

# METODY PRZYSPIESZANIA PRZEGLĄDANIA OTOCZEŃ W ALGORYTMACH LOKALNYCH PRZESZUKIWANÍ

Wojciech BOŻEJKO, Mieczysław WODECKI

**Streszczenie:** Do rozwiązywania NP-trudnych problemów optymalizacji kombinatorycznej z powodzeniem stosuje się algorytmy przybliżone oparte na metodzie lokalnych poszukiwań. Zasadniczym elementem tych metod jest otoczenie. Sposób jego generowania i przeglądania ma duży wpływ na czas działania algorytmu oraz wartości wyznaczanych rozwiązań. Czas ten można znacznie skrócić zmniejszając otoczenia, tj. usuwając „nieperspektywiczne” elementy. W pracy przedstawiamy metody zmniejszania liczby elementów otoczeń w algorytmach rozwiązywania problemów optymalizacyjnych, dla których rozwiązaniami dopuszczalnymi są permutacje.

**Słowa kluczowe:** optymalizacja kombinatoryczna, metoda lokalnych poszukiwań, otoczenie.

## 1. Wstęp

Wiele praktycznych problemów związanych z procesem podejmowania decyzji z dziedziny planowania, zarządzania oraz sterowania sprowadza się do wyznaczenia permutacji optymalnej w zbiorze wszystkich permutacji pewnego skończonego zbioru elementów. W większości są to bardzo trudne (NP-trudne) problemy optymalizacji kombinatorycznej, dla których obecnie nie jest możliwa konstrukcja algorytmów ich rozwiązywania o wielomianowej złożoności obliczeniowej. Najważniejszymi, zarówno z praktycznego jak i teoretycznego punktu widzenia, należącymi do tej grupy zagadnień są:

1. problem komiwożazere (Traveling Salesman Problem – TSP),
2. kwadratowy problem przydziału (Quadratic Assignment Problem - QAP),
3. jedno i wielomaszynowe problemy szeregowania zadań (np. flow shop, job shop).

Ze względu na złożoność obliczeniową tych problemów, w praktyce do ich rozwiązywania stosuje się metody przybliżone. Najlepsze wyniki otrzymuje się konstruując algorytmy oparte na metodzie lokalnego przeszukiwania. Polega ona na polepszaniu bieżącego rozwiązania poprzez iteracyjne generowanie i przeszukiwanie otoczeń (podzbiorów zbioru rozwiązań dopuszczalnych) w celu znalezienia lepszego rozwiązania. Decydujący wpływ na czas obliczeń oraz jakość wyznaczanych rozwiązań mają otoczenia – metody ich generowania oraz wyznaczania elementu będącego rozwiązaniem startowym w następnej iteracji (zazwyczaj jest to przeszukiwanie w celu wyznaczenia najlepszego elementu).

Jeżeli  $\pi$  jest pewny elementem (permutacją) skończonego zbioru rozwiązań dopuszczalnych  $\Pi$ , to otoczenie  $N(\pi) \subset \Pi$  zawiera elementy powstałe przez wykonanie ruchów, tj. przekształcania permutacji  $\pi$ . Powszechnie stosowane („klasyczne”) ruchy polegają na:

1. zamianie dwóch elementów w permutacji (ruch typu zamień),
2. przestawieniu elementu w permutacji na inną pozycję (ruch typu wstaw),
3. wyjęciu podciągu z permutacji i odwrócenie kolejności jego elementów (np. 2-opt, 3-opt otoczenia w problemie TSP i QAP).

Generowane przez te ruchy otoczenia mają wielomianową liczbę elementów. W algorytmach lokalnych poszukiwań stosowane są także otoczenia o wykładniczej liczbie elemen-

tów. Są one generowane przez złożenie pojedynczych ruchów. Ze względu na stosowane metody ich przeglądania można wyróżnić dwa ich typy:

1. otoczenia, dla których istnieją algorytmy przeglądania o wielomianowej złożoności obliczeniowej (np. dynasearch [5]),
2. otoczenia, dla których problem wyznaczenia elementu optymalnego jest NP-trudny (np. dla TSP i QAP).

Obecnie, celem poprawy wartości wyznaczanych rozwiązań, w konstrukcji algorytmów opartych na metodzie lokalnych poszukiwań można wyróżnić dwie (pozornie sprzeczne) tendencje:

1. Pomniejszanie otoczeń (nawet tych o wielomianowej liczbie elementów).
2. Stosowanie otoczeń o wykładniczej liczbie elementów.

