ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2002
Seria: AUTOMATYKA z. 134 Nr kol. 1554

Eugeniusz NOWICKI
Politechnika Wrocławska

ALGORYTM TABU DLA PROBLEMU GNIAZDOWEGO Z CZASAMI TRANSPORTU

Streszczenie. W pracy przedstawia się szybki aproksymacyjny algorytm, minimalizujący moment wykonania wszystkich zadań, dla problemu gniazdowego z transportem. Algorytm bazuje na technice tabu ze specyficzną definicją sąsiedztwa. Przeprowadzone eksperymenty obliczeniowe pokazują, że algorytm znajduje mniejsze wartości kryterium niż najlepsze aktualnie algorytmy aproksymacyjne.

TABU SEARCH ALGORITHM TO JOB SHOP PROBLEM WITH TRANSPORTATION TIMES

Summary. A fast approximation algorithm for a problem of finding the minimum makespan in a job shop with transportation times is presented. The algorithm is based on a tabu search technique with a specific neighborhood definition. Computational experiments show that the algorithm finds shorter makespans that the best approximations approaches.

1. Wprowadzenie

Proces produkcyjny w systemach gniazdowych z uwzględnieniem systemu transportowego można naszkicować następująco: Pierwsza operacja każdego zadania (wyrobu) polega na załadowaniu w gnieździe załadowczym na paletę jego części bazowej. Następnie paleta jest transportowana pomiędzy gniazdami produkcyjnymi, w których wykonywane są odpowiednie operacje. Ostatnia operacja odbywa się w gnieździe wyładowczym i polega na demontażu gotowego wyrobu z palety. Paleta przetransportowana do danego gniazda produkcyjnego oczekuje w buforze wejściowym na rozpoczęcie wykonywania operacji na części bazowej zadania, które jest na niej umieszczone. Po wykonaniu operacji paleta oczekuje na transport w buforze wyjściowym. Z reguły zakłada się, że pojemność bufora

wejściowego i wyjściowego każdego gniazda oraz liczba palet jest nieograniczona. Jako podstawowe kryterium przyjmuje się moment wykonania wszystkich zadań.

Ogólnie można wyróżnić dwa tryby pracy środków transportu (wózków): (*i*) tryb zabierz i zostaw (ang. pick and drop) lub (*ii*) tryb czekaj i jedź (ang. stop and go), [10]. Pierwszy z nich polega na tym, że po dostarczeniu palety z częścią bazową zadania do bufora wejściowego odpowiedniego gniazda wózek jest gotowy do transportu części bazowej innego zadania; w przypadku gdy zadanie to jest wykonywane na innej maszynie, musi najpierw do niej "dojechać" (tzw. pusty przebieg). W trybie drugim wózek jest związany z częścią bazową zadania (dokładniej z paletą, na której jest ona umieszczona) od początku do końca jego procesu technologicznego. Tryb pierwszy, będąc ogólniejszy niż drugi, jest znacznie trudniejszy do modelowania (z uwagi na potrzebę uwzględnienia pustych przebiegów wózków).

W celu rozwiązania problemu gniazdowego z transportem poleca się w literaturze stosowanie przede wszystkim algorytmów przybliżonych typu konstrukcyjnego, patrz np. [1], [5], [10]. Algorytmy te na podstawie pewnych reguł priorytetowych szeregują kolejno pojedyncze operacje na maszynach i wózkach transportowych. Czas ich pracy jest bardzo krótki, ale jakość produkowanych przez nie harmonogramów nie jest zadowalająca. Ostatnio jednak pojawiły się w literaturze algorytmy przybliżone typu popraw, bazujące na technikach tabu, [2]; patrz prace [7], [8] – dla trybu czekaj i jedź oraz prace [3], [4] – dla trybu zabierz i zostaw; w tych dwóch ostatnich rozważa się tylko jeden wózek transportowy (robot). Algorytmy te produkują znacznie lepsze harmonogramy (w sensie wartości funkcji celu) przy akceptowalnych przez praktyków czasach obliczeń; odnosi się to szczególnie do algorytmów z prac [7], [8].

