ALGORYTM PERTURBACYJNY DLA PROBLEMU PRZEPŁYWOWEGO

Mariusz MAKUCHOWSKI

Streszczenie: Proponowany w tej pracy algorytm perturbacyjny *PNEH* (dedykowany permutacyjnemu problemowi przepływowemu) pozwala na dostarczanie dowolnej liczby dobrych rozwiązań startowych, nadających się do użytku w algorytmach popraw (jako rozwiązania startowe oraz rozwiązania dywersyfikacyjne). Jakość dostarczanych rozwiązań jest na tyle dobra, iż nawet bez dodatkowego algorytmu popraw, prezentowany algorytm *PNEH* uzyskuje rozwiązania lepsze niż jego klasyczny, deterministyczny odpowiednik *NEH*. W pracy przeprowadzono eksperymenty numeryczne na znanych w literaturze przykładach testowych. Następnie oceniono wpływ najważniejszych elementów perturbacji danych na wydajność proponowanego algorytmu.

Słowa kluczowe: perturbacja danych, algorytmy perturbacyjne, algorytm NEH, problem przepływowy.

1. Wstęp

Rozwiązując różnorakie problemy optymalizacji łatwo można zauważyć, iż dla części z nich nie istnieją dokładne algorytmy wielomianowe (np. problemy należące do klasy problemów NP z wykluczeniem klasy P). W problemach takich dla instancji o dużym rozmiarze ponadwielomianowy (ze względu na czas pracy) algorytm dokładny działałby w nieakceptowalnym czasie. Alternatywą dla stosowania algorytmów dokładnych staje się stosowanie szybkich, ale nie zawsze dających optymalne rozwiązanie, algorytmów przybliżonych.

1.1. Algorytmy popraw

Algorytmy przybliżone ze względu na ogólną zasadę działania można podzielić: na algorytmy konstrukcyjne i algorytmy popraw. Idea algorytmów konstrukcyjnych polega na tym, że startując od pustego rozwiązania, w każdym kolejnym kroku iteracyjnym dokłada się w odpowiednie miejsce jeden element rozwiązania częściowego, nieuwzględniony do tej pory. Iteracje powtarzają się, aż do otrzymania rozwiązania pełnego, a to kończy działanie algorytmu. Sposób wyboru wkładanego na danym etapie elementu rozwiązania, oraz miejsca w które należy je włożyć określa dany algorytm konstrukcyjny.
Z kolei algorytmy popraw startują z pełnego uszeregowania, otrzymanego zazwyczaj przez algorytm konstrukcyjny. W kolejnych krokach iteracyjnych algorytm popraw stara się polepszyć to rozwiązanie w sensie wartości rozpatrywanego kryterium.

Kluczowym elementem podczas konstrukcji algorytmów popraw jest jakość dostarczanych rozwiązań startowych. Algorytmy rozpoczynające pracę z dobrego miejsca w przestrzeni rozwiązań znajdują przeważnie rozwiązania lepsze niż te same algorytmy startujące ze słabszych punktów i czynią to w znacznie krótszym czasie.

W algorytmach ewolucyjnych (należących do klasy algorytmów popraw) już na starcie zachodzi potrzeba dostarczenia kilkunastu różnych rozwiązań startowych. Podobnie w pozostałych algorytmach popraw dokonujących co pewien czas restartów czy przenoszących poszukiwania w inne losowe obszary przestrzeni rozwiązań, istotnym elementem jest dostarczenie wielu punktów startowych. Ważne jest przy tym, aby rozwiązania te były nie tylko dość dobre w sensie optymalizowanego kryterium, ale także by różniły się między sobą w istotny sposób. Zróżnicowane ustawienia startowe pomagają w przeszukaniu różnych regionów przestrzeni rozwiązań, a co za tym idzie zmniejszają ryzyko utknięcia algorytmu w jednym obszarze tej przestrzeni.

