ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1996
Seria: AUTOMATYKA z. 117 Nr kol. 1337

Adam JANIAK, Krzysztof CHUDZIK
Politechnika Wrocławska

ALGORYTM GENETYCZNY DLA JEDNOMASZYNOWEGO PROBLEMU SZEREGOWANIA ZADAŃ Z ZASOBAMI

Streszczenie. W artykule rozpatrzono problem szeregowania zadań na jednej maszynie z zadanymi terminami dostępności i czasami realizacji zależnymi od ilości przydzielonego zasobu przy kryterium minimalizującym maksymalną nieterminowość. Przedstawione są pewne własności problemu. Prezentowane są cztery wersje algorytmu genetycznego rozwiązującego ten problem. Przedstawione są wyniki przeprowadzonych eksperymentów numerycznych.

A GENETIC ALGORITHM FOR SINGLE MACHINE SCHEDULING PROBLEM WITH RESOURCES

Summary. The paper deals with some single machine scheduling problem with given release dates and processing times dependent on resources and with a criterion being the mninimalization of maximum lateness. Some specific properties of the problem are presented. To solve the problem four versions of genetic algorithms are presented. Some results of executed numerical experiments are presented.

1. Wstęp

W wielu zrobotyzowanych dyskretnych lub dyskretno-ciągłych procesach produkcyjnych mamy do czynienia z tzw. "wąskim gardłem", tzn. z sytuacją, gdy z punktu widzenia procesu tylko jedna maszyna (robot) jest krytyczna, a wszystkie pozostałe maszyny występujące w ciągu technologicznym, przed i po rozpatrywanej maszynie krytycznej, można traktować jako maszyny o nieograniczonej przepustowości, gdyż nie blokują one procesu technologicznego.

Niniejsza praca dotyczy właśnie problemu szeregowania zadań na jednej maszynie z czasami dostępności i kryterium maksymalnej nieterminowości. Dodatkowo, w problemie tym czasy wykonywania zadań zależą od ilości przydzielonego nieodnawialnego zasobu. Zadaniem optymalizacyjnym jest minimalizacja kryterium maksymalnej nieterminowości, przy ograniczonej ilości zasobu do rozdzielenia.

Ponieważ problem ten należy do klasy problemów NP-zupełnych, do jego rozwiązania zaproponowano algorytm przybliżony. Z mnogości technik wybrano algorytm genetyczny,

mimo iż technika ta wzbudza wiele kontrowersji. Powodem była chęć sprawdzenia przydatności tego typu obliczeń do przedstawionego problemu. Został więc zbudowany taki algorytm i zaimplementowany na komputerze.

Algorytm ten został następnie przetestowany. Wyniki testowania zostały porównane z wynikami najlepszych istniejących algorytmów przybliżonych opracowanych dla klasycznych problemów, tzn. bez zasobów [5] i specjalnie uogólnionych w pracy na przypadek z zasobami. Praca ta zawiera wyniki eksperymentów numerycznych i ich omówienie wraz z wnioskami.

2. Sformułowanie problemu optymalizacyjnego

Dana jest maszyna M, na której należy wykonać n niepodzielnych zadań, J_1 , J_2 , ... , J_n (gdzie indeksy zadań tworzą zbór N = {1, 2, ..., n}). Każde zadanie składa się z jednej operacji (dalej więc będzie mowa tylko o zadaniach). Maszyna M może wykonywać tylko jedno zadanie w danej chwili czasu.

Kolejność wykonywania zadań będzie opisywał wektor z = [z(1), z(2), ... , z(n)]. W problemie nie istnieją ograniczenia kolejnościowe na wykonywanie zadań, wobec czego dopuszczalne rozwiązania mogą tworzyć wszystkie permutacje kolejności zadań.

