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

Joanna JÓZEFOWSKA, Marek MIKA, Rafał RÓŻYCKI,
Grzegorz WALIGÓRA i Jan WĘGLARZ
Politechnika Poznańska

DYSKRETNO-CIĄGŁE PROBLEMY SZEREGOWANIA – EKSPERYMENT OBLICZENIOWY

Streszczenie. W pracy rozważa się problem szeregowania zadań niepodzielnych na identycznych maszynach równoległych z uwzględnieniem dodatkowego zasobu, ciągłego i odnawialnego, w celu minimalizacji długości uszeregowania. Prędkość wykonywania zadania w chwili t zależy od ilości zasobu ciągłego przydzielonego zadaniu w tej chwili. Dokonano porównania, na podstawie eksperymentu obliczeniowego, trzech metaheurystyk odpowiednio dostosowanych do powyższego problemu: przeszukiwania tabu, symulowanego wyżarzania i algorytmu genetycznego.

DISCRETE-CONTINUOUS SCHEDULING PROBLEMS - COMPUTATIONAL EXPERIMENT

Summary. In this paper a problem of scheduling nonpreemptible jobs on identical parallel machines with an additional continuous resource to minimize the schedule length is considered. The processing rate of a job at time t depends on the amount of the continuous resource allotted to this job at time t. On the basis of a computational experiment performances of three metaheuristics developed for the above problem have been compared: tabu search, simulated annealing and genetic algorithm.

1. Sformułowanie problemu

W pracy rozważa się problem szeregowania zadań żądających równocześnie dwóch kategorii ograniczonych zasobów odnawialnych. Zasób pierwszej kategorii (dyskretny) stanowią równoległe identyczne maszyny. Zasób drugiej kategorii (ciągły) można przydzielać do zadań w dowolnej ilości z przedziału [0,1] (może to być np. moc). Zadania są niepodzielne, niezależne i gotowe do wykonania w chwili zero. Chwilowa prędkość wykonywania zadania zależy od ilości zasobu ciągłego przydzielonego temu zadaniu w danej chwili i jest opisana następującą zależnością:

$$\dot{x}(t) = \frac{dx_i(t)}{dt} = f_i[u_i(t)], \quad x_i(0) = 0, \quad x_i(C_i) = \bar{x}_i$$

gdzie: $x_i(t)$ jest stanem zadania i w chwili t,

$u_i(t)$ jest ilością zasobu ciągłego przydzielonego do zadania i w chwili t,

f_i jest ciągłą funkcją niemalejącą, $f_i(0)=0$,

C_i jest (nieznaną a priori) chwilą zakończenia zadania i,

$\bar{x}_i$ jest stanem końcowym (rozmiarem) zadania i.

Założymy, że $0 \le u_i(t) \le 1$, $\sum_{i=1}^{n} u_i(t) = 1$.

Rozwiązanie problemu polega na znalezieniu sekwencji[1] zadań na maszynach i jednocześnie przydziału zasobu ciągłego do poszczególnych zadań w tej sekwencji, tak aby długość otrzymanego uszeregowania była minimalna. Oczywiście w ogólności problem ten jest NP-trudny, ponieważ jest co najmniej tak trudny jak problem szeregowania zadań niepodzielnych na maszynach równoległych bez dodatkowych zasobów, o którym wiadomo, że jest NP-trudny [6].

Zauważmy, że powyższy problem może być zdekomponowany na dwa ściśle powiązane podproblemy:

(i) znalezienie dopuszczalnej sekwencji zadań na maszynach,

(ii) przydział zasobu ciągłego do zadań w danej sekwencji.

Można wykazać [3], że dla wklęsłych funkcji f_i, i=1,2,...,n, optymalne jest uszeregowanie, w którym w każdej chwili wykonywanych jest m zadań, gdzie m jest liczbą maszyn, a dopuszczalna sekwencja zadań składa się z n-m+1 kombinacji, takich że każde zadanie pojawia się w co najmniej jednej kombinacji oraz, jeżeli pojawia się w więcej niż jednej kombinacji, to są to kombinacje kolejne (niepodzielność zadań). Oznaczając tę sekwencję przez S mamy zatem $S=Z_1, Z_2,...,Z_{n-m+1}$, gdzie $|Z_k|=m$, k=1,2,...,n-m+1.