W niniejszej pracy formułuje się wygodny, permutacyjno-grafowy model problemu gniazdowego z transportem pracującym w trybie zabierz i zostaw. Następnie wykorzystując tzw. eliminacyjne własności ścieżki krytycznej, proponuje się bardzo niewielkie sąsiedztwo danego rozwiązania, które można zastosować przy konstrukcji różnych metod popraw. W pracy stosuje się technikę tabu, a dokładniej twórczo adaptuje się algorytm TSAB wykorzystywany wcześniej do szybkiego znajdowania efektywnych harmonogramów w problemie gniazdowym bez uwzględnienia transportu [9]. Wyniki badań porównawczych pomiędzy proponowanym algorytmem a algorytmami z prac [3], [4] pokazują, że dostarcza on istotnie lepszych rozwiązań w znacznie krótszym czasie. Dodatkowe badania na instancjach o praktycznych rozmiarach wykazały, że algorytm pro-

dukuje średnio o 15% lepsze rozwiązania niż jednoczesne zastosowanie kilku powszechnie używanych algorytmów konstrukcyjnych w maksymalnym czasie rzędu kilku minut na IBM PC.

2. Sformułowanie problemu i model permutacyjno-grafowy

Dany jest zbiór maszyn produkcyjnych $\mathcal{M}^P = \{1, \ldots, m'\}$ oraz zbiór $\mathcal{W}$ zawierający $lw \geq 1$ identycznych środków transportowych (wózków). Wózki są ponumerowane kolejnymi liczbami $m'+1, m'+2, \ldots, m'+lw = m$; $\mathcal{W} = \{m'+1, \ldots, m\}$. Czas przejazdu pomiędzy każdą parą różnych maszyn produkcyjnych $l', l'' \in \mathcal{M}^P$ wózka pustego oraz wózka załadowanego wymaga odpowiednio $d(l', l'') \geq 0$ oraz $D(l', l'') \geq d(l', l'')$ jednostek czasu; $D(l, l) = d(l, l) = 0$, $l \in \mathcal{M}^P$. Przyjmujemy, że wielkości $d(l', l'')$, $D(l', l'')$ spełniają warunek trójkąta oraz że czas przejazdu wózka załadowanego nie zależy od transportowanego ładunku (palety z częścią bazową odpowiedniego zadania).

Proces produkcyjny polega na wykonaniu zbioru zadań $\mathcal{J} = \{1, 2, \ldots, r\}$. Każde zadanie $k \in \mathcal{J}$ składa się z sekwencji n_k^P operacji produkcyjnych indeksowanych kolejno przez $\tau_k + 1, \tau_k + 2, \ldots, \tau_k + n_k^P$, które powinny być wykonywane w tej kolejności, gdzie $\tau_k = \sum_{i=1}^{k-1} n_i^P$ oraz $\tau_1 = 0$. Łącznie trzeba wykonać $n^P = \sum_{k \in J} n_k^P$ operacji produkcyjnych $\mathcal{O}^P = \{1, 2, \ldots, n^P\}$. Dla każdej operacji produkcyjnej $j \in \mathcal{O}$ jest określona dokładnie jedna maszyna produkcyjna $\mu_j \in \mathcal{M}^P$, na której może być ona wykonana w czasie $p_j > 0$. Po wykonaniu operacji produkcyjnej $j' = \tau_k + i$, $1 \leq i < n_k^P$ zadania $k \in \mathcal{J}$ na maszynie $\mu_{j'}$, część bazowa tego zadania jest przemieszczana do bufora wyjściowego tej maszyny i następnie transportowana przez jeden z wózków do bufora wejściowego maszyny $\mu_{j''}$, gdzie $j'' = \tau_k + i + 1$. (Przyjmujemy, że czas przemieszczenia części bazowej zadania z maszyny do jej bufora wyjściowego oraz z bufora wejściowego na maszynę jest równy zero.) Wynika stąd, że pomiędzy każdą parą operacji produkcyjnych $\tau_k + i$, $\tau_k + i + 1$ zadania k jest wykonywana operacja transportowa, którą oznaczamy przez $[\tau_k + i]$, $1 \leq i < n_k^P$. Łącznie występuje $\sum_{k \in \mathcal{J}} (n_k^P - 1) = n^P - r$ operacji transportowych

$$\mathcal{O}^{Tr} = \bigcup_{k \in \mathcal{J}} \bigcup_{i=1}^{n_k^P - 1} \{[\tau_k + i]\}. \tag{1}$$

