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

Konrad WALA
Akademia Górniczo-Hutnicza

DWUSTADIALNY ALGORYTM APROKSYMACYJNY HARMONOGRAMOWANIA ZAGADNIEŃ Z NIEZALEŻNYMI MASZYNAMI RÓWNOLEGŁYMI*

Streszczenie. W pracy przedstawiono wyniki badań komputerowych procesu optymalizacji realizowanego za pomocą dwustadialnego algorytmu aproksymacyjnego zaprojektowanego dla *NP-trudnych* zagadnień permutacyjnych. Badany algorytm jest hybrydą ulepszonego algorytmu genetycznego, procedury optymalizacji lokalnej oraz 'mechanizmu tabu'. Zagadnieniem testowym jest zagadnienie harmonogramowania z niezależnymi maszynami równoległymi i funkcją celu L_{max}.

TWO-STAGE APPROXIMATE ALGORITHM FOR SCHEDULING PROBLEMS ON UNRELATED PARALLEL MACHINES

Summary. The paper presents the computer results of optinization process investigation realized by two-stage approximate algorithm for NP-hard permutation problems. Investigated algorithm is a hybrid of an improved genetic algorithm, local optimization procedure and 'tabu mechanism'. The test problem is the nonpreemtive scheduling problem of independent tasks on unrealated machines.

1. Wprowadzenie

Zagadnienia harmonogramowania procesów produkcyjnych i obliczeniowych na maszynach/procesorach równoległych należą w przeważającej liczbie przypadków do zagadnień *NP-trudnych*; nawet prosty dwumaszynowy problem P2 || C_{max} jest *NP-trudny*. Algorytmów wielomianowych można poszukiwać wyłącznie w szczególnych przypadkach równych czasów wykonywania operacji, tj. $p_j = 1$ dla każdego $j = 1, 2, \ldots, n$, gdzie n - liczba zadań zagadnienia, lub gdy ograniczymy się do funkcji celu C_{sum} (por. [1]).

Zagadnienia harmonogramowania na maszynach równoległych z funkcją celu $L_{max} = \max \{ L_j : j = 1, 2, \ldots, n \}$, gdzie $L_j = c_j - d_j$, c_j jest terminem zakończenia zadania j, a d_j jest terminem planowanym, minimalizującą maksymalne opóźnienie wykonania zadań, są ważne w systemach uwarunkowanych czasowo, a szczególnie ważne w przypadkach, gdy zadane

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

terminy d_j wykonania zadań są liniami krytycznymi, których nie wolno przekroczyć, tj. w systemach typu 'hard-real time'. Spełniający ten warunek harmonogram istnieje tylko wtedy, gdy minimalna wartość funkcji celu L_{max} jest mniejsza lub równa zero: $L^*_{max} \leq 0$. Stosowane zwykle w takich przypadkach algorytmy przybliżone EDD (ang. earliest due date) typu szeregowania listowego (ang. list scheduling), o złożoności obliczeniowej O (n log n), przydzielają każdorazowo zadanie o najmniejszym terminie d_j do aktualnie wolnej maszyny lub do maszyny, która wykona to zadanie w terminie najwcześniejszym.

Harmonogramy obliczane, wprawdzie przy minimalnym nakładzie czasu, za pomocą algorytmów EDD są w wielu przypadkach niezadowalające, szczególnie w przypadku zagadnień $R \mid\mid L_{max}$ z maszynami niezależnymi.