Przeważnie podczas konstruowania algorytmów popraw często stosuje się niewielką liczbę algorytmów konstrukcyjnych oraz dostarcza się losowe rozwiązania dopuszczalne. Powód takiego stanu jest oczywisty, projektując algorytm popraw badacz nie musi kreować kilkunastu czy kilkudziesięciu algorytmów konstrukcyjnych.

1.2. Algorytmy perturbacyjne

Zamiast pojedynczego algorytmu konstrukcyjnego oraz rozwiązań losowych, alternatywą może być zastosowanie algorytmu perturbacyjnego. Algorytm taki dostarcza dowolnie liczny zestaw rozwiązań, których jakość zbliżona jest do rozwiązań otrzymywanych przy zastosowaniu dedykowanych algorytmów konstrukcyjnych. Dodatkową zaletą zastosowania algorytmu perturbacyjnego jest to, że otrzymane rozwiązania różnią się miedzy sobą w istotny sposób ze względu na położenie w przestrzeni rozwiązań. Algorytmy perturbacyjne polegają na modyfikacji istniejących algorytmów. Modyfikacja ta polega na losowym zaburzaniu pracy algorytmu pierwotnego i może dotyczyć trzech różnych elementów:

(i) perturbacja uzyskanego rozwiązania,
(ii) perturbacja zmiennych sterujących algorytmem,
(iii) perturbacja danych instancji rozwiązywanego problemu.

Wariant (i) perturbacji jest najmniej korzystny, gdyż rozwiązania uzyskiwane poprzez niewielkie zaburzania wyjściowego rozwiązania leżą blisko siebie (w przestrzeni rozwiązań), ponadto często są słabszej jakości niż zmieniany oryginał. Wariant (ii) perturbacji wymaga ingerencji w istniejący algorytm i jest zależny od zasady jego działania. Wariant (iii) jest najlepszym rozwiązaniem, nie wymaga on ingerencji w istniejący algorytm, ale nie zawsze można go zastosować ze względu na to, iż w niektórych problemach zmiana danych instancji może wpływać na zmianę zbioru rozwiązań dopuszczalnych. Zastosowanie wariantu (iii) polega on wprowadzeniu niewielkich zaburzeń w instancji problemu oraz wygenerowaniu rozwiązania algorytmem dedykowanym danemu zagadnieniu. Na uzyskane w ten sposób rozwiązanie należy następnie nanieść oryginalne dane instancji i wyznaczyć wartość funkcji celu. Niestety dla niektórych modeli problemów, zmiana danych może skutkować tym, iż rozwiązanie staje się niedopuszczalne, wtedy należy zastanowić się nad innym modelem problemu lub nad (ii) wersją perturbacji.

Jeżeli analizowany problem jest problemem liczbowym, np. problem komiwojażera z odległościami między miastami, czy problem szeregowania zadań z czasami trwania operacji, perturbację danych (iii) można przeprowadzić na następujące sposoby:

(i) jakościowa perturbacja danych: polega na zmianie niewielkiej ilości danych na wartości losowe z zakresu [*lb*,*ub*], gdzie *lb* i *ub* oznaczają dolną i górną granicę wartości występujących danych,

(ii) ilościowa perturbacja danych: dokonuje niewielkiej zmiany wszystkich danych problemu (oczywiście z wyłączeniem danych opisujących strukturę problemu, tj. liczby miast dla problemu komiwojażera, liczby zadań i maszyn dla problemu przepływowego),
(iii) mieszana perturbacja danych: polega na zmianie pewnej liczby danych w niewielkim zakresie i jest połączeniem perturbacji ilościowej z jakościową.

2. Permutacyjny problem przepływowy

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 (j, l). Zadanie j na maszynie l (tzn. operacja (j, l) jest realizowana bez przerw w czasie $p_{(j,l)} > 0$. Ponadto 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,
(iii) każda maszyna wykonuje zadania w tej samej kolejności.

Uszeregowanie dopuszczalne jest definiowane przez momenty rozpoczęcia wykonywania $S(j, l)$ operacji (j, l), $j \in J$, $l \in M$ takie, że wszystkie powyższe ograniczenia są spełnione. Problem polega na znalezieniu uszeregowania dopuszczalnego minimalizującego moment wykonania wszystkich zadań $\max_{j \in J} C(j,m)$, gdzie $C(j,m) = S(j,m) + p_{(j,l)}$.

