ALGORYTM DYNASEARCH DLA JEDNOMASZYNOWEGO PROBLEMU SZEREGOWANIA ZADAŃ

1. Wprowadzenie

W rozpatrywanym jednomaszynowym problemie szeregowania zadań z liniami krytycznymi (w skrócie SWCT) każde zadanie należy wykonać, bez przerywania, na jednej maszynie. Uszeregowanie (rozwiązanie) jest dopuszczalne, jeżeli wykonywanie każdego zadanie kończy się przed jego linią krytyczną (jest terminowe). Należy wyznaczyć (jeżeli istnieje) taką dopuszczalną kolejność wykonywania zadań, która minimalizuje sumę ważonych czasów zakończenia wykonywania zadań. Problem ten został sformułowany przez Smith'a w pracy [9] i jest w literaturze oznaczany przez $1|d_i|\sum w_iC_i$. Należy on do klasy problemów *silnie NP-trudnych* [4].

Obecnie istnieje wiele algorytmów dokładne rozwiązywania tego problemu opartych na metodzie podziału i ograniczeń: Bensal [1], Potts i Van Wassenhowe [7], Posner [6] oraz Yupeng Pan [5]. Algorytm Posner'a pozwala na rozwiązywanie przykładów, w których liczba zadań nie przekracza 40 oraz w szczególnych przypadkach, z liczbą zadań 50 i 60 (zobacz [5]). Najlepszy obecnie algorytm (bazujący na idei metody Posner'a [6]) jest zamieszczony w pracy [5]. Dzięki zastosowaniu pewnych własności eliminacyjnych, zredukowano w nim znacznie liczbę wierzchołków drzewa rozwiązań. Pozwoliło to na rozwiązywanie przykładów o jeszcze większej liczbie zadań. Jednak stosowanie algorytmów dokładnych, do rozwiązywania przykładów praktycznych o znacznych rozmiarach, jest niemożliwe ze względu na gwałtowny wzrost czasu obliczeń.

Obecnie, do rozwiązywania NP-trudnych problemów optymalizacji kombinatorycznej stosuje się głównie „inteligentne" algorytmy przybliżone. Najpopularniejsze z nich, to metaheurystyki: poszukiwania z zabronieniami (tabu serach) symulowanego wyżarzania oraz algorytmu genetycznego. Dla wielu problemów, wyznaczane przez te algorytmy rozwiązania są z praktycznego punktu widzenia w pełni zadawalające (różnią się od najlepszych rozwiązań o mniej niż 1%). Są przy tym szybkie i stabilne. Algorytmy tabu search i symulowanego wyżarzania 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 rozwiązaniem bazowym. Postępowanie to jest zazwyczaj kontynuowane aż do osiągnięcia minimum lokalnego. W wielu problemach szeregowania zadań rozwiązaniami dopuszczal-

nymi są n elementowe permutacje. Otoczenie, w klasycznych algorytmach poszukiwań lokalnych, jest generowane przez przekształcenie (ruch) rozwiązania bazowego polegające na:

i) przestawieniu (*insert*) pewnego elementu w permutacji na inną pozycję,

ii) zamianę (*swap*) dowolnej pary elementów.

Tak wyznaczone otoczenia zawierają odpowiednio $n(n-1)/2$ oraz $(n-1)^2$ elementów.

Otoczenie – jego wielkość, sposób generowania i przeglądania ma zasadniczy wpływ na złożoność obliczeniową, czas działania oraz jakość wyznaczanych przez algorytm rozwiązań. W pracy [3] (Congram, Potas i Van Wassenhowe) zdefiniowano 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 SWCT, w konstrukcji którego wykorzystano metodę dynasearch. Do wyznaczania rozwiązań startowych stosujemy algorytm konstrukcyjny opisany w Rozdziale 4. Ponieważ, dla rozważanego problemu nie istnieją rozwiązania referencyjne dlatego, porównujemy otrzymane wyniki (dla losowo generowanych przykładów) z górnymi ograniczeniami wartości funkcji celu oraz wynikami równoległego algorytmu genetycznego zamieszczonymi w pracy Bożejko, Wodecki [2]. Sposób generowania danych testowych został zaczerpnięty z pracy [7]. Przykłady oraz najlepsze otrzymane rozwiązania zamieszczamy także na naszej stronie internetowej [10].

