情報処理学会論文誌　数理モデル化と応用　Vol. 1　No. 1　122–135 (Sep. 2008)

# 資源と作業順序の制約を拡張したスケジューリング問題に対する下界値の計算法

草部 博輝[†1]　中森 眞理雄[†1]

資源制約付きプロジェクトスケジューリング問題（Resource Constrained Project Scheduling Problem: RCPSP）は，多くの古典的スケジューリング問題の一般化されたモデルである．本稿は，利用可能な再生型資源量の時刻による変化と，各アクティビティが要求する再生型資源量の時間による変化を取り入れた，RCPSP/$\tau$ モデルに，タイムラグの概念を追加した拡張モデル，RCPSP/$\tau$+ モデルを取り扱う．本稿において，我々は RCPSP/$\tau$+ モデルに対する下界値の計算方法を提案する．提案する手法は，ラグランジュ緩和と線形計画問題への緩和を用いたものである．比較的小規模のインスタンスを用い，最適解との比較を行うことにより，得られた下界値の精度を評価する．

# Lower Bounding Method for the RCPSP/$\tau$ with Time Lags

Hiroaki Kusakabe[†1] and Mario Nakamori[†1]

Resource-constrained project-scheduling problem (RCPSP) is a general model of several classical scheduling models. In this paper, we suggest the scheduling model RCPSP/$\tau$+, which is added the time windows to the model of RCPSP/$\tau$ having the changing of limit of renewable resources in project term and of requirement of renewable resources in each activity's processing time. We present a lower bounding method for the RCPSP/$\tau$+ using lagrangean and linear programming (LP) relaxation and evaluate the lower bound accuracy comparing the oplitmal solution.

†1 東京農工大学大学院工学府
Graduate School of Engineering, Tokyo University of Agriculture and Technology

## 1. はじめに

20 世紀初頭からの生産活動の大規模化にともない，効率的な生産活動を行うためには良い生産スケジュールの立案が重要になってきた．ある現場に最新のハードウェアを導入したとしても，それを効率的に稼動させることができなければ，かえって費用の無駄を招くことになりかねない．また，新しいハードウェアを導入する前に，生産計画を見直すことにより，生産効率を向上させることも可能であろう．このように，今日の生産現場において，スケジューリング問題は最適化の対象になりうるものの 1 つである．生産スケジューリングの研究は多様な展開をなしとげ，その全貌は大きな広がりを持つに及んでおり，job-shop, flow-shop といった古典的なモデルに対しては，これまでに非常に多くの研究成果が報告されてきた．しかし，今日の生産現場では，これまでの少品種多量生産のスタイルから多品種少量生産への遷り変わり，生産資源としてのハードウェアの機能的進化といった点が多く見られる．ゆえに，生産ラインは非常に複雑になってきており，古典的な job-shop などにモデル化できないようなスケジューリングモデルも現れてきた．このような背景をふまえ，本稿では資源制約付きプロジェクトスケジューリング問題（Resource Constrained Project Scheduling Problem: RCPSP）の資源制約と先行制約を拡張した，RCPSP/$\tau$+ モデルを扱う．スケジューリング問題の多くのモデルは，クラス NP-困難に属するので，大規模な問題の最適解を現実的な時間内に得ることは非常に困難である．本稿で取り扱うスケジューリングモデルである RCPSP/$\tau$+ も，NP-困難に属する．クラス NP-困難に属する組合せ最適化問題に対する最適解を取得するためのアプローチとして，分枝限定法の実装があげられる．分枝限定法の性能は，いかに効率的な限定操作を行うかという点に依存し，これは下界値の精度に大きく関係する．また，最適解の取得にこだわらなければ，局所探索法や遺伝的アルゴリズムといった，現実的な時間内に良好な精度の実行可能解を取得するアルゴリズムの実装があげられる．最適解が未知であるインスタンスでの解の評価法として，実行可能解の目的関数値と下界値との比較がある．いずれの場合にも，下界値の精度の向上が必要である．文献 12) では，RCPSP/$\tau$+ に対し，ラグランジュ緩和や子問題への分割を用いた下界値計算法について述べているが，本稿は文献 12) の内容を発展させ，そこで触れなかった線形計画問題への緩和手法およびラグランジュ緩和と子問題への分割の併用を論じている．提案するアルゴリズムから得られる下界値の精度は，比較的小規模なインスタンスを用い，最適解との比較を行うことにより評価する．

## 2. 問題のモデル

本章では，RCPSP の基本モデル，資源の制約をより一般化した RCPSP/$\tau$ モデルおよび本稿で扱う変形モデルである RCPSP/$\tau$+ を説明する．

### 2.1 RCPSP

RCPSP は，job-shop スケジューリング問題の一般型モデルとしてよく知られている問題である[2]．RCPSP は次のようなモデルである．

プロジェクトは $n$ 個のアクティビティから構成され，各アクティビティには，$0, 1, \ldots, n-1$ の番号が与えられている．2 つのアクティビティ 0，$n-1$ はそれぞれ，プロジェクトの開始と完了を表すダミーである．各アクティビティには処理時間が与えられており，処理は中断されない．ただし，ダミーのアクティビティ 0 および $n-1$ の処理時間はともに 0 である．

アクティビティの処理実行のために，一般に複数種類の資源が提供される．これらの資源は再生型資源である．すなわち，使用回数の限度がなく，何度でも再利用可能である．各資源は一般に，種類ごとに複数個提供されており，供給量はプロジェクト期間中一定である．各アクティビティは，処理実行のためにこれら資源のいくつかを占有する．複数のアクティビティが，同時に 1 つの資源を利用することはできない．資源の占有量は種類ごとに一定であるが，各アクティビティで異なる．同時に処理実行されるアクティビティの資源占有量の合計が，供給量を超えてはならない．

ある 2 つのアクティビティ間には，先行制約が課せられることがある．2 つのアクティビティ $i$ と $j$ の間に先行制約が課せられ，$i$ が $j$ に先行するとは，$i$ の処理完了後でなければ $j$ が処理を開始できないことを表す．