Przedstawiony model uwzględnia wszystkie możliwe uszeregowania (jest ich nieskończona liczba). Można pokazać, że wśród harmonogramów optymalnych istnieje taki w którym wszystkie operacje dosunięte są maksymalnie w lewo na osi czasu. Poszukiwania rozwiązania optymalnego można ograniczyć zatem do zbioru rozwiązań z lewostronnie dosuniętymi operacjami. W takim przypadku każde rozwiązanie może być jednoznacznie reprezentowane kolejnością wykonywania zadań na maszynach. Przy powyższym założeniu, model problemu można uprościć do następującej postaci.

Niech π oznacza permutację zbioru zadań J, zaś Π zbiór wszystkich takich permutacji. Dodatkowo niech $C_{\max}(\pi)$ równa się momentowi wykonania wszystkich zadań w kolejności $\pi \in \Pi$. Ostatecznie problem sprowadza się do znalezienia permutacji $\pi^* \in \Pi$ takiej, że długość odpowiadającego jej uszeregowania przyjmuje możliwie najmniejszą wartość. Formalnie zapisujemy to w postaci:

$$C_{\max}(\pi^*) = \min_{\pi \in \Pi} C_{\max}(\pi) \ . \qquad (1)$$

3. Algorytm PNEH

Prezentowany w pracy perturbacyjny algorytm *PNEH* jest przybliżonym algorytmem konstrukcyjnym dedykowanym permutacyjnemu problemowi przepływowemu. Jego bazową funkcją jest dedykowany rozważanemu problemowi algorytm *NEH* [1,2,3]. (W pracy [1] zaproponowano po raz pierwszy algorytm *NEH*, w pracy [2] proponuje się pewne jego akceleracje zmniejszające złożoność obliczeniową, a w pracy [3] przeprowadza analizę najgorszego przypadku). Ponieważ, w przeciwieństwie do algorytmu *NEH* algorytm *PNEH* zawiera w sobie istotne elementy losowe, tzn. dostarcza on różnych rozwiązań dla

tej samej instancji problemu. W wielu przypadkach, takich jak generowanie rozwiązań początkowych dla wielościeżkowych algorytmów popraw jest to cechą niezmiernie przydatną.

Stabilność obliczeń algorytmu *NEH* wymaga komentarza, gdyż podczas pracy tego algorytmu mogą wystąpić niejednoznaczne działania typu:

(i) wybór kolejnego zadania do szeregowania z kilku zadań o jednakowej wadze,

(ii) wybór pozycji do szeregowanego zadania z kilku pozycji jednakowo dobrych.

Jednak spotykane implementacje algorytmu *NEH* zawsze działają w deterministyczny, czyli powtarzalny sposób. Ponadto, nawet gdyby w powyżej opisanych sytuacjach, zachowanie algorytmu *NEH* nie było zdeterminowane, to sytuacje te nie występują dla wszystkich instancji problemu.

Dodatkową przewagą algorytmu *PNEH* nad jego deterministycznym odpowiednikiem jest możliwość wyboru jakości dostarczanego rozwiązania. Zwiększenie jakości otrzymywanych rozwiązań odbywa się oczywiście kosztem czasu działania algorytmu. Ponieważ algorytm *PNEH* dostarcza rozwiązań mocno zróżnicowanych (siłę zróżnicowania można sobie dowolnie dobierać od zerowego zróżnicowania po rozwiązania całkowicie losowe) można stosować kilkukrotne jego uruchomienie i ostatecznie wybrać najlepsze zwrócone rozwiązanie. Liczbę powtórzeń algorytmu oznaczono w dalszej części pracy grecką literą η. Dla zwiększenia przejrzystości algorytmu, opis działań w kolejnych powtórzeniach zastąpiony został działaniem na zestawie danych.