2. Proces poszukiwania rozwiązań

Poszukując efektywnego algorytmu przybliżonego dla *NP-trudnych* zagadnień permutacyjnych, rozwiązywanych na komputerach zgodnych z IBM PC, opracowaliśmy oryginalny i efektywny proces genetycznego poszukiwania, który został zimplementowany w algorytmie BREADTH-DEPTH_GA. Algorytm BREADTH-DEPTH_GA jest hybrydą algorytmu genetycznego, procedury optymalizacji lokalnej (w języku angielskim spotykamy dwie nazwy tej procedury: local optimization procedure, hill climbing) oraz mechanizmu tabu', przy czym zastosowana ulepszona wersja algorytmu genetycznego GA'94 [5] należy do klasy Steady State GAs, gdzie $r \in \{1, 2\}$ (por.[4]), natomiast procedurę optymalizacji lokalnej wprowadzono do algorytmu genetycznego w postaci specjalnego operatora genetycznego nazwanego operatorem optymalizacji lokalnej. Celem mechanizmu tabu w algorytmie jest: (1) zapobieganie przedwczesnej zbieżności procesu poszukiwania rozwiązań do optimum lokalnego, (2) zmniejszenie nakładu obliczeń na proces poszukiwania, istotny zwłaszcza w tym stadium pracy algorytmu, kiedy aktywowana jest procedura optymalizacji lokalnej.

Proces zmiany populacji przez algorytm GA'94, tj. proces realizacji jednej iteracji algorytmu genetycznego, charakteryzują następujące cechy:

1. Wybierany jest losowo, z zadanym prawdopodobieństwem, **tylko jeden** operator genetyczny.
2. Jeżeli został wybrany operator unarny, to ze zbioru POPULACJA jest kopiowane (wybór losowy - rozkład równomierny) jedno rozwiązanie, zwane rodzicem, i generowane jest

jedno nowe rozwiązanie-potomek. Jeżeli został wybrany operator binarny (operator krzyżowania) to ze zbioru POPULACJA kopiowane są dwa rozwiązania-rodzice i generowane są dwa nowe rozwiązania-potomki; stąd $r \in \{1, 2\}$.

3. Nowo wygenerowane rozwiązanie (potomek) jest umieszczane w zbiorze POPULACJA **tylko wtedy**, kiedy zastępuje ono w zbiorze rozwiązanie gorsze.

Zaletą takiej wersji procesu genetycznego poszukiwania jest znaczna oszczędność nakładu czasu obliczeń na proces genetycznego poszukiwania. Podejście to posiada także dość istotną wadę, którą posiadają także i inne wersje algorytmów genetycznych. Zauważmy, że ewolucja zbioru POPULACJA za pomocą algorytmu GA'94 daje szansę najlepszym rozwiązaniom zbioru zostania rodzicami wiele razy częściej niż pozostałym rozwiązaniom i w ten sposób rozwiązania te zostają *'superosobnikami'*. Stała liczność elementów zbioru POPULACJA oraz fakt, że superosobnik umieszcza dużą liczbę swoich potomków w zbiorze ogranicza liczbę potomków gorszych rozwiązań umieszczanych w zbiorze POPULACJA. Tak więc w następnych generacjach superosobniki, eliminując różny od nich *'materiał genetyczny'*, zawężają przeszukiwany obszar zbioru rozwiązań, co jest często przyczyną przedwczesnej zbieżności procesu genetycznego poszukiwania do optimum lokalnego.

W opracowanym algorytmie aproksymacyjnym BREADTH-DEPTH_GA, przeznaczonym do rozwiązywania *NP-trudnych* zagadnień optymalizacji dyskretnej na komputerach zgodnych z IBM PC przyjęto, że zbiór rozwiązań POPULACJA , o stałej liczbie M rozwiązań podczas całego procesu optymalizaji (gdzie $M \in [50, 200]$), jest liniowo uporządkowany od rozwiązań najlepszych do rozwiązań najgorszych, tj. pierwsze rozwiązanie zbioru jest rozwiązaniem najlepszym, ... , ostatnie rozwiązanie (rozwiązanie nr M) jest rozwiązaniem najgorszym w zbiorze POPULACJA. W tej pracy przyjęto, że w zagadnieniach permutacyjnych funkcją oceny rozwiązań zbioru POPULACJA jest funkcja celu zagadnienia, a proces optymalizacji polega na poszukiwaniu rozwiązań minimalizujących jej wartość.

