AUTOMATYKA • 2009 • Tom 13 • Zeszyt 2

Wojciech Bożejko*, Mieczysław Wodecki**

Metoda analizy minimów lokalnych w rozwiązywaniu pewnych problemów optymalizacji dyskretnej

1. Wstęp

Wiele praktycznych problemów związanych z procesem podejmowania decyzji z dziedziny planowania, zarządzania oraz sterowania sprowadza się do rozwiązania pewnego problemu optymalizacji dyskretnej. Ponieważ zazwyczaj są to problemy NP-trudne, stąd w praktyce do ich rozwiązywania stosuje się niemal wyłącznie metody przybliżone. Obecnie najlepszymi znanymi w literaturze i najczęściej stosowanymi są metaheurystyki: poszukiwania z zabronieniami (*tabu search*), symulowane wyżarzanie (*simulated anealing*) oraz algorytmu genetycznego (*genetic algorithm*). Dla wielu problemów, przede wszystkim tych z regularnymi funkcjami celu, wyznaczone przez nie rozwiązania różnią się zaledwie o kilka procent od optymalnych. Zdecydowanie gorzej radzą one sobie z problemami, dla których nie są obecnie znane własności umożliwiające przegląd pośredni rozwiązań (jak np. własności bloków, Nowicki i Smutnicki [3] oraz Grabowski i Wodecki [2]). Jednym z mankamentów tych heurystyk jest słabe korzystanie z historii obliczeń. Stąd w praktyce bywają duże kłopoty z opuszczeniem minimum lokalnego. W tym celu zazwyczaj stosuje się restart algorytmu.

W pracy przedstawiono metodę konstrukcji algorytmów przybliżonych dla problemów optymalizacyjnych, których rozwiązaniami dopuszczalnymi są permutacje. Jej idea jest opata na następującym przypuszczeniu: jeżeli w różnych permutacjach, będących minimami lokalnymi, na pewnych pozycjach są te same elementy, to w rozwiązaniu optymalnym elementy te będą być może także na tych samych pozycjach. Do wyznaczania minimów lokalnych stosujemy algorytm poszukiwania zstępującego (*descending search*).

Proponowaną metodę konstrukcji algorytmów zastosowano do problememu wyznaczenia kolejności wykonywania zadań na dwóch maszynach, z minimalizacją sumy kosztów opóźnień zadań wykonanych nieterminowo. Zagadnienie to jest w literaturze

* Instytut Informatyki, Automatyki i Robotyki, Politechnika Wrocławska
** Instytut Informatyki, Uniwersytet Wrocławski

oznaczane przez $2|F|\sum w_i T_i$ i należy do klasy problemów *silnie NP-trudnych*. Wyniki obliczeniowe algorytmu porównujemy z wartościami dokładnymi (dla mniejszych rozmiarów danych) oraz wynikami algorytmu NEH [4].

2. Metoda analizy minimów lokalnych

Niech Π będzie zbiorem permutacji elementów ze zbioru $J = \{1, 2, ..., n\}$. Rozpatrujemy problem *optymalizacji dyskretnej* polegający na wyznaczeniu permutacji $\pi^* \in \Pi$ takiej, że

$$F(\pi^*) = \min\{\beta \in \Pi : \beta \in \Pi\},$$

gdzie funkcja celu

$$F : \Pi \to R^+ \cup \{0\}.$$

Wprowadzamy oznaczenia:

π^* – najlepsze wyznaczone przez algorytm rozwiązanie,
η – liczba elementów w populacji,
P^i – populacja w i-tej iteracji algorytmu,
$LocalOpt(\pi)$ – procedura lokalnej optymalizacji, gdzie π jest rozwiązaniem startowym,
LM^i – zbiór minimów lokalnych w i-tej iteracji, $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, ..., \hat{\pi}_\eta\}$, gdzie $\hat{\pi}_j = LocalOpt(\pi_j), \quad \pi_j \in P^i$,
FS^i – zbiór elementów i pozycji ustalonych w i-tej iteracji.

