ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2000
Seria: AUTOMATYKA z. 129 Nr kol. 1474

Ireneusz CZARNOWSKI, Piotr JĘDRZEJOWICZ
Wyższa Szkoła Morska w Gdyni

OCENA EFEKTYWNOŚCI ALGORYTMU SZEREGOWANIA ZADAŃ WIELOPROCESOROWYCH OPARTEGO NA SZTUCZNEJ SIECI NEURONOWEJ

Streszczenie. W pracy przedstawiono wyniki analizy efektywności algorytmu szeregowania zadań wieloprocesorowych opartego na sztucznej sieci neuronowej. Zadania, o których mowa, mogą, przykładowo, reprezentować struktury lub moduły programów tolerujących błędy. Problem szeregowania zadań wieloprocesorowych dotyczy systemów czasu rzeczywistego, od których wymaga się dużej niezawodności i bezpieczeństwa. W pracy przedstawiono przybliżony algorytm szeregowania oparty na strukturze sztucznej sieci neuronowej. Algorytm został poddany ocenie metodą eksperymentów obliczeniowych, których rezultaty również przedstawiono w niniejszej pracy.

EFFECTIVENESS OF AN ARTIFICIAL NEURAL NETWORK BASED ALGORITHM FOR SCHEDULING MULTIPROCESSOR TASKS

Summary. The paper proposes an artificial neural network based algorithm for scheduling multiprocessor tasks. The considered scheduling problem class is characterized by a set of multiple, identical processors and a set of multiple-processor tasks. Multiple-processor tasks have to be processed on more than one processor at a time. Decision variables include assignment of processors to tasks and *size* of each task. To solve the above problem an artificial neural network based algorithm is proposed. Computational complexity of the approach is analyzed and evaluated. To assess effectiveness of the algorithm computational experiment has been carried. The results obtained by using the proposed algorithm have been compared with those generated by the specially designed evolutionary algorithm and the hybrid evolutionary. Experiment results prove high effectiveness of the neural network approach.

1. Wprowadzenie

Koncepcję szeregowania zadań na wielu procesorach przedstawiono między innymi w [3] i [4]. W pracach tych rozważono problemy szeregowania zadań (programów) na wielu procesorach, gdzie zadania są wykonywane na więcej niż jednym procesorze w tym samym czasie. Zadania wieloprocesorowe rozważane w tej pracy zbudowane są z pewnej liczby wykonywanych równolegle wariantów zadania. W trakcie wykonywania takiego zadania każdy jego wariant wymaga osobnego procesora. Dla tak skonstruowanego wieloprocesorowego zadania można posłużyć się parametrem $size_j$, który określa liczbę

procesorów potrzebną do wykonania zadania *j*. Wartość parametru $size_j$ określa również liczbę wykonywanych wariantów zadania *j*-tego.

Przykładem programów wieloprocesorowych zbudowanych z wielu wariantów są programy tolerujące błędy. Zadanie jest zdolne tolerować błędy, jeśli po uaktywnieniu się błędu może kontynuować w sposób poprawny obliczenia. Program tolerujący błędy, zbudowany z wielu wariantów, po ich wykonaniu podejmuje decyzję (na przykład większościową) co do poprawności swojego działania.

Propozycje programów składających z niezależnych wariantów można znaleźć między innymi w pracach [1], [15] i [12], w których zaprezentowano również takie rozwiązania zapewniające zdolność tolerowania błędów, jak: programowanie w N-wersjach, odtwarzanie i poprawianie rozwiązań z użyciem specjalnych bloków odtwarzania zwanych *recovery block schemes* oraz stosowanie N-wersyjnych samo sprawdzających się programów. Zaproponowane w tych pracach rozwiązania zakładają konstruowanie programów składających się z wielu wariantów, co w konsekwencji zapewnia zwiększenie niezawodności.

