マルチスロット環境下でのJust-In-Timeジョブの重み和最大化のためのヒューリスティクス

斉藤, 凌 / Saito, Ryo

(出版者 / Publisher)
法政大学大学院理工学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 理工学・工学研究科編
(巻 / Volume)
59
(開始ページ / Start Page)
1
(終了ページ / End Page)
4
(発行年 / Year)
2018-03-31
(URL)
https://doi.org/10.15002/00021621

マルチスロット環境下での Just-In-Time ジョブの重み和最大化のためのヒューリスティクス

HEURISTICS FOR MAXIMIZING THE TOTAL WEIGHT OF JUST-IN-TIME JOBS UNDER MULTI-SLOT CONDITIONS

斉藤凌
Saito Ryo

指導教員 千葉英史

法政大学大学院理工学研究科システム理工学専攻修士課程

In just-in-time scheduling, each job has to be completed exactly on its due date. Since just-in-time scheduling has applications to both manufacturing and computer systems, a number of research papers on just-in-time scheduling problems exist. Under multi-slot conditions, each job has one due date per time slot and has to be completed on one of its due dates. It is assumed that each job has a certain weight per time slot. The problem maximizing the total weight of just-in-time jobs under multi-slot conditions was recently proven to be NP-hard. In this paper, we present three heuristics for this problem. Moreover, we implement the three heuristics, and compare their performances from computational experimentation.

***Key Words** : Scheduling, algorithm, heuristics, just-in-time, time slot, weight*

1. はじめに

スケジューリング問題において，ジョブが納期までに終了することが求められる場合がある．その際，納期より早い処理の完了は，ジョブを保管する状況を生じさせ，そのためのコストが発生することが考えられる．このコストは，納期ちょうどに処理が完了すれば発生しない．本研究では，納期ちょうどに処理が完了される（Just-In-Time）ジョブに焦点を当てる．Just-In-Time スケジューリングの応用として，企業の生産性能の向上，コストを削減する生産システムの開発，人工衛星による効率的な観察などが考えられる．

関連するスケジューリング問題の既存研究において，機械数が 1 のとき，納期ズレのジョブ数を最小にするアルゴリズムが存在する [1]．この納期ズレのジョブ数の最小化問題は，Just-In-Time のジョブ数の最大化問題と同値である．また，同種並列機械上で，Just-In-Time のジョブ数の最大化問題に対するアルゴリズムが存在する [2]．既存研究 [1, 2] では，目的関数に影響しないという理由から，スケジュールを構成するとき，Just-In-Time で処理できないジョブは考慮されない．既存研究 [1, 2] では，Just-In-Time で処理できないジョブを認めるが，本研究ではすべてのジョブが Just-In-Time で処理できる状況が仮定される．そのために，マルチスロットが仮定される．マルチスロットとは，毎週，毎日などの周期的に繰り返す時間の単位に対応する．また，ジョブは，ジョブの価値としてスロットごとに重みを持っている．重み和を最大にする Just-In-Time スケジュールを求めたい．この問題は，NP 困難であることが知られている [3]．本研究は，この問題に対するヒューリスティクスを提案する．

本研究では，3 つのヒューリスティクスを提案する．1 つ目のヒューリスティクスは，単純な貪欲法である．2 つ目のヒューリスティクスは，重み付き区間スケジューリング問題に対するアルゴリズム [4] を使う．その際，ジョブと納期を，それぞれ重み付き区間スケジューリング問題での区間と終了時刻に対応させる．重み付き区間スケジュール問題では与えられた終了時刻ちょうどにジョブをスケジュールするため，重み付き区間スケジュール問題に対するアルゴリズムは，本研究での Just-In-Time スケジュールを求める際に使えると考えた．3 つ目のヒューリスティクスは，文献 [5, 6] の知見を用いたものである．また，提案する 3 つのヒューリスティクスを実装し，計算機実験を行う．計算機実験の結果から，3 つのヒューリスティクスの性能を考察する．

2. 問題定義

m 台の同種並列機械を $M_1, M_2, \ldots, M_m$ で表し，n 個のジョブを $J_1, J_2, \ldots, J_n$ で表す．各ジョブ J_i には，処理時間 p_i が設定される．各機械は，同時に 2 つ以上のジョブを処理することができない．各ジョブは，任意の機械で処理されるが，一旦処理を開始したら途中で処理を中断できないと仮定する．