W celu zapobiegania przedwczesnej zbieżności procesu poszukiwania do optimum lokalnego i maksymalnej poprawy efektywności algorytmu aproksymacyjnego zastosowano w nim dwustadialny schemat procesu genetycznego poszukiwania, nazwany „BREADTH-DEAPTH". Pierwsze stadium procesu poszukiwania 'dobrych rozwiązań', stadium BREADTH, jest realizowane podczas βL iteracji, natomiast drugie stadium, stadium DEPTH, podczas końcowych $(1-\beta)L$ iteracji, gdzie $0{,}5 \le \beta \le 0{,}8$ i L jest liczbą rozwiązań-potomków wygenerowanych w czasie całego procesu genetycznego poszukiwania.

Zakłada się, że w pierwszym stadium optymalizacji powinien być testowany możliwie obszerny zbiór rozwiązań zagadnienia (stąd: BREADTH): w zbiorze populacja powinien znajdować się możliwie różnorodny 'materiał genetyczny'. W tym etapie procesu optymalizacji 'potomstwo superosobników' (tj. najlepszych rozwiązań) nie powinno wypierać zbyt szybko potomstwa pozostałych rozwiązań zbioru POPULACJA. W tym celu wprowadzono parametr TAB_POCZ (od tabu dla początkowych, tj. najlepszych rozwiązań zbioru POPULACJA; proponuje się: TAB_POCZ $\in \{1, 2, \ldots, \alpha\}$, gdzie $\alpha = 0{,}1M$) i zakłada się, że TAB_POCZ pierwszych (tj. najlepszych) rozwiązań liniowo uporządkowanego zbioru POPULACJA jest zabronionych (jest tabu) - rozwiązania te nie mogą zostać wybrane i skopiowane jako rodzice w procesie generacji rozwiązań-potomków.

Zauważmy jeszcze jedną zaletę takiego mechanizmu. Do początkowej populacji możemy wprowadzić *'dobre'* rozwiązania wyznaczone za pomocą innych heurystyk. Jeżeli liczba tych rozwiązań nie przekroczy wartości TAB_POCZ, to rozwiązania te w początkowej fazie stadium BREADTH będą nieaktywne, a więc nie spowodują przedwczesnej zbieżności procesu poszukiwania do optimum lokalnego. Dopiero w chwili, gdy w populacji pojawią się dostatecznie dobre rozwiązania - mogące konkurować z heurystycznymi, wprowadzone rozwiązania heurystyczne zostają przesunięte *'ze strefy tabu populacji do strefy aktywnej'*, i wtedy informacja zawarta w tych rozwiązaniach będzie wykorzystana w procesie generacji rozwiązań-potomków, tj. dopiero wtedy w procesie przeszukiwania przestrzeni rozwiązań obszary związane z rozwiązaniami heurystycznymi będą *'konkurowały z innymi na równych prawach'*.

Ponadto przyjęto, że genetyczny operator mutacji jest aktywny tylko w stadium BREADTH (prawdopodobieństwo wylosowania operatora mutacji jest większe od zera: $p_{mut} = \gamma$, $\gamma > 0$), a operator optymalizacji lokalnej jest nieaktywny ($p_{opt.lok} = 0{,}0$).

W drugim stadium procesu optymalizacji, stadium DEPTH, jest realizowany proces przetwarzania tylko najlepszych rozwiązań zbioru POPULACJA (stąd: DEPTH). W tym celu wprowadzono parametr TAB_KON (od tabu dla końcowych; proponuje się: TAB_KON$\in \{a_1, a_1 + 1, \ldots, a_2\}$, $a_1 = 0{,}5M$, $a_2 = 0{,}8M$) i zakłada się, że TAB_KON ostatnich (tj. najgorszych) rozwiązań zbioru POPULACJA jest zabronionych w stadium DEPTH - rozwiązania te nie mogą zostać wybrane i skopiowane jako rodzice w procesie generacji rozwiązań-potomków.

