RÓWNOLEGŁY ALGORYTM NEURO-TABU DLA PROBLEMU GNIAZDOWEGO SZEREGOWANIA ZADAŃ

Wojciech BOŻEJKO, Mariusz UCHROŃSKI, Mieczysław WODECKI

Streszczenie: W pracy proponujemy zastosowanie dwóch równoległych algorytmów bazujących na mechanizmie neuro-tabu do rozwiązywania klasycznego problem gniazdowego szeregowania zadań (*job shop*). Pierwszy z algorytmów oparty jest na wykonywaniu niezależnych współbieżnych algorytmów poszukiwania z zabronieniami (*tabu search*) z siecią neuronową zastępującą listę ruchów zabronionych (algorytm neuro-tabu). Drugi z algorytmów bazuje na unikalnej metodzie dywersyfikacji połączonej z mechanizmem ścieżek łączących (*path-relinking*) zastosowanym do zbioru rozwiązań elitarnych. Proponowane podejścia okazują się szczególnie efektywne dla przykładów problemów o dużych rozmiarach.

Słowa kluczowe: optymalizacja dyskretna, GPU, poszukiwanie z zabronieniami, sieć neuronowa

1. Wprowadzenie

Rozważamy w pracy problem gniazdowy szeregowania zadań, który można ująć następująco (zobacz [2]). Dane są zbiór zadań oraz zbiór maszyn. Każde zadanie składa się z pewnej liczby operacji które muszą zostać wykonane w zadanej kolejności, każde na dedykowanej sobie maszynie, przez pewien czas. Wykonanie operacji nie może być przerywane. Każda maszyna może wykonywać co najwyżej jedną operację w danym momencie czasu. Chcemy znaleźć takie uszeregowanie (tj. przyporządkowanie oepracji do mementów czasowych na maszynach) które minimalizuje maksymalny czas wykonania zadań (ang. *makespan*). Tak opisany gniazdowy problem szeregowania zadań, choć prosty do zdefiniowania, jest z punktu widzenia teorii złożoności obliczceniowej problemem silnie NP-trudnym i jest uważany za jeden z najtrudniejszych problemów optymalizacji kombinatorycznej.

Z powodu NP-trudności problemu do jego rozwiązywania rekomenduje się głównie heurystyki i metaheurystyki jak „najrozsądniejsze" metody rozwiązania. Większość proponowanych metod odnosi się do problemu minimalizacji makszymelnego czasu zakończenia wykonywania zadań. Cytując najnowsze publikacje dotyczące metaheurystyk dla rozważanego problemu, wymienić należy prace: Jain, Rangaswamy i Meeran [9]; Pezzella i Merelli [12]; Grabowski i Wodecki [6]; Nowicki i Smutnicki [11]; Bożejko i Uchroński [3]. Algorytmy heurystyczne proponują także Holthaus i Rajendran [8] oraz Bushee i Svestka [4]. Dla innych kryteriów regularnych takich jak kryterium sumacyjne oraz sumy spóźnień w literaturze proponowane są metaheurystyki bazujące na metodzie poszukiwania z zabronieniami [1] i algorytmu genetycznego [10].

W niniejszej pracy proponujemy nowe podejście do projektowania rozproszonych algorytmów poszukiwania z zabronieniami rozwiązywania trudnych problemów optymalizacji dyskretnej, takich jak problem gniazdowy, używając architektury multi-GPU, tj. klastra zołożonego z węzłów wyposazonych w karty obliczeniowe GPU.

2. Definicja problemu

Dany jest zbiór zadań $J = \{1,2,...,n\}$, które należy wykonać na maszynach ze zbioru $P = \{1,2,...,m\}$. Zadanie jest ciągiem pewnych operacji. Każdą operację należy wykonać bez przerywania na odpowiedniej maszynie w ustalonym czasie. Problem polega na wyznaczeniu kolejności wykonywania operacji na każdej maszynie, minimalizującej czas wykonania wszystkich zadań.

