ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: AUTOMATYKA z. 123 Nr kol. 1389

Krzysztof CHUDZIK, Adam JANIAK
Politechnika Wrocławska

ALGORYTM TABU SEARCH DLA WYBRANYCH PROBLEMÓW SZEREGOWANIA ZADAŃ NA POJEDYNCZEJ MASZYNIE Z PRZEZBROJENIAMI*

Streszczenie. Artykuł poświęcony jest szeregowaniu zadań na pojedynczej maszynie z przezbrojeniami sekwencyjnie zależnymi. Rozpatrywane są trzy kryteria: 1) czas zakończenia wykonywania wszystkich zadań, 2) maksymalna nieterminowość, 3) suma ważonych czasów zakończenia wykonywania zadań. Prezentowane problemy są NP-trudne. Do ich rozwiązania użyto algorytmu typu tabu search.

TABU SEARCH ALGORITHM FOR SOME SINGLE MACHINE SHEDULING PROBLEMS WITH SETUPS

Summary. The paper is devoted to single machine scheduling problems with sequence dependent setup times. The following criterion functions are considered: 1) maximum completion time (makespan), 2) maximum lateness, and 3) weighted sum of completion times. Presented problems are NP-hard. A tabu search algorithms was used to solve these problems.

1. Wprowadzenie w problematykę

Wiele praktycznych problemów szeregowania zawiera wykonywanie zadań podobnych lub identycznych pod względem wymagań produkcyjnych. Jako przykład może posłużyć wytwarzanie w elastycznych systemach produkcyjnych (ESP). ESP pozwalają produkować stosunkowo szeroki asortyment wyrobów na jednym zestawie maszyn, który jest przystosowywany do zmieniających się wymagań w zależności od wykonywanych zadań. Układając plan produkcji należy brać pod uwagę czas i koszt dostosowywania ESP do produkcji kolejnych wyrobów. Zagadnienie to uwzględniane jest w literaturze dotyczącej szeregowania zadań w postaci tzw. przezbrojeń (ang. *setups*). Wiąże się ono z wyznaczaniem wielkości partii produkcyjnych (ang. *batching*) tak, aby minimalizować czas i nakłady na przezbrojenia, a równocześnie jak najlepiej zaspokajać zapotrzebowania klienta. Problematyka

szeregowania zadań z przezbrojeniami maszyn stała się ważną częścią ogólnej teorii szeregowania zadań. Szerszy przegląd stosowanych modeli oraz literatury można znaleźć np. w pracy przeglądowej autorów [4]

W niniejszej pracy zajmujemy się problemami szeregowania zadań na pojedynczej maszynie z przezbrojeniami sekwencyjnie zależnymi. Problemy optymalizacyjne rozpatrywane w niniejszej pracy to znalezienie takich permutacji zadań, które minimalizują odpowiednio kryteria: 1) czas ukończenia wykonywania wszystkich zadań, 2) maksymalną nieterminowość oraz 3) ważoną sumę czasów zakończenia wykonywania zadań. Wszystkie rozważane problemy są NP-trudne. W świetle obecnej wiedzy na temat złożoności obliczeniowej nie jest możliwe skonstruowanie dla tych problemów algorytmów znajdujących rozwiązania optymalne w czasie wielomianowym [1]. Autorzy wybrali do rozwiązania tych problemów algorytm tabu search (TS).

Praca zawiera precyzyjne sformułowanie problemów optymalizacyjnych (rozdz. 2), opis zaimplementowanego algorytmu TS (rozdz. 3), opis przeprowadzonego eksperymentu numerycznego i jego rezultaty (rozdz. 4) oraz wnioski (rozdz. 5).

2. Sformułowanie problemu

Dany jest zbiór N zadań $J = \{ j : j = 1, 2, \ldots, N\}$. Zadania mają być wykonane na pojedynczej maszynie (lub w gnieździe czy centrum obróbkowym, traktowanym jako niepodzielna całość). Maszyna może w danej chwili czasu wykonywać co najwyżej jedno zadanie. Przerywanie wykonywania zadania jest niedozwolone. Nie występują ograniczenia na kolejność wykonywania zadań. Dla każdego zadania $j \in J$ określone są następujące parametry: $p_j \geq 0$ – czas wykonywania zadania na maszynie, $r_j \geq 0$ – termin dostępności zadania, tzn. czas, po którym można rozpocząć wykonywanie zadania, $d_j \geq 0$ – pożądany termin zakończenia wykonywania zadania oraz $w_j \geq 0$ – waga przyporządkowana zadaniu, charakteryzująca jego ważność (priorytet) w zbiorze J.

