Wojciech BOŻEJKO, Dominik NIKREWICZ,
Mariusz UCHROŃSKI, Mieczysław WODECKI
Politechnika Wrocławska

EFEKTYWNOŚĆ MODELI PROGRAMOWANIA LINIOWEGO DLA PROBLEMU MINIMALIZACJI CAŁKOWITEGO WAŻONEGO SPÓŹNIENIA NA JEDNEJ MASZYNIE

Streszczenie. Tematem pracy jest ocena efektywności modeli programowania liniowego dla jednomaszynowego problemu szeregowania zadań z mimimalizacją ważonej sumy spóźnień. Badania efektywności modeli zostały przeprowadzone w środowisku obliczeń równoległych z wykorzystaniem solvera Gurobi.

EFFICIENCY OF LINEAR PROGRAMMING MODELS FOR MINIMIZATION OF THE TOTAL WEIGHTED TARDINESS ON A SINGLE MACHINE

Summary. The subject of the work is efficiency evaluation of linear programming models for single–machine total weighted tardiness scheduling problem. The evaluation on the effectiveness of the models were carried out in a parallel computing environment using the Gurobi solver.

1. Wprowadzenie

W pracy jest rozpatrywany problem szeregowania zadań niepodzielnych i niezależnych wykonywanych na jednej maszynie. Z każdym zadaniem jest związany czas wykonania, żądany termin zakończenia oraz współczynnik kary za spóźnienie wykonywania zadania. Należy wyznaczyć kolejność zadań, która minimalizuje sumę kar. W literaturze jest on oznaczany przez $1||\sum w_iT_i$ i należy do klasy problemów NP-trudnych.

Problemy jednomaszynowe są powszechnie uważane za najprostsze zagadnienia teorii szeregowania zadań pomimo, że zdecydowana większość z nich należy do klasy problemów NP-trudnych. W praktycznych zastosowaniach umożliwiają one modelowanie i analizę pojedynczych stanowisk roboczych w złożonych systemach produkcyjnych. Prostota sformułowania, nieskomplikowanie modeli oraz łatwość implementacji algorytmów powodują, że problemy te należą do najczęściej badanych i analizowanych. Uzyskane wyniki są inspiracją do rozwoju nowych podejść i algorytmów rozwiązywania znacznie trudniejszych - wielomaszynowych problemów szeregowania zadań.

Rozpatrywany w pracy problem, pomimo prostoty sformułowania, jest od prawie pięćdziesięciu lat obiektem intensywnych badań. Sformułowano wiele jego własności (np. własności eliminacyjne bloków, Wodecki [7]), które zastosowano zarówno w konstrukcjach algorytmów dokładnych, jak i przybliżonych. Z kolei w pracy Congrama i in. [6] przedstawiono bardzo ciekawe otoczenie generowane przez złożenia niezależnych ruchów typu zamień (*ang. dynasearch swap*). Pomino, że otoczenie to ma wykładni-

czą liczbę elementów, to jest przeszukiwane (wyznaczany element minimalny) w czasie wielomianowym. Zostało ono z powodzeniem zastosowane w wielu algorytmach metaheurystycznych.

Obecnie do rozwiązywania wielu problemów optymalizacji dyskretnej jest powszechnie stosowany solver Gurobi. Problemy te, jako zadania optymalizacyjne, można formułować na wiele sposobów. W pracy badamy zależność pomiędzy różnymi sformułowaniami problemu $1||\sum w_iT_i$, jako zadania optymalizacyjnego, na jakość (błąd względny) wyznaczanych przez Gurobi rozwiązań, w środowisku obliczeń równoległych.

2. Problem szeregowania zadań na jednej maszynie

Jednomaszynowy problem szeregowania zadań na jednej maszynie z ważoną sumą spóźnień $1||\sum w_iT_i$ można sformułować następująco. Niech $\mathcal{J} = \{1, 2, \ldots, n\}$ będzie zbiorem zadań, które należy wykonać bez przerywania na maszynie. Dla każdego zadania $i \in \mathcal{J}$ wprowadzamy następujące oznaczenia:

- p_i – czas wykonania,
- d_i – żądany najpóźniejszy termin zakończenia,
- w_i – współczynnik funkcji kary za zbyt późne wykonanie (spóźnienie).