Koncepcję zadania wieloprocesorowego stanowić może uniwersalny model programu tolerującego błędy, zaś szeregowanie zadań wieloprocesorowych jest zadaniem, w którym poszukuje się kompromisu między niezawodnością uszeregowania a jego charakterystykami czasowymi.

Problemy szeregowania zadań wieloprocesorowych należą, w ogólności, do problemów NP-trudnych. Zatem do znajdywania rozwiązań stosuje się algorytmy przybliżone i heurystyczne, dające zadowalające rozwiązania w rozsądnym czasie.

Ideę użycia sieci neuronowej do rozwiązania problemu należącego do klasy NP-trudnych po raz pierwszy przedstawił Hopfield [7]. Zaproponował on użycie algorytmu o strukturze sieci neuronowej do rozwiązania problemu komiwojażera. Rozwiązanie Hopfielda wykorzystano później do rozwiązania ogólnego problemu obsługi zadań. Opis tego rozwiązania przedstawiono między innymi w pracach [8] i [16].

W dalszych częściach pracy przedstawiono wielowarstwową strukturę sieci neuronowej dla dwóch wybranych problemów szeregowania wieloprocesorowych zadań tolerujących błędy. Struktura proponowanej sieci jest zależna od aktualnie rozpatrywanego problemu i od zbioru zadań, które należy przydzielić do procesorów. Przedstawiono sposób budowania sieci oraz mechanizm uczenia oparty na algorytmie ewolucyjnym. Przeprowadzono również analizę złożoności obliczeniowej algorytmu oraz przedstawiono wyniki eksperymentów obliczeniowych.

2. Sformułowanie problemu

W rozpatrywanym problemie poszukuje się uszeregowania zbioru N niepodzielnych, niezależnych, wielowariantowych i wieloprocesorowych zadań na wielu identycznych i równoległych procesorach tak, aby zostały wykonane wszystkie zadania bez naruszenia ograniczeń czasowych spełniając nałożone kryterium. Każde z zadań charakteryzowane jest przez następujące wielkości:

- czas gotowości do wykonania - $a_j, j=1,..,N$;
- wymagany czas zakończenia wykonania - $d_j, j=1,..,N$;
- maksymalny rozmiar zadania – $M_j=max\{size_j\}, j=1,..,N$;
- czas wykonywania - $p_{ji}, j=1..N, i=1..M_j$;
- niezawodność zadania zależna od parametru $size_j$ – $R_{ji}, j=1,..N, i=1,..M_j$.

W pracy przedstawiono dwa problemy szeregowania wieloprocesorowych zadań. Pierwszy z nich, zgodnie z notacją zaproponowaną przez Grahamsa [6], oznaczony jako $P|a_j,size_j|R$, cechuje zbiór identycznych procesorów P i zbiór wielowariantowych zadań N. Każde zadanie ma określoną maksymalną wartość parametru $size=M_j$ oraz wartość minimalną równą 1. Zadania są niepodzielne i niezależne, o określonych czasach gotowości i zakończenia. Zadania mają również określone czasy wykonywania zależne od rozmiaru. Zadanie j o parametrze $size_j=M$ wymaga użycia M równoległych procesorów. Każde z zadań musi wykonać się przynajmniej w jednej swej wersji (min $\{size_j\}=1$). Kryterium optymalizacji jest wartość niezawodności uszeregowania zadań określana jako:

$$R=\prod_{j=1}^{N} R_{jk_j},$$

gdzie k_j jest wybraną wartością parametru $size_j$ zadania j, $1 \le k_j \le M_j$.

Wartość niezawodności zadania jest określona przez prawdopodobieństwo jego poprawnego wykonania, a zmienną decyzyjną jest przyporządkowanie zadań do procesorów i rozmiar każdego zadania. Zadania muszą być wykonane bez opóźnień.

Drugi z rozpatrywanych problemów oznaczono jako $P|a_j,size_j|V$. Problem ten jest analogiczny do pierwszego sformułowanego problemu, a kryterium optymalizacji jest całkowita liczba wykonywanych wariantów zadań – $V(V=\Sigma size_j)$.