全ての機械上の動作時間は，長さ L のタイムスロットで分割される．また，各ジョブは，周期的に繰り返すタイムスロット毎に納期が与えられる．すなわち，ジョブ J_i の納期を $d_i, L+d_i, 2L+d_i, \ldots$ とする．ただし，$d_i \leq L$ である．納期 d_i は，ジョブ J_i が第 1 タイムスロットで処理される場合の納期，$L+d_i$ は，第 2 タイムスロットで処理される場合の納期，以下同様である．全てのジョブは，必ずどこか 1 つのスロットの納期で Just-In-Time で処理されなければならない．

ジョブ J_i が l 番目のタイムスロットで処理完了するとき

の重みを $w_i(l)$ とする．この $w_i(l)$ は，非負の重み関数と仮定する．番号 l はスロットの番号を意味している．すなわち，$l = 1$ は，1 番目のスロットを意味し，$l = 2$ は，2 番目のスロットを意味している．すべてのジョブは，高々$\lceil \frac{n}{m} \rceil$ 番目のタイムスロットまでに処理完了することができ，$l \leq \lceil \frac{n}{m} \rceil$ と仮定する．

ジョブ J_i のスケジュールは，写像 $S : J_i \mapsto (M^S_{[i]}, C^S_i)$ で書ける．ここで，$M^S_{[i]}$ は J_i が処理される機械であり，C^S_i は J_i の処理完了時刻である．スケジュール S が以下の条件を満たす場合，S を実行可能と呼ぶ．

- 各ジョブ J_i において，制約式 $C^S_i = r^S_i \cdot L + d_i$ を満たす整数 $r^S_j \in \left\{0, 1, ..., \lceil \frac{n}{m} \rceil - 1\right\}$ が存在する．
- 任意の異なる 2 つのジョブ J_i, J_k に対して，$M^S_{[i]} = M^S_{[k]}$ ならば，$C^S_k - p_k \geq C^S_i$ または $C^S_i - p_i \geq C^S_k$．

対象の問題は以下のように書ける．

入力： ジョブ数 n，機械数 m，処理時間 p_i，納期 d_i，重み $w_i(l)$，タイムスロット長 L．

目的関数： $\sum_{i=1}^{n} w_i \left(\left\lceil \frac{C^S_i}{L} \right\rceil \right) \longrightarrow$ 最大．

最適スケジュールは，目的関数値を最大にする実行可能スケジュールである．機械数がジョブ数以上のとき，各機械が高々1 つのジョブを処理するスケジュールが存在するため，問題は簡単になる．それゆえ以下では，$n > m$ と仮定する．

3. ヒューリスティクス（その１）

1 つ目のヒューリスティクスは，単純な貪欲アプローチである．第 1 タイムスロットに注目する．第 1 タイムスロットでの重みに関して，ジョブをソートする．ソート順に，処理開始時刻か納期，もしくは両方を比べ，ジョブが機械 M_1 で処理可能かどうかを判定し，可能であれば M_1 でスケジュールする．具体的には，図 1 のように，ジョブ J_3 をジョブ J_1 とジョブ J_2 の間にスケジュール可能かを判定する場合，ジョブ J_1 の納期よりジョブ J_3 の処理開始時刻が大きく $(d_1 \leq s_3)$ かつ，ジョブ J_2 の処理開始時刻よりジョブ J_3 の納期が小さい $(d_3 \leq s_2)$ 場合，ジョブ J_3 をスケジュールする．そうでなければ，未処理のジョブとして残り次の機械 M_2 以降で処理可能かどうかを判定する．第 1 タイムスロットのスケジュールが決まった後，第 2 タイムスロットでスケジュール可能かどうか判定される．第 2 タイムスロットにおいて，第 1 タイムスロットと同じ手順でソートと，判定を行い，第 2 タイムスロットのスケジュールを決める．以降，未処理のジョブがなくなるまで繰り返す．このアルゴリズムで求められる実行可能スケジュールのタイムスロット数は，高々$\lceil \frac{n}{m} \rceil$ である．

