Metodyoptymalizacji (2015/16) – Laboratorium+Ćwiczenia
(TEMAT 3)

W trakcie tych zajęć będą realizowane projekty zespołowe (2 osobowe). W ramach projektu należy wykonać:

1. pełny opis wraz ze specyfikację,
2. zbudować model matematyczny,
3. skonstruować algorytm,
4. zaimplementować algorytm,
5. wykonać eksperymenty obliczeniowe.
6. przygotować multimadialną prezentację.

Punkty 1-3 oraz 5 będą realizowane w ramach ćwiczeń, a pozostałe – w ramach laboratorium. Warunkiem zaliczenia jest wykonanie projektu.

Terminy referatów (poniedziałek)

Prezentacje:

1. **14.03.2016 (punkty 1-3, Tematy 1,2,3,4)**
2. **21.03.2016 (punkty 1-3, Tematy 5,6,7,8)**
3. **04.04.2016 (punkty 1-3, pozostałe Tematy)**
4. **11.04.2016 (dodatkowy)**

Zaliczenia:

1. **23.05.2016 (punkty 4-6, Tematy 1,2,3,4)**
2. **30.05.2016 (punkty 4-6, Tematy 5,6,7,8)**
3. **06.06.2016 (punkty 4-6, pozostałe Tematy)**
4. **13.06.2016 (dodatkowy)**

Zad. 2.

Wyznaczyć cykl Hamiltona w grafie (zastosować algorytm z powracaniem, W. Lipski, „Kombinatoryka dla informatyków", str. 112).

ALGORYTMY LOKALNYCH POSZUKIWAŃ Z OTOCZENIAMI O WYKŁADNICZEJ LICZBIE ELEMENTÓW

Wojciech Bożejko, Mieczysław Wodecki

Streszczenie. W pracy rozpatrujemy jeden z klasycznych problemów szeregowania zadań na jednej maszynie, w którym kryterium optymalności jest suma kosztów zadań spóźnionych. Należy on do klasy problemów silnie NP-trudnych. Przedstawiamy algorytm poszukiwania zstępującego, w którym otoczenie, generowane przez serię niezależnych ruchów typu „zamień" (*swap*), ma wykładniczą liczbę elementów. Do jego przeglądania stosujemy algorytm oparty na metodzie programowania dynamicznego o złożoności obliczeniowej $O(n^3)$.

Słowa kluczowe: Szeregowanie zadań, linie krytyczne, koszt spóźnienia, metoda „dynasearch".

1. Wstęp

W pracy zajmujemy się problemem polegającym na uszeregowaniu n zadań na jednej maszynie. Maszyna ta, w dowolnej chwili, może wykonywać co najwyżej jedno zadanie. Dla zadania i, niech p_i, w_i, d_i będą odpowiednio: *czasem wykonania, wagą funkcji kosztów* oraz wymaganym terminem zakończenia (*linią krytyczną*). Jeżeli ustalona jest kolejność wykonywania zadań i C_i jest *terminem wykonania* zadania i ($i=1,2,...,n$), to *spóźnienie* $U_i=1$, gdy $C_i>d_i$ lub 0, w przeciwnym przypadku. Wielkość w_iU_i nazywamy *kosztem* spóźnienia (wykonania zadania).

Jednomaszynowy problemu minimalizacji sumy kosztów zadań spóźnionych (w skrócie *MSKS*) polega na wyznaczeniu takiej kolejności wykonywania zadań, która minimalizuje sumę kosztów spóźnień, tj. $\sum_{i=1}^{n} w_iU_i$. W literaturze jest on oznaczany przez $1||\sum w_iU_i$ i należy do klasy problemów NP-trudnych (Karp [7]). Jest to jeden z klasycznych problemów szeregowania zadań, stąd w literaturze rozpatrywanych jest wiele jego wariantów o wielomianowej złożoności obliczeniowej.

Dla problemu $1|p_i=1|\sum w_iU_i$ (wszystkie czasy wykonywania są jednakowe) istnieje algorytm o złożoności $O(n)$ (Monma [12]). Podobnie, gdy jednakowe są współczynniki funkcji kosztów w_i (problem $1||\sum U_i$ - algorytm Moore'a [13]) o złożoności $O(n \ln n)$). Problem $1|\, p_i < p_j \Rightarrow w_i \geq w_j\,|\sum w_iU_i$ ma wielomianową złożoność obliczeniową i do jego rozwiązywania Lawler [9] zaadoptował algorytm Moore'a.