3. Struktura sieci neuronowej

3.1. Architektura sieci

Struktura sieci szeregowania zadań zależy od parametrów czasowych zadań. Zbiór szeregowanych zadań dzielony jest na mniejsze podzbiory. Elementami tych podzbiorów są zadania o takich samych czasach gotowości. Zatem liczba podzbiorów zbioru, które należy uszeregować ϕ, jest równa liczbie czasów gotowości zadań i określa liczbę warstw budowanej struktury sieci. Liczba zadań w podzbiorach n_i, $i=1,..,\phi$, w odniesieniu do sieci nazywana również liczbą neuronów w warstwie, określona jest liczbą czasów gotowości o takiej samej wartości. Warstwa, dla której wartość czasu gotowości jako parametru definiującego jej istnienie jest minimalna, nazywana jest warstwą pierwszą.

Neurony transmitują swoje sygnały do warstwy zwanej warstwą pośrednią. Elementami tej warstwy są neurony o liczbie równej liczbie procesorów. Każda z warstw neuronów (zadań) przesyła sygnały do swojej warstwy pośredniej, zatem liczba warstw pośrednich równa jest liczbie warstw ϕ. Zadaniem neuronów pośrednich (sumatorów) jest sumowanie sygnałów pochodzących od neuronów reprezentujących zadania oraz od sumatorów związanych z warstwą poprzednia. W przypadku sumatorów warstwy pierwszej sumują one jedynie sygnały transmitowane przez neurony (zadania).

Neurony przesyłają sygnały do tych sumatorów, dla których wartość ich wagi połączenia z danym sumatorem jest równa 1. Połączenie to nazywane jest wówczas połączeniem aktywnym. Neurony mogą przesyłać swoje sygnały do wszystkich neuronów w warstwie pośredniej, jednakże liczba maksymalnych połączeń aktywnych nie może być większa od rozmiaru zadania ani równa 0. Dobór wag połączeń neuronów z sumatorami jest realizowany w procesie uczenia sieci, który odbywa się dla każdej z warstw z osobna. Jako mechanizm uczący wykorzystywany jest algorytm ewolucyjny.

Wartość sygnału przekazywanego do sumatorów przez neurony jest równa wartości czasu wykonywania zadania p. Wartość ta jest natomiast zależna od aktualnego rozmiaru, czyli wartości aktywnych wag danego neuronu. Sumowane sygnały w sumatorach są następnie kontrolowane przez dodatkowy blok zwany blokiem decyzyjnym lub dekoderem. Wartość tych sygnałów nie może przekraczać wymaganego czasu zakończenia wykonywania zadania d_j, od którego pochodzą sygnały wejściowe do sumatorów. W przypadku gdy nie jest spełniony warunek ograniczenia wartości sygnału blok decyzyjny uruchamia algorytm uczący. Zadaniem bloku kontrolnego jest zatem ciągły nadzór nad procesem uczenia i

sprawdzanie czy aktualny dobór wag zapewnia spełnienie ograniczenia co do wartości sygnału na wyjściu każdego z sumatorów.

Gdy wartość sygnałów na wyjściu z sumatorów jest dopuszczalna, to przesyłane są one do kolejnej warstwy pośredniej, które sumują je z sygnałami transmitowanymi przez neurony (zadania) tejże warstwy kolejnej.

Rozwiązanie problemu szeregowania wieloprocesorowych zadań wynika pośrednio z opisanej struktury sieci. W przypadku problemu $P|a_j,size_j|V$ rozwiązanie jest sumą wszystkich aktywnych połączeń neuronów.