Dany jest zbiór B rodzin zadań $F = \{ I_b : b = 1, 2, \ldots, B\}$. Rodzina jest to zbiór zadań, które są podobne lub identyczne pod względem wymagań produkcyjnych, np.: ich wykonanie jest możliwe przy użyciu tego samego zestawu narzędzi. Każde zadanie $j \in J$ należy dokładnie do jednej rodziny $I_b \in F$. Przynależność zadania do danej rodziny jest jego dodatkowym atrybutem i będziemy oznaczać to parametrem f_j (lub $f(j)$), który ma wartość równą indeksowi

* Praca finansowana ze środków Grant KBN No. 8 T11A 004 13 oraz Grant KBN No. 8 T11F 001 11.

rodziny, do której należy (tzn. jeżeli $j \in I_b$ to $f_j = b$). W procesie produkcyjnym pomiędzy kolejnymi zadaniami mogą wystąpić czasy przygotowawcze, tzw. przezbrojenia (ang. *setups*). W tym czasie nie może być wykonywane żadne zadanie. Podczas jego trwania następuje przygotowanie maszyny do wykonywania następnego zadania, np.: wymiana narzędzi. Jeżeli zadania i oraz j należą do dwóch różnych rodzin: $i \in I_a$ oraz $j \in I_b$, gdzie $a \neq b$, to pomiędzy zadaniami wymagany jest czas przezbrojenia $s_{ab} \geq 0$. Jeżeli zadania należą do tej samej rodziny, to przezbrojenie nie występuje, tzn. gdy $a = b$, to $s_{ab} = 0$. Jeżeli zadanie $j \in I_b$ wykonywane jest na maszynie jako pierwsze w sekwencji zadań, to przed nim występuje przezbrojenie $s_{0b} \geq 0$ (I_0 – jest fikcyjną rodziną wprowadzoną dla ujednolicenia oznaczeń). Tak więc przezbrojenia s_{ab} są określone dla każdej pary rodzin I_a oraz I_b, gdzie $a = 0, 1, 2, \ldots, B$. oraz $b = 1, 2, \ldots, B$ i nazywane są dlatego przezbrojeniami sekwencyjnie zależnymi. Zakłada się, że przezbrojenie może rozpocząć się przed upływem terminu dostępności r_j zadania, dla którego wykonywane jest to przezbrojenie (tzn. zadania j następującego po tym przezbrojeniu). Rozpatrując sytuacje praktyczne, czyni się dodatkowe założenie, że czasy przygotowawcze (przezbrojenia) spełniają nierówność trójkąta, tzn. $s_{ac} \leq s_{ab} + s_{bc}$ dla $a = 0, 1, 2, \ldots, B$ oraz $b,c = 1, 2, \ldots, B$. W niniejszej pracy będziemy również korzystać z tego założenia. Gdyby to założenie nie było spełnione, może okazać się, że gdy mamy przezbroić maszynę z rodziny I_a na rodzinę I_c, to bardziej opłacalne będzie dokonanie przezbrojenia z I_a na I_b, a następnie z I_b na I_c nie wykonując żadnego zadania z rodziny I_b. Rzeczywisty czas przezbrojenia byłby mniejszy niż określony parametrem s_{ac}, co sygnalizuje w praktyce błąd określenia wartości tego parametru.

Przyjmuje się dodatkowo oznaczenia S_j – czas rozpoczęcia wykonywania zadania j, C_j – czas zakończenia wykonywania zadania j. Permutację zadań oznaczać będziemy π, natomiast zadanie na pozycji i-tej w permutacji π poprzez symbol $\pi(i)$.

W oparciu o przedstawione powyżej parametry zadań i ich rodzin formułujemy trzy problemy optymalizacyjne, które brzmią:

Znaleźć taką permutację π^* wszystkich zadań ze zbioru J, która minimalizować będzie następujące kryterium:

1) czas zakończenia wykonywania wszystkich zadań $C_{max} = \max_{j \in J} C_j$,

2) maksymalną nieterminowość $L_{max} = \max_{j \in J} L_j$, gdzie $L_j = (C_j - d_j)$,

3) ważoną sumę czasów zakończenia wykonywania wszystkich zadań $\Sigma_{j \in J} (w_j C_j)$, którą przyjęto oznaczać $\Sigma w_j C_j$.

Oczywiście przy optymalizacji kryterium C_{max} nie są uwzględniane parametry d_j oraz w_j, ze względu na postać kryterium, natomiast przy L_{max} parametry w_j, a przy $\Sigma w_j C_j$ parametry d_j.

Notacja i złożoność obliczeniowa problemów jest następująca (w wyjaśnieniu zarys dowodu):

1) $1 \mid r_j, s_{fg} \mid C_{max}$ – jest problemem NP-zupełnym – jeżeli założymy $r_j = 0$, to zagadnienie sprowadza się do NP-zupełnego zagadnienia komiwojażera [1].

2) $1 \mid r_j, s_{fg} \mid L_{max}$ – jest problemem silnie NP-zupełnym – jeżeli założymy $s_{ab} = 0$, to otrzymamy dokładnie problem $1 \mid r_j \mid L_{max}$, który jest problemem silnie NP-zupełnym, np.:[7].