Niech $O = \{1,2,...,o\}$ będzie zbiorem wszystkich operacji. Zbiór ten można rozbić na ciągi odpowiadające zadaniom przy czym, zadanie $j \in J$ jest ciągiem o_j operacji, które będą kolejno wykonywane na odpowiednich maszynach (tzw. *ciąg technologiczny*). Operacje te są indeksowane liczbami $(l_{j-1}+1,...,l_{j-1}+o_j)$, gdzie $l_j = \sum_{i=1}^{j} o_i$ jest liczbą operacji pierwszych j zadań, $j = 1,2,...,n$, przy czym $l_0 = 0$, a $o = \sum_{i=1}^{n} o_i$. Operacja $v \in O$ będzie wykonywana na maszynie $\mu_v \in P$ w czasie p_v. Zbiór O można także rozbić na podzbiory $O_k = \{v \in O : \mu_v = k\}$ $(k \in P)$ operacji wykonywanych na tej samej maszynie. Niech permutacja π_k wyznacza kolejność wykonywania operacji ze zbioru O_k na maszynie k. Przez Π_k oznaczmy zbiór wszystkich permutacji elementów z O_k. Wobec tego, dowolna kolejność wykonywania wszystkich operacji na maszynach jest wyznaczona przez konkatenację m ciągów (permutację) $\pi = (\pi_1, \pi_2, ..., \pi_m)$, gdzie $\pi \in \Phi = \Pi_1 \times \Pi_2 \times ... \times \Pi_m$. Oczywiście Φ może zawierać także rozwiązania niedopuszczalne.

Każde rozwiązanie dopuszczalne problemu gniazdowego można przedstawić w postaci grafu. Dla dowolnej kolejności wykonywania operacji na maszynach (permutacji $\pi \in \Phi$), konstruujemy graf skierowany z obciążonymi wierzchołkami (*sieć*) $G(\pi) = (V, R \cup E(\pi))$, gdzie V jest zbiorem wierzchołków, a $R \cup E(\pi)$ zbiorem łuków, przy czym:

1. $V = O \cup \{s,c\}$, gdzie s i c są dodatkowymi (fikcyjnymi) operacjami reprezentującymi odpowiednio „*start*" i „*zakończenie*". Wagą wierzchołka $v \in O$ jest czas wykonywania p_v, odpowiadającej mu operacji $(p_s = p_c = 0)$.

2. $R = \bigcup_{j=1}^{n} \left[\bigcup_{i=1}^{o_j - 1} \left\{ \left(l_{j-1}+i, l_{j-1}+i+1 \right) \right\} \cup \left\{ \left(s, l_{j-1}+1 \right) \right\} \cup \left\{ \left(l_{j-1}+o_j, c \right) \right\} \right]$.

Zbiór R zawiera łuki łączące kolejne operacje tego samego zadania oraz łuki z wierzchołka s do pierwszej operacji każdego zadania i łuki od ostatniej operacji każdego zadania do wierzchołka c.

3. $E(\pi) = \bigcup_{k=1}^{m} \bigcup_{i=1}^{|O^k|-1} \left\{ \left({}_k(i), {}_k(i+1) \right) \right\}$.

Łatwo zauważyć, że łuki ze zbioru $E(\pi)$ łączą operacje wykonywane na tej samej maszynie (π_k jest permutacją operacji z O_k).

Permutacja (kolejność operacji na maszynach) $\pi \in \Phi$ jest dopuszczalna wtedy i tylko wtedy, gdy odpowiadający jej graf $G(\pi)$ nie zawiera cykli.

Niech permutacja $\pi \in \Phi$ będzie rozwiązaniem dopuszczalnym dla problemu gniazdowego, a $G(\pi)$ odpowiadającym jej grafem. Ciąg wierzchołków $(v_1, v_2, ..., v_k)$ grafu $G(\pi)$ taki, że $(v_i, v_{i+1}) \in R \cup E(\pi)$ dla $i = 1, 2, ..., k-1$, nazywamy *drogą* (lub *ścieżką*) z wierzchołka v_1 do v_k. Przez $C(v,u)$ oznaczmy najdłuższą drogę (*drogę krytyczną*) w $G(\pi)$ z wierzchołka v do u, a przez $L(v,u)$ *długość* (sumę wag wierzchołków) tej drogi. Łatwo zauważyć, że czas wykonywania operacji $C_{\max}(\pi)$ w kolejności π jest równy długości $L(s,c)$ drogi krytycznej $C(s,c)$. Wobec tego, rozwiązanie problemu gniazdowego sprowadza się do wyznaczenia permutacji dopuszczalnej $\pi \in \Phi$, dla której odpowiadający jej graf ma najkrótszą drogę krytyczną, tj. minimalizuje $L(s,c)$.

