AUTOMATYKA • 2008 • Tom 12 • Zeszyt 3

Wiesław Popielarski*

Szeregowanie zadań w zarządzaniu procesami biznesowymi

1. Wprowadzenie

Z problemem szeregowania zadań spotykamy się niemalże w każdej dziedzinie ludzkiej działalności, począwszy od układania indywidualnego planu dnia, poprzez grafiki zadań wykonywanych w pracy, planowanie linii produkcyjnych czy systemów logistycznych, kontrolę przepływu klientów w banku, petentów w urzędzie, kontrolę przylotów i odlotów, kończąc na wszechobecnych wieloprocesowych systemach operacyjnych. Jednakże świadomość matematycznych podstaw szeregowania, jego wieloaspektowości, czy też chociażby umiejętność posługiwania się odpowiednimi narzędziami, jest stosunkowo niska wśród osób, które stykają się z tą problematyką w swoim codziennym życiu zawodowym. Wydaje się, że wciąż zbyt wiele z nich, szczególnie menadżerów wyższych szczebli, w procesie planowania wierzy bardziej swojej intuicji niż obiektywnym rezultatom badań, co wpływa na późniejsze wyniki uzyskiwane przez zarządzane przez nich zespoły. Artykuł jest pomyślany jako swego rodzaju wstęp do problematyki szeregowania zadań. W dalszej części przedstawione zostaną wybrane modele matematyczne dla systemów jedno- i wieloprocesorowych tak, by ogólnie przybliżyć problemy spotykane podczas analizy szczególnego zagadnienia. Następnie przedstawiona zostanie krótka dyskusja o różnicach między modelami a rzeczywistymi problemami. Na zakończenie zostanie zadane pytanie o możliwe dalsze obszary badań.

2. Wybrane modele szeregowania zadań

Podstawowym podziałem stosowanym w klasyfikacji modeli jest ich rozróżnienie ze względu na liczbę przetwarzających maszyn oraz ich konfigurację. Stąd też rozróżnić można modele dla jednej lub wielu maszyn (procesorów) z tym, że w drugim przypadku maszyny mogą być połączone równolegle lub szeregowo. Szeregowe połączenie wielu maszyn jest charakterystyczne dla problemów określanych pojęciem shopu (sklepu). Jest to taka

* Doktorant, Katedra Automatyki, Akademia Górniczo-Hutnicza w Krakowie

konfiguracja, w której każde zadanie musi być przetworzone przez każdy z procesorów. Innym, nie mniej ważnym podziałem modeli jest podział na te, w których zadania są określone przed rozpoczęciem przetwarzania (szczególnie ich liczba, charakterystyki tj. zależność od zadań poprzedzających, czas przetwarzania itp.), i takie, w których liczba i charakterystyka zadań mogą zmieniać się w czasie. Pierwszą rodzinę modeli określa się mianem deterministycznych, natomiast drugą – stochastycznych.

Istnieje wiele modeli, których przedstawienie, nawet w zarysie, przekracza ramy tej publikacji. Dlatego wybrano jedynie kilka z nich, aby na ich przykładzie pokazać pewne problemy związane z analizą, dowodzeniem poprawności znalezionego optymalnego rozwiązania, ewentualnie dowodzeniem, że dany problem należy do klasy *NP*. Dowód poprawności znalezionego algorymu jest przeprowadzany w celu pokazania, że otrzymywane wyniki są optymalne z punktu widzenia zadanego celu (np. minimalny czas przetwarzania). Nie istnieje ogólna metoda dowodzenia. Z reguły dowodzi się przez sprowadzenie do sprzeczności lub znacznie rzadziej przez indukcję, lecz większość dowodów wymaga przede wszystkim pomysłowości. Wszystkie rozważane modele są modelami deterministycznymi. Dowody, które przedstawiono, są zarysowane w [3] oraz [1], natomiast pojęcia z teorii złożoności zostały zaczerpnięte z [2].

