ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2002
Seria: AUTOMATYKA z. 134 Nr kol. 1554

Jakub BAUMAN, Joanna JÓZEFOWSKA
Politechnika Poznańska

SZEREGOWANIE ZADAŃ O ZNANEJ SEKWENCJI Z UWZGLĘDNIENIEM KOSZTÓW WYPRZEDZEŃ I OPÓŹNIEŃ

Streszczenie. W pracy rozważa się problem szeregowania zadań, o dowolnych oczekiwanych terminach zakończenia, na jednej maszynie z kryterium minimalizacji kosztów nieterminowości wykonania zadań. Funkcje kosztów wyprzedzeń i opóźnień są liniowe, asymetryczne i indywidualne dla poszczególnych zadań. Zaprezentowano algorytm o złożoności O($n\log n$) znajdujący optymalne uszeregowanie danej sekwencji zadań. Koncepcja polega na zastosowaniu funkcji określającej koszt modyfikacji uszeregowania, aktualizowanej po uszeregowaniu kolejnego zadania.

SCHEDULING A SEQUENCE OF JOBS WITH EARLINESS-TARDINESS COSTS

Summary. We consider one-machine scheduling problem with individual due dates to minimize earliness-tardiness cost. The functions of the earliness-tardiness cost are linear, asymmetric and task dependent. We propose an O($n\log n$) algorithm to find an optimal schedule for a given sequence of jobs. A concept of a schedule modification cost function updated after adding each task is used.

1. Wprowadzenie

W nowoczesnych systemach sterowania produkcją zwraca się uwagę zarówno na koszty magazynowania wyrobów, jak również na koszty związane z opóźnieniem dostaw. Taka koncepcja jest często nazywana produkcją dokładnie na czas (Just-in-time, JIT). Produkcja JIT spowodowała wprowadzenie nowej klasy problemów szeregowania zadań, gdzie karane są zarówno wyprzedzenia, jak i opóźnienia wykonania zadania. Koszty wyprzedzenia to np. koszty związane z magazynowaniem wyrobów gotowych, które zostały wyprodukowane przed oczekiwanym terminem, a koszty opóźnienia to np. koszty kar

umownych związanych z opóźnieniem dostaw. W ogólności funkcje kosztów mogą być zarówno liniowe, jak i nieliniowe. Celem jest minimalizacja sumarycznego kosztu.

W ogólnym przypadku problem jest NP-trudny nawet dla jednej maszyny, co pokazali Garey, Tarjan, Wilfong [2]. Problem był rozważany również w pracach Yano i Kim [3], Abdul-Razaq i Potts [4], Szwarc [5], Ow i Morton [6,7], Fry, Darby-Dowman i Armstrong [8]. Klasyfikację tego typu problemów podali Baker i Scudder [9].

W niniejszej pracy rozważa się problem z liniowymi funkcjami kosztów wyprzedzeń i opóźnień dla jednej maszyny.

Problem można podzielić na dwa podproblemy: znalezienia sekwencji zadań i znalezienia optymalnych czasów zakończenia zadań dla danej sekwencji. Optymalne uszeregowanie zadań przy danej sekwencji można znaleźć przez rozwiązanie problemu programowania liniowego. Garey, Tarjan, Wilfong [2] zaproponowali bardziej efektywną procedurę GTW dla szczególnego przypadku z niezależnymi od zadań i symetrycznymi kosztami wyprzedzeń i opóźnień. Procedurę tę można zaimplementować, aby działała w czasie $O(n\log n)$. Inne przypadki problemu rozważa Chrétienne [1]. Przedstawił on rozszerzenie GTW na przypadek asymetrycznych kosztów oraz zaproponował metodę dla przypadku asymetrycznych i zależnych od zadań kosztów, o złożoności $O(n^3\log n)$. W niniejszym artykule zaproponowano algorytm szeregowania dla tego samego przypadku o złożoności $O(n\log n)$.

W rozdziale 2 podano sformułowanie problemu i podano własności rozwiązania optymalnego. Następnie, w rozdziale 3, przedstawiamy funkcję wzrostu kosztów $K_k(x)$, która jest podstawą algorytmu przedstawionego w rozdziale 4. Na koniec, w rozdziale 5, przedstawiono wnioski i kierunki dalszych prac.