3.1. Opis implementacji

Przedstawiając ogólnie zagadnienie perturbacyjnego algorytm *PNEH* możemy jego zasadę działania zapisać w trzech krokach:

(i) konstruujemy zestaw η zmodyfikowanych instancji rozwiązywanego problemu,

(ii) wyznaczamy rozwiązania utworzonych instancji przy użyciu algorytmu *NEH*,

(iii) oceniamy każde otrzymane rozwiązanie dla oryginalnych danych problemu i wybieramy najlepsze z nich. Wybrane rozwiązanie zwracane jest jako otrzymane algorytmem *PNEH*.

W kroku (i) przedmiotowego algorytmu, generujemy η zmodyfikowanych instancji rozwiązywanego problemu. Modyfikacji podlegają czasy $p_{(j,l)}$ wykonywania niektórych operacji $(j,l),\ j \in J, l \in M$. Prawdopodobieństwo modyfikacji każdej z operacji jest jednakowe, a jego poziom określa parametr α przyjmujący wartości z zakresu od 0% do 100%. Dla każdej modyfikowanej operacji (j,l) losowana jest wartość $RND_{(j,l)}$ z zakresu $[LB,UB]$, gdzie zakresy przedziałów wyznaczane są jako najmniejszy i największy czas trwania operacji występujący w instancji, $LB = \min_{j \in J, l \in M} p_{(j,l)}$, $UB = \max_{j \in J, l \in M} p_{(j,l)}$. Nową wartość $p'_{(j,l)}$ czasu trwania zmienianej operacji (j,l) wyznacza się jako liniową kombinację wartości oryginalnej $p_{(j,l)}$ oraz losowej wartości $RND_{(j,l)}$:

$$p'_{(j,l)} = (1-\beta) \cdot p_{(j,l)} + \beta \cdot RND_{(j,l)}, \tag{2}$$

gdzie β przyjmujące wartości z zakresu od 0% do 100% jest ustalonym parametrem sterującym perturbacją.

W kroku (ii) algorytmu *PNEH* rozwiązywane są wszystkie utworzone w poprzednim kroku perturbacyjne instancje problemu. W tym celu dla każdej instancji uruchomiony zostaje deterministyczny algorytm *NEH*. W wyniku działania tego kroku otrzymujemy zestaw harmonogramów oraz wynikający z nich zestaw kolejności wykonywania zadań.

Ostatnim (iii) krokiem jest wyznaczenie η harmonogramów oryginalnych czynności na podstawie zestawu, kolejności wykonywania zadań, uzyskanego w kroku (ii). Ostatecznie z zestawu otrzymanych harmonogramów należy wybrać najlepszy, w sensie przyjętego kryterium będącego długością uszeregowania (1). W dalszej części pracy algorytm *PNEH* z ustalonymi parametrami perturbacji będzie oznaczany przez: $PNEH(\eta,\alpha,\beta)$.

3.2. Analiza złożoności obliczeniowej

Analiza złożoności obliczeniowej algorytmu *PNEH* jest łatwa do przeprowadzenia. W kroku (i) wykonujemy perturbacje o całkowitej złożoności $O(\eta \cdot nm)$. Identyczną złożoność posiada krok (iii), wyznaczający η harmonogramów. W kroku (ii) wywoływany jest η krotnie algorytm *NEH,* którego implementacja (z zastosowaniem technik akceleracji opisanych w pracy[2]) posiada złożoność obliczeniową równą $O(n^2m)$. Ostatecznie złożoność obliczeniowa algorytmu *PNEH* wynosi $O(\eta \cdot n^2m)$.

3.3. Analiza zdegenerowanej perturbacji

W skrajnym przypadku, gdy w perturbacji parametr α lub β osiąga wartość zerową, perturbacja nie zmienia żadnej wartości danych instancji i algorytm *PNEH* działa dokładnie tak samo jak algorytm *NEH*.