2.1. Modele pojedynczej maszyny

Modele pojedynczej maszyny są ważne z dwóch zasadniczch powodów. Pierwszy z nich to ich użyteczność. Mimo że w rzeczywistych problemach z reguły rozważa się środowiska wieloprocesorowe, to można znaleźć zagadnienia, dla których ten sposób modelowania jest najwłaściwszy. Przykładem może być problem pojedynczego wąskiego gardła w systemie wieloprocesorowym. Generalnie wiele złożonych problemów dekomponuje się do modeli pojedynczej maszyny. Drugi powód jest taki, że modele te charakteryzują się pewnymi unikalnymi cechami, niewystępującymi w innych konfiguracjach, i jako takie mogą dostarczać podstaw do heurystyk stosowanych w bardziej skomplikowanych środowiskach.

Ważony całkowity czas przetwarzania $1\|\sum w_jC_j$

Ważony całkowity czas przetwarzania (*the total weighted completion time*) jest stosunkowo prostym modelem, który może być zastosowany na przykład w zarządzaniu zadaniami na poziomie pojedynczego pracownika. Waga w_j zadania j może być uważana za współczynnik istotności. Ten problem przedstawia jedną z lepiej znanych zasad w teorii szeregowania mianowicie: ważony najkrótszy czas przetwarzania (*weighted shortest processing time* – WSPT). Zgodnie z tą zasadą zadania są szeregowane w porządku malejącym wyrażenia $\frac{w_j}{p_j}$, gdzie p_j oznacza czas przetwarzania zadania j. Reguła WSPT jest optymalna dla $1\|\sum w_jC_j$.

Dowód: Przez sprowadzenie do sprzeczności.

Załóżmy, że pewne uszeregowanie S jest optymalne, ale zostało uzyskane w inny sposób jak WSPT. Stąd w uszeregowaniu S muszą być co najmniej dwa sąsiadujące ze sobą zadania j i k takie, że $\frac{w_j}{p_j} < \frac{w_k}{p_k}$. Załóżmy, że przetwarzanie zadania j rozpoczęło się w chwili t. Wówczas ważony czas zakończenia zadań j i k wynosi $(t+p_j)w_j + (t+p_j+p_k)w_k$. Utwórzmy teraz uszeregowanie S' z uszeregowania S, zamieniając kolejność zadań j i k. Wówczas ważony czas zakończenia zadań k i j, przy założeniu, że przetwarzanie zadania k rozpoczęło się w chwili t, wyniesie $(t+p_k)w_k + (t+p_k+p_j)w_j$. Podzielmy oba wyrażenia przez p_k i odejmijmy je od siebie, wówczas otrzymamy:

$$\frac{tw_j}{p_k} + \frac{p_j w_j}{p_k} + \frac{tw_k}{p_k} + \frac{p_j w_k}{p_k} + w_k - \frac{tw_k}{p_k} - w_k - \frac{tw_j}{p_k} - w_j - \frac{p_j w_j}{p_k} = \frac{p_j w_k}{p_k} - w_j \qquad (1)$$

obliczając dalej:

$$\frac{p_j w_k}{p_k} - w_j = p_j \left(\frac{w_k}{p_k} - \frac{w_j}{p_j} \right),$$

i biorąc pod uwagę, że

$$\frac{w_j}{p_j} < \frac{w_k}{p_k},$$

wyrażenie

$$p_j \left(\frac{w_k}{p_k} - \frac{w_j}{p_j} \right) > 0.$$

Stąd wynika, że całkowity czas przetwarzania zadań w uszeregowaniu S' jest mniejszy od całkowitego czasu w uszeregowaniu S, czyli uszeregowanie S nie jest uszeregowaniem optymalnym, co jest w sprzeczności z założeniem.

