ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1995
Seria: AUTOMATYKA z.116 Nr kol. 1296

Konrad WALA, Wojciech CHMIEL
Akademia Górniczo-Hutnicza, Kraków

NOWY SCHEMAT PROCESU GENETYCZNEGO POSZUKIWANIA NA PRZYKŁADZIE ZAGADNIEŃ HARMONOGRAMOWANIA Z MASZYNAMI RÓWNOLEGŁYMI*

Streszczenie. W pracy przedstawiono wyniki badań eksperymentalnych procesu optymalizacji realizowanego za pomocą algorytmu genetycznego dla testowego zagadnienia harmonogramowania z niezależnymi maszynami równoległymi. Badany algorytm genetyczny realizuje oryginalny proces genetycznego poszukiwania.

A NEW GENETIC SEARCH SCHEME ON AN EXAMPLE OF PARALLEL MACHINE SCHEDULING PROBLEM

Summary. This paper presents results of experimental examination of optimization process realized with the aid of genetic algorithm on the test example of parallel and unrelated machine scheduling problem. The investigated genetic algorithm realises original genetic search process.

NEUE GENETISCHE VERFARENSCHEMA AN DER BEISPIEL DES PARALLELEN MASCHINENZEITPLANUNGPROBLEM

Zusammenfassung. In dieser Forsezungsarbeit werden die Ergebnisse der Experimentaluntersuchungen des Optimierungsvorganges vorgestellt, mit der Hilfe eines genetischen Algorithmus, auf den Beispiel der parallel Maschinen. Dieser genetische Alorithmus realisiert orginal Proceß genetische Nachforschungen.

1. Wprowadzenie

Wiele trudnych problemów optymalizacji kombinatorycznej może być w chwili obecnej efektywnie rozwiązywanych za pomocą metaheurystyk z zakresu sztucznej inteligencji. Uczenie maszynowe stało się w ostatnich latach popularnym polem badań sztucznej inteligencji. Bazuje ono na idei konstrukcji programów optymalizacji kombinatorycznej posiadających zdolność samodoskonalenia się na podstawie doświadczeń w celu kierowania procesu poszukiwania do "obiecujących regionów" przestrzeni rozwiązań, gdzie istnieje większe prawdopodobieństwo znalezienia wysokiej jakości rozwiązań optymalizowanego problemu. Można zaobserwować rosnące zinteresowanie w ekspertymentalnym badaniu metaheurystyk, ażeby, z jednej strony lepiej poznać i zrozumieć istotę ich mechanizmu, a z

* Praca jest częściowo finansowana przez KBN, temat AGH nr 11.120.227

drugiej strony opracować coraz efektywniejsze algorytmy rozwiązywania konkretnych problemów. Do szczególnie interesujących metaheurystyk realizujących proces maszynowego uczenia się należy zaproponowana w 1975 roku przez Hollanda [4], strategia poszukiwania ogólnego przeznaczenia, odwołująca się do zasad ewolucji, nazywana algorytmem genetycznym.

Ogólnie, algorytm genetyczny jest iteracyjną procedurą poprawiania rozwiązań zawierającą zbiór rozwiązań *POPULACJA*, zbiór pseudogenetycznych operatorów nazywanych krótko operatorami genetycznymi, oraz funkcję oceny rozwiązań przeznaczoną do realizacji prawa selekcji rozwiązań zbioru *POPULACJA*: *"przeżywają najlepsi"* (por., monografie [3], [6]). Wiele wysiłku badaczy zostało włożone w opracowanie specyficznych operatorów genetycznych, które nie mają swojego odzwierciedlenia w przyrodzie. Zwykle funkcja oceny jest funkcją celu lub jej wersją skalaryzowaną często zawierającą dodatkową składową w postaci ważonej funkcji kary za naruszenie ograniczeń. Algorytm genetyczny realizuje strategię poszukiwania ogólnego przeznaczenia, to znaczy jest metaheurystyką odwołującą się do praw ewolucji. Pracując ze zbiorem *POPULACJA* algorytm eksploatuje niejawnie własności, które są wspólne dla "dobrych" rozwiązań.

