ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1996
Seria: AUTOMATYKA z.118 Nr kol. 1338

Eugeniusz NOWICKI
Politechnika Wrocławska

ZASTOSOWANIE TECHNIKI TABU SEARCH DO HARMONOGRAMOWANIA ELASTYCZNYCH GNIAZD PRODUKCYJNYCH

Streszczenie. W pracy przedstawia się szybki aproksymacyjny algorytm oparty na technice tabu search, dla problemu szeregowania zadań w elastycznych gniazdach produkcyjnych z kryterium będącym czasem wykonania wszystkich zadań. Specjalna metoda implementacji istotnie poprawia lokalne przeszukiwanie i zwiększa prędkość algorytmu.

APPLICATION OF TABU SEARCH TECHNIQUE TO SCHEDULING A FLEXIBLE JOB SHOP

Summary. A fast approximation algorithm for a problem of finding a minimum makespan in a flexible job shop is presented. The algorithm is based on a tabu search technique. A special method of implementation improves the local search significantly and increases the speed of the algorithm.

1. Wstęp

W niniejszej pracy bada się problem harmonogramowania elastycznych gniazd produkcyjnych z kryterium minimalizującym moment zakończenia wykonywania wszystkich zadań produkcyjnych. Przyjmuje się, że w każdym gnieździe produkcyjnym (stanowisku obsługi) znajduje się określona liczba identycznych maszyn. Każde zadanie jest reprezentowane przez pewien ciąg operacji. Dla każdej operacji jest określone gniazdo produkcyjne, w którym ma być ona wykonana, i czas jej wykonania. Realizacja operacji w danym gnieździe produkcyjnym wymaga zaangażowania jednej (dowolnej) maszyny z tego gniazda. Dla uproszczenia, w pracy nie rozważa się innych elementów, takich jak: czasy transportu pomiędzy gniazdami, skończona pojemność magazynów, palety, uchwyty obróbkowe oraz wózki transportowe itp. [10]. Mimo to badany problem jest jednym z najtrudniejszych problemów szeregowania zadań na maszynach, jako że jest uogólnieniem

znanego z literatury problemu gniazdowego, uważanego powszechnie za bardzo trudny. Z punktu widzenia teorii złożoności obliczeniowej jest to problem NP-trudny, co usprawiedliwia stosowanie do jego rozwiązania algorytmów heurystycznych, tym bardziej że istniejące w literaturze algorytmy dokładne, oparte przede wszystkim na metodzie podziału i ograniczeń, pozwalają go rozwiązać w sensownym czasie tylko dla bardzo małej liczby operacji, maszyn i stanowisk.

Aktualnie badany problem jest rozwiązywany przez algorytmy heurystyczne typu konstrukcyjnego. Algorytmy te, zgodnie z pewnymi regułami priorytetowymi, szeregują kolejno pojedyńcze operacje i po uszeregowaniu wszystkich kończą swoje działanie. Czas ich pracy jest bardzo krótki, ale jakość produkowanych przez nie rozwiązań nie zawsze jest zadowalająca. W sytuacjach, kiedy możemy pozwolić sobie na zwiększenie czasu obliczeń, warto zastosować algorytmy typu popraw, a w szczególności technikę tabu search [1,2], która dla wielu podobnych zagadnień umożliwiła skonstruowanie bardzo obiecujących algorytmów (patrz np. [7,8,9] i praca przeglądowa [12] oraz [4,5]).

W pracy przedstawia się algorytm typu tabu search, który jest odpowiednią modyfikacją algorytmu zaprezentowanego w [7] dla problemu gniazdowego oraz algorytmu z [6] i [9] dla problemu przepływowego z maszynami równoległymi. Specyficzna konstrukcja otoczenia uszeregowania i wykorzystanie odpowiednich własności do "szybkiego" jego przeszukania pozwoliły znacznie zredukować czasochłonność wyboru najlepszego rozwiązania z otoczenia. W konsekwencji prezentowany algorytm istotnie poprawia początkowe rozwiązania, dostarczone z powszechnie stosowanych algorytmów konstrukcyjnych dla tego typu zagadnienia, w czasie rzędu kilku minut na IBM PC. Rozwiązywane w tym czasie problemy mają rozmiar akceptowalny w praktyce.

