RÓWNOLEGŁY ASYNCHRONICZNY ALGORYTM TABU SEARCH DLA PROBLEMU GNIAZDOWEGO SZEREGOWANIA ZADAŃ

Wojciech BOŻEJKO, Mieczysław WODECKI

Streszczenie: W pracy rozpatrujemy problem gniazdowy z kryterium minimalizacji czasu zakończenia wykonywania zadań, który w literaturze jest oznaczany przez $J||C_{max}$. Problem ten jest jednym z najtrudniejszych problemów kombinatorycznych i należy do klasy problemów *silnie NP-trudnych*. Przedstawiamy algorytm równoległy jego rozwiązywania oparty na metodzie poszukiwania z (ang. *tabu search*, w skrócie *TS*), w którym wykorzystano ideę zrównoleglenia opartą na asynchronicznej komunikacji pomiędzy równolegle działającymi procesami - przeszukującymi przestrzeń rozwiązań.

Słowa kluczowe: problem gniazdowy, przeszukiwanie z tabu, algorytm równoległy.

1. Wstęp

Problemem gniazdowy (ang. *job-shop problem*), można w skrócie przedstawić następująco. Dany jest zbiór zadań i zbiór maszyn. Każde zadanie jest ciągiem operacji, które należy wykonywać kolejno, bez przerywania, na maszynie odpowiedniej dla każdej operacji, w określonym czasie. W dowolnym momencie maszyna może wykonywać co najwyżej jedną operację. Problem ten polega na wyznaczeniu harmonogramu (przydziale operacji przedziału czasu do jej wykonywania na odpowiedniej maszynie), minimalizującego czas wykonywania wszystkich zadań. Należy on do klasy problemów silnie *NP-trudnych* i jest uważany za jeden z klasycznych problemów optymalizacji kombinatorycznej. Podstawowe wyniki badań dotyczące problemu gniazdowego są zamieszczone w pracach: Grabowskiego [2] Carliera i Pinsona [1], Nowickiego i Smutnickiego [5], Vaessensa i in. [9], Pezelli i Merelli [7] oraz Grabowskiego i Wodeckiego [3].

2. Sformułowanie problemu

Korzystając z definicji i oznaczeń stosowanych przez Nowickiego i Smutnickiego w pracy [10], problem gniazdowy można zdefiniować następująco:

PROBLEM: Dany jest zbiór zadań $J = \{1, 2, \ldots, n\}$, które należy wykonać na maszynach ze zbioru $P = \{1, 2, \ldots, m\}$. 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 $O = \{1, 2, \ldots, o\}$ 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. $$E(\pi)=\bigcup_{k=1}^{m}\bigcup_{i=1}^{|O^k|-1}\left\{\left(\pi_k(i),\pi_k(i+1)\right)\right\}.$$
   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 $L(s,c)$.

Niech $C(s,c)=(s,v_1,v_2,\ldots,v_w,c)$, gdzie $v_i \in O$ oraz $1 \le i \le w$ będzie drogą krytyczną w gafie $G(\pi)$ z wierzchołka początkowego s do końcowego c. Drogę tą można rozbić na rozłączne podciągi wierzchołków (subpermutacje permutacji π)

$$B=[B^1,B^2,\ldots,B^r],$$

zwane ***blokami*** w permutacji π lub na drodze krytycznej $C(s,c)$ (zobacz Grabowski [2] oraz Grabowski, Nowicki i Smutnicki [5]), przy czym

1) $B^k=(\pi(a^k),\pi(a^k+1),\ldots,\pi(b^k-1),\pi(b^k)), 1 \le a^k \le b^k \le m,\ k=1,2,\ldots,r$,
2) B^k, $k=1,2,\ldots,r$ zawiera operacje wykonywane na tej samej maszynie,
3) dwa dowolne bloki zawierają operacje wykonywane na różnych maszynach.

Operacje $\pi(a^k)$ i $\pi(b^k)$ w bloku B^k są nazwane odpowiednio *pierwszą* i *ostatnią*. W pracach [2],[3] udowodniono twierdzenie zwane ***właściwościami eliminacyjnymi*** bloków.