Czas p_i wykonywania każdego zadania J_i zależy w sposób liniowy od ilości przydzielonego nieodnawialnego zasobu u_i, jednakowego pod względem rodzaju dla wszystkich zadań, według zależności:

$$p_i = b_i - a_i u_i \text{ , gdzie } a_i \text{ , } b_i > 0, \forall\, i \in N \text{ są zadanymi parametrami modelu.} \qquad (1)$$

Na każde z zadań nałożone jest ograniczenie technologiczne na ilość zasobu możliwego do przydzielenia :

$$\alpha_i \le u_i \le \beta_i \text{ , } \forall\, i \in N. \qquad (2)$$

Ilość zasobu przydzielonego do poszczególnych zadań możemy opisać w postaci wektora u = [u(1), u(2), ... , u(n)].

Na całkowitą ilość przydzielonego do wszystkich zadań zasobu nałożone jest ograniczenie:

$$\sum_{i=1}^{n} u_i \le R. \qquad (3)$$

gdzie R jest globalną ilością zasobu, którym można dysponować.

Dla każdego z zadań J_i, $i \in N$, dane są czasy dostępności $r_i \ge 0$ (termin, od którego można rozpocząć wykonywanie zadania) oraz pożądany termin zakończenia wykonywania zadania d_i , którego przekroczenie jest dopuszczalne, ale wiąże się to z poniesieniem kary. Zarówno r_i , jak i d_i są z góry ustalone i niezależne od zasobu.

Zadaniem optymalizacyjnym będzie znalezienie takiego uszeregowania z* oraz takiego dopuszczalnego (tzn. spełniającego ograniczenia (2) i (3)) rozdziału zasobu u*, które będą minimalizowały kryterium maksymalnej nieterminowości:

$$L_{max} = L_{max}(z,u) = \max_{1 \le i \le n} \{ L_i(z,u) \} \quad (4)$$

gdzie $L_i(z,u) = C_i(z,u) - d_i$, a C_i to czas zakończenia wykonywania zadania J_i.
(Przez: S_i będzie oznaczony czas rozpoczęcia wykonywania zadania J_i).

Wykorzystując schematyczny zapis problemów szeregowania z zasobami [2], sformułowany powyżej problem będziemy oznaczać jako:

$$1 \mid r_i, p_i = b_i - a_i u_i, \Sigma u_i \le R \mid L_{max}. \quad (5)$$

W pracy [2] wykazano, że rozpatrywany problem jest silnie NP-zupełny, zatem nie można skonstruować do jego rozwiązania optymalnych algorytmów wielomianowych. Dlatego też w niniejszej pracy do jego rozwiązania opracowano algorytmy przybliżone typu genetycznego.

3. Pewne własności problemu

Własność 1

Rozpatrywany problem można sformułować w pewnej innej równoważnej postaci, która jest bardziej wygodna do analizy.

Niech M_1 oraz M_3 będą maszynami o nieograniczonej przepustowości (tzn., że każda z tych maszyn może wykonywać jednocześnie nieograniczoną liczbę zadań), natomiast M_2 - maszyną dotychczas rozpatrywaną (tzn. o ograniczonej przepustowości, która w danej chwili może wykonywać tylko jedno zadanie). Każde z zadań ma być wykonywane po kolei na każdej z tych trzech maszyn, tzn. najpierw na M_1, potem na M_2, i ostatecznie zakończone jest na M_3. Dla ustalonej kolejności wykonywania zadań z, oraz dopuszczalnego rozdziału zasobów u, każde zadanie J_i (i=1, 2, ..., n) jest wykonywane na M_1 przez okres r_i od momentu 0 do momentu r_i, czas $p_i(u_i) = b_i - a_i u_i$ od momentu $S_i(z,u) \ge r_i$ do momentu $C_i(z,u) = S_i(z,u)+p_i(u_i)$ zadanie J_i spędza na M_2; natomiast czas q_i od momentu $C_i(z,u)$ do momentu $Q_i(z,u) = C_i(z,u)+q_i$ przebywa na M_3, gdzie $q_i = d-d_i$, a z kolei $d = \max_{1 \le i \le n} d_i$.