Kolejnym skrajnym przypadkiem perturbacji jest sytuacja, gdy oba parametry α i β równocześnie osiągają wartości 100%, wtedy dane rozwiązywanych problemów nie są w żaden sposób skorelowane z oryginalnymi danymi a otrzymane rozwiązanie (kolejność zadań) jest zupełnie losowe. Algorytm *PNEH* działa wtedy jak generator losowych rozwiązań. Regulując wartościami parametrów α i β możemy wybierać "siłę" perturbacji. Zwiększając "siłę" perturbacji dostarczane rozwiązania pochodzą z coraz większego obszaru przestrzeni rozwiązań. Zmniejszając zaś "siłę" perturbacji, dostarczane uszeregowania skupiają się wokół rozwiązania generowanego algorytmem *NEH* (w sensie pewnej miary odległości w przestrzeni rozwiązań). Prezentowany algorytm, można więc zastosować jako narzędzie w algorytmach popraw, do generowania zarówno dobrej jakości dowolnej liczby rozwiązań startowych jak i do tworzenia dowolnej liczby rozwiązań dywersyfikacyjnych.

4. Badania testowe

Celem badań testowych jest przebadanie wpływu η,α,β parametrów perturbacji na jakość otrzymywanych rozwiązań algorytmu *PNEH*. Jakość danego uszeregowania będzie oceniana na podstawie długości otrzymanego harmonogramu względem rozwiązania uzyskanego algorytmem *NEH*. Wszystkie prezentowane w niniejszej pracy algorytmy

zostały napisane w języku C# w środowisku Microsoft Visual Studio 2010 Professional. Badania numeryczne przeprowadzane były na komputerze wyposażonym w procesor i7 i pamięć 8GB pracującym w systemie 64 bitowym Windows 7.

4.1. Przykłady testowe

Algorytm był testowany na szczególnie trudnych 120 przykładach zaproponowanych przez Taillard'a w pracy [4]. Rozmiar tych instancji waha się od 20 zadań i 5 maszyn (100 operacji) do 500 zadań i 20 maszyn (10 000 operacji). Dane testowe podzielone są na 12 grup po 10 przykładów o tej samej liczbie zadań i maszyn. Każda grupa opisywana jest dodatkowo parą parametrów n/m (liczba zadań/liczba maszyn). Należy tu zauważyć, że trudność testowych instancji wynika nie tylko z ich dużego rozmiaru, ale również z faktu, iż zostały one wybrane wśród tysięcy przykładów generowanych losowo, jako te najbardziej „złośliwe". Dodatkowym elementem przemawiającym za wyborem tych przykładów są znane dla nich rozwiązania optymalne lub prawie optymalne [5]. 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 [6].

W ramach poniższych testów dla każdego przykładu o numerze $id \in \{1,...,120\}$ utworzone zostały uszeregowania algorytmem *NEH* i $PNEH(\eta,\alpha,\beta)$. Długości tych uszeregowań oznaczamy kolejno przez $C^{NEH}(id)$ i $C^{PNEH(\eta,\alpha,\beta)}(id)$. Następnie na ich podstawie obliczono błąd rozwiązania otrzymanego przez perturbacyjny algorytm *PNEH* w stosunku do deterministycznego algorytmu *NEH* jako:

$$\rho(id) = \frac{C^{PNEH(\eta,\alpha,\beta)}(id) - C^{NEH}(id)}{C^{NEH}(id)} \cdot 100\%. \tag{3}$$

Ostatecznie dla każdej z grup $gr \in \{1,\ldots 12\}$ uśredniono błąd względny z 10 instancji otrzymując średni błąd względny grupy $\overline{\rho}(gr) = \sum_{id=10\cdot gr-9}^{10\cdot gr} \rho(id)/10$.

4.2. Wyniki badań

W pierwszym teście przebadano wariant jakościowej perturbacji danych. Szczegółowym badaniom poddano wpływ parametru α na średnią wartość względnego błędu $\overline{\rho}$. Współczynnik α przyjmuje wartości z zakresu od 0% do 100% i decyduje on o prawdopodobieństwie zmiany długości czasu trwania każdej operacji w problemie. Wyniki pierwszego eksperymentu zawarte są w tabeli 1.