Na podstawie zamieszczonych w literaturze wyników można stwierdzić, że obie te metody, odpowiednio stosowane, są równie skuteczne.

Proces przeglądania otoczenia można znacznie przyspieszyć zmniejszając liczbę jego elementów, tj. eliminując (przez przegląd pośredni) z otoczenia „gorsze” elementy. Szczególnie może być to efektywne przy dużych otoczeniach. Cała operacja jest opłacalna, jeżeli algorytm eliminacji ma niewielką złożoność obliczeniową. W literaturze przedstawiono trzy sposoby wyznaczania ograniczonych otoczeń:

1. kryteria eliminacyjne – określenie relacji częściowego porządku na zbiorze elementów,
2. bloki – podciągi elementów, w których zamiana kolejności nie poprawia wartości funkcji celu,
3. reprezentanci ruchów – ruchy z pewnego podzbioru „lepsze” od innych.

Metody konstrukcji ograniczonych otoczeń ilustrujemy na przykładzie dwóch reprezentatywnych, silnie NP-trudnych problemów szeregowania zadań (z różnymi funkcjami celu):

- i. jednomaszynowego, z minimalizacją sumy kosztów opóźnień,
- ii. problemu przepływowego, z minimalizacją czasu zakończenia.

Przedstawiamy charakterystyczne dla tych problemów własności umożliwiające pominięcie, w procesie generowania, pewnych elementów z otoczeń.

## 2. Problemy szeregowania zadań

Problemy szeregowania zadań są intensywnie badane od ponad 30 lat. Pomimo prostoty sformułowania, w ogromnej większości należą do klasy problemów silnie NP-trudnych. Znajdują zastosowanie w wielu dziedzinach gospodarki związanych z procesem podejmowania decyzji. Są znaczące zarówno z teoretycznego jaki i praktycznego punktu widzenia.

### 2.1. Jednomaszynowy problem szeregowania

Dany jest zbiór  $n$  ponumerowanych zadań  $J=\{1,2, \dots, n\}$ , które należy wykonać, bez przerywania, na jednej maszynie. Maszyna ta, w dowolnej chwili, może wykonywać co najwyżej jedno zadanie. Dla zadania  $i$  ( $i=1,2, \dots, n$ ), niech  $p_i, w_i, d_i$  będą odpowiednio: *czasem wykonywania*, *wagą funkcji kosztów* oraz *linią krytyczną*. Jeżeli ustalona jest kolejność wykonywania zadań oraz  $C_i$  ( $i=1,2,\dots,n$ ) jest terminem zakończenia wykonywania zadania  $i$ , to  $T_i = \max\{0, C_i - d_i\}$  nazywamy *opóźnieniem*, a  $f_i(C_i) = w_i \cdot T_i$  *kosztem opóźnienia* zadania. Rozważany problem polega na wyznaczeniu takiej kolejności wykonywania zadań, która minimalizuje *sumę kosztów opóźnień*, tj.  $\sum w_i T_i$ .

Niech będzie  $\Pi$  zbiorem permutacji elementów z  $J$ . Dla permutacji  $\pi \in \Pi$  przez:

$$F(\pi) = \sum_{i=1}^n f_{\pi(i)}(C_{\pi(i)}),$$

oznaczamy *koszt permutacji* (tj. sumę kosztów opóźnień, gdy są one wykonywane w kolejności występowania w  $\pi$ ), gdzie:  $C_{\pi(i)} = \sum_{j=1}^i p_{\pi(j)}$ . Problem minimalizacji sumy kosztów opóźnień (total weighted tardiness problem - TWTP) sprowadza się do wyznaczenia permutacji optymalnej (o minimalnym koszcie) w zbiorze wszystkich permutacji  $\Pi$ .

W literaturze jest on oznaczany przez  $n \|1\| \sum w_i T_i$  i należy do klasy problemów *silnie NP-trudnych* (Lawler [10] oraz Lenstra i in. [11]). Algorytmy optymalne pozwalają rozwiązywać (w rozsądnym czasie) przykłady, w których liczba zadań jest nie większa niż 40 oraz dla pewnych specyficznych danych 50. Ze względu na ich małą efektywność, w praktycznych zastosowaniach, dużą rolę odgrywają algorytmy przybliżone. Najlepsze, oparte na metodzie lokalnych poszukiwań zamieszczone są w pracach Crauwels i in. [4] oraz Congram i in. [5]. Są szybkie i wyznaczają rozwiązania niewiele różniące się od optymalnych (dla referencyjnych przykładów).