Maksymalne opóźnienie

Przykład ten został wybrany do analizy ze względu na dwa aspekty. Po pierwsze, minimalny czas przetwarzania zadań przez pracownika nie musi być dla niego wiążącym celem. Natomiast może być on zainteresowany najmniejszą stratą (największym zyskiem), jaką może uzyskać dzięki odpowiedniemu uszeregowaniu oczekujących zadań. Po drugie, maksymalne opóźnienie jest interesującym przypadkiem z punktu widzenia analizy modelu matematycznego, którego generalizacja jest problemem *NP*-trudnym, natomiast dla jego szczególnych przypadków znaleziono rozsądne algorytmy.

Maksymalne opóźnienie $1|prec|h_{\max}$

Niech $h_{\max} = \max(h_1(C_1), h_2(C_2), ..., h_n(C_n))$, gdzie h_j, $j = 1, ..., n$ są niemalejącymi funkcjami kosztu. Problem ten ma wielomianowe rozwiązanie za pomocą programowania dynamicznego. Zakończenie przetwarzania ostatniego zadania wypada w $C_{\max} = \sum p_j$, który nie zależy od uszeregowania. Niech J oznacza zbiór zadań już uszeregowanych, które będą przetwarzane w przedziale czasu $\left[C_{\max} - p_j, C_{\max}\right]$. Dopełnienie zbioru J, zbiór J^c, oznacza zbiór zadań, które wciąż nie zostały uszeregowane. J' jest podzbiorem tych zadań z J^c, których bezpośredni następcy (zadania wykonywane bezpośrednio po nich) są już w J. Wówczas poniższy algorytm zwraca optymalne uszeregowanie.

MIN-MAX-COST

$J = \{\}$;
$J^c = \{1, 2, ..., n\}$;
J' = zbiór wszystkich zadań, które nie mają następców
while ($!J^c$.isEmpty()) {

wybierz j^* takie, że $h_{j^*}(\sum_{j \in J^c} p_j) = \min_{j \in J'} h_j(\sum_{k \in J^c} p_k)$ (2)

J.add(j^*);
J^c.remove(j^*);
wyznacz nowy J';
}
return J;

Dowód jest pominięty.

Przykład

Rozważmy poniższe trzy zadania (tab. 1), z których każde dwa nie zależą od siebie.

Tabela 1

Zadania	1	2	3
p_j	4	3	5
$h_j(C_j)$	$1+C_i$	$1{,}2C_i$	10

$C_{\max} = 12$ oraz $h_2(12) > h_1(12) > h_3(12)$ (14,4 > 13 > 10). Stąd zadanie 3 powinno być uruchomione jako ostatnie w chwili 12 – 5 = 7. Następnie $h_2(7) > h_1(7)$ (8,4 > 8), czyli zadanie 1 powinno być uruchomione jako przedostatnie w chwili 7 – 4 = 3.

Wyznaczone optymalne uszeregowanie ma następującą postać: 2, 1, 3.

Specjalnym przypadkiem $1|prec|h_{\max}$ jest problem $1\|L_{\max}$, w którym h_j jest zdefiniowane jako $C_j - d_j$. Bez dowodu stwierdzimy, że uszeregowanie optymalne dla tego przypadku znajduje się zgodnie z regułą najwcześniejszego czasu zakończenia d_j (*Earliest Due Date First*). Jednak bardziej interesująca jest generalizacja $1\|L_{\max}$, czyli problem $1|r_j|L_{\max}$. Okazuje się, że nawet w przypadku modeli pojedynczej maszyny istnieją problemy o złożoności *NP*.

Twierdzenie: Problem $1|r_j|L_{\max}$ jest *NP*-trudny.

Dowód: Dowód opiera się na redukcji problemu 3-PARTITION do $1|r_j|L_{\max}$.

Problem 3-PARTITION:

Niech będą dane dodatnie liczby całkowite a_1, a_2, ..., a_{3t}, b takie, że: $b/4 < a_j < b/2$ oraz $\sum_{j=1}^{3t} a_j = tb$. Należy sprawdzić, czy istnieje t parami rozłącznych 3-elementowych zbiorów S_i takich, że $\sum_{j\in S_i} a_j = b$ dla $i = 1, 2, \ldots, t$.