2. Sformułowanie problemu

Rozważany w pracy problem szeregowania można sformułować następująco: Dany jest zbiór zadań $J = \{1, 2, \ldots, n\}$, zbiór operacji $O = \{1, 2, \ldots, o\}$ oraz zbiór stanowisk obsługi $M = \{1, 2, \ldots, m\}$. Zadanie j, $j \in J$ składa się z sekwencji o_j operacji indeksowanych odpowiednio przez $(\tau_j + 1, \tau_j + 2, \ldots, \tau_j + o_j)$, które należy wykonywać w tej kolejności, gdzie $\tau_j = \sum_{i=1}^{j-1} o_i$ jest liczbą operacji w zadaniach $1, 2, \ldots, j-1$, $\tau_1 = 0$ oraz $\sum_{j \in J} o_j = o$. W każdym stanowisku obsługi l, $l \in M$ znajduje się zbiór $M_l = \{1, 2, \ldots, m_l\}$ zawierający

m_l identycznych maszyn. Operacja i, $i \in O$ powinna być wykonana w stanowisku $\mu_i \in M$ w czasie $p_i > 0$ na jednej, dowolnie wybranej maszynie z tego stanowiska. Przyjmuje się, że każda maszyna może wykonywać w danej chwili nie więcej niż jedną operację oraz wykonywanie operacji na danej maszynie nie może być przerywane. Dopuszczalne uszeregowanie jest definiowane przez zbiór par (K_i, C_i), $i \in O$ takich, że powyższe ograniczenia są spełnione, gdzie $K_i \in M_{\mu_i}$ jest maszyną, która wykonuje operację i w stanowisku μ_i, zaś C_i momentem zakończenia wykonywania tej operacji w tym stanowisku. Problem polega na znalezieniu dopuszczalnego uszeregowania minimalizującego czas wykonania wszystkich operacji $\max_{i \in O} C_i$.

Przedstawiony problem różni się tym od klasycznego problemu gniazdowego (notowanego za Grahamem jako $J \,||\, C_{max}$), że w danym gnieździe produkcyjnym może znajdować się kilka maszyn. Powoduje to, że należy dodatkowo wyznaczyć dla każdej operacji maszynę, na której będzie ona wykonywana.

W celu sformułowania pełnego modelu matematycznego powyższego zadania wygodnie jest, podobnie jak w [6-9], posłużyć się pewnym skierowanym grafem reprezentującym uszeregowanie. Niech $O_l = \{i \in O : \mu_i = l\}$ określa zbiór operacji, które powinny być wykonane w stanowisku l, $l \in M$. W każdym stanowisku l zbiór operacji O_l należy podzielić na m_l podzbiorów O_{lk}, $k \in M_l$; operacje ze zbioru O_{lk} będą wykonywane na maszynie k. Kolejność wykonywania operacji ze zbioru O_{lk} określa permutacja $\pi_{lk} = (\pi_{lk}(1), \ldots, \pi(o_{lk}))$, gdzie $\pi_{lk}(w)$ oznacza operację ze zbioru O_{lk}, która występuje w w-tej pozycji w π_{lk}, zaś $o_{lk} = |O_{lk}|$. Stąd kolejność wykonywania wszystkich operacji jest określona przez permutację $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$, gdzie $\pi_l = (\pi_{l1}, \pi_{l2}, \ldots, \pi_{lm_l})$. Zbiór wszystkich takich permutacji wynikających z wszystkich możliwych rozbić zbiorów O_l oraz różnych kolejności wykonywania operacji ze zbiorów O_{lk} oznaczamy przez Π. Teraz możemy przejść do zdefiniowania zapowiedzianego wcześniej grafu. Dla permutacji π, $\pi \in \Pi$ niech $G(\pi) = (O, E^* \cup E(\pi)$ będzie skierowanym grafem ze zbiorem wierzchołków O oraz zbiorem łuków $E^* \cup E(\pi)$, gdzie $E^* = \bigcup_{j \in J} \bigcup_{i=1}^{o_j - 1} \{(\tau_j + i, \tau_j + i + 1)\}$ określa wymaganą technologiczną kolejność przechodzenia zadań przez stanowiska, zaś $E(\pi) = \bigcup_{l \in M} \bigcup_{k \in M_l} \bigcup_{w=1}^{o_{lk}-1} \{(\pi_{lk}(w), \pi_{lk}(w+1))\}$ wybraną kolejność wykonywania operacji na maszynach. Każdy wierzchołek i tego grafu ma wagę p_i, a każdy łuk wagę zero. Niech $r_i(\pi)$ oznacza wartość najdłuższej drogi dochodzącej do wierzchołka i w grafie $G(\pi)$, łącznie z tym wierzchołkiem. Oczywiście, w przypadku gdy graf $G(\pi)$ jest cykliczny, niektóre

z wartości $r_i(\pi)$ mogą być równe nieskończoności. W przeciwnym wypadku zestaw par (K_i, C_i), $i \in O$, określa najlepsze (w sensie kryterium $C_{\max}$) uszeregowanie dopuszczalne zgodne "kolejnościowo" z permutacją π, gdzie K_i jest maszyną ze stanowiska μ_i taką, że $i \in \{\pi_{\mu_i K_i}(1), \ldots, \pi_{\mu_i K_i}(o_{\mu_i K_i})\}$, zaś $C_i = r_i(\pi)$. Ostatecznie oznaczając przez $C_{\max}(\pi)$ długość najdłuższej drogi w $G(\pi)$ (ścieżka krytyczna), problem nasz można sformułować następująco: Znaleźć kolejność $\pi^* \in \Pi$ taką, że $C_{\max}(\pi^*) = \min_{\pi \in \Pi} C_{\max}(\pi)$.