Twierdzenie 1. Niech $B=[B^1,B^2,\ldots,B^r]$ będzie rozbiciem na bloki drogi krytycznej w acyklicznym grafie $G(\pi)$, $\pi \in \Phi$. Jeżeli $G(\omega)$ jest dopuszczalny i został wygenerowany z $G(\pi)$ przez zmianę kolejności elementów w π oraz $C_{\max}(\omega) < C_{\max}(\pi)$, to w $G(\omega)$

i) przynajmniej jedna operacja z pewnego bloku B^k, $k \in \{1,2,\ldots,r\}$ poprzedza pierwszy element $\pi(a^k)$ tego bloku, lub

ii) przynajmniej jedna operacja z pewnego bloku B^k, $k \in \{1,2,\ldots,r\}$ występuje za ostatnim elementem $\pi(b^k)$ tego bloku.

+

Wobec tego, zmiana kolejności operacji w dowolnym bloku nie generuje rozwiązania o mniejszej wartości funkcji celu. Fakt ten będzie wykorzystywany do eliminowania elementów zbędnych (nie dających poprawy wartości funkcji celu.

3. Metoda przeszukiwania z tabu

Poszukiwania z tabu jest jedną z najbardziej efektywnych metod wykorzystujących techniki lokalnych poszukiwań do wyznaczania rozwiązań suboptymalnych dla problemów optymalizacji kombinatorycznej. Jednym z zasadniczych elementów algorytmów opartych na tej metodzie jest otoczenie. Procedura jego generowania oraz przeszukiwania ma decydujący wpływ na tempo zbieżności oraz czas obliczeń algorytmu.

3.1. Generowanie oraz przeszukiwanie otoczeń

W algorytmach *TS* zamieszczonych w literaturze jest stosowane wiele typów ruchów generujących otoczenie. Z Twierdzenia 1 wynika, że ruch typu wstaw (ang. *insert, i -ruch*) jest najbardziej naturalnym sposobem generowania otoczenia dla problemu gniazdowego. Generalnie, ruch ten polega na przestawieniu pewnego elementu w permutacji $\pi \in \Phi$ na inną pozycję. Niech $\pi(x),\pi(y)$ ($\pi(x),\pi(y) \in O$ oraz $x \ne y$) będzie parą operacji wykonywanych na tej samej maszynie. Para ta definiuje ruch typu wstaw, i_y^x (przekształcenie generujące nową permutację $i_y^x(\pi)=\pi_y^x$ z π) polegający na usunięciu operacji $\pi(x)$ z jej pozycji i wstawienie na pozycję bezpośrednio za (lub przed) operację

$\pi(y)$ w zależności od tego, czy w π operacja $\pi(x)$ jest przed lub po $\pi(y)$. Graf wygenerowany przez ruch i_y^x odpowiadający wygenerowanej permutacji oznaczmy przez $G(\pi_y^x)$.

Otoczeniem permutacji $\pi \in \Phi$, generowanym przez i*-ruchy* z pewnego ustalonego zbioru ruchów dopuszczalnych $M(\pi)$ jest zbiór permutacji

$$N(M(\pi)) = \{\pi_y^x : i_y^x \in M(\pi)\}.$$

Ponieważ każdemu rozwiązaniu dopuszczalnemu odpowiada pewien graf skierowany, stąd

$$N(M(\pi)) = \{G(\pi_y^x) : i_y^x \in M(\pi)\}.$$

Definicja ruchu oraz ustalenie zbioru ruchów generujących otoczenia jest jednym z podstawowych elementów algorytmu *TS*. Zbiór ruchów nie powinien być ani za ,,duży" ani też zbyt ,,mały". Niech

$$B^k = (\pi(a^k), \pi(a^k+1), \ldots, \pi(b^k-1), \pi(b^k)),$$

będzie k-tym blokiem rozbicia $B = [B^1, B^2, \ldots, B^r]$ drogi krytycznej w acyklicznym grafie $G(\pi)$. Przez $M^{in}(B^k)$ oznaczmy zbiór *ruchów wewnętrznych* tj. ruchów, które są wykonywane wewnątrz tego bloku, czyli na operacjach $\pi(a^k+1), \ldots, \pi(b^k-1)$. Wobec tego

$$M^{in}(B^k) = \{i_y^x : x, y \in \{a^k+1, \ldots, b^k-1\} \wedge x \neq y\}.$$

Wszystkie takie ruchy tworzą zbiór

$$M^{in}(\pi) = \bigcup_{k=1}^{r} M^{in}(B^k),$$

ruchów wewnętrznych w permutacji π.