Problem z najwcześniejszymi możliwymi terminami rozpoczęcia r_i ($1|r_i|\sum U_i$ - bez wag funkcji kosztów) jest silnie NP-trudny (Kise, Baraki i Mine [8]). W pracy tej zamieszczono także algorytm o złożoności $O(n^2)$, dla problemu $1|\, r_i < r_j \Rightarrow d_i \leq d_j\,|\sum_j U_j$.

Jeżeli na zbiorze zadań określimy relację częściowego porządku, wówczas problem jest silnie NP-trudny nawet, dla jednostkowych czasów wykonywania zadań (Garey i Johnson

[6]). Lenstra i Rinnoy Kan [11] udowodnili, że jest on także silnie NP-trudny, jeżeli relacja częściowego porządku jest sumą niezależnych łańcuchów.

W literaturze opublikowane są algorytmy wyznaczania rozwiązania optymalnego dla rozpatrywanego w tej pracy problemu minimalizacji sumy kosztów spóźnień. Są to algorytmy oparte na metodzie programowania dynamicznego (Lawler i Moore [10] złożoność $O(n\min\{\sum_j p_j, \max_j\{d_j\}\})$, Sahni [16] złożoność $O(n\min\{\sum_j p_j, \sum_j w_j, \max_j\{d_j\}\})$ oraz podziału i ograniczeń (Villarreal i Bulfin [18], Potts i van Wassenhowe [14], [15] oraz Bożejko i Wodecki [2] - algorytm wieloprocesorowy). Ponieważ algorytmy te mają wykładnicze zapotrzebowanie na pamięć i czas obliczeń, dlatego mogą być stosowane jedynie do rozwiązywania przykładów o niewielkich rozmiarach (kilkadziesiąt zadań). Do rozwiązywania NP-trudnych problemów optymalizacji kombinatorycznej stosuje się obecnie niemal wyłącznie algorytmy przybliżone. Wyznaczane przez te algorytmy rozwiązania są z praktycznego punktu widzenia w pełni zadawalające (często różnią się od najlepszych rozwiązań o mniej niż 1%). Najlepsze z nich należą do grupy metod poszukiwań lokalnych (local search), których działanie sprowadza się do bezpośredniego przeglądania pewnego obszaru zbioru rozwiązań dopuszczalnych. Mechanizm ten polega na generowaniu, z bieżącego rozwiązania, podzbioru rozwiązań (otoczenia), z którego jest wybierany najlepszy element. W następnej iteracji jest on kolejnym bieżącym rozwiązaniem. Postępowanie to jest zazwyczaj kontynuowane aż do osiągnięcia minimum lokalnego. Otoczenie – jego wielkość, sposób generowania i przeglądania ma zasadniczy wpływ na złożoność obliczeniową, czas działania oraz jakość wyznaczanych przez algorytm rozwiązań. W wielu problemach szeregowania rozwiązaniami dopuszczalnymi są n elementowe permutacje. Stosowane są w nich zazwyczaj otoczenia generowane przez ruchy typu wstaw (*insert*) lub zamień (*swap*) i mają odpowiednio $(n-1)^2$ lub $n(n-1)/2$ elementów. Dodatkowe kryteria eliminacyjne (np. dla problemu $1||\sum w_iT_i$) zmniejszają ich liczbę, ale praktycznie rząd ich wielkości pozostaje taki sam. Dlatego, algorytmy przeglądania tych otoczeń mają złożoność obliczeniową co najmniej $O(n^2)$.

W algorytmach rozwiązywania problemu komiwojażera są z powodzeniem stosowane otoczenia o wykładniczej liczbie elementów. Dokładnie są one opisane w pracy przeglądowej Ahuja i in.[1]. W algorytmie rozwiązywania problemu $1||\sum w_iT_i$ (Congram, Potts i Van Wassenhowe [4]) jest stosowane otoczenie zawierające wykładniczą względem n liczbę elementów, $2^{n-1}-1$. Każdy element jest generowany przez wykonanie serii niezależnych ruchów typu zamień. Przedstawiono także algorytm (zwany algorytmem „dynasearch"), oparty na metodzie programowania dynamicznego, wyznaczający element optymalny z tego otoczenia w czasie $O(n^3)$ i wymagający $O(n)$ pamięci.