Ścieżkę krytyczną w grafie $G(\pi)$ można opisać jako ciąg $u = (u_1, u_2, \ldots, u_s)$ wierzchołków tego grafu, gdzie $u_i \in O$, $1 \leq i \leq s$, a s jest długością ścieżki. Oczywiście, $C_{\max}(\pi) = \sum_{i=1}^{s} p_{u_i}$. Ścieżkę u, podobnie jak dla problemu gniazdowego, można przedstawić w postaci $u = (B_1, B_2, \ldots, B_t)$, gdzie

$$B_h = (\pi_{l_h k_h}(e_h), \pi_{l_h k_h}(e_h + 1), \ldots, \pi_{l_h k_h}(f_h)),$$

$1 \leq h \leq t$, jest ciągiem zawierającym operacje wykonywane na tej samej maszynie w pewnym stanowisku (maszynę tę i stanowisko oznaczamy odpowiednio przez k_h oraz l_h) poczynając od operacji z pozycji e_h, $1 \leq e_h \leq o_{l_h k_h}$, a kończąc na operacji z pozycji f_h, $e_h \leq f_h \leq o_{l_h k_h}$, w permutacji $\pi_{l_h k_h}$. Dodatkowo przyjmujemy, że operacje z dwóch kolejnych ciągów B_h, B_{h+1} są wykonywane w różnych stanowiskach, tzn. $l_h \neq l_{h+1}$, $h = 1, 2, \ldots, t-1$. Dalej ciąg B_h będziemy nazywać h-tym *blokiem* operacji na ścieżce krytycznej. Uwzględniając wcześniej wprowadzone pojęcia, dla każdej operacji i z bloku B_h zachodzi $\mu_i = l_h$, $i \in O_{l_h k_h}$, oraz $C_{\max}(\pi) = \sum_{h=1}^{t} \sum_{i \in B_h} p_i$.

3. Metoda tabu search

Technika tabu search została wyczerpująco opisana w wielu pracach, patrz np. [1,2]. Dlatego też tutaj ograniczymy się tylko do krótkiego jej szkicu. Podstawowym elementem tej techniki jest pojęcie ruchu, otoczenia oraz listy tabu. Przez ruch rozumiemy funkcję, która odwzorowuje jedno uszeregowanie w drugie. Dla każdego uszeregowania dopuszczalnego określony jest pewien zestaw ruchów, generujący dla niego otoczenie. Algorytm wykorzystujący technikę tabu search startuje z pewnego uszeregowania dopuszczalnego wygenerowanego przez proste algorytmy konstrukcyjne i zwanego dalej uszeregowaniem bazowym. Następnie określa się dla tego uszeregowania otoczenie, znajduje się w nim najlepsze w sensie wartości funkcji celu uszeregowanie i dalej traktuje się je jako uszere-

gowanie bazowe w następnej iteracji. W celu przeciwdziałania przed wpadnięciem w cykl, po wykonaniu ruchu na tzw. liście tabu (pamięć krótkoterminowa) zapisuje się pewne jego atrybuty. Z kolei w trakcie generowania otoczenia nie rozpatruje się ruchów, których atrybuty znajdują się na liście tabu; ruchy te są ruchami zabronionymi. Zmniejszenie restrykcyjności listy tabu osiąga się przez wprowadzanie poziomu aspiracji ruchu.

Dalej, w kolejnych rozdziałach, przedstawimy szczegółowo poszczególne elementy naszego algorytmu zwanego dalej algorytmem TS.

4. Ruch i otoczenie

W większości algorytmów, w których rozwiązanie jest określone przez permutację (lub zestaw permutacji), jako ruch przyjmuje się: (i) pobranie pewnego elementu permutacji i włożenie go przed pewnym elementem znajdującym się na określonej pozycji w tej permutacji, (ii) zamianę położenia dwóch wybranych elementów permutacji. Ruchy typu (i) umożliwiają z reguły szybsze wyznaczenia wartości funkcji celu dla otrzymanej (po wykonaniu ruchu) permutacji. Dlatego też będziemy rozważać tylko ruchy tego typu. Precyzyjnie, jako ruch $v = (l, a, x, b, y)$ zastosowany do permutacji $\pi \in \Pi$, będziemy rozumieć czynność polegającą na pobraniu operacji $z = \pi_{la}(x)$ z pozycji x na maszynie a w stanowisku l i włożeniu jej bezpośrednio przed operację z pozycji y na maszynie b w tym samym stanowisku, gdy $y \leq o_{lb}$ lub włożeniu jej bezpośrednio po operacji z pozycji o_{lb}, gdy $y = o_{lb} + 1$. W przypadku $b = a$, tzn. przesuwania operacji z w obrębie tej samej maszyny, operacje $\pi_{la}(x+1), \ldots, \pi_{la}(y-1)$ są przesuwane o jedną pozycję w lewo, jeżeli $x < y$, lub operacje $\pi_{la}(y), \ldots, \pi_{la}(x-1)$ są przesuwane o jedną pozycję w prawo, gdy $x > y$. W przypadku $b \neq a$, tzn. przesuwania operacji z na inną maszynę, operacje $\pi_{la}(x+1), \ldots, \pi_{la}(o_{la})$ są przesuwane w lewo, zaś operacje $\pi_{lb}(y), \ldots, \pi_{lb}(o_{lb})$ w prawo. W wyniku ruchu v otrzymujemy nową permutację notowaną jako π_v.

