ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1994
Seria: AUTOMATYKA z.114 Nr kol. 1250

Adam JANIAK[†], Przemysław KOBYLAŃSKI[‡]
[†] Instytut Cybernetyki Technicznej,
[‡] Instytut Organizacji i Zarządzania,
Politechnika Wrocławska

ALGORYTM GENETYCZNY ROZWIĄZUJĄCY PRZEPŁYWOWY PROBLEM OBSŁUGI Z ZASOBAMI

Streszczenie: W artykule opisany został przepływowy problem obsługi, w którym czasy operacji, na kilku maszynach, zależą liniowo od ilości przydzielonego, podzielnego w sposób ciągły, lokalnie i globalnie ograniczonego, zasobu. Przedstawiono własności problemu oraz zaproponowano kilka implementacji algorytmu genetycznego do jego rozwiązywania.

GENETIC ALGORITHM FOR THE PERMUTATION FLOW-SHOP PROBLEM WITH RESOURCES

Summary: The paper deals with a permutation flow-shop problem where the processing times of jobs on some machines are linear functions with respect to the amount of continuously-divisible, locally and totally constrained resources. Some heuristic algorithm of a genetic type was applied to solve this problem. This algorithm employs some substantial problem properties that are presented.

EIN GENETISCHER ALGORITHMUS ZUR LÖSUNG DES FLIESSBEDIENUNGPROBLEMS MIT RESSOURCEN

Zusammenfassung: In dem Aufsatz wird ein Fließbedienungsproblem behandelt, in dem die Operationszeiten an einigen Maschinen von der Menge einer zugeordneten, stetig dividierbaren, lokal und global beschränkten Ressource linearweise abhängen. Eigenschaften des Problems werden dargestellt und einige Implementierungen eines genetischen Algorithmus für seine Lösung werden vorgeschlagen.

1. Wstęp

Klasyczny przepływowy problem obsługi (ang. flow-shop) opisany został dokładnie w literaturze. na przykład w pracach Bakera [1], Lagewega et al. [9] czy Grabowskiego et al. [6]. W pracach tych czasy wykonywania zadań opisane były stałymi liczbowymi. Jednak w wielu procesach produkcyjnych czasy operacji zmieniają się w przedziale ograniczonym maksymalnym (normalny) i minimalnym (graniczny) czasem wykonania. Czasy te zależą w sposób ciągły od przydzielonych zasobów, takich jak energia, paliwo, gaz, elektryczność, nakłady finansowe, których dostarczenie zmniejsza czas wykonania operacji. Problemy takie wymagają bardziej ogólnego podejścia do szeregowania, gdyż określają nie tylko kolejność wykonywania zadań, ale również rozdział ograniczonej ilości zasobu.

Artykuł nasz przedstawia rozszerzenie klasycznego przepływowego problemu obsługi na przypadek, kiedy czasy wykonania czynności są liniowo malejącymi funkcjami ilości zasobu, podzielnego w sposób ciągły i ograniczonego lokalnie i globalnie.

W następnej części zostanie formalnie opisany problem. W części trzeciej przedstawione zostaną własności problemu, które wykorzystujemy w algorytmie genetycznym opisanym w części czwartej. Część piąta jest opisem eksperymentu obliczeniowego, z którego wnioski zamieszczone są w części szóstej.

2. Sformułowanie problemu

Rozpatrywany problem można sformułować następująco. Każde z n zadań $J_1, \ldots, J_n$ musi być wykonane na m maszynach $M_1, \ldots, M_m$ w jednakowej kolejności na każdej maszynie. Każda maszyna M_v, $v = 1, 2, \ldots, m$, może obsługiwać w danej chwili tylko jedno zadanie. Zbiór indeksów operacji do wykonania na M_v oznaczany jest przez N^v. Zadanie J_i, $i = 1, 2, \ldots, n$, składa się z ciągu m operacji $O_{i1}, \ldots, O_{im}$, gdzie O_{iv} odnosi się do wykonania J_i na M_v podczas nieprzerwalnego czasu p_{iv}. Przyjmuje się, że dla indeksów maszyn ze zbioru M^1 czas wykonywania $p_{iv} \triangleq const$, $v \in M^1$, a dla pozostałych maszyn, których indeksy v należą do zbioru M^2, przy czym $M^1 \cap M^2 = \emptyset$, $M^1 \cup M^2 = \{1, 2, \ldots, m\}$:

$$p_{iv} \triangleq b_{iv} - a_{iv} r_{iv}, \; i = 1, 2, \ldots, n, \tag{1}$$

gdzie $a_{iv} > 0$, $b_{iv} > 0$ są znanymi parametrami, a r_{iv} jest ilością zasobu przydzielonego operacji O_{iv} ($v \in M^2$). Mówimy, że ilość zasobu przydzielonego operacjom O_{iv}, $i \in N^v$ ($v \in M^2$), zapisywana w postaci wektora $r_v \triangleq [r_{1v}, r_{2v}, \ldots, r_{iv}, \ldots, r_{n,v}]$, jest dopuszczalna, jeśli spełnia następujące ograniczenia:

$$\alpha_{iv} \leq r_{iv} \leq \beta_{iv}, \; i = 1, 2, \ldots, n \tag{2}$$

$$\sum_{v \in M^2} \sum_{i=1}^{n} r_{iv} \leq \hat{R}, \tag{3}$$

gdzie $\hat{R}$ jest całkowitą ilością zasobu, jaką dysponujemy, oraz α_{iv} i β_{iv} są zadanymi technologicznymi ograniczeniami na minimalną i maksymalną ilość zasobu przydzielonego

operacji O_{iv}. Jednocześnie

$$\sum_{v\in M^2}\sum_{i=1}^{n}\alpha_{iv} \leq \hat{R} \leq \sum_{v\in M^2}\sum_{i=1}^{n}\beta_{iv},\ 0 \leq \alpha_{iv} \leq \beta_{iv} < \frac{b_{iv}}{a_{iv}}.$$

Przez R oznaczany będzie zbiór wszystkich dopuszczalnych przydziałów zasobu $r = [r_{v_1}, r_{v_2}, \ldots, r_{v_j}, \ldots, r_{v_{m'}}]$ w rozpatrywanym problemie, gdzie $v_j \in M^2$, dla $j = 1, 2, \ldots, m'$, a m' jest rozmiarem zbioru M^2.

Kolejność wykonywania zadań może być reprezentowana przez permutację

$$\sigma = \langle \sigma(1), \sigma(2), \ldots, \sigma(i), \ldots, \sigma(n) \rangle$$

indeksów zadań ze zbioru $\{1, 2, ..., n\}$, gdzie $\sigma(i)$ opisuje ten indeks zadania, które jest na i–tej pozycji w σ.

Przepływowy problem obsługi (ang. flow-shop) z czasami operacji zależnymi od ilości przydzielonego zasobu, może być rozwiązywany w odniesieniu do wielu kryteriów optymalizac W poniższej pracy będziemy minimalizować długość czasu potrzebnego na zakończenie wszystkich zadań (nazywanego *całkowitym czasem realizacji*), który przy ustalonej kolejności zadań $\sigma \in \Pi$ i przydziale zasobu $r \in R$ oznaczać będziemy przez $M(\sigma, r)$. Innymi słowy, problem polega na znalezieniu takiego rozwiązania (σ^*, r^*), tj. takiej kolejności zadań $\sigma^* \in \Pi$ i takiego dopuszczalnego przydziału zasobu $r^* \in R$, że całkowity czas realizacji jest minimalizowany, tj.

$$M(\sigma^*, r^*) = \min_{\sigma\in\Pi}\min_{r\in R} M(\sigma, r),$$

$$M^* \triangleq M(\sigma^*, r^*).$$

Powyższy problem ma wielką wagę praktyczną. Jest uogólnieniem bardzo trudnego klasycznego przepływowego problemu obsługi, który był rozpatrywany na przykład przez Bakera [1], Lagewega et al. [9] i Grabowskiego et al. [6].

W [8] zostało udowodnione, że dwumaszynowy przepływowy problem obsługi, ze stałymi czasami operacji na jednej maszynie i liniowo zależnymi od zasobu na drugiej, jest NP–trudn

Do rozwiązywania rozpatrywanego problemu stosowanych będzie kilka implementacji algorytmu genetycznego, w których wykorzystywać będziemy własności problemu, omówione w następnej części.

3. Własności problemu

Można udowodnić, że rozpatrywany problem posiada następujące własności.