Bezpośrednio z Twierdzenia 1 wynika poniższa własność, która jest podstawą do wyznaczania subotoczeń, tj. eliminowania zbędnych ruchów (Grabowski[2]).

Własność 1. Jeżeli acykliczny graf $G(\omega)$ został wygenerowany z pewnego $G(\pi)$, $\pi \in \Phi$ przez wykonanie ruchu ze zbioru $M^{in}(\pi)$, to $C_{\max}(\omega) \geq C_{\max}(\pi)$. +

Ruchy ze zbioru $M^{in}(\pi)$ można więc pominąć, bowiem po wykonaniu każdego z nich nie ma natychmiastowej poprawy wartości funkcji celu.

Dla dowolnej operacji z pewnego bloku $B^k = (\pi(a^k), \pi(a^k+1), \ldots, \pi(b^k-1), \pi(b^k))$ $(k = 1, 2, \ldots, r)$ rozbicia B permutacji π rozpatrujemy co najwyżej jeden ruch na prawo (za blok) i co najwyżej jeden na lewo (przed blok). Definiujemy więc następujące zbiory ruchów na prawo (***za blok***) $M_{af}^k(\pi) = \{i_{b^k}^x : \pi(x) \in B^k \setminus \{\pi(b^k)\}\}$ i zbiór ruchów na lewo (***przed blok***) $M_{bf}^k(\pi) = \{i_{a^k}^x : \pi(x) \in B^k \setminus \{\pi(a^k)\}\}$. Zbiór $M_{af}^k(\pi)$ zawiera wszystkie ruchy operacji z B^k na pozycję bezpośrednio za ostatnią operację $\pi(b^k)$. Podobnie, zbiór $M_{bf}^k(\pi)$ zawiera wszystkie ruchy operacji z B^k na lewo, bezpośrednio przed pierwszą operację $\pi(a^k)$. W algorytmie *TS*, przy generowaniu otoczenia będziemy korzystali ze zbioru ruchów

$$M(\pi) = \bigcup_{k=1}^{r} (M_{af}^k(\pi) \cup M_{bf}^k(\pi)).$$

3.2. Badanie dopuszczalności rozwiązań

Ponieważ dla problemu gniazdowego nie wszystkie permutacje $\pi \in \Phi$ są dopuszczalne, więc do otoczenia $N(M(\pi))$ mogą należeć permutacje niedopuszczalne (tj. takie, których generowane grafy zawierają cykle). Łatwo zauważyć, że jeżeli ruch $i^x_{b^k} \in M^k_{af}(\pi)$ (lub $i^x_{a^k} \in M^k_{bf}(\pi)$) zawiera przyległe pary operacji, tj. $\pi(x) = \pi(b^k - 1)$ (lub $\pi(x) = \pi(a^k + 1)$), wówczas graf wygenerowany przez ten ruch jest zawsze acykliczny. W dalszej części przedstawiono warunki, które muszą być spełnione, aby z acyklicznego grafu $G(\pi)$, po wykonaniu ruchu $i^x_{b^k} \in M^k_{af}(\pi)$, $\pi(x) \neq \pi(b^k - 1)$ (lub $i^x_{a^k} \in M^k_{bf}(\pi)$, $\pi(x) \neq \pi(a^k + 1)$) otrzymać graf acykliczny.

Dla operacji $v \in O$, przez $\alpha(v)$ oraz $\gamma(v)$ oznaczamy odpowiednio poprzednika i następnika (jeżeli istnieją) v w porządku technologicznym. Podobnie, przez $\beta(v)$ i $\delta(v)$ oznaczamy odpowiednio poprzednika i następnika (jeśli istnieją) v na maszynie.