Zadaniem optymalizacyjnym będzie znalezienie permutacji z^* na M_2 oraz rozdziału zasobu u^* minimalizujących czas zakończenia ostatniego zadania na maszynie M_3, tzn. zachodzi:

$$Q_{max}(z^*,u^*) = \min_z \min_u Q_{max}(z,u), \text{ gdzie } Q_{max}(z,u) = \max_{1 \le i \le n} Q_i(z,u) = Q_{max}$$

Równoważność (w sensie optymalnego sterowania (z^*,u^*)) pomiędzy obydwoma problemami wynika z faktu, że dla dowolnego z oraz u zachodzi:

$$Q_{max}(z,u) = \max_{1 \le i \le n} Q_i(z,u) = \max_{1 \le i \le n} (C_i(z,u)+q_i) = \quad (6)$$

$$= \max_{1 \le i \le n} (C_i(z,u)+d-d_i) = \max_{1 \le i \le n} L_i(z,u)+d = L_{max}(z,u)+d$$

Innymi słowy, sterowanie (z^*,u^*) optymalne dla jednego problemu jest równocześnie optymalne dla drugiego problemu, ponieważ oba kryteria różnią się tylko o wartość stałą d.

Definicje

Zdefiniujemy najpierw pojęcie segmentu (szczegółowy opis w [2]).

Niech ciąg zadań $<J_{z(i1)}, ..., J_{z(i)}, ..., J_{z(i2)}>$ na maszynie M_2 będzie ścieżką krytyczną dla problemu szeregowania sformułowanego we własności 1 dla permutacji z przy optymalnym (tzn. minimalizującym wartość kryterium dla tej ustalonej permutacji z) rozdziale zasobu u_z^*. Zachodzi więc:

$$Q_{max}(z, u_z^*) = r_{z(i1)} + \sum_{i=i1}^{i2} (p_{z(i)}(u_{z(i)}^*)) + q_{z(i2)} \qquad (7)$$

Podciąg zadań $<J_{z(i1)}, ..., J_{z(i)}, ..., J_{z(i2)}>$ na maszynie M_2, zawierający maksymalną liczbę zadań należących do tych samych dróg krytycznych (zadania $J_{z(i1)}$ i $J_{z(i2)}$ mogą należeć także do innych dróg krytycznych nie przechodzących przez pozostałe zadania tego podciągu), będzie nazywany segmentem zadań w permutacji z przy u_z^*.

Niech liczba segmentów wynosi k_z, a J_{k1} oraz J_{k2} będą odpowiednio pierwszym i ostatnim zadaniem k-tego segmentu.

Własność 2 [2]

Dla każdego uporządkowania zadań z, jeżeli uporządkowanie z' zostało uzyskane z uporządkowania z przez zamianę kolejności wykonywania zadań i jeśli $Q_{max}(z', u_{z'}^*) < Q_{max}(z, u_z^*)$, wówczas co najmniej jedno zadanie z k-tego segmentu w z zostało przesunięte przed pierwsze (J_{k1}) lub za ostatnie (J_{k2}) zadanie tego segmentu dla k=1 albo k=2, albo k= k_z. Przy tym, jeśli zadanie J_i z k-tego segmentu zostało przesunięte przed pierwsze zadanie J_{k1} tego segmentu i i≠k2, wówczas zachodzi: $r_i - r_{k1} < 0$, natomiast jeśli J_i zostało przesunięte za ostatnie zadanie J_{k2} k-tego segmentu i i≠k1, to zachodzi $q_i - q_{k2} < 0$.

4. Algorytm genetyczny

4.1. Ogólna struktura algorytmu genetycznego

Ogólny schemat algorytmu genetycznego jest dobrze znany, więc przypomnimy go tylko dla porządku:

1. N:=0
2. FAZA INICJACJI
3. FAZA PRZETRWANIA (SELEKCJI)
4. FAZA KRZYŻOWANIA
5. FAZA MUTACJI
6. N:=N+1
7. Jeżeli nie KONIEC, skocz do 3.

Wartość zmiennej N jest numerem generacji (iteracji algorytmu). W wierszu 2 tworzona jest populacja początkowa. W wierszu 3 stosowany jest mechanizm selekcji chromosomów, tzn. wyboru osobników do następnej generacji zależnie od ich przystosowania do środowiska. W wierszu 4 tworzone są nowe krzyżówki, a w wierszu 5 mutacje chromosomów. Algorytm dochodząc do wiersza 7 wykonuje skok do wiersza 3 i cykl algorytmu (tzw. czas życia populacji) powtarza się. Algorytm kończy swoje działanie w wierszu 7, jeżeli zostanie spełniony warunek końca pracy algorytmu, np.: algorytm wykona określoną liczbę przebiegów.