ヒューリスティクス 1

1. $l := 1, j := 1$，ジョブを未処理の状態とする．
2. ジョブを第 l タイムスロットの重み順にソートを行う．
3. 未処理の全てのジョブをソート順に，機械 M_j でジョブが処理できるかを判定し，できればそのジョブを機械 M_j で処理し，ジョブを処理の状態とする．そうでなければ，未処理のままとする．
4. $j < m$ で未処理のジョブが残っていれば，$j := j + 1$，step2 へ戻る
5. 未処理のジョブがあれば，$l := l + 1, j := 1$，step2 に戻る．なければ，終了．

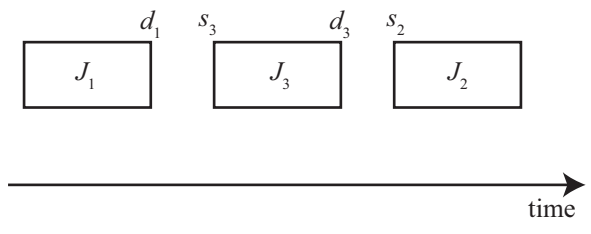

図 1 ジョブ J_3 が処理される場合の例

ヒューリスティクス 1 の時間計算量は，step2 が $O(n \log n)$，step3 が $O(n^2)$，step4 が $O(1)$，step1，5 が $O(1)$ である．step2，3，4 は高々$m \cdot \lceil \frac{n}{m} \rceil$ 回繰り返すため，全体の時間計算量は，$O(n^3)$ である．

4. ヒューリスティクス（その２）

2 つ目のヒューリスティクスは，重み付き区間スケジューリング問題に対するアルゴリズム [4] を用いたものである．区間スケジューリング問題を説明する．

(1) 区間スケジューリング問題

使用できる資源がある．例えば，教室や体育館等とする．多くの人がある期間，その資源を利用したいと要求している．要求は，時刻 s から時刻 f までその資源を利用したいが予約することができるか？という形式をしている．

区間スケジューリング問題は以下のように定義される．$1, 2, \cdots, n$ のラベルの付けられた n 個の要求が与えられる．各要求 i には，開始時刻 s_i と終了時刻 f_i が付随する．すべての要求 i で $s_i \leq f_i$ である．2 つの要求 i と j は，要求区間が交差しないとき，すなわち，要求 i の終了時刻が要求 j の開始時刻よりも早い $(f_i \leq s_j)$ か，要求 i の開始時刻が要求 j の終了時刻よりも遅い $(f_j \leq s_i)$ ときは，共存可能であるという．これを一般化し，要求の部分集合 A は，A の異なる 2 つのすべての要求 $i, j \in A$ が共存可能であるとき，共存可能であるという．目標は，最大の共存可能な部分集合を選ぶことである．

(2) 重み付き区間スケジューリング問題

区間スケジューリング問題に，重みを加え，より一般化した問題を考える．各要求 i には w_i の価値あるいは，重みが付随している．すなわち，i 番目の要求をスケジュールすれば，それだけ，価値が得られると考える．したがって，重み付き区間スケジューリング問題の目標は，共存可能な要求集合のうちで価値を最大にするものを求めることである．以下のように書ける．

入力： n 個の区間，各区間 i に対する開始時刻 s_i と終了時刻 f_i と重み w_i．

目的： 互いに交差しない区間の集合 $S \subset \{1, \ldots, n\}$ で，総重み $\sum_{i \in S} w_i$ の最大化．

対象の Just-In-Time スケジューリング問題を解くとき，重み付き区間スケジューリング問題のアルゴリズムの適用を考える．機械数 1 とし，ジョブを区間とみなし，ジョブの納期を区間の終了時刻とみなし，処理時間と納期から区間の開始時刻を決めれば，重み付き区間スケジューリング問題のアルゴリズムを適用できる．対象の Just-In-Time スケジューリング問題では，機械の台数が m で，すべてのジョブは高々$\lceil \frac{n}{m} \rceil$