Operacja transportowa $j \in \mathcal{O}^{Tr}$ może być wykonana przez jeden z wózków $\mu_j \in \mathcal{W}$. W praktyce przemysłowej możliwe są dwie sytuacje: (i) wózek μ_j jest wcześniej określony lub (ii) podlega wyborowi (dodatkowa zmienna decyzyjna). Dalej będziemy rozpatrywać

tylko tę pierwszą sytuację. Tak więc, realizacja operacji transportowej $j = [\tau_k + i] \in \mathcal{O}^{Tr}$ polega na przewiezieniu przez zadany wózek μ_j części bazowej zadania k z maszyny produkcyjnej $l' = \mu_{\tau_k+i}$ do maszyny produkcyjnej $l'' = \mu_{\tau_k+i+1}$ w czasie $p_j = D(l', l'')$. Zauważmy, że pomiędzy wykonaniem dwóch kolejnych operacji transportowych $j' = [\tau_{k'} + i']$, $j'' = [\tau_{k''} + i'']$ na tym samym wózku $w = \mu_{j'} = \mu_{j''}$ wózek musi przejechac bez ładunku z maszyny $l' = \mu_{\tau_{k'}+i'+1}$ do maszyny $l'' = \mu_{\tau_{k''}+i''}$ w czasie $d(l', l'')$. Czas ten interpretujemy jako wymagany czas *przezbrojenia* $s(j', j'')$ wózka pomiędzy dwiema kolejnymi operacjami transportowymi przez niego wykonywanymi.

Podobnie jak w klasycznym problemie gniazdowym, nie dopuszczamy podzielności operacji produkcyjnych i transportowych i zakładamy, że w każdej chwili czasowej:
(i) każda maszyna produkcyjna może wykonywać co najwyżej jedną operację produkcyjną,
(ii) każdy wózek może wykonywać co najwyżej jedną operację transportową oraz
(iii) nie można jednocześnie wykonywać więcej niż jedną operację produkcyjną lub transportową danego zadania. Oznacza to, że nie musimy rozróżniać maszyn produkcyjnych od wózków (maszyn transportowych) oraz operacji produkcyjnych od operacji transportowych, z jednym wyjątkiem, że pomiędzy kolejnymi operacjami transportowymi wykonywanymi na tej samej maszynie transportowej (wózku) występują przezbrojenia. Niech $\mathcal{O} = \mathcal{O}^P \cup \mathcal{O}^{Tr}$ oznacza zbiór wszystkich operacji produkcyjnych i transportowych, $\mathcal{M} = \mathcal{M}^P \cup \mathcal{W}$ – zbiór wszystkich maszyn produkcyjnych i transportowych, zaś $S(j)$ – moment rozpoczęcia wykonywania operacji $j \in \mathcal{O}$ na maszynie μ_j. Uszeregowanie dopuszczalne definiujemy jako zestaw $S(j) \geq 0$, $j \in \mathcal{O}$ taki, że sformułowane powyżej ograniczenia są spełnione. Problem polega na znalezieniu dopuszczalnego uszeregowania minimalizującego moment wykonania wszystkich operacji $\max_{j\in\mathcal{O}} (S(j) + p_j)$.

Przejdziemy teraz do sformułowania modelu permutacyjno-grafowego. W tym celu wprowadzimy dodatkowe niezbędne oznaczenia. Niech $\mathcal{O}_l = \{j \in \mathcal{O} : \mu_j = l\}$ określa zbiór operacji wykonywanych na maszynie $l \in \mathcal{M}$. Niech kolejność wykonywania tych operacji wyznacza permutacja $\pi_l = (\pi_l(1), \ldots, \pi_l(o_l))$, gdzie $o_l = |\mathcal{O}_l|$. Wtedy zestaw permutacji $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$ – zwany dalej też permutacją – jest zmienną decyzyjną określającą kolejność wykonywania wszystkich operacji ze zbioru $\mathcal{O}$ na poszczególnych maszynach; zbiór wszystkich takich permutacji oznaczamy przez Π.