Pierwszą rzeczą, którą trzeba wziąć pod uwagę przy definiowaniu, jakie ruchy v należy wykonać z danej permutacji π, jest niedopuszczenie do powstania cyklu w grafie $G(\pi_v)$. Po drugie, należy dokonać sensownego wyboru wielkości tego zbioru ruchów, oznaczanego dalej przez $V(\pi)$. Duży zbiór $V(\pi)$ wymaga znacznych nakładów obliczeniowych na wyznaczenie ruchu dającego najlepszą wartość funkcji celu. Z kolei istotne zmniejszenie tego zbioru może uniemożliwić przeglądanie tych obszarów rozwiązań dopuszczalnych, w

których znajdują się rozwiązania optymalne lub do nich zbliżone. Dlatego też, podobnie jak w [9], wykorzystamy "eliminacyjne własności ścieżki krytecznej" do konstrukcji zbioru $V(\pi)$. Mianowicie, nie będziemy rozważać ruchów, które polegają na pobieraniu operacji nie leżących na ścieżce krytycznej, a dokładniej, nie należących do bloków operacji o liczności większej niż dwa. Usprawiedliwione jest to faktem, że dla eliminowanych ruchów $C_{\max}(\pi_v) \geq C_{\max}(\pi)$. Podobna nierówność zachodzi, jeżeli przesuwamy operacje w obrębie tej samej maszyny oraz: (*i*) przesuwamy pierwszą operację bloku w lewo, (*ii*) przesuwamy ostatnią operację bloku w prawo, (*iii*) przesuwamy operację z wnętrza bloku (tzn. operację nie pierwszą i nie ostatnią w bloku) na pozycje z wnętrza bloku. W konsekwencji proponujemy następujący zbiór ruchów:

$$V(\pi) = \bigcup_{1\leq h\leq t,\ |B_h|>1} \bigcup_{x=e_h}^{f_h} V_{hx}(\pi) \setminus \{v \in V_{hx}(\pi) : G(\pi_v) \text{ jest cykliczny}\}$$

gdzie

$$V_{hx}(\pi) = V^1_{hx}(\pi) \cup V^2_{hx}(\pi),$$

oraz

$$V^1_{hx}(\pi) = \bigcup_{b\in M_{l_h},\ b\neq k_h} \{v = (l_h, k_h, x, b, y) : 1 \leq y \leq o_{l_h b} + 1\},$$

$$V^2_{hx}(\pi) = \{v = (l_h, k_h, x, k_h, y) : y \in P_h(x)\}$$

Zbiór $V_{hx}(\pi)$ określa ruchy, które wykonujemy po pobraniu operacji $z = \pi_{l_h k_h}(x)$ z bloku B_h z pozycji x. Zbiór ten jest podzielony na dwa zbiory $V^1_{hx}(x)$, $V^2_{hx}(x)$. Pierwszy określa ruchy polegające na wkładaniu operacji z na inne maszyny niż maszyna k_h, na której była ona wykonywana, a drugi na przemieszczaniu operacji z tylko na maszynie k_h. Zbiór $P_h(x)$ oznacza pozostałe pozycje w permutacji $\pi_{l_h k_h}$ wynikające z uzględnienia warunków (*i*)-(*iii*); $P_h(e_h) = \{e_h + 2, e_h + 2, \ldots, o_{l_h k_h} + 1\}$, $P_h(f_h) = \{1, 2, \ldots, f_h - 2\}$, $P_h(x) = \{1, 2, \ldots, \min(x - 2, e_h)\} \cup \{f_h + 1, f_h + 1, \ldots, o_{l_h k_h} + 1\}$, $e_h < x < f_h$. Ze zbioru $P_h(f_h)$ została wyeliminowana pozycja $f_h - 1$, ponieważ przesunięcie operacji z przed operację z tej pozycji dawałoby taką samą permutację, co przesunięcie operacji $\pi_{l_h k_h}(f_h - 1)$ przed operację z pozycji $f_h + 1$. Dotyczy to również zbioru $P_h(e_h + 1)$ i eliminacji pozycji e_h.