RCPSP はこれらの資源制約，先行制約を満たし，プロジェクトの完了に必要な時間，すなわちアクティビティ $n-1$ の完了時刻を最小化する問題である．

また，再生型資源のみではなく，消費型資源の概念を導入したモデルもある．このモデルは，各アクティビティに，処理方法が複数種類与えられており，必要な消費型資源量が異なる．各処理方法をモードと呼ぶ．アクティビティは，複数の処理モードを持つので，このモデルは，マルチモード RCPSP（multi-mode resource constrained project scheduling problem: MMRCPSP）と呼ばれる．MMRCPSP の例として，次のような例がある．

プロジェクト全体で，ある一定量の燃料が使用可能であるとする．当然，燃料は 1 度使うと再利用できない．アクティビティを通常モードで処理する場合は，使用する燃料は少なくてすむ．しかし，高速モードで処理する場合は，より多くの燃料を必要とする．

このように，MMRCPSP には，消費型資源とアクティビティの処理時間との間にトレードオフが生じる．各アクティビティの処理モードが単数である場合は，MMRCPSP と区別するために，シングルモード RCPSP（single-mode resource constrained scheduling problem: SMRCPSP）と呼ぶこともある．SMRCPSP は，クラス NP-困難に属する[1] ことが知られている．SMRCPSP は MMRCPSP に帰着可能であるので，MMRCPSP も NP-困難である[3]．

### 2.2 RCPSP/$\tau$

変形モデルの 1 つに，RCPSP/$\tau$ がある．このモデルは，文献 3) で提案されており，RCPSP と比べて資源制約に違いがある．プロジェクト期間中，各資源の供給量は一定ではなく，時刻によって変化することがある．つまり，ある時刻 $t$ における資源 $r$ の使用可能量は，時刻 $t+1$ のそれと同じとは限らない．また，アクティビティが占有する資源量も一定ではなく，処理開始からの経過時間によって変化することがある．つまり，あるアクティビティが，処理開始から $u$ 単位時間経過後に占有する資源 $r$ の量が，$u+1$ 単位時間経過後のそれと同じとは限らない．これらの変化の様子は既知である．

### 2.3 RCPSP/$\tau$+

本節では，RCPSP/$\tau$+[11] モデルについて述べる．このモデルは RCPSP/$\tau$ から，先行制約に変化を与えたものである．先行制約を課せられた 2 つのアクティビティ間に，2 種類のタイムラグに関する制約が課されることがある．これらはそれぞれ，次のような制約である．2 つのアクティビティ $i$，$j$ を考える．これらには，$i$ が $j$ に先行するという制約が与えられている．アクティビティ $j$ は $i$ の処理完了後，ある猶予時間が経過するまでに処理を開始しなければならない．これを**猶予制約**（within constraint）と呼ぶ．また，アクティビティ $j$ は $i$ の処理完了後，ある待機時間が経過してからでないと処理を開始できない．これを**待機制約**（after constraint）と呼ぶ．猶予制約は，次のような場合に応用することが可能である．

- 鉄鋼業における延鉄工程において，素材の温度が下がる前に加工処理を行わなければならない．
- 化学薬品を塗布する工程において，塗布した薬品が変化を起こしてその働きを失う前に次の作業を行わなければならない．
- セメントを用いた作業を行う場合，セメントが乾燥して固まる前に塗りつけなければならない．

また待機制約は，次のような場合に応用することが可能である．

- 塗料やセメントを塗布した後，次の作業を行うためには乾燥を待たなければならない．

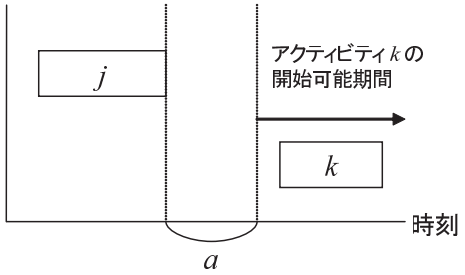


**図 1**　待機制約
Fig. 1　After constraint.

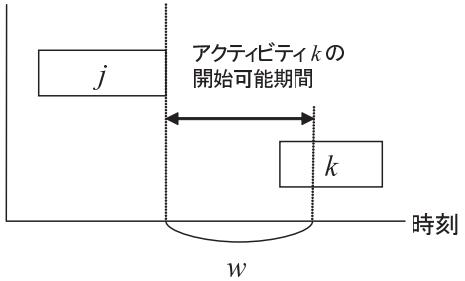


**図 2**　猶予制約
Fig. 2　Within constraint.

- 化学薬品の塗布の後，次の作業を行うためには薬品の定着を待たなければならない．

ここで，猶予制約および待機制約に対して，以下の記法を定義する．アクティビティ$j$，$k$間に待機制約が課せられ，$j$が$k$に先行する場合，$(j,k)$と表記する．RCPSPで用いられる先行制約は，$(j,k)$かつ待機時間が0である場合である．アクティビティ$j$，$k$間に猶予制約が課せられ，$j$が$k$に先行する場合，$\langle j,k\rangle$と表記する．$(j,k)$および$\langle j,k\rangle$のいずれも課されないのであれば，待機時間および猶予時間は定義されない．**図 1** に，$(j,k)$，待機時間$a$の場合の例を，**図 2** に，$\langle j,k\rangle$，猶予時間$w$の場合の例を示す．

アクティビティの処理時間，各時刻における利用可能資源量，アクティビティの処理中に占有される資源量，作業開始の順序関係，猶予時間および待機時間はすべて既知である．アクティビティ0の処理開始時刻は0であるものとする．目的関数は，メイクスパン，すなわちアクティビティ$n-1$の完了時刻の最小化である．RCPSP/$\tau$+は，先述の例にある鉄鋼業の延鉄スケジューリングだけではなく，半導体露光装置のスケジューリングや，塗料およびセメントを用いる工程のスケジューリングに応用することが考えられる．利用可能な資源を人間と考えた場合，勤務時間帯が複数種類設定されているような生産現場では時刻によって作業員の数が変化することが考えられる．また，必要な作業員数が変化するような作業も考えられる．このような場合にも，応用が可能である．