Rozwiązania zagadnień są kodowane w postaci ciągów, których elementy są wybierane ze skończonego zbioru (alfabetu), a własności są definiowane przez ustalenia pewnych elementów ciągów. Rozwiązania zbioru *POPULACJA*, zwane rodzicami, są poddawane unarnym i binarnym przekształceniom za pomocą operatorów genetycznych. Uzyskane w ten sposób nowe rozwiązania nazywane są potomkami. Jest to krok rekombinacji algorytmu genetycznego, którego efekt jest taki, że "niejawnie" dobre własności są identyfikowane i podlegając przekształceniom wyznaczają rozwiązania nowe, które posiadają własności najlepszych rozwiązań zbioru *POPULACJA*. Krok rekombinacji jest powtarzany do chwili zadziałania kryterium stopu.

2. Algorytm genetyczny

Przyjęto, że zbiór *POPULACJA* ma stałą liczność, tj. liczba M rozwiązań znajdujących się w zbiorze *POPULACJA* nie zmienia się podczas całego procesu optymalizacji. Eksploatowany przez nas, już we wcześniejszych pracach [2], [5], [7], [8], [9], proces poszukiwania genetycznego pod nazwą algorytmu genetycznego *GA'94* można ogólnie opisać za pomocą następujących kroków.

Algorytm *GA'94*.

Start. Określ wartości parametrów:

-$\mu \geq 1$ - współczynnik wstępnej selekcji rozwiązań;

-M - rozmiar populacji;

- $p_i \geq 0$, $i=1,2,...,v$ - prawdopodobieństwa wylosowania operatorów genetycznych, gdzie v to liczba zastosowanych w algorytmie operatorów;
- L - liczba kroków algorytmu (określa liczbę generowanych potomków w czasie działania algorytmu);

Krok 1. Wyznaczenie populacji początkowej.
Wygeneruj, w sposób losowy, μM rozwiązań. W zbiorze *POPULACJA* umieść M najlepszych rozwiązań, biorąc pod uwagę funkcję oceny.

Krok 2. Wybór operatora genetycznego.
Wylosuj numer operatora genetycznego; każdy z v operatorów jest losowany z prawdopodobieństwem $p_i \geq 0$, $i=1,2,...,v$, przy czym $p_1+p_2+...+p_v=1$.

Krok 3. Wybór rodziców/rodzica.
Dla wybranego operatora wylosuj, zgodnie z rozkładem równomiernym, ze zbioru *POPULACJA*, zależnie od typu operatora, jedno (w przypadku operatora unarnego) lub dwa (w przypadku operatora krzyżowania) rozwiązania, które nazywamy rodzicami.

Krok 4. Generowanie potomków.
Za pomocą wylosowanego operatora genetycznego dokonaj modyfikacji rozwiązania-rodzica/rozwiązań-rodziców i wyznacz w ten sposób rozwiazanie-potomek /rozwiązania-potomki.

Krok 5. Poprawa rozwiązań w zbiorze *POPULACJA*.
Dla każdego potomka oblicz wartość funkcji oceny potomka. Jeżeli ta wartość jest lepsza od wartości najgorszego rozwiązania zbioru *POPULACJA*, to umieść takiego potomka w zbiorze *POPULACJA* usuwając zarazem z tego zbioru rozwiązanie najgorsze.

Krok 6. Warunek STOP'u.
Jeżeli wygenerowano zadaną liczbę L potomków to STOP; wydrukuj najlepsze rozwiązanie zbioru *POPULACJA*. W przeciwnym przypadku idź do kroku 2.

Wielkości μ, M, L, p_1, p_2,...,p_v są parametrami algorytmu. Współczynnik μ ($\mu \geq 1$) nazywamy współczynnikiem wstępnej selekcji rozwiązań. Jeżeli $\mu>1$, to w kroku 1 po wyznaczeniu μM rozwiązań następuje wstępna selekcja rozwiązań: M najlepszych jest umieszczanych w zbiorze *POPULACJA*, natomiast $(\mu-1)M$ gorszych zostaje odrzuconych.

W klasycznym algorytmie genetycznym proces poszukiwania jest zorganizowany w postaci następującego procesu ewolucji zbioru *POPULACJA*. W każdym kroku rekombinacji algorytmu cały "stary" zbiór jest zastępowany przez "nowy". Nowy zbiór *POPULACJA* (krok $t+1$) jest formowany na drodze selekcji rozwiązań ze starego zbioru (otrzymanego w kroku t), przy czym niektóre (ale tylko niektóre !) z rozwiązań podlegają przekształceniom za pomocą operatorów genetycznych, w celu reprodukcji nowych. Po pewnej liczbie generacji algorytm jest zbieżny - najlepsze rozwiązanie zbioru *POPULACJA* jest, mamy nadzieję, rozwiązaniem optymalnym.

