ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2002
Seria: AUTOMATYKA z. 136 Nr kol. 1556

Tomasz HOMEL, Konrad WALA
Akademia Górniczo-Hutnicza

METAHEURYSTYKA TABU W OPTYMALIZACJI WIELORZĘDOWEGO ROZMIESZCZENIA MASZYN

Streszczenie. W artykule rozważany jest problem wielorzędowego rozmieszczenia maszyn w hali produkcyjnej. Zaprezentowano oryginalny model, gdzie rozwiązanie jest sformalizowane w postaci permutacji numerów maszyn oraz zaproponowano algorytmy oparte na metaheurystyce tabu. Przedstawiono wyniki badań komputerowych, które ilustrują przydatność modelu i algorytmów dla praktyki projektowej.

TABU METAHEURISTIC IN OPTIMIZATION OF THE MANY-ROW MACHINE LAYOUT PROBLEM

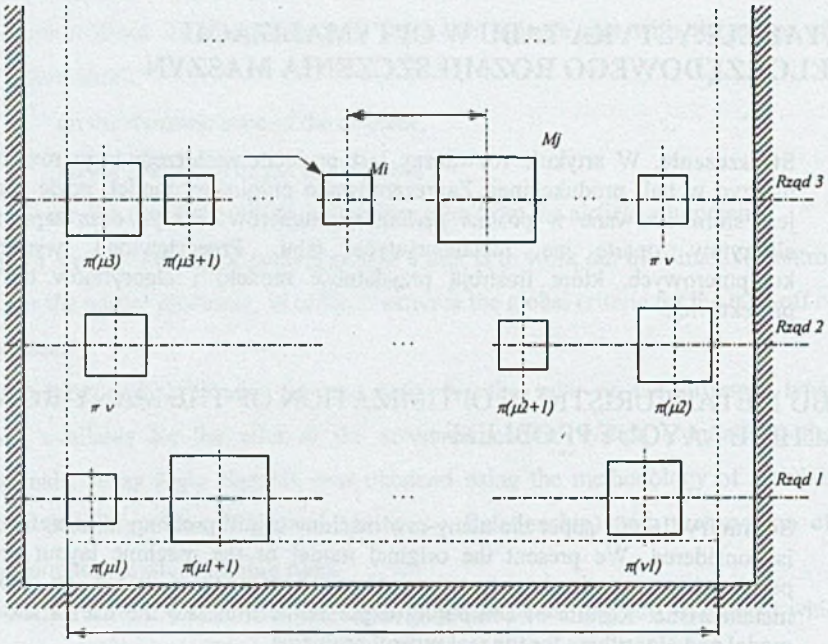
Summary. In this paper the many-row machine layout problem in manufacturing cell is considered. We present the original model of the machine layout problem of permutation optimization type and propose algorithms based on tabu search metaheuristic. Results of computing experiments illustrate the use of the proposed model and algorithms for the real project problems.

1. Wprowadzenie

Po rozwiązaniu problemu podziału zbioru maszyn przedsiębiorstwa na komórki lub hale produkcyjne powstaje nowy problem dla każdej hali oddzielnie, polegający na takim rozmieszczeniu maszyn, które minimalizuje całkowite koszty transportu półproduktów (np. detali) pomiędzy maszynami. Poniżej przedstawiono kombinatoryczny model problemu wielorzędowego rozmieszczenia maszyn (stanowisk roboczych) w hali produkcyjnej (por. [3]), natomiast w sekcji 3 opisano zaproponowane algorytmy konstrukcyjne oraz badane wersje algorytmów popraw opartych na metaheurystyce tabu, tj. algorytmów *TS* (ang. Tabu

Search). W sekcji 4 przedstawiono wyniki numeryczne badań efektywności różnych wersji algorytmów *TS*.