4.2. Opis operatorów krzyżowania i mutacji

Omówione zostaną teraz używane przy analizie algorytmu genetycznego operatory zarówno klasyczne,jak i specyficzne dla rozpatrywanego problemu szeregowania.

Reprezentacją rozwiązania dla rozpatrywanego problemu (chromosomem) będzie permutacja indeksów zadań (genów). Dla takiej reprezentacji zaproponowano trzy operatory krzyżówki i cztery mutacji.

4.2.1. Operatory krzyżowania

- Krzyżowanie klasyczne a)

Tniemy chromosomy matki i ojca na dwie części. Miejsce cięcia jest wybierane losowo, jednak położenie miejsca cięcia w obu chromosomach jest jednakowe. Chcąc zbudować syna (córkę), zachowujemy pierwszą część chromosomu ojca (matki) i uzupełniamy ją drugą częścią chromosomu matki (ojca).

W wyniku takiej operacji możemy otrzymać sekwencje nie będące permutacjami (pewnych indeksów może brakować, a inne mogą się powtórzyć). W związku z tym należy sekwencję poprawić w następujący sposób, aby stała się poprawnym chromosomem:

Dla syna (córki) przeglądamy sekwencję od strony lewej do prawej (lub odwrotnie), szukając powtarzających się indeksów. Po znalezieniu takiego zastępujemy go indeksem brakującym o najmniejszym numerze.

- Krzyżowanie klasyczne b)

Losujemy dwa miejsca cięcia jednakowe w chromosomach zarówno ojca, jak i matki. Fragment pomiędzy wylosowanymi miejscami wycinamy z chromosomu ojca (matki) i wstawiamy go na tę samą pozycję do chromosomu matki (ojca). Otrzymaną parę - syn i córka - poprawiamy tak jak w poprzednim przypadku.

- Krzyżowanie specyficzne

Do zbudowania tego operatora skorzystano z własności 2.

Aby zbudować chromosom syna (córki) zachowujemy z chromosomu ojca (matki) wszystkie geny odpowiadające zadaniom będącym granicami segmentów w tym chromosomie (w tej permutacji zadań przy optymalnym dla tej permutacji rozdziale zasobów). Pozostałe geny chromosomu ojca (matki) zachowujemy z prawdopodobieństwem 0.5. Wszystkie geny, które nie zostały zachowane, tworzą miejsca do obsadzenia w chromosomie syna (córki). Przeglądając kolejno od lewej chromosom matki (ojca), jeśli stwierdzimy, że jest on brakującym genem chromosomu syna (córki), stawiamy go na pierwszym, licząc od lewej, miejscu do obsadzenia w chromosomie syna (córki).

4.2.2. Operatory mutacji

- **Mutacja klasyczna a):** Mutacja ta polega na zamianie w permutacji dwóch sąsiednich losowo wybranych indeksów zadań (genów).
- **Mutacja klasyczna b):** Mutacja ta polega na zamianie w permutacji dwóch losowo wybranych indeksów zadań (genów).
- **Mutacja klasyczna c):** Mutacja ta polega na odwróceniu permutacji, tzn. na zastąpieniu permutacji [z(1), z(2), ... , z(n)] permutacją [z(n), z(n-1), ... , z(1)].
- **Mutacja specyficzna:** Mutacja ta polega na przesunięciu losowo wybranego genu poza segment (patrz własność 2), wewnątrz którego się znajdował.

4.3. Algorytm wyznaczania optymalnego rozdziału zasobu dla zadanej permutacji

Omówione operatory pozwalają stworzyć nowe permutacje zadań, jednak po każdym utworzeniu nowych członków populacji musi nastąpić wyznaczenie optymalnego dla nich rozdziału zasobu. W tym celu posłużono się tutaj algorytmem Hamachera & Tufekciego [1], który, jako algorytm dla wartości całkowitoliczbowych, został przystosowany do liczb rzeczywistych i zmodyfikowany na potrzeby badanego problemu.