Własność 1. *Całkowity czas realizacji $M(\sigma, r)$, dla dowolnego dopuszczalnego rozwiązania (σ, r), jest równy*

$$M(\sigma, r) = \max_{1=i_0\leq i_1\leq i_2\leq\ldots\leq i_{m-1}\leq i_m=n} \sum_{j=0}^{m-1}\sum_{i=i_j}^{i_{j+1}} p_{\sigma(i)j+1}, \tag{4}$$

gdzie czasy wykonywania operacji są obliczone dla przydziału zasobu r.

Dla danego dopuszczalnego rozwiązania (σ, r) ciąg liczb naturalnych

$$\langle 1, i_1, i_2, \ldots, i_{m-1}, n\rangle$$

spełniających $1 \leq i_1 \leq \ldots \leq i_v \leq \ldots \leq i_{m-1} \leq n$ oraz ciąg operacji (patrz (4))

$$\langle O_{\sigma(1)1}, O_{\sigma(2)1}, \ldots, O_{\sigma(i_1)1};$$

$$O_{\sigma(i_1)2}, O_{\sigma(i_1+1)2}, \ldots, O_{\sigma(i_2)2};$$

$$\vdots$$

$$O_{\sigma(i_{v-1})v}, O_{\sigma(i_{v-1}+1)v}, \ldots, O_{\sigma(i_v)v};$$

$$\vdots$$

$$O_{\sigma(i_{m-1})m}, O_{\sigma(i_{m-1}+1)m}, \ldots, O_{\sigma(n)m}\rangle$$

nazywać będziemy, odpowiednio, *pozycjami ścieżki* i *ścieżką* związaną z tymi pozycjami w σ przy r.

Sumę czasów operacji tworzących ścieżkę w σ przy $r \in R$ będziemy nazywać *długością ścieżki.*

Jest oczywiste (patrz (4)), że długość każdej ścieżki w σ przy r nie jest dłuższa od wartości $M(\sigma, r)$.

Przez r^*_σ będziemy oznaczać *optymalny przydział zasobu dla ciągu* σ, tj. przydział zasobu, dla którego $M(\sigma, r^*_\sigma) = \min_{r \in R} M(\sigma, r)$.

Optymalny przydział zasobu r^*_σ, dla każdego $\sigma \in \Pi$, w przypadku liniowego modelu operacji, jest znajdywany przez użycie algorytmu Hamachera i Tufekciego [7] (z pewnymi modyfikacjami).

Dla danego rozwiązania (σ, r^*_σ) (tj. danej kolejności zadań σ z optymalnym przydziałem zasobu r^*_σ) pozycje ścieżki $\langle 1, i_1, i_2, \ldots, i_{m-1}, n\rangle$ i ścieżkę związaną z tymi pozycjami, dla której zachodzi

$$M(\sigma, r^*_\sigma) = \sum_{i=1}^{i_1} p_{\sigma(i)1} + \sum_{i=i_1}^{i_2} p_{\sigma(i)2} + \ldots + \sum_{i=i_{m-1}}^{n} p_{\sigma(i)m},$$

gdzie czasy operacji są policzone dla przydziału zasobu r^*_σ, będziemy nazywać, odpowiednio, *pozycjami ścieżki krytycznej* oraz *krytyczną* (najdłuższą) *ścieżką* w σ (przy r^*_σ).

Jak łatwo zauważyć, może być wiele ścieżek krytycznych w σ przy r^*_σ i rozwiązanie problemu znalezienia optymalnego przydziału zasobu $r^*_\sigma \in R$, dla dowolnej kolejności $\sigma \in \Pi$, nie jest jednoznaczne. Może być wiele optymalnych przydziałów zasobu z tą samą wartością kryterium $M(\sigma, r^*_\sigma)$, jednak z różną liczbą ścieżek krytycznych w σ lub krytycznych operacji (tj. operacji należących do krytycznej ścieżki).

Łatwo sprawdzić, że zachodzi następująca własność.