3. Mechanizm tabu z zastosowaniem sieci neuronowej

W klasycznej metodzie tabu pewnien ruch jest wybierany w każdej iteracji algorytmu. Ruch ten jest pamiętany na pewnej liście o długości $maxt$ zwanej listą tabu i jest zabroniony przez $maxt$ liczbę iteracji. Po wykonaniu przez algorytm tabu search $maxt$ iteracji ruch ten jest usuwany z listy i może być wykonany ponownie. Można powiedzieć, że ruch ten traci swój status ruchu zabronionego „nagle". Poniżej zostanie przedstawiony mechanizm, w którym status ruchu zabronionego zmniejsza się w sposób wykładniczy.

W rozpatrywanym algorytmie tabu search każdy ruch reprezentowany jest przez neuron. Dla rozpatrywanego w pracy sąsiedztwa [11] sieć neuronowa składa się z $o-1$ neuronów. Podejście to z powodzeniem zostało zastosowane dla kwadratowego problemu przydziału [7] oraz dla problemu przepływowego [13]. Niech i-ty neuron reprezentuje ruch polegający na zamianie dwóch przyległych elementów na pozycjach i i $i+1$ w permutacji π. W zaproponowanej architekturze sieci neuronowej historia każdego neuronu jest pamiętana jako jego stan wewnętrzny (*efekt tabu*). Jeśli w pewnej iteracji neuron zostanie aktywowany to na jego wyjściu zostanie ustalona wartość 1, a na wyjściach pozostałych neuronów zostanie ustalona watość 0. Aktywowany w danej iteracji neuron nie może być ponownie aktywowany przez kolejnych s iteracji. Każdy neuron opisany jest przez następujące równania:

$$\eta_i(t+1) = \alpha \Delta_i(t), \tag{1}$$

$$\Delta_i(t) = \frac{C_{max}(\pi_\upsilon^{(t)}) - C^*_{max}}{C^*_{max}}, \tag{2}$$

$$\gamma_i(t+1) = \sum_{d=0}^{s-1} k^d x_i(t-d), \tag{3}$$

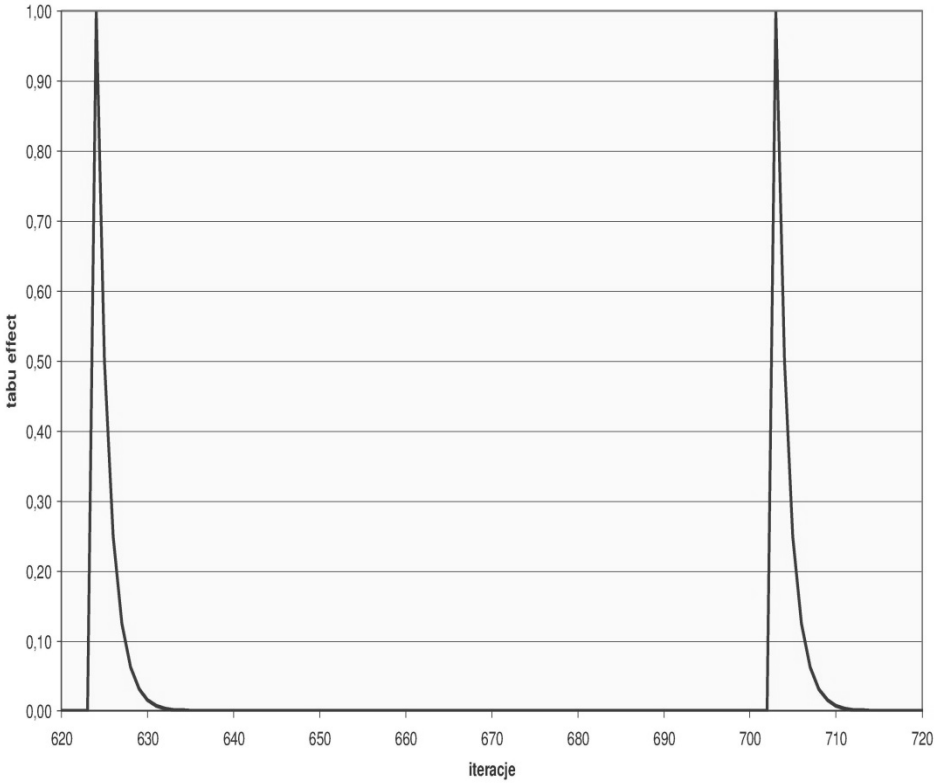
gdzie $x_i(t)$ oznacza wyjście neuronu i w iteracji t. Symbol $C_{max}(\pi_\upsilon^{(t)})$ oznacza wartość funkcji celu dla permutacji uzyskanej w wyniku wykonania ruchu υ w iteracji t, tzn. $\pi^{(t)}$. Symbol $\Delta_i(t)$ oznacza znormalizowaną, bieżącą wartość funkcji celu, a C^*_{max} oznacza najlepszą znalezioną dotychczas wartość funkcji celu. Parametry α i k są