Dla danej sekwencji dopuszczalnej znalezienie optymalnego przydziału zasobu ciągłego do zadań (to jest takiego przydziału, w wyniku którego otrzymana długość uszeregowania dla tej sekwencji jest minimalna) polega na rozwiązaniu wypukłego problemu programowania matematycznego [3]. Zatem jednym ze sposobów uzyskania rozwiązania optymalnego byłoby znalezienie optymalnego przydziału zasobu ciągłego dla wszystkich dopuszczalnych sekwencji z pewnego zbioru, o którym wiadomo, że zawiera sekwencję optymalną, a następnie wybór najlepszego ze znalezionych rozwiązań. Niestety, w ogólności moc zbioru, o którym można udowodnić, że zawiera sekwencję optymalną, rośnie wykładniczo ze wzrostem liczby zadań. Uzasadnia to zastosowanie heurystyk, np. symulowanego wyżarzania (simulated annealing, SA), przeszukiwania tabu (tabu search, TS) i algorytmu genetycznego (genetic algorithm, GA). W naszym podejściu będziemy przeszukiwali przestrzeń wszystkich sekwencji dopuszczalnych, którą oznaczymy przez $\mathcal{S}$. Choć sekwencje te nie stanowią pełnych rozwiązań rozpatrywanych problemów szeregowania (niezbędny jest jeszcze przydział zasobu ciągłego), w opisie algorytmów przeszukiwania będziemy je także nazywali "rozwiązaniami" w celu uzyskania zgodności z ogólną postacią tych algorytmów. Wartością funkcji kosztu będzie długość uszeregowania dla danej sekwencji dopuszczalnej przy optymalnym dla niej rozdziale zasobu ciągłego. Ten ostatni będziemy uzyskiwali w drodze rozwiązania odpowiedniego

[1] Mamy tu na myśli sekwencję k-elementowych kombinacji zadań, k≤m.

problemu programowania wypukłego z wykorzystaniem specjalnie przystosowanego solvera CFSQP 2.3.

Wymienione wyżej heurystyki zostały dostosowane i zaimplementowane do rozwiązywania rozpatrywanej klasy problemów szeregowania, co omówiono w rozdziałach 2, 3 i 4. Rozdział 5 zawiera opis eksperymentu obliczeniowego, a wnioski i kierunki dalszych badań przedstawiono w rozdziale 6.

2. Algorytm symulowanego wyżarzania

2.1. Reprezentacja rozwiązania dopuszczalnego

Jak wspomniano w rozdziale 1, jako rozwiązanie dopuszczalne traktujemy sekwencję dopuszczalną, reprezentowaną przez uporządkowaną parę n-elementowych ciągów. Pierwszy z nich zawiera permutacje numerów zadań w kolejności ich wykonywania. Drugi określa, na której maszynie będzie wykonywane odpowiednie zadanie z ciągu pierwszego. Na przykład dla n=8 oraz m=3 para ciągów: {1,8,6,2,7,3,5,4}, {1,2,3,2,2,3,1,1} odpowiada sekwencji dopuszczalnej: S={1,8,6}, {1,2,6}, {1,7,6}, {1,7,3}, {5,7,3}, {4,7,3}.

2.2. Mechanizm generacji sąsiedztwa

Sąsiedztwo generowane jest na dwa sposoby. Można dokonać zmiany w ciągu zadań lub też w ciągu maszyn. Zmiana w ciągu zadań polega na wylosowaniu dwóch zadań i zamiany ich miejscami. Jeżeli w przytoczonym wyżej przykładzie wybierzemy zadania 8 oraz 3 i zamienimy je miejscami, wówczas otrzymamy następującą parę ciągów (zamienione zadania oznaczono pogrubioną czcionką): {1,**3**,6,2,7,**8**,5,4}, {1,2,3,2,2,3,1,1} oraz odpowiadającą jej sekwencję dopuszczalną: S={1,**3**,6}, {1,2,6}, {1,7,6}, {1,7,**8**}, {5,7,**8**}, {4,7,**8**}. Zamiana w ciągu maszyn polega na przeniesieniu jednego, losowo wybranego zadania na inną, również losowo wybraną maszynę. Jeśli np. w rozważanym przykładzie zadanie 7 przeniesiemy na maszynę 3, to otrzymamy następującą parę ciągów {1,8,6,2,7,3,5,4}, {1,2,3,2,**3**,3,1,1} i odpowiednio sekwencję: S={1,8,6}, {1,2,6}, {1,**2**,7}, {1,2,3}, {5,**2**,3}, {4,**2**,3}.

