

**Łukasz Indykiewicz**  
**Karol Kozłowski**  
**Stanisław Kożuchowicz**  
**Karol Nikšcin**  
Mateusz Zarzyczny  
Daniel Zasępa  
Łukasz Sierant  
ktoś

# Algorytmy konstrukcyjne w problemach gniazdowych

## Wstęę

Algorytmy przybliżone stosowane do rozwiązywania problemów gniazdowych można podzielić na dwie grupy:

- algorytmy konstrukcyjne (ang. constructive algorithms)
- algorytmy lokalnego poszukiwania (ang. local search algorithms)

Algorytmy konstrukcyjne zgodnie z pewnymi regułami budują „częściowe” rozwiązania i w momencie otrzymania pełnego rozwiązania kończą swoje działanie. W przypadku problemów szeregowania zadań budowa częściowego rozwiązania polega na szeregowaniu kolejnych operacji aż do uszeregowania ich wszystkich (rozwiązanie pełne). Algorytmy z drugiej grupy startują z pewnego rozwiązania (wybranego arbitralnie lub otrzymanego przez algorytmy konstrukcyjne) i następnie w kolejnych krokach poprawiają to rozwiązanie.

Algorytmy popraw, a w szczególności algorytmy oparte na technice *tabu search*, produkują znacznie lepsze rozwiązania niż algorytmy konstrukcyjne. Wymagają one jednak większych nakładów obliczeniowych. Intensywny wzrost mocy obliczeniowej komputerów sprawia, że czynnik ten staje się coraz mniej istotny w przypadku, gdy zadowala nas dobre rozwiązanie przybliżone. Jeżeli jednak chcemy znaleźć rozwiązanie optymalne, to konieczność wielokrotnego wykorzystania algorytmu przybliżonego w trakcie działania algorytmu dokładnego powoduje, że musimy preferować algorytmy konstrukcyjne. Co więcej, występują także takie sytuacje produkcyjne (np. awarie), kiedy bardzo szybkie algorytmy są niezbędne.

W tym sprawozdaniu zajmiemy się tylko algorytmami konstrukcyjnymi. Skupimy się na algorytmach dla klasycznego problemu gniazdowego oraz

problemu gniazdowego z maszynami równoległymi.

## Klasyczny problem gniazdowy

Algorytmy konstrukcyjne dla problemu gniazdowego dzielimy na dwie klasy:

- algorytmy priorytetowe
- algorytmy typu wstaw

Do pierwszej klasy należą algorytmy, które w danej iteracji wybierają jedną operację (powiedzmy  $j$ ) wśród operacji jeszcze nieuszeregowanych i możliwych aktualnie do uszeregowania (tzn. operacji, których wszystkie poprzedniki zostały już uszeregowane). Operacja  $j$  jest lokowana na maszynie  $\mu(j)$ , za ostatnią operacją w permutacji częściowej operacji już uszeregowanych. Do wyznaczenia operacji  $j$  stosuje się różnego rodzaju reguły priorytetowe SPT, LPT, MWR, RANDOM itp. Z reguły złożoność obliczeniowa algorytmów priorytetowych jest rzędu  $O(n^2)$ .

Algorytmy z klasy drugiej wykorzystują technikę wstawień użytą po raz pierwszy przy konstrukcji algorytmu NEH, dla klasycznego problemu przepływowego. Ich główna idea polega na tym, że operacji  $j$  w danej iteracji algorytmu nie umieszcza się na pozycji za ostatnią operacją w permutacji częściowej, lecz pozycja ta podlega wyborowi; często operacja  $j$  jest próbnie wstawiana na wszystkie pozycje zajęte przez uszeregowane wcześniej operacje na maszynie  $\mu(j)$ . Ogólnie złożoność obliczeniowa algorytmów typu wstaw jest rzędu  $O(n^3)$ . Wykorzystując jednak pewne specyficzne własności problemu, można ten czas zredukować do  $O(n^2)$ .