2. Sformułowanie problemu

Rozważa się n niepodzielnych zadań, które mają być szeregowane na jednej maszynie. Wszystkie zadania są dostępne w chwili zero. W każdej chwili na maszynie może być wykonywane co najwyżej jedno zadanie. Każde zadanie i scharakteryzowane jest przez czas wykonania p_i, $i = 1, ..., n$, żądany termin zakończenia d_i oraz współczynnik kosztów wyprzedzenia α_i i opóźnienia β_i. Koszt wyprzedzenia jest dodatni, gdy $d_i - C_i \geq 0$ (gdzie C_i oznacza termin zakończenia wykonywania zadania), w przeciwnym wypadku jest równy zero

i podobnie koszt opóźnienia jest dodatni, gdy $C_i - d_i \geq 0$, w przeciwnym wypadku jest równy zero. Zakładamy, że funkcje określające koszty wyprzedzeń i opóźnień są liniowe i zależne od zadań. Całkowity koszt uszeregowania S określa wzór:

$$f(S) = \sum_{i=1}^{n} (\alpha_i \max\{0, d_i - C_i\} + \beta_i \max\{0, C_i - d_i\}) \quad (1)$$

W przypadku gdy kolejność wykonania zadań jest znana, problem sprowadza się do znalezienia wektora optymalnych czasów zakończenia poszczególnych zadań. Wektor ten można znaleźć rozwiązując problem programowania liniowego.

Chrétienne [1] zaproponował algorytm o złożoności $O(n^3 \log n)$ do rozwiązania tego problemu. Poniżej przedstawiamy bardziej efektywny algorytm dla tego problemu, o złożoności $O(n \log n)$.

3. Koncepcja funkcji wzrostu kosztu uszeregowania

Załóżmy, że dla danej sekwencji zadań znamy optymalne uszeregowanie, i chcemy przesunąć ostatnie zadanie n w lewo, aby skrócić uszeregowanie. Przesunięcie w lewo ostatniego zadania jest dopuszczalne tylko wtedy, gdy w uszeregowaniu występuje przerwa. Poza tym, aby uszeregowanie pozostało dopuszczalne, przesuwanie ostatniego zadania może wymusić konieczność przesunięcia jego poprzedników. Oczywiście, przesunięcie każdego zadania w lewo zwiększa dodatkowo wartość całkowitego kosztu uszeregowania. Oznaczmy przez $K_k(x)$ całkowity koszt przesunięcia ostatniego (k-tego) zadania w lewo o x jednostek, uwzględniając koszt wymuszonego przesunięcia jego poprzedników.

Funkcja $K_n(x)$ może być znaleziona iteracyjnie podczas szeregowania zadań. Procedura jest następująca. Szeregujemy pierwsze zadanie. Jeżeli $x_1 = d_1 - p_1 > 0$, to uszeregowanie zadania takie, że $C_1 = d_1$, jest dopuszczalne i optymalne. Przesunięcie zadania w lewo o x powoduje wzrost kosztu zgodnie z liniową funkcją $K_1(x) = \alpha_1 x$, gdzie $0 \leq x = d_1 - C_1 \leq x_1$. Przesunięcie zadania w prawo powoduje wzrost kosztu zgodnie z liniową funkcją $K_1(x) = \beta_1(-x)$. Przesuwanie w prawo można pominąć, gdyż wobec dostawiania kolejnych zadań tylko z prawej strony nie spowoduje ono nigdy zmniejszenia wartości funkcji celu. W przypadku gdy $d_1 - p_1 < 0$, to pierwsze zadanie nie może być wykonane przed terminem, a w optymalnym uszeregowaniu $C_1 = p_1 > d_1$, zatem minimalny koszt jego opóźnienia wynosi $\beta_1(d_1 - p_1)$.

Przy szeregowaniu kolejnych zadań mogą wystąpić dwa przypadki: $d_k > C_{k-1} + p_k$ oraz $d_k < C_{k-1} + p_k$.

Rozważmy przypadek $d_k > C_{k-1} + p_k$. Uszeregowanie zadania k takie, że $C_k = d_k$, jest dopuszczalne i nie powoduje dodatkowego wzrostu całkowitego kosztu. Jednakże musimy zaktualizować funkcję $K_k(x)$, ponieważ została rozszerzona jej domena. Obliczamy $x_k = d_k - p_k - C_{k-1}$. Przesuwanie zadania k w lewo o $x \in [0, x_k]$ powoduje koszt wyprzedzenia tylko zadania k, który wynosi $\alpha_k x$. Przesunięcie o więcej niż x_k wymaga przesunięcia poprzednich zadań, co powoduje większy koszt. Oznaczmy $X_k = \sum_{i=1}^{k} x_i$. W przedziale $[x_k, X_k]$ zachodzi $K_k(x) = \alpha_k x + K_{k-1}(x)$.