W niniejszej pracy przedstawiamy algorytm, oparty na metodzie poszukiwania zstępującego (descending search), rozwiązywania problemu szeregowania $1||\sum w_iU_i$, w konstrukcji którego wykorzystano metodę „dynasearch" do generowania i przeglądania otoczenia. Zastosowano także kryteria eliminacyjne znacznie przyśpieszające obliczenia.. Do wyznaczania rozwiązań startowych stosujemy algorytm konstrukcyjny META opisany w pracy [14]. Ponieważ, dla rozważanego problemu nie istnieją rozwiązania referencyjne, dlatego porównujemy otrzymane wyniki (dla losowo generowanych przykładów) z wynikami algorytmu konstrukcyjnego oraz odpowiednio zaadaptowanego algorytmu tabu search przedstawionego w pracy [3] dla rozwiązywania problemu minimalizacji sumy kosztów spóźnień $1||\sum w_iT_i$.

2. Definicje oraz własności problemu.

Niech $N=\{1,2, \ldots, n\}$ będzie zbiorem zadań, a $S(n)$ zbiorem wszystkich permutacji elementów z N. Dla permutacji $\pi \in S(n)$ przez $C_{\pi(i)}$, $(C_{\pi(i)} = \sum_{j=1}^{i} p_{\pi(j)})$ oznaczamy czas zakończenia wykonywania i–tego zadania w permutacji π (tj. zadania $\pi(i)$, jeżeli zadania są wykonywane zgodnie z kolejnością ich występowania w π). Rozpatrywany problem *MSKS* polega wyznaczeniu permutacji optymalnej π^* takiej, że

$$F(\pi^*) = \min\{F(\delta): \ \delta \in S(n)\},$$

gdzie funkcja celu

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

a $f_{\pi(i)}(C_{\pi(i)}) = w_{\pi(i)} U_{\pi(i)}$ jest kosztem spóźnienia wykonywania zadania $\pi(i) \in N$.

Na zbiorze zadań N wprowadzamy relację częściowego porządku R. Zadanie i jest w relacji z zadaniem j, tj. $i \mathrm{R}\, j$ wtedy i tylko wtedy, gdy istnieje permutacja optymalna, w którym zadanie i poprzedza j. Niech $\Gamma_i^+ = \{j \in N: \ i \mathrm{R}\, j\}$ oraz $\Gamma_i^- = \{j \in N: \ j \mathrm{R}\, i\}$ będą odpowiednio zbiorami następników oraz poprzedników zadania i w relacji R. Dwa zadania są w relacji R, jeżeli spełniają jeden z warunków poniższego twierdzenia, zwanych kryteriami eliminacyjnymi.

Twierdzenie 1. Jeżeli dla pary zadań $i, j \in N$ spełniony jest jeden z warunków:

(a) $p_i \le p_j$, $w_i \ge w_j$, $d_i \le d_j$, lub
(b) $p_i \le p_j$, $w_i \ge w_j$, $d_i \le \sum_{l \in \Gamma_j^-} p_l + p_j$, lub
(c) $w_i \ge w_j$, $d_i \le d_j$, $d_i \ge \sum_{l \in N \setminus \Gamma_j^+} p_l + p_j$,

to istnieje permutacja optymalna, w której zadanie i poprzedza j.

Dowód. Warunek (a) pochodzi z pracy Shwimera [17], a (b) i (c) są uogólnionymi wersjami Twierdzenia 1 i 2 z pracy Emmonsa [5]. ■

Jeżeli dla pary zadań $i, j \in N$ zachodzi relacja $i \mathrm{R}\, j$, to $i \in \Gamma_j^-$ oraz $j \in \Gamma_i^+$. Aby nie powstał cykl, po ustaleniu każdej nowej relacji (pomiędzy parą zadań), wykonujemy tranzytywne domknięcie modyfikując zbiory:

$$\forall l \in \{i\} \cup \Gamma_i^-, \ \Gamma_l^+ \leftarrow \Gamma_l^+ \cup \{j\} \cup \Gamma_j^+,$$
$$\forall l \in \{j\} \cup \Gamma_j^+, \ \Gamma_l^- \leftarrow \Gamma_l^- \cup \{i\} \cup \Gamma_i^-.$$

Relacja R pozwala ma eliminację wielu elementów z otoczenia, bez utraty rozwiązań optymalnych.