2. Sformułowanie problemu

W rozpatrywanym jednomaszynowym problemie szeregowania, każde z *n* zadań należy, bez przerywania, wykonać na jednej maszynie. Dla zadania *i*, niech p_i będzie *czasem wykonywania*, d_i - *żądanym terminem zakończenia* (linią krytyczną), a w_i - *wagą funkcji kosztów* (celu). W dowolnym uszeregowaniu zadanie jest *terminowe*, jeżeli kończy się przed linią krytyczną. Uszeregowanie jest natomiast *dopuszczalne*, jeżeli wszystkie zadania są terminowe. W rozważanym problemie należy wyznaczyć rozwiązanie dopuszczalne, które minimalizuje sumę ważonych czasów zakończenie wykonywania zadań, tj. $\sum_{i=1}^{n} w_i C_i$, gdzie C_i jest czasem zakończenia zadania *i*.

Niech $N = \{1, 2, \ldots, n\}$ będzie zbiorem zadań. Przez Φ_n oznaczamy zbiór wszystkich permutacji elementów zbioru *N*. Permutacja $\pi \in \Phi_n$ jest dopuszczalna (terminowa), jeżeli

$$\forall i \in N,\ d_{\pi(i)} \geq C_{\pi(i)}, \quad \text{gdzie} \quad C_{\pi(i)} = \sum_{j=1}^{i} p_{\pi(j)}$$

jest czasem zakończenia wykonywania zadania $\pi(i)$ w permutacji (uszeregowaniu) π.

Problem SWCT polega na wyznaczeniu elementu optymalnego π^* takiego, że

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

gdzie $\Pi \subseteq \Phi_n$ jest zbiorem rozwiązań dopuszczalnych (terminowych), a funkcja celu

$$F(\pi) = \sum_{i=1}^{n} w_{\pi(i)} * C_{\pi(i)}, \quad \pi \in \Pi.$$

Uszeregowanie (permutacja) zadań $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ jest zgodne z liniami krytycznymi, w skrócie jest EDD (Earliest Due Date), jeżeli $d_{\pi(1)} \leq d_{\pi(2)} \leq, \ldots, \leq d_{\pi(n)}$.

Lemat 1. Jeżeli dla instancji problemu SWCT uszeregowanie zadań EDD jest terminowe, to zbiór rozwiązań dopuszczalnych $\Pi \neq \varnothing$, a w przeciwnym przypadku $\Pi = \varnothing$.

Dowód. Lemat można łatwo udowodnić, stosując metodę nie wprost. ■

3. Metoda dynasearch.

Podstawowym elementem algorytmów lokalnej optymalizacji jest otoczenie. Wielkość otoczenia, sposób jego generowania oraz przeglądania ma decydujący wpływ na czas obliczeń, zbieżność oraz wartości wyznaczanych przez algorytm rozwiązań. W tradycyjnych algorytmach otoczenie jest generowane przez pojedyncze transformacje (ruchy).

Niech *k* oraz *l* ($0 \leq k, l \leq n$) będzie parą pozycji w permutacji $\pi \in \Pi$, 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 (swap) 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.

Korzystając z definicji i oznaczeń zamieszczonych w pracy [3], 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 $\pi \in \Pi$ 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 oparty na metodzie programowania dynamicznego. W algorytmie tym 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 $\pi \in \Pi$. 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 swap dynasearch $DM(\pi)$, 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

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

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

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

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

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

dla $j = 2,3,\ldots,n$ z warunkami początkowymi $F(\pi_0) = 0$ oraz $F(\pi_1) = w_{\pi(1)}p_{\pi(1)}$.

Optymalną wartość funkcji celu $F(\pi_n)$ oraz odpowiadającą jej permutację można wyznaczyć stosując metodę „wstecz". Algorytm, który to realizuje (oparty na metodzie programowania dynamicznego) ma złożoności obliczeniowej $O(n^3)$ i wymaga $O(n)$ pamięci.

4. Algorytmy rozwiązywania problemu SWCT

W rozdziale tym przedstawimy dwa algorytmy przybliżone dla rozwiązywania rozpatrywanego problemu szeregowania zadań z liniami krytycznymi na jednej maszynie. Pierwszy z nich jest algorytmem konstrukcyjnym, a drugi jest oparty na idei poszukiwania zstępującego z wykorzystaniem otoczenia swap dynasearch.