Przejdziemy teraz do analizy liczności zbioru $V(\pi)$. Zauważmy, że $|V_{hx}(\pi)| \approx |O_{l_h}|$. Stąd i przyjmując dla uproszczenia, że graf $G(\pi_v)$ jest acykliczny dla każdego ruchu v, otrzymamy $|V(\pi)| \approx \sum_{h=1}^{t} |B_h||O_{l_h}|$. Ponieważ wyznaczenie $C_{\max}(\pi_v)$ (lub stwierdzenie,

że graf $G(\pi_v)$ jest cykliczny) wymaga $O(o)$ kroków, to przeglądnięcie otoczenia $\mathcal{O}(\pi) = \{\pi_v : v \in V(\pi)\}$ wymaga aż $O(o\sum_{h=1}^{t} |B_h|\,|O_{l_h}|)$ czasu. Dalej naszkicujemy metodę, która pozwala na przeglądnięcie tego otoczenia w czasie $O(o\sum_{h=1}^{t} |B_h|)$.

Wspomnianą powyżej metodę będziemy omawiać dla pewnego ustalonego bloku B_h, $1 \le h \le t$ oraz ustalonej operacji $z = \pi_{l_h k_h}(x)$, $e_h \le x \le f_h$ z permutacji $\pi_{l_h k_h}$. Połóżmy $l = l_h$ oraz $a = k_h$. Pokażemy, że przeglądnięcie całego zbioru ruchów $V_{hx}(\pi)$ (tzn. sprawdzenie, czy graf $G(\pi_v)$ jest acykliczny oraz wyliczenie $C_{\max}(\pi_v)$ dla wszystkich $v \in \hat{V}_{hx}(\pi) \stackrel{\text{def}}{=} V_{hx}(\pi) \setminus \{v \in V_{hx}(\pi) : G(\pi_v) \text{ jest cykliczny}\}$) wymaga $O(o)$ kroków.

Na początek wprowadźmy kilka dodatkowych pojęć. Niech $A(i)$ ($B(i)$) oznacza operację, która jest bezpośrednim następnikiem (poprzednikiem) technologicznym operacji i, tzn. $(i, A(i)) \in E^*$ ($(B(i), i) \in E^*$). Jeżeli operacja i nie ma następnika (poprzednika) technologicznego, to przyjmujemy $A(i) = 0$ ($B(i) = 0$). Dalej oznaczmy przez d_v (c_v) operację, która po wykonaniu ruchu $v = (l, a, x, b, y)$ z π, będzie bezpośrednim następnikiem (poprzednikiem) operacji z i jest wykonywana na maszynie b, tzn. $(z, d_v) \in E(\pi_v)$ ($(c_v, z) \in E(\pi_v)$). Z definicji π_v wynika, że $d_v = \pi_{lb}(y)$, jeżeli $y \le o_{lb}$ oraz $d_v = \pi_{lb}(o_{lb}+1) \stackrel{\text{def}}{=} 0$, gdy $y = o_{lb}+1$. Podobnie $c_v = \pi_{lb}(y-1)$, jeżeli $y > 1$ oraz $c_v = \pi_{lb}(0) \stackrel{\text{def}}{=} 0$, gdy $y = 1$.

Najpierw podamy metodę, która pozwala "szybko" znaleźć wszystkie ruchy $v \in V_{hx}(\pi)$ nie powodujące cyklu w grafie $G(\pi_v)$. Prawdziwa jest następująca własność.

Własność 1. *Niech $\pi \in \Pi$, graf $G(\pi)$ jest acykliczny oraz $v = (l, a, x, b, y) \in V_{hx}$. Graf $G(\pi_v)$ jest acykliczny wtedy i tylko wtedy gdy w grafie $G(\pi)$ nie istnieje droga z wierzchołka $A(z)$ do jednego z wierzchołków $\pi_{lb}(1), \pi_{lb}(2), \ldots, \pi_{lb}(y-1) = c_v$ oraz nie istnieje droga z jednego z wierzchołków $\pi_{lb}(y) = d_v, \pi_{lb}(y+1), \ldots, \pi_{lb}(o_{lb})$ do wierzchołka $B(z)$.*

W przypadku gdy $A(z) = 0$ lub $y = 1$ ($B(z) = 0$ lub $y = o_{lb}+1$), wymienione powyżej drogi nie istnieją,ponieważ graf $G(\pi)$ nie zawiera wierzchołka 0.

Własność 1 pozwala w czasie $O(o)$ określić wszystkie ruchy $v \in V_{hx}$, dla których graf $G(\pi_v)$ jest acykliczny. Rzeczywiście, wystarczy wyznaczyć wierzchołki $AW \in O_l$ w grafie $G(\pi)$, do których istnieje droga z wierzchołka $A(z)$ oraz wierzchołki $BW \in O_l$ w grafie $G(\pi)$, z których istnieje droga do wierzchołka $B(z)$. Można to wykonać w czasie $O(o)$. Zauważmy, że jeżeli $\pi_{lb}(w) \notin AW$, to $\pi_{lb}(w-1) \notin AW$, $w = 2, 3, \ldots, o_{lb}$. Podobnie, jeżeli $\pi_{lb}(w) \notin BW$, to $\pi_{lb}(w+1) \notin BW$, $w = 1, 2, \ldots, o_{lb}-1$. Stąd i z Własności 1 wynika, że graf $G(\pi_v)$ jest acykliczny wtedy i tylko wtedy, gdy $c_v \notin AW$ i $d_v \notin BW$. Można to

sprawdzić w czasie $O(1)$, przy odpowiedniej strukturze pamiętania zbiorów AW i BW. Ostatecznie, ponieważ $|V_{hx}(\pi)| \approx |O_l| \leq o$, otrzymujemy wspomnianą wyżej złożoność.