Własność 2. *Dla dowolnej kolejności $\sigma \in \Pi$ istnieje optymalny przydział zasobu r^*, taki że albo*

1. *dla każdej krytycznej operacji* O_{iv} $(v \in M^2)$ *w* σ $r^*_{iv} = \beta_{iv}$, *albo*

2. *dla każdej niekrytycznej operacji* O_{iv} *(tzn. nie należącej do żadnej ścieżki krytycznej w* σ*)* $r^*_{iv} = \alpha_{iv}$ *i ograniczenie dla całkowitej ilości zasobu jest spełnione, tj.*

$$\sum_{v \in M^2} \sum_{i=1}^{n} u^*_{iv} = R^*.$$

Mówiąc o optymalnym przydziale zasobu (dla ustalonego σ), będziemy mieli na myśli przydział spełniający Własność 2. Używany przez nas algorytm optymalnego przydziału zasobu podaje rozwiązanie posiadające tę własność. Łatwo zauważyć, że przydział $r^* \in R$ spełniający Własność 2 wyznacza minimalną liczbę ścieżek (oraz operacji) krytycznych w σ.

Niech $CP_\sigma \triangleq \langle 1, j'_1, j'_2, \ldots, j'_s, \ldots, j'_{s'_\sigma}, n\rangle$ będzie ciągiem kolejnych (różnych) pozycji ścieżek we wszystkich ścieżkach krytycznych w σ (tj. $1 < j'_1 < j'_2 < \ldots < j'_{s'_\sigma} < n$).

Podciąg kolejnych zadań $\langle J_{\sigma(j'_{s-1})}, J_{\sigma(j'_{s-1}+1)}, \ldots, J_{\sigma(j'_s-1)}, J_{\sigma(j'_s)}\rangle$, gdzie j'_{s-1} i j'_s są kolejnym pozycjami ścieżek krytycznych w CP_σ, będziemy nazywać $s-$tą *sekcją* w σ (przy r^*) i będziemy oznaczać przez $\overline{S}_s$. Symbolem S_s oznaczać będziemy zbiór zadań z $s-$tej sekcji, $s = 1, 2, \ldots, s_\sigma$, gdzie $s_\sigma = s'_\sigma + 1$.

W dowodach własności korzysta się z następujących definicji.

Podciąg zadań z $s-$tej sekcji w σ, który nie zawiera pierwszego zadania $J_{\sigma(j'_{s-1})}$ ani zadania ostatniego $J_{\sigma(j'_s)}$, będziemy nazywać $s-$tą *sekcją wewnętrzną* w σ (przy r^*) i będziemy oznaczać przez $\overline{S'}_s$, $s = 1, 2, \ldots, s_\sigma$.

Podciąg kolejnych operacji $< O_{\sigma(j'_{s-1})v}, O_{\sigma(j'_{s-1}+1)v}, \ldots, O_{\sigma(j'_s-1)v}, O_{\sigma(j'_s)v} >$ leżących na ścieżce krytycznej w σ (przy r^*), wykonywanych na tej samej maszynie M_v (gdzie j'_{s-1} i j'_s są kolejnymi pozycjami ścieżek krytycznych w CP_σ), będziemy nazywać $s-$tym *segmentem operacji* na maszynie M_v i oznaczać przez OS^v_s.

Analogicznie do definicji wewnętrznej sekcji, podciąg operacji z OS^v_s:

$$\langle O_{\sigma(j'_{s-1}+1)v}, O_{\sigma(j'_{s-1}+2)v}, \ldots, O_{\sigma(j'_s-1)v}\rangle$$

będziemy nazywać $s-$tym *wewnętrznym segmentem operacji* na maszynie M_v w σ (przy r^*).

Własność 3. *Dla każdej kolejności zadań* σ *istnieje optymalny przydział zasobu* r^*_σ *(posiadają Własność 2), który dla każdego wewnętrznego segmentu w* σ*, minimalizuje wartość, będącą sumą czasów wykonania jego operacji.*

Inaczej mówiąc, dla każdego σ istnieje r^*_σ taki, że dla każdego wewnętrznego segmentu (w σ przy r^*_σ) całkowita ilość zasobu przydzielonego do jego operacji jest jednocześnie przydzielona optymalnie ze względu na długość ścieżki złożonej z tych operacji.

Własność 4. *Jeśli ciąg* π *był uzyskany przez zamianę kolejności wykonania zadań w wewnętrznej sekcji ciągu* σ*, to* $M(\pi, r^*_\pi) \geq M(\sigma, r^*_\sigma)$.