SMRCPSPは，RCPSP/$\tau$+の特殊ケースととらえることができる．RCPSPの先行制約は，待機時間0の待機制約および猶予時間が十分に大きな値である猶予制約と見なすことができる．このように，SMRCPSPはRCPSP/$\tau$+に帰着可能であるので，RCPSP/$\tau$+も，クラスNP-困難に属する．

### 2.4　既往の研究

RCPSPに対しては，今日までにさまざまな研究報告がなされてきた．分枝限定法を用いた手法に関する研究に，文献5)および文献8)がある．文献5)で用いられている下界値の計算方法は，資源制約を緩和してクリティカルパスを生成し，これに含まれないアクティビティを加えることにより，下界値を上昇させるというものである．また，文献8)では，資源制約のために同時に処理できない2つのアクティビティ$i$，$j$に対し，$i$が$j$に先行する場合とその逆との場合で2つの子問題に分割し，それぞれからクリティカルパスの長さを計算するという方法を用いている．基本的なスケジュール構築法であるpriority ruleに関する研究としては，文献6)があげられる．文献3)は，RCPSPに関する基礎的な事柄がまとめられている．また，解法として，主に遺伝的アルゴリズムが用いられている．文献7)では，ベンチマーク用インスタンス生成エンジンである，ProGenについて記述されている．これは，アクティビティ数，処理モード数，処理時間およびプロジェクト期間といった情報を与えることにより，シングルモードもしくはマルチモードのインスタンスを生成するエンジンである．ProGenから生成されたインスタンスを実験に用いた研究報告も存在する．

## 3.　0-1 整数問題への定式化

RCPSP/$\tau$+は，次のような0-1整数計画問題に定式化できる．

minimize

$$z = \sum_{t=0}^{t_{max}-1} t x_{n-1t} \tag{1}$$

subject to

$$\sum_{t=0}^{t_{max}-1} tx_{st} \leq \sum_{t=0}^{t_{max}-1} tx_{jt} + p_j + w_{js},\quad j \in J,\quad s \in S_j \tag{2}$$

$$\sum_{t=0}^{t_{max}-1} tx_{st} \geq \sum_{t=0}^{t_{max}-1} tx_{jt} + p_j + a_{js},\quad j \in J,\quad s \in S_j \tag{3}$$

$$\sum_{j=0}^{n-1} \sum_{u=0}^{\min(t,p_j-1)} d_{jru} x_{j(t-u)} \leq l_{rt},\quad t \in T, r \in R \tag{4}$$

$$\sum_{t=0}^{t_{max}-1} x_{jt} = 1,\quad j \in J \tag{5}$$

$$x_{jt} \in \{0,1\},\quad j \in J,\ t \in T \tag{6}$$

ただし，各記号の意味は，以下のとおりである．

- $n$：全アクティビティ数．
- $m$：全資源種類数．
- $J$：全アクティビティの集合．$J = \{0, \ldots, n-1\}$
- $R$：全資源種類の集合．$R = \{0, \ldots, m-1\}$
- $t_{max}$：プロジェクト期間．非負整数．
- $T$：離散で定義される時刻の集合．$T = \{0, \ldots, t_{max}-1\}$
- $p_j$：アクティビティ$j$ の処理時間．非負整数．
- $S_j$：アクティビティ$j$ の後続アクティビティの集合．
- $w_{js}$：アクティビティ$s \in S_j$ の処理開始の猶予時間．
  アクティビティ$s$ は，$j$ の処理完了後 $w_{js}$ 単位時間経過するまでに処理を開始しなければならない．
- $a_{js}$：アクティビティ$s \in S_j$ の処理開始の待機時間．
  アクティビティ$s$ は，$j$ の処理完了後 $a_{js}$ 単位時間経過しなければ処理を開始できない．
- $d_{jru}$：アクティビティ$j$ が，処理開始後，$u$ 単位時間経過時に要求する資源 $r$ の量．
- $l_{rt}$：時刻 $t$ における，資源 $r$ の利用可能量．
- $x_{jt}$：0−1 変数．アクティビティ$j$ が時刻 $t$ に処理を開始されるならば 1，それ以外は 0．

ノンプリエンプティブであれば，$\sum_{t=0}^{t_{max}-1} tx_{jt}$ がアクティビティ$j$ の処理開始時刻を表し，アクティビティ$n-1$ の処理時間は 0 なので，式 (1) はメイクスパンを表す．式 (2) および (3) はそれぞれ，猶予制約，待機制約である．式 (4) は資源制約である．また，式 (5) と (6) を合わせて，ノンプリエンプティブであることを示す．アクティビティ$j$，$k$ に対し $\langle j,k \rangle$ であり，そのほか特に断りがない場合は，$(j,k)$ かつ $a_{jk} = 0$ が同時に課せられているものとする．また，$(j,k)$ のみ課す場合は，$\langle j,k \rangle$ の $w_{jk}$ を十分大きな値にすればよい．

## 4. 下界値の計算

本章では，RCPSP/$\tau$+ に対する下界値の計算方法を示す．提案する手法は，猶予制約，待機制約および資源制約をラグランジュ緩和する手法と，資源制約，猶予制約および整数制約を緩和して線形計画問題を解くことによる手法である．

### 4.1 変数の固定

下界値取得の準備として，いくつかの変数を 0 もしくは 1 に固定する．次のような，RCPSP/$\tau$+ の緩和問題 R を考える．

minimize

(1)

subject to

(3), (5), (6)

$$\sum_{u=0}^{\min(t,p_j-1)} d_{jru} x_{j(t-u)} \leq l_{rt},\quad j \in J,\quad t \in T,\quad r \in R \tag{7}$$