2.3. Początkowa wartość parametru kontrolnego

Początkowa wartość parametru kontrolnego T_0 (temperatura początkowa) obliczana jest według wzoru: $T_0 = \dfrac{\Delta\overline{C}}{\ln\left(c_0^{-1}\right)}$,

gdzie $\overline{\Delta C}$ oznacza średnią różnicę kosztów dla 25 próbnie wygenerowanych sąsiadów, a χ_0 oznacza początkową wartość współczynnika akceptacji określającego, jaki powinien być

procent zaakceptowanych przejść dla temperatury T_0. Wartości parametru kontrolnego maleją liniowo wg następującego wzoru: $T_{k+1}=0.95T_k$, dla $k = 0,1,2,\ldots$.

2.4. Długość łańcuchów Markowa

Długość łańcucha Markowa L_k określa, ile przejść będzie generowanych dla danej wartości parametru kontrolnego, a co za tym idzie - również moment, w którym należy dokonać zmiany wartości tego parametru według reguły opisanej w poprzednim punkcie. L_k jest określone w taki sposób, że dla każdej wartości T_k parametru kontrolnego powinna być zaakceptowana pewna liczba przejść η_{min}. Jednakże dla coraz mniejszych wartości T_k prawdopodobieństwo akceptacji maleje, co pociąga za sobą wydłużanie się łańcuchów Markowa. Aby się przed tym zabezpieczyć, ograniczamy z góry ich długość przez pewną stałą $\overline{L}$. Długość łańcuchów należy zatem do przedziału $[\eta_{min}, \overline{L}]$. W zastosowanym algorytmie przyjęto $\eta_{min}=3n$ oraz $\overline{L}=10n$.

3. Algorytm genetyczny

3.1. Reprezentacja rozwiązania dopuszczalnego

Każda sekwencja dopuszczalna jest tu reprezentowana przez pojedynczy chromosom. Zaproponowano reprezentację, w której pojedynczy chromosom składa się z pary wektorów: n-elementowego oraz (n-m)-elementowego. Pierwszy wektor („wektor zadań") zawiera n-elementową permutację zbioru zadań $\{1,2,\ldots,n\}$. Porządek zadań określa kolejność ich wykonania, tzn. aktualnie wolna maszyna jest zajmowana przez kolejne, nie rozpoczęte jeszcze zadanie. Drugi wektor („wektor maszyn") zawiera (m-n)-elementowy ciąg ze zbioru $\{1,2,\ldots,m\}$. Są to numery maszyn, zwalnianych w kolejnych kombinacjach Z_j, $j=1,2,\ldots,n-m$, sekwencji dopuszczalnej. Odwzorowanie takiej dwutablicowej struktury w sekwencję dopuszczalną pokażemy na przykładzie, w którym liczba maszyn m=3 i liczba zadań n=6. Niech chromosom X ma postać: $X=\{2,4,1,6,5,3\}$, $\{02,2,3\}$. Odpowiada mu następująca sekwencja dopuszczalna $S=\{2,4,1\}$, $\{2,6,1\}$, $\{2,5,1\}$, $\{2,5,3\}$. Przyjęcie takiej reprezentacji umożliwiło opracowanie kilku operatorów genetycznych przystosowanych do rozważanego problemu.

3.2. Populacja początkowa

W większości implementacji algorytmu genetycznego populacja początkowa tworzona jest losowo. Takie założenie przyjęto również dla algorytmu genetycznego użytego do rozwiązywania dyskretno-ciągłych problemów szeregowania. W naszym przypadku szczególną uwagę należało skoncentrować na zapewnieniu dopuszczalności generowanych

losowo chromosomów. Metoda użyta do generowania obu wektorów składowych chromosomu gwarantuje taką dopuszczalność, a dodatkowo również dużą różnorodność populacji początkowej.