Teraz rozważmy przypadek $d_k < C_{k-1} + p_k$. Uszeregujmy zadanie k tak, że $C_k = C_{k-1} + p_k$ i obliczmy $x_k = C_{k-1} + p_k - d_k$ oznaczające rozmiar opóźnienia k-tego zadania. Funkcję kosztu uszeregowania k-tego zadania zilustrowano na rys. 1. Zatem:

$$K_k(x) = \begin{cases} -\beta_k x_k + K_{k-1}(x), \text{ dla } 0 < x < x_k \\ \alpha_k + K_{k-1}(x), \text{ dla } x > x_k \end{cases} \tag{2}$$

Jeżeli β_k jest wystarczająco duże, to przesuwanie k-tego zadania w lewo zmniejsza wartość funkcji $K_k(x)$. Ta sytuacja jest pokazana na rys. 1. W prezentowanym algorytmie zadanie jest przesuwane tak długo, dopóki nie wystąpi zwiększenie wartości funkcji $K_k(x)$.

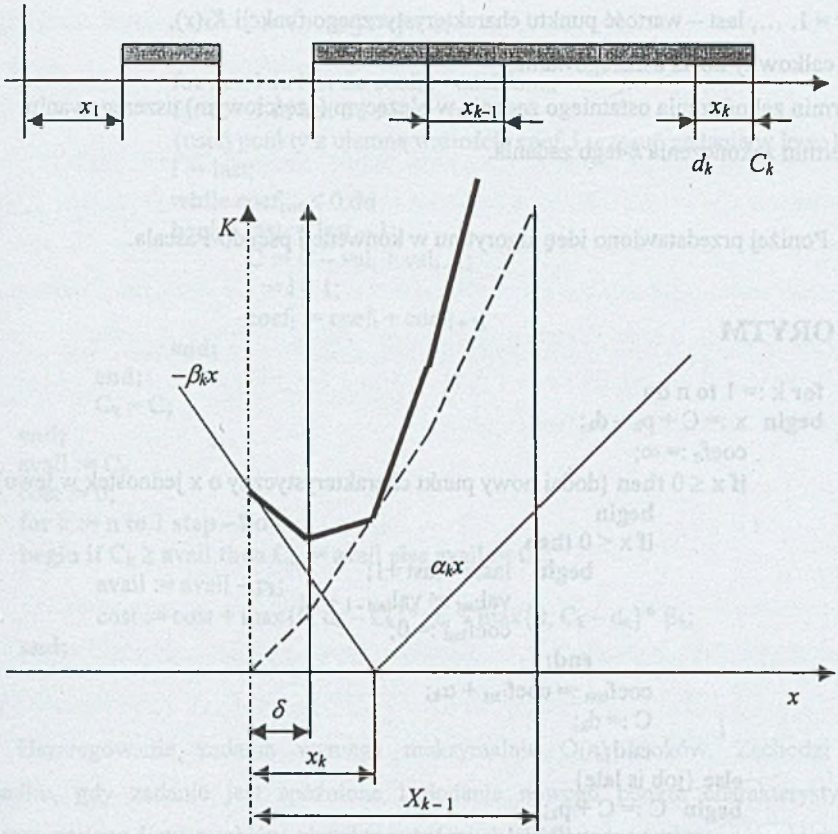
Można zaobserwować pewne istotne własności funkcji $K_k(x)$ przedstawione w poniższych lematach.

Lemat 1. Funkcja $K_k(x)$ jest przedziałami liniowa, ciągła i rosnąca, $k = 1, ..., n$.

Punkty, w których następuje zmiana współczynnika nachylenia funkcji $K_k(x)$, będziemy nazywać punktami charakterystycznymi tej funkcji.

Lemat 2. Zachodzą następujące własności: $K_k(x) \geq 0$, $x^* \leq x_k$, $x^* \beta_k \geq K_{k-1}(x^*)$, $k = 1, ..., n$, gdzie x^* jest punktem, w którym $K_k(x)$ osiąga minimum.