3) $1 \mid r_j, s_{fg} \mid \Sigma w_j C_j$ – jest problemem silnie NP-zupełnym – jeżeli założymy $s_{ab} = 0$, to otrzymamy problem $1 \mid r_j \mid \Sigma w_j C_j$, który jest problemem silnie NP-zupełnym, np.: [7].

3. Algorytm tabu search

Ze względu na złożoność obliczeniową większości przypadków szeregowania zadań z przezbrojeniami maszyn, szukając algorytmów rozwiązujących, uwagę kierujemy na algorytmy przybliżone lub tzw. metaheurystyki, np.: algorytmy genetyczne [3], czy tabu search [8]. Zastosowanie techniki tabu search stało się przedmiotem niniejszej pracy.

Technika tabu search [5,6] jest dosyć popularna, dlatego nie będziemy omawiać jej szczegółowo. W sposób dokładny opisane zostaną te elementy, które są specyficzne dla rozwiązywanego zagadnienia.

3.1. Zarys metody tabu search

Algorytm oparty na tej technice rozpoczyna start od pewnego początkowego rozwiązania bazowego. Dokonywane jest przeszukiwanie otoczenia rozwiązania bazowego w celu znalezienia najlepszego rozwiązania w tym otoczeniu (może ono być gorsze od bazowego). To rozwiązanie staje się rozwiązaniem bazowym dla następnej iteracji algorytmu i cykl się powtarza, aż nie zostanie osiągnięty założony warunek stopu algorytmu. Otoczenie jest zbiorem rozwiązań, w które można przekształcić rozwiązanie bazowe za pomocą ruchu. Ruch jest funkcją parametryczną (na ogół niezbyt skomplikowaną) przekształcającą jedno rozwiązanie w drugie, np. w permutacji: przenieś element z pozycji i na pozycję j. Aby unikać wpadania w cykle, po wykonaniu ruchu zapisuje się na tzw. listę (lub listy) tabu pewne atrybuty ruchu i/lub wygenerowanego rozwiązania. Podczas przeszukiwania otoczenia nie przyjmujemy rozwiązania za bazowe dla następnej iteracji, jeśli na liście tabu znajdują się

atrybuty tego rozwiązania lub ruchu do niego prowadzącego. Czasem takie ograniczenie jest zbyt silne i rozwiązanie takie może być przyjęte za bazowe, jeśli jego wartość jest poniżej wartości tzw. funkcji aspiracji dla tego rozwiązania (przy minimalizacji). Prostą funkcją aspiracji może być np.: wartość najlepszego rozwiązania znalezionego przez algorytm do danej chwili. Aby dodatkowo poprawić efektywność algorytmu, stosuje się tzw. pamięć długoterminową. W przypadku gdy algorytm wykona określoną liczbę iteracji bez poprawy wartości funkcji kryterialnej, odtwarzany jest jego stan z iteracji, w której znalazł najlepsze rozwiązanie. W pamięci tej przechowywane jest rozwiązanie bazowe wraz z towarzyszącą mu listą (lub listami) tabu. Po wykonaniu ruchu po odtworzeniu stanu, do pamięci długoterminowej dołączana jest informacja, jaki ruch został wykonany oraz uzupełnia się listę (listy) tabu o atrybuty tego ruch, aby uniknąć jego powtórzenia po ponownym odtworzeniu stanu algorytmu. Pamięć taka realizowana jest w postaci stosu o określonej długości. Gdy liczba elementów przekracza ustaloną wartość, usuwa się element najwcześniej położony. Określa się też liczbę nawrotów do danego stanu. Jako pierwszy element umieszcza się rozwiązanie startowe.

3.2. Rozwiązanie startowe

Rozwiązanie startowe generowane jest za pomocą algorytmu przybliżonego H. Dla poszczególnych funkcji kryterialnych przybiera on postać:

H dla C_{max} – reguła szeregowania jest następująca: spośród zadań aktualnie dostępnych wybierz i wstaw na końcu permutacji zadań uszeregowanych zadanie, które po uszeregowaniu będzie miało najmniejszy czas zakończenia wykonywania (uwzględniamy czasy przezbrojeń).

H dla L_{max} – jest to modyfikacja algorytmu Schrage (np.: [2]): spośród zadań aktualnie dostępnych wybierz i wstaw na końcu permutacji zadań uszeregowanych zadanie o najmniejszym d_j (uwzględniamy oczywiście czasy przezbrojeń w wyliczeniu S_j).

H dla $\Sigma w_j C_j$ – reguła szeregowania jest następująca: spośród zadań aktualnie dostępnych wybierz i wstaw na końcu permutacji zadań uszeregowanych zadanie o najmniejszym stosunku p_j/w_j (uwzględniamy czasy przezbrojeń w wyliczeniu S_j).