## Algorytmy priorytetowe

Ogólnie mówiąc każdy algorytm priorytetowy  $P(R)$  w kolejnej iteracji szereguje dokładnie jedną operację  $j \in O$  wykorzystując pewną regułę priorytetową  $R$ . Istotnym elementem w opisie tego algorytmu jest pojęcie *operacji gotowych* do uszeregowania oraz operacji *konfliktowych*.

Operacja  $j \in O$  jest gotowa do uszeregowania, jeżeli wszystkie jej poprzedniki zostały już uszeregowane. Niech dla danego zbioru  $U$ ,  $U \in O$ ,

$$OG(U) = \{i \in O \setminus U : B_i \subset U\}$$

oznacza zbiór operacji, które nie należą do tego zbioru, a których wszystkie

bezpośrednie poprzedniki znajdują się w tym zbiorze. Zauważmy, że jeżeli operacje ze zbioru  $U$  są już uszeregowane, to operacje z  $OG(U)$  są gotowe do uszeregowania. W sytuacjach skrajnych, gdy  $U = \emptyset$  lub  $U = O$ , zbiór  $OG(U)$  zawiera odpowiednio operacje nie mające poprzedników lub jest pusty.

Przejdziemy teraz do zdefiniowania operacji konfliktowej. W tym celu wprowadzimy kilka dodatkowych pojęć. Dla ustalonego zbioru  $U$  operacji uszeregowanych niech  $S(i)$  oznacza moment rozpoczęcia wykonywania operacji  $i \in U$ , zaś  $t_l$  oznacza moment zakończenia wykonywania ostatniej uszeregowanej operacji na maszynie  $l$ ,  $l \in M$ ; jeśli na tej maszynie nie jest jeszcze uszeregowana żadna operacja, to  $t_l = 0$ . Wtedy dla każdej operacji  $j$  gotowej do uszeregowania ( $j \in OG(U)$ ),

$$r(j) = \max \left\{ \max_{i \in B_j} (S(i) + p_i), t_{\mu(j)} \right\}$$

określa najwcześniejszy możliwy moment rozpoczęcia jej wykonywania (tzn. jej głowę). Ostatecznie, niech

$$\Delta = \min_{i \in OG(U)} (r(i) + p_i)$$

oznacza najmniejszy z najwcześniejszych możliwych momentów zakończenia wykonywania operacji ze zbioru  $OG(U)$ , zaś

$$M' = \{ l \in M : \forall_{i \in OG(U)} \mu(i) = l, r(i) + p_i = \Delta \}$$

zbiór maszyn, na których mogą być wykonywane operacje z najwcześniejszym możliwym momentem zakończenia wykonywania równym  $\Delta$ .

Operację  $j \in O$  będziemy nazywać operacją konfliktową, jeżeli: maszyna  $\mu(j)$ , na której może być ona wykonywana

- znajduje się w zbiorze  $M'$
- jest gotowa do uszeregowania
- jej najwcześniejszy możliwy moment rozpoczęcia wykonywania  $r(j)$  jest mniejszy niż  $\Delta$ .

Definicja ta pozwala wyróżnić dla poszczególnych maszyn  $l \in M$  zbiory operacji konfliktowych

$$OK(l) = \{ i \in OG(U) : \mu(i) = l, r(i) < \Delta \}$$

Algorytm  $P(R)$  w każdej iteracji, po arbitralnym wyborze maszyny  $l' \in M'$ , szereguje jedną z operacji konfliktowych ze zbioru  $OK(l')$ . Konflikt

polega na tym, że każda z operacji konfliktowych jest potencjalną kandydatką do uszeregowania. Dopiero zastosowanie odpowiedniej reguły priorytetowej  $R$  pozwala wybrać jedną z nich.

## Algorytm $P(R)$

**Krok 0.** Połóż  $\pi_l := \emptyset$  dla  $l \in M$ ,  $r(i) := 0$  dla  $i \in O$  oraz  $U := \emptyset$

**Krok 1.** Wyznacz zbiór operacji gotowych do uszeregowania  $OG(U)$ , moment czasowy  $\Delta$  oraz zbiór maszyn  $M'$ . Wybierz pewną maszynę  $l' \in M'$  i wyznacz dla niej zbiór operacji konfliktowych  $OK(l')$ .