2. Model matematyczny problemu rozmieszczenia maszyn



Rys. 1. Wielorzędowe rozmieszczenie maszyn w hali produkcyjnej
Fig. 1. Machine rows arrangement in the manufacturing cell

Niech π będzie permutacją numerów n maszyn, które podlegać będą rozmieszczeniu. Dalej, niech c_{ij} będzie kosztem jednostkowym (na jednostkę odległości) transportu półproduktów pomiędzy maszynami *(i, j)* oraz b_{ij} będzie długością drogi transportu pomiędzy tymi maszynami, gdzie tablica odległości $[b_{ij}]_{n*n}$ jest wyliczana dla każdej permutacji maszyn π i zależy zarówno od położenia maszyn określonych przez permutację π jak i od zastosowanego systemu transportu międzymaszynowego w hali. Przy tak przyjętych założeniach możemy zapisać całkowity koszt transportu półproduktów pomiędzy maszynami hali w postaci następującej funkcji celu:

$$Q(\pi) = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j=i}}^{n} c_{\pi(i)\pi(j)} b_{\pi(i)\pi(j)} \Rightarrow \min, \qquad (1)$$

gdzie: $\pi(j) \in \{1, 2, ..., n\}$, $j = 1, 2, ..., n$; $\pi(j)$ - numer maszyny usytuowanej na *j-tej* pozycji w hali maszyn. Permutacja π wyznacza kolejność, z jaką będą rozmieszczane maszyny w hali; sposób rozmieszczenia ilustruje rys. 1.

Niech h jest maksymalną długością rzędów w hali produkcyjnej, a_i jest szerokością maszyny i oraz d_{ij} minimalnym odstępem między maszynami (i, j). Minimalna długość rzędu potrzebna do rozmieszczenia wszystkich maszyn zgodnie z permutacją π jest równa $D(\pi) = a_{\pi(1)} + d_{\pi(1)\pi(2)} + a_{\pi(2)} + d_{\pi(2)\pi(3)} + ... + d_{\pi(n-1)\pi(n)} + a_{\pi(n)}$. Jeżeli minimalna długość rzędu wszystkich maszyn D przewyższa maksymalną długość rzędu hali h, $D(\pi) > h$, tworzymy dodatkowe rzędy, w których kolejno rozmieszczamy maszyny zgodnie z permutacją π. Niech k jest numerem porządkowym rzędu maszyn w hali. Każdy rząd k hali jest utworzony z maszyn podsekwencji $(\pi(\mu k), \pi(\mu k+1), ..., \pi(\nu k))$ permutacji π. Długość rzędu k maszyn określona jest zależnością $D(\pi(\mu k), \pi(\nu k)) = a_{\pi(\mu k)} + d_{\pi(\mu k)\pi(\mu k+1)} + ... + a_{\pi(\nu k)}$, $k = 1, 2, ..., K$, gdzie K jest liczbą utworzonych rzędów maszyn. Do pierwszego rzędu maszyn należą maszyny podsekwencji $\pi(\mu 1), ..., \pi(\nu 1))$ spełniające warunki:

$D(\pi(\mu 1), \pi(\nu 1)) \leq h$ i $D(\pi(\mu 1), \pi(\nu 1+1)) > h$

Sekwencja maszyn umieszczona w drugim rzędzie spełnia zależności:

$D(\pi(\mu 2), \pi(\nu 2)) \leq h$ i $D(\pi(\mu 2), \pi(\nu 2+1)) > h$

I tak dalej, gdzie $\pi = (\pi(1), \pi(2), ..., \pi(n)) = (\pi(\mu 1), ..., \pi(\nu 1), \pi(\mu 2), ..., \pi(\nu 2), \pi(\mu 3), ..., \pi(\nu K))$ i $\mu 1 = 1$, $\mu 2 = \nu 1+1$, $\mu 3 = \nu 2+1$, ..., $\mu K = \nu(K-1) + 1$, $\nu K = n$.

3. Algorytmy optymalizacji

Do rozwiązania tak zdefiniowanego *NP-trudnego* problemu kombinatorycznego zaproponowano dwa algorytmy konstrukcyjne, których celem jest wyznaczenie rozwiązań początkowych dla badanych algorytmów popraw *TS*.