Wykorzystując wprowadzone oznaczenia, dla permutacji $\pi \in \Pi$ definiujemy skierowany graf $G(\pi) = (\mathcal{O}, E^T \cup E^K(\pi))$ ze zbiorem wierzchołków $\mathcal{O}$ reprezentującym

poszczególne operacje oraz zbiorami łuków

$$E^T = \bigcup_{k \in \mathcal{J}} \bigcup_{j=\tau_k+2}^{\tau_k+n_k^P} \{(j-1,[j-1]),\ ([j-1],j)\}, \tag{2}$$

$$E^K(\pi) = \bigcup_{l \in \mathcal{M}} \bigcup_{i=2}^{o_l} \{(\pi_l(i-1),\pi_l(i))\} \tag{3}$$

reprezentującymi ograniczenia wynikające odpowiednio: z marszruty technologicznej poszczególnych operacji w zadaniach oraz kolejności π wykonywania operacji na maszynach. Każdy wierzchołek j ma wagę p_j. Każdy łuk $(\pi_l(i-1),\pi_l(i))$, $m' < l \leq m$, ze zbioru $E^K(\pi)$ ma wagę $s(\pi_l(i-1),\pi_l(i))$, zaś wszystkie pozostałe łuki z tego zbioru oraz łuki ze zbioru E^T mają wagę zerową. Niech $C_{\max}(\pi)$ oznacza długość najdłuższej ścieżki (ścieżka krytyczna) w grafie $G(\pi)$ dla permutacji $\pi \in \Pi$; w przypadku gdy graf $G(\pi)$ jest cykliczny, przyjmujemy, że $C_{\max}(\pi) = \infty$. Ostatecznie, rozważany problem możemy sformułować jako znalezienie takiej permutacji $\pi^* \in \Pi$, że odpowiadający jej graf $G(\pi^*)$ ma najmniejszą długość ścieżki krytycznej $C_{\max}(\pi^*)$.

Istotną rolę w dalszych rozważaniach pełni ścieżka krytyczna w grafie $G(\pi)$ zapisywana jako ciąg $u = (u_1, u_2, \ldots, u_{lu})$ wierzchołków (operacji), których liczbę oznaczamy przez lu. Ścieżkę u można przedstawić w postaci konkatenacji pewnej liczby (oznaczmy ją przez lb) zwartych podciągów operacji wykonywanych na tej samej maszynie, zwanych *blokami*. Dokładniej, $u = (B_1, B_2, \ldots, B_{lb})$, gdzie $B_h = (\pi_{l_h}(e_h), \pi_{l_h}(e_h+1), \ldots, \pi_{l_h}(f_h))$, $1 \leq h \leq lb$, jest ciągiem (h-tym blokiem na ścieżce u) zawierającym operacje wykonywane na maszynie oznaczanej przez l_h, poczynając od operacji z pozycji e_h, $1 \leq e_h \leq o_{l_h}$, a kończąc na operacji z pozycji f_h, $e_h \leq f_h \leq o_{l_h}$ w permutacji π_{l_h}; przyjmujemy, że operacje z dwóch kolejnych bloków B_{h-1}, B_h, są wykonywane na różnych maszynach, tzn. $l_{h-1} \neq l_h$, $h = 2, \ldots, lb$. Zauważmy, że wierzchołek $\pi_{l_{h-1}}(f_{h-1})$ odpowiadający ostatniej operacji z bloku B_{h-1} jest połączony z wierzchołkiem $\pi_{l_h}(e_h)$ odpowiadającym pierwszej operacji z bloku B_h łukiem z E^T. Wierzchołki odpowiadające sąsiednim operacjom bloku są połączone łukami z $E^K(\pi)$.