Tab. 1. Wpływ zmiany prawdopodobieństwa α w perturbacji jakościowej na $\overline{\rho}$ średni błąd algorytmu $PNEH(\eta=10,\alpha,\beta=100\%)$ względem algorytmu *NEH*

Grupa	Względny błąd $\overline{\rho}$ [%] algorytmu $PNEH(\eta=10,\alpha[\%],\beta=100\%)$									
n/m	$\alpha=0$	$\alpha=1$	$\alpha=2$	$\alpha=4$	$\alpha=6$	$\alpha=10$	$\alpha=20$	$\alpha=40$	$\alpha=60$	$\alpha=100$
20/5	0,0	-1,3	-1,3	-1,2	-1,2	-0,5	1,5	6,0	7,6	12,2
20/10	0,0	-1,6	-1,5	-0,7	-0,1	0,8	3,4	7,4	10,6	15,0
20/20	0,0	-0,6	-0,6	-0,1	0,8	0,7	3,9	6,7	8,0	12,2
50/5	0,0	0,1	0,2	0,3	0,9	1,5	2,7	5,0	8,0	10,9
50/10	0,0	0,0	0,5	0,4	1,6	2,0	4,7	8,6	11,5	15,5
50/20	0,0	-0,9	-0,8	0,2	0,7	2,1	4,1	8,3	10,9	16,1
100/5	0,0	-0,1	0,2	0,5	0,7	1,2	2,7	4,4	5,8	8,6
100/10	0,0	-0,1	0,4	0,8	1,2	2,2	4,0	7,6	10,0	13,3
100/20	0,0	0,3	0,7	1,2	1,7	3,0	4,7	8,9	11,1	15,7
200/10	0,0	0,1	0,3	0,9	1,1	1,7	3,4	6,2	8,2	11,3
200/20	0,0	0,1	0,6	1,4	1,8	3,1	4,8	8,8	11,0	15,0
500/20	0,0	0,4	0,7	1,3	1,7	2,7	4,5	7,1	9,0	11,4
razem	0,0	-0,3	0,0	0,4	0,9	1,7	3,7	7,1	9,3	13,1

Drugi z przeprowadzonych testów bada wpływ "siły" perturbacji ilościowej na wydajność algorytmu *PNEH*. Zmieniany, w zakresie od 0% do 100%, w teście parametr β określa procentowy wpływ losowej wartości *RND* na nową długość $p'_{(j,l)}$ operacji (j,l), $j\in J$, $l\in M$. Otrzymane wartości średniego względnego błędu algorytmu *PNEH* z ilościową perturbacją danych zawarte są w tabeli 2.

Z wyników zawartych w tabelach 1 i 2 widać, iż algorytm *NEH* ($PNEH(\eta=10,\alpha=0\%,\beta=0\%)$) jest ponad 13% lepszy (w sensie wartości funkcji celu dostarczanych rozwiązań) niż algorytm losowy ($PNEH(\eta=10,\alpha=100\%,\beta=100\%)$) zwracający najlepsze z dziesięciu wylosowych rozwiązań. Z dalszej analizy wynika także, że przy stosunkowo niewielkiej perturbacji danych zarówno ilościowej ($\eta=10,\alpha=1\%,\beta=100\%$) jak i jakościowej ($\eta=10,\alpha=100\%,\beta=1\%,...,10\%$) algorytm *PNEH* nie tylko dostarcza różne losowe rozwiązania, ale rozwiązania te są statystycznie lepsze niż dostarczane algorytmem *NEH*. Podczas porównywania wydajności obu algorytmów należy pamiętać, że obserwacje w opisanych powyżej testach dotyczą sytuacji, w których jednokrotne uruchomienie algorytmu *PNEH* wybiera najlepsze z dziesięciu ($\eta=10$) perturbacyjnych rozwiązań.