Analogicznie jak Własność 4, można udowodnić, że zachodzi następująca własność

Własność 5. *Jeśli ciąg π został otrzymany z ciągu σ*

- *przez przeniesienie zadań z pierwszej wewnętrznej sekcji $\overline{S'}_1$ w σ przed pierwsze zadanie z $\overline{S}_1$, lub*
- *przez przeniesienie zadań z ostatniej wewnętrznej sekcji $\overline{S'}_{s_\sigma}$ w σ za ostatnie zadanie z $\overline{S}_{s_\sigma}$, to*

$$M(\pi, r_\pi^*) \geq M(\sigma, r_\sigma^*).$$

Powyższe własności zostaną użyte w algorytmie genetycznym zarówno w fazie krzyżówki, jak i mutacji.

4. Opis implementacji algorytmu genetycznego

Przedstawimy teraz ogólny schemat algorytmu genetycznego, a następnie dla każdego z jego faz, zaproponujemy różne możliwości implementacji. Część z nich została przetestowana podczas eksperymentu obliczeniowego, którego wyniki przedstawimy w następnej części. W fazie krzyżówki i mutacji, poza klasycznymi podejściami znanymi z literatury [5], wykorzystaliśmy własności problemu opisane w poprzedniej części.

W ogólnym schemacie algorytmu używamy chromosomów, będących permutacjami indeksów zadań. Dla każdego chromosomu znajdujemy optymalny przydział zasobu (tzn. minimalizujący całkowity czas realizacji) przez zastosowanie algorytmu Hamachera i Tufekciego [7].

4.1. Ogólny schemat algorytmu

Ogólny schemat algorytmu zapisać można następująco:

1. $N := 0$;
2. *Faza inicjowania*;
3. *Faza przetrwania*;
4. *Faza krzyżówki*;
5. $N := N + 1$;
6. *Faza mutacji*;
7. Jeśli nie jest spełniony *warunek zakończenia*, to przejdź do kroku 3;
8. *Faza analizy*.

W kolejnych podpunktach opiszemy poszczególne fazy algorytmu.

4.2. Faza inicjowania

Tworzymy początkową populację przez pozostawienie co najwyżej Q chromosomów z najlepszymi wartościami kryterium, które otrzymane były przez iterowanie algorytmów przybliżonych Dannenbringa [3], Campbella et al. [2] i Nawaza [10].

Ogólny schemat iteracji zapisać można następująco:

1. za pomocą algorytmu Hamachera i Tufekciego przydziel optymalnie zasób dla chromosomu $(1, 2, \ldots, n)$,
2. znajdź za pomocą algorytmu przybliżonego (Dannenbringa, Campbella et al. lub Nawaza) taką kolejność zadań, by całkowity czas realizacji, obliczony dla ostatniego przydziału zasobu, był minimalny,
3. dla kolejności zadań z poprzedniego kroku znajdź optymalny przydział zasobu, i jeśli całkowity czas realizacji poprawił się, to przejdź do kroku drugiego.

4.3. Faza przetrwania

Jest wiele możliwości definiowania fazy przetrwania ([4], [5]). Oprogramowaliśmy dwie z nich:

- **deterministyczna faza przetrwania:** zachowujemy co najwyżej Q chromosomów z najlepszą wartością kryterium,
- **losowa faza przetrwania:** chromosom ma tym większe prawdopodobieństwo przetrwa im lepszą ma wartość kryterium. Prawdopodobieństwo to jest równe:

$$\frac{ACTUALWORST - criterion}{Q \times ACTUALWORST - ACTUALSUM},$$

gdzie ACTUALWORST jest najdłuższym czasem realizacji wsród wszystkich czasów odpowiadających chromosomom bieżącej populacji, Q jest rozmiarem bieżącej populacji, ACTUALSUM jest sumą czasów realizacji dla wszystkich chromosomów w bieżącej populacji, a *criterion* jest czasem realizacji dla kolejności zadań opisanej danym chromosomem.

4.4. Faza krzyżówki

Spośród całej populacji wybieramy C par rodziców (odrzucamy pary złożone z jednakowych chromosomów) i tworzymy C par potomków. Poprzez analogię do koligacji rodzinnych, jednego z rodziców zwykło się nazywać ojcem, a drugiego matką, jak również jednego z potomków nazywa się synem, a drugiego córką.

Podobnie jak w fazie przetrwania prawdopodobieństwo wylosowania chromosomu uzależniamy od wielkości kryterium.