制約式 (7) は，ある時刻 $t$ において処理中のアクティビティ$j$ の要求資源量が，供給可能量を上回ってはならないことを意味する．制約式 (4) と比べ，複数のアクティビティ間で資源の同時利用が可能であると考えればよい．R を，各アクティビティの処理開始時刻が最小になるように解く．制約 (4) によるオーバラップの制限を受けないため，アクティビティ$j$ の先行アクティビティの処理完了時刻を早くすれば，$j$ の処理開始を早くできる．R をこのように解き，各アクティビティ$j$ に処理開始時刻の下界 $lb_j$ を与える．R は，次のような手続きで解くことができる．アクティビティ0 に処理開始時刻 0 を与える．すべての先行アクティビティに処理開始時刻が与えられたアクティビティに対し，制約 (3) と制約 (7) を満た

```
procedure initialize()
begin
  for j := 0 to n − 1 do
  begin
   I*_j := φ;
   lb_j := NA;
  end
  lb_0 := 0;
  for each s ∈ S_0 do
   I*_s := I*_s ∪ {0};
end.
procedure set_start_time_lb()
begin
  initialize();
  for itr := 1 to n − 1 do
   for j := 1 to n − 1 do
   begin
    if (lb_j = NA) and (|I_j \ I*_j| = 0) then
     for t := max{lb_i + p_i + a_ij | i ∈ I*_j} to t_max − 1 do
      if satisfy(j, t) then
      begin
       lb_j := t;
       for each s ∈ S_j do
        I*_s := I*_s ∪ {j};
       break;
      end
   end
end.
```

**図 3**　アクティビティ処理開始可能時間の下界値の設定
Fig. 3　Setting lower bound of startable time of each activity.

```
procedure initialize()
begin
  for i := n − 1 down to 0 do
  begin
   S*_i := φ;
   ub_j := NA;
  end
  ub_{n−1} := UB;
  for each i ∈ I_{n−1} do
   S*_i := S*_i ∪ {n − 1};
end.
procedure set_start_time_ub()
begin
  initialize();
  for itr := 1 to n − 1 do
   for j := n − 2 down to 1 do
   begin
    if (ub_j = NA) and (|S_j \ S*_j| = 0) then
     for t := min{lb_s − p_j − a_js | s ∈ S*_j} down to 0 do
      if satisfy(j, t) then
      begin
       ub_j := t;
       for each i ∈ I_j do
        S*_i := S*_i ∪ {j};
       break;
      end
   end
end.
```

**図 4**　アクティビティ処理開始可能時間の上界値の設定
Fig. 4　Setting upper bound of startable time of each activity.

すような最小の処理開始時刻を与える．以上の手続きを，**図 3** に示す．$I_j$ は，アクティビティ $j$ の先行アクティビティの集合，$I_j^*$ は，$I_j$ の要素のうち，処理開始時刻の下界が与えられたものの集合である．手続き satisfy は，アクティビティ $j$ が時刻 $t$ に処理を開始した場合に，制約 (7) を満たせば真，そうでなければ偽を返す．

アクティビティ $j$ の処理開始時刻の上界値 $ub_j$ は，最適値の上界 $UB$ を基に取得する．本稿では，$UB$ の取得に文献 11) の初期解構築の手法を用いた．$UB$ が得られなかった場合は，$UB = t_{max} - 1$ とする．各アクティビティ $j$ に対して $ub_j$ を取得する手続きを**図 4** に示す．$S_i^*$ は，$S_i$ の要素のうち，処理開始時刻の上界値を与えられたアクティビティの集合である．

この手続きは，$UB$ を基にバックワードスケジューリングスケジューリングを行うものである．すなわち，$t = 0, \ldots, UB$，$r \in R$ において，資源供給量を $l_{rUB-t}$，アクティビティ $j$ の要求資源量を $d_{jrp_j-1-u}$ とし，制約 $(j, s)$ に代えて $(s, j), w_{sj} = w_{js}$ を課したインスタンスを R に緩和して解けばよい．

各アクティビティ $j$ 対する $lb_j$ および $ub_j$ の取得により，次の不等式を得る．

$$lb_j \leq st_j \leq ub_j, j \in J \tag{8}$$

式 (8) より，次のように変数を固定できる．

$$x_{jt} = 0, t < lb_j, t > ub_j, j \in J \tag{9}$$

$st_j$ はアクティビティ $j$ の処理実行開始時刻である．あるアクティビティ $j$ において $lb_j = ub_j$ であった場合，$st_j = lb_j$，$x_{jlb_j} = 1$ とすることができる．このようなアクティビティ $j$ が存在

```
procedure reduce_resource_limit()
begin
  for j := 0 to n − 1 do
   if st_j is fixed then
    for u = 0 to p_j − 1 do
     for r = 0 to m − 1 do
     begin
      l_{rst_j+u} −= d_jru;
      d_jru := 0
     end
end.
```

**図 5**　利用可能資源の削減
Fig. 5　Reducing resource limits.

する場合，$j$ の要求資源量および利用可能資源量に変更を加える．$r \in R$，$u = 0, \ldots, p_j - 1$ において，$d_{jru} = 0$ とする．また，$r \in R$，$t = lb_j + u$（$u = 0, \ldots, p_j - 1$）において，利用可能資源量から $d_{jru}$ を引く．このようにすれば，資源の制約について考える際に，$x_{jlb_j} = 1$ と固定されたアクティビティ $j$ を無視することができる．この手続きを **procedure reduce_resource_limit()** とし，**図 5** に示す．

### 4.2　ラグランジュ緩和による下界値計算

本節では，ラグランジュ緩和を用いた下界値の計算法を示す．制約 (2)，(3) および (4) をそれぞれ，乗数 $\lambda_{rt}$，$\omega_{js}$ および $\pi_{js}$ を用いてラグランジュ緩和する．これに制約 (7) を課すと，次のような最小化問題 LR を得る．

minimize

$$\begin{aligned} L = & \sum_{t=0}^{t_{max}-1} t x_{n-1t} + \sum_{j=0}^{n-1} \sum_{s \in S_j} \lambda_{js} \left( \sum_{t=0}^{t_{max}-1} t x_{st} - \sum_{t=0}^{t_{max}-1} t x_{jt} - p_j - w_{js} \right) \\ & + \sum_{j=0}^{n-1} \sum_{s \in S_j} \omega_{js} \left( \sum_{t=0}^{t_{max}-1} t x_{jt} + p_j + a_{js} - \sum_{t=0}^{t_{max}-1} t x_{st} \right) \\ & + \sum_{t=0}^{t_{max}-1} \sum_{r=0}^{m-1} \pi_{rt} \left( \sum_{j=0}^{n-1} \sum_{u=0}^{\min(t, p_j-1)} d_{jru} x_{j(t-u)} - l_{rt} \right) \end{aligned} \tag{10}$$

subject to

(7), (5), (6)