Tab. 2. Wpływ zmiany współczynnika β w perturbacji ilościowej na $\overline{\rho}$ średni błąd algorytmu $PNEH(\eta=10,\alpha=100\%,\beta)$ względem algorytmu *NEH*

Grupa	Względny błąd $\overline{\rho}$ [%] algorytmu $PNEH(\eta=10,\alpha=100\%,\beta[\%])$									
n/m	$\beta=0$	$\beta=1$	$\beta=2$	$\beta=4$	$\beta=6$	$\beta=10$	$\beta=20$	$\beta=40$	$\beta=60$	$\beta=100$
20/5	0,0	-0,8	-1,1	-1,1	-1,0	-1,6	-1,2	2,2	7,0	12,2
20/10	0,0	-1,0	-1,2	-1,4	-1,4	-1,4	-0,4	2,7	7,6	15,0
20/20	0,0	-0,8	-1,1	-1,1	-0,9	-1,1	0,1	3,0	6,8	12,2
50/5	0,0	-0,4	-0,3	-0,4	-0,3	-0,3	0,3	2,5	5,9	10,9

50/10	0,0	-0,9	-1,2	-0,7	-1,0	-0,6	0,6	4,7	9,4	15,5
50/20	0,0	-1,4	-1,3	-1,3	-1,2	-0,7	0,1	4,2	9,3	16,1
100/5	0,0	-0,2	-0,2	-0,2	-0,2	-0,1	0,1	1,7	5,2	8,6
100/10	0,0	-0,7	-0,7	-0,5	-0,6	-0,3	0,9	3,7	8,5	13,3
100/20	0,0	-0,5	-0,6	-0,5	-0,3	-0,3	1,1	5,1	9,6	15,7
200/10	0,0	-0,3	-0,2	-0,1	-0,1	0,0	0,7	3,1	6,4	11,3
200/20	0,0	-0,6	-0,5	-0,5	-0,4	0,0	0,9	4,7	9,1	15,0
500/20	0,0	-0,1	-0,1	-0,1	0,0	0,2	1,0	4,2	7,9	11,4
razem	0,0	-0,6	-0,7	-0,7	-0,6	-0,5	0,3	3,5	7,7	13,1

Kolejnym testem jest zbadanie wpływu parametru η na jakość generowanych rozwiązań. Warto tu zaznaczyć, że czas działania algorytmu *PNEH* dla danej instancji problemu jest dokładnie η razy większy niż algorytmu *NEH*. W tabeli 3 zawarte zostały wartości $\overline{\rho}$ średnich błędów względnych poszczególnych grup, dla zmieniającego się parametru η.

Tab. 3. Wpływ ilości η perturbowanych instancji na $\overline{\rho}$ średni błąd algorytmu $PNEH(n, \alpha = 10\%, \beta = 10\%)$ względem algorytmu *NEH*

Grupa	Względny błąd $\overline{\rho}$ [%] algorytmu $PNEH(n, \alpha = 10\%, \beta = 10\%)$									
n/m	$\eta = 1$	$\eta = 2$	$\eta = 4$	$\eta = 6$	$\eta = 10$	$\eta = 20$	$\eta = 40$	$\eta = 60$	$\eta = 100$	$\eta = 1000$
20/5	0,2	-0,1	-0,6	-0,9	-0,9	-1,5	-1,7	-1,8	-2,0	-2,2
20/10	-0,3	-0,8	-1,4	-1,6	-1,7	-1,9	-2,0	-2,1	-2,1	-2,7
20/20	0,2	-0,5	-1,1	-1,2	-1,4	-1,5	-1,6	-1,6	-1,8	-2,2
50/5	0,3	-0,2	-0,3	-0,3	-0,3	-0,4	-0,4	-0,5	-0,5	-0,6
50/10	0,0	-0,4	-0,7	-0,9	-1,0	-1,3	-1,4	-1,6	-1,6	-2,2
50/20	0,1	-0,6	-1,0	-1,2	-1,5	-1,7	-1,9	-2,0	-2,1	-2,4
100/5	0,1	0,0	-0,1	-0,2	-0,2	-0,3	-0,3	-0,3	-0,4	-0,4
100/10	0,1	-0,2	-0,4	-0,6	-0,6	-0,7	-0,8	-0,8	-0,9	-1,1
100/20	0,3	-0,1	-0,3	-0,4	-0,5	-0,7	-0,9	-0,9	-0,9	-1,4
200/10	0,0	-0,1	-0,2	-0,2	-0,2	-0,3	-0,3	-0,4	-0,4	-0,6
200/20	-0,1	-0,2	-0,4	-0,4	-0,6	-0,7	-0,7	-0,8	-0,8	-1,1
500/20	0,3	0,1	-0,1	-0,1	-0,1	-0,1	-0,2	-0,3	-0,3	-0,4
razem	0,1	-0,3	-0,5	-0,7	-0,8	-0,9	-1,0	-1,1	-1,2	-1,4