Twierdzenie 2. Jeżeli $G(\pi^x_{b^k})$ został wygenerowany z acyklicznego grafu $G(\pi)$ przez wykonanie ruchu $i^x_{b^k} \in M^k_{af}(\pi)$, $1 \le k \le r$, przy czym $\pi(x) \neq \pi(b^k - 1) \wedge \gamma(\pi(x)) \neq \alpha(\pi(b^k))$ oraz w $G(\pi)$ zachodzi

$$L(\pi(b^k), c) + min\{p_{\alpha(\pi(b^k))}, p_{\pi(b^k - 1)}\} + p_{\gamma(\pi(x))} > L(\gamma(\pi(x)), c), \quad (1)$$

to graf $G(\pi^x_{b^k})$ jest acykliczny. +

Przez analogię otrzymujemy podobne twierdzenie

Twierdzenie 3. Jeżeli $G(\pi^x_{a^k})$ został wygenerowany z acyklicznego grafu $G(\pi)$ przez wykonanie ruchu $i^x_{a^k} \in M^k_{bf}(\pi)$, przy czym $(x \neq a^k + 1,\ 1 \le k \le r)$, $\alpha(\pi(x)) \neq \gamma(\pi(a^k))$ oraz w $G(\pi)$ zachodzi

$$L(s, \pi(a^k)) + min\{p_{\gamma(\pi(a^k))}, p_{\pi(a^k + 1)}\} + p_{\alpha(\pi(x))} > L(s, \alpha(\pi(x))), \quad (2)$$

to graf $G(\pi^x_{a^k})$ jest acykliczny. +

Niech

$$\overline{M}^k_{af}(\pi) = \{i^x_{b^k} \in M^k_{af}(\pi):\ [x = b^k - 1] \vee\ [(\text{graf } G(\pi^x_{b^k}) \text{ spełnia (1)}) \wedge (\gamma(\pi(x)) \neq \alpha(b^k))]\},$$

$$\overline{M}^k_{bf}(\pi) = \{i^x_{a^k} \in M^k_{af}(\pi):\ [x = a^k + 1] \vee\ [(\text{graf } G(\pi^x_{a^k}) \text{ spełnia (2)}) \wedge (\alpha(\pi(x)) \neq \gamma(a^k))]\},$$

będą zbiorami ruchów z $M^k_{af}(\pi)$ i $M^k_{bf}(\pi)$, których wykonanie generuje z $G(\pi)$ acykliczny graf (rozwiązanie dopuszczalne). Ostatecznie, do generowania otoczenia w algorytmie tabu będą stosowane ruchy ze zbioru

$$\overline{M}(\pi) = \bigcup_{k=1}^{r} (\overline{M}^k_{af}(\pi) \cup \overline{M}^k_{bf}(\pi)).$$

Obliczenie wartości funkcji celu, dla wygenerowanego przez ruch rozwiązania dopuszczalnego, sprowadza się do wyznaczenia drogi krytycznej w grafie. Wykonanie tych obliczeń wymaga sporo czasu, dlatego będą one liczone równolegle.

3.3. Elementy metody poszukiwania z tabu

Ruchy zakazane – a właściwie pewne ich pewne atrybuty – są pamiętane na liście LT w postaci uporządkowanych par operacji. Jest ona zorganizowana na zasadzie kolejki *FIFO*. W chwili startu algorytmu lista LT jest pusta. Jeżeli graf $G(\pi^x_{b^k})$ (lub $G(\pi^x_{a^k})$) jest generowany z $G(\pi)$ przez wykonanie ruchu $i^x_{b^k} \in \overline{M}^k_{af}(\pi)$ (lub $i^x_{a^k} \in \overline{M}^k_{bf}(\pi)$), to parę operacji $(\delta(\pi(x)),\pi(x))$ (lub para $(\pi(x),\beta(\pi(x)))$ reprezentująca ten ruch umieszcza się na LT. Przed dodaniem nowej pary sprawdza się, czy lista nie jest pełna. Jeżeli tak, to jest z niej usuwany najstarszy element.

Dla grafu $G(\pi)$, ruch $i^x_{b^k} \in \overline{M}^k_{af}(\pi)$ (lub ruch $i^x_{a^k} \in \overline{M}^k_{bf}(\pi)$) jest ***zakazany***, jeżeli $\Gamma^-(\pi(x)) \cap B^k \neq \emptyset$ (lub $\Gamma^+(\pi(x)) \cap B^k \neq \emptyset$), gdzie

$$\Gamma^-(\pi(x)) = \{u \in O : (\pi(x),u) \in LT\}, \quad \Gamma^+(\pi(x)) = \{u \in O : (u,\pi(x)) \in LT\}.$$

Zbiór $\Gamma^-(\pi(x))$ (lub zbiór $\Gamma^+(\pi(x))$) zawiera operacje, które będą wykonywane ***po*** (lub ***przed***) operacją $\pi(x)$, biorąc pod uwagę aktualną zawartość listy LT.