Rozbijmy indeksy bloków B_h na dwie rozłączne grupy: $I^1 = \{1 \leq h \leq lb:\ (|B_h| = 1) \vee (1 \leq l_h \leq m')\}$ – indeksy bloków jednoelementowych lub bloków występujących na maszynach produkcyjnych oraz $I^2 = \{1 \leq h \leq lb:\ (|B_h| > 1) \wedge (m' < l_h \leq m)\}$ – indeksy bloków wieloelementowych występujących na maszynach transportowych. Wtedy długość

ścieżki krytycznej $C_{\max}(\pi)$ można przedstawić w postaci:

$$C_{\max}(\pi) = \sum_{h \in I^1} \sum_{j \in B_h} p_j + \sum_{h \in I^2} \Big(\sum_{j \in B_h} p_j + \sum_{i=e_h+1}^{f_h} s(\pi_{l_h}(i-1), \pi_{l_h}(i)) \Big). \tag{4}$$

Pierwszy składnik tej sumy oraz pierwszy fragment drugiego składnika określa czas wykonywania wszystkich operacji z bloków, zaś drugi fragment drugiego składnika – czasy przezbrojeń pomiędzy kolejnymi operacjami transportowymi z poszczególnych bloków występujących na maszynach transportowych.

3. Sąsiedztwa i ich własności

Podstawowym elementem algorytmów aproksymacyjnych typu popraw jest pojęcie sąsiedztwa $\mathcal{N}(\pi)$ danego rozwiązania (permutacji π). Ogólnie mówiąc, jest to zbiór permutacji $\alpha \in \Pi \setminus \{\pi\}$, które znajdują się w "pobliżu" permutacji π. Do określenia tej bliskości wykorzystuje się pojęcie ruchu, tzn. funkcji v, która odwzorowuje zbiór Π w Π. Zdefiniowanie dla danej permutacji $\pi \in \Pi$ ruchu v, a dokładniej — określenie zbioru wykonywanych z tej permutacji ruchów $V(\pi)$, pozwala określić jej sąsiedztwo $\mathcal{N}(\pi) = \{\pi_{(v)} : v \in V(\pi)\}$; symbol $\pi_{(v)}$ oznacza tutaj wartość funkcji v dla permutacji π.

Przy wyborze zbioru ruchów występują dwie sprzeczne tendencje. Z jednej strony zbiór $V(\pi)$ powinien być jak największy, ponieważ zwiększa to szanse na dokładniejszą penetrację przestrzeni rozwiązań (zbioru permutacji Π). Z drugiej strony powinien być jak najmniejszy, aby czas przeglądnięcia sąsiedztwa $\mathcal{N}(\pi)$ – pod kątem znalezienia najlepszego rozwiązania – był jak najkrótszy. Istotne jest także, aby procedura sprawdzania, czy graf $G(\pi_{(v)})$ jest acykliczny, była jak najmniej pracochłonna; najlepiej, gdy dla każdego ruchu v graf ten jest zawsze acykliczny. Dlatego też szczególnie interesujące wydają się być ruchy $v = (a, b)$ polegające na wymianie położenia dwóch kolejnych operacji a, b wykonywanych na tej samej maszynie; zbiór wszystkich takich ruchów oznaczamy przez $V^0(\pi) = \{v = (a, b) : a = \pi_l(i-1), b = \pi_l(i), 2 \le i \le o_l, l \in \mathcal{M}\}$. Zbiór $V^0(\pi)$ zawiera $n - m$ ruchów, co może być jednak znacznie za dużo, gdyż do wyznaczenia jednej wartości $C_{\max}(\pi_{(v)})$ potrzeba aż $O(n)$ czasu, gdzie $n = 2n^P - r$. Większość z nich można jednak wyeliminować bez wyliczania $C_{\max}(\pi_{(v)})$, co wynika z następującej własności, stanowiącej uogólnienie odpowiedniego twierdzenia z pracy [9] dla problemu gniazdowego bez transportu.