## 2.2. Problem przepływowy

W permutacyjnym problemie przepływowym (*permutation flow shop* - PFS) każde z zadań należy wykonać kolejno na wszystkich maszynach, przy czym kolejność wykonywania zadań na każdej maszynie musi być taka sama. Optymalizacja polega na wyznaczeniu kolejności wykonywania zadań, która minimalizuje całkowity czas ich wykonywania. W literaturze problem ten jest oznaczany przez  $F|m|C_{\max}$ . W pracy [7] udowodniono, że jeżeli liczba maszyn wynosi trzy lub więcej, wówczas jest on silnie NP-trudny.

Stosując oznaczenia z pracy Grabowski [8] oraz Nowicki i Smutnicki [12] PFS można sformułować następująco. Dany jest zbiór  $n$  zadań  $J = \{1, 2, \dots, n\}$  oraz zbiór  $m$  maszyn  $M = \{1, 2, \dots, m\}$ . Zadanie  $j \in J$  jest ciągiem  $m$  operacji  $O_{j1}, O_{j2}, \dots, O_{jm}$ . Operację  $O_{jk}$  należy wykonać, bez przerywania, na maszynie  $k$  w czasie  $p_{jk}$ . Wykonywanie zadania na maszynie  $k$  (dla  $k=2, \dots, m$ ) może się rozpocząć dopiero po zakończeniu wykonywania tego zadania na maszynie  $k-1$ . Należy wyznaczyć kolejność, minimalizującą czas wykonania wszystkich zadań.

Niech  $\pi = (\pi(1), \pi(2), \dots, \pi(n))$  będzie pewną permutacją zadań, a  $\Pi$  zbiorem wszystkich permutacji elementów z  $J$ . Każda permutacja  $\pi \in \Pi$  wyznacza jednoznacznie kolejność wykonywania zadań na maszynach (na każdej taką samą). Tak więc, w omawianym problemie, należy wyznaczyć permutację  $\pi^* \in \Pi$  taką, że:

$$C_{\max}(\pi^*) = \max_{\pi \in \Pi} C_{\max}(\pi),$$

gdzie  $C_{\max}(\pi)$  jest terminem zakończenia wykonywania wszystkich zadań na maszynach, gdy są one wykonywane w kolejności  $\pi$  (tj. zadanie  $\pi(i)$  jest wykonywane jako  $i$ -te w kolejności,  $i=1, 2, \dots, n$ ).

Algorytmy optymalne pozwalają na rozwiązywanie, w rozsądnym czasie, przykładów o rozmiarze nie większym niż 20 zadań i 5 maszyn. W ostatnich latach opublikowano wiele algorytmów przybliżonych. Najlepszy z nich, oparty na metodzie poszukiwań z zabronieniami, jest zamieszczony w pracy Grabowski i Wodecki [9].

### 2.3 Metody lokalnych poszukiwań

Obecnie najlepszymi znanymi w literaturze metodami konstrukcji algorytmów przybliżonych dla rozwiązywania NP-trudnych problemów optymalizacji dyskretnej są metaheurystyki: poszukiwania z zabronieniami (tabu search) oraz symulowane wyżarzanie (simulated annealing) oparte na metodzie lokalnych przeszukiwań.

Przez  $\Pi$  oznaczmy zbiór rozwiązań dopuszczalnych. Niech  $\pi \in \Pi$  będzie dowolnym rozwiązaniem, a  $N(\pi) \subset J$  jego otoczeniem.

**Algorytm 2.1.** Metoda lokalnych przeszukiwań

Wyznacz rozwiązanie startowe  $\pi \in \Pi$ ;

$\pi^* \leftarrow \pi$ ;

**repeat**

Wybierz element  $\beta \in N(\pi)$ ;

**if**  $F(\beta) < F(\pi^*)$  **then**

$\pi^* \leftarrow \beta$ ;

$\pi \leftarrow \beta$ ;

**until** *Warunek\_Zakończenia*.

Procedura wyboru elementu z otoczenia (tj. algorytm generowania i przeglądania) ma decydujący wpływ na wyznaczane rozwiązania oraz złożoność obliczeniową metody lokalnych przeszukiwań.

### 3. Ograniczone otoczenia

W rozdziale tym przedstawimy trzy obecnie stosowane metody eliminacji pewnych elementów otoczenia (wyznaczania ograniczonych otoczeń). Sprowadzają się one do przeglądu pośredniego rozwiązań. Muszą być szybkie i proste w implementacji, aby ich stosowanie skróciło czas działania całego algorytmu, bez pogarszania wyników.