3.2. Algorytm uczący

W procesie uczenia sieci wykorzystywany jest algorytm genetyczny [13]. Podstawową strukturę algorytmu w postaci pseudokodu przedstawia rys.1. Do podstawowych mechanizmów zastosowanych w procesie ewolucji należą: krzyżowanie jednopunktowe, mutacja oraz selekcja elitarna. Te podstawowe narzędzia algorytmu działają na populacjach, których elementami są chromosomy o reprezentacji binarnej. Chromosomy jako potencjalne rozwiązanie definiują sposób wykonania się danego zadania a w strukturze sieci wartość wag połączeń. Długość chromosomu jest równa iloczynowi liczby procesorów P i liczby neuronów w warstwie n_l. Do podstawowych parametrów algorytmu należą:

- liczebność populacji – S;
- liczba pokoleń – L;
- prawdopodobieństwo krzyżowania – p_k;
- prawdopodobieństwo mutacji – p_m.

4. Złożoność obliczeniowa algorytmu

W tej części zostanie przeprowadzona analiza złożoności obliczeniowej omawianego algorytmu. Pierwszym krokiem (rys.2) działania algorytmu jest inicjalizacja struktury sieci. Odpowiednia procedura wczytuje parametry zadań (tj. parametry czasowe, niezawodnościowe). Złożoność tej operacji wynosi $O(N)$, gdzie N jest rozmiarem danych wejściowych określającym liczbę zadań. Wczytane parametry zadań są podstawą podziału zbioru zadań na podzbiory zadań o takich samych czasach gotowości. Procedura podziału zadań jest zatem zależna od parametrów zadań. Wobec powyższego można rozpatrzyć dwa szczególne przypadki. Pierwszy przypadek dotyczy zadań, z których każde jest określone przez inny czas gotowości ($a_1<a_2<..,<a_n$). W drugim przypadku wszystkie zadania posiadają

identyczny czas gotowości. Od parametru czasu gotowości zadań zależy bowiem struktura sieci a w szczególności liczba warstw. Generalnie złożoność obliczeniowa zastosowanej procedury podziału i określenia parametrów sieci jest równa *O(N)*.

```
1: procedure algorytm_uczący;
2: begin
3: t=0;
4: inicjacja populacji początkowej Pop(0)
5: ocena Pop(0)
6: while t<L do
7:  t=t+1
8:  selekcja chromosomów z Pop(t-1)
9:  inicjacja populacji potomnej Pop(t)
10: ocena Pop(t)
11:end while
12:end begin

gdzie t jest licznikiem pętli.
```

Rys.1. Pseudokod algorytmu uczącego [13]
Fig.1. Pseudocode of the learning algorithm [13]

Dalsze rozważania dotyczące określenia złożoności przeprowadzimy dla każdego przypadku z osobna.

Przypadek N-warstwowej sieci:

Dla każdej z warstw sieci w L krokach generowana jest populacja, której chromosomy reprezentują potencjalne rozwiązania, czyli dobór wag połączeń. Generowanie populacji pierwotnej (krok 4 algorytmu uczącego) ma złożoność obliczeniową *O(SP)*. W kroku 5 każdy z chromosomów populacji podlega ocenie. Procedura oceny zbudowana jest z modułu, którego zadaniem jest określenie wartości sygnałów na wyjściu sumatorów dla danego wektora wag. Następnie uaktywniona zostaje funkcja sprawdzająca poprawność tych sygnałów i określająca wartość funkcji celu dla danej warstwy w sieci. Złożoność obliczeniowa oceny chromosomów w populacji wynosi *O(SM+SPlogP)*. W kolejnych krokach algorytmu uczącego (8-9) generowana jest populacja potomna.