Działanie algorytmu rozpoczyna się od wyznaczenia populacji początkowej P^0 (zazwyczaj jest ona generowana losowo). Za suboptymalne rozwiązanie π^* przyjmujemy najlepszy element z populacji P^0. Niech i będzie numerem iteracji algorytmu. Nowa, i+1 populacja (tj. zbiór permutacji P^{i+1}) jest generowana w następujący sposób. Dla bieżącej populacji P^i wyznacza się zbiór minimów lokalnych LM^i (dla każdego elementu $\pi \in P^i$ wykonując procedurę $LocalOpt(\pi)$, tj. procedurę lokalnej optymalizacji). Następnie ustala się elementy występujące na tych samych pozycjach w wielu minimach lokalnych, tworząc zbiór elementów i pozycji ustalonych FS^{i+1} (ustalanych element ma dodatkowy parametr *wiek*). Każda permutacja nowej populacji P^{i+1} ma ustalone elementy (na ustalonych pozycjach) ze zbioru FS^{i+1}. Na pozostałe (wolne) pozycje są losowo wyznaczane elementy wolne. Jeżeli ponadto w bieżącej populacji istnieje permutacja $\beta \in LM^i$, dla której $F(\beta) < F(\pi^*)$, to za permutację π^* przyjmujemy β. Algorytm kończy działanie po wygenerowaniu z góry ustalonej liczby iteracji.

Z ewolucją populacji związane są dwie następujące funkcje:

$FixSet(LM^i, FS^i)$ – procedura wyznaczająca zbiór elementów i pozycji ustalonych w następnej iteracji algorytmu, $FS^{i+1} = FixSet(LM^i, FS^i)$,
$NewPopul(FS^i)$ – procedura generująca nową populację w następnej iteracji algorytmu, $P^{i+1} = NewPopul(FS^i)$.

Poniżej przedstawiamy schemat algorytmu opartego na analizie minimów lokalnych.

Algorytm analizy minimów lokalnych

wyznaczyć losową populację początkową $P^0 = \{\pi_1, \pi_2, ..., \pi_\eta\}$;
wyznaczyć π^* takie, że
$F(\pi^*) = \min\{F(\pi): \pi \in P^0\}$
$i = 0$; {*numer iteracji*}
$FS^0 \leftarrow \varnothing$ {*zbiór elementów i pozycji ustalonych*}
repeat
Wyznaczyć zbiór minimów lokalnych $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, ..., \hat{\pi}_\eta\}$, gdzie
$\hat{\pi}_j = LocalOpt(\pi_j), \quad \pi_j \in P^i$;
for j:=1 **to** η **do**
if $F(\hat{\pi}_j) < F(\pi^*)$ **then**
$\pi^* \leftarrow \hat{\pi}_j$;
wyznaczyć zbiór elementów i pozycji ustalonych
$FS^{i+1} = FixSet(LM^i, FS^i)$
wygenerować nową populację $P^{i+1}=NewPopul(FS^i)$;
i:=i+1;
until not ***Kryterium zatrzymania***.

Do wyznaczania minimów lokalnych jest stosowany szybki algorytm oparty na metodzie lokalnych popraw. Generalnie, metoda ta polega na iteracyjnym polepszaniu bieżącego rozwiązania poprzez lokalne przeszukiwanie. Rozpoczyna się od pewnego rozwiązania początkowego (startowego) x^0. W każdej iteracji, dla bieżącego rozwiązania x^i wyznacza się jego otoczenie $N(x^i)$ – podzbiór zbioru rozwiązań dopuszczalnych. Otoczenie jest generowane przez ruchy, tj. ustalone przekształcenia rozwiązania x^i. Następnie z otoczenia jest wyznaczany najlepszy element x^{i+1}, który przyjmuje się za bieżące rozwiązanie w następnej iteracji. Liczba iteracji takiego algorytmy nie powinna być większa niż n, a jego złożoność obliczeniowa $O(n^3)$. Dzięki temu, w krótkim czasie, będzie można badać dużą liczbę różnych minimów lokalnych.

Na początku każdej iteracji algorytmu opartego na metodzie analizy minimów, za zbiór elementów i pozycji ustalonych FS^i przyjmujemy FS^{i-1} tj. $FS^i \leftarrow FS^{i-1}$. Po wyznaczeniu minimów lokalnych (procedura *LocalOpt*), na elementach zbioru FS^i są wykonywane następujące operacje:

– zmiana wieku każdego elementu,
– usunięcia najstarszych elementów,
– wstawienia nowych elementów oraz nadanie parametrowi *wiek* wartości 1.