Rozważmy $n = 4t - 1$ zadań zdefiniowanych następująco:

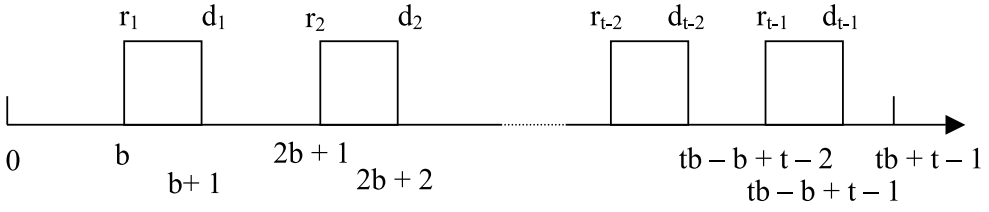
$$r_j = jb + (j - 1), \quad p_j = 1, \quad d_j = jb + j \quad \text{dla} \quad j = 1, ..., t - 1,$$

oraz dla pozostałych $4t - 1 - (t - 1) = 3t$ zadań:

$$r_j = 0, \quad p_j = a_{j-t+1}, \quad d_j = tb + (t - 1) \quad \text{dla} \quad j = t, ..., 4t - 1.$$

Z definicji tej wynika, że zadania $j_1, j_2, ..., j_{t-1}$ mogą być uruchamiane nie wcześniej jak w chwilach $jb + (j - 1)$ i nie mogą zakończyć się później jak w chwilach $jb + j$ (patrz rys. 1).

Pozostałe $3t$ zadań mogą być uruchamiane dowolnie wcześnie ($r_j = 0$) i powinny zakończyć się nie później jak w chwili $tb + t - 1$. Całkowita długość czasu, w którym te zadania powinny być uszeregowane wynosi $tb + t - 1 - (t - 1)*1 = tb$.



Rys. 1. Uszeregowanie pierwszych t–1 zadań na jednym procesorze

Zauważmy, że każdy z wolnych interwałów ma tą samą długość wynoszącą b (np. $tb + t - 1 - tb + b - t + 1 = b$). Przyjmując, że czas przetwarzania p_j każdego z $3t$ zadań wynoszący a_j wyraża się zależnością $b/4 < a_j < b/2$, redukujemy badany problem do problemu 3-PARTITION, który należy do klasy *NP*-zupełnej, co kończy dowód.

2.2. Modele maszyn równoległych

Modele maszyn równoległych są ważne ze strony zarówno teoretycznej, jak i praktycznej. Z teoretycznej, ponieważ są one generalizacją modeli pojedynczej maszyny. Z praktycznej, bo w rzeczywistych problemach z reguły spotyka się systemy, których zasoby są zrównoleglone. Szczególne miejsce w problematyce maszyn równoległych zajmuje problem równomiernego obciążenia procesorów przez zadania tak, aby całkowity czas przetwarzania był minimalny. Okazuje się jednak, że już $P2 \| C_{\max}$ należy do klasy *NP*-trudnej.

Dowód: Spróbujmy zredukować problem podziału zbioru (PARTITION) do $P2 \| C_{\max}$.

Problem PARTITION określony jest następująco.

Niech będą dane dodatnie liczby całkowite $a_1, a_2, ..., a_t, b$ takie, że: $\frac{1}{2}\sum_{j=1}^{t} a_j = b.$ Należy znaleźć dwa rozłączne podzbiory S_1 i S_2 takie, że $\sum_{j \in S_i} a_j = b$ dla $i = 1, 2.$

Niech $P1$ i $P2$ odpowiadają podzbiorom S_1 i S_2, natomiast liczby $a_1, a_2, ..., a_t$ zadaniom przydzielanym tym procesorom. Zauważmy również, że $b \le C_{\max} \le 2b.$ Wówczas problem $P2 \| C_{\max}$ redukuje się do problemu decyzyjnego odpowiadającego na pytanie, czy istnieje takie uszeregowanie zadań na procesorach $P1$ i $P2$, że $C_{\max} = b.$