Pomimo stosowania w algorytmie listy tabu, pojedynczy ruch do przodu może doprowadzić do „ugrzęźnięcia" w minimum lokalnym. Dlatego też stosujemy perturbacje, wykonując odpowiednie złożenia ruchów. Częściowo eliminują one tę wadę tradycyjnych algorytmów *TS*. Ogólna idea perturbacji sprowadza się do wyznaczenia wielu odpowiednich ruchów, które można wykonać jednocześnie w pojedynczej iteracji algorytmu.

Przyśpieszanie obliczeń

Definiujemy zbiór ruchów, które mogą przyśpieszyć tempo zbieżności, a jednocześnie przybliżyć obliczenia od bieżącego minimum lokalnego. Niech η_k będzie takie, że

$$C_{\max}(\eta_k(\pi)) = \min\{\min\{C_{\max}(m(\pi) : m \in \overline{M}^k_{af}(\pi)\}, \min\{C_{\max}(m(\pi) : m \in \overline{M}^k_{bf}(\pi)\}$$

oraz

$$M^{(-)} = \{\eta_k \in M : \ C_{\max}(\eta_k(\pi)) < C_{\max}(\pi)\} = \{\eta_1,\eta_2,\ldots,\eta_z\}.$$

Z Twierdzenia 1 wynika, że wykonując ruch $\eta \in M^{(-)}$ można wygenerować graf $G(\eta(\pi))$ „lepszy" niż $G(\pi)$. Dlatego też w ramach perturbacji wykonujemy jednocześnie **wszystkie** ruchy ze zbioru $M^{(-)}$, czyli ruch złożony $R(\eta_1,\eta_2,\ldots,\eta_z)$) generując graf $G(R(\pi))$. Poszczególne ruchy ze zbioru $M^{(-)}$ są związane z różnymi blokami grafu $G(\pi)$ i dzięki temu graf $G(R(\pi))$ jest acykliczny. Z ruchu złożonego, każdego para operacji jest dodawana do listy LT. Perturbacja przyśpieszająca jest wykonywana, gdy co najmniej przez *MaxretP* kolejnych iteracji nie ma poprawy najlepszej bieżącej wartości funkcji celu.

Rozpraszanie obliczeń

Znaczne oddalenie się od bieżącego minimum, w sensie wartości funkcji celu, daje nam tzw. silnie oddalająca perturbacja będąca złożeniem ruchów „pogarszających". Wykonujemy ją wówczas, gdy przez pewną liczbę kolejnych iteracji nie można wykonać perturbacji przyśpieszającej. Niech γ_k będzie takie, że:

$$C_{\max}(\gamma_k(\pi)) = \max\{\max\{C_{\max}(m(\pi) : m \in \overline{M}^k_{af}(\pi)\}, \max\{C_{\max}(m(\pi) : m \in \overline{M}^k_{bf}(\pi)\}$$

oraz

$$M^{(+)} = \{\gamma_k \in M : C_{\max}(\gamma_k(\pi)) > C_{\max}(\pi)\} = \{\gamma_1, \gamma_2, \ldots, \gamma_t\},$$

będzie zbiorem ruchów generujących z otoczenia π permutacje ,,gorsze" od π. Złożenie tych ruchów daje znaczne oddalenie się (przede wszystkim ze względu na wartość funkcji celu) od bieżącego minimum lokalnego. Tę perturbację wykonujemy wówczas, gdy w dwóch kolejnych iteracjach algorytmu nie wykonalno perturbacji poprawiającej. Na listę LT zapisujemy atrybuty najgorszego z ruchów.

Schemat algorytmu przeszukiwania z tabu

Niech $\pi \in \Pi$ będzie dowolną permutacją, LT listą tabu, a π^* najlepszym do tej pory znalezionym rozwiązaniem (na początek przyjmujemy za π^* permutację π).

Krok 1.

Jeżeli należy wykonać perturbację, to idź do Kroku 5;

Wyznaczyć otoczenie $N(\pi)$ permutacji π nie zawierające elementów zabronionych przez listę LT ;

Krok 2.

Znaleźć permutację $\delta \in N(\pi)$ taką, że:

$C_{\max}(\delta) = \min\{ C_{\max}(\beta) : \beta \in N(\pi) \}$;

Krok 3.