Przyjęto, że w stadium DEPTH rolę operatora mutacji przejmuje operator optymalizacji lokalnej (tj. $p_{mut} = 0{,}0$; $p_{opt.lok} = \gamma$), który poprawia rozwiązania-rodzice poprzez pełny

przegląd otoczeń (sąsiedztwa) poprawianych rozwiązań. Zauważmy, że pełny przegląd otoczeń rozwiązań-rodziców wymaga znacznych nakładów czasu na proces szukania, stąd operator optymalizacji lokalnej jest stosowany tylko w drugim stadium procesu optymalizacji do najlepszych rozwiązań znacznie ograniczonego zbioru POPULACJA.

Operatory genetyczne, w tym operatory krzyżowania, posiadają podczas całego procesu optymalizacji stałe wartości prawdopodobieństw losowania, co ułatwia konstrukcję i eksploatację algorytmu, z wyjątkiem operatora mutacji i optymalizacji lokalnej: (1) w pierwszym stadium procesu optymalizacji jest aktywny operator mutacji, a nieaktywny operator optymalizacji lokalnej, (2) w drugim stadium procesu role tych operatorów się zamieniają - operator optymalizacji lokalnej jest aktywny, natomiast mutacji jest nieaktywny.

Zauważmy także, że wprowadzenie do algorytmu parametrów typu 'TAB' nie zwiększa nakładu obliczeń na proces poszukiwania, natomiast parametr TAB_KON znacznie ogranicza aktywny zbiór POPULACJA w stadium DEPTH, koncentrując obliczenia tylko na rozwiązaniach najlepszych, co daje dodatkowe oszczędności czasu obliczeń.

3. Algorytm przybliżony

W przypadku formalizacji rozwiązania zagadnienia w postaci permutacji informacją zawartą w rozwiązaniu jest tylko *uszeregowanie* elementów permutacji. W takim przypadku w algorytmie genetycznym można zastosować tylko operatory zmieniające kolejność elementów rozwiązań. W badanym algorytmie przybliżonym zastosowano dwa unarne operatory genetyczne: (1) operator mutacji, (2) operator optymalizacji lokalnej. Przyjęto, że operator mutacji przestawia dwa elementy permutacji, które znajdują się na pozycjach określonych na drodze losowania. Natomiast operator optymalizacji lokalnej jest standardową procedurą optymalizacji lokalnej realizującą zupełny przegląd otoczenia (ang. neighbourhood), w której otoczenie permutacji $\Pi = (\Pi(1), \ldots, \Pi(i-1), \Pi(i), \ldots, \Pi(j-1), \Pi(j), \ldots, \Pi(n))$ jest także określone przez zamianę dwóch elementów permutacji $N(\Pi) = \{\Pi' : \Pi' = (\Pi(1), \ldots, \Pi(i-1), \Pi(j), \ldots, \Pi(j-1), \Pi(i), \ldots, \Pi(n))\}$ (por.[5]).

Dla rozwiązań, w których ważna jest kolejność elementów, operatory krzyżowania (peratory binarne) są bardziej skomplikowane, ponieważ klasyczne (ślepe) krzyżowanie może prowadzić do rozwiązań niedopuszczalnych. W badanym algorytmie zastosowano dwa następujące operatory krzyżowania: (1) PMX (ang.partially matched crossover), (2) OX (ang. order crossover); są one opisane na przykład w monografiach [3], [4].

Algorytm **BREADTH-DEAPTH_GA** dla zagadnienia $R||L_{max}$

Wejście: tablice czasów $[p_{ij}]_{m*n}$, $[d_j]_n$

Krok 1. Utworzenie początkowego zbioru POPULACJA.

W sposób losowy wygeneruj M permutacji zbioru liczb naturalnych {1, 2, ... ,n}. Dla każdej permutacji Π, za pomocą *procedury* L_{max} (por.punkt 4) oblicz wartość funkcji celu $L_{max}(\Pi)$. Utwórz liniowo uporządkowany zbiór POPULACJA porządkując wygenerowane permutacje zgodnie z niemalejącymi wartościami L_{max}.