Michalewicz w swoich pracach (por., rozdział 4 monografii [6]) eksploatuje zmodyfikowane podejście do algorytmu genetycznego nazwane przez niego algorytmem *modGA*. Modyfikacja w odniesieniu do algorytmu klasycznego polega na tym, że w *modGA* nowy zbiór *POPULACJA* jest formowany w ten sposób, że tylko część $r<M$ rozwiązań

starego zbioru jest wybierana jako rodzice do przekształcenia (rekombinacji) oraz r rozwiązań do odrzucenia. Selekcja jest realizowana zgodnie z wartością funkcji oceny rozwiązań: rozwiązania z wartością większą niż średnia wartość funkcji oceny rozwiązań zbioru *POPULACJA* mają większą szansę zostania rodzicami, natomiast rozwiązania z wartością funkcji oceny mniejszą niż średnia mają większą szansę być wybrane do odrzucenia. Nowy zbiór *POPULACJA* składa się z co najmniej $(M-r)$ rozwiązań starego zbioru (wszystkich rozwiązań, które zostały odrzucone) oraz co najwyżej r nowych rozwiązań-potomków, otrzymanych na drodze przekształceń przez operatory genetyczne z wybranych r rodziców ze starego zbioru (należy pamiętać, że nowe rozwiązanie jest wstawiane do populacji, jeśli jego "jakość" jest lepsza niż najgorszego rozwiązania w starej populacji). W ten sposób, w jednym kroku rekombinacji tylko r, $r<M$, rozwiązań podlega procesowi selekcji i przekształcenia.

Można zauważyć kilka istotnych zalet *modGA* w porównaniu do klasycznego algorytmu genetycznego. Po pierwsze, *modGA* lepiej wykorzystuje zasoby pamięci. W algorytmie tym unika się umieszczania wielokrotnych kopii tych samych rozwiązań w nowym zbiorze *POPULACJA*. Dodatkową cechą *modGA* jest mniejsza złożoność obliczeniowa całego algorytmu w porównaniu z wersją klasyczną. Po drugie, operatory genetyczne są stosowane do kompletnego rozwiązania, a nie do ich poszczególnych składowych. W ten sposób wszystkie operatory genetyczne mają ten sam status i łatwiej jest rozgraniczyć wpływ konkretnych operatorów na otrzymywane rozwiązania, biorąc pod uwagę prawdopodobieństwo ich wylosowania. I tak, jeżeli w algorytmie zastosowano trzy operatory (np. mutacji, krzyżowania, inwersji), to rodzice, podczas kolejnego kroku algorytmu, podlegają albo mutacji, albo krzyżowaniu, lub operacji inwersji. Po trzecie, tylko kilka (r, $r<M$) elementów zbioru *POPULACJA* jest poddawanych przekształceniom w każdym kroku rekombinacji, tj. w procesie tworzenia nowej generacji, tak więc *modGA* należy do klasy algorytmów *"Steady State GA"* [10].

W używanej przez nas wersji, *GA'94*, algorytmu genetycznego, opisanej powyżej, tylko jedno (w przypadku gdy w kroku 2 algorytmu jest wybrany operator unarny: $r=1$) lub dwa (w przypadku operatora krzyżowania: $r=2$) rozwiązania zbioru *POPULACJA* są wymieniane podczas każdego kroku rekombinacji. Tak więc algorytm ten należy także do klasy *Steady State GA* i posiada te same cechy, wymienione powyżej, co algorytm *modGA*. Ponadto mechanizm losowania rodziców jest najprostszy z możliwych, co zapewnia dodatkowe oszczędności czasu obliczeń.