Jeśli $C_{\max}(\delta) < C_{\max}(\pi^*)$, to $\pi^* \leftarrow \delta$;

Umieść atrybuty δ na liście LT; $\pi \leftarrow \delta$;

Krok 4.

Jeżeli **Warunek_Zakończenia**, to EXIT,

a w przeciwnym przypadku idź do Kroku 1.

Krok 5.

Wykonaj odpowiednio perturbację przyśpieszajacą lub rozpraszającą

oraz idź do Kroku 1;

Asynchroniczny algorytm równoległy tabu dla problemu gniazdowego

Wielościeżkowy równoległy algorytm tabu do rozwiązania problemu gniazdowego został przedstawiony przez Taillarda [8]. W jego implementacji zastosowano metodę równoległych niezależnych wątków poszukiwań, dzięki czemu uzyskano skrócenie czasu obliczeń. W pracy proponujemy algorytm asynchroniczny, który pozwala nie tylko na przyspieszenie obliczeń, ale także poprawę wartości wyznaczanych rozwiązań.

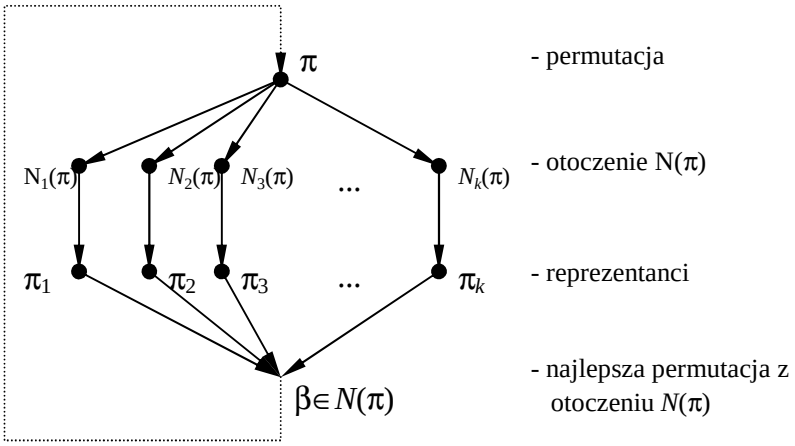
Niech $N_i(\pi)$, i=1,2,...,n będzie podzbiorem otoczenia $N(\pi)$ permutacji π zawierającym permutacje generowane z π przez przestawienie zadania z pozycji i na inną pozycję. Otocznie $N(\pi)$ można więc przedstawić jako sumę

$$N(\pi) = \bigcup_{i=1}^{n} N_i(\pi).$$

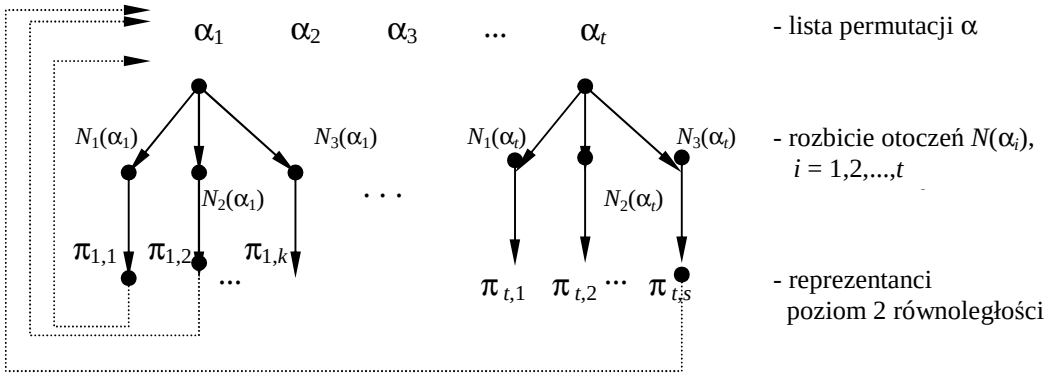
Rozbicie to umożliwia równoległe przeglądanie zbioru $N(\pi)$. W pierwszej kolejności wybieramy najlepszy element (reprezentanta) każdego podzbioru $N_i(\pi)$, i=1,2,...,n, a następnie wybieramy najlepszego z reprezentantów.