Lemat 3. Maksymalna liczba punktów charakterystycznych jest równa $n + 1$.



Rys. 1. Funkcja $K_k(x)$ dla k-tego zadania $d_k < C_{k-1} + p_k$ i dużego β_k

Fig. 1. Function $K_k(x)$ for the kth job $d_k < C_{k-1} + p_k$ and large β_k

4. Algorytm

Przyjmujemy, że porządek zadań jest znany. W każdej iteracji algorytm dodaje jedno zadanie do uszeregowania i znajduje optymalne rozmieszczenie dla zadań już uszeregowanych. Oznacza to, że w każdej iteracji znajdywane jest optymalne rozwiązanie dla podzbioru zadań. Ostatecznie otrzymywane jest uszeregowanie optymalne n zadań.

W opisie algorytmu posłużono się następującymi oznaczeniami:

last – liczba punktów charakterystycznych,

$coef_k$, $k = 1, \ldots,$ last – przyrost współczynnika funkcji $K_k(x)$ w k-tym punkcie charakterystycznym,

val_k, $k = 1, \ldots,$ last – wartość punktu charakterystycznego funkcji $K_k(x)$,

cost – całkowity koszt uszeregowania,

C – termin zakończenia ostatniego zadania w bieżącym (częściowym) uszeregowaniu,

C_k – termin zakończenia k-tego zadania.

Poniżej przedstawiono ideę algorytmu w konwencji pseudo-Pascala.

ALGORYTM

```
begin  for k := 1 to n do
       begin  x := C + p_k – d_k;
              coef_0 := ∞;
              if x ≤ 0 then {dodaj nowy punkt charakterystyczny o x jednostek w lewo}
                     begin
                     if x < 0 then
                            begin   last := last +1;
                                    val_last := val_last - 1 – x;
                                    coef_last := 0;
                            end;
                     coef_last := coef_last + α_k;
                     C := d_k;
                     end;
              else {job is late}
              begin  C := C + p_k;
                     y := max{0, val_last – x};
                     coef_last := coef_last – β_k;
                     i := 1; {podziel przedział zawierający y}
                     while val_i < y do
                     begin  valpom_i := val_i;
                     coefpom_i := coef_i;
                     i := i + 1;
                     end;
                     valpom_i := y;
                     coefpom_i := α_k + β_k;
                     if val_i = y then
                     begin  coefpom_i := coefpom_i + coef_i;
                            i := i + 1;
                            j := i;
                     end;
                     else
                     begin  last := last + 1;
                            j := i + 1;
                     end;
                     while j ≤ last do
                     begin  valpom_j := val_i;
                            coefpom_j := coef_i;
```

```
                    i := i + 1; j := j + 1;
                end;
                for i := 1 to last do coef_i := coefpom_i;
                for i := 1 to last do val_i := valpom_i;
                {usuń punkty z ujemną wartością coef_i i przesuń zadania w lewo}
                i := last;
                while coef_last ≤ 0 do
                begin last := last – 1;
                        C := C – val_i + val_i–1;
                        i := i – 1;
                        coef_i := coef_i + coef_i+1;
                end;
            end;
            C_k = C;
        end;
        avail := C_n
        cost := 0;
        for k := n to 1 step –1 do
        begin if C_k ≥ avail then C_k := avail else avail := C_k;
                avail := avail – p_k;
                cost := cost + max{0, d_k – C_k}* α_k + max{0, C_k – d_k}* β_k;
        end;
end;
```

Uszeregowanie zadania wymaga maksymalnie $O(n)$ kroków. Zachodzi to w przypadku, gdy zadanie jest spóźnione i dodanie nowego punktu charakterystycznego wymusza zmianę listy punktów charakterystycznych. Wobec tego złożoność obliczeniowa prezentowanej wersji algorytmu jest rzędu $O(n^2)$. Zastosowanie specjalnej struktury danych pozwala zmniejszyć tę złożoność do $O(n\log n)$. Wspomniana struktura danych zwana jest kopcem lub stertą. Tarjan [10] wykazał, że operacje takie jak *insert* i *delete_min* mogą być w niej wykonane w czasie $O(\log n)$. Prezentowana powyżej wersja algorytmu jest jednak bardziej czytelna i pozwala zilustrować jego główną ideę.

Twierdzenie 1

Algorytm znajduje optymalne uszeregowanie dla danej sekwencji zadań.