Część chromosomów z populacji rodzicielskiej przechodzi jako materiał genetyczny do populacji potomnej zgodnie z selekcją elitarną o złożoności *O(SlogS)*, a następnie są one poddane operatorom mutacji i krzyżowania. Zastosowana procedura krzyżowania po losowym wybraniu pary chromosomów dokonuje wymiany pomiędzy nimi części genów. Złożoność tej operacji wynosi *O(P)*. Drugi z operatorów wybiera losowo gen z chromosomu i zmienia ich wartość na przeciwną. Operacja ta jest stała w czasie *O(c)*. Wartość prawdopodobieństw p_k i p_m nie ma zatem wpływu na złożoność obliczeniową w algorytmie

uczącym. Z rozważań tych wynika, że złożoność algorytmu uczącego na jedną iterację jest równa $O(SPlogP+SM)+O(SlogS)$, gdzie $M=max\{M_i\}$.

```
1: procedure algorytm_neuronowy
2: begin
3: inicjalizacja struktury sieci (określenie φ, n_i)
4: skok:
5: for i=1 to φ do
6:  repeat
7:  uaktywnij algorytm uczenia
8:  until uzyskasz_najlepszy_dobór_wag
9: if rozwiązanie jest niedopuszczalne then goto skok
10:end for
11:end begin
```

Rys.2. Ogólna struktura proponowanego algorytmu neuronowego
Fig.2. General structure of the proposed algorithm

Proces uczenia odbywa się dla każdej warstwy osobno, stąd całkowita złożoność obliczeniowa N-warstwowej sieci wynosi $O(NSPlogP+ NSM)+O(NSlogS)$.

Przypadek sieci jednowarstwowej:

W tym przypadku procedura ucząca ma odnaleźć wartość wag dla jednej z warstw. Wszystkie bowiem zadania zgodnie z założeniem należą do tej samej warstwy. Chromosom w tym przypadku składa się z $P{\cdot}N$ elementów (genów), zatem proces generacji populacji pierwotnej ma złożoność obliczeniową wynoszącą $O(NSP)$. Ocena chromosomów w populacji ma złożoność $O(SNP)+O(SPlogP+SNM)$. Generowanie populacji potomnej poprzedzone jest selekcją elitarną chromosomów ($O(SlogS)$), które następnie poddane są krzyżowaniu i mutacji. Losowo wybrane chromosomy ulegają krzyżowaniu w czasie $O(NP)$. W rezultacie rozważań złożoność procedury uczącej na jedną iterację wynosi $O(NPS)+O(SPlogP+NSM)+O(SlogS)$.

5. Wyniki eksperymentów obliczeniowych

Przedstawiony w artykule algorytm oparty na strukturze sztucznej sieci neuronowej poddano ocenie przez eksperymenty obliczeniowe. Algorytm wykorzystano do wyznaczenia rozwiązań dla obu rozpatrywanych problemów szeregowania. Dla każdego z problemów przygotowano 90 zestawów danych o liczbie zadań od 11-28. Maksymalna wartość parametru *size* dla szeregowanych zadań wynosiła 4. Zadania szeregowano na 4-10 procesorach. Dla rozpatrywanych zestawów danych otrzymano struktury sieci o liczbie warstw od 2-25. Otrzymane rezultaty eksperymentów porównano z innymi wynikami otrzymanymi dla tych

samych zestawów danych i przy użyciu algorytmu ewolucyjnego [9] i algorytmu hybrydowego [5].

Tabela 1

Średni, minimalny i maksymalny błąd względny dla $P|a_j, size_j|R$ i $P|a_j, size_j|V$