współczynnikami skalującymi. Symbol $\eta_i(t+1)$ (*gain effect*) określa jakość ruchu υ. Zmienna $\gamma_i(t+1)$ (*tabu effect*) przechowuje historię neuronu i dla ostatnich s iteracji. Neuron jest aktywowany jeżeli ma niską wartość efektu tabu i zapewnia lepszą redukcję C_{max}. Dokładniej neuron i zostaje aktywowany jeżeli ma najmniejszą wartość $\{\eta_i(t+1)+\gamma_i(t+1)\}$ spośród wszystkich neuronów.

Jeżeli $0<k<1$ i $s=t$ to równanie (3) przyjmie następującą postać:

$$\gamma_i(t+1)=k\gamma_i(t)+x_i(t), \tag{4}$$

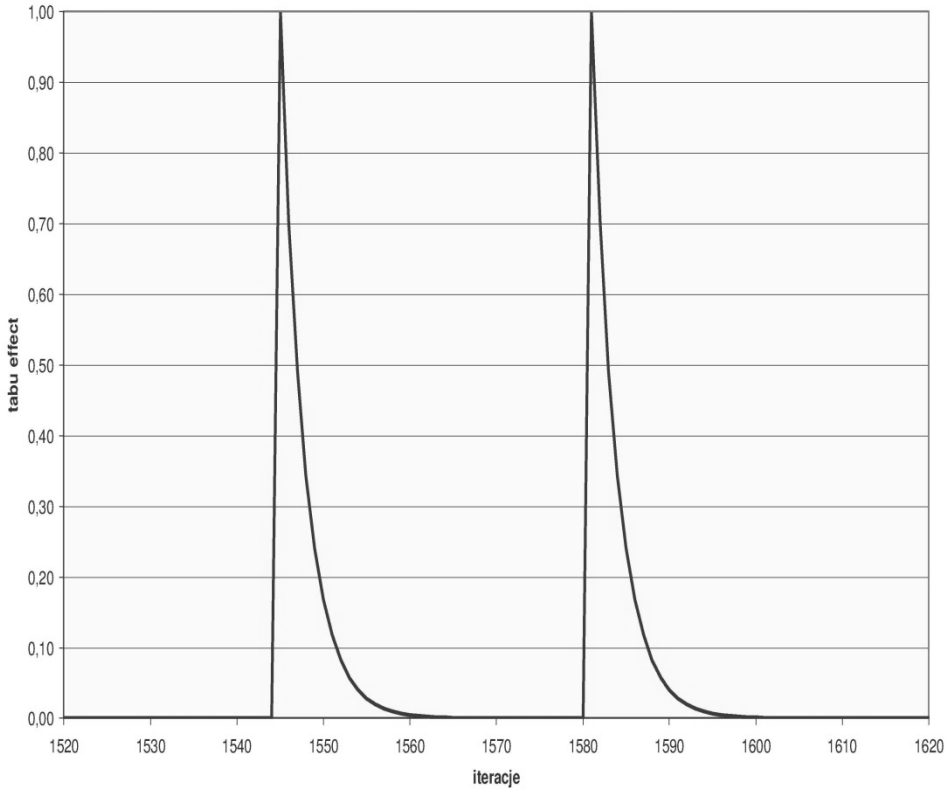
gdzie $\gamma_i(0)=0$ oraz $x_i(0)=0$ dla każdego i. Z równania (4) wynika, że wartość $\gamma_i(t)$ każdego neuronu maleje wykładniczo (Rysunek 1 i 2).