#### 3.1. Kryteria eliminacyjne

Kryteria eliminacyjne są głównie stosowane przy rozwiązywaniu problemów jednomaszynowych z sumokosztowymi funkcjami celu. W tym przypadku algorytmy ich wyznaczania mają złożoność obliczeniową co najwyżej  $O(n^2)$ . Sposób ich konstrukcji przedstawimy na przykładzie problemu TWTP.

Na zbiorze zadań  $J$  wprowadzamy *relację częściowego porządku*  $\mathfrak{R}$ .

Zadanie  $i$  jest w relacji z zadaniem  $j$ , tj.  $i\mathfrak{R}j$  wtedy i tylko wtedy, gdy istnieje permutacja optymalna, w którym zadanie  $i$  poprzedza  $j$ .

Niech

$$\Gamma_i^+ = \{j \in N : i\mathfrak{R}j\} \text{ oraz } \Gamma_i^- = \{j \in N : j\mathfrak{R}i\},$$

będą odpowiednio zbiorami *następników* oraz *poprzedników* zadania  $i$  w relacji  $\mathfrak{R}$ .

Relacja  $\mathfrak{R}$  pozwala na eliminację wielu elementów z otoczenia, bez utraty rozwiązań optymalnych. Jeżeli  $i\mathfrak{R}j$  wówczas generuje się tylko takie permutacje, w których element

$i$  poprzedza  $j$ . W literaturze przedstawiono różne własności pozwalające ustalić relację  $\mathfrak{R}$  na zbiorze  $J$ . Przykładem może być poniższe twierdzenie.

Dwa zadania są w relacji  $\mathfrak{R}$ , jeżeli spełniają jeden z warunków (zwanych kryteriami eliminacyjnymi) twierdzenia.

**Twierdzenie 1.** Jeżeli dla pary zadań  $i, j \in J$  spełniony jest jeden z warunków:

- (a)  $p_i \leq p_j, w_i \geq w_j, d_i \leq d_j$ , lub
- (b)  $p_i \leq p_j, w_i \geq w_j, d_i \leq \sum_{l \in \Gamma_j^-} p_l + p_j$ , lub
- (c)  $w_i \geq w_j, d_i \leq d_j, d_i \geq \sum_{l \in N \setminus \Gamma_j^+} p_l + p_j$ ,

to istnieje permutacja optymalna, w której zadanie  $i$  poprzedza  $j$ . ■

**Dowód.** Warunek (a) pochodzi z pracy Shwimera [14], a (b) i (c) są uogólnionymi wersjami Twierdzenia 1 i 2 z pracy Emmons [6].

Jeżeli dla pary zadań  $i, j \in N$  zachodzi relacja  $i\mathfrak{R}j$ , to  $i \in \Gamma_j^-$  oraz  $j \in \Gamma_i^+$ . Aby nie powstał cykl, po ustaleniu każdej nowej relacji (pomiędzy parą zadań), wykonujemy *transzytywne domknięcie* modyfikując zbiory:

$$\begin{aligned} \forall l \in \{i\} \cup \Gamma_i^-, \Gamma_l^+ &\leftarrow \Gamma_l^+ \cup \{j\} \cup \Gamma_j^+, \\ \forall l \in \{j\} \cup \Gamma_j^+, \Gamma_l^- &\leftarrow \Gamma_l^- \cup \{i\} \cup \Gamma_i^-. \end{aligned}$$

Powyższe kryteria są, od wielu lat, z powodzeniem stosowane w najlepszych algorytmach (dokładnych jak i przybliżonych). W pracach [1] oraz [13] przedstawiono pewne własności pozwalających na konstrukcję wielu podobnych kryteriów. Brak jednak analizy ich „siły”, tj. jak wiele nowych relacji wnoszą one do zbioru  $\mathfrak{R}$ .

Odpowiednio zmodyfikowane, opisane w tym rozdziale, własności są z mniejszym powodzeniem stosowane w algorytmach rozwiązywania innych jednomaszynowych problemów szeregowania zadań z sumokosztowymi funkcjami celu.

Podobne kryteria można także sformułować dla problemów wielomaszynowych. Jednak ze względu na niewielką ich efektywność (duża złożoność obliczeniowa oraz mała liczba uzyskanych relacji) nie są one, jak na razie, w praktyce stosowane.