3.Schemat procesu genetycznego poszukiwania

Zauważmy, z jednej strony, że ewolucja zbioru *POPULACJA* za pomocą algorytmu *GA'94* daje szansę najlepszym rozwiązaniom zbioru zostania rodzicami, wiele razy częściej niż

pozostałe rozwiązania i w ten sposób rozwiązania te zostają "*superosobnikami*" populacji. Stała liczność elementów zbioru *POPULACJA* oraz fakt, że superosobnik umieszcza dużą liczbę swoich potomków w zbiorze ogranicza liczbę potomków umieszczanych w zbiorze *POPULACJA* gorszych rozwiązań. Tak więc, w następnych generacjach superosobniki, eliminując różny od nich "*materiał genetyczny*", zawężają przeszukiwany obszar zbioru rozwiązań, co jest przyczyną szybkiej zbieżności rozwiązań ewolującego zbioru *POPULACJA* do optimum lokalnego. Z drugiej strony, w końcowym stadium procesu optymalizacji można oszczędzić na czasie trwania obliczeń ograniczając proces poszukiwania do regionów, które są otoczeniem (obszarem sąsiedztwa) właśnie superosobników. Wydaje się celowe sprawdzenie tych regionów na przykład za pomocą procedury optymalizacji lokalnej, którą można uważać także za specyficzny unarny operator genetyczny (por., [2],[5],[7],[8],[9]). Biorąc pod uwagę powyższe rozważania, proponujemy następujący dwuetapowy schemat procesu genetycznego poszukiwania, nazywany "*BREADTH-DEPTH*", w którym algorytm *GA'94* jest podstawową procedurą realizacji procesu ewolucji zbioru *POPULACJA*.

Przyjmijmy dalej, że zbiór *POPULACJA* jest liniowo uporządkowany za pomocą funkcji oceny rozwiązań: pierwsze rozwiązanie zbioru jest rozwiązaniem najlepszym, ..., ostatnie rozwiązanie (rozwiązanie nr M) jest rozwiązniem najgorszym w zbiorze. Przypomnijmy jeszcze w tym miejscu, że zbiór rozwiązań ewolującej populacji testuje jednocześnie wiele regionów zbioru rozwiązań zagadnienia, przy czym operatory krzyżowania (operatory binarne) umożliwiają próbkowanie nowych, i to "obiecujących" obszarów zbioru rozwiązań (bowiem algorytm wykorzystuje informacje "zebraną" podczas procesu ewolucji zbioru *POPULACJA*, której nośnikiem jest każde rozwiązanie) (por., [4]). Zakładamy, że funkcja operatorów krzyżowania w obu etapach pozostaje niezmienna: wartości prawdopodobieństw losowania tych operatorów w kroku 2 algorytmu *GA'94* nie podlegają zmianie.

Pierwszy etap procesu optymalizacji, etap *BREADTH*, jest realizowany podczas początkowych βL iteracji, natomiast drugi etap, etap *DEPTH*, podczas końcowych $(1-\beta)\ L$ iteracji, gdzie $0.5 \leq \beta < 1$ i L jest liczbą potomków wygenerowanych w czasie całego procesu optymalizacji. Trudno tu przedstawić dokładną receptę na dobór wartości β, bowiem zależy ona od rodzaju rozwiązywanego problemu. Najczęściej określa się ją eksperymentalnie wykonując eksperymenty obliczeniowe i porównując otrzymane wyniki.

Zakłada się, że w pierwszym etapie procesu optymalizacji powinien być testowany możliwie obszerny zbiór rozwiązań zagadnienia (stąd: *breadth*): w zbiorze *POPULACJA* powinien znajdować się różnorodny *materiał genetyczny*. W tym etapie procesu optymalizacji potomstwo *superosobników* nie powinno wypierać potomstwa pozostałych rozwiązań zbioru *POPULACJA*. W tym celu wprowadza się parametr TAB_POCZ (od tabu początkowe i proponuje się: TAB_POCZ$\in\{1,2,..,\alpha\}$, $\alpha \leq 0.1M$) i zakłada się,że TAB_POCZ pierwszych (tj. najlepszych) rozwiązań liniowo uporządkowanego zbioru *POPULACJA* jest zabronionych

(jest tabu) - rozwiązania te nie mogą zostać wybrane jako rodzice. Ponadto przyjmuje się, że prawdopodobieństwo losowania genetycznego operatora mutacji w drugim kroku *GA'94* jest równe γ, $\gamma>0$, a prawdopodobieństwo losowania operatora optymalizacji lokalnej jest równe zero.