W pracy proponujemy równoległy hybrydowy algorytm tabu search. Stosujemy rozbicie otoczenia na podzbiory przy wyznaczaniu najlepszego elementu. Taki równoległy algorytm *PSTS* (ang. *parallel synchronous tabu search*, równoległy synchroniczny algorytm poszukiwania z tabu) jest w klasyfikacji Voss'a oznaczony symbolem *SPSS* (ang. *single starting point single strategy*, ten sam punkt startowy ta sama strategia wszystkich wątków), rys. 1.

Elementem hybrydowym proponowanego algorytmu *PATS* (ang. *parallel asynchronous tabu search*, równoległy asynchroniczny algorytm poszukiwania z tabu, rys. 2), nie występującym w klasycznym równoległym poszukiwaniu tabu, jest lista $\alpha = (\alpha_1, \alpha_2,..., \alpha_s)$ bieżących rozwiązań. W algorytmie klasycznym są jedynie pojedyncze rozwiązania. Lista α nawiązuje do listy *OPEN*, występującej w innym algorytmie heurystycznego poszukiwania, a mianowicie w algorytmie A*. Algorytm *PATS* działa bazując na dwóch poziomach zrównoleglania. Poziom pierwszy odpowiada równoległemu przeglądaniu otoczeń bieżących rozwiązań znajdujących się na liście α. Otoczenia rozwiązań $\alpha_1, \alpha_2,..., \alpha_s$ są badane równolegle poprzez podział otoczenia na podzbiory i równoległe wyznaczanie reprezentantów, co stanowi drugi, głębszy poziom zrównoleglenia algorytmu.



Rys. 1. Schemat działania algorytmu *PSTS*



Rys. 2. Schemat działania algorytmu *PATS*

Schemat działania równoległego algorytm tabu dla modelu obliczeń równoległych EREW PRAM podajemy poniżej:

Główny procesor

- Zleć procesorom podrzędnym poziomu 1 równoległe wyznaczanie reprezentantów permutacji z listy α;

- pobierz reprezentantów od procesorów podrzędnych poziomu 2 (asynchronicznie, gdy tylko się pojawią);
- odrzuć reprezentantów zabronionych przez listę tabu, chyba że ich wartość funkcji celu jest mniejsza od najlepszej znalezionej do tej pory;
- dodaj reprezentantów do listy α (zamiast elementów najgorszych, gdy lista jest już zapełniona).

Procesory podrzędne na 1 poziomie zrównoleglenia

- Pobierz permutację α_i z listy α procesora głównego;
- podziel otocznie $N(\alpha_i)$ na podzbiory $N_j(\alpha_i)$, j=1,2,...,n;
- zleć procesorom podrzędnym poziomu 2 równoległe wyznaczenie reprezentantów w podzbiorach $N_j(\alpha_i)$, j=1,2,...,n.

Procesory podrzędne na 2 poziomie zrównoleglenia

- Pobierz podzbiór $N_j(\alpha_i)$ permutacji α_i od procesora podrzędnego z poziomu 1;
- wyznacz reprezentanta – permutację π_{ji} otocznia $N_j(\alpha_i)$;
- wyślij π_{ji} do procesora głównego.

Dla długości $|\alpha|$ =1 listy α powyższy schemat jest równoważny z synchronicznym algorytmem tabu *PSTS*. Natomiast, dla $|\alpha| > 1$ otrzymamy asynchroniczny algorytm tabu *PATS*, ponieważ procesory podrzędne pracują niezależnie i nie są zsynchronizowane.

4. Wyniki obliczeniowe

Obliczenia wykonano na 80 powszechnie znanych przykładach danych testowych oznaczanych przez TA1-TA80 o ośmiu rozmiarach, $n \times m = 15\times15,\ 20\times15,\ 20\times20,\ 30\times15,\ 30\times20,\ 50\times15,\ 50\times20,\ 100\times20$ i różnym stopniu trudności. Są one zamieszczone na stronie ORLib [6]. Dla tej grupy danych znane są rozwiązania optymalne jedynie dla 32 przykładów. W Tablicy 1 przedstawiamy wyniki porównawcze rozwiązań wyznaczonych przez opisany w poprzednim rozdziale algorytm *PATS*, z najlepszym obecnie znanym algorytm *TSOSGW* zamieszczonymi w pracy Grabowskiego i Wodeckiego [6].

