METODA WSTAWIEŃ
W KLASYCZNYCH PROBLEMACH SZEREGOWANIA.
Cz. I. PROBLEM PRZEPŁYWOWY.

Eugeniusz Nowicki, Mariusz Makuchowski

Streszczenie: Prezentowana praca jest pierwszą częścią opracowania dotyczącego metody wstawień w zastosowaniu do budowy algorytmów konstrukcyjnych dla klasycznych problemów szeregowania operacji produkcyjnych. Przedstawia się w niej szereg tego typu algorytmów będących modyfikacją aktualnie najlepszego znanego w literaturze algorytmu konstrukcyjnego NEH, zbudowanego na bazie metody wstawień, dla jednego z najbardziej klasycznych problemów szeregowania jakim jest problem przepływowy. Jako podstawowe kryterium przyjmuje się moment wykonania wszystkich zadań. Dodatkowo prezentuje się szczegółowe wyniki numerycznych badań porównawczych pomiędzy proponowanymi algorytmami a algorytmem NEH.

Słowa kluczowe: metoda wstawień, problem przepływowy, algorytmy konstrukcyjne.

1. Wprowadzenie

W ostatnich kilku dekadach obserwuje się burzliwy rozwój teorii szeregowania, a w szczególności jej istotnego fragmentu związanego z szeregowaniem operacji produkcyjnych na maszynach. Jest to spowodowane przede wszystkim bardzo dużą aplikacyjnością otrzymywanych rezultatów, które pozwoliły na zbudowanie efektywnych algorytmów stanowiących jądra komputerowo zintegrowanych systemów wytwarzania. Specyfika problemów oraz wymagania praktyczne powodują jednak, że zainteresowanie badaczy skupia się przede wszystkim na konstrukcji efektywnych algorytmów przybliżonych zamiast algorytmów dokładnych (znajdujących uszeregowanie optymalne). Wynika to przede wszystkim z faktu, że czas działania algorytmu jest bardzo istotnym parametrem, zaś zdecydowana większość problemów należy do klasy tzw. problemów NP-trudnych, co praktycznie wyklucza istnienie szybkich algorytmów dokładnych.

Ogólnie algorytmy przybliżone można podzielić na dwie klasy: algorytmy konstrukcyjne oraz algorytmy popraw. Idea algorytmów konstrukcyjnych polega na tym, że startując z uszeregowania pustego, w każdym kolejnym kroku iteracyjnym szeregują na odpowiedniej maszynie jedną z dotychczas nie uszeregowanych operacji aż do uszeregowania ich wszystkich, co jest równoważne z otrzymaniem uszeregowania pełnego i zakończeniem działania. Sposób wyboru szeregowanej w danym kroku operacji, maszyny na której ma być ona szeregowana oraz pozycji, na którą jest ona wstawiana na wybranej maszynie określa dany algorytm konstrukcyjny. Z kolei algorytmy popraw startują z pełnego uszeregowania (otrzymanego z reguły przez algorytmy konstrukcyjne) i następnie w kolejnych krokach iteracyjnych starają się poprawić to uszeregowanie w sensie wartości rozpatrywanego kryterium.

Już z tego bardzo ogólnego opisu wynika, że algorytmy konstrukcyjne działają w czasie zdecydowanie krótszym niż algorytmy popraw. Niestety jakość produkowanych przez nie

uszeregowań jest znacznie gorsza. Tym niemniej potrzeba zastosowania „dobrego” (w sensie wartości funkcji celu) uszeregowania początkowego dla algorytmów popraw powoduje, że badania na tymi algorytmami są w dalszym ciągu ważne i w pełni uzasadnione. Co więcej są takie sytuacje produkcyjne (np. awarie), w których bardzo szybkie algorytmy są konieczne, ponieważ czas działania algorytmów popraw dla problemów o praktycznych rozmiarach waha się z reguły od kilku do kilkunastu minut podczas gdy algorytmy konstrukcyjne wymagają czasu tylko rzędu kilku sekund.