3. Otoczenie „swap dynasearch"

Podstawowym elementem algorytmów lokalnej optymalizacji jest otoczenie, generowane przez ruchy – przekształcenia (transformacje) pojedynczych rozwiązań. Ruch typu „zamień" (*swap*) polega na zmianie pozycjami dwóch elementów w permutacji.

Niech k oraz l ($0 \le k,l \le n$) będzie parą pozycji w permutacji $\pi \in S(n)$, gdzie

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

Ruch r_l^k (typu zamień) generuje permutację π_l^k ($r_l^k(\pi) = \pi_l^k$) zamieniając pozycjami zadanie $\pi(l)$ z $\pi(k)$. Wówczas, jeżeli $k < l$ (gdy $k > l$ jest podobnie) to permutacja

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

Złożoność obliczeniowa wykonania ruchu r_l^k wynosi $O(1)$. Oznaczmy przez $M(\pi)$ zbiór wszystkich takich ruchów. Otoczenie generowane z permutacji π przez te ruchy, ma $n(n-1)/2$ elementów. W tradycyjnych algorytmach otoczenie jest generowane przez pojedyncze transformacje (ruchy). Korzystając z definicji i oznaczeń zamieszczonych w pracy [4], wprowadzamy otoczenie generowane przez serie ruchów typu zamień.

Ruchy $r_j^i, r_l^k \in M(\pi)$ nazywamy *niezależnymi*, jeżeli

$$\max\{i,j\} < \min\{k,l\} \quad \text{lub} \quad \min\{i,j\} > \max\{k,l\}.$$

Otoczenie zwane „swap dynasearch" $DM(\pi)$ permutacji $\pi \in S(n)$, generowane przez ruchy typu zamień, zawiera wszystkie permutacje powstałe z π przez wykonanie serii parami niezależnych ruchów ze zbioru $M(\pi)$. Można łatwo pokazać, że $|DM(\pi)| = 2^{n-1} - 1$. Obecnie przedstawimy opis algorytmu opartego na metodzie programowania dynamicznego, umożliwiającego przeglądanie tego otoczenia (wyznaczanie elementu minimalnego) w czasie wielomianowym (algorytmem o wielomianowej złożoności obliczeniowej).

Do wyznaczenia serii ruchów niezależnych generujących z permutacji π najlepszy element otocznia $DM(\pi)$ stosujemy algorytm, w którym uszeregowanie jest konstruowane od początku (od pierwszej pozycji permutacji). Kolejny element dołącza się na koniec bieżącej podpermutacji a następnie, w razie potrzeby (dla zmniejszenia wartości funkcji celu), zamienia się go z innym.

Rozpatrujemy permutację $\pi \in S(n)$. Permutacja częściowa β zadań ze zbioru N należy do zbioru podpermutacji (j,π) dla $j = 0,1,\ldots,n$, jeżeli może być otrzymana z podpermutacji $(\pi(1), \pi(2), \ldots, \pi(j))$ przez wykonanie serii niezależnych ruchów ze zbioru $M(\pi)$. Aby więc wyznaczyć permutację o najmniejszej wartości funkcji celu z otoczenia $DM(\pi)$ należy rozpatrywać jedynie permutacje, które należą do zbioru (n,π).

Przez π_j oznaczamy permutację zadań ze zbioru $\{\pi(1), \pi(2), \ldots, \pi(j)\}$, która ma minimalną wartość funkcji celu spośród wszystkich permutacji ze zbioru (j,π), tj.

$$F(\pi_j) = \min\{F(\beta) : \beta \in (j,\pi)\}.$$

Permutację π_j można otrzymać z pewnej permutacji π_i ($0 \le i < j$), która ma minimalną wartość funkcji celu z wszystkich permutacji w pewnym zbiorze (i,π). Jeśli $i = j-1$, to zadanie $\pi(j)$ umieszczamy na końcu π_i. Natomiast, gdy $0 \le i < j-1$ wówczas dodatkowo zamieniamy miejscami $\pi(i+1)$ i $\pi(j)$. Oba te przypadki można zapisać następująco:

1. Jeżeli $i = j-1$, wówczas zadanie $\pi(j)$ dołączamy do π_{j-1}. Permutacja

$$\pi_j = (\pi_{j-1}, \pi(j)) \text{ oraz } F(\pi_j) = F(\pi_{j-1}) + f_{\pi(j)}(C_{\pi(j)}).$$

2. Niech $0 \le i < j-1$. Zadanie $\pi(j)$ po dołączeniu jest zamieniane z $\pi(i+1)$, więc

$$\pi_j = (\pi_i, \pi(j), \pi(j+2), \ldots, \pi(j-1), \pi(i+1)),$$

a wartość funkcji celu

$$F(\pi_j) = F(\pi_i) + f_{\pi(j)}(C_{\pi(i)} + p_{\pi(j)}) + \sum_{k=i+2}^{j-1} f_{\pi(k)}(C_{\pi(k)} + p_{\pi(j)} - p_{\pi(i+1)}) + f_{\pi(i+1)}(C_{\pi(j)}).$$

Stąd, wartość $F(\pi_j)$ można obliczyć z następującej zależności rekurencyjnej:

$$F(\pi_j) = \min \begin{cases} F(\pi_{j-1}) + f_{\pi(j)}(C_{\pi(j)}), \\ \min\limits_{0 \le i \le j-2} \{F(\pi_i) + f_{\pi(j)}(C_{\pi(i)} + p_{\pi(j)}) + \\ \sum\limits_{k=i+2}^{j-1} f_{\pi(k)}(C_{\pi(k)} + p_{\pi(j)} - p_{\pi(i+1)}) + f_{\pi(i+1)}(C_{\pi(j)}), \end{cases}$$

dla $j = 2,3,\ldots,n$ z warunkami początkowymi $F(\pi_0) = 0$ oraz $F(\pi_1) = w_{\pi(1)}U_{\pi(1)}$. W punkcie 2, po dołączeniu do podpermutacji zadania $\pi(j)$ jest ono zamieniane z $\pi(i+1)$ tylko wówczas, gdy po zamianie jest spełniona relacja R .

Optymalną wartość funkcji celu $F(\pi_n)$ oraz odpowiadającą jej permutację można wyznaczyć stosując metodę programowania dynamicznego, a następnie przeglądania „wstecz" procesu generowania wartości $F(\pi_n)$. Odpowiedni algorytm ma złożoności obliczeniowej $O(n^3)$ i wymaga $O(n)$ pamięci [4].

4. Algorytmy przybliżone.

Poniżej przedstawimy ogólny schemat działania algorytmu poszukiwań zstępujących z zastosowaniem otoczenia typu „swap dynasearch". Obliczenia rozpoczynamy od pewnego rozwiązania startowego $\pi^{(0)}$ wyznaczonego losowo lub przez dowolny algorytm konstrukcyjny. W k-tej iteracji algorytmy, w otoczeniu „swap dynasearch" $DM(\pi^{k-1})$ bieżącego rozwiązania $\pi^{(k-1)}$, szukamy permutacji o minimalnej wartości funkcji celu. Stosując programowanie dynamiczne obliczamy kolejno wartości $F(\pi_j^{(k-1)})$, dla $j = 1,2,\ldots,n$ a następnie, metodą wstecz, wyznaczamy permutację $\pi^{(k)}$. Jeżeli $F(\pi_n^{(k)}) < F(\pi_n^{(k-1)})$, wówczas $\pi^{(k)}$ jest bieżącym rozwiązaniem w następnej iteracji algorytmy, a w przeciwnym przypadku - algorytm kończy działanie. Permutacja $\pi^{(k-1)}$ jest wyznaczonym przez algorytm minimum lokalnym.

Algorytm poszukiwania zstępującego **ZS_DYN**:

Niech π^0 będzie dowolną permutacją startową, a π^* najlepszym do tej pory wyznaczonym rozwiązaniem.