3.3. Ruch i otoczenie

Przez ruch v rozumiemy przeniesienie zadania z pozycji i na pozycję j w permutacji bazowej:

π_v=(π(1), ..., $\pi(i$-1), $\pi(i$+1), ..., $\pi(j)$, $\pi(i)$, $\pi(j$+1), ..., $\pi(N)$), jeżeli $i<j$

π_v=(π(1), ..., $\pi(j$-1), $\pi(i)$, $\pi(j)$, ..., $\pi(i$-1), $\pi(i$+1), ..., $\pi(N)$), jeżeli $i>j$

Ruch jest definiowany przez parę uporządkowaną (i,j) taką że $i,j \in \{1, ..., N\}$. Taka definicja ruchu została przyjęta również w [8]. Można ograniczyć zakres przeszukiwania, zwiększając szybkość pracy algorytmu. Pozwala nam na to założenie o spełnianiu nierówności trójkąta przez czasy przezbrojeń. Zdefiniujmy pojęcie ścieżki krytycznej dla naszych problemów.

Ścieżka krytyczna dla C_{max} – do ścieżki krytycznej należy ciąg zadań ($\pi(u)$, $\pi(u$+1), ..., $\pi(U)$), gdzie u jest największą pozycją w π, dla której $S_{\pi(u)} = r_{\pi(u)}$. Jeżeli takiego zadania nie ma w π, to przyjmujemy u =1. Indeks $U = N$.

Ścieżka krytyczna dla L_{max} – do ścieżki krytycznej należy ciąg zadań ($\pi(u)$, $\pi(u$+1), ..., $\pi(U$-1), $\pi(U)$). U jest najniższym indeksem w permutacji π, dla którego zachodzi $L_{\pi(U)} = L_{max}$ dla danej permutacji. Tutaj, jak wyżej, u jest największą pozycją w π taką, że $u<U$, dla której $S_{\pi(u)} = r_{\pi(u)}$ (jeżeli takiego zadania nie ma w π, to przyjmujemy u =1).

Dla problemu Σw_jC_j nie udało się sformułować podobnej własności i wszystkie zadania uważa się za należące do ścieżki krytycznej.

Jak łatwo zauważyć, usunięcie elementu $\pi(i)$, leżącego poza ścieżką krytyczną, nie powoduje zmniejszenia wartości funkcji kryterialnej. Wstawienie go do ścieżki krytycznej również nie spowoduje zmniejszenia wartości funkcji kryterialnej, ponieważ dla przezbrojeń spełniona jest nierówność trójkąta, jak to zaznaczyliśmy w sformułowaniu problemu (rozdz. 2). Ruchami, prowadzącymi do obniżenia wartości funkcji kryterialnej, są więc ruchy polegające na wyjęciu zadania ze ścieżki krytycznej i umieszczeniu go na innej wybranej pozycji w permutacji.

Takie ograniczenie liczby analizowanych ruchów pozwala zmniejszyć czas analizy otoczenia i zwiększyć tym samym efektywność algorytmu. Dalsze skrócenie czasu obliczeń możemy uzyskać na drodze przeglądania otoczenia w ustalonej kolejności, ponieważ w takim wypadku możliwe jest szybkie naliczenie wartości funkcji kryterialnej.

3.4. Szybkie wyliczenie wartości funkcji kryterialnej

Problem z kryterium C_{max}: Usuwamy z π wybrane zadanie. Następnie naliczamy dla tej permutacji od π(1) do $\pi(N$-1) parametry $S_{\pi(j)}$, $C_{\pi(j)}$, luz czasowy – $luz_{\pi(j)}$ – czas występujący

przed zadaniem, gdy maszyna nie wykonuje żadnego zadania ani przezbrojenia. Następnie od $\pi(N\text{-}2)$ do $\pi(1)$ wyliczamy zapas czasu $zap_{\pi(j)} = luz_{\pi(j+1)} + zap_{\pi(j+1)}$, przy czym $zap_{\pi(N-1)} = 0$ z definicji. Zapas czasu mówi nam, o ile możemy opóźnić termin rozpoczęcia wykonywania danego zadania, nie powodując zmiany wartości funkcji kryterialnej. Wartość przesunięcia, która przekracza zapas czasu, dodaje się bezpośrednio do wartości funkcji kryterialnej. Złożoność tych kroków jest rzędu $O(N)$. Ponieważ znamy terminy rozpoczęcia i zakończenia wykonywania zadań, to mając policzony zapas czasu, natychmiast jesteśmy w stanie podać wartość funkcji kryterialnej bez potrzeby naliczania kryterium do końca permutacji (czyli w czasie rzędu O(1)). Cała procedura łącznie ze wstawianiem ma więc złożoność $O(N)$. Należy zauważyć, że metoda ta jest skuteczna jedynie, gdy czasy przezbrojeń spełniają nierówność trójkąta, gdyż wtedy dodanie zadania do permutacji nie spowoduje zmniejszenia czasu rozpoczęcia żadnego z zadań pozostawionych wcześniej w π.