Własność 1. *Niech permutacja $\pi \in \Pi$ generuje acykliczny graf $G(\pi)$ ze ścieżką krytyczną u i blokami operacji B_h, $h = 1, \ldots, lb$. Niech ruch $v = (a, b) \in V^o(\pi)$. Jeżeli spełniony*

jest jeden z poniższych dwóch warunków:

(i) co najmniej jedna z operacji a, b nie należy do żadnego z bloków,

(ii) operacje a, b należą do pewnego bloku B_h, $h \in I^1$ oraz żadna z nich nie jest pierwszą lub ostatnią operacją tego bloku,

to

$$C_{\max}(\pi_{(v)}) \geq C_{\max}(\pi). \tag{5}$$

Ten dowód własności wynika z dwóch następujących spostrzeżeń: (I) po wykonaniu ruchu v spełniającego jeden z warunków (i)-(ii) w grafie $G(\pi_{(v)})$ istnieje ścieżka zawierająca wszystkie wierzchołki ścieżki u oraz (II) przezbrojenia $s(j', j'')$ spełniają tzw. słaby warunek trójkąta, tzn.

$$s(j', j'') \leq s(j', k) + p_k + s(k, j'') \tag{6}$$

dla każdych trzech operacji transportowych j', k, j''. Wykorzystując te spostrzeżenia oraz równanie (4) bezpośrednio otrzymujemy nierówność (5).

Niech $V^1(\pi)$ oznacza zbiór wszystkich ruchów $v \in V(\pi)$ nie spełniających warunku (i) oraz warunku (ii) powyższej własności. Zbiór ten zawiera ruchy polegające na zamianie położenia pierwszych dwóch i ostatnich dwóch operacji każdego bloku na maszynach produkcyjnych oraz zamianie położenia kolejnych par operacji każdego bloku na maszynach transportowych. W szczególności na maszynie produkcyjnej dla bloku dwuelementowego jest wykonywany tylko jeden ruch, a dla bloku wieloelementowego – dwa ruchy. Z kolei na maszynie transportowej dla każdego bloku B_h wykonuje się $|B_h| - 1$ ruchów. Tak więc

$$|V^1(\pi)| = \sum_{h \in I^1} \min\{2, |B_h| - 1\} + \sum_{h \in I^2} (|B_h| - 1). \tag{7}$$

Wynika stąd, że zbiór ruchów $V^1(\pi)$ jest znacznie mniejszy niż $V^0(\pi)$. Badania numeryczne pokazały, że dla przykładów zawierających $n^P = 500$ operacji produkcyjnych, $m' = 20$ maszyn produkcyjnych oraz $lw = 10$ maszyn transportowych zawiera on średnio około 40 elementów. Na uwagę zasługuje także fakt, że dla każdego ruchu $v \in V^1(\pi)$ graf $G(\pi_{(v)})$ jest acykliczny. Własność ta – trywialna dla problemów gniazdowych bez transportu – wynika tutaj ze słabego warunku trójkąta spełnionego przez czasy przezbrojeń (puste przebiegi wózków).

Reasumując, zbiór ruchów $V^1(\pi)$ jest stosunkowo niewielki, co pozwala na szybkie przeglądnięcie sąsiedztwa i w konsekwencji oznacza bardzo krótki czas wykonania jednej iteracji algorytmu popraw, w którym będzie on zastosowany. Jednocześnie przeprowad-

zone badania numeryczne pokazały, że jest on wystarczająco duży, aby skutecznie prowadzić proces poszukiwań w przestrzeni rozwiązań.

W tej pracy proponowany zbiór ruchów wykorzystujemy przy konstrukcji sąsiedztwa w algorytmie popraw typu tabu – zwanym dalej algorytmem TS. Wszystkie pozostałe elementy algorytmu TS, takie jak: postać listy tabu T, definicja ruchów zabronionych, strategia przeglądania sąsiedztwa $\mathcal{N}^1(\pi) = \{\pi_{(v)} : v \in V^1(\pi)\}$ oraz metody intensyfikacji i dywersyfikacji poszukiwań są zbliżone do tych, które zastosowano w algorytmie TSAB z pracy [7] i dlatego nie będą przedstawiane.

4. Wyniki badań testowych

Algorytm TS zakodowano w Delphi 5 i uruchamiano na Pentium II (330MHz). Głównym celem badań testowych była odpowiedź na dwa pytania. Jak zachowuje się algorytm TS w porównaniu do algorytmów z pracy [4] na instancjach tam badanych (w [3] testuje się tylko instancje bez operacji produkcyjnych) oraz o ile lepsze rozwiązania generuje TS od rozwiązań otrzymanych przez algorytmy konstrukcyjne na instancjach losowych?