**Krok 2.** Stosując regułę priorytetową  $R$ , znajdź w zbiorze operacji konfliktowych  $OK(l')$  operację  $j$  do uszeregowania.

**Krok 3.** Połóż  $\pi_{l'} := \pi_{l'} \cup j$ ,  $S(j) := r(j)$ ,  $U := U \cup \{j\}$  oraz  $r(i) := \max(r(i), S(j) + p_j)$  dla operacji  $i \in O \setminus U$ , dla których  $\mu(i) = l'$  lub  $i \in A_j$ . Jeżeli  $U = O$ , to STOP. W przeciwnym wypadku idź do kroku 1.

Interpretując jedną iterację algorytmu  $P(R)$  jako wykonanie kroków 1, 2, 3, a krok 0 jako inicjację, możemy stwierdzić, że algorytm  $P(R)$  po wykonaniu  $n$  iteracji kończy swoje działanie. Wyprodukowana przez niego permutacja znajduje się w  $\pi = (\pi_1, \pi_2, \dots, \pi_m)$ , a odpowiadające jej uszeregowanie w  $S(i)$ ,  $i \in O$ . W algorytmie nie wyznacza się bezpośrednio wartości  $t_l$ . Fakt przedłużenia zajętości maszyny  $l'$  po uszeregowaniu na niej operacji  $j$ , uwzględnia się w kroku 3 przez modyfikację głów  $r(i)$  operacji  $i$  jeszcze nieuszeregowanych, które będą na niej wykonywane;  $r(i) := \max(r(i), S(j) + p_j)$ .

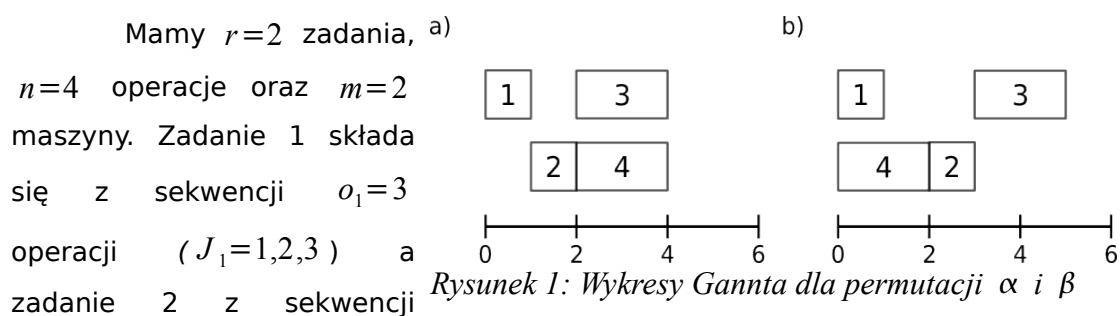
Algorytm  $P(R)$  działa poprawnie, tzn. generuje permutację  $\pi \in \Pi(T)$ , tylko wtedy, gdy graf relacji  $T$  jest acykliczny (lub równoważnie zbiór  $\Pi(T)$  nie jest pusty). W przeciwnym wypadku w pewnej iteracji mniejszej niż  $n$  zbiór  $OG(U)$  wyznaczany w kroku 1 będzie pusty. Pozwala to na użycie algorytmu  $P(R)$  jako detektora cyklu; wystarczy w kroku 1 po wyznaczeniu zbioru  $OG(U)$  dodać warunek: „jeżeli  $OG(U) = \emptyset$ , to STOP”.

Warto także zauważyć, że algorytm  $P(R)$  generuje tzw. uszeregowania aktywne (ang. active schedules). Uszeregowanie jest aktywne, jeśli żadna operacja nie może być wcześniej rozpoczęta bez jednoczesnego opóźnienia rozpoczęcia wykonywania przynajmniej jednej innej operacji. Precyzyjniej,