Problem z kryterium L_{max}: Metoda jest bardzo podobna. Naliczamy dodatkowo $L_{\pi(j)}$ dla każdego zadania i zapamiętujemy największy jako L_{max}. Zapasy czasu wyliczamy w następujący sposób od $\pi(N\text{-}2)$ do $\pi(1)$: $zap_{\pi(j)} = \min(L_{max} - L_{\pi(j)}, luz_{\pi(j+1)} + zap_{\pi(j+1)})$, przy czym $zap_{\pi(N-1)} = L_{max} - L_{\pi(N-1)}$. Zmiana kryterium następuje tak jak dla C_{max}, z tym że uwzględniamy dodatkowo jeszcze wartość L zadania wstawianego. Złożoność obliczeniowa tej procedury jest więc identyczna jak dla problemu C_{max} czyli $O(N)$.

Problem z kryterium $\Sigma w_j C_j$: Problem ten jest trudniejszy. Autorom nie udało się skonstruować algorytmu o złożoności $O(N)$. Prezentowana metoda ma złożoność obliczeniową rzędu $O(N^2)$. Jednak w praktyce jest znacznie szybsza od mechanicznego wstawiania zadania wyjętego i liczenia kryterium dla tak uzyskanej permutacji. Czas jej działania, jak pokażą wyniki eksperymentu obliczeniowego, jest porównywalny z metodami dla C_{max} i L_{max}. Tak jak poprzednio naliczamy $S_{\pi(j)}$, $C_{\pi(j)}$, $luz_{\pi(j)}$,oraz wartość kryterium. Definiujemy pojęcie bloku: blok stanowią zadania, między którymi nie występuje luz czasowy. Idea dalszych obliczeń opiera się na tym, że zadania w bloku, o ile są przesuwane, to są przesuwane o ten sam czas. Można więc wyznaczyć dla nich wspólną wagę. Tak więc buduje się tabelę, która dla danego zadania zawiera sumę jego wagi i zadań następujących po nim w bloku. Dodatkowo pamięta się informację, od jakiego indeksu zaczyna się następny blok. Wykonać to można w czasie rzędu $O(N)$. Przeliczając kryterium po wstawieniu zadania, dodajemy przesunięcie bloku przemnożone przez jego wagę i przechodzimy do następnego bloku. Czas przesunięcia następnego bloku modyfikujemy o luz czasowy przed nim występujący.

Postępowanie kończymy, gdy dojdziemy do końca permutacji lub luzy czasowe pochłoną przesunięcie.

3.5. Lista tabu i funkcja aspiracji

W prezentowanym algorytmie zastosowano dwie listy tabu.

Pierwsza z nich gromadzi wartości funkcji kryterialnej z ostatnich 15 iteracji.

Druga lista tabu koncepcyjnie jest zbliżona do przedstawionej w pracy [8]. Zamieszczone tam rozwiązanie pozwala wprowadzać ograniczenia kolejnościowe dla zadań i szybko testować, czy rozpatrywane rozwiązanie jest dopuszczalne. Umieszcza się tam pary zadań w takiej kolejności, w jakiej nie powinny one wystąpić w permutacji wynikowej, tzn. para zadań (a,b) nakłada status tabu na ruchy powodujące powstawanie permutacji, w których zadanie a występowałoby przed b (i to niekoniecznie bezpośrednio). Różnica w niniejszej pracy polega na tym, że umieszczana jest na liście tabu para $(\pi(i),\pi(j-1))$, jeżeli $i<j$ oraz $(\pi(j+1), \pi(i))$, jeżeli $i>j$. To rozwiązanie jest bardziej restryktywne w stosunku do generowanych rozwiązań niż prezentowane w [8] i algorytm wykorzystujący to rozwiązanie w testach próbnych zachowywał się lepiej. Długość listy tabu przyjęto 20 pozycji.

Funkcję aspiracji przyjęto na poziomie najlepszego znalezionego rozwiązania przez algorytm od momentu uruchomienia lub odtworzenia stanu z pamięci długoterminowej.

3.6. Kryterium stopu algorytmu

W zaimplementowanym algorytmie przyjęto, że maksymalna liczba iteracji wynosi 2000. Algorytm może zostać zatrzymany wcześniej, jeśli wartość funkcji kryterialnej bieżącego rozwiązania równa jest dolnemu oszacowaniu wartości optymalnej funkcji kryterialnej (patrz rozdz.4), co oznacza, że znaleziono rozwiązanie optymalne. Algorytm zatrzymywany jest również, jeżeli wyczerpała się zawartość pamięci długoterminowej oraz wykonano 200 iteracji od ostatniego odtworzenia stanu algorytmu z pamięci długoterminowej, podczas których nie uzyskano poprawy bieżącego rozwiązania w stosunku do odtworzonego. Długość pamięci długoterminowej przyjęto 3, natomiast liczbę nawrotów do danego stanu 2.

4. Eksperyment numeryczny

Dla generacji instancji przyjęto następujące zasady. Zmienne były losowane jako całkowitoliczbowe. Poniżej podano wartości ({}) lub przedziały ([]), do których należał parametr (generator liczb pracował wg rozkładu jednostajnego na danym przedziale).