Różnorodność sytuacji produkcyjnych powoduje, że zdecydowana większość istniejących algorytmów konstrukcyjnych jest wysoce specjalizowana dla konkretnych problemów szeregowania modelujących określone sytuacje produkcyjne i z reguły brak w nich elementów wspólnych. Istotnym wyjątkiem są algorytmy bazujące na metodzie wstawień. Ogólnie mówiąc algorytmy te składają się z dwóch faz: fazy wstępnej i fazy zasadniczej. W fazie wstępnej tworzy się listę operacji według nierosnących wartości priorytetów. Faza druga rozpoczyna się od uszeregowania pustego i składa się z tylu kroków iteracyjnych ile operacji należy uszeregować. W każdym kroku iteracyjnym szereguje się kolejną operacje z listy. Szeregowanie to polega na próbnym wstawianiu tej operacji na wszystkie pozycje w zbudowanym w poprzednich krokach częściowym uszeregowaniu z jednoczesnym wyznaczeniem wartości pewnej funkcji oceniającej każde próbne wstawienie. Pozycja dla której funkcja oceniająca przyjmuje wartość najmniejszą jest pozycją, na którą wstawia się ostatecznie szeregowaną operacje. Z powyższego opisu widać, że elementami identyfikującymi dany algorytm bazujący na metodzie wstawień są tylko dwa elementy: postać priorytetów oraz postać funkcji oceniającej. Co więcej, z reguły jako funkcję oceniającą przyjmuje się rozważaną funkcje kryterialną. Powoduje to, że algorytmy te są praktycznie uniwersalne i można jej stosunkowo łatwo adaptować do różnego rodzaju problemów szeregowania operacji produkcyjnych. Niestety jak dotychczas odczuwa się brak w literaturze wyczerpujących badań nad algorytmami tego typu, mimo że panuje powszechne przekonanie – potwierdzone przeprowadzanymi testami – iż algorytmy te produkują rozwiązania najbardziej bliskie rozwiązaniom optymalnym w całej klasie algorytmów konstrukcyjnych.

Prezentowana praca jest pierwszą częścią opracowania dotyczącego metody wstawień w zastosowaniu do budowy algorytmów konstrukcyjnych dla klasycznych problemów szeregowania operacji produkcyjnych takich jak problem przepływowy i problem gniazdowy. Problemy te są powszechnie uważane za sztandarowe problemy teorii szeregowania, ponieważ modelują wiele rzeczywistych procesów produkcyjnych, stanowiąc jednocześnie bazę do modelowania innych, bardziej skomplikowanych procesów.

W pracy skupiamy się na problemie przepływowym. Dla tego problemu znany jest tylko jeden algorytm bazujący na technice wstawień, zaproponowany w 1983 r. w [1] i zwany tam algorytmem NEH od pierwszych liter nazwisk jego twórców. Szczegółowe badania porównawcze jednoznacznie pokazują, że jest to aktualnie najlepszy algorytm konstrukcyjny dla badanego problemu. Zaproponowane przez nas modyfikacje tego algorytmu pozwoliły jednak na zbudowanie wielu jego mutacji, które zachowują się znacznie lepiej, tzn. przy praktycznie dopuszczalnym zwiększeniu czasu obliczeń produkują uszeregowania o zauważalnie mniejszych wartościach funkcji celu. Badania porównawcze przeprowadzono na testowych przykładach z pracy [2], dostępnych przez sieć Internetu i powszechnie stosowanych do testowania algorytmów przybliżonych dla badanego zagadnienia. Rozmiar tych przykładów waha się od 1000 do 10 000 operacji i dlatego wydaje się być adekwatny do większości sytuacji praktycznych.

2. Problem przepływowy i jego permutacyjne sformułowanie.

Problem przepływowy można określić następująco. Dany jest zbiór zadań $J = \{1,2,...,n\}$ oraz zbiór maszyn $M = \{1,2,..., m\}$. Zadanie j, $j \in J$ ma być wykonywane kolejno na maszynach 1, 2, ..., m. Czynność polegającą na wykonywaniu zadania j na maszynie l nazywamy operacją i notujemy jako parę (l, j), Zadanie j na maszynie l (tzn. operacja (l, j) jest realizowane bez przerw w czasie $p_{l,j} > 0$. Przyjmuje się, że: (*i*) maszyna l, $l \in M$ może wykonywać co najwyżej jedną operację w danej chwili, (*ii*) nie można jednocześnie wykonywać więcej niż jednej operacji danego zadania w danej chwili oraz (*iii*) każda maszyna wykonuje zadania w tej samej kolejności. Uszeregowanie dopuszczalne jest definiowane przez momenty rozpoczęcia wykonywania $S(l, j)$ operacji (l, j), $l \in M$, $j \in J$ takie, że wszystkie powyższe ograniczenia są spełnione. Problem polega na znalezieniu uszeregowania dopuszczalnego minimalizującego moment wykonania wszystkich zadań $\max_{j \in J} C(m, j)$ gdzie $C(m, j) = S(m, j) + p_{m,j}$.