uszeregowanie  $S(i)$ ,  $i \in O$  jest aktywne, jeżeli nie istnieje uszeregowanie dopuszczalne  $\hat{S}(i)$ ,  $i \in O$  takie, że  $\hat{S}(i) \leq S(i)$ ,  $i \in O$  i przynajmniej dla jednej operacji nierówność ta jest ostra. Jest oczywiste, że w zbiorze wszystkich uszeregowień aktywnych musi znajdować się uszeregowanie optymalne. Co więcej, łatwo pokazać, że jeśli zastosuje się arbitralną regułę priorytetową  $R$ , to algorytm  $P(R)$  może wygenerować każde uszeregowanie aktywne. Dokładniej należy rozumieć to tak, że dla danego uszeregowania aktywnego można podać taką regułę  $R$ , że algorytm  $P(R)$  wygeneruje to uszeregowanie. Tym samym dla każdego problemu gniazdowego określonego przez dowolną acykliczną relację  $T$  istnieje przynajmniej potencjalna możliwość znalezienia przez algorytm  $P(R)$  uszeregowania optymalnego.

Uszeregowanie aktywne nie zawsze spełnia jednak pewną zasadę, bardzo często preferowaną przez praktyków. Zasada ta mówi: jeżeli w danej chwili  $i$  jest wolna maszyna, na której można wykonywać pewną operację (wcześniej wszystkie jej poprzedniki zostały już wykonane), to należy ją zacząć wykonywać. Przestrzeganie tej zasady powoduje, że nie ma przestojów na maszynach, jeżeli tylko można na nich wykonywać jakieś operacje. Uszeregowanie spełniające tę zasadę będziemy nazywać uszeregowaniem bez opóźnień (ang. non-delay scheduling). Wydawałoby się, że w celu znalezienia uszeregowania optymalnego należy generować tylko uszeregowania bez opóźnień. Niestety, nie jest to prawdą, co pokazuje poniższy przykład.

### Przykład 13.1



zawierającej tylko  $o_2=1$  operację  $(J_2=4)$ ;  $T_0=(1,2),(2,3)$ . Operacje 1, 3 należy wykonywać na maszynie 1, a pozostałe na maszynie 2. Czasy wykonywania poszczególnych operacji są następujące:  $p_1=p_2=1$ ,  $p_3=p_3=2$ .

Zbiór wszystkich permutacji dopuszczalnych zawiera dwie permutacje:  $\alpha=((1,3),(2,4))$  oraz  $\beta=((1,3),(4,2))$ . Odpowiadające tym permutacjom uszeregowania  $S^\alpha(j)$  oraz  $S^\beta(j)$  przedstawiono na rysunku 1. Obydwa uszeregowania są aktywne, ale tylko uszeregowanie  $S^\beta(j)$  jest uszeregowaniem

bez opóźnień. w uszeregowaniu  $S^\alpha(j)$  w przedziale czasowym  $[0,1)$  maszyna 2 nie wykonuje żadnej operacji, mimo że mogłaby wykonywać operację 4. Moment wykonania wszystkich operacji dla uszeregowania  $S^\beta(j)$  ( $C_{\max}(\beta, T_0)=5$ ) jest jednak większy niż dla uszeregowania  $S^\alpha(j)$  ( $C_{\max}(\alpha, T_0)=4$ ). Analizując wszystkie dopuszczalne uszeregowania bez opóźnień (w naszym przykładzie jest tylko jedno takie uszeregowanie), nie znajdziemy więc uszeregowania optymalnego.

Z powyższego przykładu wynikałoby, że nie należy raczej preferować uszeregowień bez opóźnień. Nie jest to jednak do końca prawda, ponieważ badania testowe pokazują, iż często uszeregowania bez opóźnień są lepsze niż uszeregowania aktywne, szczególnie w przypadkach, gdy na danym kroku iteracyjnym reguła priorytetowa jest wybierana losowo z pewnego ustalonego zbioru reguł. Warto tu zauważyć, że algorytm  $P(R)$  można łatwo zmodyfikować, aby dla każdej reguły priorytetowej  $R$ , generował takie właśnie uszeregowanie. W tym celu należy zmienić definicję wielkości  $\Delta$ , zbioru maszyn  $M'$  oraz zbiorów operacji konfliktowych  $OK(l)$ ,  $l \in M'$  na następujące:

$$\begin{aligned}
 A &= \min_{i \in OG(U)} r(t), \\
 M' &= \{l \in M : \forall i \in OG(U) \text{ } /i(t) = l, r(i) = A\}, \\
 OK(l) &= \{i \in OG(U) : f(i) = l, r(i) = A\}.
 \end{aligned}$$

(13.6) (13.?)