Krok 2. Wybór operatora genetycznego.

Jeżeli liczba wygenerowanych potomków jest mniejsza lub równa βL (stadium BREADTH procesu optymalizacji), to spośród operatorów genetycznych mutacji PMX i OX wybierz jeden, przy czym prawdopodobieństwa losowania operatorów, odpowiednio, są równe: γ, p_{PMX}, $p_{OX} = 1 - \gamma - p_{PMX}$.

Jeżeli liczba wygenerowanych potomków jest większa od βL (stadium DEAPTH), to spośród operatorów genetycznych optymalizacji lokalnej, PMX i OX wybierz jeden, przy czym prawdopodobieństwa losowania operatorów, odpowiednio, są równe: γ, p_{PMX}, p_{OX}.

Krok 3. Wybór rodziców/rodzica.

Jeżeli liczba wygenerowanych potomków jest mniejsza lub równa βL, to dla wybranego operatora genetycznego wybierz i skopiuj, zgodnie z rozkładem równomiernym, spośród rozwiązań od numeru TAB_POCZ + 1 do M zbioru POPULACJA jedno (w przypadku operatora unarnego) lub dwa (w przypadku operatora krzyżowania) rozwiązania i nazwij je rodzicami. Jeżeli liczba wygenerowanych potomków jest większa od βL, to dla wybranego operatora wybierz i skopiuj, także zgodnie z rozkładem równomiernym, spośród rozwiązań od numeru 1 do TAB_KON jedno lub dwa rozwiązania (w zależności od wybranego typu operatora) i nazwij je rodzicami.

Krok 4. Generowanie potomków.

Za pomocą wylosowanego operatora genetycznego dokonaj modyfikacji permutacji-rodzica/permutacji-rodziców i wyznacz w ten sposób permutację-potomek/permutacje-potomki.

Krok 5. Poprawa rozwiązań zbioru POPULACJA.

Dla każdego potomka oblicz, za pomocą *procedury* L_{max}, wartość funkcji celu. Jeżeli wartość ta jest lepsza (tj.mniejsza) od wartości funkcji celu najgorszego (tj. ostatniego. w zbiorze) rozwiązania zbioru POPULACJA, to umieść takiego potomka w zbiorze POPULACJA usuwając zarazem z tego zbioru rozwiązanie ostatnie.

Krok 6. Warunek STOPU'u.

Jeżeli wygenerowano zadaną liczbę L potomków, to STOP: najlepszym, znalezionym w dwustadialnym procesie optymalizacji, rozwiązaniem jest pierwsze rozwiązanie zbioru POPULACJA. W przeciwnym przypadku idź do kroku 2.

Parametrami algorytmu są wielkości: M, L, β, TAB_POCZ, TAB_KON, γ, p_{PMX}, p_{OX}.

4. Harmonogramowanie zadań na maszynach niezależnych

Zagadnienie $R||L_{max}$ harmonogramowania *n* niezależnych i niepodzielnych zadań na *m* niezależnych maszynach, którego celem jest minimalizacja maksymalnego opóźnienia terminów zakończenia wszystkich zadań L_{max}, należy do zagadnień *NP-trudnych* (wersja decyzyjna zagadnienia $P2||L_{max}$ jest zagadnieniem *NP-zupełnym*, por.[2]). Należy podkreślić, że zagadnienia harmonogramowania zadań na maszynach niezależnych należą do najtrudniejszych zagadnień harmonogramowania. Jeżeli przyjmiemy, że permutacja numerów zadań $\Pi = (\Pi(1), \Pi(2), \ldots, \Pi(n))$ określa kolejność przydziału zadań do najwcześniej wolnej maszyny, to wartość L_{max} wyznacza następująca procedura.