4.4. Parametry badanego algorytmu genetycznego

Ponieważ największy wpływ na działanie algorytmu ma wybór odpowiedniego operatora krzyżowania i mutacji, więc eksperymentom poddano cztery następujące przypadki (w nawiasie są oznaczenia):

- specyficzne krzyżowanie i specyficzna mutacja (GA1),
- klasyczne krzyżowanie i specyficzna mutacja (GA2),
- specyficzne krzyżowanie i klasyczna mutacja (GA3),
- klasyczne krzyżowanie i klasyczna mutacja (GA4).

Do eksperymentów numerycznych, jako klasyczne, wybrano: krzyżowanie klasyczne a) i mutację klasyczną b).

Ponadto przyjęto, że:

- liczba iteracji algorytmu genetycznego wynosi 50,
- maksymalna liczność populacji wynosi 20 chromosomów (permutacji),
- populacja startowa wynosi 10 losowo wygenerowanych permutacji.
- wybór najlepszych osobników (o najmniejszej wartości funkcji kryterium) do kolejnej populacji (w następnej iteracji) jest deterministyczny,
- prawdopodobieństwo wyboru chromosomu do zostania rodzicem zależy od jego jakości,
- liczba par rodziców branych do fazy krzyżowania wynosi maksymalnie 5,
- w jednej iteracji algorytmu genetycznego jest 5 operacji mutowania.

5. Wyniki eksperymentów numerycznych

(Porównanie czterech wariantów algorytmu genetycznego z losowo generowanymi populacjami startowymi z klasycznymi algorytmami heurystycznymi)

Przebadano każdy wariant algorytmu genetycznego (GA1, ..., GA4), generując dla każdego grupę przykładów o określonej strukturze składającej się z 288 różnych przykładów.

Przy generowaniu grupy przykładów wzięto pod uwagę następujące parametry, których kombinacja dawała określony przykład:

- liczbę zadań w przykładzie - konstruowano przykłady po 20 i 100 zadań,
- zasobowy profil ogółu zadań, tzn. zbadanie, jaka część zadań posiada czas wykonywania operacji zależny od zasobu - konstruowano przykłady, gdzie 25% i 100% zależnych było od zasobów,
- globalna ilość zasobu - wyróżniono trzy poziomy: mała, średnia i duża ilość zasobu,
- parametry krzywej czas-zasób - konstruowano przykłady dla dwóch różnych metod wyznaczania wartości parametrów a_i i b_i (patrz wzór (1)), przyjęto też bez straty ogólności $\alpha_i=0$ i $\beta_i=1$,
- terminy dostępności (r_i)- konstruowano przykłady dla czterech różnych gęstości rozłożenia tych terminów na osi czasu,
- pożądane terminy zakończenia wykonywania zadań (d_i) - konstruowano przykłady dla trzech różnych gęstości rozłożenia tych terminów na osi czasu.

Łatwo sprawdzić, że istnieje 288 różnych kombinacji powyższych parametrów.

Otrzymane wyniki działania algorytmu genetycznego porównano z wynikami uzyskanymi przez heurystyczne algorytmy konstrukcyjne generujące permutacje zadań. Do porównań wybrano algorytmy: Jacksona, algorytm szeregujący według niemalejących d_i (lub

w drugiej notacji problemu - nierosnących q_i), Schrage [7] i algorytm Smutnickiego i Nowickiego [5].

W przypadku dwóch ostatnich algorytmów wywoływano je trzykrotnie dla różnych ustalonych czasów wykonywania:

1) $p_i = p_i(\alpha_i)$,

2) $p_i = p_i(\beta_i)$,

3) zadaniom o największej wartości parametru a_i przydzielano maksymalną ilość zasobu (by możliwie skrócić czas wykonywania zadań aż do wyczerpania zasobu).