Rys. 1. Zmiany wartości $\gamma(t)$ dla $k=0.5$

Wielu proponowanych w literaturze algorytmach opartych na metodzie tabu search wyposażonych jest w mechanizm pozwalający na wykonaniu ruchu zabronionego, który bezpośrednio lub pośrednio prowadzi do rozwiązań bazowych o wartości funkcji celu mniejszej niż dotychczas znaleziona. Mechanizm ten jest nazywany kryterium aspiracji. W proponowanym algorytmie neuro-tabu search taka funkcja może zostać zaimplementowana

poprzez ignorowanie efektu tabu dla ruchu υ dla którego $\Delta_i < 0$. Jednak na drodze przeprowadzonych eksperymentów obliczeniowych, mechanizm ten dla implementowanego algorytmu nie przynosi pożądanych efektów. Dlatego też algorytm neuro-tabu search dla problemu gniazdowego nie został wyposażony w taką funkcję.



Rys. 2. Zmiany wartości $\gamma(t)$ dla $k = 0.7$.

4. Równoległy algorytm neuro-tabu

Jako drugą metodę rozwiązania rozważanego problemu proponujemy podejście oparte na rozpatrywanym w pracy Nowickiego i Smutnickiego [11] z modyfikacją polegającą na użyciu algorytmu neuro-tabu zamiast klasycznego algorytmu poszukiwania z zabronieniami. Proponowany w niniejszej pracy algorytm $iNTS$ operuje na zbiorze rozproszonych rozwiązań (elitarnych) uzyskanych za pomocą funkcji $NIS(\gamma, \delta, C^R)$ (opis na Rys. 3), gdzie γ, δ są rozwiązaniami referencyjnymi (porządkiem operacji) a C^R jest referencyjną wartością funkcji kryterialnej. Rozważana funkcja używa generatora sąsiedztwa opartego na zamianie przyległych operacji (*swap of adjacent operations*, zobacz

Bożejko [4]). Oznaczmy przez $N(\pi)$ zbiór ruchów otrzymanych przez zamianę dwóch przyległych operacji na ścieżce krytycznej w rozwiązaniu π (aby otrzymać rozwiązanie dopuszczalen, zobacz [6]). Podstawowym zadaniem funkcji *NIS* jest wygenerowanie rozwiązania $\varphi = NIS(\gamma, \delta, C^R)$ leżącego „pomiędzy" γ a δ (rozwiązaniamia referencyjnymi) aby uruchomić eksporację algorytmem *iNTS* (zobacz Rys. 4). Aby kontorlować odległość pomiędzy danymi rozwiązaniami α, β zastosowano miarę *tau Kendalla* $D(\alpha, \beta)$ wyrażającą minimalną liczbę przyległych zamian (inwersji) potrzebnych do przekształcenia permutacji α^{-1} w β^{-1} (więcej o miarach odległości dla permutacji znaleźć można w pracy Diaconisa [5]). Przyjęto następujące wartości parametrów strojących: $maxE = 8$ (liczba elitarnych rozwiązań), $maxD = 5$ (maksymalna odległość, używana w głównej pętli), $maxV = 0.5$.

Algorytm 1. $NIS(\gamma, \delta, C^R)$

Wejście: γ, δ – rozwiązania referencyjne; C^R – referencyjna wartość kryterium;

Wyjście: φ – rozwiązanie; poprawione referencyjne kryterium C^R;

$\pi \leftarrow \gamma$; $iter \leftarrow 0$. Znajdź δ^{-1} oraz $D(\gamma, \delta)$

Repeat

$iter \leftarrow iter + 1$; Wyznacz $N(\pi)$;

Dla każdego $v \in N(\pi)$ policz i zapamiętaj $C_{max}(\pi_{(v)})$;

Wyznacz $N^+ = \{v = (x, y) \in N(\pi) : \delta^{-1}(y) < \delta^{-1}(x)\}$;

if $N^+ \neq \varnothing$ **than** $K \leftarrow N^+$ **else** $K \leftarrow N(\pi)$;

Wybierz taki ruch $w \in K$, że

$C_{max}(\pi_{(w)}) = \min_{v \in K} C_{max}(\pi_{(v)})$;

Oznacz $\pi_{(w)}$ przez α;

$\pi \leftarrow \alpha$; $\varphi \leftarrow \pi$;

if $C_{max}(\pi) < C^R$ **than** $C^R \leftarrow C_{max}(\pi)$ **and exit**;

until $iter \geq maxV \cdot D(\gamma, \delta)$; { $maxV \in (0,1)$ - parametr}

Rys. 3. Funkcja $NIS(\gamma, \delta, C^R)$.