$x_{jt}$ を変数とする関数 (10) ではラグランジュ乗数は定数と見なすことができる．式 (10) を変形し，定数項を取り除くと式 (11) を得る．

$$\begin{aligned} L = & \sum_{t=0}^{t_{max}-1} t x_{(n-1)t} + \sum_{j=0}^{n-1} \sum_{t=0}^{t_{max}-1} \left[ \sum_{r=0}^{m-1} \pi_{rt} \sum_{u=0}^{\min(t, p_j-1)} d_{jru} x_{j(t-u)} \right. \\ & \left. + t x_{jt} \left\{ \sum_{i \in I_j} (\lambda_{ij} - \omega_{ij}) - \sum_{s \in S_j} (\lambda_{js} - \omega_{js}) \right\} \right] \end{aligned} \tag{11}$$

目的関数式 (11) は，アクティビティごとに独立である．よって緩和問題 LR は，アクティビティレベルの子問題に分解可能である．そこで，各 $j$ に対応する子問題を $\mathrm{LR}_j$ とし，その目的関数を $L_j(t)$ とすると，$L$ を最小化するためには，式 (12) のように各 $L_j(t)$ の最小値の和をとればよい．

$$\min L = \sum_{j=0}^{n-1} \min L_j(t) \tag{12}$$

また，LR は次のような最小化問題となる．

minimize

$$\begin{aligned} L_j(t) = y_j & \sum_{t=0}^{t_{max}-1} t x_{(n-1)t} + \sum_{t=0}^{t_{max}-1} \left[ \sum_{r=0}^{m-1} \pi_{rt} \sum_{u=0}^{\min(t, p_j-1)} d_{jru} x_{j(t-u)} \right. \\ & \left. + t x_{jt} \left\{ \sum_{i \in I_j} (\lambda_{ij} - \omega_{ij}) - \sum_{s \in S_j} (\lambda_{js} - \omega_{js}) \right\} \right] \end{aligned} \tag{13}$$

subject to

(5), (6), (7)

LR を解いた後，ラグランジュ双対問題を解き，ラグランジュ乗数を更新し，再度 LR を解くことを繰り返す．

ここで，$L_j(t)$ を式 (14) のように変形する．

$$L_j(t) = y_j \sum_{t=0}^{t_{max}-1} t x_{(n-1)t} + \sum_{t=0}^{t_{max}-1} \left\{ \sum_{r=0}^{m-1} \pi_{rt} \sum_{u=0}^{\min(t, p_j-1)} d_{jru} x_{j(t-u)} \right\}$$

$$+\sum_{t=0}^{t_{max}-1} tx_{jt}\left\{\sum_{i\in I_j}(\lambda_{ij}-\omega_{ij})-\sum_{s\in S_j}(\lambda_{js}-\omega_{js})\right\} \tag{14}$$

$\mathrm{LR}_j$ の解はアクティビティの処理開始時刻の係数，すなわち式 (15) の値の符号に大きく依存する．

$$\sum_{i\in I_j}(\lambda_{ij}-\omega_{ij})-\sum_{s\in S_j}(\lambda_{js}-\omega_{js}) \tag{15}$$

符号が正であれば，処理開始時刻は極端に早くなり，負であれば極端に遅くなることがありうる．なぜなら，式 (14) を最小化するには，第 3 項目の値が第 2 項目の値より目的関数に与える影響が大きければ，第 3 項目の値を小さくすればよい．そのためには，式 (15) の値が正であればアクティビティ $j$ の処理開始時刻を下界 $lb_j$ にすればよく，逆に負であれば $ub_j$ にすればよい．このことにより，式 (15) の値によって，$x_{jlb_j}=1$ と $x_{jub_j}=1$ という解を交互に繰り返す可能性がある．このような解の振動は，ラグランジュ乗数の収束を妨げ，良い下界値が得られない原因となりやすい[4]．よって，ここで猶予制約および待機制約を，次のような不等式で置き換える．

$$\rho_{ijt}+\sigma_{jt}\le 1, j\in J, i\in I_j, t\in T \tag{16}$$

$$\rho\in\{0,1\} \tag{17}$$

$$\sigma\in\{0,1\} \tag{18}$$

ここで，各記号は次のとおりである．

- $\rho_{ijt}$：$0-1$ 変数．$(i,j)$ に関して時刻 $t$ が，$0\le t\le c_i+a_{ij}$ または $c_i+w_{ij}+p_j<t$ を満たせば 1，それ以外は 0.
- $\sigma_{jt}$：$0-1$ 変数．時刻 $t$ が $st_j\le t\le c_j$ を満たせば 1，それ以外は 0.