Algorytm konstrukcyjny WBH:

Step 1: Posortować zadania według niemalejących wartości linii krytycznych d_i;

Step 2: Ustalić dwa pierwsze zadania w takiej kolejności, aby uszeregowanie było dopuszczalne, a wartość funkcji celu (dla tych zadań) minimalna;

Step 3: For *k*=3 **to** *n* **do**

wstawić *k*-te zadanie na jedną z pozycji $1,2,\ldots,k$ tak, aby *k* elementowa podpermutacja była dopuszczalna, a wartość funkcji celu minimalna.

Złożoność obliczeniowa algorytmu wynosi $O(n^2)$.

Poniżej przedstawimy ogólny schemat działania algorytmu poszukiwań zstępujących z zastosowaniem otoczenia typu swap dynasearch.

Obliczenia rozpoczynamy od pewnego dopuszczalnego 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 PS_DYN:

Niech π^0 będzie dowolną (dopuszczalną) 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.**

W celu poprawienia jakości otrzymywanych rozwiązań zaimplementowaliśmy także algorytm w wersji *multi-start*, tzn. algorytm poszukiwania zstępującego uruchamiany wielokrotnie z rozwiązań otrzymywanych przez losowe zaburzenie (perturbację) bieżącego minimum lokalnego (algorytmu *PS_DYN)*. Ten zabieg pozwolił na większą dywersyfikację poszukiwań w przestrzeni rozwiązań dopuszczalnych. Wprowadziliśmy także stałą *Max_it*=100 ograniczającą liczbę takich imulti-startów algorytmu. Stąd, jego złożoność obliczeniowa (z dokładnością do tej stałej) wynosi $O(Max\_it \cdot n^3)$.

5. Wyniki obliczeniowe

Przedstawione algorytmy testowano na wielu losowych przykładach. 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]. Niech $P = \sum_{i=1}^{n} p_i$. Wartości linii krytycznych są zależne od *P* oraz dwóch parametrów *L* i *R*, gdzie *L* przyjmuje wartości od 0.2 do 1.0 co 0.1, a *R* wartości od 0.2 do 1.6 co 0.2. Wartość d_i jest generowana losowo z przedziału $[P(L - R/2, P(L + R/2]$. Dla każdej pary parametrów *R* i *L* (takich par jest 40) było generowanych 10 przykładów. W sumie, dla każdego *n* (*n*=20, 40, 60, 80, 100, 120), wygenerowano 400 przykładów. Nie każdy z tych przykładów miał rozwiązanie dopuszczal-

ne. Sprawdzano to, korzystając z Lematu 1. Liczba przykładów, spośród 400 wygenerowanych, które miały niepusty zbiór rozwiązań dopuszczalnych jest przedstawiona w drugim wierszu Tablicy 1, oznaczonym przez *nfi*.

W pierwszej kolejności porównujemy algorytmy konstrukcyjne EDD i WBH z dolnymi ograniczeniami wartości funkcji celu. Algorytm ich obliczania (polegający na rozwiązaniu zadania Przydziału Pracy) jest opisany w pracy [2].

Tablica 1

Wyniki algorytmów konstrukcyjnych EDD i WBH

	n=20		*n*=40		*n*=60		*n*=80		*n*=100		*n*=120	
nfi	229		245		249		251		250		251	
	δ^1_{avg}	$\delta^2_{\max}$	δ_{avg}	$\delta_{\max}$	δ_{avg}	$\delta_{\max}$	δ_{avg}	$\delta_{\max}$	δ_{avg}	$\delta_{\max}$	δ_{avg}	$\delta_{\max}$
EDD	36.2	49.6	43.5	55.1	49.7	61.7	48.6	68.6	53.7	72.4	56.9	93.7
WBH	18.4	26.6	21.4	32.7	25.8	35.4	27.1	39.7	26.2	33.6	31.4	41.8

[1]średnie względne procentowe odchylenie od dolnego ograniczenia.
[2]maksymalne względne procentowe odchylenie od dolnego ograniczenia.

Algorytm konstrukcyjny WBH ma w stosunku do dolnych ograniczeń średnio o ponad 50% mniejszy błąd względny niż algorytm EDD.

W Tablicy 2 przedstawiamy wyniki obliczeniowe algorytmu PS_DYN(*m*), gdzie *m* jest liczbą uruchomień algorytmu zstępującego po wykonaniu perturbacji (tj. algorytmu z mechanizmem *multi-startu*). Dla porównania, zamieszczamy także wyniki działania równoległego algorytmu genetycznego PGA(*k*) z pracy [2], gdzie *k* jest liczbą procesorów na których wykonywano obliczenia. Algorytm *PS_DYN*(100) w sumie wykonywał podobną liczbę iteracji jak algorytmy genetyczne.