Algorytm 2. *iNTS*

Wejście: π^0 – rozwiązanie uzyskane algorytmem INSA;

Output: π^* – najlepsze znalezione rozwiązanie oraz jego wartość kryterium C^*;

Podstaw $(\pi^1, C^1) \leftarrow NTS(\pi^0)$ oraz $C^* \leftarrow C^1$;

for $i \leftarrow 2,\ldots,maxE$ **do**

$\varphi \leftarrow NIS(\pi^{i-1}, \pi^0, C^*)$; $(\pi^i, C^i) \leftarrow NTS(\varphi)$;

$C^* = \min\{C^*, C^i\}$;

Repeat

Znajdź $1 \le l \le maxE$ takie, że

$D(\pi^k, \pi^l) = \max\{D(\pi^k, \pi^i) : 1 \le i \le maxE\}$;

Podstaw $\varphi \leftarrow NIS(\pi^k, \pi^l, C^*)$ oraz $(\pi^l, C^l) \leftarrow NTS(\varphi)$;

if $C^l < C^k$ **than** podstaw $(\pi^*, C^*) \leftarrow (\pi^l, C^l)$ oraz $k \leftarrow l$;

until $\max\{D(\pi^k, \pi^i) : 1 \le i \le maxE\} < maxD$.

Rys. 4. Algorytm *iNTS*

5. Eksperymenty obliczeniowe

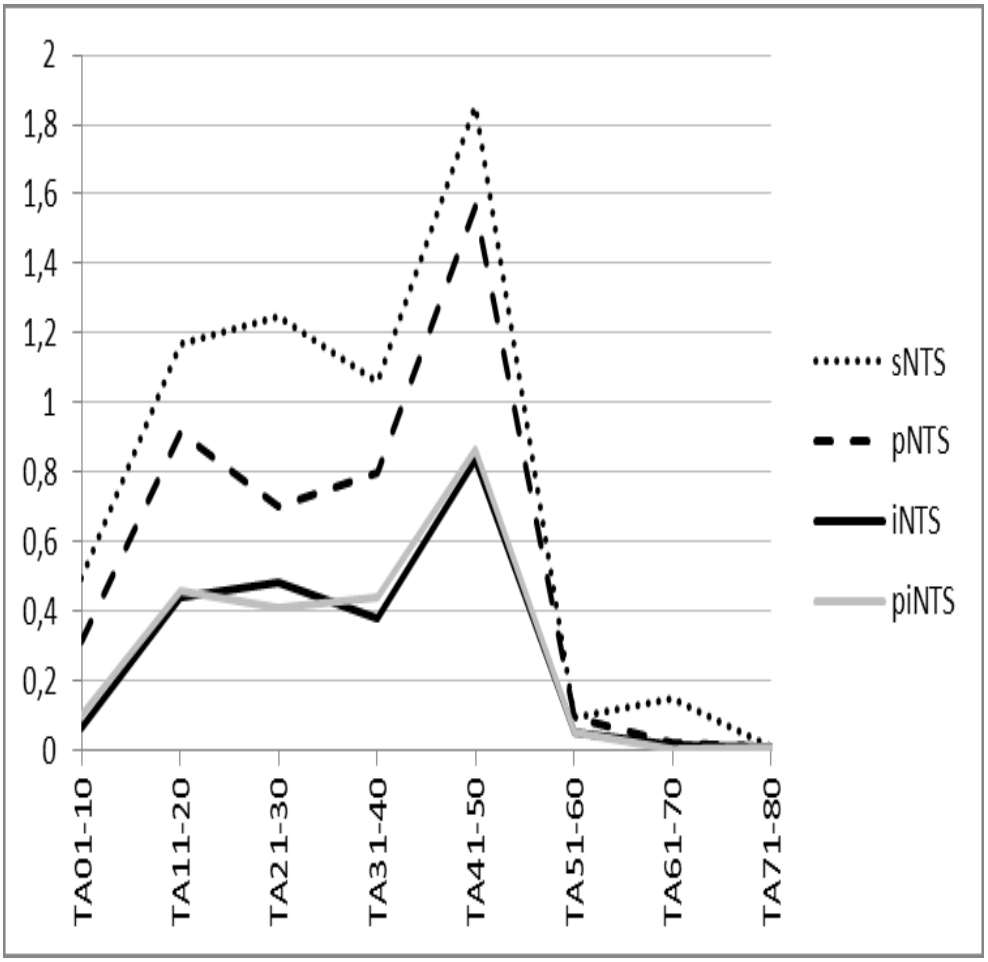
Algorytmy zostały uruchomione na 6-rdzeniowym serwerze z procesorem Intel Core i7 CPU X980 (3.33GHz) połączonym z kartą oblczeniową GPU nVidia Tesla S2050 (1792 rdzeni) procującym pod kontrolą 64-bitowego systemu Linux Ubuntu 10.04.4 LTS I przetestowane na przykładach testowych Taillarda [14]. Wyniki obliczeń porównanow w Tabeli 1. Oznaczenia poszczególnych kolumn oznaczają:

- *sNTS* – sekwencyjny algorytm Neuro Tabu Search z pracy Bożejko i Uchroński [3],
- *pNTS* – równoległy (dla liczby wątków $p = 16$) algorytm Neuro Tabu Search algorithm, model *MPSS* (*Multiple starting Points, Single Strategy,* według klasyfikacji Voß'a [14] równoległych algorytmów poszukiwania za zabronieniami – wszystkie wątki pracowałe według tej samej strategii, ale startowały z różnych rozwązań początkowych) bez komunikacji; dla każdego procesora rozwiązanie startowe było generowane przez algorytm *NTS* wykonywane przez $process\_id * 100$ iteracji,
- *iNTS* – algorytm *NTS* bazujący na mechanizmie dywersyfikiacji i intensyfikacji oraz zbiorze rozwiązań elitarnych opisanym w Rozdziale 4,
- *piNTS* – równoległy algorytm *iNTS*. Zbiór rozwiązań elitarnych generowany jest współbieżnie na GPU.

Tab. 1. Procentowe względne odchylenie do najlepszych znanych rozwiązań

Problem	$n \times m$	*sNTS*	*pNTS* (*p*=16)	*iNTS*	*piNTS* (*p*=8)
TA01-10	15×15	0,4948	0,3141	0,0652	0,0975
TA11-20	20×15	1,1691	0,9129	0,4412	0,4596
TA21-30	20×20	1,2486	0,7033	0,4803	0,4065
TA31-40	30×15	1,0592	0,7965	0,3764	0,4395
TA41-50	30×20	1,8565	1,5634	0,8328	0,8643
TA51-60	50×15	0,0915	0,0915	0,0520	0,0520
TA61-70	50×20	0,1479	0,0210	0,0140	0,0035
TA71-80	100×20	0,0090	0,0090	0,0090	0,0090
średnia		**0,7596**	**0,5515**	**0,2839**	**0,2915**

Czasy wykoniania dla wszystkich instancji testowych były następujące: (sekwencyjny) NTS – 132m27.319s, (równoległy) $pNTS$ ($p = 4$) – 169m45.748s (dłuższy czas działania algorytmu spowodowany jest metodą generowania rozwiązań startowych oraz synchronizacją), $iNTS$ – około 60 godzin. Jak można zaobserwować proponowany algorytm $iNTS$ uzyskał średni poziom odchylenia względnego (*percentage relative deviation, PRD*) do najlepszych znanych rozwiązań (dla instancji testowych Taillarda) na poziomie 0,28%. Z kolei równoległ *piNTS* uzyskał średnie PRD dla grupy przykładów o dużych rozmiarach TA61-TA70 (z przykładami dla 1000 operacji) 10 razy niższe niż proponowany algorytm $iNTS$. Porównując rezultaty uzyskane przez algorytmy $sNTS$ oraz $pNTS$ można zauważyć, że użycie środowiska wieloprocesorowego skutkuje dużym obniżeniem uzyskiwanych błędów *PRD* do najlepszych znanych rozwiązań (zobacz Rys. 5).