Dla dalszych rozważań wygodnie jest zastosować tzw. permutacyjne sformułowanie badanego problemu, w którym zmienną decyzyjną jest permutacja zbioru zadań J, zaś wartość kryterium, czyli moment wykonania wszystkich zadań, jest długością najdłuższej ścieżki w pewnym grafie. Mając na uwadze, że w prezentowanych algorytmach na danym kroku iteracyjnym jest uszeregowana tylko część zadań ze zbioru J, konieczne jest przyjęcie pewnych upraszających założeń notacyjnych. Niech A oznacza dowolny podzbiór zbioru zadań J, π - permutację tego zbioru, zaś Π - zbiór wszystkich takich permutacji. Formalnie π oraz Π powinny być indeksowane przez A, lecz dla uproszczenia notacji indeks ten pomijamy i dodatkowo przyjmujemy, że symbol $|\pi|$ oznacza liczność zbioru $A = \{\pi(1),\pi(2),...,\pi(|\pi|)\}$. Po tych uwagach możemy przejść bezpośrednio do permutacyjnego sformułowania badanego problemu. Dla $\pi \in \Pi$ określamy skierowany graf $G(\pi) = (M \times J, E^T \cup E^K)$ ze zbiorem wierzchołków $M \times J$ oraz łuków $E^T \cup E^K$ o postaci

$$E^T = \bigcup_{l=2}^{m} \bigcup_{i=1}^{|\pi|} \{((l-1,i),(l,i))\}, \qquad E^K = \bigcup_{l=1}^{m} \bigcup_{i=2}^{|\pi|} \{((l,i-1),(l,i))\}.$$

Każdy wierzchołek (l, i) reprezentuje operację $(l,\pi(i))$ i ma obciążenie równe $p_{l,\pi(i)}$. Łatwo zauważyć, że dla danej permutacji $\pi \in \Pi$ moment rozpoczęcia wykonywania $S(l,\pi(i))$ operacji $(l,\pi(i))$ równa się długości najdłuższej ścieżki w grafie $G(\pi)$ wychodzącej z wierzchołka (1,1) i dochodzącej do wierzchołka (l, i) (bez obciążenia tego wierzchołka). Co więcej, długość najdłuższej ścieżki (ścieżka krytyczna), oznaczanej dalej przez $C_{\max}(\pi)$, równa się momentowi wykonania wszystkich zadań, Ostatecznie problem sprowadza się do znalezienia $\pi^* \in \Pi$ takiej, że $C_{\max}(\pi^*) = \min_{\pi \in \Pi} C_{\max}(\pi)$.

3. Algorytm NEH

Jak już wspominaliśmy aktualnie najlepszym algorytmem konstrukcyjnym dla problemu przepływowego jest algorytm NEH z pracy [1], oparty jest na technice wstawień. W fazie wstępnej (faza pierwsza) algorytmu NEH przyjmuje się, że priorytet $\omega(j)$ zadania j jest równy sumie czasów wykonywania tego zadania na wszystkich maszynach, tzn.

$$\omega(j) = \sum_{li=1}^{m} p_{l,i} \,. \tag{1}$$

Intuicja przyjęcia takiej postaci priorytetu polega na tym, że wydaje się iż trzeba rozpocząć szeregowanie od największego zadania a następnie coraz mniejsze kolejne zadania „upychać" w powstałe dziury w uszeregowaniu. W fazie zasadniczej (faza druga) jako funkcję oceniająca próbne wstawienie danego zadania przyjmuje się funkcje kryterialną, czyli moment wykonania wszystkich aktualnie uszeregowanych zadań, łącznie z próbnie wstawionym zadaniem. Poniżej przedstawiamy szczegółowy schemat algorytmu NEH.

Algorytm NEH

Faza I. Dla każdego zadania j wyznacz jego priorytet $\omega(j)$, $j \in J$. Znajdź permutację zadań α taką, że $\omega(\alpha(1)) \geq \omega(\alpha(2)) \geq ... \geq \omega(\alpha(n))$.

Faza II. Połóż $\pi := (\alpha(1))$, $U := \{\alpha(1)\}$. Dla każdego $t = 2,3,...,n$ wykonaj:

Krok 1. Połóż $j := \alpha(t)$ i wyznacz t permutacji $\gamma^i = (\pi(1),...,\pi(i-1), j, \pi(i),...,\pi(t-1))$, $i = 1,2,...,t$ **,** przez włożenie zadania j przed każdą pozycję w permutacji π i dodatkowo za ostatnią pozycję w tej permutacji.. Wśród tych permutacji wyznacz permutację γ^s taką, że $1 \leq s \leq t$ oraz $C_{\max}(\gamma^i) = \min_{1 \leq i \leq t} C_{\max}(\gamma^i)$. Połóż $\pi := \gamma^s$ oraz $U := U \cup \{j\}$.