3.1. Algorytmy konstrukcyjne

Algorytm „**Constr1**" (o złożoności $O(n^2)$) na podstawie jednostkowego kosztu transportu c_{ij} rozmieszcza maszyny tak, aby zminimalizować „lokalne" koszty transportu.

Krok 1. W macierzy $[c_{ij}]_{nxn}$ znajdź maksymalny element $c_{ij} = max \{c_{kl} : (k,l \in \{1, 2, ..., n\}) \wedge (k \neq l)\}$ i przypisz $c_{ik} = c_{jk} := -\infty$ dla $k = 1, 2, ..., n$ oraz jako rozwiązanie częściowe przypisz $\pi := (i, j)$.

Krok 2. Jeżeli częściowe rozwiązanie $\pi = (\pi(\mu), \pi(\mu+1), ..., \pi(\nu))$ nie zawiera wszystkich maszyn, to dla macierzy kosztów $[c_{ij}]_{nxn}$ znajdź maksymalny element $c_{ij} = max \{ c_{k\pi(\mu)}, c_{\pi(\nu)k} : k \in \{1, 2, ..., n\}\}$. Jeżeli $j = \pi(\mu)$, to przypisz $\pi := (i, \pi(\mu), ..., \pi(\nu))$ oraz $c_{ik} := -\infty$ dla $k = 1, 2, ..., n$; w przeciwnym przypadku podstaw $\pi := (\pi(\mu), ..., \pi(\nu), j)$ oraz $c_{kj} := -\infty$ dla $k = 1, 2, ..., n$.

Algorytm „**Constr2**" (o złożoności $O(nlogn)$), wyznacza sekwencję maszyn π według obliczonej funkcji priorytetu $(\sum_{i=1,n} c_{ij})/n$ rozmieszczając maszyny rozpoczynając od największej wartości priorytetu – w położeniu centralnym sekwencji π, do najmniejszej wartości priorytetu – położenia skrajne sekwencji π

Krok 1. Dla każdej maszyny $j \in \{1, 2, ..., n\}$ oblicz wartość priorytetu:

$$w(j) = (\sum_{i=1}^{n} c_{ij})/n .$$

Krok 2. Utwórz listę maszyn $(m_1, m_2, ..., m_n)$ według niemalejącej wartości funkcji priorytetu: $w(m_1) \leq w(m_2) \leq ... \leq w(m_n)$.

Krok 3. Wyznacz rozwiązanie π przypisując: $\pi(1) := m_1$, $\pi(n) := m_2$, $\pi(2) := m_3$, $\pi(n-1) := m_4$, $\pi(3) := m_5$, i tak dalej.

3.2. Algorytm tabu

Idea algorytmu tabu polega na lokalnej i powtarzalnej modyfikacji poprawianego rozwiązania, nazywanego rozwiązaniem aktualnym bądź bazowym, z jednoczesnym zapamiętywaniem wykonanych modyfikacji w celu uniknięcia powrotu do sprawdzonych rozwiązań. Przejście w procesie poszukiwania od rozwiązania aktualnego do zmodyfikowanego jest bardzo często nazywane ruchem. W pracy przyjęto, że jedna modyfikacja rozwiązania π (przypomnijmy: permutacja π określa kolejność rozmieszczenia maszyn w rzędach hali) polega na zamianie (ang. swap) położenia dwóch dowolnych maszyn w permutacji maszyn. Rozwiązanie otrzymane z rozwiązania π przez przestawienie maszyn $\{i, j\}$ jest dalej zawsze oznaczane symbolem $\pi(i, j)$. Tak więc otoczenie (sąsiedztwo) $N(\pi)$ rozwiązania π jest następującym zbiorem zmodyfikowanych rozwiązań :

$$N(\pi) = \{\pi(i, j): (i \neq j) \wedge (i, j \in \{1, 2, ..., n\}) \}$$

Ogólnie, w iteracji k, algorytm popraw *TS* wybiera „najlepszego" sąsiada y z otoczenia $N(\pi^{k-1})$ i niezależnie od relacji między wartościami funkcji celu $q(y)$ oraz $q(\pi^{k-1})$ przechodzi do nowego rozwiązania y, kładąc $\pi^k = y$. W trakcie wykonywania kolejnych iteracji pamiętane jest aktualnie najlepsze znalezione rozwiązanie π^{TS} i funkcja celu $q^{TS} = q(\pi^{TS})$.