Procedura L_{max}

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

Krok 1. Dla $i = 1, 2, \ldots, m$ podstaw $T(i) = 0$.

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

(i) określ numer najwcześniej wolnej maszyny $k = \arg\min \{ T(i): i = 1, 2, \ldots, n \}$;

(ii) przydziel zadanie P(j) do maszyny k podstawiając $T(k) := T(k) + p_{k\,P(j)}$

oraz $c_{P(j)} = T(k)$.

Krok 3. Oblicz maksymalne opóźnienie terminów zakończenia zadań $L_{max} = \max \{ c_j - d_j : j = 1, 2, \ldots, n \}$.

W badaniach komputerowych zastosowano generator testów, nazwany *generatorem A*, dla zagadnienia $R||L_{max}$, w którym użyto generatora losowych liczb całkowitych o równomiernym rozkładzie RANDOM_UNIFORM. Wartości czasów wykonania zadań p_{ij} ($i = 1, 2, \ldots, m$; $n = 1, 2, \ldots, n$) każdego testu, dla ustalonych parametrów m, n, A_1, A_2, są obliczane za pomocą procedury:

p_{ij} := RANDOM_UNIFORM $[A_1, A_2]$, $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$

gdzie A_1, A_2 są liczbami całkowitymi określającymi przedział wartości czasów wykonania zadań. Natomiast terminy d_j są obliczane w następujący sposób. Po wygenerowaniu tablicy czasów wykonywania zadań $[\ p_{ij}\ ]_{mxn}$ losowana jest permutacja n zadań: $\Pi^* = (\Pi^*(1), \Pi^*(2), \ldots, \Pi^*(n))$. Następnie, za pomocą *procedury* L_{max}, dla uszeregowania zadań Π^*, jest obliczany harmonogram, a więc i czasy zakończenia c^*_j wszystkich zadań. Przyjęto, że terminy $d_j = c_j$, tak więc wśród $n!$ permutacji istnieje co najmniej jedna permutacja Π^*, która zapewnia realizację wszystkich zadań przed liniami krytycznymi, tj. o wartości funkcji celu $L_{max}(\Pi^*) = 0$. Jest to bardzo korzystna własność stosowanych w badaniach komputerowych testów, ponieważ dla każdego z nich istnieje rozwiązanie z wartością funkcji celu $L_{max} = 0$.

W tabeli 1 przedstawiono wyniki eksperymentów komputerowych dla czterech zadań testowych określonych przez generator testów dla następujących parametrów: liczba zadań n = 30, liczba maszyn m = 5, czasy wykonania zadań $p_{ij} \in [1, 100]$. Podczas realizacji eksperymentów niektóre parametry algorytmu BREADTH-DEPTH_GA miały tę samą wartość, mianowicie liczba iteracji algorytmu była nieduża: L = 5000, rozmiar populacji M = 100, TAB_KON = 50 (połowa liczby rozwiązań w populacji), prawdopodobieństwa losowania operatorów genetycznych miały następujące wartości: g = 0.1, p_{PMX} = 0.5, p_{OX} = 0.4.

Tabela 1

Nr eksp.	f_{pocz}	b = 0.4				b = 0.6				b = 0.8			
		0	3	6	9	0	3	6	9	0	3	6	9
1	138	9	11	32	53	50	41	42	9	34	34	67	24
		13	1	23	52	41	26	37	31	42	36	44	50
		26	23	40	23	10	2	29	19	31	47	48	18
2	131	-4	1	-4	10	-2	-12	-12	-12	-12	-9	-12	1
		6	-4	-6	8	13	0	0	-6	-9	-4	13	-13
		12	3	10	1	2	-12	-4	0	-6	5	-10	-12
3	138	42	26	33	22	7	17	25	48	50	18	16	31
		19	15	42	45	21	12	10	-10	16	22	29	2
		16	25	32	56	38	23	18	15	42	46	19	44
4	105	39	27	17	43	61	34	41	23	24	26	22	36
		56	30	10	22	41	39	29	47	37	28	47	33
		54	52	47	37	65	27	42	18	28	24	19	35