Tab. 1. Średni błąd względny δ_{aprd} oraz średni czas obliczeń t_{aprd} dla 300n iteracji.

Problem	$n\times m$	*TSOSGW*		*PATS*(1)*	
		δ_{aprd}	t_{aprd}	δ_{aprd}	t_{aprd}
TA01-10	15×15	0.27	7.6	0.39	7.0
TA11-20	20×15	3.87	7.1	4.52	6.3
TA21-30	20×20	6.31	0.4	8.66	0.3
TA31-40	30×15	1.75	20.1	3.07	16.7
TA41-50	30×20	5.82	9.9	7.38	7.9
TA51-60	50×15	0.01	4.1	0.09	3.8
TA61-70	50×20	0.36	9.2	0.71	8.2
TA71-80	50×20	0.00	4.4	0.04	4.1
średnio		2.30	8.8	3.11	6.8

* wyniki obliczeń na jednym procesorze.

Ponieważ *PATS* jest uproszczoną wersją bardzo dobrego algorytmu *TSOSGW*, stąd

średni błąd względny wyznaczanych przez ten algorytm rozwiązań jest znacznie większy. Pominięcie pewnych czasochłonnych modułów pozwoliło z kolei na skrócenie czasu obliczeń. Wstępne testy wykonane z wykorzystaniem większej liczby procesorów wskazują na znaczna poprawę parametrów algorytmu.

5. Podsumowanie

W pracy przedstawiliśmy konstrukcję algorytmu równoległego rozwiązywania permutacyjnego problemu gniazdowego opartą na metodzie przeszukiwania z tabu. Zastosowaliśmy ideę asynchronicznej komunikacji pomiędzy procesorami, pozwalającą zwiększyć efektywność działania algorytmu. W pełni potwierdziły to eksperymenty obliczeniowe wykonane na przykładach zaczerpniętych z literatury. Otrzymane wyniki wskazują, że zaprezentowane podejście, uzupełnione o inne nowoczesne metody dywersyfikacji i intensyfikacji obliczeń, może stać się bazą nowych bardzo efektywnych równoległych algorytmów metaheurystycznych, które w pełni będą wykorzystywały potencjał istniejących systemów obliczeń równoległych.

Literatura

1. Carlier J., Pinson E.: An Algorithm for Solving the Job Shop Problem, Management Science, 35, 1989, 164-176.
2. Grabowski J.: Uogólnione zagadnienie optymalizacji kolejności operacji w dyskretnych systemach produkcyjnych, Prace Naukowe ICT PWr, seria: Monografie, 1979.
3. Grabowski J., Nowicki E., Smutnicki C.: Block algorithm for scheduling of operations in job-shop system, Przeglad Statystyczny, 35, 1988, 67-80.
4. Grabowski J., Wodecki M.: A very fast tabu search algorithm for the job shop problem, in: Rego C., Alidaee B., editors. Adaptive memory and evolution; tabu search and scatter search, Dordrecht, Kluwer Academic Publishers, 2005, 117-144.
5. Nowicki E., Smutnicki C.: A Fast tabu serach algorithm for the job shop problem, Management Scence, 42, 1996, 797-813.
6. OR Library http://www.ms.ic.ac.uk/jeb/ orlib/schinfo.html
7. Pezzella F., Merelli E.: A tabu search method guided by shifting bottleneck for the job-shop scheduling problem, EJOR, 2000, 297-310.
8. Taillard E.: Parallel taboo search techniques for the job shop scheduling problem, ORSA Journal on Computing, 6, 1994, 08-117.
9. Vaessens R., Aarts E., Lenstra J.K.: Job shop scheduling by local search, Informs Journal of Computing, 8, 1996, 303-317.

Dr Wojciech BOŻEJKO
Instytut Informatyki Automatyki
i Robotyki Politechniki Wrocławskiej
ul. Janiszewskiego 11/17, 50-372 Wrocław
Wyższa Szkoła Zarządzania „Edukacja”
we Wrocławiu
e-mail: wojciech.bozejko@pwr.wroc.pl

Dr Mieczysław WODECKI
Instytut Informatyki Uniwersytetu
Wrocławskiego
ul. Joliot-Curie 50-383 Wrocław
Wyższa Szkoła Zarządzania „Edukacja” we
Wrocławiu
e-mail: mwd@ii.uni.wroc.pl