W powyższym schemacie zbiór U określa zadania, które zostały uszeregowane na danym etapie działania algorytmu. Po zakończeniu działania algorytmu wyprodukowana przez niego permutacja $\pi^{NEH} = \pi$.

Przeprowadzimy teraz analizę złożoności obliczeniowej algorytmu NEH. Faza I wymaga $O(n \log n)$ czasu. Ponieważ wyznaczenie wartości funkcji oceniającej wymaga $O(nm)$ czasu i w jednej iteracji w fazie II (krok 1) jest ona wyznaczana co najwyżej n razy, to do wykonania jednej iteracji w fazie II potrzeba $O(n^2 m)$ czasu. Stąd i z faktu, że w fazie II wykonuje się n-1 iteracji (kroków 1) wynika, że do realizacji tej faza potrzeba $O(n^3 m)$ czasu. Jest to również złożoność obliczeniowa całego algorytmu.

Przedstawiony powyżej czas działania algorytmu NEH można jednak znacznie zredukować wykorzystując specyficzny sposób wyznaczania wartości funkcji oceniającej przedstawiony w pracy [3]. Dla wygody czytelnika, krótko go naszkicujemy. Bezpośrednio przed rozpoczęciem wykonywania danej iteracji t dysponujemy permutacją $\pi = (\pi(1),\pi(2),...,\pi(t-1))$, według której zostało już uszeregowane t-1 zadań wyspecyfikowanych w zbiorze U. Dla każdej operacji (l,i), $l \in M$, $i \in U$ wyznaczamy w grafie $G(\pi)$ długość $r(l,i)$ najdłuższej ścieżki wychodzącej z wierzchołka (1,1) i dochodzącej do wierzchołka (l, i) oraz długość $q(l,i)$ najdłuższej ścieżki wychodzącej z wierzchołka (l, i) i dochodzącej do wierzchołka (m, t-1); zajmuje to łącznie $O(nm)$ czasu. Wykorzystując te wielkości, czas wyznaczenia wartości funkcji oceniającej $C_{\max}(\gamma^i)$ włożenie zadania j przed ustaloną pozycję i w permutacji π można zredukować z $O(nm)$ do $O(m)$. W tym celu wyliczamy w czasie $O(m)$ długość $R(l)$, l=1,...,m najdłuższej

ścieżki wychodzącej z wierzchołka (1,1) i dochodzącej do wierzchołka (*l*, *i*) w grafie $G(\gamma^i)$ z następujących oczywistych zależności rekurencyjnych:

$$R(1) = r(1,\pi(i-1)) + p_{1,j}, \quad R(l) = \max\{R(l-1),\ r(l,\pi(i-1) + p_{l,j}\},\ l=2,...,m.$$

Następnie w czasie $O(m)$ wyliczany $C_{\max}(\gamma^i)$ wykorzystując zależność

$$C_{\max}(\gamma^i) = \max_{1\le l\le m}(R(l) + q(l,\pi(i))).$$

Stąd wynika, że jedną iterację w fazie II można wykonać w czasie $O(nm)$ co powoduje, że złożoność obliczeniowa algorytmu NEH redukuje się do $O(n^2m)$. Badania numeryczne pokazują, że praktycznie redukcja ta oznacza $n/2$ - krotne skrócenie czasu obliczeń.

Na koniec prezentacji algorytmu NEH, warto zauważyć, że jego dominacja wśród algorytmów konstrukcyjnych została także częściowo potwierdzona teoretycznie przed analizę najgorszego przypadku. W pracy [5] pokazano, że algorytmy konstrukcyjne zbudowane na bazie innych najbardziej popularnych metod (metody priorytetowej oraz metody relaksacji) mają współczynnik najgorszego przypadku nie mniejszy niż $(m+1)/2$; współczynnik najgorszego przypadku jest definiowany jako maksymalna, po wszystkich przykładach m maszynowych, wartość ilorazu wartości funkcji celu produkowanej przez dany algorytm do minimalnej wartości funkcji celu. Z kolei w [4] i [5] pokazano, że współczynnik najgorszego przypadku algorytmu NEH jest nie większy niż $(m+1)/2$ oraz nie mniejszy niż $\sqrt{m/2}$. Co więcej, istnieje podejrzenie, że jego rzeczywista wartość jest równa podanemu dolnemu ograniczeniu.