Niech $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ będzie permutacją elementów z $\mathcal{J}$ (kolejnością wykonywania zadań), a Φ zbiorem wszystkich takich permutacji. Wtedy

$$C_{\pi(i)} = \sum_{j=1}^{i} p_{\pi(j)}, \tag{1}$$

jest momentem zakończenia zadania $\pi(i) \in \mathcal{J}$. Jeżeli $C_{\pi(i)} \leqslant d_{\pi(i)}$, to zadanie jest *terminowe* (wykonane na czas), a w przeciwnym wypadku *spóźnione*. Opóźnienie zadania:

$$T_{\pi(i)} = \max\{0, C_{\pi(i)} - d_{\pi(i)}\}. \tag{2}$$

Koszt wykonania wszystkich zadań:

$$F(\pi) = \sum_{i=1}^{n} w_{\pi(i)}T_{\pi(i)}. \tag{3}$$

Rozwiązanie problemu jednomaszynowego z ważoną sumą spóźnień polega na znalezieniu takiej permutacji $\pi^* \in \Phi$, dla której sumaryczny koszt wykonania wszystkich zadań jest najmniejszy

$$F(\pi^*) = \min\left\{\mathcal{F}(\pi) :\ \pi \in \Phi\right\}. \tag{4}$$

W dalszej części pracy przedstawiamy dwa modele binarno-całkowitoliczbowe rozpatrywanego problemu szeregowania jako zadania programowania matematycznego.

2.1. Model pierwszy M1

Rozpatrywany w pracy problem przedstawiamy w postaci zadania mieszanego ze zmiennymi binarnymi oraz calkowitoliczbowymi.

Problem $1||\sum w_iT_i$, model M1.

Wyznaczyć:

$$\min_{x\in\{0,1\}^{n^2}} \sum_{i=1}^{n}\sum_{j=1}^{n} w_j x_{ij} T_i \tag{5}$$

przy ograniczeniach:

$$T_k \geqslant \sum_{i=1}^{k}\sum_{j=1}^{n} p_j x_{ij} - \sum_{j=1}^{n} x_{kj} d_j, \quad k = 1, 2, \ldots, n, \tag{6}$$

$$T_k \geqslant 0, \tag{7}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, 2, \ldots, n, \tag{8}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, \ldots, n, \tag{9}$$

$$x_{ij} \in \{0, 1\}, \quad i, j = 1, 2, \ldots, n, \tag{10}$$

$$T_i \in \mathbb{Z}, \quad i = 1, 2, \ldots, n, \tag{11}$$

gdzie x_{ij} jest zmianną binarną równą 1, jeśli na pozycji i jest zadania j, a 0 w przeciwnym wypadku. Spóźnienie k-tego zadania – definicja (2). W ograniczeniu (6) pierwsza suma jest momentem zakończenia k-tego zadania, a druga wyznacza jego najpóźniejszy termin zakończenia d_k. Oba ograniczenia (6) i (7) są równoważne $T_k \geqslant \max\{0, C_k - d_k\}$, co przy minimalizacji funkcji celu prowadzi docelowo do równości $T_k = \max\{0, C_k - d_k\}$, tożsamej z definicją spóźnienia. Liczba zmiennych binarnych w przedstawionym modelu problemu wynosi n^2,

2.2. Model drugi M2

Problem $1||\sum w_i T_i$**, model M2**.
Zminimalizować:

$$\sum_{i} w_i T_i \tag{12}$$

przy ograniczeniach:

$$T_j - S_j \geqslant p_j - d_j \quad j = 1, \ldots, n, \tag{13}$$

$$T_j \geqslant 0 \quad j = 1, \ldots, n, \tag{14}$$

$$S_i \geqslant S_j + p_j - K \cdot x_{ij} \quad j < i,\ j, k = 1, \ldots, n, \tag{15}$$

$$S_j \geqslant S_i + p_i - K \cdot (1 - x_{ij}) \quad j < i,\ j, k = 1, \ldots, n, \tag{16}$$

$$x_{ij} \in \{0, 1\}, \quad i, j = 1, 2, \ldots, n. \tag{17}$$

$$S_j \geqslant S_0 \quad j = 1, 2, \ldots, n, \tag{18}$$

$$T_j, S_j \in \mathbb{Z}, \quad j = 1, 2, \ldots, n, \tag{19}$$

gdzie S_i i T_i są zmiennymi całkowitoliczbowymi, a $x_{i,j}$ jest zmienną binarną równą 1 jeżeli zadanie i poprzedza zadanie j, a 0 w przeciwnym wypadku. Przez K oznaczamy dużą liczbę całkowitą. Ograniczenia 15 i 16 realizują nienakładanie się zadań. Liczba zmiennych binarnych wynosi n^2, a zmiennych całkowitych $2n$.