3.3. Przystosowanie chromosomu

Miarą przystosowania chromosomu jest długość uszeregowania dla danej sekwencji dopuszczalnej przy optymalnym dla niej przydziale zasobu ciągłego. Należy zauważyć, że przystosowanie chromosomu (intuicyjnie jego „jakość") jest odwrotnie proporcjonalna do długości uszeregowania.

3.4. Operatory genetyczne

Operator selekcji. Główny wpływ na zbieżność algorytmu genetycznego ma operator selekcji. Do rozważanego problemu jako operator selekcji wybrano ranking liniowy. Nie wymaga ona żadnych technik skalowania (wymaganych np. w metodzie „koła ruletki") i daje szanse trafienia do następnej populacji nawet chromosomom najgorzej przystosowanym (w przeciwieństwie np. do metody turniejowej).

Operatory rekombinacji. Operatory te odpowiadają za sposób przeszukiwania przestrzeni rozwiązań. Można je podzielić na dwie grupy: operatory działające na pojedynczych chromosomach i operatory działające na parach chromosomów, tzw. rodzicach. Do pierwszej grupy zaliczamy mutację i przenumerowywanie, do drugiej dwa typy krzyżowania.

Mutacja. Przez mutację, w naszym przypadku, rozumiemy zmiany dokonywane na wektorze maszyn,polegające na tym, że na wylosowanej pozycji z tego wektora wstawia się, również wylosowany, ale różny od dotychczasowego, nowy numer maszyny. Chromosom X poddany mutacji na pozycji 2 wektora maszyn, mógłby wyglądać następująco: X'={2,4,1,6,5,3}, {2,1,3}. Jak widać, mutacja nie zmienia kolejności wykonania zadań na maszynach, lecz tylko ich przydział do maszyn.

Przenumerowywanie. Przenumerowywanie odpowiada za zmiany dokonywane w wektorze zadań pojedynczego chromosomu. Najpierw losowana jest liczba i, $2 \le i \le n$, zadań, które mają podlegać przenumerowywaniu. Następnie generowany jest i -elementowy ciąg nie powtarzających się, losowych numerów zadań. Przenumerowywanie polega na zastąpieniu w wektorze zadań zadania występującego w takim ciągu zadaniem następnym z tego ciągu. Zadanie ostatnie z ciągu jest zastępowane zadaniem pierwszym. Niech np. przenumerowywanie dotyczy i=3 zadań z chromosomu X i niech wylosowany ciąg trzech zadań ma postać {4,5,2}. Chromosom X' po przenumerowywaniu wyglądałby następująco: X'={4,5,1,6,2,3}, {2,2,3}.

Krzyżowanie typu I. Krzyżowania tego typu dokonuje się na wektorach maszyn dwóch chromosomów rodziców. Jest to zwykłe krzyżowanie dwupunktowe, nie wymagające kontroli

dopuszczalności otrzymanych chromosomów potomnych. Przykład krzyżowania chromosomów Y i Z (dla n=7 i m=2) zamieszczono poniżej.

Y={2,4,5,1,7,6,3}, {2,|1,2,2,|1} Z={4,5,2,7,3,1,6}, {1,|2,1,2,|2}

Y'={2,4,5,1,7,6,3}, {2,|2,1,2,|1} Z'={4,5,2,7,3,1,6}, {1,|1,2,2,|2}

Krzyżowanie typu II. Ten typ krzyżowania stosowany jest do wektorów zadań. Jest on bardziej skomplikowany ze względu na konieczność spełnienia warunków dopuszczalności (permutacja!) przez chromosomy potomne. Wykorzystano tutaj znaną z literatury metodę krzyżowania OX [8]. Poniżej pokazano przykład takiego krzyżowania.

Y={2,|4,5,1,7,|6,3}, {2,1,2,2,1} Z={4,|5,2,7,3,|1,6}, {1,2,1,2,2}

Y'={3,|4,5,1,7,|6,2}, {2,1,2,2,1} Z'={1,|5,2,7,3,|6,4}, {1,2,1,2,2}

3.5. Inne parametry algorytmu

Jako rozmiar populacji przyjęto 50 chromosomów. Prawdopodobieństwa obu krzyżowań ustalono na poziomie 0.1, natomiast prawdopodobieństwa mutacji i przenumerowywania na poziomie 0.2. W testach wstępnych dla powyższych wartości parametrów algorytm wykazywał najlepszą zbieżność.