4. **Modyfikacje algorytmu NEH**

Analizując szczegółowo algorytm NEH zauważamy, że wstawienie operacji *j* na danym kroku iteracyjnym *t* następuje w sposób optymalny dla ustalonej już częściowej kolejności wykonywania $\pi = (\pi(1),...,\pi(t-1))$ zadań ze zbioru *U*. Nie oznacza to jednak, że po znalezieniu dla zadania *j* najlepszej pozycji *s* i wyprodukowaniu nowej permutacji częściowej $\gamma^s = (\pi(1),...,\pi(s-1), j, \pi(s),...,\pi(t-1))$, nie można zmniejszyć wartości funkcji celu $C_{\max}(\gamma^s)$ zmieniając w otrzymanej permutacji położenie innych zadań. Ze względów czasowych jest oczywiste, że należy raczej skupić się na zmianie położenia tylko niewielkiej liczby zadań a najlepiej tylko pewnego jednego zadania $x \neq j$.

Przystępując do realizacji tak ogólnie zarysowanej idei modyfikacji algorytmu NEH należy odpowiedzieć na dwa pytania: (1) jak określić zadanie *x*, którego położenie będziemy zmieniać? oraz (2) na jaka pozycję powinno ono być wstawione? Odpowiedz na pytanie 2 wydaje się być oczywista. Należy zadanie *x* „wyjąć" z *t* elementowej permutacji $\pi = \gamma^s$ otrzymując *t*-1 elementową permutację oznaczaną dalej przez β i następnie wstawić zadanie *x* na taką pozycje w permutacji β, dla której wartość funkcji celu jest najmniejsza. Odpowiedź na pytanie 1 nie jest już taka oczywista i dlatego należy rozważyć pewna liczbę reguł wyboru zadania *x*.

Biorąc powyższe pod uwagę, proponujemy modyfikację algorytmu NEH, zwaną dalej algorytmem IR (od ang. Insert i Reinsert), która polega na dodaniu po kroku 1 fazy II algorytmu NEH dodatkowego kroku 2 o następującej postaci:

Krok 2. Określ zadanie $x \neq j$ stosując ustaloną regułę wyboru. Utwórz permutację $\beta = (\beta(1),...,\beta(t-1))$ przez usunięcie zadania x z permutacji π. Wyznacz t permutacji $\gamma^i = (\beta(1),...,\beta(i-1),x,\beta(i),...,\beta(t-1))$, $i = 1,2,...,t$, przez włożenie zadania x przed każdą pozycję w permutacji β i dodatkowo za ostatnią pozycję w tej permutacji. Wśród tych permutacji wyznacz permutację γ^s taką, że $1 \leq s \leq t$ oraz $C_{\max}(\gamma^s) = \min_{1 \leq i \leq t} C_{\max}(\gamma^i)$. Połóż $\pi := \gamma^s$

Zauważmy, że z wyjątkiem określenia zadania x oraz utworzenia permutacji β, krok 2 jest identyczny jak krok 1. Stąd, jeżeli złożoność obliczeniowa tych wyjątków nie będzie większa niż $O(nm)$, to złożoność obliczeniowa algorytmu IR pozostanie taka sama jak algorytmu NEH.

W celu określenia zadania x proponujemy następujące cztery reguły wyboru:

Reguła 1. Na ścieżce krytycznej w grafie $G(\pi)$ znajdź operację o największym czasie wykonywania (wyłączając operacje zadania j). Jako zadanie x przyjmij zadanie, w którego skład wchodzi ta operacja.

Reguła 2. Dla każdego zadania z permutacji π (z wyjątkiem zadania j) wyznacz obciążenie tego zadania ścieżką krytyczną definiowane jako suma czasów wykonywania tych jego operacji, które wchodzą w skład ścieżki krytycznej w grafie $G(\pi)$. Jako zadanie x przyjmij zadanie o największym obciążeniu.

Reguła 3. Wśród zadań z permutacji π (z wyjątkiem zadania j) znajdź zadanie o największej liczbie operacji wchodzących w skład ścieżki krytycznej w grafie $G(\pi)$. Jeżeli wybór jest jednoznaczny, przyjmij to zadanie jako x. W przeciwnym wypadku jako zadanie x przyjmij zadanie, które jednocześnie ma największą wartość priorytetu (1).

Reguła 4. Jako zadanie x przyjmij to zadanie z permutacji π (z wyjątkiem zadania j), którego usuniecie z tej permutacji spowoduje największe zmniejszenie wartości funkcji celu.