ここで $c_j$ は，アクティビティ $j$ の処理完了時刻である．変数 $\rho_{ijt}$ および $\sigma_{jt}$ は，$x_{jt}$ に依存する．アクティビティ $i$ に対して処理開始時刻が与えられると，$\rho_{ijt}$ は，アクティビティ $j\in S_i$ が制約 (2) および (3) を満たして実行可能な時刻 $t$ に対して 0 をとる．またアクティビティ $j$ に対し，$\sigma_{jt}$ は処理実行中である時刻 $t$ に対して 1 をとる．$\sigma_{jt}=1$ である $j$ および $t$ に対して $\rho_{ijt}=0$ であれば，制約 (2)，(3) を満たすことになるので，式 (16) は，これらを代替することができる．$\rho_{ijt}$ と $\sigma_{jt}$ の例を**図 6** に示す．

これらをふまえ，乗数 $\pi_{rt}$，$\mu_{ijt}$ を用いてラグランジュ緩和すると，次の最小化問題を得る．

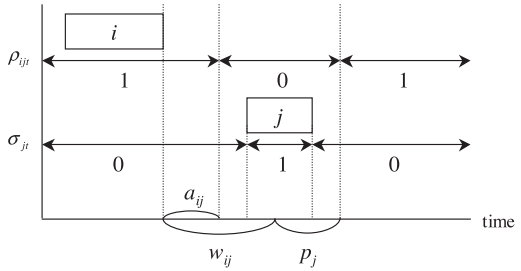


**図 6**　$\rho_{ijt}$ と $\sigma_{jt}$
Fig. 6　$\rho_{ijt}$ and $\sigma_{jt}$.

minimize

$$L^*=\sum_{t=0}^{t_{max}-1} tx_{n-1t}+\sum_{t=0}^{t_{max}-1}\sum_{r=0}^{m-1}\pi_{rt}\left(\sum_{j=0}^{n-1}\sum_{u=0}^{min(t,p_j-1)} d_{jru}x_{j(t-u)}-l_{rt}\right)$$
$$+\sum_{j=0}^{n-1}\sum_{t=0}^{t_{max}-1}\sum_{i\in I_j}\mu_{ijt}(\rho_{ijt}+\sigma_{jt}-1) \tag{19}$$

subject to

(5), (6), (7), (17), (18)

この緩和問題を $\mathrm{LR}^*$ と呼ぶことにする．$x_{jt}$ を変数とする関数 (19) ではラグランジュ乗数は定数と見なすことができる．式 (19) を変形し，定数項を無視すると，式 (20) を得る．

$$L^*=\sum_{t=0}^{t_{max}-1} tx_{(n-1)t}+\sum_{j=0}^{n-1}\sum_{t=0}^{t_{max}-1}\left(\sum_{r=0}^{m-1}\pi_{rt}\sum_{u=0}^{min(t,p_j-1)} d_{jru}x_{j(t-u)}\right.$$
$$\left.+\sum_{s\in S_j}\mu_{jst}\rho_{jst}+\sum_{i\in I_j}\mu_{ijt}\sigma_{jt}\right) \tag{20}$$

目的関数式 (20) は，アクティビティごとに独立である．よって $\mathrm{LR}^*$ はアクティビティレベルの子問題に分解可能である．そこで各 $j$ に対応する子問題を $\mathrm{LR}_j^*$，その目的関数を $L_j^*(t)$ とすると，$L^*$ を最小化するためには，式 (21) のように各 $L_j^*(t)$ の最小値の和をとればよい．

$$\min L^* = \sum_{j=0}^{n-1} \min L_j^*(t) \tag{21}$$

また，$\mathrm{LR}_j^*$ は次のような最小化問題になる．ただし，変数 $y_j$ は，$j = n-1$ であれば 1，それ以外は 0 である．

minimize

$$L_j^*(t) = y_j \sum_{t=0}^{t_{max}-1} t x_{n-1t} + \sum_{t=0}^{t_{max}-1} \left( \sum_{r=0}^{m-1} \pi_{rt} \sum_{u=0}^{min(t,p_j-1)} d_{jru} x_{j(t-u)} + \sum_{s \in S_j} \mu_{jst} \rho_{jst} + \sum_{i \in I_j} \mu_{ijt} \sigma_{jt} \right), j \in J \tag{22}$$

subject to

(5), (6), (7), (17), (18)

$\mathrm{LR}_j^*$ は，次のようにとらえられる．

- ダミーを除くアクティビティ数は 1.
- $L_j(t)$ は，アクティビティ $j$ が時刻 $t$ に処理を開始するためのコスト.
- 制約 (7) を満たせば，アクティビティは処理を行える.
- 処理は 1 回のみ開始できる．中断はしない.
- 以上の条件下で，コストが最小になる処理開始時刻を求める.

$\mathrm{LR}_j^*$ の目的関数 $L_j^*(t)$ の最小値を得るには，制約 (7) を満たす処理開始時刻 $t$ に対して，$x_{jt} = 1$ としたときの $L_j^*(t)$ の値を事前に求め，これが最小となるような $t$ を探すだけでよい.

$\mathrm{LR}^*$ を解いた後，ラグランジュ乗数を更新し，再度 $\mathrm{LR}^*$ を解く．この処理を繰り返し，得られた目的関数値のうち，最大のものを下界値として出力する.

### 4.3　ラグランジュ双対問題

精度の良い下界値を得るためには，$\mathrm{LR}^*$ のラグランジュ乗数 $\pi_{rt}$，$\mu_{ijt}$ をより良い値に更新する必要がある．本節では，$\mathrm{LR}^*$ のラグランジュ双対問題 LD を考え，劣勾配法を用いて解くことにより，ラグランジュ乗数を更新する．LD は次のような最大化問題である.

maximize

$L$

subject to

$$\pi_{rt} \geq 0, r \in R, t \in T \tag{23}$$

$$\mu_{jst} \geq 0, j \in J, s \in S_j, t \in T \tag{24}$$

$\mathrm{LR}^*$ のある実行可能解 $x$ における，ラグランジュ乗数 $\pi_{rt}$，$\mu_{ijt}$ に対する劣勾配をそれぞれ $\partial_\pi(x)$，$\partial_\mu(x)$ とする．$\partial_\pi(x)$ および $\partial_\mu(x)$ はそれぞれ，式 (25) および式 (26) で与えられる.

$$\partial_\pi(x) = \sum_{j=0}^{n-1} \sum_{u=0}^{\min(t,p_j-1)} d_{jru} x_{j(t-u)} - l_{rt}, r \in R, t \in T \tag{25}$$

$$\partial_\mu(x) = \rho_{ijt} + \sigma_{jt} - 1, j \in J, i \in I_j, t \in T \tag{26}$$