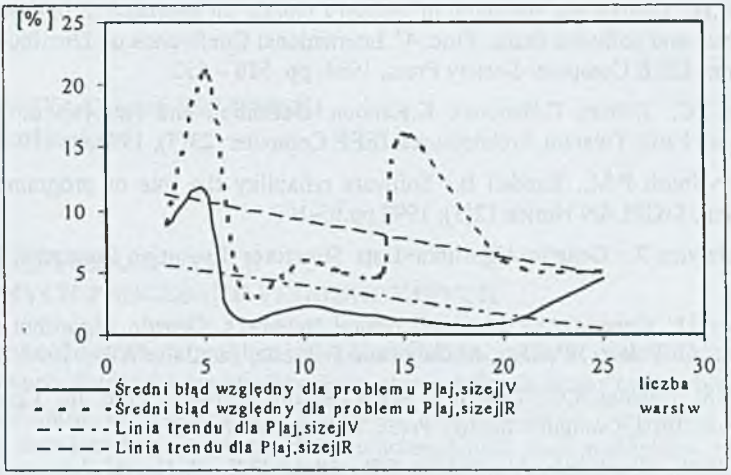
Liczba proc.	$P\|a_j, size_j\|R$									$P\|a_j, size_j\|V$								
	Algorytm ewolucyjny			Algorytm hybrydowy			Sieć neuronowa			Algorytm ewolucyjny			Algorytm hybrydowy			Sieć neuronowa		
	Średni (%)	Min (%)	Max (%)	Średni (%)	Min (%)	Max (%)	Średni (%)	Min (%)	Max (%)	Średni (%)	Min (%)	Max (%)	Średni (%)	Min (%)	Max (%)	Średni (%)	Min (%)	Max (%)
4	18,82	0	54	8,88	0	46	0,00	0	0	6,00	2	10	4,25	0	9	5,00	2	10
5	15,42	0	50	4,67	0	19	0,00	0	0	9,25	0	18	4,73	0	14	3,27	0	18
6	15,67	1	89	3,42	0	14	0,00	0	0	15,17	2	33	6,64	0	28	1,50	0	12
7	19,67	0	98	9,64	0	67	2,86	0	16	23,14	11	43	12,29	0	35	6,86	0	35
8	20,13	0	95	11,54	0	96	6,71	0	28	20,29	8	37	10,07	0	30	6,27	0	30
9	17,47	0	94	11,50	0	92	4,17	0	14	23,20	11	33	11,36	0	30	7,55	9	28
10	17,20	0	91	11,00	0	89	2,17	0	10	20,00	14	29	10,88	0	22	6,38	5	22
średnio	***17,77***			***8,66***			***2,27***			***16,72***			***8,60***			***4,95***		

W tabeli 1 przedstawiono średni błąd względny dla obu rozpatrywanych problemów. Błąd ten został określony w stosunku do najlepszych otrzymanych wyników dla rozpatrywanych problemów. Na rysunku 3 przedstawiono wykres zależność średniego błędu względnego od liczby warstw – określonych podzbiorów ϕ dla poszczególnych zestawów danych. Linia trendu wskazuje na malejącą wartość błędu względnego wraz ze wzrostem warstw w sieci. Otrzymane wyniki nie są optymalne ale w porównaniu do innych rozwiązań są lepsze. Ponadto wartości błędu dla obu rozpatrywanych problemów kształtują się na podobnym poziomie.

6. Podsumowanie

Otrzymane rezultaty eksperymentów obliczeniowych dla przedstawionej sieci neuronowej w problemie szeregowania wieloprocesorowych zadań świadczą o efektywności algorytmu, czego potwierdzeniem jest ich porównanie z rozwiązaniami otrzymanymi z wykorzystaniem innych metod heurystycznych. Przeprowadzone eksperymenty wykazały porównywalną skuteczność algorytmu dla zestawów danych o różnej wielkości oraz o różnej złożoności struktury sieci.

Kolejnym krokiem w badaniu opisanego algorytmu będzie sprawdzenie jego efektywności dla typowych problemów szeregowania zadań, na przykład w ogólnym systemie obsługi zadań i przy kryterium maksymalizacji długości uszeregowania czy maksymalizacji opóźnienia.