W celu uniknięcia cyklu pewne rozwiązania z sąsiedztwa $N(\pi^k)$ uważa się za zabronione i nie bierze się pod uwagę podczas wyznaczania y. Są to rozwiązania, które były już rozwiązaniami lokalnie optymalnymi w ostatnich kilku - kilkunastu iteracjach. Rozwiązania te nie są bezpośrednio pamiętane, ale z reguły są pamiętane w postaci pewnych ich „atrybutów". Przyjęto, że atrybutem modyfikacji charakteryzującym ją i zapisywanym w pamięci krótkoterminowej *KLT* (krótkoterminowa lista tabu) jest przestawiana para $\{i, j\}$, stąd pamięć krótkoterminowa ma postać tablicy $KLT = [KLT_{ij}]_{n*n}$, gdzie $T \geq KLT_{ij} \geq 0$, KLT_{ij} – pozostały czas tabu pary $\{i, j\}$. Jeżeli nowe rozwiązanie $y = \pi^{k-1}(i, j)$ powstałe w wyniku modyfikacji $\{i, j\}$ zostało rozwiązaniem aktualnym, parze $\{i, j\}$ zostaje przypisany czas tabu o wartości T, tj. $KLT_{ij} = T$ – para $\{i, j\}$ otrzymuje status tabu na T iteracji, począwszy od iteracji k. Wartość KLT_{ij} jest zmniejszana o 1 po każdej iteracji i z chwilą spełnienia warunku $KLT_{ij} = 0$ zdjęty jest status tabu z pary $\{i, j\}$.

Zapisywanie na liście *KLT* atrybutów zmodyfikowanych rozwiązań i w konsekwencji traktowanie pewnych ruchów jako zabronionych ma, oprócz oczywistych zalet, także poważną wadę. Może to bowiem powodować zabronienie wykonania modyfikacji, która jest jednak interesująca z punktu widzenia dalszego procesu poszukiwania. Przykładowo, prowadzi do rozwiązań o wartości funkcji celu mniejszej niż dotychczas znaleziona. W celu uniknięcia tej wady wprowadza się kryterium aspiracji. Przyjęto, że funkcją aspiracji zmodyfikowanego rozwiązania $\pi(i, j)$ jest funkcja celu $q(\pi(i, j))$ oraz poziom aspiracji jest równy wartości funkcji celu q^{TS} najlepszego rozwiązania. Jeżeli dane przestawienie $\{i, j\}$ jest zabronione ($KLT_{ij} > 0$), ale wartość funkcji aspiracji rozwiązania $\pi(i, j)$ jest mniejsza niż poziom aspiracji, $q(\pi(i, j)) < q^{TS}$, to modyfikacja jest traktowana jako niezabroniona i akceptowana jako nowe aktualne rozwiązanie $\pi^k = \pi(i, j)$. W tym przypadku mamy pewność, że takie rozwiązanie jest badane po raz pierwszy.

F. Glover sugeruje uzupełnianie podstawowej wersji algorytmu *TS* dodatkowymi procedurami zwiększającymi efektywność procesu poszukiwania. Jedną z nich jest (por. [1]) „częstotliwościowa" pamięć długoterminowa $DLT = [DLT_{ij}]_{n*n}$ (długoterminowa lista tabu),

w której dla pary *{i, j}* zliczana jest liczba przestawień DLT_{ij} , liczona od początku procesu poszukiwania do aktualnej iteracji *k*, jeżeli przestawienie tej pary generuje nowe – zaakceptowane - rozwiązanie $\pi^k = \pi(i,j)$. Glover zauważył, że częste przestawianie tych samych elementów permutacji prowadzi do przeszukiwania tylko określonego „rejonu" zbioru rozwiązań i sugeruje wprowadzenie do optymalizowanej funkcji celu kary proporcjonalnej do wartości DLT_{ij}, co może skierować proces poszukiwania w nieeksplorowane rejony zbioru rozwiązań. Wykorzystując pamięć długoterminową *DLT*, zastosowano następującą funkcję oceny modyfikowanych rozwiązań w iteracji *k+1*:

$$q'(\pi(i,j)) = q(\pi(i,j)) + \alpha \frac{DLT(i,j)}{k},$$

gdzie α jest wagą funkcji kary określającą udział funkcji kary w ocenie nowego rozwiązania oraz DLT_{ij}/k jest względną częstotliwością przestawiania pary *{i, j}* w procesie poszukiwania.

Oznaczenia:

π_{st}	Rozwiązanie początkowe wyznaczone przez algorytm konstrukcyjny
q_k	Wartość funkcji celu w *k*-tej iteracji
π^{ST}	Rozwiązanie poprawione przez algorytm *TS*
q^{TS}	Najlepsza wartość funkcji celu znaleziona przez algorytm *TS* i poziom aspiracji
k, K	Licznik i maksymalna liczba wykonanych iteracji algorytmu *TS*
ch	Zmienna pomocnicza przyjmująca wartość *„TRUE"*, jeżeli zastosowano kryterium aspiracji
k_lok	Licznik iteracji wykonanych bez poprawy funkcji celu
MaxIt	Maksymalna liczba iteracji algorytmu *TS* bez poprawy funkcji celu
KLT, DLT	Krótkoterminowa i długoterminowa lista tabu
T	Okres tabu
α	Stała różnicowania procesu poszukiwań

Schemat algorytmu *TS* zawierającego pamięć krótkoterminową, długoterminową oraz kryterium aspiracji przedstawia się następująco:

Krok 1. Początek. Podstaw : $\pi := \pi_{st}$; $\pi^{TS} := \pi$; $q^{TS} := q(\pi)$; $k\_lok := 0$; $ch := FALSE$

Krok 2. Dla k:=1 do K wykonaj :

Wyznacz najlepsze rozwiązania nowe $\pi(i^*, j^*)$ i tabu $\pi(i', j')$ w otoczeniu :

$$\pi(i^*, j^*) = \arg\min\{q(\pi(i,j)) + \alpha \cdot \frac{DLT_{ij}}{k} : KLT_{ij} = 0\}$$

$$\pi(i', j') = \arg\min\{q(\pi(i,j)) : KLT_{ij} > 0\}$$

Podstaw : $\pi := \pi(i^*, j^*)$; $q_k := q(\pi)$.

Poprawa rozwiązań:

Jeżeli $q^{TS} > q_k$, to podstaw: $q^{TS} := q_k$; $\pi^{TS} := \pi$; $k\_lok := 0$;

Jeżeli $q^{TS} > q(\pi(i', j'))$, to podstaw: $ch := TRUE$; $k\_lok := 0$

$\pi := \pi(i', j')$; $q_k := q(\pi)$; $\pi^{ST} := \pi$; $q^{ST} := q_k$

Podstaw: $k\_lok := k\_lok + 1$ i jeżeli $k\_lok = MaxIt$ to $STOP$.

Korekta stanu pamięci :

Dla $\forall i, j$ podstaw: $KLT_{ij} := max\{0, KLT_{ij} - 1\}$.

Jeżeli ch = TRUE to podstaw: $KLT_{i'j'} := T$; $ch := FALSE$ i $DLT_{i'j'} := DLT_{i'j'} + 1$,

w przeciwnym przypadku: $KLT_{i^*j^*} := T$ i $DLT_{i^*j^*} := DLT_{i^*j^*} + 1$.