番目のタイムスロットまでに完了できる．そのため，重み付き区間スケジューリング問題のアルゴリズムを高々$m \cdot \lceil \frac{n}{m} \rceil$回繰り返せば，対象の Just-In-Time スケジューリング問題を近似的に解くことができる．

ここで，重み付き区間スケジューリング問題に対する手法 [4] を説明する．P(i) とは，図 2 のように，ジョブを納期順にソートを行ったときに，J_i の処理開始時刻より前にある最も右よりのジョブ J_k の添え字と定義する．すなわち，P(i) = k である．OPT(i) をジョブ $J_1, J_2, \ldots, J_i$ に対する最適値とすると，漸化式 $\mathrm{OPT}(i) = \max(w_i + \mathrm{OPT}(\mathrm{P}(i)), \mathrm{OPT}(i-1))$ が成立つ．この式は，ジョブ J_i が最適解に含まれるかどうかを判定している．この式を含むアルゴリズム Compute-Opt(i) を以下に示す．

Compute-Opt(i)

IF $i = 0$ then
　　0 を返す
Else
　　$\max(w_i + \text{Compute-Opt}(\mathrm{P}(i)), \text{Compute-Opt}(i-1))$
Endif

Compute-Opt(i) を用いて，アルゴリズムを提案する．ただし，未処理の状態のジョブ数を n' とする．

ヒューリスティクス 2

1. $l = 1, j = 1, n' = n$, すべてのジョブを未処理の状態とする．
2. ジョブを納期の早い順にソートする．
3. 未処理の各ジョブに対して，P(i) を求める．
4. M_j において，Compute-Opt(n') を実行し，スケジュールと決定したジョブを処理の状態とし，n' の値を更新する．
5. $j < m$，未処理のジョブがあれば，$j := j + 1$，step2 に戻る．
6. 未処理のジョブがあれば，$l := l + 1$，$j = 1$，step2 に戻る．無ければ終了．

ヒューリスティクス 2 の時間計算量は，step2 が $O(n \log n)$，step3 が $O(n^2)$，step4 が $O(n)$，step1，5，6 は $O(1)$ である．step2，3，4 を高々$m \cdot \lceil \frac{n}{m} \rceil$回繰り返すため，全体の時間計算量は $O(n^3)$ である．

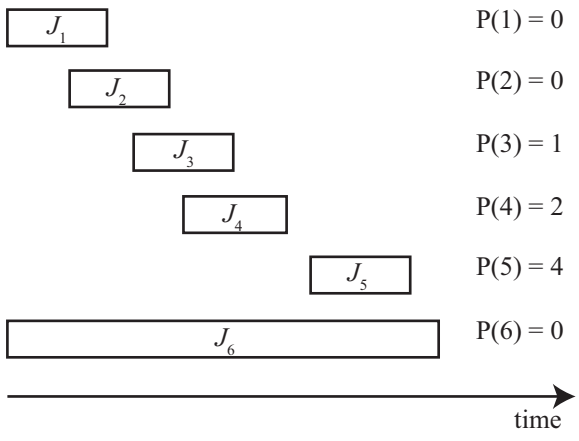

図 2 P(i) の値の例

5. ヒューリスティクス（その３）

3 つ目のアルゴリズムは，文献 [5, 6] で提案されたアイデアを使ったものである．重みを考慮しないとき，各機械で必要なタイムスロット数の最大値を最小にすることを目的とする問題が存在して，この問題を効率よく解くためのアルゴリズムも存在する [5].

また，重み付きマルチスロット Just-In-Time スケジューリング問題に対して，多項式時間で解ける場合の議論が存在し [6]，その概要は以下の通りである．

ジョブ集合 X の時間幅 $[s_X, f_X]$ は，

$$s_X = \min\{d_j - p_j | J_j \in X\},\ f_X = \max\{d_j | J_j \in X\}$$

を用いて，定義される．ジョブ集合 J の分割とは，

$$f_{X_1} \le s_{X_2}, f_{X_2} \le s_{X_3}, \ldots, f_{X_{l-1}} \le s_{X_l}$$

を満たすような互いに素な集合 $X_1, X_2, \ldots, X_l$ ($X_1 \cup X_2 \cup \cdots \cup X_l = J$) への分割のことである．

重み付きマルチスロット Just-In-Time スケジューリング問題の入力に対して，ジョブ集合 J の分割を考える．

2 つのジョブ J_i と J_j に対して，これらが共存可能であれば，これらを併合して新たなジョブ J_k とみなすことができる．具体的には，$d_i \le s_j$ のとき，J_k に関して以下のように決める．

$$p_k = d_j - d_i + p_i,$$
$$d_k = d_j,$$
$$w_k(l) = w_i(l) + w_j(l).$$

各分割集合に対して，併合できるジョブの組合せが 1 通りのとき，多項式時間で解くことができる [6].