Instancje testowane w [4] zostały wygenerowane na bazie dwóch przykładów Mutha i Thompsona, problemów gniazdowych bez transportu; pierwszy zawiera 6 zadań, 36 operacji oraz 6 maszyn, zaś drugi 10 zadań, 100 operacji oraz 10 maszym (tzw. "setka"). Dalej będziemy się zajmować tylko tym drugim, ponieważ jest on znacznie bardziej skomplikowany. W pracy [4] rozważano dwie klasy problemów: w pierwszej czasy wykonywania operacji produkcyjnych p_j nie zostały zmienione w stosunku do czasów setki, zaś w drugiej czasy p_j zastąpiono przez $\lceil f \cdot p_j \rceil$, gdzie $0 < f < 1$. Dodatkowe parametry wynikające z uwzględnienia transportu generowano następująco: Przyjęto, że w systemie jest tylko jeden wózek $lw = 1$ (robot), a maszyny produkcyjne są umieszczone na jednej linii w kolejności $1, \ldots, m'$. Dla czasów transportu wózka załadowanego rozpatrzono dwie sytuacje: (*i*) $D(l', l'') = D|l' - l''|$ z różnymi wartościami D (czas transportu proporcjonalny do odległości pomiędzy odpowiednimi maszynami produkcyjnymi) oraz (*ii*) $D(l', l'')| = T$ (stały czas transportu T). Podobne sytuacje rozważono dla czasów transportu wózka pustego: (*i*) $d(l', l'') = d|l' - l''|$ oraz (*ii*) $d(l', l'') = t$. Po wykonaniu serii eksperymentów z różnymi wartościami f, D, d, T, t otrzymano ostatecznie 14 najtrudniejszych instancji.

W pracy [4] testowano trzy różne wersje algorytmu tabu: pierwszą i drugą uruchamiano 6 razy, a trzecią 12 razy, przy różnych parametrach sterujących. Niech C^{HK} oznacza najmniejszą wartość funkcji celu wygenerowaną dla danej instancji wśród $6+6+12=24$ przebiegów. Wszystkie przebiegi rozpoczynały się z uszeregowania o wartości C^{IN} otrzymanego przez odpowiedni algorytm konstrukcyjny, zwany dalej algorytmem IN.

Nasze badania testowe polegały na wyznaczeniu, dla każdej z 14 wyżej opisanych instancji, wartości funkcji celu C^{TS} wyprodukowanej przez algorytm TS oraz dwóch wartości względnej poprawy Φ(IN,HK), Φ(HK,TS), gdzie $\Phi(A,B) = 100\%(C^A - C^B)/C^A$. Wyniki wraz z wartościami C^{IN}, C^{HK} (z pracy [4]), "kodami" instancji oraz sumarycznym czasem przebiegu wózka załadowanego SumTr=$\sum_{j \in O^{Tr}} p_j$ podano w tablicy 1; przykładowo kod f1,0.D1.t1 oznacza, że $f=1$, $D=1$, $t=1$.

Tablica 1

Wyniki badań testowych algorytmu TS na przykładach z pracy [4]

Nr	Kod instancji	SumTr	C^A			Poprawa w %	
			IN	HK	TS	Φ(IN,HK)	Φ(HK,TS)
1	f1,0.D1.d1	223	1088	990	967	9,0	2,3
2	f1,0.D1.t0	223	1088	989	964	9,1	2,5
3	f1,0.D1.t1	223	1088	995	961	8,5	3,4
4	f1,0.D2.d1	446	1123	993	980	11,6	1,3
5	f1,0.D3.d1	669	1163	1070	1033	8,0	3,5
6	f1,0.D5.t2	1115	1444	1325	1321	8,2	0,3
7	f1,0.T1.t1	90	1092	1006	956	7,9	5,0
8	f1,0.T2.t1	180	1094	1015	958	7,2	5,6
9	f1,0.T5.t2	450	1102	1090	974	1,1	10,6
10	f0,5.D1.d1	223	600	555	505	7,5	9,0
11	f0,5.D1.t1	223	595	544	492	8,6	9,6
12	f0,5.D2.d1	446	689	633	630	8,1	0,5
13	f0,5.D2.t0	446	611	578	553	5,4	4,3
14	f0,5.D2.t1	446	635	613	604	3,5	1,5
all						7,4	4,2