$N = \{40, 200\}$ – liczba zadań,

$B=N/x_f$ – liczba rodzin, $x_f = \{10, 4\}$ – średnia liczba zadań należących do jednej rodziny,

$s_{max} = \{0, 10, 100\}$ – maksymalny czas przezbrojenia,

$s_{ab} \in [s_{max}/2, s_{max}]$ (aby spełnić nierówność trójkąta) dla $a \neq b$, $s_{ab} = 0$ dla a=b,

$p_j \in [1, 30]$

$AC=¼\, s_{max} B + 15\, N$ – średni czas zakończenia wykonywania wszystkich zadań dla $r_j = 0$,

$r_j \in [0, x_r\, AC]$ dla $x_r = \{0.0, 0.5, 1.0, 1.5\}$,

$d_j \in [0, x_d\, AC]$ dla $x_d = \{0.5, 1.0, 1.5\}$,

$w_j \in [1, 10]$

$f_j \in [1, B]$

Zmienne wyróżnione przez podkreślenie stanowiły parametry generacji instancji. Jak łatwo zauważyć, ich kombinacja daje 144 warianty. Do testów użyto 132 warianty. Pominięto warianty z x_r=0 i s_{max}=0, które z założenia są rozwiązywane optymalnie i w czasie wielomianowym przez algorytmy przybliżone zaimplementowane w fazie inicjacji, np.: dla kryterium L_{max} wariant ten daje problem $1||L_{max}$, który algorytm Schrage rozwiązuje optymalnie. W ten sposób nie uzyskujemy sztucznej poprawy rezultatów eksperymentu. Dla każdego wariantu wygenerowano 10 instancji, co daje ogólnie 1320 testowanych instancji.

Algorytm zaimplementowano w języku C++. Obliczenia wykonano w arytmetyce całkowitoliczbowej na komputerze klasy PC z procesorem Pentium MMX 233 MHz.

Aby ocenić jakość wyników uzyskiwanych przez TS, rezultaty porównano z dolnym oszacowaniem funkcji celu (ang. *Lower Bound* - LB). Dla poszczególnych kryteriów mają one następującą postać.

LB dla C_{max} – Etap I: Ustalamy czas rozpoczęcia szeregowania zadań na $t = \min s_{0b}$, gdzie $b = 1,...B$. Dopóki $t < r_{max} = \max r_j$, dla $j = 1, ...,N$, zadania szeregujemy wg niemalejących parametrów r_j, nie uwzględniając czasów przezbrojeń. Po uszeregowaniu kolejnego zadania ustalamy wartość t równą czasowi zakończenia wykonywania tego zadania. Gdy $t \geq r_{max}$ zmieniamy regułę postępowania. Etap II: Do czasu t dodajemy czasy wykonywania wszystkich zadań, które nie zostały uszeregowane w etapie I. Następnie

doliczamy czasy przezbrojeń tych zadań w sposób następujący: Niech b przebiega indeksy rodzin zadań uwzględnianych w etapie II. Wyliczamy wartości $s_b = \min s_{ab}$ dla a =1, ..., B. Do czasu t dodajemy wszystkie te wartości z wyjątkiem największej (przyjmujemy w ten sposób, że kontynuujemy wykonywanie zadań tej rodziny po etapie I). Otrzymany w ten sposób czas t traktujemy jako LB.

LB dla L_{max} – Ustalamy czas rozpoczęcia szeregowania zadań na $t = \min s_{0b}$, gdzie $b = 1,...B$. Następnie szeregujemy zadania używając modyfikacji algorytmu Schrage (np.[2]) dla problemu wyjściowego zrelaksowanego do problemu bez czasów przezbrojeń i z dozwolonym przerywaniem wykonywania zadań ($1|prmt, r_j | L_{max}$). Problem ten jest rozwiązywalny optymalnie w czasie O(n lg n) [2]. W danej chwili czasowej wykonywane jest dostępne zadanie o najmniejszym pożądanym terminie zakończenia wykonywania. Wynik uzyskany przez ten algorytm traktowany jest jako LB.

LB dla Σw_jC_j – Wybierany jest maksymalny wynik z dwóch algorytmów LB1 i LB2.

Algorytm LB1: Ustalamy czas rozpoczęcia szeregowania zadań na $t = \min s_{0b}$, gdzie $b = 1,...B$. Dokonujemy relaksacji problemu wyjściowego do problemu $1|| \Sigma w_jC_j$, który jest rozwiązywany optymalnie poprzez szeregowanie z użyciem reguły Smitha w czasie O(n lg n) (np. [7]). Reguła Smitha polega na uszeregowaniu zadań wg niemalejących ilorazów p_j/w_j.

Algorytm LB2: Dopuszczamy możliwość wykonywania zadań równolegle i ustalamy termin rozpoczęcia wykonywania zadań $S_j = \max(s_{0,(j)}, r_j)$ dla $j = 1, ...,N$. Dla tak dokonanego harmonogramowania wyliczamy wartość funkcji kryterialnej.

