, że zarówno stosowanie algorytmów przybliżonych bazujących na klasie rozwiązań superaktywnych, jak i pseudoaktywnych, wydaje się być bardzo dobrym podejściem. Algorytm bazujący na rozwiązaniach superaktywnych, względem analogicznego algorytmu bazującego na rozwiązaniach pseudoaktywnych, odznacza się dwukrotnie krótszym czasem działania oraz tylko niewiele słabszymi (w sensie wartości funkcji celu) dostarczonymi rozwiązaniami. Przykładowo dla testowanych łatwych instancji algorytm bazujący na rozwiązaniach superaktywnych, może

znaleźć w najlepszym przypadku rozwiązania o średnim błędzie 0,32%, podczas gdy algorytm bazujący na rozwiązaniach pseudoaktywnych może znaleźć rozwiązania ze średnim błędem mniejszym niż 0,01% (błąd liczony względem wartości funkcji celu rozwiązania optymalnego). Wynika z tego, że klasa rozwiązań pseudoaktywnych jest szczególnie polecana dla przykładów łatwych, gdzie osiągane błędy względne są bardzo małe. Natomiast klasę rozwiązań superaktywnych poleciłbym dla przykładów trudnych, gdzie znajdowane rozwiązania są ciągle dalekie od ideału (wartość funkcji celu jest dużo większa niż wartość funkcji celu rozwiązania optymalnego).

Zastosowanie algorytmu PGA dla przykładów łatwych, a algorytmu SGA dla przykładów trudnych było dobrą decyzją. Dodatkowe testy na trudnych przykładach wykazały przewagę algorytmu SGA nad PGA. Oba algorytmy działały jednakowo długo, przy czym algorytm SGA wykonywał 2 razy więcej iteracji niż algorytm PGA. Dostarczane algorytmem SGA rozwiązania były około 1% lepsze w sensie wartości funkcji celu od rozwiązań dostarczonych przez algorytm PGA. Natomiast w przypadku przykładów łatwych algorytm SGA mógłby w najlepszym wypadku znaleźć rozwiązania gorsze o 0,31% niż znalezione rozwiązania algorytmem PGA. Wynika to z faktu, że najlepsze rozwiązania w klasie rozwiązań superaktywnych są gorsze od rozwiązań dostarczonych algorytmem PGA (patrz tab. 1 i tab. 3).

Ponadto na szczególną uwagę zasługuje fakt, że dla *wszystkich* przykładów łatwych algorytm PGA znalazł najlepsze możliwe rozwiązania. Można powiedzieć, że znalazł rozwiązania optymalne w klasie rozwiązań pseudoaktywnych (patrz tab. 1 i tab. 3). Wśród tych rozwiązań aż 21 na 22 to rozwiązania optymalne (patrz tab. 2).

Porównując osiągnięte rezultaty względem literaturowego algorytmu GASA można zauważyć przewagę proponowanych algorytmów. Wszystkie porównywane algorytmy są algorytmami genetycznymi (algorytm GASA zawiera w sobie dodatkowo elementy symulowanego wyżarzania). Algorytmy SGA i PGA dostarczają rozwiązań zdecydowanie lepszych w sensie wartości funkcji celu niż literaturowy algorytm GASA działający w porównywalnym czasie. Pisząc o porównywalnym czasie, uwzględniam zarówno realny czas działania algorytmu, jak i różnicę w szybkości pracy komputerów. Dla łatwych instancji algorytm GASA dostarczył rozwiązań z średnim błędem 3,68%, podczas, gdy algorytm PGA w identycznym czasie (po uwzględnieniu różnicy szybkości komputerów) wygenerował rozwiązań o średnim błędzie 0,01%. Podobnie dla przykładów trudnych algorytm GASA dostarczył rozwiązania z średnim błędem 8,32%, podczas, gdy algorytm SGA wygenerował rozwiązania z średnim błędem 1,38%. Definicja omawianych błędów zawarta jest w podrozdziale 4.1.

Warto także zwrócić uwagę na fakt całkowitej rezygnacji, w testowanej wersji algorytmu SGA, z tradycyjnej mutacji, na rzecz mutacji osobników podobnych (patrz dodatkowa dywersyfikacja w rozdziale 4). Prezentowane podejście nie wymaga precyzyjnego strojenia a uzyskiwane rozwiązania są statystycznie lepsze (w sensie wartości funkcji celu) niż w przypadku podejścia tradycyjnego.

Literatura

- [1] Wismer D.A.: *Solution of the flowshop scheduling-problem with no intermediate queues*. Operations Research, 20, 1972, 689
- [2] Hall N., Sriskandarajah C.: *A survey of machine scheduling-problems with blocking and no-wait in process*. Operations Research, 44 (3), 1996, 510
- [3] Grabowski J., Pempera J.: *Sequencing of jobs in some production system*. European Journal of Operational Research, 125, 2000, 535
- [4] Lenstra J., Rinnooy Kan A.: *Computational complexity of discrete optimization problems*. Annals of Discrete Mathematics, 4, 1979, 121
- [5] Brucker P., Jurisch B., Sieveres B.: *A branch and bound algorithm for the job-shop-scheduling-problem*. Discrete Applied Mathematics, 49, 1994, 107
- [6] Nowicki E., Smutnicki C.: *A fast taboo search algorithm for the job shop scheduling problem*. Managment Science, 42 (6), 1996, 797
- [7] Pezzella F., Merelli E.: *A tabu search method guided by shifting bottleneck for job-shop-scheduling-problem, branch and bound algorithm for the job-shop-scheduling-problem*. European Journal of Operational Research, 120, 2000, 297
- [8] Sahni S., Cho Y.: *Complexity of scheduling shops with no-wait in process*. Mathematics of Operations Research, 4, 1979, 448
- [9] Mascis A., Pacciarelli D.: *Job shop scheduling with blocking and no-wait constraints*. European Journal of Operational Research, 142 (3), 2002, 498
- [10] Macchiaroli R., Mole S., Riemma S.: *Modelling and optimization of industrial manufacturing processes subject to no-wait constraints*. International Journal of Production Research, 37 (11), 1999, 2585
- [11] Raaymakers W., Hoogeveen J.: *Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing*. European Journal of Operational Research, 126, 2000, 131
- [12] Schuster C., Framinan J.: *Approximative procedures for no-wait job shop scheduling*. Operations Research Letters, 31, 2003, 308
- [13] Graham R., Lawler E., Lenstra J., Rinnooy Kan A.: *Optimization and approximation in deterministic sequencing and scheduling: a survey*. Annals of Discrete Mathematics, 5, 1979, 287
- [14] Holland J.H.: *Genetic Algorithms*. Scientific American, 267, 1992, 44
- [15] Makuchowski M.: *Problemy gniazdowe z operacjami wielomaszynowymi. Własności i algorytmy*. Raport Politechniki Wrocławskiej, seria: Preprinty, nr 37/2004