5.1. Analiza dokładności algorytmu genetycznego

W żadnym z badanych przypadków rezultat otrzymany przez algorytm genetyczny nie okazał się lepszy od rozwiązań znalezionych przez konstrukcyjne algorytmy heurystyczne. Posłużymy się współczynnikiem oznaczonym BCR(N,GA,i) (Best Criterion Ratio) po N iteracjach, gdzie i oznacza liczony przykład, a GA wariant GA1 ... GA4. BCR jest zdefiniowany jako stosunek wartości najlepszego rozwiązania znalezionego przez GA po N iteracjach do wartości najlepszego rozwiązania znalezionego przez algorytmy konstrukcyjne. Dla każdego wariantu GA wyliczona może być średnia arytmetyczna BCR(N,GA,i) dla wszystkich liczonych przykładów (i=1...288), i wartość ta będzie oznaczana AVBCR(GA) (Average Value of BCR). Wyliczono też osobno te wartości dla problemów zawierających odpowiednio 20 (AVBCR20) i 100 (AVBCR100) zadań (dla N=50).

Tablica 1

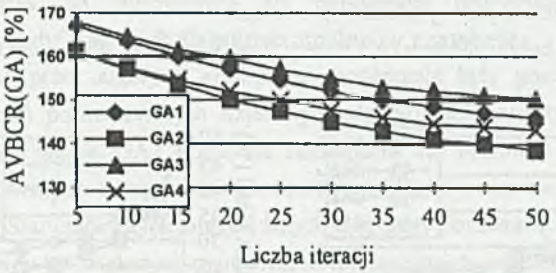
Porównanie jakości rozwiazań otrzymywanych przez GA

	iteracja 0	GA1	GA2	GA3	GA4
AVBCR	184%	146.01%	138.79%	150.51%	143.29%
AVBCR20	173%	123.15%	114.75%	126.11%	118.31%
AVBCR100	195%	168.88%	162.81%	174.91%	168.27%

Dokonano też zebrania wyników AVBCR(GA), liczonych na poszczególnych etapach obliczeń, tj. po liczbie iteracji 5, 10, 15, ..., 50. Wyniki te przedstawiono na rys 1.

Na podstawie zaprezentowanych wyników widać, że algorytm genetyczny dostarcza rozwiązania o wartości średnio na poziomie 139 - 150% najlepszego znalezionego przez algorytm heurystyczny. Widać ogromną różnicę w jakości dostarczonego rozwiązania dla problemów zawierających 20 i 100 zadań. Dla mniejszej liczby zadań wyniki są korzystniejsze. Należy jednak zauważyć, że dla większej liczby zadań współczynnik ten już na starcie (tzn. dla iteracji początkowej) miał gorsze wartości. We wszystkich przypadkach najlepsza okazała się

druga wersja algorytmu (GA2). Da się też zauważyć z rys. 1, że obliczenia były przerywane, gdy współczynniki AVBCR wykazywały jeszcze tendencje malejące.



Rys. 1.

5.2. Czas działania algorytmu

Algorytm był badany na komputerze klasy IBM PC 386 DX 40.

Tablica 2

Czas działania GA

	GA1	GA2	GA3	GA4
AVT	3m 36s	3m 15s	2m 09s	3m 07s
AVT20	0m 24s	0m 22s	0m 23s	0m 22s
AVT100	6m 49s	6m 09s	3m 53s	5m 53s

AVT (Average Value of Time) - średni czas działania algorytmu dla jednego przykładu liczony jest jako średnia dla wszystkich przykładów, AVT20 i AVT100 - tak jak AVT, ale dla przykładów o liczności odpowiednio 20 i 100 zadań. Najdłuższe czasy obliczeń dla 20 zadań wynoszą od około 1min do 1min 30 sek., a dla 100 zadań od około 20min do 25min. Pod względem czasu obliczeń najlepsze wyniki osiąga GA3.