W drugim etapie procesu optymalizacji realizowany jest proces poprawiania tylko najlepszych rozwiązań (stąd: *depth*), i w tym celu wprowadza się parametr TAB_KON (od tabu końcowe, i proponuje się: TAB_KON$\in\{\alpha 1,..,\alpha 2\}$, $\alpha 1 \geq 0.3M$, $\alpha 2 \leq 0.7M$) i zakłada się, że TAB_KON ostatnich elementów (tj. najgorszych rozwiązań) w uporządkowanym liniowo zbiorze *POPULACJA* jest zabronionych w drugim etapie - rozwiązania te nie mogą zostać wybrane jako rodzice. Ponadto przyjmuje się, że w tym etapie operator mutacji jest nieaktywny (prawdopodobieństwo jego wylosowania jest równe zero), natomiast operator optymalizacji lokalnej poprawia najlepsze rozwiązania dokonując pełnego przeglądu ich otoczeń: prawdopodobieństwo wylosowania operatora optymalizacj lokalnej w kroku 2 algorytmu *GA'94* jest równe γ.

4. Operatory genetyczne

Zauważmy, że w przypadku formalizacji rozwiązania zagadnienia w postaci permutacji Π informacją zawartą w rozwiązaniu jest tylko *kolejność* elementów. W takim przypadku w algorytmie genetycznym można zastosować tylko operatory zmieniające kolejność elementów rozwiązań. W zakodowanym w języku C++ algorytmie na komputery zgodne ze standardem IBM PC zastosowano dwa unarne operatory genetyczne. W celu wyznaczenia jednego potomka Π^1 rodzica Π operatory te wykonują następujące instrukcje.

1. Operator optymalizacji lokalnej.

(a). Określ otoczenie $S(\Pi)=\{\Pi':\Pi'=(\Pi(1),...,\Pi(i-1),\Pi(j),...,\Pi(j-1),\Pi(i),...,\Pi(n)),\ j\neq i\}$ permutacji $\Pi=(\Pi(1),...,\Pi(i-1),\Pi(i),...,\Pi(j-1),\Pi(j),...,\Pi(n))$;

(b). Znajdź permutację $\Pi' = arg\ min\{f(\hat{\Pi}):\ \hat{\Pi}\in S(\Pi),\hat{\Pi}\neq\Pi\}$, gdzie $f(\Pi)$ oznacza funkcję oceny rozwiązania Π;

(c). Jeżeli $f(\Pi') < f(\Pi)$, to podstaw $\Pi:=\Pi'$ i przejdź do **(a)**, w przeciwnym przypadku podstaw $\Pi^1:=\Pi$ i powróć do programu głównego.

2. Operator mutacji.

Wybierz w sposób losowy (rozkład równomierny) dwa elementy permutacji Π, powiedzmy $\Pi(i)$ i $\Pi(j)$, i dokonaj zamiany ich położenia wyznaczając w ten sposób potomka $\Pi^1 = (\Pi(1),...,\Pi(i-1),\Pi(j),...,\Pi(j-1),\Pi(i),...,\Pi(n))$.

Zauważmy, że operator mutacji zmienia tylko kolejność elementów w rozwiązaniu. Podobnie otoczenie $S(\Pi)$ jest definiowane przez zamianę kolejności elementów rozwiązania Π.

Dla rozwiązań, w których ważna jest kolejność elementów, operatory krzyżowania są bardziej skomplikowane, ponieważ klasyczne (ślepe) krzyżowanie może prowadzić do wyznaczenia rozwiązań niedopuszczalnych. Na przykład stosując jednopunktowy klasyczny operator do rodziców-permutacji:

$\Pi_1 = (1,2,3,4,5,6,7,8,9,10)$, $\Pi_2 = (7,3,1,10,9,2,5,4,8,6)$

otrzymujemy (jeżeli punkt krzyżowania jest po 5 elementach) potomki

$\Pi^1 = (1,2,3,4,5,2,5,4,8,6)$, $\Pi^2 = (7,3,1,10,9,6,7,8,9,10)$

z których żaden nie jest rozwiązaniem dopuszczalnym, tj. permutacją. Tak więc należy zastosować ściśle określone heurystyczne operatory krzyżowania. W naszych eksperymentach komputerowych stosowaliśmy trzy operatory krzyżowania:

3. Operator PMX.

4. Oprator OX.

5. Operator COMPX.

Operatory te określają dwóch potomków Π^1 i Π^2 poddając przekształceniom permutacje Π_1 i Π_2. Operatory *PMX* (partially matched crossover) i *OX* (order crossover) są opisane w monografii [3] (ss. 170-174), natomiast operator *COMPX* (composition crossover) został po raz pierwszy zaproponowany w pracy [8]; potomki są obliczane jako kompozycja rodziców: $\Pi^1 = \Pi_1 \otimes \Pi_2$, $\Pi^2 = \Pi_2 \otimes \Pi_1$.