Przedyskutujemy teraz metodę, która pozwala wyznaczyć $C_{\max}(\pi_v)$ dla wszystkich $v \in \dot{V}_{hx}(\pi)$ też w czasie $O(o)$. Podstawową rolę będzie tutaj pełnić permutacja γ, która jest otrzymana z π przez pobranie operacji z z permutacji π_{la} i *niewłożenie jej* do żadnej innej permutacji π_{lb}. Odpowiadający jej graf $G(\gamma) = [G(\pi) \setminus \{(\pi_{la}(x-1), z), (z, \pi_{la}(x+1))\}] \cup \{(\pi_{la}(x-1), \pi_{la}(x+1))\}$. Jeżeli $x = 1$ lub $x = o_{la}$, to odpowiednie pary w powyżej równości nie występują. Na koniec przez $r_i(\gamma)$ ($q_i(\gamma)$) oznaczmy wartość najdłuższej drogi dochodzącej do wierzchołka i (wychodzącej z wierzchołka i) w grafie $G(\gamma)$ łącznie z tym wierzchołkiem, $r_0(\gamma) = 0$ ($q_0(\gamma) = 0$).

Własność 2. *Niech $\pi \in \Pi$ oraz graf $G(\pi)$ jest acykliczny. Dla każdego $v \in \dot{V}_{hx}(\pi)$ mamy*

$$C_{\max}(\pi_v) = \max\{C_{\max}(\gamma),\ \max(r_{B(z)}(\gamma), r_{c_v}(\gamma)) + p_z + \max(q_{A(z)}(\gamma), q_{d_v}(\gamma))\}.$$

Konstrukcja grafu $G(\gamma)$ z grafu $G(\pi)$ wymaga $O(1)$ czasu. Wyliczenie wartości $r_i(\gamma)$, $q_i(\gamma)$ dla $i \in O_l$ oraz wartości $C_{\max}(\gamma)$ wymaga $O(o)$. Wyznaczenie jednej wartości $C_{\max}(\pi_v)$, wykorzystując Własność 2, wymaga $O(1)$. Stąd do wyznaczenia ich wszystkich (w przybliżeniu $|O_l|$) potrzeba $O(|O_l|)$ czasu. Ostatecznie, ponieważ $|O_l| \leq o$, to wyliczenie $C_{max}(\pi_v)$ dla wszystkich $v \in \dot{V}_{hx}(\pi)$ wymaga $O(o)$ czasu.

5. Lista tabu

Na liście tabu T o ustalonej długości $MaxT$ (parametr algorytmu) będziemy zapamiętywać pewne atrybuty aktualnie wizytowanej permutacji π, które zależą od wykonanego z niej ruchu $v = (l, a, x, b, y)$. Atrybutami tymi będzie para lub dwie pary sąsiednich operacji z permutacji π_{la}. Jeżeli $b \neq a$, wtedy na listę T zapisujemy dwie pary operacji (c_v, z) oraz (z, d_v). W przypadku gdy $b = a$, zapisujemy tylko jedną parę; jeżeli $x < y$, to (z, d_v), zaś gdy $x > y$, to (c_v, z). Przy zapisywaniu danej pary na listę T najstarsza para jest usuwana. Początkowo T jest całkowicie zapełniona przez pary (0,0).

Ruch $u = (l, a, x, b, y)$ nie może być wykonany z permutacji π (ma status tabu), jeżeli w $\beta = (\pi_v)_{lb}$ pojawia się para niekoniecznie sąsiednich operacji $(\beta(w'), \beta(w'')) \in T$, której nie było w π_{lb}, gdzie $w' < w''$. Restrykcję tę można wyjaśnić tym, że pojawienie się takiej pary sugeruje, iż permutacja β lub zbliżona do niej była już wcześniej rozpatrywana.