4. Algorytm przeszukiwania tabu

4.1. Reprezentacja rozwiązania dopuszczalnego

Sekwencja dopuszczalna jest tu reprezentowana przez (n-m+1)-elementowy ciąg m-elementowych kombinacji zadań.

4.2. Rozwiązanie startowe

Rozwiązanie startowe dla algorytmu TS zostało uzależnione od wektora rozmiarów zadań $\bar{x}$. W pierwszym kroku generowane jest losowo rozwiązanie dopuszczalne, które jest następnie przekształcane w oparciu o wektor $\bar{x}$.

Generacja tego rozwiązania przebiega następująco:

- kombinacja Z_1 generowana jest losowo jako m-elementowa kombinacja bez powtórzeń z n-elementowego zbioru zadań,
- każda następna kombinacja Z_k, k=2,3,...,n-m+1, powstaje z poprzedniej przez wylosowanie pozycji w kombinacji (od 1 do m) i wstawienie na nią losowo wybranego zadania, które dotąd nie wystąpiło.

Następnie rozwiązanie to jest przekształcane w taki sposób, że zadanie, które wystąpiło największą liczbę razy, zostaje zastąpione zadaniem o największym rozmiarze itd. W rezultacie zadanie o większym rozmiarze pojawia się w większej liczbie kombinacji, co jest oczywiście korzystne.

4.3. Mechanizm generacji sąsiedztwa

Rozwiązanie sąsiednie jest otrzymywane z rozwiązania dopuszczalnego przez zastąpienie zadania występującego w nim na pewnej pozycji innym zadaniem. Jak wiadomo, liczba wszystkich pozycji w sekwencji dopuszczalnej wynosi $(n-m+1)\cdot m$. Zadanie może być zastąpione przez inne zadanie tylko wtedy, gdy występuje w rozwiązaniu więcej niż raz i tylko w pierwszej lub ostatniej z tych kombinacji, w których występuje (niepodzielność). Sąsiedztwo rozwiązania dopuszczalnego składa się z rozwiązań otrzymanych przez wykonanie wszystkich możliwych wymian tego typu. Załóżmy, ze zadanie i występuje w kombinacji Z_k oraz w kombinacji albo Z_{k-1} albo Z_{k+1}. Wówczas zadanie i z kombinacji Z_k jest zastępowane:

(i) zadaniem j z kombinacji Z_{k-1}, jeśli j nie należy do Z_k i należy do Z_{k-1}

albo (ii) zadaniem l z kombinacji Z_{k+1}, jeśli l nie należy do Z_k i należy do Z_{k+1}.

Przykład 1. Niech n=5, m=3, S={1,2,3}, {2,3,4}, {3,4,5}. Zbiór sekwencji sąsiednich jest następujący:

S_1={1,4,3}, {2,3,4}, {3,4,5}, S_2={1,2,4}, {2,3,4}, {3,4,5}, S_3={1,2,3}, {1,3,4}, {3,4,5}, S_4={1,2,3}, {2,3,5}, {3,4,5}, S_5={1,2,3}, {2,3,4}, {2,4,5}, S_6={1,2,3}, {2,3,4}, {3,2,5}.

4.4. Lista rozwiązań zabronionych

Lista rozwiązań zabronionych (lista tabu) zarządzana jest zgodnie z metodą REM (Reverse Elimination Method - [2]). Podstawową strukturą dla tej metody jest przejście, czyli modyfikacja prowadząca z rozwiązania dopuszczalnego do rozwiązania sąsiedniego. Przejście jest zdefiniowane jako uporządkowana trójka: (numer kombinacji, zadanie usuwane, zadanie wstawiane). Pojedynczym elementem listy tabu jest tzw. RCS (Residual Cancellation Sequence), który jest zbiorem przejść. Jeżeli rozwiązanie startowe oznaczone będzie numerem 1, wówczas RCS numer j (j-ty element listy tabu) jest minimalnym zbiorem przejść, które trzeba wykonać, aby z rozwiązania bieżącego wrócić do rozwiązania numer j (tzn. odwiedzonego w j-tej iteracji).