Step 1: $\pi^* \leftarrow \pi^0$; $\pi \leftarrow \pi^0$;

Step 2: Stosując metodę programowania dynamicznego wyznaczyć permutację o najmniejszej wartości funkcji celu z otoczenia „swap dynasearch", tj. permutację β taką, że

$$F(\beta) = \min\{F(\delta):\ \delta \in DM(\pi)\};$$

***Step* 3: if** $F(\beta) < F(\pi)$ **then** $\pi \leftarrow \beta$ **and goto** ***Step* 2**

else $\pi^* \leftarrow \pi$ **and EXIT.** ■

Algorytm **ZS_DYN** jest uruchamiany wielokrotnie z rozwiązań otrzymywanych przez losowe zaburzenie (perturbację) bieżącego minimum lokalnego (algorytmu **ZS_DYN**). Ten zabieg pozwolił na większą dywersyfikację poszukiwań w przestrzeni rozwiązań dopuszczalnych. Wprowadziliśmy stałą *Max_it* ograniczającą liczbę takich multi-startów algorytmu. Wartość tej stałej ustalaliśmy tak, aby czas działania algorytmu (mierzony w milisekundach) był porównywalny do czas działania innego algorytmu, do którego porównywaliśmy algorytm poszukiwania zstępującego **ZS_DYN**.

Do wyznaczania rozwiązań startowych, dla algorytmów popraw, stosujemy algorytm konstrukcyjny META [14], wybierający najlepsze z rozwiązań wyznaczonych przez algorytmy konstrukcyjne: SWPT, EDD, AU oraz COVERT.

5. Wyniki obliczeniowe

Przedstawione algorytmy testowano na wielu losowych przykładach. Sposób ich generowania jest adaptacja metody przedstawionej w pracy [15]. Dla każdego zadania i, czas wykonywania p_i jest realizacją zmiennej losowej o rozkładzie jednostajnym na przedziale [1, 100], a waga funkcji kosztów w_i – na przedziale [1, 10]. Linie krytyczne d_i zależą od sumy czasów wykonywania zadań $P = \sum_{i=1}^{n} p_i$ oraz dodatkowo dwóch parametrów RDD (relative range of due dates) i TF (average tardiness factor). Wartości tych parametrów należą do zbioru {0.2, 0.4, 0.6, 0.8, 1.0}. Wartość d_i jest generowana losowo z przedziału [P(1 - TF – RDD/2), P(1 – TF + RDD/2)]. Dla każdej z 25 par wartości parametrów RDD i TF generowano 5 przykładów. W sumie, dla każdego n, generowano 125 przykładów. Do wyznaczania rozwiązania startowego stosowano algorytm konstrukcyjny META. Wyniki porównawcze algorytmu popraw z zabronieniami (tabu serach, TS) i algorytmu zstępującego z otoczeniem „dynasearch" ZS_DYN są zamieszczone w Tabeli 1. W algorytmie TS wykorzystano technikę multi-ruchów generujących (podobnie jak w metodzie „dynasearch") otoczenie o wykładniczej liczbie elementów. Do przeglądania otoczenia generowanego przez multi-ruchy jest stosowany algorytm heurystyczny.

Tabela 1. Średnia poprawa rozwiązań algorytmu META [14]

Liczba zadań n	**Algorytm TS**			**Algorytm ZS_DYN**		
	poprawa	**liczba iteracji**	**średni czas [ms]**	**poprawa**	***Max_it***	**średni czas [ms]**
40	14,139%	$2n^2$	32	32,055%	7	32
50	12,530%	$2n^2$	65	32,998%	7	67
100	13,658%	$2n^2$	536	33,967%	3	540
średnio	13,442%			33,007%		

Na podstawie otrzymanych wyników można jednoznacznie stwierdzić, że algorytm poszukiwania zstępującego ZS_DYN z otoczeniem „dynasearch" osiąga znacznie lepsze wyniki od algorytmu poszukiwania za zabronieniami TS. Poprawa algorytmu TS względem rozwiązań algorytmu konstrukcyjnego META wyniosła średnio nieco ponad 13%, podczas gdy algorytmu ZS_DYN aż 33%, przy tym samym czasie obliczeń komputera klasy PC z procesorem Intel Celeron 450MHz..