Idea konstrukcji wszystkich powyższych reguł jest identyczna i polega na tym, że należy usunąć to zadanie, które ma największy „wpływ" na wartość funkcji celu. W regule 4 jest ona realizowana bezpośrednio a w pozostałych regułach – pośrednio. Realizując tą idee liczymy na to, że zmieniając położenie takiego zadania znajdziemy po wykonaniu kroku 2 zauważalnie lepszą, w sensie wartości funkcji celu, permutację niż permutacja znaleziona po kroku 1.

Zauważmy, że czas realizacji każdej z reguł nie jest większy niż $O(nm)$. Dla reguł 1-3 uwaga ta jest oczywista, zaś dla reguły 4 wymaga krótkiego komentarza. Proponujemy rozpocząć wykonywanie tej reguły od wyliczenia w czasie $O(nm)$ długości $r(l,i)$, $q(l,i)$ najdłuższych dróg w grafie $G(\pi)$ zgodnie z definicją podaną w rozdziale 3. Wtedy wartość funkcji celu $C_{\max}(\delta)$ dla permutacji δ otrzymanej z permutacji π po usunięciu zadania $\pi(i)$ można wyznaczyć w czasie $O(m)$ wykorzystując oczywistą zależność

$C_{\max}(\delta) = \max_{1\le l\le m}(r(l,\pi(i-1)) + q(l,\pi(i+1)))$; $i \neq j$, $i = 1,...,t$, gdzie $\pi(0) = \pi(t+1) = 0$, $r(l,0) = q(l,0) = 0$ dla $l \in M$. Z faktu, że wszystkich permutacji δ jest nie więcej niż n wynika, że czas realizacji reguły 4 jest $O(nm)$. Reasumując oznacza to, że złożoność obliczeniowa algorytmu IR przy zastosowaniu reguł 1-4 wynosi $O(n^2m)$. Dalej algorytm IR z regułą wyboru k będziemy nazywać algorytmem IRk, k =1,2,3,4.

Zmiana położenia zadania x w permutacji π może spowodować, że zadanie j wstawione w kroku 1 nie zajmuje już „optymalnej" pozycji. Dlatego też proponujemy dalszą modyfikację algorytmów IR1-IR4 polegającą na zmianie położenia tego zadania a dokładniej na znalezieniu dla niego najlepszej pozycji w sensie wartości funkcji celu. W konsekwencji sugerujemy modyfikację algorytmu IRk, zwaną dalej algorytmem IRRk, która sprowadza się do dodaniu po kroku 2, dodatkowego kroku 3 o następującej postaci:

Krok 3. Utwórz permutację $\delta = (\delta(1),...,\delta(t-1))$ przez usunięcie zadania j z permutacji π. Wyznacz t permutacji $\gamma^i = (\delta(1),...,\delta(i-1), j, \delta(i),...,\delta(t-1))$, $i = 1,2,...,t$, przez włożenie zadania j przed każdą pozycję w permutacji δ i dodatkowo za ostatnią pozycję w tej permutacji. Wśród tych permutacji wyznacz permutację γ^s taką, że $1 \le s \le t$ oraz $C_{\max}(\gamma^s) = \min_{1\le i\le t} C_{\max}(\gamma^i)$. Połóż $\pi := \gamma^s$

Podobnie jak poprzednio, dodany krok 3 jest obliczeniowo identyczny jak krok 1 i dlatego złożoność obliczeniowa każdego z algorytmów IRR1-IRR4 wynosi $O(n^2m)$.

5. **Wyniki badań testowych**

Głównym celem badań testowych jest zbadanie, na ile rozwiązania wyprodukowane przez osiem zaproponowanych algorytmów IR1-IR4, IRR1-IRR4 są lepsze od rozwiązań generowanych przez algorytm NEH. Algorytmy będziemy testować na szczególnie trudnych 120 przykładach zaproponowanych przez Taillarda w pracy [2], których rozmiar waha się od 20 zadań i 5 maszyn (100 operacji) do 500 zadań i 20 maszyn (10 000 operacji); przykłady podzielone są na 12 grup po 10 przykładów o tej samej liczbie zadań i maszyn; dana grupa jest notowana jako *n*/*m*. Należy tu zauważyć, że trudność testowych przykładów wynika nie tylko z ich dużego rozmiaru ale z faktu, iż zostały one wybrane wśród dziesiątek tysięcy przykładów generowanych losowo, jako najbardziej „złośliwe". Dodatkowym elementem przemawiającym za wyborem tych przykładów są znane dla nich rozwiązania optymalne lub prawie optymalne, [6]. Rozwiązania te zostały otrzymane w wyniku ogromnie czasochłonnych i absorbujących wiele komputerów sesjach obliczeniowych stosując metodę podziału i ograniczeń. Jako rozwiązania początkowe przyjmowano rozwiązania otrzymane przez algorytm popraw bazujący na technice tabu search z pracy [7]. Dalej wartości funkcji celu dla tych rozwiązań będziemy nazywać wartościami bazowymi i notować jako C^B.