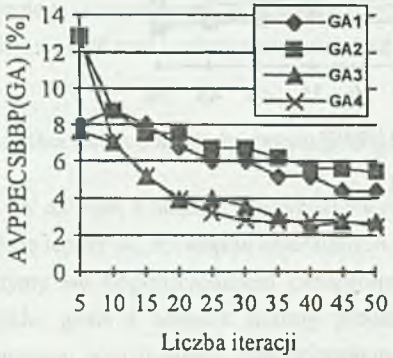
5.3. Porównanie operatorów krzyżowania i mutacji

Wprowadzone zostaną dwa parametry:

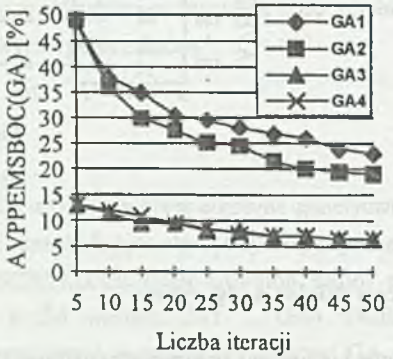
AVPPECSBBP (Average Value of Partial PErcentage of Children Strictly Better than Both Parents) - średnia liczba dzieci, powstałych w fazie krzyżowania, o wartości kryterium niższej niż wartości kryterium obojga rodziców i podzielona przez liczbę dzieci powstałych w fazie krzyżowania. Wartość ta jest wyrażona w procentach.

AVPPEMSBOC (Average Value of Partial PErcentage of Mutants Strictly Better than Original Chromosome) - średnia liczba mutantów o wartości kryterium niższej niż wartość kryterium chromosomu, z którego powstał, podzielona przez liczbę mutantów powstałych w fazie mutacji. Wartość ta jest wyrażona w procentach.

Powyższe wartości były wyliczane co 5 iteracji (stąd Partial), tj. dla iteracji od 1 do 5, od 6 do 10 itd.



Rys. 2.



Rys. 3.

Analizując powyższe wykresy, można stwierdzić, że liczba chromosomów lepszych od swoich rodziców maleje z każdą iteracją. Przebiegi wartości współczynnika nie pozwalają rozstrzygnąć, która metoda krzyżowania jest lepsza, bowiem średnio najlepsze wyniki daje krzyżowanie klasyczne wraz z mutacją specyficzną (GA2), a najgorsze ta sama metoda z mutacją klasyczną.

Na podstawie tych wyników widać też ogromną przewagę mutacji specyficznej nad klasyczną. Co więcej, porównując otrzymane wartości z tymi dla operacji krzyżowania, widzimy przewagę mutacji nad krzyżowaniem. Oznacza to, że większa liczba lepszych chromosomów powstaje drogą mutacji, a nie krzyżowania.

5.4. Wyniki działania algorytmu genetycznego startującego z populacji dostarczonej przez konstrukcyjne algorytmy heurystyczne

Do tego eksperymentu wybrano wersję GA2 jako dającą najlepsze rezultaty. Przeprowadzono obliczenia dla 282 przykładów. Poprawa wartości rozwiązania nastąpiła w 23 przypadkach, tj. 8% wszystkich wykonanych eksperymentów. Poprawa ta wynosiła średnio 0.6% wartości rozwiązania początkowego.

6. Wnioski

Prezentowany algorytm nie spełnił pokładanych w nim nadziei. Okazał się on algorytmem gorszym od stosowanych dla porównania heurystycznych algorytmów konstrukcyjnych, zmodyfikowanych dla potrzeb problemów z zasobami.

Generowane przez algorytm genetyczny rozwiązania były gorsze pod względem jakości od rozwiązań porównawczych uzyskanych algorytmami heurystycznymi. Najlepsza pod tym względem wersja GA2 podawała rozwiązanie na poziomie 139% najlepszych rozwiązań porównawczych. Poza tym widać różnicę w jakości rozwiązania, gdy uwzględni się rozmiar problemu. Startujące z tych samych danych algorytmy porównawcze dają rozwiązania lepsze dla problemów o większym rozmiarze, uwidaczniając tu bardziej swoją przewagę. Wynik ten jest raczej spodziewany, ponieważ założono stałą i niezależną od wielkości problemu obliczeniowego liczbę iteracji algorytmu genetycznego. To stawia w korzystniejszej sytuacji problemy, których reprezentacja danych jest mniejsza, bo algorytm przegląda większy procent wszystkich dopuszczalnych rozwiązań. Co prawda, analiza wykresów współczynników jakości (rys. 1. AVBCR(GA)) wskazuje, iż dalsze obliczenia mogą przynieść poprawę rozwiązania, ale wzrośnie, i tak już długi, czas obliczeń. Najlepsza wersja algorytmu genetycznego GA2 nie zdołała też w istotny sposób poprawić rozwiązań generowanych przez algorytmy porównawcze, co również rokuje, że zwiększenie liczby iteracji algorytmu genetycznego nie pozwoli uzyskać w rozsądnym czasie lepszych rozwiązań.