Pierwsza kolumna tabeli 1 podaje numer eksperymentu, natomiast druga wartość funkcji celu L_{max} najlepszego rozwiązania w populacji początkowej (wygenerowanej losowo

w kroku 1 algorytmu): f_{pocz}. W każdym eksperymencie przeprowadzono 36 procesów optymalizacji, które rozpoczynały się od tej samej populacji początkowej. Dla konkretnych wartości parametru β (β= 0.4, 0.6, 0.8) podanych w pierwszym wierszu tabeli, oraz konkretnych wartości parametru TAB_POCZ (TAB_POCZ = 0, 3, 6, 9) - drugi wiersz tabeli, przeprowadzono po trzy procesy optymalizacji. W wierszach od 4 do 15 tabeli podano wartości funkcji celu najlepszych rozwiązań znajdujących się w zbiorze POPULACJA po L = 5000 iteracjach procesu poszukiwania.

Przeprowadzone eksperymenty wskazują na dużą skuteczność procesu poszukiwania oraz na celowość wprowadzenia parametru TAB_POCZ. Algorytmy genetyczne są algorytmami stochastycznymi. Zauważmy (tabela 1), że algorytm dla tych samych parametrów i tej samej populacji początkowej wyznacza różne rozwiązania lokalnie optymalne. Stąd wydaje się, że interesującą organizacją procesu obliczeń za pomocą proponowanego algorytmu jest następująca: (1) ustal niedużą liczbę iteracji procesu optymalizacji, np. L = 5000, (2) jeżeli obliczone rozwiązanie jest niesatysfakcjonujące, to powtórz instrukcję (1).

LITERATURA

1. Błażewicz J.: Selected topics in scheduling theory. Annals of Discrete Mathematics, Elsevier Science Publishers B.V., vol.31, 1987, pp. 1 - 60.
2. Lenstra J.K., Rinnooy Kan A.H.G., Brucker P.: Complexity of machine scheduling problems. Annals of Discrete Mathematics, Elsevier Science Publishers B.V., vol.1, 1997.
3. Goldberg D.E.: Genetic algorithms in search, optimization and machine learning. Addison-Wesley Pub.Company,Inc., N.Y., 1989 .
4. Michalewicz Z.: Genetic algorithms + data structures = evolution programs. Springer-Verlag, Berlin 1992.
5. Wala K., Chmiel W.: Nowy schemat procesu genetycznego poszukiwania na przykładzie zagadnień harmonogramowania z maszynami równoległymi. Zeszyty Naukowe Politechniki Śląskiej, Automatyka z.116, 1995, ss.67-77.

Recenzent: Dr hab. inż. Jan Kałuski, prof. Pol.Śl.
Wpłynęło do Redakcji do 30.06.1996 r.

Abstract

The paper presents the computer results of optimization process investigation realized by two-stage approximate algorithm for NP-hard permutation problems. Investigated algoithm is a hybrid of an improved genetic algorithm, local optimization procedure and 'tabu mechanism'. It realizes the BREADTH_DEPTH genetic search process, where in course of

one iteration initially one genetic operator is randomly chosen and then r, $r \in \{1,2\}$, solutions are selected from the population and processed: one solution if unary operator is chosen and two in case of crossover operator; thus the algorithm belongs to the class of Steady State GAs. At the first stage of genetic search process called BREADTH some number of best population sloutions, TOPTAB top-solutions,are tabu and can not be chosen as parents. It is most likely that in this case the genetic search process can be performed in the whole search space. In the secend stage of genetic search process caled DEPTH the genetic search process is enhanced to the space regions connected with the best population solutions for local tuning. Thus, in this stage, the mutation operator is replaced by local optimization procedure and some number of worst population solutions, ENDTAB end-solutions, are tabu and cannot be chosen as parents.