ヒューリスティックアルゴリズム 3

1. 文献 [5] での手法を使って，タイムスロット数を最小にするスケジュールを求める．
2. 得られたスケジュールにおいて，各機械，各タイムスロットでジョブの併合を行う．
3. 文献 [6] での手法を使って，実行可能スケジュールを求める．

アイデアは，step1 で得られたスケジュールを，分割集合とみなすことである．つまり，step1 で得られたスケジュールで，各機械，各タイムスロットで処理されるジョブをまとめて，分割集合と考える．ヒューリスティクス 3 の時間計算量は，step1 が $O(n^2)$，step2 が $O(n)$，step3 が $O(n^4 \log m)$ である．全体の時間計算量は，$O(n^4 \log m)$ である．

6. 計算機実験

本研究で提案した 3 つのヒューリスティクスを実装し，計算機実験を行った．求めた値は，いずれも 100 回の平均を示している．以下の設定で行った．実験 1 の条件を以下に示す．

- ジョブ数 $n \in \{200, 300, \ldots, 2000\}$.
- 機械数 $m = 50$.
- $w_j(l) \in \{1, 2, \ldots, 10000\}$.
- $p_i \in \{1, 2, \ldots, 50\}$，$d_i \in \{p_i, p_i + 1, \ldots, 50\}$，$L = 50$.

実験結果を図 3 に示す．目的関数値はジョブ数の増加とともに，増加している．ヒューリスティクス 3 はヒューリスティクス 1, 2 より良い値を求める．ヒューリスティクス 3 は組合せを決め，その後，最適なタイムスロットに配置している．ヒューリスティクス 2 は，タイムスロット順に，機械 M_j でスケジュールを決めている．ヒューリスティクス 1 も同様に，タイムスロット順である．ヒューリスティクス 1，2 は，第 1 タイムスロットから順にスケジュールを決めていることから，タイムスロットが後ろの重みを考慮していない．ヒューリスティクス 3 は，後ろのタイムスロットの値も考慮し，スケジュールを決めているがことから，ヒューリスティクス 3 が良い値を求めると推測する．

実験 2 の条件，n，m，p_i，d_i，L の値は，実験 1 と同じである．重みに関して，l に関して非増加を付け加えた．

実験結果を図 4 に示す．目的関数値はジョブ数の増加とともに，増加している．ヒューリスティクス 1 はヒューリスティクス 2，3 より良い解を求める．ヒューリスティクス 2,3 は，少し対数グラフに近い形をしている．

実験結果より，ヒューリスティクス 1 は，大きい重みが前のタイムスロットにあるほど良い値を求めるアルゴリズムであると推測できる．その利点として，早く終了することが望まれるジョブが多い場合に利用できると考えられる．最後に，平均 CPU 時間を示す．条件は，実験 2 を行ったときのものである．実験 1，2 を通し，平均 CPU 時間に違いがなかったため，1 つだけ紹介する．結果は図 5 である．