Rys. 5. Porównanie efektywności algorytmów dla przykładów Taillarda [14]

6. Wnioski

W pracy proponujemy metodologię projektowania algorytmów równoległych rozwiązywania trudnych problemów optymalizacji kombinatorycznej dla architektury współbieżnych wyposażonych zarówno w pamięć rozproszoną (klastry), jak i pamięć współdzieloną (GPU), oraz ich kombinacje (multu-GPU). Metodologia ta okazuje się być szczególnie efektywna przy rozwiązywaniu problemów o dużych rozmiarach, dla których nieskuteczne okazują sie zarówno metody dokładne, jak I sekwencyjne metody metaheurystyczne, takie jak *sNTS*.

Literatura

1. Armentano V.A., Scrich C.R.: Tabu search for minimizing total tardiness in a job shop. International Journal of Production Economics 63(2), 131–140 (2000)
2. Bożejko W.: On single-walk parallelization of the job shop problem solving algorithms. Computers & Operations Research 39, 2258–2264 (2012)
3. Bożejko W., Uchroński M.: A Neuro-Tabu Search Algorithm for the Job Shop Problem. In: L. Rutkowski et al. (Eds.), Proceedings of the ICAISC 2010, Lecture Notes in Artificial Intelligence No. 6114, Springer, 387–394 (2010)
4. Bushee D.C., Svestka. J.A.: A bi-directional scheduling approach for job shops. International Journal of Production Research 37(16), 3823–3837 (1999)
5. Diaconis P.: Group Representations in Probability and Statistics. Lecture Notes - Mono-graph Series Vol. 11, Institute of Mathematical Statistics, Harvard University (1988).
6. Grabowski J., Wodecki M.: A very fast tabu search algorithm for the job shop problem. w: C. Rego, B. Alidaee (Eds.), Adaptive memory and evolution, tabu search and scatter search. Kluwer Academic Publishers, Dordrecht (2005)
7. Hasegawa M., T. Ikeguchi, K. Aihara, Exponential and chaotic neuro–dynamical tabu searches for quadratic assignment problems. Control and Cybernetics 29, 773–788 (2000)
8. Holthaus O., Rajendran C.: Efficient jobshop dispatching rules: further developments. Production Planning and Control 11, 171–178 (2000)
9. Jain A.S., Rangaswamy B., Meeran S.: New and stronger job-shop neighborhoods: A focus on the method of Nowicki and Smutnicki (1996). Journal of Heuristics 6(4), 457-–480 (2000)
10. Mattfeld D.C., Bierwirth C.: An efficient genetic algorithm for job shop scheduling with tardiness objectives. European Journal of Operational Research 155(3), 616–630 (2004)
11. Nowicki E., Smutnicki C.: An advanced tabu search algorithm for the job shop problem. Journal of Scheduling 8(2), 145–159 (2005)
12. Pezzella F., Merelli E.: A tabu search method guided by shifting bottleneck for the job-shop scheduling problem. European Journal of Operational Research 120, 297–310 (2000)
13. Solimanpur M., P. Vrat, R. Shankar, A neuro–tabu search heuristic for the flow shop scheduling problem. Computers and Operations Research 31, 2004, 2151–2164.
14. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 278–285 (1993)
15. Voß S.: Tabu search: Applications and prospects. In: D.Z. Du and P.M. Pardalos (eds.), Network Optimization Problems, pp. 333–353. World Scientific Publishing Co., Singapore (1993)

Dr hab. Wojciech Bożejko
Instytut Informatyki, Automatyki
i Robotyki
Politechniki Wrocławskiej
50-372 Wrocław, ul. Janiszewskiego 11-17
tel./fax: 71 320 27 45 / 71 321 26 77
e-mail: wojciech.bozejko@pwr.wroc.pl

Dr hab. Mieczysław Wodecki
Instytut Informatyki
Uniwersytet Wrocławski
50-383 Wrocław, Joliot-Curie 15
tel./fax: 71 325 12 71 / 71 375 62 44
e-mail: mwd@ii.uni.wroc.pl

Mgr inż. Mariusz Uchroński
Instytut Informatyki, Automatyki i Robotyki
Politechniki Wrocławskiej
50-372 Wrocław, ul. Janiszewskiego 11-17
tel./fax: 71 320 27 45 / 71 321 26 77
e-mail: mariusz.uchronski@pwr.wroc.pl