W eksperymencie obliczeniowym użyliśmy dwóch sposobów krzyżowania:

- **krzyżówka klasyczna** polegająca na podzieleniu chromosomów rodziców na dwie części (rozmiar pierwszej części jest wybierany losowo i jednakowy dla obu rodziców). Potomków otrzymuje się poprzez połączenie pierwszej części chromosomu ojca z drugą częścią chromosomu matki (u syna) i odpowiednio u córki, pierwszej części chromosomu matki z drugą częścią chromosomu ojca. Powstałe w ten sposób chromosomy mogą nie być poprawnymi permutacjami. W takim przypadku powtarzające się elementy zamienia się na te, które w permutacji nie występują;
- **krzyżówka specjalna** polegająca na pozostawieniu na swoich pozycjach w chromosomie syna tych zadań z chromosomu ojca, które leżą na pozycjach krytycznych (odpowiednio w chromosomie córki z chromosomu matki). Pozostałe zadania pozostawiamy na swoich pozycjach z prawdopodobieństwem 1/2, natomiast wolne pozycje uzupełniamy tymi zadaniami z chromosomu drugiego rodzica, które nie pozostały w chromosomie.

4.5. Faza mutacji

Oprogramowaliśmy wiele technik mutacji, jednak w eksperymencie obliczeniowym używamy dwóch:

- **mutacja klasyczna** polegająca na losowym wyborze dwóch zadań i zamianie ich miejscami w chromosomie (podobna do operacji sąsiedztwa używanej w symulowanym wyżarzaniu lub Tabu Search [4]),
- **mutacja specjalna**,losowe przeniesienie losowo wybranego zadania poza sekcję, w której się znajdowało.

4.6. Warunek zakończenia

Algorytm może być zatrzymany po osiągnięciu zadanej liczby iteracji, po wykonaniu określonej liczby iteracji bez poprawy kryterium lub po upływie zadanego czasu. W eksperymencie obliczeniowym przyjęliśmy, że algorytm będzie kończył pracę po 50 iteracjach.

4.7. Faza analizy

Podczas pracy programu, będącego implementacją algorytmu genetycznego, zbierane są dane o populacjach, które powstają w kolejnych iteracjach. W fazie analizy informacje te przetwarzane są prostymi metodami statystycznymi i prezentowane w formie raportów.

5. Eksperyment obliczeniowy

5.1. Opis eksperymentu

W eksperymencie wszystkie przykłady problemu składały się ze stu operacji i podzielone były ze względu na liczbę maszyn i zadań na dwa typy:

1. $m = 10$, $n = 10$,

2. $m = 5$, $n = 20$.

Każdy przykład problemu należał do jednej z trzech rodzin problemów:

1. wszystkie operacje na jednej losowo wybranej maszynie zależą od zasobu,
2. wszystkie operacje na dwóch losowo wybranych maszynach zależą od zasobu,
3. wszystkie operacje na trzech losowo wybranych maszynach zależą od zasobu.

Czas każdej operacji opisany był modelem liniowym

$$p = b - ar, \; r \in [\alpha, \beta],$$

przy czym dla każdej operacji $\alpha = 0$, natomiast $\beta = 1$, jeśli operacja zależała od zasobu lub $\beta = 0$ w przeciwnym przypadku.

Wśród przykładów problemu wydzielić można dwie grupy:

1. z parametrem b będącym losową liczbą całkowitą z przedziału $[1, 9]$,
2. z parametrem b będącym losową liczbą całkowitą z przedziału $[5, 9]$.

W obu grupach parametr a jest losową liczbą całkowitą z przedziału $[1, \max(1, b-1)]$.

Ze względu na całkowitą ilość zasobu $\hat{R}$ przykłady problemów podzielić można na trzy grupy:

1. o niskim poziomie zasobu,
2. o średnim poziomie zasobu,
3. o wysokim poziomie zasobu.

Wartości $\hat{R}$ w zależności od rodziny, typu oraz poziomu zasobu są następujące:

- rodzina 1, typ 1: $\hat{R} = 1$, $\hat{R} = 5$, $\hat{R} = 9$,
- rodzina 1, typ 2: $\hat{R} = 2$, $\hat{R} = 10$, $\hat{R} = 18$,
- rodzina 2, typ 1: $\hat{R} = 2$, $\hat{R} = 10$, $\hat{R} = 18$,
- rodzina 2, typ 2: $\hat{R} = 4$, $\hat{R} = 20$, $\hat{R} = 36$,
- rodzina 3, typ 1: $\hat{R} = 3$, $\hat{R} = 15$, $\hat{R} = 27$,
- rodzina 3, typ 2: $\hat{R} = 6$, $\hat{R} = 30$, $\hat{R} = 54$.