3.3. Metoda skoku powrotnego

Nowicki i Smutnicki (zobacz w [2]), pierwotnie dla gniazdowego problemu harmonogramowania, zaproponowali metodę łączącą technikę intensyfikacji i dywersyfikacji procesu poszukiwania, nazwaną metodą poszukiwania ze skokiem powrotnym. W trakcie działania klasycznego algorytmu *TS* pamiętamy określoną liczbę $Max\Omega$ ostatnio wyznaczonych rozwiązań lokalnie optymalnych (rozwiązań, które poprawiały wartość funkcji celu) w postaci listy Ω o strukturze stosu, wraz z modyfikacjami rozwiązań które jeszcze nie zostały sprawdzone w procesie poszukiwania. Po wykonaniu *MaxIt* iteracji bez poprawy wartości funkcji celu proces poszukiwań nie kończy się, ale następuje skok powrotny do ostatniego zapamiętanego rozwiązania lokalnie optymalnego. Następnie wychodząc z niego, po innej trajektorii niż poprzednio, proces poszukiwania jest kontynuowany. Precyzyjny opis metody skoku powrotnego można znaleźć w monografii Nowickiego [2].

4. Badania komputerowe

W przeprowadzonych eksperymentach obliczeniowych porównano efektywność różnych wersji algorytmu *TS*. W tablicy 1 przestawiono zestaw danych eksperymentalnych.

Tablica 1

Zestaw danych

Numer zestawu danych	nazwa zestawu maszyn (liczba)	parametry hali	
		H	e
1	m16	24	5
2	real16	34	4
3	m18	44	5
4	m18	44	5
5	m18	39	5
6	m22	69	5
7	m25	54	5
8	m28	34	8

Nazwa zestawu maszyn - kod alfanumeryczny, identyfikujący zestaw danych, zawiera liczbę mówiącą o ilości maszyn, np. m20 oznacza zestaw danych do rozmieszczenia 20 maszyn. Zestawy danych dostępne są na stronie internetowej www.homelt.republika.pl/dane.zip.

Parametry hali :

H- Długość hali produkcyjnej przeznaczona dla maszyn

e- Minimalne odstępy między rzędami maszyn

Tablica 2

Wyniki badań komputerowych

				Algorytm TS typ A			Algorytm TS typ B			Algorytm TS typ C				Algorytm TS typ D			
Numer zestawu danych	zestaw danych	Al. Kon.	Qst	T	l_A	Qts_A	T	l_B	Qts_B	alfa	T	l_C	Qts_C	DL	Lr	l_D	Qts_D
1	m16	2	4043,08	10	26	3697,55	10	27	3697,48	1500	10	50	3662,83	3	3	133	3661,39
2	real16	2	90241,96	10	13	75797,38	10	13	75797,38	200	5	13	75797,38	3	3	58	75764,50
3	m18	2	27263,08	10	30	25322,64	10	30	25322,64	500	10	34	25322,64	4	4	127	25192,78
4	m18	1	27263,08	5	30	25326,50	5	30	25326,50	200	5	29	25322,64	2	2	46	25322,64
5	m18	1	28837,10	5	21	24846,42	5	21	24846,42	100	5	16	24846,42	4	4	128	24234,41
6	m22	2	17215,42	10	30	15683,50	10	30	15683,50	350	10	30	15683,50	3	3	175	15088,13
7	m25	1	34290,83	5	33	30937,25	5	33	30937,25	450	5	62	30359,58	3	4	257	30357,70
8	m28	1	25673,41	10	41	24413,25	10	41	24413,25	300	10	54	24392,54	3	3	176	24391,64

W tablicach 2 i 3 oraz na rys. 2, 3, i 4 przedstawiono wyniki badań komputerowych czterech wersji algorytmu *TS* wyposażonego w następujące mechanizmy: A – tylko pamięć krótkoterminową; B – pamięć krótkoterminową i kryterium aspiracji; C – pamięć krótko- i

długoterminową oraz kryterium aspiracji; D – wersja C łącznie z mechanizmem skoku powrotnego.

W tablicy 2 zastosowano następujące oznaczenia:

Al. Kon. - rodzaj zastosowanego algorytmu konstrukcyjnego :1- Constr1, 2 – Constr2;

Qst - wartość funkcji celu uzyskana przez algorytm konstrukcyjny;

T - okres tabu zastosowany w algorytmie *TS*;

Qts_A, Qts_B, Qts_C, Qts_D – wartość funkcji celu uzyskana odpowiednio przez warianty A, B, C, i D algorytmu *TS*;

i_A, i_B, i_C, i_D – liczba iteracji po jakiej, odpowiednio, warianty A, B, C, D algorytmu *TS* osiągnęły rozwiązanie suboptymalne.