Zauważmy, że permutacja $\Pi=(\Pi(1),\Pi(2),...,\Pi(n))$ jest bijekcją zbioru $N=\{1,2,...,n\}$ na siebie oraz para $(\chi(N), \otimes)$ jest znana jako skończona grupa permutacji, gdzie $\chi(N)$ jest zbiorem wszystkich odwzorowań N na N i $\otimes$ oznacza złożenie odwzorowań w $\chi(N)$.

5. Harmonogramowanie procesów na maszynach równoległych

Zagadnienie *Rm//Cmax* harmonogramowania procesów na niezależnych maszynach pracujących równolegle, którego celem jest minimalizacja terminu zakończenia wszystkich zadań $Cmax= max\{C_j\}$, $j=1,2,...,n$, należy do zagadnień silnie *NP-trudnych* (por. praca przeglądowa [1]), gdzie C_j jest terminem zakończenia realizacji zadania j, a n jest liczbą zadań, które należy przydzielić do jednej z m maszyn. Jeżeli przyjmiemy, że permutacja numerów zadań $\Pi=(\Pi(1),\Pi(2),...,\Pi(n))$ określa kolejność przydziału zadań do najwcześniej wolnej maszyny, to termin *Cmax* określa następująca procedura.

Procedura Cmax.

Wejście: permutacja zadań $\Pi=(\Pi(1),\Pi(2),...,\Pi(n))$;

Krok 1. Dla $i=1,2,...,m$ podstaw $T(i) := 0$;

Krok 2. Dla $j=1$ do n wykonaj:

(i) określ numer najwcześniej wolnej maszyny $i^* = arg\ min\{T(i),\ i \in \{1,2,...m\}\}$;
(ii) przydziel zadanie $\Pi(j)$ do maszyny i^* podstawiając $T(i^*) := T(i^*) + p_{i^*\Pi(j)}$;
gdzie p_{ij} jest czasem wykonania zadania j na maszynie i;

Krok 3. Podstaw $Cmax := max\{T(i): i \in \{1,2,...m\}\}$, gdzie $T(i)$ jest terminem zakończenia zadań przydzielonych do maszyny i.

Poniżej zostaną przedstawione wyniki eksperymentów komputerowych dla następującego zagadnienia testowego. Należy wykonać $n = 25$ zadań na $m=5$ niezależnych maszynach (zagadnienie R5//*Cmax*), gdzie czas wykonania zadań p_{ij} dla $i=1,2,...,5$; $j=5k+1, 5k+2,...,5(k+1)$ i $k=0,1,...,4$ jest określony wzorem:

$$p_{ij} = \begin{cases} (i-k)(j-5k) & \text{jeżeli } (i-k) > 0 \\ (5-k+i)(j-5k) & \text{w przypadku przeciwnym} \end{cases}$$

Rozwiązanie, w którym zadania 1, 2,...,5 są przydzielone do maszyny 1, zadania 6, 7,...,10 są przydzielone do maszyny 2,..., zadania 21,22,...,25 są przydzielone do maszyny 5, jest rozwiązaniem optymalnym, przy czym *Cmax*=15.

Wszystkie eksperymenty optymalizacji startują z tego samego, otrzymanego na drodze losowania, początkowego zbioru *POPULACJA*, w którym najlepsze rozwiązanie posiada wartość *Cmax* = 51. Stałymi parametrami algorytmu optymalizacji podczas eksperymentów były wartości: μ=1, M =150, TAB_POCZ=5, TAB_KON=100, L= 10 000. W sumie wykonano 60 eksperymentów, po 15 eksperymentów oznaczonych symbolami Exp.1, Exp.2, ..., Exp.15, dla wartości β = 0.6; 0.7; 0.8; 0.9. Poszczególne eksperymenty Exp.1, Exp.2, ...,Exp.15, różnią się wartościami prawdopodobieństw $p_1, p_2,...,p_5$ wyboru operatorów genetycznych przez algorytm. Dwa najlepsze rozwiązania o wartości funkcji celu *Cmax*=25 otrzymano, zauważmy - po niedużej liczbie L =10 000 iteracji, dla następujących zestawów parametrów algorytmu:

(1) $\beta=0.7; p_1/p_2=0.4;\ p_3 = 0.2; p_4 = 0.3;\ p_5 = 0.1$

(2) $\beta=0.9; p_1/p_2=0.3;\ p_3 = 0.3; p_4 = 0.3;\ p_5 = 0.1$

Tablica 1 przedstawia przebieg procesu optymalizacji dla zestawu parametrów (1), gdzie *Cmax* jest wartością funkcji celu najlepszego rozwiązania w zbiorze *POPULACJA*.