Wniosek: $Pm \| C_{\max}$ należy do klasy *NP*-trudnej. Z tego względu w rozwiązywaniu tej klasy problemów używane są przede wszystkim heurystyki lub algorytmy aproksymacyjne. Jedną z takich metod jest zasada najdłuższego czasu przetwarzania (*Longest Processing Time First* – LPT). Zasada ta przypisuje w chwili $t = 0$ m najdłuższych zadań do m procesorów. W przypadku gdy którykolwiek z procesorów jest wolny, najdłuższe zadanie spośród jeszcze nieprzydzielonych jest przypisane temu procesorowi. Okazuje się, że:

$$\frac{C_{\max}(LPT)}{C_{\max}(OPT)} \le \frac{4}{3} - \frac{1}{3m} \tag{3}$$

Dowód będzie pominięty.

Inną heurystyką, optymalną dla pewnych szczególnych modeli wieloprocesorowych, jest zasada ścieżki krytycznej (*Critical Path* – CP). Rozważmy abstrakcyjny model $P\infty \mid prec \mid C_{\max}$. Optymalne uszeregowanie określić można stosując poniżej opisany algorytm:

1. Przydziel zadania zaczynające się w chwili $t = 0$;
2. Za każdym razem, gdy zadanie zostanie przetworzone, uruchom wszystkie zadania, których z kolei wszystkie zadania zależne poprzedzające (*predecessors*) zostały zakończone.

Na podstawie tego algorytmu można określić tzw. krytyczne zadania, tzn. takie, które nie mogą być opóźnione. Zadania te tworzą ścieżkę krytyczną. Zasada *Critical Path* daje największy priorytet temu zadaniu (zadaniom), którego łańcuch zadań w grafie poprzedania jest najdłuższy.

2.3. Flow Shop

W niektórych zagadnieniach na jednym zadaniu musi być wykonana pewna liczba operacji. Te same operacje powinny być wykonane na każdym zadaniu w tej samej kolejności. Maszyny przetwarzające przychodzące zadania są ustawione szeregowo (sekwencyjnie). Mówimy wówczas o środowisku Flow Shop. Przykładami mogą być fabryka papierowych worków, produkująca różne ich rodzaje, montownie samochodów, przepływ dokumentów w firmie. Kolejne etapy przetwarzania/produkcji są takie same bez względu na rodzaj zadania/wytwarzanych dóbr. Różnica jest tylko w samym czasie przetwarzania, który zależy od rodzaju zadania. Generalnie uszeregowania typu Shop są problemami, dla których trudno jest znaleźć wielomianowy algorytm. W celu zilustrowania tych trudności poniżej przedstawiono dwa przykładowe zagadnienia.

Problem $F2 \| C_{\max}$

Optymalna sekwencja może być znaleziona w następujący sposób.

Podzielmy zadania na dwa zbiory:

zbiór I – zawierający wszystkie zadania, dla których $p_{1j} < p_{2j}$;
zbiór II – wszystkich zadań z $p_{1j} > p_{2j}$.

Zadania, dla których $p_{1j} = p_{2j}$ mogą być przypisane do któregokolwiek ze zbiorów. Zadania ze zbioru I są przetwarzane pierwsze w porządku niemalejącym, następnie przetwarzane są zadania ze zbioru *II* w porządku nierosnącym. Szeregowanie takie określa się jako SPT(1)-LPT(2) (również zasadą Johnsona). Okazuje się, że każde uszeregowanie SPT(1)-LPT(2) jest optymalne dla $F2 \| C_{\max}$. Niestety, zasada Johnsona może być stosowana jedynie w środowisku z dwoma procesorami. Jak się okazuje, już nawet problem $F3 \| C_{\max}$ należy do klasy problemów *NP*-trudnych.