### 3.2. Bloki

Elementy otoczenia  $N(\pi)$  powinny być generowane z  $\pi$  przez przekształcenia umożliwiające szybkie obliczanie wartości funkcji celu, dla wyznaczonego elementu. W przypadku permutacji jest to zamiana pozycjami elementu (lub elementów) albo przestawienie na inną pozycję elementu (lub elementów). Bloki są podciągami elementów w permutacji, w których zmiana kolejności elementów nie generuje permutacji o mniejszej wartości funkcji celu. Po wyznaczeniu bloków, na etapie generowania otoczenia można więc pominąć wiele „gorszych” elementów.

#### **Jednomaszynowy problem TWTP**

W dowolnej permutacji  $\pi \in \Pi$  istnieją podpermutacje (podciągi zadań) takie, że:

1. wykonywanie każdego zadania z podpermutacji kończy się przed jego linią krytyczną (wszystkie zadania są terminowe), albo

2. wykonywanie każdego zadania z podpermutacji kończy się za jego linią krytyczną (wszystkie zadania są opóźnione).

Podpermutacje te nazywamy *blokami*.

Blok zadań  $\pi^T$  z permutacji  $\pi \in \Pi$  nazywamy *T-blokiem*, jeżeli:

- a) każde zadanie  $j \in \pi^T$  jest terminowe oraz  $d_j \geq C_{last}$ , gdzie  $C_{last}$  jest terminem zakończenia wykonywania ostatniego zadania z  $\pi^T$ ,
- b)  $\pi^T$  jest maksymalną podpermutacją spełniającą ograniczenie a).

Jest łatwo zauważyć, że jeżeli  $\pi^T$  jest *T-blokiem*, to  $\min\{j \in \pi^T : d_j\} \geq C_{last}$ .

Blok zadań  $\pi^D$  w permutacji  $\pi \in \Pi$  nazywamy *D-blokiem*, jeżeli:

- a) każde zadanie  $j \in \pi^D$  jest spóźnione oraz  $d_j < C_{first} + p_j$ , gdzie  $C_{first}$  jest terminem rozpoczęcia wykonywania pierwszego zadania z  $\pi^D$ ,
- b)  $\pi^D$  jest maksymalną podpermutacją spełniającą ograniczenie a').

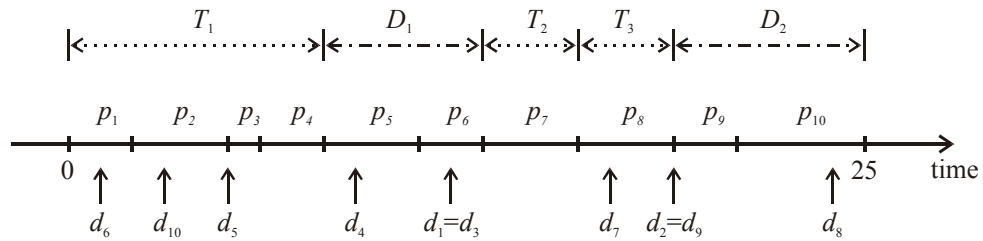
Łatwo udowodnić, że w dowolnej permutacji zadań z  $\pi^D$ , każde zadanie jest w permutacji  $\pi$  opóźnione.

Rozpatrując kolejno elementy w permutacji (np. zaczynając od pierwszego) można ją rozbić na bloki (*T-bloki* oraz *D-bloki*). Taki algorytm ma złożoność obliczeniową  $O(n)$ .

**Przykład.** Parametry zadań, dla problemu TWTP są zamieszczone w tabeli poniżej.

$i$	1	2	3	4	5	6	7	8	9	10
$p_i$	2	3	1	2	3	2	3	3	2	4
$d_i$	12	19	12	9	5	1	17	24	19	3

Niech  $\pi = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$ . Permutacja  $\pi$  ma trzy *T-bloki*:  $T_1 = (1, 2, 3, 4)$ ,  $T_2 = (7)$ ,  $T_3 = (8)$  oraz dwa *D-bloki*:  $D_1 = (5, 6)$ ,  $D_2 = (9, 10)$ . Są one przedstawione na Rysunku 1.



Rys. 1. Bloki permutacji  $\pi = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$ .