(13.8)

Omówimy teraz krótko różne postacie reguł priorytetowych wykorzystywanych w kroku 2 algorytmu  $P(R)$ , które wyznaczają, w zbiorze operacji konfliktowych  $OK(l)$  operację  $j$  do uszeregowania. Najczęściej stosowane w literaturze zostały przedstawione w tabeli 13.1. Jeśli dana reguła nie określa w sposób jednoznaczny operacji  $j$  do uszeregowania (np. gdy przy

Reguła  $R$

STT

LTT

SRPT

LRPT

LTRPT SPT LPT

Random

Tabda 13.1. Reguły priorytetowe  $R$

Opis reguły



operacja z najkrótszym czasem wykonywania       $\sim \sim$       operacja z  
najdłuższym czasem wykonywania

operacja zadania o najkrótszym pozostałym czasie wykonywania operacja  
zadania o najdłuższym pozostałym czasie wykonywania operacja z największą  
sumą czasu wykonywania i wartości ogona operacja zadania o najkrótszym czasie  
wykonywania operacja zadania o najdłuższym czasie wykonywania operacja  
wybrana losowo

regule  $STT$  w zbiorze operacji konfliktowych znajduje się kilka operacji o  
tym samym najkrótszym czasie wykonywania), to wybór operacji  $j$  -wśród  
wszystkich operacji spełniających tą regułę - następuje arbitralnie lub stosuje się  
dodatkowo jeszcze inną regułę. Jest rzeczą zadziwiającą, że preferuje się w  
literaturze pary reguł dokładnie przeciwnych, np. reguła  $STT$  i  $LTT$ . Wynika to  
jednak z faktu, że istnieją takie instancje rozważanego problemu, w których tylko  
jedna reguła z takiej pary (różna dla różnych instancji) powoduje, że algorytm  
 $P\{R\}$  generuje uszeregowanie o akceptującej odległości od uszeregowania  
optymalnego. Świadczy to niewątpliwie o „słabości” algorytmów priorytetowych.  
Wydawałoby się, że naturalnym sposobem polepszenia jakości otrzymywanych  
uszeregowień jest kilkakrotne uruchomienie algorytmu  $P(R)$  dla różnych reguł.  
Okazuje się jednak, że znacznie lepsze efekty daje tylko jednokrotne uruchomienie  
 $P\{R\}$  z losowym wyborem reguły  $R$  na każdym kroku iteracyjnym.

Na koniec zauważmy, że jedna iteracja algorytmu  $P(R)$  wymaga  $O(n)$   
czasu (przy założeniu, że reguła  $R$  wybierze operacje do uszeregowania w czasie

$O(n)$ ). Stąd złożoność obliczeniowa tego algorytmu wynosi  $O(n^2)$ .

Kończąc tę sekcję zilustrujemy jeszcze działanie algorytmu  $P\{R\}$  dla reguły  $R = STT$  na danych z przykładu 11.1.

Przykład 13.2 Algorytm  $P(STT)$  zastosujemy dla relacji  $T = T_0$  (korzeń drzewa rozwiązań). W przypadku niejednoznaczności reguły  $STT$  dziemy wykorzystywać regułę SRPT, a jeżeli to nie wystarczy, wybieramy operację o najmniejszym numerze. Ponadto, w przypadku niejednoznaczności w wyborze maszyny  $i'$  w kroku 1, wybieramy maszynę o najmniejszym numerze.