6. Strategia przeglądania otoczenia i warunki stopu

Niech π będzie permutacją bazową na początku pewnej iteracji algorytmu TS. Strategia przeglądania otoczenia $\mathcal{O}(\pi) = \{\pi_v : v \in V(\pi)\}$ wykorzystuje pojęcie reprezentanta zbioru ruchów, wprowadzone pierwszy raz w [8]. Dla każdego zbioru ruchów $\hat{V}_{hx}(\pi)$ określamy ruch $v(\hat{V}_{hx}(\pi)) \in \hat{V}_{hx}(\pi)$, zwany jego reprezentantem taki, że $C_{\max}(\pi_{v(\hat{V}_{hx}(\pi))}) = \min_{v \in \hat{V}_{hx}(\pi)} C_{\max}(\pi_v)$. Dalej zamiast zbioru ruchów $V(\pi)$ rozpatrujemy tylko zbiór reprezentantów $W(\pi) = \{v(\hat{V}_{hx}(\pi)) : 1 \le h \le t, |B_h| > 1, e_h \le x \le f_h\}$. Następnie w tym zbiorze znajdujemy ruchy $R(\pi) = \{v \in W(\pi) : (v \text{ nie ma statusu tabu}) \vee C_{\max}(\pi_v) < C^{TS}\}$, które nie mają statusu tabu lub dla których wartość funkcji celu jest mniejsza niż najlepsza dotychczas znaleziona przez algorytm TS (oznaczana tutaj przez C^{TS}). Z kolei w zbiorze $R(\pi)$ znajdujemy ruch v' taki, że $C_{\max}(\pi_{v'}) = \min_{v \in R(\pi)} C_{\max}(\pi_v)$. Jeżeli zbiór $R(\pi)$ jest pusty, to usuwamy tyle najstarszych elementów z listy T, aż będzie on nie pusty. Po wyborze ruchu v' odpowiednio modyfikujemy listę T (patrz rozdział 5), podstawiamy $C^{TS} := C_{\max}(\pi_{v'})$, $\pi^{TS} := \pi_{v'}$, jeżeli $C_{\max}(\pi_{v'}) < C^{TS}$ oraz permutację $\pi_{v'}$ traktujemy jako permutację bazową (tzn. $\pi := \pi_{v'}$) dla następnej iteracji algorytmu.

W algorytmie wykorzystuje się dwa warunki stopu. Pierwszy polega na zatrzymaniu poszukiwań, kiedy wykonaliśmy już $MaxIter$ iteracji bez poprawy wartości funkcji celu. Drugi kończy poszukiwania, gdy wykonaliśmy łącznie $MaxTotIter$ iteracji. Wielkości $MaxIter$ oraz $MaxTotIter$ są parametrami algorytmu i podlegają wyborowi.

7. Wyniki badań testowych

Algorytm TS został zakodowany w Pascalu i był uruchamiany na minikomputerze IBM PC 586 (100MHz). Przedstawimy tutaj tylko wstępne wyniki testów. Testy te zostały przeprowadzone na szczególnie trudnych przykładach zaprezentowanych w [11] dla problemu gniazdowego; 10 przykładów dla każdego zestawu 15/15, 20/15, 20/20, 30/15, 30/20, 50/15, 50/20, 100/20 (pierwszy element kodujący zestaw oznacza liczbę zadań n, a drugi liczbę stanowisk m) - łącznie 80 przykładów. Ponieważ w przykładach tych w stanowisku występuje tylko jedna maszyna, to dla każdego z nich utworzono 5 przykładów o różnej liczbie maszyn m_l w poszczególnych stanowiskach. Wielkości m_l były generowane wg rozkładu równomiernego na zbiorze $\{1, 2, 3\}$. Łącznie przebadano 400 przykładów.

Dla każdego przykładu wyznaczano: (*i*) najmniejszą wartość funkcji celu C^B otrzymaną przez 10 algorytmów konstrukcyjnych o różnych regułach priorytetowych, [3], (*ii*) najmniejszą wartość funkcji celu C^{POP} otrzymaną przez dziesięciokrotne uruchomienie klasycznego algorytmu typu popraw POP startującego z różnych permutacji otrzymanych przez algorytmy konstrukcyjne z (*i*); POP zatrzymuje się, gdy w otoczeniu $O(\pi)$ nie ma rozwiązania o lepszej wartości funkcji celu, (*iii*) wartość funkcji celu C^{TSk} otrzymaną po wykonaniu $k = MaxTotIter$, $MaxTotIter = 1000, 5000$, iteracji badanego algorytmu TS. Jako rozwiązanie początkowe w TS przyjęto kolejność otrzymaną przez algorytm konstrukcyjny z regułą SPT oraz listę tabu o długości $MaxT = 8$.

Na bazie tych wartości wyznaczono względną procentową poprawę

$$\Phi^A = 100\%(C^B - C^A)/C^B$$

wartości C^A w stosunku do wartości porównawczej C^B, $A \in \{POP, TS1000, TS5000\}$. Następnie, dla każdej kombinacji n oraz m wyznaczono średnie arytmetyczne $\bar{\Phi}^A$ z 50 wartości Φ^A. Wyniki przedstawiono w tablicy 1. Z analizy tablicy 1 wynika, że średnie