Dowód: Przez redukcję z problemu 3-PARTITION.

Przyjmijmy, że liczba zadań n wynosi $4t + 1$ oraz:

$$p_{01} = 0, \quad p_{20} = b, \quad p_{30} = 2b,$$

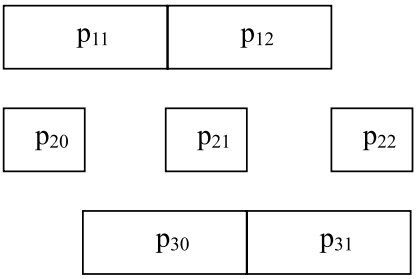
$$p_{1j} = 2b, \quad p_{2j} = b, \quad p_{3j} = 2b \quad \text{dla } j = 1, ..., t-1,$$

$$p_{1t} = 2b, \quad p_{2t} = b, \quad p_{3t} = 0,$$

$$p_{1j} = 0, \quad p_{2j} = a_{j-t}, \quad p_{3j} = 0 \quad \text{dla } j = t+1, ..., 4t+1.$$

Spróbujmy uszeregować zadania w taki sposób, aby $C_{\max} = (2t+1)b$. Można to osiągnąć, jeśli pierwsze $t + 1$ zadań uszeregowanych jest zgodnie z sekwencją 0, 1, ..., t. Tych $t + 1$ zadań tworzy rodzaj szablonu dla pozostałych $4t + 1 - t - 1 = 3t$ zadań, pozostawiając na drugim procesorze t przerw o szerokości b. Biorąc pod uwagę założenie $b/4 < a_j < b/2$ $C_{\max}$ może być osiągnięte, jeżeli problem 3-PARTITION ma rozwiązanie.

Na rysunku 2 przedstawiono uszeregowanie dla $t = 2$.



Rys. 2. Uszeregowanie pierwszych trzech zadań na trzech procesorach. W przerwach pomiędzy zadaniami p_{20} a p_{21} oraz p_{21} a p_{22} należy umieścić pozostałe $3t = 6$ zadań

3. Modele matematyczne a rzeczywiste problemy

Jak się można było przekonać, mimo że modele matematyczne przedstawiają problem szeregowania zadań w sposób uproszczony, jednak pozwalają często na określenie, czy dla danego szczególnego, rzeczywistego problemu warto poszukiwać wielomianowych rozwiązań. Okazuje się, że znaczna większość problemów jest w istocie problemami *NP*-trudnymi. Oznacza to, że jest duże prawdopodobieństwo tego, że nie warto inwestować czasu i środków na znalezienie rozsądnego algorytmu wyznaczającego rozwiązanie optymalne. Analiza modeli może również dostarczyć zasad lub algorytmów dla szczególnych przypadków. Rolą zespołu analizującego jest wówczas dekompozycja rzeczywistego zagadnienia na elementarne problemy, dla których znaleziono już rozsądne rozwiązania, lub heurystyki, których błąd przynajmniej daje się określić. Podczas takiej analizy (niekoniecznie dotyczącej samej dekompozycji) należy zwrócić uwagę szczególnie na te aspekty rzeczywistego problemu, które znacząco różni go od modeli matematycznych. Choć określenie tego zestawu cech zależy od rozważanego problemu i samo w sobie jest zagadnieniem nietrywialnym, jednak jest ich pewien wspólny, przedstawiony poniżej, zbiór.

- Modele z reguły rozpatrują n zadań, przy czym n jest znane *a priori*. Rzeczywiste problemy są często dynamiczne.
- Modele z reguły biorą pod uwagę jeden parametr, względem którego poszukiwane jest rozwiązanie optymalne. W rzeczywistym zagadnieniu można poszukiwać rozwiązania optymalnego dla pewnego zbioru takich parametrów.
- W modelach parametry są stałe w czasie (np. priorytet zadań, czas przetwarzania), w rzeczywistych problemach parametry te mogą się zmieniać w zależności od aktualnego stanu przetwarzanych zadań.
- W modelach parametr, względem którego poszukiwane jest rozwiązanie optymalne, ma dokładną, ustaloną wartość. W rzeczywistych problemach ten parametr nie musi być ściśle określony. Może on należeć do pewnego dopuszczalnego zbioru wartości.