Ilustrację rozpoczynamy od iteracji 1. Po wykonaniu kroku 0,  $7r = H(0,0)$ ,  $r(1) = r(2) = \dots = r(12) = 0$  oraz  $U = \emptyset$ . W kroku 1 zbiór operacji gotowych  $OG(U) = \{1,8,11\}$ , stąd

$$\min\{r(i) + p_i, r(8) + p_8, r(11) + p_{11}\} = \min\{0 + 2, 0 + 2, 0 + 2\} = 2 = A,$$

$i' = 1$  oraz zbiór operacji konfliktowych na maszynie 1 ma postać

$$OK\{U\} = \{1,8,11\}.$$

Stosując regułę STT w kroku 2, otrzymamy trzy operacje: 1, 8, 11; wszystkie mają ten sam czas wykonania. Dopiero zastosowanie reguły SRPT pozwala wybrać operację  $j = 11$  do uszeregowania; operacja 1 jest operacją zadania 1 z pozostałym czasem wykonywania  $P_2 + P_3 + \dots + p_7 = 1 + 2 + 2 + 1 + 1 + 1 = 5 = 8$ , operacja 8 jest operacją zadania 8 z pozostałym czasem wykonywania  $P_9 + p_{10} = 2 + 1 = 3$ , zaś operacja 11 operacją zadania 3 z pozostałym czasem wykonywania  $P_{12} = 2$ . Następnie wykonując krok 3, otrzymamy  $ir = ((11), 0)$ ,  $5(11) = 0$ ,  $U = \{11\}$ ,  $r(i) = \max(0, 5(11) + p_i) = \max(0, 0 + 2) = 2$  dla  $i = 1, 3, 5, 7, 8, 10$  (operacje 1, 3, 5, 7, 8, 10 mają być wykonywane na maszynie  $i' = 1$ ) oraz  $r(12) = 2$  (operacja 12 jest bezpośrednim następnikiem operacji  $j = 11$ ;  $A \setminus \{12\}$ ); patrz rysunek 13.2. Ponieważ  $U \neq \emptyset$  przechodzimy zatem do kroku 1 iteracji 2.

W kroku 1 zbiór operacji gotowych  $OG\{U\} = \{1,8,12\}$ , stąd

$$\min\{r(i) + p_i, r(8) + p_8, r(12) + p_{12}\} = \min\{2 + 2, 2 + 2, 2 + 2\} = 4 = A,$$

$i' = 1$  oraz zbiór operacji konfliktowych na maszynie 1 ma teraz postać

$$OA'(1) = \{1,8\}.$$

Stosując kolejno reguły STT i SRPT dostaniemy  $j = 8$ . Następnie wykonując krok 3, dostaniemy  $w = ((11,8), 0)$ ,  $-5(8) = 2$ ,  $U = \{8,11\}$ ,  $r(1) = r(3) = r(5) = r(7) = r(10) = 4$ ,  $r(9) = 4$  i przechodzimy do kroku 1 iteracji 3.

W kroku 1 zbiór operacji gotowych  $OG(U) = \{1,9,12\}$ ,

$$\min\{r(i) + p_i, r(9) + p_9, r(12) + p_{12}\} = \min\{4 + 2, 4 + 2, 2 + 2\} = 4 = A.$$



$l = 2$  oraz  $O_{ii}(2) = \{12\}$ . Stąd wynika, że będziemy szeregować operację  $j = 12$ , co spowoduje, że  $\pi = ((11,8), (12))$ ,  $5(12) = 2$ ,  $U = \{8,11,12\}$ , i

Rys. 13.3. Wykres Gantta dla permutacji optymalnej

$r(2) = r(4) = r(6) = r(9) = 4$ . Następnie przechodzimy do kroku 1 kolejnej iteracji. Po wykonaniu 12 iteracji otrzymujemy permutację

$$J\$_r = ((11,8,1,10,3,5,7), (12,9,2,4,6))$$

i odpowiadające jej uszeregowanie, przedstawione na rysunku 13.2, z wartością funkcji celu  $C_{\max}(\pi^*, 2o) = 14$ . Jak łatwo sprawdzić, uszeregowanie to jest aktywne, ale niestety nie jest optymalne. Uszeregowanie dla jednej z permutacji optymalnych

$$\pi^- = ((1,11,3,8,5,10,7), (2,12,4,9,6))$$

podano na rys. 13.3. Uszeregowanie to jest też aktywne, ale ma wartość funkcji celu  $C_{\max}(7rj|To) = 12$ ,

Rys. 13.2. Wykres Gantta dla permutacji  $\pi$  otrzymanej przez algorytm  $P(STT)$