Algorytmy zaprogramowano w Delphi 4.0 a obliczenia przeprowadzono na komputerze Pentium II (333Mhz). Dla każdego przykładu wyznaczano względną poprawę

$$\Phi^A = 100\% \, (C^{NEH} - C^A)/C^{NEH} \tag{2}$$

wartości C^{NEH} funkcji celu otrzymanej algorytmem *NEH* przez zastosowanie algorytmu A generującego wartość funkcji celu C^A, $A \in$ { IR1,IR2,IR3,IR4,IRR1,IRR2,IR3, IRR4}.

Następnie dla każdej z 12 grup przykładów i każdego testowanego algorytmu *A* wyliczono wartość średnią $Av[\Phi^A]$ z 10 wartości Φ^A dla poszczególnych przykładów. Wyniki przedstawiono w tabeli 1.

Tab. 1. Względne poprawy algorytmu NEH

Grupa	Względna poprawa algorytmu NEH przez algorytm *A* w % ($Av[\Phi^A]$ dla algorytmu *A*)							
n/*m*	IR1	IR2	IR3	IR4	IRR1	IRR2	IRR3	IRR4
20/5	0,84	0,23	0,49	0,33	1,15	1,12	1,16	1,03
20/10	0,84	0,64	0,69	1,51	1,17	1,18	1,32	1,75
20/20	-0,36	0,22	0,16	0,96	0,19	0,50	0,59	1,45
50/5	0,11	0,24	0,13	0,05	-0,21	0,17	0,10	0,07
50/10	0,39	0,28	0,40	0,72	0,41	0,65	0,51	0,81
50/20	0,19	0,01	-0,12	0,93	0,29	0,90	0,44	1,34
100/5	0,01	-0,02	0,04	0,07	0,03	0,01	0,04	0,08
100/10	0,43	0,59	0,46	0,79	0,43	0,51	0,60	0,55
100/20	0,37	0,51	0,56	0,80	0,73	0,62	0,62	1,11
200/10	0,24	0,43	0,29	0,32	0,35	0,55	0,38	0,56
200/20	0,28	0,36	0,20	0,92	0,67	0,57	0,64	1,00
500/20	0,23	0,29	0,16	0,53	0,31	0,44	0,27	0,56
wszystkie	0,30	0,32	0,29	0,66	0,46	0,60	0,55	0,86

Z analizy tabeli 1 wynika, że w przypadku algorytmów IR reguła 4 zachowuje się najlepiej. Pozostałe reguł zachowują się prawie identycznie z lekką przewagą reguły 2. Podobna sytuacja występuje w przypadku algorytmów IRR z tym, że teraz przewaga reguły 2 nad regułami 1 i 3 jest bardziej widoczna. Potwierdza to nasze intuicyjne przewidywania, że należy zmieniać położenie tego zadania, które ma największy wpływ na wartość funkcji celu, ponieważ reguła 4 wyraża to w sposób bezpośredni a następnie w kolejności reguła 2.

Generalnie, wszystkie testowane algorytmy poprawiają algorytm NEH z tym, że zdarzają się takie grupy przykładów, dla których niektóre z nich zachowują się gorzej. Odnosi się to tylko do algorytmów IR1, IR2, IR3. Powtórna zmiana położenia wstawionego wcześniej zadania *j* realizowana przez algorytmy IRR wydaje się być - w obliczu wyników z tabeli 1 - w pełni uzasadniona. Względna poprawa rozwiązań produkowanych przez algorytm NEH przez najlepszy z testowanych algorytmów, jakim okazał się algorytm IRR4, wynosi średnio 0,86 %, wahając się od 0,07% do 1,75% w zależności od grupy przykładów.