Mnożąc przez siebie liczbę grup, rodzin i typów, otrzymujemy 36 kategorii przykładów. Eksperyment nasz polegał na rozwiązaniu pięciu losowo wygenerowanych przykładów z każdej z 36 kategorii (razem 180 przykładów), za pomocą czterech różnych implementacji algorytmu genetycznego:

- *RCRM* z losową krzyżówką i losową mutacją,
- *RCSM* z losową krzyżówką i specjalną mutacją,
- *SCRM* ze specjalną krzyżówką i losową mutacją,
- *SCSM* ze specjalną krzyżówką i specjalną mutacją.

Za każdym razem maksymalny rozmiar populacji równy był 50, a obliczenia kończyły się po 50 iteracjach.

5.2. Wyniki eksperymentu

W prawie wszystkich obliczonych przykładach zaobserwowaliśmy poprawę maksymalnego czasu zakończenia w stosunku do populacji początkowej, generowanej za pomocą znanych heurystyk Dannenbringa, Campbella et al. i Nawaza. Mimo że poprawa była niewielka (średnio o 0.63%), to dopatrzeć się można tu pewnych prawidłowości.

Lepszą poprawę (średnio o 0.96% i maksymalnie o 3.45%) otrzymuje się stosując algorytm genetyczny *RCSM* (z losową krzyżówką i specjalną mutacją) niż w przypadku pozostałych trzech algorytmów.

6. Wnioski

Eksperyment obliczeniowy pokazał przydatność zastosowania algorytmów genetycznych w rozwiązywaniu problemów szeregowania zadań z uwzględnieniem zasobów. Potwierdził słuszność doboru mutacji specjalnej i skierował naszą uwagę na poszukiwanie lepszej krzyżówki niż ta, którą stosowaliśmy w eksperymencie.

LITERATURA

[1] Baker K.R.: A comparative study of flow-shop algorithms. Opns. Res. 23, 1975, 62.

[2] Campbell H.G., Dudek R.A., Smith M.L.: A heuristic algorithm for the n job, m machine sequencing problem. Mgmt. Sci., 16, 1970, B6 30-B6 37.

[3] Dannenbring D.G.: An evaluation of flow-shop sequencing heuristics. Mgmt. Sci., 23, 1977, pp.1174-1182.

[4] Davis L.: Handbook of Genetic Algorithms. ed. Van Nostrand Reinhold, New York, 1991.

[5] Goldberg D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, Mass. 1989.

[6] Grabowski J., Skubalska E. and Smutnicki C.: On flow-shop scheduling with release and due dates to minimize maximum lateness. J. Opl. Res. Soc., 34, 1983, 615.

[7] Hamacher H.W., Tufekci S.: Algebraic flows and time-cost tradeoff problems. Annals of Discrete Math., 19, 1984, pp.165-182.

[8] Janiak A.: Two-machine permutation flow-shop problem with transferable resource. Technical Report of Institute of Engineering Cybernetics Techn. Univer. of Wroclaw (w druku), 1987.

[9] Lageweg B.J., J.K. Lenstra and A.H.G. Rinnooy Kan: A general bounding scheme for the permutation flow-shop problem. Oper. Res., 26, 1978, 52.

[10] Nawaz M., Enscore E.E. Jr., Ham I.: A heuristic algorithm for the m-machine n-job flow-shop sequencing problem. OMEGA 11, 1983, pp.91-95.

Recenzent: Prof.dr hab.inż. Konrad Wala

Wpłynęło do Redakcji do 30.04.1994 r.

Abstract

The paper deals with a permutation flow-shop problem where the processing times of the jobs on some machines are linear, decreasing functions with respect to the amount of continuously-divisible, nonrenewable, locally and totally constrained resources (e.g. energy, catalyzer, raw materials). The purpose is to find a processing order of jobs (the same on each machine) and a resource allocation that minimize the length of the time required to complete all jobs, i.e. makespan. Since the problem is strongly NP-hard, some heuristic algorithm of a genetic type was applied to solve it. This algorithm strongly employs some substantial problem properties that were presented. The results of some computational experiment are also included.