Niech $P^i = \{\pi_1, \pi_2, ..., \pi_\eta\}$ będzie η-elementową populacją w i-tej iteracji algorytmu. Dla każdej permutacji $\pi_j \in P^i$, stosując algorytm poszukiwań lokalnych (procedura

LocalOpt(π_j)), wyznaczamy zbiór minimów lokalnych $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, ..., \hat{\pi}_\eta\}$. Dowolna permutacja $\hat{\pi}_j \in LM^i$ jest postaci:

$$\hat{\pi}_j = (\hat{\pi}_j(1), \hat{\pi}_j(2), ..., \hat{\pi}_j(n)).$$

Przez

$$nr(a,l) = \left|\{\hat{\pi}_j \in LM^i : \ \hat{\pi}_j(l) = a\}\right|,$$

oznaczamy liczbę permutacji ze zbioru LM^i, w których na pozycji l znajduje się element a. Jeżeli $a \in J$ jest elementem wolnym w i-tej iteracji oraz $\frac{nr(a,l)}{\eta} \geq \Phi(i)$ (Φ jest parametrem ustalanym eksperymentalnie), to element a ustalamy na pozycji l.

Następnie wiek każdego ustalonego elementu zwiększamy o 1. Jeżeli $a \in J$ jest elementem ustalonym oraz jego wiek $\chi(a) > \Gamma(i)$ (gdzie $\Gamma(i)$ jest parametrem wyznaczanym eksperymentalnie), to element a zwalniamy (usuwamy) ze zbioru FS^i. Bardziej szczegółowy opis omawianej metody oraz jej parametrów jest zamieszczony w pracy Bożejki i Wodeckiego [1].

3. Dwumaszynowy problem szeregowania zadań

Dany jest zbiór n zadań:

$$J = \{1, 2, 3, ..., n\},$$

które mają być wykonane przy użyciu dwóch maszyn ze zbioru $M = \{1, 2\}$. Zadanie $i \in J$ należy wykonać kolejno na każdej z maszyn, przy czym wykonywanie zadania na maszynie drugiej można rozpocząć dopiero po zakończeniu wykonywania tego zadania na maszynie pierwszej oraz kolejność wykonywania zadań na obu maszynach musi być taka sama. Maszyna, w dowolnej chwili, może wykonywać co najwyżej jedno zadanie oraz wykonywanie zadania nie może być przerwane.

Dla zadania $i \in J$, niech w_i, d_i, p_i^1, p_i^2 będą odpowiednio *wagą funkcji kosztów*, *linią krytyczną* oraz *czasem wykonania zadania* na maszynie *pierwszej i drugiej*. Jeżeli ustalona jest kolejność wykonywania, a C_i jest terminem zakończenia wykonywania i-tego zadania (na drugiej maszynie), to $T_i = \max\{0, C_i - d_i\}$ zwane jest *opóźnieniem*, a $f_i(C_i) = w_i * T_i$ kosztem opóźnienia (wykonywania zadania).

Problem *minimalizacji sumy kosztów opóźnień zadań wykonywanych na dwóch maszynach* (TFST) polega na wyznaczeniu takiej kolejności wykonywania zadań, która minimalizuje sumę kosztów ich wykonywania, czyli minimalizuje $\sum_{k=1}^{n} f_k(C_k)$.

Niech Π będzie zbiorem wszystkich permutacji zbioru zadań J. Każda permutacja $\pi \in J$ wyznacza jednoznacznie pewne uszeregowanie zadań ze zbioru J – kolejność wykonywania, jednakową na obu maszynach. W uszeregowaniu tym zadanie $\pi(i) \in J$ jest

wykonywane, na każdej maszynie, jako *i*-te w kolejności ($1 \le i \le n$). Oznaczmy przez $C^{j}_{\pi(i)}$ termin zakończenia wykonywania zadania $\pi(i)$ na maszynie j (j = 1, 2). Wówczas $C_{\pi(i)} = C^{2}_{\pi(i)}$ jest terminem całkowitego zakończenia wykonywania tego zadania w permutacji, a termin zakończenia wykonywania ostatniego zadania, czyli $C_{\pi(i)}$ nazywamy *długością permutacji* π.