Dowód: Przez indukcję ze względu na k. Uszeregowanie jest na pewno optymalne dla $k = 1$, ponieważ pierwsze zadanie jest szeregowane na czas, gdy ($p_k \le d_k$) albo z minimalnym możliwym opóźnieniem ($C_k = p_k$). Pokażemy, że jeżeli twierdzenie jest prawdziwe dla k, to jest prawdziwe dla $k + 1$.

Nie wprost. Załóżmy że S_{k+1} jest uszeregowaniem otrzymanym z S_k zgodnie z algorytmem i że istnieje uszeregowanie S'_{k+1}, takie że $\text{cost}(S'_{k+1}) < \text{cost}(S_{k+1})$. Rozważymy dwa przypadki.

Przypadek 1. Jeżeli $C_k + p_{k+1} - d_{k+1} \le 0$, to koszt uszeregowania zadania $k+1$ wynosi zero, ponieważ $C_{k+1} = d_{k+1}$. Zatem z założenia $\text{cost}(S'_{k+1}) < \text{cost}(S_{k+1}) = \text{cost}(S_k)$. Usuwając $(k+1)$-sze zadanie z uszeregowania S'_{k+1} otrzymamy uszeregowanie S'_k spełniające nierówności $\text{cost}(S'_k) \le \text{cost}(S'_{k+1}) < \text{cost}(S_{k+1}) = \text{cost}(S_k)$, co jest sprzeczne z założeniem, że S_k jest optymalne dla k zadań. Zatem twierdzenie jest prawdziwe dla Przypadku 1.

Przypadek 2. Jeżeli $x_{k+1} = C_k + p_{k+1} - d_{k+1} > 0$, czyli zadanie $k+1$ jest spóźnione. Funkcja $K_k(x)$ jest definiowana następująco:

$$K_{k+1}(x) = K_k(x) + \begin{cases} -x \cdot \beta_{k+1} & \text{gdy } x < x_{k+1} \\ (x - x_{k+1})\alpha_{k+1} & \text{gdy } x \ge x_{k+1} \end{cases} \tag{3}$$

Oznaczmy przez x^* długość przedziału, o który $(k+1)$-sze zadanie jest przesuwane w lewo w uszeregowaniu S_k. Zauważmy, że $0 \le x^* = C_k + p_{k+1} - C_{k+1}$. Zgodnie z algorytmem, x^* jest punktem, w którym funkcja $K_{k+1}(x)$ osiąga minimum. Z Lematu 2 wynika, że $x^* \le x_{k+1}$ i szeregowanie wg algorytmu daje $\text{cost}(S_{k+1}) = \text{cost}(S_k) + K_k(x^*) + (x_{k+1} - x^*)\beta_{k+1}$.

Jeżeli czas zakończenia $(k+1)$-szego zadania w uszeregowaniu S'_{k+1} jest większy niż $C_k + p_{k+1}$, to:

$$\text{cost}(S'_k) + x_{k+1}\beta_{k+1} \le \text{cost}(S'_{k+1}) < \text{cost}(S'_{k+1}) = \text{cost}(S_k) + K_k(x^*) + (x_{k+1} - x^*)\beta_{k+1} \tag{4}$$

czyli:

$$\text{cost}(S'_k) < \text{cost}(S_k) + [K_k(x^*) - x^*\beta_{k+1}] \tag{5}$$

Z Lematu 2 wiemy, że $x^*\beta_{k+1} \ge K_k(x^*)$, zatem z (5) $\text{cost}(S'_k) < \text{cost}(S_k)$, co jest sprzeczne z założeniem indukcyjnym, czyli w tym przypadku twierdzenie jest prawdziwe.

Jeżeli natomiast czas zakończenia $(k+1)$-szego zadania w uszeregowaniu S'_{k+1} jest mniejszy niż $C_k + p_{k+1}$, to rozważając ostatni blok B w uszeregowaniu S_{k+1}, czyli zbiór zadań uszeregowanych za ostatnią przerwą czasową w S_{k+1} otrzymamy:

$$\text{cost}(S_{k+1}) = \text{cost}(S_{k+1} \backslash B) + \text{cost}^*(B). \tag{6}$$

Rozważmy teraz uszeregowanie S'_{k+1}. Wiemy że:

$$\text{cost}(S'_{k+1}\backslash B) + \text{cost}(B) = \text{cost}(S'_{k+1}) < \text{cost}(S_{k+1}) \quad (7)$$

Z Lematu 2 wynika, że $\text{cost}^*(B)$ jest minimalny, czyli $\text{cost}^*(B) \leq \text{cost}(B)$. Zatem z (6) i (7) otrzymujemy:

$$\text{cost}(S'_{k+1}\backslash B) < \text{cost}(S_{k+1}\backslash B) \quad (8)$$

To jest sprzeczne z założeniem indukcyjnym, zatem twierdzenie jest prawdziwe również w tym przypadku, co kończy dowód.