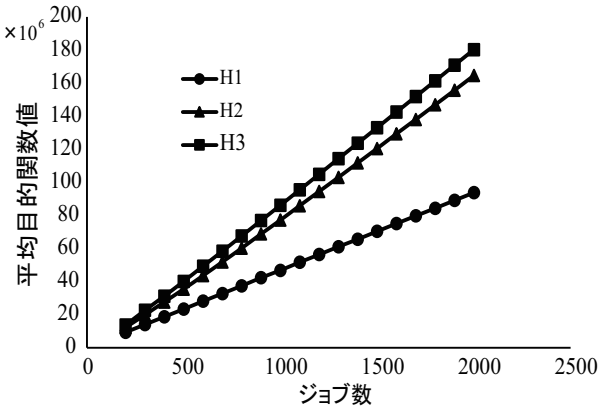

図 3 平均目的関数値の推移

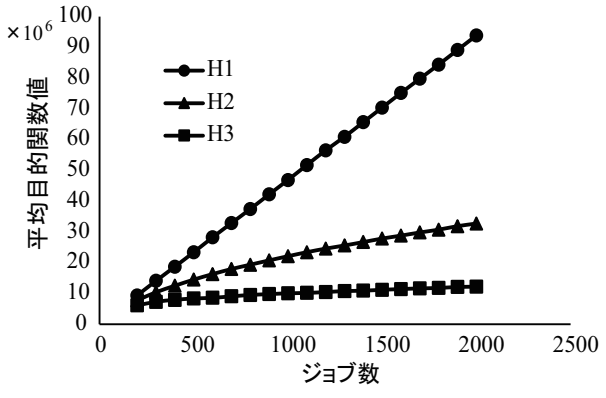

図 4 平均目的関数値の推移

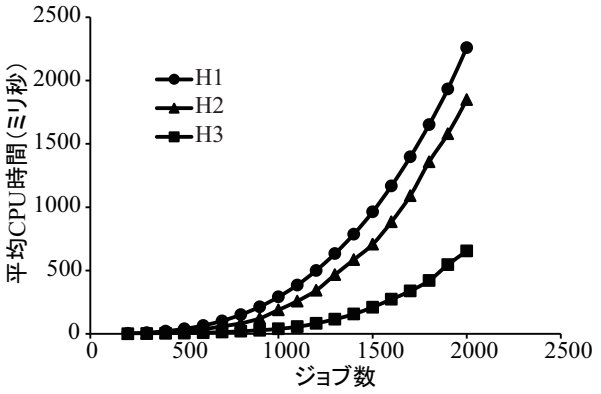

図 5 平均 CPU 時間の推移

平均 CPU 時間は，ジョブ数の増加によって，増加している．ヒューリスティクス 3 が，最も高速に解を求めている．

7. おわりに

マルチスロット環境下での Just-In-Time ジョブの重み和最大化問題に対し，3 つのヒューリスティクスを提案し，実装を行った．実験 1 から重みがランダムに設定している場合は，ヒューリスティクス 3 の方が良い解を求めることが分かり，実験 2 から重みがタイムスロットに関し，非増加である制約を加えるとヒューリスティクス 1 が良い解を求めることが分かった．また，平均 CPU 時間はヒューリスティクス 3 が良いと分かった．したがって，問題の入力により，それぞれ良いヒューリスティクスが異なるといえる．

重みがランダムである場合は，ヒューリスティクス 3 を適用し，重みがタイムスロットに関して非増加である場合，ヒューリスティクス 1 を適用することが良い．

参考文献

[1] A. Lann and G. Mosheiov, “Single machine scheduling to minimize the number of early and tardy jobs”, Computers & Operations Research, vol.23, no.8, pp.769-781, 1996.

[2] O. Čepek and S.C. Sung, “A quadratic time algorithm to maximize the number of just-in-time jobs on identical parallel machines”, Computers & Operations Research, vol.32, pp.3265-3271, 2005.

[3] E. Chiba and S. Imahori, “Maximizing the total weight of just-in-time jobs under multi-slot conditions is NP-hard”, IEICE Transactions on Information and Systems, vol.E99-D, no.2, pp.525-528, 2016.

[4] J. Kleinberg and É. Tardos, 「アルゴリズムデザイン」, 浅野孝夫, 浅野泰仁, 小野孝男, 平田富夫訳, 共立出版, 2008.

[5] O. Čepek and S.C. Sung, “Just-in-time scheduling with periodic time slots,” Scientiae Mathematicae Japonicae, vol.60, no.2, pp.295-301, 2004.

[6] E. Chiba, T. Kageyama, Y. Karuno, and H. Goto, “Maximizing the total weight value of just-in-time jobs in identical parallel machines with periodic time slots,” Proc. IEEE International Conference on Industrial Engineering and Engineering Management, pp.1349-1353, 2012.