3. Metoda rozwiązania

Przedstawione modele matematyczne problemu zostały zaimplementowane w języku programowania Python zgodnie z API programistycznym dedykownaym dla Gurobi [3], a następnie rozwiązane przy użyciu solvera Gurobi [1] w wersji 11.0.1. Gurobi to zaawansowany solver optymalizacyjny dla programowania matematycznego, ceniony za swoją efektywność i wszechstronność. Dzięki zoptymalizowanym algorytmom, potrafi w krótkim czasie rozwiązywać szeroki zakres problemów optymalizacyjnych, od programowania liniowego po kwadratowe. Gurobi umożliwia sformułowanie rozwiązywanych problemów w postaci [2]:

- programowania liniowego (**LP** – *ang. Linear Programming*),
- mieszanego programowania liniowego (**MILP** – *ang. Mixed-Integer Linear Programming*),
- programowania kwadratowego (**QP** – *ang. Quadratic Programming*),
- mieszanego programowania kwadratowego (**MIQP** – *ang. Mixed-Integer Quadratic Programming*),
- programowania z ograniczeniami kwadratowymi (**QCP** – *ang. Quadratic Constrained Programming*),
- mieszanego programowania z ograniczeniami kwadratowymi (**MIQCP** – *ang. Mixed-Integer Quadratic Constrained Programming*).

Gurobi ma szerokie zastosowanie w wielu dziedzinach gospodarki, między innymi w: logistyce, finansach, systemach sterowania produkcją i analizie danych umożliwiając optymalizację procesów a w konsekwencji, obniżkę kosztów funkcjonowania.

4. Eksperymenty obliczeniowe

Eksperymenty obliczeniowe wykonano na wielu przykładach o różnej trudności ich rozwiązania. Instancje testowe o rozmiarach $n = 40, 50, 100$ zostały pobrane z pliku zamieszczonego na stronie [4]. Natomiast, pozostałe instancje o rozmiarach $n = 200$ zostały wygenerowane losowo, zgodnie z rozkładem jednostajnym, z następujących przedziałów:

- p_i: przedział całkowitoliczbowy $[1, 100]$,
- w_i: przedział całkowitoliczbowy $[1, 10]$,
- d_i: przedział $[P(1 - T_F - R_{DD}/2), P(1 - T_F + R_{DD}/2)]$,

gdzie $P = \sum_{i=1}^{v} p_i$, a ponadto $R_{DD} = 0.2, 0.4, 0.6, 0.8, 1.0$ (względny zakres żądanych terminów zakończenia) oraz $T_F = 0.2, 0.4, 0.6, 0.8, 1.0$ (średni współczynnik spóźnienia). Dla każdej z 25 par wartości R_{DD} i T_F wygenerowanych zostało pięć instancji testowych. W sumie więc wygenerowanych zostało 125 instancji problemu o rozmiarze $n = 200$. Za wartości referencyjne dla przykładów o rozmiarze $n = 200$ przyjęto wartości rozwiązań wyznaczonych przez algorytm konstrukcyjny AU – Apparent Urgency (Potts [5]). Jakość wyznaczonych rozwiązań była oceniana na podstawie wartości procentowego błędu względnego (*ang. Percentage Relative Deviation*, PRD).