Rys.3. Zależność wartości błędu względnego od liczby warstw sieci
Fig.3. Dependence of relative error value on network layers number

LITERATURA

1. Avizienis A., Chen L.: On the implementation of the N-version programming for software fault tolerance during execution, Proc. IEEE COMPSAC 77, 1977, pp. 149-155.
2. Biagioni E., Abe T., Ishii S: Applying Neural Network to Scheduling Problems, Conference of the Information Processing Society of Japan, Tokyo, September 1989.
3. Błażewicz J., Drabowski M., Węglarz J.: Scheduling Multiprocessor Tasks to Minimize Schedule Length, IEEE Transactions on Computers, 35, 1986, pp. 389 - 393.
4. Błażewicz J., Ecker K.H., Pesch E., Schmidt G., Węglarz J.: Scheduling Computer and Manufacturing Processes, Springer, Berlin, 1996.
5. Czarnowski I., Skakowski A.: Hybrydowy Model Szeregowania Programów Tolerujących Błędy, Algorytmy Ewolucyjne i Optymalizacja Globalna, Potok Złoty May 1999, pp.83-90.
6. Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals Discrete Math., 5, 1979, pp. 287 - 326.
7. Hopfield J.J., Tank D.W.: "Neural" Computations of Depencision in Optimisations Problems, Biological Cybernetics, vol. 52, 1985, 141-152
8. Janiak A.: Wybrane Problemy i Algorytmy Szeregowania Zadań i Rozdziału Zasobów, Akademicka Oficyna Wydawnicza PLJ, Warszawa 1999.
9. Jędrzejowicz P., Czarnowski I., Szreder H., Skakowski A.: Evolution-Based Scheduling of Fault-Tolerant Programs on Multiple Processor, Lecture Note in Computer Science nr1586, Springer 1999, pp.210-219.

10. Kim K.H.: Distributed execution of recovery blocks: an approach to uniform treatment of hardware and software faults, Proc. 4th International Conference on Distributed Computing Systems, IEEE Computer Society Press, 1984, pp. 526 – 532.

11. Laprie J.C., J.Arlat, C.Beounes, K.Kanoun: Definition and Analysis of Hardware-and-Software Fault-Tolerant Architectures, IEEE Computer, 23(7), 1990, pp. 39-51.

12. Melliar-Smith P.M., Randell B.: Software reliability the role of programmed exception handling, SIGPLAN Notice 12(3), 1997 pp.95-100.

13. Michalewicz Z.: Genetic Algorithm+Data Structures=Evolution Programs. Spring-Verlag, 1992.

14. Whitley D.: Genetic Algorithm and Neural Networks, Genetic Algorithm and Computer Science. G.Winter, J.Periaux, M.Galan and P.Cuesta, eds. John Wiley 1995, pp.203-216.

15. Yau S.S., Cheung R.C.: Design of Self-Checking Software, Proc. Int. Conf. on Reliable Software, IEEE Computer Society Press, 1975, pp. 450 – 457.

16. Zohu D.N., Cherkassky V., Balwin T.R., Olson D.E.: A Neural Approach to Job Shop Scheduling, IEEE Transactions on Neural Networks, vol 2, 1991, 175-179.

Recenzent: Prof.dr hab.inż. R.Gessing

Abstract

The paper proposes an artificial neural network based algorithm for scheduling multiprocessor tasks. The considered scheduling problem class is characterized by a set of multiple, identical processors and a set of multiple-processor tasks. Multiple-processor tasks have to be processed on more than one processor at a time. During an execution of these tasks communication among processors working on the same task is implicitly hidden in a black box denoting an assignment of this task to a subset of processors during some time interval. Tasks are independent and non-preemptable with processing times, ready times and deadlines differing per task. Two optimization criteria are considered - sum of the task sizes scheduled without delays and schedule reliability. Decision variables include assignment of processors to tasks and *size* of each task. To solve the above problem an artificial neural network based algorithm is proposed. The paper gives a description of the network architecture and of the learning algorithm used. Computational complexity of the approach is analyzed and evaluated. To evaluate effectiveness of the algorithm computational experiment has been carried. The results obtained by using the proposed algorithm have been compared with those generated by the specially designed evolutionary algorithm and the hybrid evolutionary. Experiment results prove high effectiveness of the neural network approach.