Czas obliczeń nie jest zadowalający. Wynosi on średnio kilkadziesiąt sekund dla małych problemów do kilku minut dla większych. Najszybszy jest algorytm GA3, ale jest on równocześnie algorytmem najgorszym pod względem jakości rozwiązania. Czas obliczeń wzrasta dosyć szybko wraz ze wzrostem rozmiaru problemu, w czym ma duży udział odpowiednio zmodyfikowany pseudowiclomianowy algorytm Hamachera i Tufekciego [1] odpowiedzialny za rozdział zasobów dla każdej losowo generowanej i otrzymywanej permutacji.

Analizując przydatność zaproponowanych operatorów genetycznych, możemy stwierdzić, że specyficzny dla tego problemu operator krzyżowania nie wniósł poprawy rozwiązania w stosunku do operatorów klasycznych. Natomiast specyficzny operator mutacji okazał się dobrym pomysłem. Algorytmy GA1 i GA2 posługujące się tym operatorem uzyskiwały o wiele wyższy procent lepszych potomków w tej fazie niż przy stosowaniu mutacji klasycznej. Uwzględnienie specyfiki problemu pozwoliło uzyskać tu zdecydowanie lepsze rezultaty. To spostrzeżenie można wykorzystać przy konstruowaniu następnych algorytmów do rozwiązywania sformułowanego w tej pracy problemu (nie muszą to być algorytmy genetyczne).

LITERATURA

1. Hamacher H. W., Tufekci S.: Algebraic flows and time-cost tradeoff problems. Annals of Discrete Mathematics, vol. 19, 1984, pp. 165-182.
2. Janiak A.: Dokładne i przybliżone algorytmy szeregowania zadań i rozdziału zasobów w dyskretnych procesach przemysłowych. Seria Monografie, Wydawnictwo Politechniki Wrocławskiej, Wrocław 1991.
3. Kański G.: Algorytm genetyczny dla problemu szeregowania zadań na jednej maszynie z czasami dostępności i kryterium maksymalnej nieterminowości oraz z rozdziałem zasobów nieodnawialnych. Praca magisterska. Politechnika Wrocławska, Wrocław 1994.
4. Michalewicz Z.: Genetic Algorithms + Data Structures = Ewolutions Programs. Springer-Verlag, 1992.
5. Nowicki E., Smutnicki Cz.: An approximation algorithm for a single-machine scheduling problem with release times and delivery times. Discrete Applied Mathematics, vol. 48, 1994, pp. 69-79.
6. Serdar Uckun, Sugato Bagchi, Kazuhiko Kawamura: Managing genetic search in job shop scheduling. IEEE Expert, 1993, pp. 15-24.
7. Lawler B.J, Lenstra J.K., Rinnoy Kan J.K., Shmoys D.B.: Sequencing and Scheduling: Algorithms and Complexity, Repotr BS-R8909, Center for Mathematics and Computer Science, Amsterdam, The Netherlands, 1989.

Recenzent: Dr hab. inż. Jan Kałuski, prof. Pol.Śl.

Wpłynęło do Redakcji do 30.06.1996 r.

Abstract

The paper deals with some single machine scheduling problem with given release dates and processing times dependent on resources and with a criterion being the minimalization of maximum lateness. The problem under consideration belongs to the class of strongly NP-complete problems, and there are no polynomially-solved algorithms for it. Some specific properties of the problem are presented. To solve the problem four versions of genetic algorithms are presented. These versions are costructed on the basis of classical cross-over and mutation operators and also some specyfing ones emplying problem properties proved. Some results of executed numerical experiments are presented. The main aim of these experiments was comparison of constructed genetic algorithms with classical heuristic ones generalized to the case with resources and also evaluation of usefulness of specific genetic operators proposed in the paper.