Eksperymenty obliczeniowe zostały przeprowadzone na serwerze wyposażonym w 64 GB pamięci RAM oraz dwa procesory Intel(R) Xeon(R) CPU E5-2670 v3. Obliczenia zostały wykonane dla różnej liczby procesorów $p = 1, 2, 4, 8, 16$. Dla przykładów z liczbą zadań $n = 40, 50, 100$ ustalono czas obliczeń $t = 1800s$. Natomiast, dla największych przykładów $n = 200$ czas ten zwiększono do $t = 3600s$. Zwiększenie czasu obliczeń było konieczne, bowiem przy ograniczeniu czasu obliczeń do $t = 1800s$, dla żadnego przykładu, nie otrzymano rozwiązania dopuszczalnego. Pomimo tego zabiegu, dla modelu M2, nie otrzymano rozwiązań dopuszczalnych dla 30 ze 125 instancji.

Wyniki przeprowadzonych eksperymentów obliczeniowych zostały przedsatwione w tabeli 1. Tabela zawiera średnie wartości błędu względnego PRD, dla różnych rozmiarów instancji $n = 40, 50, 100, 200$ oraz różnej liczby procesorów $p = 1, 2, 4, 8, 16$. Generalnie, dla wszystkich rozmiarów (z wyjątkiem $n = 200$) wraz ze wzrostem liczby procesorów maleje wartość błędu PRD (rozwiązania są bliższe optymalnym). W przypadku mniejszych rozmiarów danych $n = 40, 50, 100,$ na podstawie uzyskanych wyników, nie można jednoznacznie stwierdzićnie, który z modeli umożliwia otszymanie lepszych rozwiązań. Jednak, dla największych przykładów $n = 200$, korzystając z modelu M2, nie otrzymano rozwiązań dopuszczalnych. Reasumukąc, biorąc pod uwagę zarówno jakość wyznaczanych rozwiązań jak i liczbę zmiennych, efektywniejszy jest model M1 rozpatrywanego w pracy jednomaszynowego problemu szeregowania zadań.

Tabela 1

Wartości PRD dla różnej liczby procesorów dla modelu M1 i M2

$p\backslash n$	40		50		100		200	
	M1	M2	M1	M2	M1	M2	M1	M2
1	0,0162	0,0237	0,0354	0,3693	0,2803	0,1331	0,3779	-
2	0,0129	0,0176	0,0359	0,0198	0,2628	0,1042	0,8122	-
4	0,0067	0,0105	0,0192	0,0177	0,1895	0,0811	0,7422	-
8	0,0051	0,0093	0,0174	0,0137	0,1878	0,0738	0,3553	-
16	0,0035	0,0058	0,0167	0,0158	0,1591	0,0895	0,4831	-

5. Wnioski i uwagi

W pracy przedstawiono dwa modele matematyczne dla jednomaszynowego problemu szeregowania zadań z ważoną sumą spóźnień. Rozpatrywane modele zostały zaprogramowane oraz rozwiązane z użyciem solvera Gurobi w środowisku obliczeń równoległych. Uzyskane wyniki wskazują na niewielką przewagę (dla mniejszych przykładów danych) modelu M2 nad M1 w kontekście jakości uzyskiwanych rozwiązań w środowisku obliczeń równoległych. Natomiast, model M1 pozwala na rozwiązywanie problemów o znacznie większych rozmiarach.

LITERATURA

1. Gurobi Optimizer, https://www.gurobi.com/solutions/gurobi-optimizer/ (dostęp 10.06.2024).

2. Gurobi - Problem Types, https://www.gurobi.com/features/gurobi-optimizer-delivers-support-for-all-major-problem-types/ (dostęp 10.06.2024).
3. Gurobi Python interface, https://www.gurobi.com/documentation/current/refman/py_python_api_overview.html (dostęp 10.06.2024).
4. OR-Library, Weighted tardiness, http://people.brunel.ac.uk/ mastjjb/jeb/orlib/wtinfo.html (dostęp 10.06.2024).
5. Potts C.N., Van Wassenhove L.N.: Single Machine Tardiness Sequencing Heuristics, *IIE Transactions*, 1991, vol. 23, p. 346-354.
6. Richard K., Congram R.K., Potts C.N., van de Velde S.L.: An Iterated Dynasearch Algorithm for the Single-Machine Total Weighted Tardiness Scheduling Problem, INFORMS Journal on Computing, 2002, Vol. 14(1), p. 52-67.
7. Wodecki M.: A Branch-and-Bound Parallel Algorithm for Single-Machine Total Weighted Tardiness Problem, *Advanced Manufacturing Technology*, 2008, p. 996-1004.