5. Podsumowanie

W artykule zaproponowaliśmy algorytm o złożoności $O(n\log n)$ do rozwiązywania problemu szeregowania danej sekwencji niepodzielnych zadań z indywidualnymi żądanymi terminami zakończenia dla kryterium minimalizacji całkowitego kosztu opóźnień i wyprzedzeń. Rozważane funkcje kosztów są liniowe, a ich współczynniki dane i różne dla poszczególnych zadań. Proponowany algorytm o złożoności $O(n\log n)$ jest bardziej efektywny niż algorytm o złożoności $O(n^3\log n)$ zaproponowany przez Chrétienne'a [1]. Przedstawiony algorytm będzie wykorzystany w procedurach podziału i ograniczeń, jak również przeszukiwania tabu zastosowanych do znajdywania optymalnej sekwencji zadań.

LITERATURA

1. Chrétienne P.: Minimizing the Earliness and Tardiness Costs of a Sequence of Tasks on a Single Machine, Research Report Laboratoire d'Informatique de Paris 6, 1999.
2. Garey M., Tarjan R., Wilfong G.: One-processor scheduling with symmetric earliness and tardiness penalties, Mathematics of Operations Research, vol. 13, 1988, s. 330-348.
3. Yano C., Kim Y.: Algorithms for Single Machine Scheduling Problems Minimizing Tardiness and Earliness. Technical Report #86-40, Department of Industrial Engineering, University of Michigan, Ann Arbor, 1986.
4. Abdul-Razaq T., Potts C.: Dynamic Programming State-Space Relaxation for Single-Machine Scheduling. Journal of the Operational Research Society vol. 39, 1988, s. 141-152.

5. Szwarc W.: Minimizing Absolute Lateness in Single Machine Scheduling With Different Due Dates. Working Paper, University of Wisconsin, Milwaukee, 1988.
6. Ow P., Morton T.: Filtered Beam Search in Scheduling. International Journal of Production Research vol. 26, 1988, s. 35-62.
7. Ow P., Morton T.: The Single Machine Early-Tardy Problem. Management Science vol. 35, 1989, s. 177-191.
8. Fry T., Darby-Dowman K., Armstrong R.: Single Machine Scheduling to Minimize Mean Absolute Lateness. Working Paper, College of Business Administration, University of South Carolina, Columbia, 1988.
9. Baker K., Scudder G.: Sequencing with Earliness and Tardiness Penalties: A Review. Operations Research, vol. 38, 1990, s. 22-36.
10. Tarjan R. E.: Data structures and network algorithms. Society for Industrial and Applied Mathematics, Philadelphia, P.A., 1983.

Recenzent: Prof. dr hab. inż. Andrzej Świerniak

Abstract

In the paper a JIT scheduling problem is considered. This is a one-machine problem with linear cost functions that are task dependent and asymmetric. The problem consists of two sub-problems. The first one is to find a good sequence of jobs and the second one is to build an optimal schedule by inserting idle times. We consider the second sub-problem. It is known that it can be solved by a linear programming procedure or by an algorithm by Chrétienne [1] of complexity $O(n^3 \log n)$ A special case of the problem with symmetric and task-independent costs was considered by Garey, Tarjan and Wilfong [2] who proposed an $O(n \log n)$ procedure for this special case.

We propose an algorithm solving the general problem with arbitrary costs. The algorithm is working by iterative adding the jobs from a sequence. The base for the algorithm is a special function that describes a cost of modifications of the schedule. These modifications consist in shifting the last job in a partial sequence to keep an optimal solution during inserting next job to the schedule. The function has some interesting properties. Very important is that the function is increasing and piece-wise linear.

The algorithm we propose has the complexity $O(n \log n)$ and in this sense improves the algorithm proposed by Chrétienne [1]. We plan to use it in a branch and bound as well as in a tabu search procedure for finding optimal sequences of tasks.