**Lemat 1.** Jeżeli w *D-bloku*  $\pi^D$ , zadania występują według nierosnących wartości ilorazów  $w_{\pi(i)} / p_{\pi(i)}$  ( $\pi(i) \in \pi^D$ ), to jest to optymalne uszeregowanie zadań w tym bloku. ■

**Twierdzenie 1.** [2] Jeżeli w permutacji  $\pi \in \Pi$  zadania w *D-blokach* są uszeregowane optymalnie, to dowolna zmiana kolejności elementów w dowolnym bloku nie generuje permutacji o mniejszej wartości funkcji celu (od  $F(\pi)$ ). ■

Przeprowadzone eksperymenty obliczeniowe wskazują, że stosując Twierdzenie 1 można z otoczenia generowanego przez ruchy typu zamień lub wstaw wyeliminować około 40% elementów. Szereg własności oraz zastosowanie bloków w algorytmie rozwiązywania problemu TWTP przedstawiono w pracy Bożejko, Grabowski i Wodecki [2].

Podobne własności można udowodnić dla innych jednomaszynowych problemów szeregowania zadań, np. dla ważnych z punktu widzenia zastosowań problemów E/T (earlyness-tardiness problems).

Wyznaczenie pewnych podciągów w permutacji i udowodnienie podobnych własności, jak w Twierdzeniu 1, jest dla problemów wielomaszynowych znacznie trudniejsze. Bloki zawierają małą liczbę elementów, a ich wyznaczenie jest czasochłonne. Dlatego, w implementacji najlepszych obecnie algorytmów nie są one stosowane.

### Problem przepływowy PFS

Korzystając z definicji i oznaczeń zamieszczonych w pracach Grabowski [8] oraz Nowicki i Smutnicki [12], dla każdej permutacji  $\pi \in \Pi$  definiujemy digraf:

$$D(\pi) = (O, AV(\pi) \cup AH(\pi)),$$

gdzie zbiór wierzchołków  $O = \{O_{j1}, O_{j2}, \dots, O_{jm}\}$ . Waga wierzchołka  $O_{jk}$  jest równa  $p_{jk}$  (czas wykonywania operacji). Natomiast, zbiór łuków jest sumą:

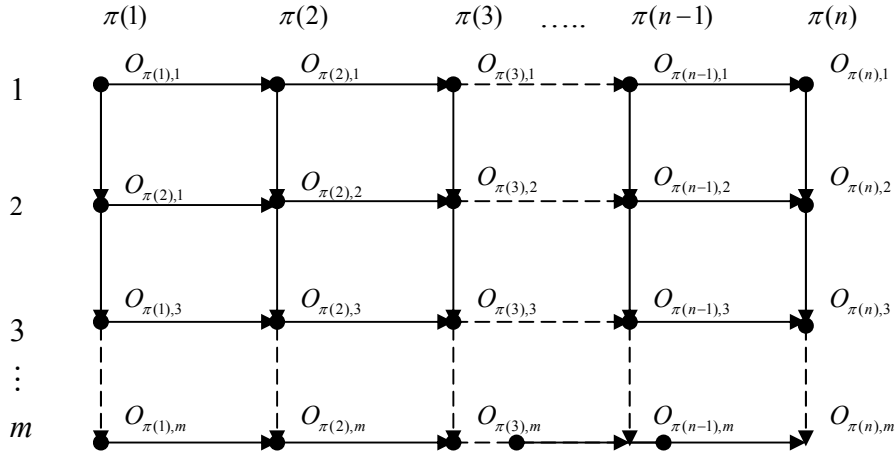
$$AV(\pi) = \bigcup_{i=1}^{m-1} \bigcup_{j=1}^n \{(O_{\pi(j)i}, O_{\pi(j+1)i})\},$$

zbioru łuków pionowych oraz

$$AH(\pi) = \bigcup_{i=1}^m \bigcup_{j=1}^{n-1} \{(O_{\pi(j)i}, O_{\pi(j+1)i})\},$$

zbioru łuków poziomych.

Graf ten jest schematycznie przedstawiony na Rysunku 2.



→ łuki poziome ze zbioru  $AH(\pi)$  - kolejność wykonywania zadań,  
 ↓ łuki pionowe ze zbioru  $AV(\pi)$  - kolejność wykonywania operacji.

Rysunek 2. Graf skierowany  $D(\pi)$ .