Metoda REM wykorzystuje fakt, ze dowolne rozwiązanie może być odwiedzone ponownie tylko wtedy, gdy jest sąsiadem rozwiązania bieżącego, tzn. gdy zbiór przejść różniący te rozwiązania składa się z dokładnie jednego przejścia. W rezultacie każde przejście, występujące na liście tabu jako jednoelementowy RCS, jest zabronione w następnej iteracji. Zatem w celu sprawdzenia statusu przejścia wystarczy sprawdzić tylko te elementy listy tabu, które składają się z dokładnie jednego przejścia.

Lista tabu modyfikowana jest w następujący sposób. Na początku procesu wszystkie RCS-y są puste. Jeżeli wykonano przejście (k,i,j), to tworzony jest nowy jednoelementowy RCS zawierający przejście odwrotne (tzn. przejście (k,j,i)), który dodawany jest do listy. Następnie wszystkie niepuste RCS-y modyfikowane są wg poniższego wzorca:

jeżeli przejście (k,i,j) należy do RCS, to jest z niego usuwane (skreślane),

w przeciwnym wypadku do RCS dodawane jest przejście (k,j,i).

Kluczowym problemem dla tej metody jest zapewnienie, że każdy RCS jest minimalnym zbiorem przejść różniących rozwiązanie bieżące od już odwiedzonego. Aby spełnić ten warunek, należy przy każdej modyfikacji dowolnego RCS-u wykonać, o ile to możliwe, operację redukcji.

Po zapełnieniu listy tabu RCS odpowiadający najbardziej odległemu rozwiązaniu jest usuwany z listy (a nie najstarszy RCS jak w algorytmie FIFO). Strategia ta oparta jest na założeniu, ze szansa powrotu do najdalszego rozwiązania jest najmniejsza.

5. Eksperyment obliczeniowy

Eksperyment obliczeniowy obejmował dobór parametrów poszczególnych heurystyk w celu uzyskania możliwie najlepszej efektywność przeszukiwania, analizę zbieżności algorytmów oraz porównanie ich efektywności dla różnych instancji problemu. Algorytmy zostały zaimplementowane w języku C++ na komputerze Silicon Power Challenge. Problem programowania matematycznego optymalizujący przydział zasobu ciągłego rozwiązywano za pomocą specjalnie przystosowanego solvera CFSQP (A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality Constraints) 2.3 [7]. Przyjęto, że kryterium stopu dla solvera jest spełnione, gdy bezwzględna różnica wartości funkcji celu w kolejnych iteracjach jest mniejsza lub równa 1.e–5. Ze względu na duże nakłady obliczeniowe eksperyment obejmował trzy grupy instancji po dziesięć zadań: dla 2, 3 i 4 maszyn. W każdej grupie wygenerowano 100 instancji o losowych rozmiarach zadań oraz funkcjach $f_i = u_i^{1/\alpha}, \alpha \in \{1,2\}$, gdzie α dobierano również w sposób losowy. Kryterium stopu dla wszystkich algorytmów była liczba odwiedzonych elementów przeszukiwanej przestrzeni. Liczbę tę przyjęto równą 2000. W tablicy 1 przedstawiono dla każdego z algorytmów liczbę instancji, dla których dany algorytm znalazł najlepsze rozwiązanie (a), średnie względne odchylenie od najlepszego znalezionego rozwiązania (b), oraz maksymalne względne odchylenie od najlepszego znalezionego rozwiązania (c). Czasy obliczeń (w tablicy podano średni czas dla jednej instancji) są porównywalne dla wszystkich algorytmów, gdyż w każdym wypadku decydujący jest czas rozwiązywania problemu wypukłego programowania matematycznego.