Z analizy tej tablicy wynika, że algorytm TS uruchomiony tylko raz dla każdej instancji zawsze znajduje lepsze uszeregowanie niż najlepsze uszeregowanie otrzymane wśród 24

przebiegów algorytmu z pracy [4]; średnia poprawa na badanych instancjach wynosi 4,2%. Warto tu także zauważyć, że łączny czas pracy na pojedynczej instancji 24 przebiegów algorytmu z [4] był rzędu kilkunastu godzin, podczas gdy algorytm TS pracuje w czasie kilku minut; algorytm z [4] testowano na komputerze Sun Ultra 2 (167 MHz). Oznacza to, że algorytm TS istotnie przewyższa porównywany algorytm z uwagi na lepszą jakość produkowanych uszeregowań i znacznie krótszy czas obliczeń.

Algorytm TS testowano także na serii 800 instancji otrzymanych z 80 instancji Taillarda dla problemu gniazdowego bez transportu (liczba operacji n^P w tych instancjach waha się od 225 do 2000, zaś liczba maszyn $m' \in \{15, 20\}$). Czasy transportu generowano z przedziału $[1, 10]$, zaś liczbę wózków z przedziału $[m'//3, m']$. TS poprawiał średnio o 15% rozwiązania dostarczone przez algorytmy konstrukcyjne w czasie kilku minut.

LITERATURA

1. Bilge Ü., Ulusoy G.: A time window-approach to simultaneous scheduling of machines and material handling system in an FMS. Operations Research 43, 1995, pp. 1058-1070.
2. Glover F., Laguna M.: Tabu Search. Kluwer Academic Publishers, Massachusetts USA, 1997.
3. Hurink J., Knust S.: A tabu search algorithm for scheduling a single robot in a job-shop environment. Osnabrücker Schriften zur Mathematik, Reihe P, No. 213, 1999.
4. Hurink J., Knust S.: Tabu search algorithms for the job-shop problems with a single transport robot. Osnabrücker Schriften zur Mathematik, Reihe P, No. 231, 2001.
5. Langston M.A.: Interstage transportation planning in the deterministic flow-shop environment, Operations Research 40, 1992, pp. 113-125.
6. Nowicki E.: The permutation flow shop with buffers. A local search approach. European Journal of Operational Research 116, 1999, pp. 205-219.
7. Nowicki E.: Metoda tabu w problemach szeregowania zadań produkcyjnych. Oficyna Wydawnicza Politechniki Wrocławskiej, Ser. Monografie 27, Wrocław 1999.
8. Nowicki E.: Zastosowanie techniki tabu do harmonogramowania elastycznych gniazd produkcyjnych z czasami transportu i przezbrojeń. Z. N. Pol. Śl., seria Automatyka 129, 2000, pp. 295-303.
9. Nowicki E., Smutnicki C.: A fast taboo search algorithm for the job shop problem. Management Science 42, 1996, pp. 797-813.
10. Sawik T.: Planowanie i sterowanie produkcji w elastycznych systemach montażowych. WNT, Warszawa 1996.

Recenzent: Dr hab. inż. Eugeniusz Toczylowski, Prof. Pol. Warsz.

Abstract

The paper deals with the criterion of the makespan minimisation for the job shop problem with transportation times. A fast and easily implemented approximation algorithm is proposed based on a non trivial generalisation of the path elimination properties known for the classic job shop problem. This algorithm is able to achieve excellent results for instances up to 2000 operations, 100 jobs and 20 machine due to exploiting of some structural properties of the problem combined with a local search technique controlled by a tabu search strategy. Computational results are presented for test data arising from benchmark instances enlarged by transportation and empty mowing times.