Oznaczmy przez:

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

wagę permutacji $\pi \in \Pi$.

Łatwo zauważyć, że rozważane zagadnienie TFST sprowadza się do wyznaczenia permutacji optymalnej (o najmniejszej wadze) w zbiorze wszystkich permutacji $\pi \in \Pi$. Do jego rozwiązania będzie stosowany algorytm oparty na metodzie analizy minimów lokalnych.

4. Wyznaczanie minimów lokalnych

Do wyznaczania minimów lokalnych zastosowano algorytm zstępujący oparty na metodzie lokalnych popraw. Algorytm rozpoczyna obliczenia od pewnego rozwiązania początkowego (startowego). W każdej iteracji, dla bieżącego rozwiązania π, wyznacza się jego otoczenie $N(\pi)$ – podzbiór zbioru rozwiązań dopuszczalnych. Otoczenie jest generowane przez ruchy, tj. ustalone przekształcenia rozwiązania π. Następnie, z otoczenia jest wyznaczany najlepszy element β, który przyjmuje się za bieżące rozwiązanie w następnej iteracji.

Algorytm zstępujący
Niech $\beta \in \Pi$ będzie rozwiązaniem startowym;
repeat
$\pi \leftarrow \beta$;
Wyznaczyć permutację $\beta \in N(\pi)$ taką, że $F(\beta) = \min\{F(\delta): \delta \in \Pi\}$;
until $F(\pi) > F(\beta)$.

Poniżej zamieszczamy krótki opis najważniejszych elementów algorytmu.

Ruch i otoczenie

W każdej iteracji algorytmu zstępującego jest wyznaczane otoczenie – podzbiór zbioru rozwiązań dopuszczalnych. Jest ono generowany przez ruchy – przekształcenia określone na zbiorze wszystkich permutacji.

Niech k i l ($k < l$) będzie parą pozycji w permutacji:

$$\pi = (\pi(1), \pi(2), \ldots, \pi(k-1), \pi(k), \pi(k+1), \ldots, \pi(l-1), \pi(l), \pi(l+1), \ldots, \pi(n)).$$

Ruch typu *wstaw* (*insert*) oznaczany przez i_l^k, polega na przestawieniu zadania $\pi(k)$ z pozycji k na pozycję l. Generuje on permutację $i_l^k(\pi) = \pi_l^k \in \Pi$, gdzie:

$$\pi_l^k(i) = \begin{cases} \pi(k), & i = l, \\ \pi(i), & l < i < k, \\ \pi(i+1), & i = k, k+1, \ldots, l-1. \end{cases}$$

Ruch i_l^k w przypadku, gdy $k > l$ definiujemy podobnie. Wszystkich takich ruchów (moc otoczenia) wynosi $n(n-1)$.