6. Podsumowanie

W pracy przedstawiono zagadnienie szeregowania zadań z liniami krytycznymi na jednej maszynie z minimalizacją sumy kosztów spóźnień $1||\sum w_i U_i$. Do jego rozwiązywania skonstruowano algorytm oparty na metodzie poszukiwań lokalnych z otoczeniem generowanym przez serię klasycznych ruchów typu zamień (*swap*). Do przeglądania tego otoczenia, które ma wykładniczą liczbę elementów, zastosowano metodę programowania dynamicznego. Umożliwia ona wyznaczenie najlepszego rozwiązania w czasie wielomianowym. Dodatkowo, wprowadzono relację częściowego porządku na zbiorze zadań, eliminującą pewne elementy. Przeprowadzono eksperymenty obliczeniowe na danych generowanych losowo, a otrzymane rozwiązania porównano z wynikami innych algorytmów.

Praca naukowa finansowana częściowo ze środków KBN w latach 2003-2006 jako projekt badawczy, nr T11A01624

Literatura:

1. Ahuja R.K., Ergun O., Orlin B., Punnen A.P.: A survey of very large-scale neighborhood search techniques, Discrete Applied Mathematics 123 (2002), 75-102.
2. Bożejko W., Wodecki M.: A Branch-and-Bound Parallel Algorithm for Single-Machine Total Weighted Tardiness Problem, (w przygotowaniu).
3. Bożejko W., Wodecki M.: Algorytmy metaheurystyczne rozwiązywania problemu szeregowania zadań, AUTOMATYKA, t.3 z.1 (1999), 345-354.
4. Congram K.H., Potts C.N., Van de Velde S.: An interated dynasearch algorithm for the single machine total weighted tardiness problem, INFORMS, Journal on Computing 14 (2002), 52-67.
5. Emmons H.: One-Machine Sequencing to Minimize Certain Functions of Job Tardiness, Operations Research, 17 (1969), 701-705.
6. Garey M.R., Johnson D.S.: Scheduling tasks with nonuniform deadlines on two processor, Journal of ACM, 23 (1976), 461-467.
7. Karp R.M.: Reducibility among Combinatorial Problems, Complexity of Computations, R.E. Millerand J.W. Thatcher (Eds.), Plenum Press, New York 1972, 85-103.
8. Kise H., Ibaraki T., Mine H.,: A solvabe case of the one-machine scheduling problem with ready times and due times, Operations Research, 26 (1978), 121-126.
9. Lawler E.L.: Private communication.
10. Lawler E.L., Moore J.M.: A Functional Equation and its Applications to Resource Allocation and Sequencing Problems, Management Sci., 16 (1969), 77-84.
11. Lenstra J.K., Rinnoy Kan A.H.G.: Complexity results for scheduling chains on a single machine, Europ. J. Oper. Res. 4 (1980), 270-275.

12. Monma C.I.: Linear-time algorithms for scheduling on parallel processor, Operations Research, 30 (1982), 116-124.
13. Moore J.M.: An *n*-job, one machine sequencing algorithm for minimising the number of late jobs, Mgmt. Sci,. 15 (1968), 102-109.
14. Potts C.N., Van Wassenhove L.N.: A Branch and Bound Algorithm for the Total Weighted Tardiness Problem, Operations Research, 33 (1985), 177-181.
15. Potts C.N., Van Wassenhove L.N.: Algorithms for Scheduling a Single Machine to Minimize the Weighted Number of Late Jobs, Management Science, vol. 34, 7 (1988), 843-858.
16. Sahni S.K.: Algorithms for Scheduling Independent Jobs, J.Assoc. Comput. Match., 23 (1976), 116-127.
17. Shwimer J.: On the N-Job, One-Machine, Sequence-Independent Scheduling Problem with ardiness Penalties: a Branch-and-Bound Solution, Management Sci. 18 (1972), 301-313.
18. Villareal F.J., Bulfin R.L.: Scheduling a Single Machine to Minimize the Weighted Number of Tardy Jobs, IEE Trans., 15 (1983), 337-343.

dr Wojciech Bożejko
Instytut Cybernetyki Technicznej
Politechniki Wrocławskiej
ul. Janiszewskiego 11/17, 50-372 Wrocław
e-mail: wbo@ict.pwr.wroc.pl

dr Mieczysław Wodecki
Instytut Informatyki
Uniwersytetu Wrocławskiego
ul. Przesmyckiego 20, 51-151 Wrocław
e-mail: mwd@ii.uni.wroc.pl