Najdłuższa droga w digrafie  $D(\pi)$  z wierzchołka  $O_{\pi(1)l}$  do  $O_{\pi(n)m}$  zwana jest *drogą krytyczną* w permutacji  $\pi$ . Można łatwo wykazać, że jej długość jest równa  $C_{\max}(\pi)$ . Drogi tę (permutację  $\pi$ ) można rozbić na spójne podciągi zadań  $B_1, B_2, \dots, B_m$  zwane *blokami*, przy czym:

- a)  $B_k = (\pi(f_k), \pi(f_k + 1), \dots, \pi(l_k - 1), \pi(l_k))$ ,  $f_k \leq l_k$ ,  $f_1 = 1$ ,  $l_k = k$ ,  $\pi(l_k) = \pi(f_{k+1})$ ,  
 $k = 1, 2, \dots, m-1$ ,
- b)  $B_k$  zawiera operacje wykonywane na tej samej maszynie,  $k = 1, 2, \dots, m$ ,
- c) każde dwa bloki zawierają operacje wykonywane na różnych maszynach.

Operacje  $\pi(f_k)$  oraz  $\pi(l_k)$  bloku  $B_k$  nazywamy odpowiednio *pierwszą* i *ostatnią operacją* bloku. Z twierdzeń zamieszczonych w pracy Grabowskiego [8] wynika, że aby z permutacji  $\pi$  otrzymać permutację o mniejszej długości drogi krytycznej należy pewne zadanie z pewnego bloku przestawić przed pierwsze lub za ostatnie zadanie tego bloku.

Generując otoczenie  $N(\pi)$  permutacji  $\pi$  można pominąć te z permutacji, które różnią się od  $\pi$  kolejnością zadań w pewnym bloku z  $\pi$ .

W najlepszych algorytmach Grabowski i Wodecki [9] oraz Nowicki i Smutnicki [12] rozwiązywania problemu PFS w generowaniu otoczeń są z powodzeniem stosowane własności eliminacyjne bloków.

### 3.3. Reprezentanci ruchów

Metody wyznaczania reprezentantów ruchów przedstawimy na przykładzie problemu TWTP oraz otoczeń generowanych przez ruchy typu „wstaw” (insert). Jeżeli permutacja  $\pi \in \Pi$  oraz  $k, l \in J$ ,  $k < l$ , to ruch *wstaw*  $i_l^k$ , przestawia zadanie  $\pi(k)$  z pozycji  $k$  na pozycję  $l$  (zadania  $\pi(k+1), \pi(k+2), \dots, \pi(l)$  są przesuwane o jedną pozycję w lewo). Ruch  $i_l^k$  generuje permutację  $\pi_l^k = i_l^k(\pi) \in N(\pi)$ . Podobnie określa się permutację generowaną przez ruch  $i_l^k$ , dla  $k > l$ .

**Twierdzenie 2.** [3] Dla ciągu ruchów  $i_1^k, i_2^k, \dots, i_{t(k)}^k$  zadania  $\pi(k)$  zachodzi:

$$F(\pi_1^k) \geq F(\pi_2^k) \geq \dots \geq F(\pi_{t(k)}^k),$$

gdzie  $t(k) = \max \{j : 1 \leq j \leq n \text{ oraz } C_{\pi_j^k(j)} \leq d_{\pi(k)}\}$ . ■

Ruch  $i_{t(k)}^k$  jest więc reprezentantem ciągu  $i_1^k, i_2^k, \dots, i_{t(k)-1}^k, i_{t(k)}^k$ .

Następne twierdzenie ilustruje związek bloków z reprezentantami ruchów.

**Twierdzenie 3.** [3] Niech  $\pi(a)$  i  $\pi(b)$  będą pierwszym i ostatnim zadaniem  $T$ -bloku w permutacji  $\pi$ . Jeżeli  $1 \leq k < a$  oraz  $a \leq t(k) \leq b$ , to dla ciągu ruchów

$i_{t(k)+1}^k, i_{t(k)+2}^k, \dots, i_{b-1}^k, i_b^k$  zachodzi:

$$F(\pi_{t(k)+1}^k) \leq F(\pi_{t(k)+2}^k) \leq \dots \leq F(\pi_{b-1}^k) \leq F(\pi_b^k).$$

Parametr  $t(k)$  jest określony w Twierdzeniu 2. ■



Ruch  $i_{t(k)+1}^k$  jest reprezentantem ciągu  $i_{t(k)+1}^k, i_{t(k)+2}^k, \dots, i_{b-1}^k, r_b^k$ .

W pracy Bożejko i Wodecki [3] udowodniono podobne kryteria wyznaczania reprezentantów dla  $T$  i  $D$  bloków oraz inaczej definiowanych ruchów. Przeprowadzone eksperymenty obliczeniowe wskazują, że stosując bloki oraz reprezentantów ruchów można wyeliminować nawet do 75% elementów z otoczenia.