Tablica 2

Średnia procentowa poprawa względem algorytmu WBH

n	20	40	60	80	100	120
$PRD^{PGA(1)}$	-4.796	-5.685	-5.700	-6.901	-10.103	-9.816
$PRD^{PGA(8)}$	-6.110	-7.902	-8.474	-9.025	-11.595	-9.608
$PRD^{PS\_DYN(1)}$	-10,858	-16,908	-18,548	-17,515	-15,339	-11,459
$PRD^{PS\_DYN(100)}$	**-11,281**	**-17,289**	**-18,667**	**-17,631**	**-15,362**	**-11,491**

Niech F^{WBH} będzie wartością rozwiązania wyznaczonego przez algorytm WBH, a F^{Ω} wartością rozwiązania wyznaczonego przez algorytm $\Omega \in \{PS\_DYN(1), PS\_DYN(100), PGA(1), PGA(8)\}$. Dla każdego przykładu liczono średnią względną procentową poprawę wartości rozwiązana F^{WBH} względem wartości rozwiązania F^{Ω}, tj. liczono

$$PRD^{\Omega} = ((F^{WBH} - F^{\Omega}) / F^{\Omega}) \cdot 100\% .$$

Algorytm *PS_DYN* został zaimplementowany i przetestowany na komputerze klasy PC z procesorem Celeron 450MHz. Maksymalny czas obliczeń algorytmu *PS_PGA*(*m*) dla pojedynczego przykładu największego rozmiar nie przekraczał kilku sekund.

Na podstawie otrzymanych wyników można stwierdzić, że już jednoprzebiegowy algorytm poszukiwania zstępującego *PS_PGA*(1) daje o wiele lepsze wyniki niż oba algorytmy genetyczne. Dodanie mechanizmu *multi-startu* pozwoliło te wyniki jeszcze nieznacznie poprawić.

6. Podsumowanie

W pracy przedstawiono zagadnienie szeregowania zadań z liniami krytycznymi na jednej maszynie. 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 programowanie dynamiczne. Umożliwia ona wyznaczenie najlepszego rozwiązania w czasie wielomianowym. Dodatkowo, wprowadzono tzw. *multi-start*. Przeprowadzono eksperymenty obliczeniowe na danych generowanych losowo, a otrzymane rozwiązania porównano z wynikami innych algorytmów. Zastosowanie technik poszukiwania z zabronieniami, nawrotów (*backtrack-jump*) i innych bardziej złożonych elementów współczesnych metaheurystyk pozwoli z pewnością wykorzystać metodę dynasearch w jeszcze większym stopniu.

LITERATURA:

1. Bensal S.: Schingle machine scheduling to minimize weighted sum of completion times with secondary criterion-a branch and bound approach, European J. Oper. Res. 5, 1980, pp. 177-181.
2. Bożejko W., Wodecki M.: Parallel genetic algorithm for minimize total weighted completion time, LNCS, Springer Verlag (w druku).

3. Congram K.H., Potts C.N., Van de Velde S.: An interated dynasearch algorithm for the single machine total weighted tardiness problem, INFORMS, J.Comput. 14, 2002, pp. 52-67.
4. Lenstra J.J., Rinnoy KanA.H.G., Brucker P.: Complexity of machine scheduling problems, Ann. Discrete Math., 1979, pp. 287-326.
5. Pan Y.: An improved branch and bound algorithm for single machine scheduling with deadlines to minimize total weighted completion time, Operation Research Letters 31, 2003, pp. 492-496.
6. Posner M.: Minimizing weighted completion times with deadlines, Oper. Res. 33, 1985, pp. 562-574.
7. Potts C.N., Van Wassenhove L.N.: Algorithm for single machine sequencing with dedlines to minimize total weighted completion times, European J. Oper Res. 12, 1983, pp. 379-387.
8. Rinnoy Kan A.G.H, Lageweg, B.J. Lenstra J.K.: Minimizing total cost one-machine scheduling, Operations Research, 26, 1975, pp. 908-972.
9. Smith W.E.: Various optimizers for single-stage production, Naval Res. Logist. Quart. 3, 1956, pp. 59-66.
10. Benchmarks Library http://www.ii.uni.wroc.pl/~mwd/wobz.html