Tablica 1

Wyniki badań testowych algorytmu TS

n/m	m_l	Średnie względne poprawy dla algorytmu			Czas na PC586 w [sec] dla algorytmu		
		POP	$TS1000$	$TS5000$	POP	$TS1000$	$TS5000$
15/15	1...3	3.6	18.9	19.4	1	6	28
20/15	1...3	3.7	21.4	22.0	3	12	57
20/20	1...3	3.6	14.9	15.2	4	17	82
30/15	1...3	3.0	21.2	21.8	6	30	142
30/20	1...3	2.9	15.8	16.4	9	44	209
50/15	1...3	3.0	14.2	14.6	18	91	443
50/20	1...3	2.9	17.4	17.9	26	129	635
100/20	1...3	1.3	11.1	11.2	112	550	2710
wszystkie		3.0	16.9	17.3			

względne poprawy $\bar{\Phi}^{TS1000}$ wahają się w granicach od 11.1 do 21.4 procent. Czas pracy algorytmu $TS1000$ rośnie ze wzrostem liczby operacji; od 6 sekund dla 225 operacji do około 9 minut dla 2000 operacji. Różnica pomiędzy $\bar{\Phi}^{TS1000}$ a $\bar{\Phi}^{TS5000}$ jest tylko rzędu 0.5 %, zaś czas pracy algorytmu $TS5000$ jest pięciokrotnie większy niż algorytmu $TS1000$. Dlatego też wydaje się, że w sytuacjach praktycznych wystarczy wykonać około tysiąca iteracji algorytmu TS. Porównanie algorytmu $TS1000$ z algorytmem POP z punktu widzenia jakości produkowanych rozwiązań wyraźnie wskazuje na $TS1000$; średnia względna poprawa dla wszystkich przykładów przy algorytmie $TS1000$ równa się 16.9 %, podczas gdy przy POP tylko 3.0 %. Zwiększenie czasu obliczeń jest w pełni rekompensowane znacznie lepszym rozwiązaniem.

Z przedstawionych wstępnych wyników testów wynika już, że TS może być z powodzeniem stosowany do rozwiązywania problemów o praktycznych rozmiarach. Warto tu także zauważyć, iż w algorytmie tym łatwo uwzględnić niezerowe czasy transportu pomiędzy gniazdami (wystarczy wprowadzić obciążenia na łukach E^*) czy też występowanie nieidentycznych maszyn w gniazdach produkcyjnych.

LITERATURA

1. Glover F.: Tabu search. Part I. ORSA Journal on Computing, 1, 1989, 190-206.
2. Glover F.: Tabu search fundamentals and uses, Research Report, University of Colorado, Boulder, April 1995.
3. French S.: Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop, Wiley, New York 1982.
4. Korcyl A., Sawik T.: Algorytmy typu tabu dla zadania równoważenia obciążeń elastycznego systemu montażowego. Zeszyty Naukowe Politechniki Śląskiej, z. 114, Gliwice 1994, s. 174-185.
5. Kadluczka P., Wala K.: Tabu search heuristic for generalized graph partitioning problem. Zeszyty Naukowe AGH seria: Automatyka z. 64, Kraków 1993, s. 473-487.
6. Nowicki E.: Problem przepływowy z dwoma stanowiskami obsługi: algorytm tabu search. XI Międzynarodowe Sympozjum „Zastosowania teorii systemów. Zakopane'95". Elekrotechnika t. 14, z. 4, Zakopane 1995, s. 345-350.
7. Nowicki E., Smutnicki C.: A fast taboo search algorithm for the job shop. Raport ICT Pol. Wrocł. serii: PREPRINTY nr 8/1993. (Management Science July 1996, w druku).

8. Nowicki E., Smutnicki C.: A fast taboo search algorithm for the flow shop. Raport ICT Pol. Wrocl. serii: PREPRINTY nr 8/1994. (European J. Oper. Res. 1996, w druku).

9. Nowicki E., Smutnicki C.: Flow shop with parallel machines. A tabu search approach. Raport ICT Pol. Wrocl. serii: PREPRINTY nr 30/1995. (wstępnie zaakceptowana do specjalnego wydania European J. Oper. Res. poświęconego technice tabu search).

10. Sawik T.: Optymalizacja dyskretna w elastycznych systemach produkcyjnych, WNT, Warszawa 1992.

11. Taillard E.: Benchmarks for basic scheduling problems. European J. Oper. Res. 64, 1993, s. 278-285.

12. Vaessens R.J.M., Aarts E.H.I., Lenstra J.K.: Job shop scheduling by local search. Memorandum COSOR 94-05, Eidhoven, The Netherlands, 1994.

Recenzent: Dr hab. inż. Mirosław Zaborowski, prof. Pol.Śl.

Wpłynęło do Redakcji do 30.06.1996 r.

Abstract

A fast and easily implemented approximation algorithm for a problem of finding a minimum makespan in a flexible job shop is presented. The algorithm is based on a tabu search technique with a specific neighborhood definition. A neighborhood is defined using so-called blocks of operations on a critical path. The essence of the algorithm consists in using some properties which allow us to search over the most profitable part of a neighborhood. A special advanced method of implementation improves the local search significantly and increases the speed of the algotithm. Computational experiments (up to 2000 operations and 20 centers) show its excellent numerical properties. The algorithm is eassily tuned; primal values of tuning parameters are recommended in the paper.