Obecnie, w najlepszych algorytmach rozwiązywania problemów wielomaszynowych nie są stosowane własności reprezentantów ruchów.

#### 4. Podsumowanie

W pracy przedstawiono podstawowe metody wyznaczania ograniczonych otoczeń stosowanych w algorytmach lokalnych poszukiwań. Dzięki eliminacji (przez przegląd pośredni) „gorszych” elementów można znacznie skrócić czas działania algorytmu. Niestety, ale metody eliminacji elementów są mało uniwersalne i mocno związane ze specyfiką problemu. Ponadto, dla pewnych problemów wielomaszynowych koszt ich stosowania (przy dużej złożoności obliczeniowej algorytmu) jest znacznie większy niż zysk wynikający z przeglądania mniejszych otoczeń. Ze względu jednak na stosowanie w algorytmach lokalnych poszukiwań otoczeń o wykładniczej liczbie elementów (np. do dywersyfikacji obliczeń [9]), badania nad ich ograniczaniem powinny być kontynuowane.

Praca naukowa finansowana częściowo ze środków KBN w latach 2003-2006 jako projekt badawczy, nr 4T11A01624

#### Literatura:

1. Adrabiński A., Grabowski J., Wodecki M.: Algorytm rozwiązywania zagadnienia kolejnościowego postaci  $n || 1 || \sum w_i T_i$ , Archiwum Automatyki i Telemechaniki, t. 23, z.4, 1988, 623-635.
2. Bożejko W., Grabowski J., Wodecki M.: Block approach-tabu search algorithm for single machine total weighted tardiness problem, 2005, (praca w redakcji).
3. Bożejko W., Wodecki M.: Ograniczone otoczenia w algorytmach lokalnych poszukiwań, 2005, (praca w redakcji).
4. Crauwels H.A.J., Potts C.N., Van Wassenhove L.N.: Local Search Heuristics for the Single Machine Total Weighted Tardiness Scheduling Problem, INFORMS Journal on Computing, Vol. 10, No. 3, 1998.
5. Congram R.K., Potts C.N., Van Wassenhove S.L.: An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem, INFORMS Journal on Computing, 14(1), 2002, 52-67.
6. Emmons H.: One-Machine Sequencing to Minimize Certain Functions of Job Tardiness, Operations Research, 17, 1969, 701-705.
7. Garey M.R., Johnson D.S., Seti R.: The complexity of flowshop and jobshop scheduling, Mathematics of Operations Research, 1, 1976, 117-129.
8. Grabowski J.: A new algorithm of solving the flow-shop problem, Operations Research in Progress, J.D. Reidel Publishing Company, 1982, 57-75.
9. Grabowski J. Wodecki M.: A very Fast tabu search algorithm for the permutation flow shop problem with makespan criterion, Computers and Operations Research, 31, 2004, 1891-1909.

10. Lawler E.L.: Efficient Implementation of Dynamic Programming Algorithms for Sequencing Problems, Report BW106, Mathematisch Centrum, Amsterdam, 1979.
11. Lenstra, J.K. Rinnoy Kan, Brucker P.: Complexity of Machine Scheduling Problems, Annals of Discrete Mathematics, 1, 1977, 343-362.
12. Nowicki E., Smutnicki C.: A fast tabu search algorithm for the permutation flow-shop problem, European Journal of Operational Research, 91, 1996, 160-175.
13. Selim Akturk, Bayram Yildirim M.: A new dominance rule for the total total weighted tardiness problem, Production Planning & Control, vol. 10, no. 2, 1999, 138-149.
14. Shwimer J.: On the N-Job, One-Machine, Sequence-Independent Scheduling Problem with tardiness Penalties: a Branch-and-Bound Solution, Management Sci., 18, 1972, 301-313.

dr Wojciech Bożejko  
Instytut Cybernetyki Technicznej  
Politechniki Wrocławskiej  
ul. Janiszewskiego 11/17, 50-372 Wrocław  
e-mail: [wbo@ict.pwr.wroc.pl](mailto:wbo@ict.pwr.wroc.pl)

dr Mieczysław Wodecki  
Instytut Informatyki  
Uniwersytetu Wrocławskiego  
ul. Przesmyckiego 20, 51-151 Wrocław  
e-mail: [mwd@ii.uni.wroc.pl](mailto:mwd@ii.uni.wroc.pl)