Rezultaty eksperymentu numerycznego zebrano w tabelach od 1 do 3. Ze względu na dużą liczbę wariantów nie są prezentowane uśrednienia wyników dla wszystkich wariantów osobno. Prezentujemy uśrednienia dla parametrów generacji instancji, aby pokazać ich wpływ na rezultaty osiągane przez algorytm tabu search.

Dalej będziemy oznaczać: TS – wynik uzyskany przez algorytm tabu search, H – wynik uzyskany przez algorytm przybliżony użyty w fazie inicjacji dla danego problemu, LB – wartość dolnego oszacowania funkcji celu dla danego problemu.

Poszczególne kolumny w tabelach oznaczają:

Parametr –parametr generacji instancji, dla którego wyliczono dane uśrednione,

$\rho_{TS} = (TS\text{-}LB)/LB*100\%$ – górne oszacowanie błędu względnego dla TS,

$\rho_H = (TS\text{-}LB)/LB*100\%$ – górne oszacowanie błędu względnego dla H,

$\delta = (TS\text{-}LB)/(H\text{-}LB)*100\%$ (jeżeli $H = LB$, to przyjmowano $\delta = 0$),

i_{best} – numer iteracji TS, w której znaleziono najlepsze rozwiązanie,

i_{end} – liczba iteracji wykonanych przez algorytm,

t_{best} – czas znalezienia najlepszego rozwiązania,

t_{end} – czas pracy algorytmu,

l. ins. – liczba instancji, dla której dokonano uśrednienia w poszczególnym wierszu.

Wszystkie parametry zostały policzone dla każdej instancji z osobna, a następnie uśrednione (dlatego wartości δ w tabeli nie są prostym ilorazem wartości ρ_{TS}/ρ_{H} z danego wiersza). Jak wspomniano wcześniej, przy liczeniu statystyk pominięto warianty, które z założenia są rozwiązywane w czasie wielomianowym przez algorytmy przybliżone zaimplementowane w fazie inicjacji, dlatego parametr „l. ins." w tabelach przybiera różne wartości w obrębie jednego parametru.

Dla problemu z kryterium L_{max} (tabela 2) parametry ρ_{TS} i ρ_{H} nie mają sensu, ponieważ funkcja kryterialna może przybierać wartości tak dodatnie, jak i ujemne.

5. Wnioski

Z przekrojowych wyników zamieszczonych w tabelach wynika, że prezentowany algorytm jest stosunkowo dobrym narzędziem optymalizacyjnym. W większości przypadków wnosi on bardzo istotne poprawy w stosunku do rozwiązań uzyskiwanych przez algorytmy heurystyczne. Rozpatrując te wyniki, należy pamiętać, że wyniki uzyskane przez algorytm TS porównywane były z dolnym oszacowaniem wartości optymalnej funkcji celu. Oszacowanie to wnosi błąd, różny w zależności od wariantu generowanych instancji. Autorzy wykonali również statystyki dla każdego wariantu osobno (ze względu na ograniczenie objętości pracy nie mogą być one szczegółowo prezentowane). Wynika z nich, że wartości średnie wskaźników jakości rozwiązań poszczególnych wariantów wynoszą maksymalnie: dla C_{max} – ρ_{TS} = 91%, dla L_{max} – δ = 78%, oraz dla $\Sigma w_j C_j$ – ρ_{TS} = 183%. Ten ostatni przypadek zachodzi dla wariantu bardzo niekorzystnego ze względu na jakość LB.

Należy zauważyć, że liczba zadań ma niewielki wpływ na wyniki obliczeń, natomiast wpływa oczywiście na czas obliczeń. Średnia liczba zadań należących do rodziny nie wpływa zasadniczo ani na jakość rozwiązań, ani na czas obliczeń. Natomiast rozkład terminów dostępności zadań, pożądanych czasów zakończenia wykonywania oraz maksymalny czas przezbrojenia wywierają wpływ, tak na czas obliczeń, jak i na wartość wskaźników jakości algorytmu TS.

Tabela 1

Wyniki eksperymentu numerycznego dla kryterium C_{max}

Parametr	ρ_{TS} [%]	ρ_{H} [%]	δ [%]	i_{best}	i_{end}	t_{best} [s]	t_{end} [s]	l. ins.
N=40	9.20	17.69	37.23	502	1301	0.71	1.82	660
N=200	11.80	19.69	43.26	899	1480	20.82	31.88	660
x_f=10	8.16	15.16	42.47	645	1363	9.79	16.06	660
x_f=4	12.85	22.22	38.03	757	1418	11.74	17.63	660
s_{max}=0	0.11	5.95	0.95	105	284	1.03	3.04	360
s_{max}=10	5.62	16.03	34.65	1071	1892	17.85	23.60	480
s_{max}=100	23.18	30.90	75.32	777	1719	10.98	20.46	480
x_r=0.0	3.73	7.26	65.48	672	1478	12.82	23.74	240
x_r=0.5	18.59	30.09	36.55	747	1353	12.99	19.08	360
x_r=1.0	13.51	26.03	37.34	907	1570	12.67	18.24	360
x_r=1.5	3.94	7.58	30.03	467	1191	5.27	8.64	360