Odpowiedź na pytanie czy taka względna poprawa jest godna uwagi zależy oczywiście od tego jak duża jest maksymalna możliwa względna poprawa. Dlatego też przeprowadzono dalsze badania testowe polegające na tym, że dla każdego przykładu wyznaczono maksymalną możliwą względną poprawę $\Phi^{\max}$,

$$\Phi^{\max} = 100\% \, (C^{NEH} - C^B) / C^{NEH} . \qquad (3)$$

W powyższym wzorze zamiast minimalnej wartości funkcji celu wprowadzono wartość bazową C^B, która - jak już wspominaliśmy - dość dobrze ją przybliża. Następnie dla każdej z 12 grup przykładów wyliczono wartość średnią $Av[\Phi^{\max}]$ z 10 wartości $\Phi^{\max}$ dla poszczególnych przykładów. Wyniki przedstawiono w drugiej kolumnie tabeli 2. Dodatkowo w trzeciej i czwartej kolumnie powtórzono z tabeli 1 wartości średnich względnych popraw dla najlepszych algorytmów z grupy IR i IRR, tzn. algorytmów A = IR4, IRR4.

Tab. 2. Porównanie względnych popraw z maksymalnymi względnymi poprawami

Grupa	$Av[\Phi^{\max}]$	$Av[\Phi^A]$ dla alg. A		$\frac{Av[\Phi^A]}{Av[\Phi^{\max}]}$ w % dla alg. A	
n/m		IR4	IRR4	IR4	IRR4
20/5	3,11	0,33	1,03	10,5	33,1
20/10	4,70	1,51	1,75	32,1	37,2
20/20	3,53	0,96	1,45	27,2	41,0
50/5	0,70	0,05	0,07	7,6	9,7
50/10	4,79	0,72	0,81	15,1	16,9
50/20	5,81	0,93	1,34	16,0	23,1
100/5	0,48	0,07	0,08	14,4	17,6
100/10	2,29	0,79	0,55	34,4	24,3
100/20	5,16	0,80	1,11	15,5	21,5
200/10	1,36	0,32	0,56	23,4	41,2
200/20	4,14	0,92	1,00	22,2	24,1
500/20	2,16	0,53	0,56	24,5	26,1
wszystkie	3,18	0,66	0,86	20,2	26,3

Z analizy tabeli 2 wynika, że maksymalna względna poprawa rozwiązań produkowanych przez algorytm NEH wynosi średnio 3,18 %, wahając się od 0,48% do 5,81% w zależności od grupy przykładów. Porównując te wielkości z wartościami popraw algorytmu NEH przez algorytm IR4 lub IRR4 należy zauważyć, że algorytmy te istotnie poprawiają rozwiązania dostarczone przez algorytm NEH. Szczególnie jasno wynika to z dwóch ostatnich kolumn tabeli 2, w których podano jaki procent maksymalnej względnej poprawy stanowi względna poprawa produkowana odpowiednio przez algorytm IR4 oraz IRR4. Przykładowo, dla algorytmu IRR4 wartość ta wynosi średnio 26%, wahając się od 10% do 41%.

Podsumowując wyniki przeprowadzonych badań testowych możemy stwierdzić, że proponowane modyfikacje algorytmu NEH dostarczają algorytmów produkujących rozwiązania o istotnie mniejszych wartościach funkcji celu. Co więcej, czas obliczeń dla najbardziej czasochłonnego algorytmu, jakim jest algorytm IRR4, nie przekracza 3 sekund dla największych testowanych przykładów o rozmiarze 10 000 operacji. Oznacza to, iż proponowany w tej pracy kierunek badań jest bardzo obiecujący i powinien być kontynuowany.

Literatura

1. Nawaz M., Enscore Jr. E.E., Ham I.: A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem. OMEGA International Journal of Management Science, 11, 1983, str. 91-95.
2. Taillard E.: Benchmarks for basic scheduling problems, European Journal of Operational Research, 64, 1993, str. 278-285.
3. Taillard E.: Some efficient heuristic methods for flow shop sequencing. European Journal of Operational Research, 47, 1990, str. 65-74.
4. Nowicki E., Smutnicki C.: New results in the worst-case analysis for flow-shop scheduling. Discrete Applied Mathematics, 46, 1993, str. 21-41.
5. Nowicki E.: Metoda tabu w problemach szeregowania zadań produkcyjnych**,** Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 1999.
6. Vaessens R.J.M.: OR library, Internet: http://mscmga.ms.ic.ac.uk.info.html. Files flowshop1.txt, flowshop2.txt.
7. Nowicki E., Smutnicki C.: A fast taboo search algorithm for the permutation flow-shop problem. European Journal of Operational Research, 91, 1996, str. 160-175.

Dr hab. inż. Eugeniusz NOWICKI
Mgr inż. Mariusz MAKUCHOWSKI
Instytut Cybernetyki Technicznej, Politechnika Wrocławska
53-342 Wrocław, ul. Janiszewskiego 11/17
tel.: (071) 320-29-61
e-mail: enowicki@ict.pwr.wroc.pl