Tablica 1

Przykład przebiegu procesu optymalizacji

Liczba iteracji	0	459	572	1874	2455	2652	4272	7001	7009	7121	7170	7979	10000
Cmax	51	47	46	45	44	43	39	37	32	29	26	25	25

W tablicy 2 przedstawiono wartość średnią funkcji celu C*śr* najlepszych rozwiązań zbioru *POPULACJA* z 15 eksperymentów dla testowanych wartości β.

Tablica 2

Średnia wartość funkcji celu najlepszych rozwiązań otrzymanych w 15 eksperymentach

Beta	0.6	0.7	0.8	0.9
Cce	30.333	30.066	30.467	30.600

Z kolei w tablicy 3 podano wyniki obliczeń uzyskane podczas wszystkich 4×15 eksperymentów, przy czym każda pozycja tablicy zawiera wartość celu C*max* najlepszego rozwiązania w zbiorze *POPULACJA* oraz numer iteracji, podczas której rozwiązanie to zostało otrzymane.

Tablica 3

Wartość funkcji celu najlepszych rozwiązań otrzymanych podczas 4x15 eksperymentów

Beta:	0.6		0.7		0.8		0.9	
Exp1	33	6679	33	7070	33	8224	33	9388
Exp2	32	6178	31	8449	32	8117	32	9648
Exp3	30	6141	30	7019	31	8312	25	9381
Exp4	29	7025	29	8136	29	8017	31	9337
Exp5	30	6131	30	7014	31	9081	32	9041
Exp6	34	6157	31	9652	34	8273	34	9136
Exp7	29	6159	31	7533	28	8506	29	9714
Exp8	32	6341	26	8376	26	8997	28	9178
Exp9	29	6093	25	7979	26	8060	26	9168
Exp10	28	6004	28	7005	30	8226	33	9057
Exp11	33	6551	33	7733	33	8222	33	9331
Exp12	32	6433	31	9521	33	9904	30	9011
Exp13	27	8389	31	7274	31	8097	31	9037
Exp14	30	6121	30	7009	28	9917	30	9085
Exp15	27	7792	32	7004	32	8119	32	9304

Względna efektywność procesu optymalizacji, przedstawiona na rys. 1 dla 4×15 eksperymentów, została oceniona za pomocą wzoru

$$E = \frac{f_{max} - f_j}{f_{max} - f_{min}} \cdot \frac{L_{max} - L_j}{L_{max} - L_{min}}$$

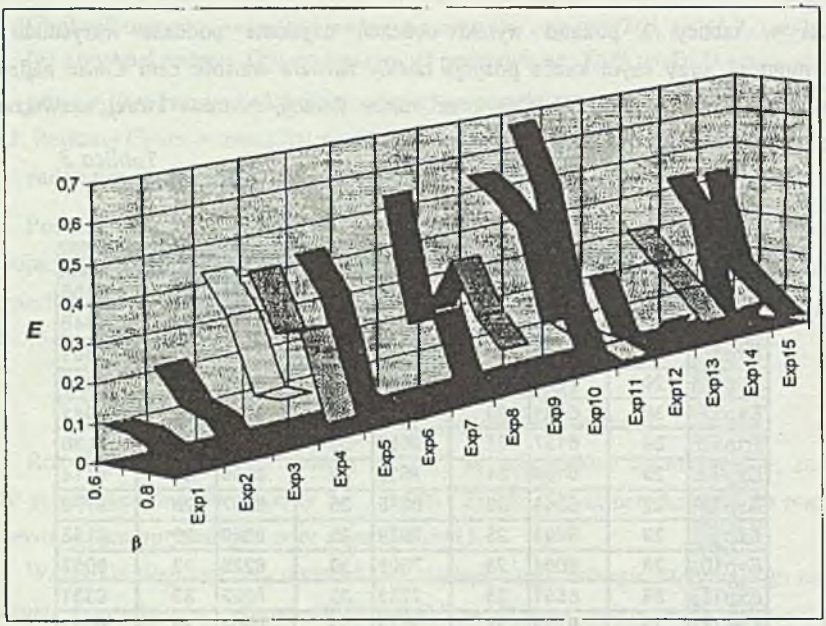
gdzie:

f_{max}, L_{max} - wartość funkcji celu najgorszego rozwiązania w zbiorze *POPULACJA* uzyskanego we wszystkich eksperymentach i numer iteracji, podczas której rozwiązanie to zostało obliczone;

f_{min}, L_{min} - wartość funkcji celu najlepszego rozwiązania w zbiorze *POPULACJA* uzyskanego we wszystkich eksperymentach i numer iteracji, podczas której rozwiązanie to zostało obliczone;

f_j, L_j - wartość funkcji celu najlepszego rozwiązania w zbiorze *POPULACJA* uzyskanego w eksperymencie *j* i numer iteracji, podczas której rozwiązanie to zostało obliczone.

Przedstawione wyniki obliczeń ilustrują tylko proces optymalizacji i nie mogą być podstawą do formułowania ogólnych wniosków.



Rys. 1. Względna efektywność procesu optymalizacji wykonanych eksperymentów

Fig. 1. Relative efficiency of the optimization process of the realized experiments

LITERATURA

[1] Błażewicz J.: Selected topics in scheduling theory. Annals of Discrete Mathematics, vol.31, 1-60, (1987), Elsevier Science Publishers B.V, North-Holland.

[2] Chmiel W., Wala K.: Effective approximation algorithm for the permutational optimization problems. Zeszytu Naukowe AGH, Automatyka, tom 64, 261-275, Kraków 1993.

[3] Goldberg D.E.: Genetic algorithms in search, optimization and machine learning. Addison-Wesley Pub. Company, Inc.,New York 1989.

[4] Holland J.: Adaptation in natural and artificial systems. Univ. of Michigan Press, An *Arbor, MI*, 1975.

[5] Kadłuczka P., Wala K.: New genetic search algorithm for generalized graph partitioning problem. Materiały II Międzynarodowego Sympozjum "Metody i Modele w Automatyce i Robotyce", Międzyzdroje'95 (w druku).

[6] Michalewicz Z.: Genetic algorithms + data structures = evolution programs. Springer-Verlag, Berlin 1992.

[7] Wala K., Chmiel W.: Local search and genetic heuristics for the permutational optimization problems. Proc. of 7th International Symposium on System-Modelling-Control, vol.2, 252-257, Zakopane, May 1993.

[8] Chmiel W., Wala K.: Investigations of crossover genetic operators for permutational optimization problems. Materiały XI Międzynarodowego Sympozjum Zastosowań Teorii Systemów, Zakopane'95 (w druku).

[9] Wala K., Gądek-Madeja H.: A new genetic algorithm for multilevel generalized assignment problem. Materiały XI Międzynarodowego Sympozjum Zastosowań Teorii Systemów, Zakopane'95 (w druku).

[10] 10.Whitley D., Genitor A.: A different genetic algorithm. Proc. of Rocy Mountain Conf. on Artifical Intelligence, Denver 1988.

Recenzent: Dr inż. Jerzy Mościński

Wpłynęło do Redakcji do 30.06.1995 r.

Abstract

The paper presents the results of a computer (numerical) investigation of the optimization process realized by genetic algorithm called *GA'94* for the example of *NP-hard* parallel unrelated machine scheduling problem. The investigation has been realized by means of experimental software package which is specially prepared for IBM PC compatible computers. The algorithm exploits five genetic operators where three of them are crossover operators (*PMX, OX, COMPX*), one is mutation operator and one is local search procedure. We present a collection of numerical result for R5//Cmax, $n = 25$, test task which characterizes relations of the optimization process to the proposed genetic search scheme. The search BREADTH-DEPTH scheme, realises original genetic search process consisting of two stages. We assume that in the first stage of the optimization process some number of the best population solutions, TAB_POCZ top-solutions, are tabu and can not be chosen as parents. In this case the genetic search process can be performed in the whole search space: offspring of the "super individual" of the population can not prevent other individuals of the population from contributing offspring in the next generations. In the second stage of the optimization process some number of worst population solutions, TAB_KON lower-solutions, are tabu and cannot be chosen as parents. In this stage of the optimization process the genetic search is enhanced to the solutions space region connected which the best (super) solutions for local tuning.