– W modelach stochastycznych parametry określane są przez dobrze zdefiniowane rozkłady gęstości prawdopodobieństwa, natomiast w rzeczywistości rozkłady te mogą nie być znane.

Następnym krokiem w określeniu uszeregowania zadań dla rzeczywistego zagadnienia jest wybór metody jego rozwiązywania. Podstawowymi możliwościami są tu:

– dekompozycja złożonego zagadnienia do zagadnień prostych lub takich, dla których zbudowano już modele i znaleziono akceptowalne rozwiązanie;
– redukcja złożonego zagadnienia do prostszego, rozwiązywalnego; w ten sposób wprawdzie z jednej strony traci się na dokładności rozwiązania, ale z drugiej pozwala na znalezienie akceptowalnego rozwiązania stosunkowo małym kosztem;
– poszukiwanie heurystyk, algorytmów ewolucyjnych i innych struktur (sieci neuronowe, uczenie maszynowe), które najlepiej określałyby zależności pomiędzy badanymi parametrami i odnajdowały akceptowalne rozwiązanie.

Oprócz wyżej wspomnianych, można także zastanowić się nad takim określeniem ograniczeń dla zadań i ich parametrów, aby można byłoby rozwiązać problem metodą podziału i ograniczeń. Nawet jeśli czas użyty na obliczenia jest nieakceptowalnie długi, to niewątpliwym zyskiem takiego podejścia jest budowa narzędzia, które można by użyć do szacowania dokładności uzyskanych rozwiązań dla pewnych zbiorów danych testowych.

4. Podsumowanie

Przedstawione modele, choć jedne z prostszych w analizie, nie są modelami trywialnymi. Rzeczywiste zagadnienia wydają się być jeszcze trudniejsze. Dlatego też poza klasycznym podejściem polegającym na wyodrębnianiu pewnych dobrze określonych modeli, w analizie zagadnień szeregowania stosuje się również, z różnym skutkiem, bardziej wyrafinowane metody jak chociażby sieci neuronowe czy algorytmy ewolucyjne. Wydaje się, że te ostatnie coraz częściej stają się obiektem badań informatyków ze względu na obiecujące wyniki. I chociaż otwierają się nowe obszary zastosowań, jednak należy mieć świadomość z trudności, jakie można napotkać przy poszukiwaniu takich algorytmów. Przede wszystkim istnieje problem dobrego określenia tzw. sąsiedztwa danego uszeregowania, ponieważ wpływa ono decydująco na jakość otrzymanego rozwiązania. Jeżeli proces generowania sąsiedztwa jest nieprawidłowy, może się okazać, że algorytm będzie mieć problemy ze znalezieniem lepszego rozwiązania niż początkowe (uzyskane w inny sposób np. losowo lub przez inną heurystykę). Drugim problemem nie mniej istotnym w wielu zagadnieniach, jest określenie błędu między uzyskanym rozwiązaniem a rozwiązaniem optymalnym. Jak do tej pory, nie pojawiła się spójna analiza tej problematyki, a dokładność algorytmów ewolucyjnych zależy przede wszystkim od dokładności pierwszego rozwiązania (uzyskanego inną heurystyką lub algorytmem aproksymacyjnym).

Literatura

[1] Blazewicz J., Ecker K., Schmidt G., Weglarz J., *Scheduling Computer and Manufacturing Processes*. Berlin, Springer-Verlag 1996.
[2] Papadimitriou Ch., *Złożoność obliczeniowa*. Warszawa, WNT 2007.
[3] Pinedo M., *Scheduling: Theory, Algorithms, and Systems.* 2nd Edition. New York, Pearson Higher Education 2005.