乗数 $\pi_{rt}$，$\mu_{ijt}$ に対する，$h$ 回目の更新手続きにおけるステップ長をそれぞれ，$\alpha^h$，$\beta^h$ とする．また，そのときの $\mathrm{LR}^*$ の実行可能解を $x^h$ とする．$LB$ は $h$ 回目に得られた下界値である．$\alpha^h$ および $\beta^h$ をそれぞれ，式 (27) および式 (28) に従って更新する.

$$\alpha^{h+1} = \frac{UB - LB}{||\partial_\pi(x^h)||^2} \tag{27}$$

$$\beta^{h+1} = \frac{UB - LB}{||\partial_\mu(x^h)||^2} \tag{28}$$

また，$h$ 回目の繰返しにおけるラグランジュ乗数 $\mu_{ijt}$ の値を $\mu_{ijt}^h$ とし，次の式 (30) に従って更新する.

$$\pi_{rt}^{h+1} = \max\{\pi_{rt}^h + \alpha^h \partial_\pi(x^h), 0\}, r \in R, t = 0, \ldots, UB \tag{29}$$

$$\mu_{ijt}^{h+1} = \max\{\mu_{ijt}^h + \beta^h \partial_\mu(x^h), 0\}, j \in J, i \in I_j, t = 0, \ldots, UB \tag{30}$$

### 4.4　LP 緩和によるアプローチ

ラグランジュ緩和によるアプローチでは，制約 (3) ないしは (16) が課されていないため，緩和問題の解が作業開始の順序関係を満たさない場合がある．しかし $\mathrm{LR}^*$ に待機制約を課すと，各アクティビティが独立ではなくなり，アクティビティレベルの子問題への分割が不可能になる．ここで，project scheduling with irregular starting time costs（PSISTC）[9] というスケジューリングモデルを考える．PSISTC は次のような点で RCPSP/$\tau$+ と異な

るモデルである．

- アクティビティ $j$ を時刻 $t$ に処理開始させるには，コスト $w_j(t)$ が必要．
- アクティビティの処理には資源を必要としない．
- 猶予制約は課されない．
- 目的関数はアクティビティの処理開始コストの総和の最小化．

PSISTC は，次のように定式化される．

minimize

$$z = \sum_{j=0}^{n-1} \sum_{t=0}^{t_{max}-1} w_j(t) x_{jt} \tag{31}$$

subject to

(5), (6)

$$\sum_{s=t}^{t_{max}-1} x_{is} + \sum_{s=0}^{t+\delta_{ij}-1} x_{js} \leq 1, \quad j \in J, i \in I_j, t \in T \tag{32}$$

Chaudhuri らは文献 10) において，式 (32) が待機制約となりうること，整数制約 (6) の緩和を行っても，解の整数性が保たれることを示した．$\delta_{ij}$ は，アクティビティ $i$，$j$ の処理実行開始のタイムラグである．RCPSP/$\tau$+ では，$\delta_{ij} = p_i + a_{ij}$ とすればよい．これらのことから，制約 (6) を式 (33) に緩和し，(7) を取り除いた線形計画問題を解くことにより，RCPSP/$\tau$+ の下界値を取得できる．

$$0 \leq x_{jt} \leq 1 \tag{33}$$

コスト $w_j(t)$ の値は，$L_j(t)$ において $x_{jt} = 1$ としたときの値を用いればよい．また，$x_{jt} = 1$ とすると制約 (7) に違反する $j$，$t$ に対して，事前に $L_j(t)$ に十分大きな値を与えておけば，間接的に (7) を課することができる．RCPSP/$\tau$+ を PSISTC に緩和し，これを LPR と呼ぶことにする．LPR を解き，4.3 節のようにラグランジュ乗数を更新し，再度 LPR を解くことによって下界値を取得する．

### 4.5　待機制約の追加による下界値計算

本節では，R の解における制約 (4) の違反に対し，待機制約を一時的に追加して再度 R を解くことにより，下界値を上昇させる．例として，次のインスタンスを考える．

- $n = 4$
- $m = 1$

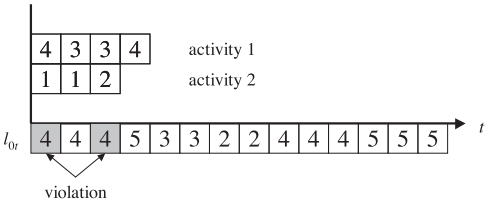


**図 7**　資源制約違反
Fig. 7　Violation of resource constraint.

- $t_{max} = 14$
- $p_0 = 0, p_1 = 4, p_2 = 3, p_3 = 0$
- $d_{10} = [4, 3, 3, 4], d_{20} = [1, 1, 2]$
- $l_0 = [4, 4, 4, 5, 3, 3, 2, 2, 4, 4, 4, 5, 5, 5]$
- $(0, 1)$, $(0, 2)$, $(1, 3)$, $(2, 3)$
- $a_{01} = 0$, $a_{02} = 0$, $a_{13} = 0$, $a_{23} = 0$

このインスタンスの緩和問題 R を解くと，$x_{00} = 1$，$x_{10} = 1$，$x_{20} = 1$，$x_{34} = 1$ という解が得られる．しかし，この解は，アクティビティ 1 および 2 が，時刻 $t = 0$ および $t = 2$ において，制約式 (4) に違反する．このときの様子を，**図 7** に示す．この制約違反を解消するために，問題を $R_{12}$ と $R_{21}$ に分割する．$R_{12}$ に待機制約 $(1, 2)$ を，$R_{21}$ に待機制約 $(2, 1)$ を追加する．待機時間 $a_{12}$ は，制約 (4) を満たすことができる処理開始時刻が存在するような，最小の値にする．ただし R の解での，アクティビティ 1 と 2 のオーバラップにおけるタイムラグより大きな値を用いる．例のインスタンスにおいて，制約 $(1, 2)$ を追加する場合，2 つのアクティビティは，処理開始時刻が同じなので，待機時間は $-p_1$，すなわち $-4$ より大きな値を考える．$w_{12} = -2$ とすれば，$x_{10} = 1$，$x_{22} = 1$ として，アクティビティ 1 と 2 の間における，制約 (4) の違反を回避できる．よってここでは，$(1, 2), a_{12} = -2$ という待機制約を加え，$R_{12}$ を解く．オーバラップして制約 (4) に違反した 2 つのアクティビティ $j_1$，$j_2$ に対して一時的に加える待機制約の待機時間取得の手続きを，**図 8** に示す．これは，引数で与えられたアクティビティ $j_1$ の処理完了からの待機時間を出力する．手続き satisfy は，アクティビティ $j_1$，$j_2$ が，それぞれ時刻 $\tau_{j_1}$，$\tau_{j_2}$ に処理を開始して，$j_1$，$j_2$ が要求する資源量の和が供給量を上回らなければ真を，そうでなければ偽を返す．