5. Wyniki obliczeniowe

Przedstawiony w poprzednim rozdziale algorytm został zaprogramowany w języku C++ i był testowane na wielu losowo generowanych przykładach. Sposób generowania danych jest zmodyfikowaną wersją, przedstawionej w pracy Potts i Van Wassenhove [5], metody dla problemu z jedną maszyną. Każdy przykład składa się z n czwórek (w_i, p_i^1, p_i^2, d_i), gdzie n jest liczbą zadań, a w_i, p_i^1, p_i^2, d_i są odpowiednio: wagą funkcji kosztów, czasem wykonywania na pierwszej i drugiej maszynie oraz liną krytyczną dla i-tego zadania. Waga funkcji kosztów oraz czasy wykonywania są realizacją zmiennych losowych o rozkładzie jednostajnym odpowiednio na przedziale [1,10] oraz [1,100]. Wartości linii krytycznych są zależne od wcześniej wygenerowanych czasów wykonywania oraz dwóch parametrów *RDD* (*relative of due dates*) oraz *TF* (*average tardiness factor*), które przyjmują wartości ze zbioru {0.2, 0.4, 0.6, 0.8, 1.0}. Linie krytyczne są realizacją zmiennej losowej o rozkładzie jednostajnym na przedziale $[C_{\max}(1 - TF - RDD/2), C_{\max}(1 - TF + RDD/2)]$, gdzie $C_{\max}$ jest maksymalnym czasem wykonywania n zadań, z wcześniej wygenerowanymi czasami wykonywania, tj. rozwiązaniem zagadnienia minimalizacji czasu wykonywania wszystkich zadań $(2|F|C_{\max})$. Wartości parametrów *TF* i *RDD* determinują trudność generowanych przykładów. Wzrost pierwszego współczynnika powoduje zmniejszanie się wartości linii krytycznych, a zmniejszanie drugiego – zawężanie przedziału, z którego są one losowane. Dla każdej pary parametrów *RDD* i *TF* (w sumie 25 par) generowano 5 przykładów, czyli dla ustalonej liczby zadań generowano 125 przykładów. Uzyskane wyniki przedstawiono w tabeli 1. W kolumnie drugiej i trzeciej, dla liczby zadań n = 20, 30, 40 i 50, zamieszczono średni błąd względny algorytmów przybliżonych NEH i AML w stosunku do wartości optymalnych (wyznaczonych przez odpowiednio zaadaptowany algorytm optymalny z pracy Wodeckiego [6]). Natomiast w kolumnie czwartej, przedstawiono średni błąd względny rozwiązań algorytmu AML w stosunku do wartości wyznaczonych przez NEH (tj. poprawę tych rozwiązań).

Tabela 1
Średni procentowy błąd względny algorytmów

Liczba zadań n	Średni procentowy błąd względny		Średnia procentowa względna poprawa*
	Algorytmu NEH	Algorytmu AML	
20	2.11%	0.00%	–2.11%
30	2.94%	0.05%	–3.04%
40	4.01%	0.11%	–4.38%
50	7.36%	0.48%	–13.23%
100	–	–	–18.72%
150	–	–	–23.57%
200	–	–	–29.13%

* Średnia procentowa względna poprawa rozwiązania wyznaczonego przez NEH.

Na podstawie zamieszczonych wyników można stwierdzić, że algorytm oparty na metodzie analizy minimów lokalnych AML wyznacza bardzo dobre rozwiązania. Pomimo że do wyznaczania minimów lokalnych zastosowano bardzo prosty algorytm zstępujący, otrzymane wyniki (dla liczby zadań n=20,30,40 i 50) niewiele różnią się od optymalnych. Jednocześnie są one znacznie lepsze od tych wyznaczonych przez algorytm NEH. Z kolei dla większych rozmiarów danych (z powodu braku wartości optymalnych) porównano jedynie oba algorytmy przybliżone. W tym przypadku algorytm AML znacznie poprawił rozwiązania wyznaczone przez NEH. Szczególnie jest to widoczne dla dużych przykładów.

6. Podsumowanie

W pracy przedstawono metodę rozwiązywania permutacyjnych problemów optymalizacyjnych opartą na analizie minimów lokalnych. Zastosowano ją w konstrukcji algorytmu rozwiązywania NP-trudnego dwumaszynowego problemu szeregowania zadań z minimalizacją sumy kosztów opóźnień. Przeprowadzono eksperymenty obliczeniowe, które wykazały dużą efektywność opisanej metody.

Literatura

[1] Bożejko W., Wodecki M., *A hybrid evolutionary algorithm for the permutation optimization problem*. ISDA 05, IEEE Computer Society, 2005, 326–331.

[2] Grabowski J., Wodecki M., *A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion*. Computers & Operations Research, 31, 2004, 1891–1909.

[3] Nowicki E., Smutnicki C., *A fast tabu search algorithm for the permutation flow-shop problem*. European Journal of Operational Research, 91, 1996, 160–175.

[4] Navaz M., Encore E., Ham I., *A heuristic algorithm for the m-machine n-job flow-shop sequencing problem*. Omega, 11, 1983, 91–95.
[5] Potts C.N., Van Wassenhove L.N., *A Branch and Bound Algorithm for the Total Weighted Tardiness Problem*. Operations Research, 33, 1985, 177–181.
[6] Wodecki M., *A branch-and-bound parallel algorithm for single-machine total weighted tardiness problem*. Int. J. Adv. Manuf. Technol., 37, 2008, 996–1004.