Tabela 2

Wyniki eksperymentu numerycznego dla kryterium L_{max}

Parametr	δ [%]	i_{best}	i_{end}	t_{best} [s]	t_{end} [s]	l. ins.
N=40	9.38	285	835	0.51	1.31	660
N=200	8.49	505	721	20.76	27.25	660
x_f=10	8.70	417	844	11.19	15.31	660
x_f=4	9.17	373	712	10.08	13.24	660
s_{max}=0	1.17	1	33	0.00	0.11	360
s_{max}=10	15.87	326	775	9.58	13.94	480
s_{max}=100	7.83	759	1340	19.66	25.24	480
x_r=0.0	30.31	1022	1921	31.80	43.35	240
x_r=0.5	8.44	503	854	12.53	15.81	360
x_r=1.0	1.99	192	442	3.84	5.18	360
x_r=1.5	2.12	71	275	1.42	2.46	360
x_d=0.5	11.38	611	1063	15.13	19.96	440
x_d=1.0	7.23	386	738	11.31	14.79	440
x_d=1.5	8.21	187	532	5.47	8.08	440

Tabela 3

Wyniki eksperymentu numerycznego dla kryterium $\Sigma w_i C_i$

Parametr	ρ_{TS} [%]	ρ_{H} [%]	δ [%]	i_{best}	i_{end}	t_{best} [s]	t_{end} [s]	l. ins.
N=40	35.43	100.80	59.08	226	1964	0.89	7.93	660
N=200	39.18	132.00	57.39	972	1999	64.80	158.29	660
x_f=10	34.30	124.17	55.83	590	1983	31.40	81.28	660
x_f=4	40.32	108.63	60.64	609	1980	34.29	84.94	660
s_{max}=0	10.45	10.69	93.36	165	1933	9.98	114.75	360
s_{max}=10	22.32	38.28	62.59	735	1999	44.11	83.35	480
s_{max}=100	72.44	273.81	27.52	789	1999	38.73	59.14	480
x_r=0.0	74.49	297.34	30.50	759	1998	41.61	63.00	240
x_r=0.5	62.59	152.00	67.99	552	1938	27.57	74.53	360
x_r=1.0	17.95	52.93	64.59	630	1999	31.57	75.59	360
x_r=1.5	6.60	23.64	60.61	509	1995	33.55	112.61	360

Podsumowując można stwierdzić, że prezentowany algorytm nadaje się do rozwiązywania problemów o rozmiarze małym i średnim, dając stosunkowo dobre rozwiązania.

LITERATURA

1. Błażewicz J.: *Złożoność obliczeniowa problemów kombinatorycznych*, WNT, Warszawa 1988.
2. Carlier J.: *The one-machine squencing problem*, Europ. Journal of Operational Research 11, p. 42-47, 1982.
3. Chudzik K., Janiak A.: *Single Machine Schedduling with Job Ready and Setup Times – Genetic Approach*, Proceedings of the Fourth International Symposium on Methods and Models in Automation and Robotics, Miedzyzdroje, Poland, p. 1071-1075, 1997.
4. Chudzik K., Janiak A.: *Szeregowanie zadań z przezbrojeniami maszyn: przegląd zagadnień i literatury*, Automatyka, Tom 1, Zeszyt 1, Wydawnictwo AGH, Kraków, 1997, s. 71-79.
5. Glover F.: *Tabu Search: Part I*, ORSA J. Computing 1, p. 190-206, 1989.
6. Glover F.: *Tabu Search: Part II*, ORSA J. Computing 2, p. 4-32, 1990.
7. Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Shmoys D.B.: *Sequencing and scheduling algorithms and complexity*, Report BS-R8909, Centre for Mathematics and Computer Science, Amsterdam, 1989.
8. Nowicki E., Zdrzałka S.: *Single Machine Scheduling with Major and Minor Setup Times: A Tabu Search Approach*, Journal of the Operational Research Society 47, p. 1054-1064, 1996.

Recenzent: Prof.dr hab.inż Jerzy Klamka

Abstract

The paper is devoted to the single machine scheduling problems with sequence dependent setup times. Processing and ready times, due dates and weights are given for each job. A solution method of three optimisation problems is presented in the paper. The problems are to find such permutations of jobs that following criterion functions are minimised: 1) maximum completion time (makespan), 2) maximum lateness, and 3) weighted sum of completion times. Presented problems are NP-hard. A tabu search algorithm was used to solve problems under consideration. Efficient techniques of search in neighbourhood were constructed for this algorithm. The algorithm was tested for many instances with randomly generated job parameters and setup times. The results of the computational experiments and some conclusions are also given.