Tablica 3

Procentowe wskaźniki zysku

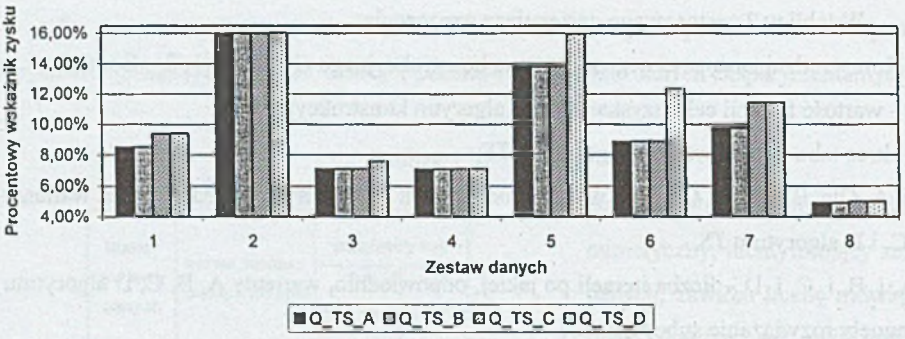
	A		B		C		D	
Numer zestawu danych	X-A	Q_TS_A %	X-B	Q_TS_B %	X-C	Q_TS_C %	X-D	Q_TS_D %
	średnio:	8.47%		8.47%		8.76%		9.44%
1	345.53	8.55%	345.60	8.55%	380.25	9.40%	381.69	9.44%
2	14444.58	16.01%	14444.58	16.01%	14444.58	16.01%	14477.46	16.04%
3	1940.44	7.12%	1940.44	7.12%	1940.44	7.12%	2070.30	7.59%
4	1936.58	7.10%	1936.58	7.10%	1940.44	7.12%	1940.44	7.12%
5	3990.68	13.84%	3990.68	13.84%	3990.68	13.84%	4602.69	15.96%
6	1531.92	8.90%	1531.92	8.90%	1531.92	8.90%	2127.29	12.36%
7	3353.58	9.78%	3353.58	9.78%	3931.25	11.46%	3933.13	11.47%
8	1260.16	4.91%	1260.16	4.91%	1280.87	4.99%	1281.77	4.99%

W tablicy 3 oraz na rys. 2 i 4 zastosowano następujące oznaczenia:

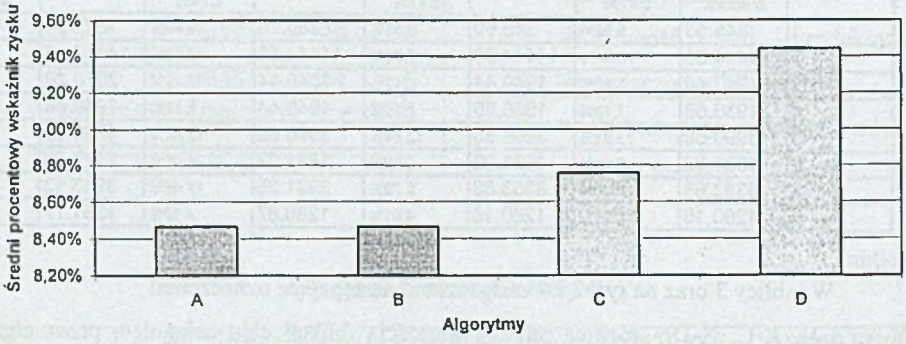
X-A, X-B, X-C, X-D - różnica między wartością funkcji celu osiągniętą przez algorytm konstrukcyjny i wartością funkcji celu osiągniętą przez warianty, odpowiednio A, B, C i D algorytmu *TS*;