Tablica 1

Wyniki eksperymentu obliczeniowego

n,m	t[sec]	GA			SA			TS		
		a	b x 10^{-4}	c	a	b x 10^{-4}	c	a	b x 10^{-4}	c
10,2	1200	43	1.49		23	2.95		71	0.7	
10,3	4000	25	1.01		6	3.34		84	0.15	
10,4	6000	19	1.98		7	5.67		94	0.20	

Jak widać, największą liczbę najlepszych rozwiązań znajduje algorytm przeszukiwania tabu, zachowując przy tym najmniejsze średnie i maksymalne odchylenie od najlepszego

rozwiązania. Analiza zbieżności algorytmów wykazała, że we wszystkich przypadkach, istotna poprawa rozwiązania powyżej 2000 iteracji jest mało prawdopodobna.

6. Podsumowanie

W pracy przedstawiono wyniki eksperymentu obliczeniowego, zrealizowanego w celu porównania efektywności trzech heurystyk: przeszukiwania tabu, symulowanego wyżarzania i algorytmu genetycznego, odpowiednio dostosowanych do dyskretno - ciągłych problemów szeregowania zadań z wklęsłymi funkcjami prędkość wykonywania - ilość zasobu ciągłego. Eksperyment wykazał, że najlepsze rozwiązania w zadanym czasie otrzymuje algorytm przeszukiwania tabu. W ramach dalszych prac przewiduje się przebadanie dwóch sposobów uniknięcia czasochłonnych obliczeń optymalnego przydziału zasobu: przez heurystyczne obliczenie rozdziału zasobu ciągłego oraz przez zmniejszenie przeszukiwanej przestrzeni sekwencji dopuszczalnych.

LITERATURA

1. Aarts E. H. L., van Laarhoven P. J. M.: Simulated Annealing: Theory and Applications. Reidel, Dordrecht, 1987.
2. Glover F.: Tabu Search, Part 2. ORSA J. Computing, 2, 1990, 4-32.
3. Józefowska J, Węglarz J.: On a methodology for discrete-continuous scheduling. Report of the Inst. of Comp. Science, Poznań University of Technology, RA-95/004, 1995.
4. Józefowska J., Różycki R., Węglarz J.: A genetic algorithm for discrete-continuous scheduling problems, w: Pearson D.W., Steele N.C., Albrecht R.F., (eds.): Artifical Neural Nets and Genetic Algorithms, Springer Verlag 1995, pp. 273-276.
5. Józefowska J., Różycki R., Węglarz J.: A tabu search algorithm for discrete-continuous scheduling problems, Proc. of the Int. Conf. on Applied Decision Technologies, w: Proc. of the Int. Conf. on Applied Decision Technologies, Brunel University London, 3-5 April 1995, s. 253-260.
6. Karp R. M.: Reducibility among combinatorial problems, w: R.E. Miller, J. W. Tatcher (eds.), Complexity of Computer Computations, Penum Press, New York, 1972, 85-104.
7. Lawrence C., Zhou J. L., Tits A. L.: Users guide for CFSQP Version 2.3 (Released August 1995), available by e-mail: andre@eng.umd.edu.
8. Michalewicz Z.: Genetic Algorithms + Data Structures = Evolution Programs, Springer Verlag 1992.

Recenzent: Prof.dr hab.inż. Ryszard Gessing

Wpłynęło do Redakcji do 30.06.1996 r.

Abstract

Problems of scheduling nonpreemptable jobs which require simultaneously a machine from a set of parallel identical machines and a continuous, renewable resource are considered. For each job these are known: its processing speed as a continuous, concave function a continuous resource allotted at time t and its processing demand. The optimization criterion is the schedule length.

The problem can be decomposed into two interrelated subproblems: (i) to sequence jobs on (parallel) machines, and (ii) to find an optimal (continuous) resource allocation among jobs already sequenced. Problem (ii) can be formulated as a convex programming problem with linear constraints and solved using corresponding solvers. Thus the problem remains to generate a set of all feasible sequences of jobs on machines (this guarantees finding an optimal schedule in the general case). However, the cardinality of this set grows exponentially with the number of jobs. Thus, we propose to use heuristic search methods defined on the space of feasible sequences.

Three metaheuristics: tabu search, simulated annealing and genetic algorithm have been implemented and compared computationally. The computational experiment has been carried out on Silicon Power Challenge computer with four RISC TFP 75 MHz processors. The results of the experiment show that the tabu search performs best. Some directions for further research have been pointed out.