Z analizy tabeli 3 wynika, iż podejście perturbacyjne w analizowanym problemie jest wysoce skuteczne. Nawet liczba ($\eta = 2$) generowanych perturbacyjnie instancji z właściwie wysterowanym zaburzeniem (na poziomie $\alpha = 10\%, \beta = 10\%$) pozwala uzyskać lepszą jakość rozwiązań niż deterministycznym algorytmem *NEH*. Jakość prezentowanego algorytmu jest na tyle wysoka, iż może on być stosowany jako samodzielny algorytm konstrukcyjny generujący rozwiązania wyższej jakości niż wiodący algorytm *NEH* uważany do tej pory za najlepszy algorytm konstrukcyjny dedykowany rozważanemu problemowi. Oczywiście w literaturze przedmiotu znane są algorytmy konstrukcyjne dostarczające trochę lepsze rozwiązania niż algorytm *NEH*, ale tak naprawdę są to tylko różne jego modyfikacje.

5. Wnioski

Prezentowany w pracy algorytm *PNEH* jest wysokiej jakości perturbacyjnym algorytmem konstrukcyjnym bazującym na deterministycznym algorytmie *NEH*. W porównaniu do tego drugiego prezentowany algorytm wykazuje następujące korzystne własności:

(i) dostarcza rozwiązań lepszej jakości niż algorytm *NEH*,
(ii) użytkownik sam dobiera kompromis pomiędzy jakością a czasem działania,
(iii) można używać wielokrotnie tego samego algorytmu do otrzymywania różnych rozwiązań startowych i dywersyfikacyjnych,
(iv) użytkownik sam dobiera kompromis pomiędzy siłą zróżnicowania generowanych rozwiązań, a ich jakością.

Z drugiej strony proponowane podejście jest słabsze niż jego deterministyczna postać w następującym aspekcie. Przy skróceniu czasu działania algorytmu *PNEH* do czasu pracy algorytmu *NEH*, prezentowany algorytm dostarcza słabszych rozwiązań.

Dalsze badania algorytmu *PNEH* będą się skupiać nad jego zastosowaniem w generowaniu rozwiązań startowych w algorytmach popraw wykorzystujących techniki wielokrotnego startu. Inny kierunkiem rozwoju prezentowanego algorytmu będzie opracowanie dla niego, technik automatycznego strojenia parametrów perturbacji.

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.: Some efficient heuristic methods for flow shop sequencing. European Journal of Operational Research, 47, 1990, str. 65-74.
3. Nowicki E., Smutnicki C.: New results in the worst-case analysis for flow-shop scheduling. Discrete Applied Mathematics, 46, 1993, str. 21-41.
4. Taillard E.: Benchmarks for basic scheduling problems, European Journal of Operational Research, 64, 1993, str. 278-285.
5. Vaessens R.J.M.: OR library, Internet: http://mscmga.ms.ic.ac.uk.info.html. Files flowshop1.txt, flowshop2.txt.
6. 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 inż. Mariusz MAKUCHOWSKI
Instytut Informatyki, Automatyki i Robotyki
Politechnika Wrocławska
50-372 Wrocław, ul. Janiszewskiego 11/17
tel./fax: (0-71) 320-29-61
e-mail: mariusz.makuchowski@pwr.wroc.pl