Q_TS_A, Q_TS_B, Q_TS_C, Q_TS_D - procentowy wskaźnik zysku, jaki został otrzymany po zastosowaniu wariantu, odpowiednio A, B, C, i D algorytmu *TS* w odniesieniu do wyniku, jaki otrzymano po zastosowaniu algorytmu konstrukcyjnego;

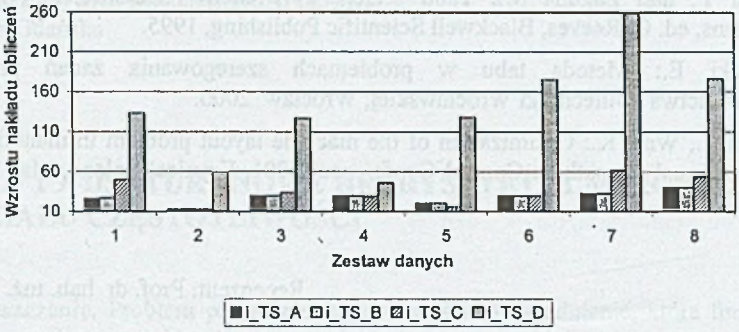
i_TS_A, i_TS_B, i_TS_C, i_TS_D - liczba iteracji wariantów A, B, C, D algorytmu *TS*, po której otrzymano najlepsze rozwiązanie.



Rys. 2. Procentowy wskaźnik zysk z zastosowania algorytmu TS typu A, B, C, D
Fig. 2. Percentage profit of use algorithms TS type A, B, C, D.



Rys. 3. Średni wskaźnik zysku otrzymanego w wyniku zastosowania algorytmu TS typu A,B,C,D
Fig. 3. Average percentage profit of use algorithms TS type A,B,C,D



Rys. 4. Wzrost wykonanej liczby iteracji przy zastosowaniu al. TS typu A,B,C,D
Fig. 4. Iteration growth of use algorithms TS type A,B,C,D

5. Podsumowanie badań komputerowych

Do sporządzenia zaprezentowanych zestawień wyników przeprowadzono wiele eksperymentów z wykorzystaniem różnych parametrów i zestawów danych. Zastosowanie metody skoku powrotnego tylko do niektórych problemów przyniosło wyraźną poprawę funkcji celu. Parametry zadań testowych, dla których zastosowanie metody skoku przyniosło zadowalające rezultaty, przedstawiono w tabl. 1 i 2, jednocześnie rys. 3 ilustruje procentową poprawę funkcji celu dla badanych wariantów algorytmu *TS*. Rysunek 4 prezentuje wzrost nakładu obliczeń (liczba wykonanych iteracji) przy zastosowaniu kolejnych wariantów algorytmu *TS*. Wydaje się, że pomimo wzrostu nakładu obliczeń, czasami nawet kilkukrotnego, koszty związane z czasem uzyskania rozwiązania są pomijalnie mniejsze w porównaniu z otrzymanymi przez zastosowanie metody skoku powrotnego zyskami w wartości funkcji celu.

LITERATURA

1. Glover F. and Laguna M.: Tabu search. In Modern Heuristics for Combinatorial Problems, ed. C. Reeves, Blackwell Scientific Publishing, 1995.
2. Nowicki E.: Metoda tabu w problemach szeregowania zadań produkcyjnych Wydawnictwa Politechniki Wrocławskiej, Wrocław 2000.
3. Homel T., Wala K.: Optimization of the machine layout problem in manufacturing cell, "International Carpathian Control Conference '2001, Krynica, Poland, May 22-25, 2001, pp. 139-144.

Recenzent: Prof. dr hab. inż. Adam Janiak

Abstract

The many-row machine layout problem in manufacturing cell is considered. We present the original model of the machine layout problem of combinatorial optimization type where the solution is formalized as a permutation of machines numbers $\pi = (\pi(1), \pi(2), ..., \pi(n))$ and the nonlinear objective function determines the total internal-cell transport cost. We propose two constructive algorithms and investigate efficiency of four improvement algorithm based on tabu search metaheuristic. Results of computing experiments illustrate the use of the proposed model and algorithms for the real size projects.