アクティビティ $j_1$，$j_2$ が，$st_{j_2} - st_{j_1} - p_{j_1} + 1$ から $-1$ のあらゆる待機時間によるオーバラップで制約 (4) を満たせなければ $j_1$ と $j_2$ はオーバラップできないことが分かる．この

ときは $a_{j_1j_2} = 0$ を待機時間として返してやればよく，$a_{j_1j_2} = 0$ の場合を検査する必要はない．

$R_{21}$ に対するタイムラグも同様に求め，制約 $(2,1)$ を追加し，$R_{21}$ を解く．$R_{12}$ と $R_{21}$ の目的関数値のうち，小さい方の値を下界値とする．処理の概要を，**図 9** に示す．手続き get_lower_bound() は，R を解いた後にコールされる．サブルーチン overlap$(j,k)$ は，与えられた解において，2 つのアクティビティ $j$，$k$ が同時に処理されている時間が存在すれば真，そうでなければ偽を返す．resource_violation$(j,k)$ は，与えられた解において，2 つのアクティビティ $j$，$k$ が要求する資源量の合計が，利用可能量を上回るような $r \in R$，$t \in T$ が存在する場合に真を，そうでなければ偽を返す．add_constraint$(j,k)$ は，例題を用いて述べたように制約 $(j,k)$ を追加する．delete_constraint$(j,k)$ は，制約 $(j,k)$ が存在する場合，これを取り除く．solve() は，緩和問題を解き，目的関数値を返す．ただし実行可能解が存在しない場合は，$t_{max}$ より大きな値を返す．

```
procedure set_time_lag(j1, j2)
begin
  for a_{j1j2} = st_{j2} − st_{j1} − p_{j1} + 1 to -1 do
   for τ_{j1} = lb_{j1} to ub_{j1} do
   begin
    τ_{j2} = τ_{j1} + p_{j1} + a_{j1j2};
    if lb_{j2} ≤ τ_{j2} ≤ ub_{j2} then
     if satisfy(τ_{j1}, τ_{j2}) then
      return a_{j1j2};
   end
   return 0;
end.
```

**図 8**　処理開始のタイムラグの設定
Fig. 8　Setting starting time lag.

```
procedure get_lower_bound()
begin
  for j := n − 2 down to 2 do
   for k := j − 1 down to 1 do
    if overlap(j, k) then
     if resource_violation(j, k) then
     begin
      a_{jk} = set_time_lag(j, k);
      add_constraint(j, k, a_{jk});
      obj_{jk} = solve();
      delete_constraint(j, k);
      a_{k,j} = set_time_lag(k, j);
      add_constraint(k, j, a_{kj});
      obj_{kj} = solve();
      delete_constraint(k, j);
      if lb_{n−1} ≤ min(obj_{jk}, obj_{kj}) ≤ t_{max} − 1 then
       return min(obj_{jk}, obj_{kj})
     end
end.
```

**図 9**　待機制約の追加による下界値取得
Fig. 9　Getting lower bound adding after constraint.

この手法は，緩和問題 R だけでなく，LPR の分割にも応用することができる．この場合，1 度 LPR を解いてから待機制約の追加により分割された子問題を解き，ラグランジュ乗数を更新するという繰返しを行う．

## 5. 数 値 実 験

本章では，これまでに述べた手法の数値実験の結果を示す（**表 1**，**表 2**，**表 3**，**表 4**）．最適値との比較により，得られる下界値の精度を検証する．対象は，web ページ PSPLIB [*1] で公開されている，$n = 12$，22，32，42 の RCPSP インスタンスをそれぞれ 10 個ずつ，ランダムに拡張したインスタンスである．それぞれ $t_{max} = 200$，$m = 4$ である．実行環境は次のとおりである．

**表 1**　$n = 12$ での結果
Table 1　Results on $n = 12$.

| inst. | 最適値 | R | LR* | LPR | 分割 | 分割 LPR |
|---|---|---|---|---|---|---|
| 1 | 52 | 52 | 52 | 52 | 52 | 52 |
| 2 | 160 | 160 | 160 | 160 | 160 | 160 |
| 3 | 48 | 48 | 48 | 48 | 48 | 48 |
| 4 | 101 | 35 | 46.21 | 49.41 | 35 | 49.56 |
| 5 | 127 | 127 | 127 | 127 | 127 | 127 |
| 6 | 80 | 74 | 77.50 | 77.57 | 80 | 80 |
| 7 | 90 | 78 | 80.38 | 80.62 | 90 | 90 |
| 8 | 83 | 50 | 51.86 | 52.04 | 56 | 56 |
| 9 | 57 | 55 | 55 | 55 | 55 | 55 |
| 10 | 53 | 53 | 53 | 53 | 53 | 53 |

⋆1 http://129.187.106.231/psplib/

**表 2**　$n = 22$ での結果
Table 2　Results on $n = 22$.

| inst. | 最適値 | R | LR* | LPR | 分割 | 分割 LPR |
|---|---|---|---|---|---|---|
| 11 | 58 | 58 | 58 | 58 | 58 | 58 |
| 12 | 87 | 63 | 64.60 | 65.69 | 63 | 65.61 |
| 13 | 73 | 60 | 64.10 | 64.51 | 66 | 68.77 |
| 14 | 88 | 57 | 57.74 | 59.57 | 88 | 88 |
| 15 | 61 | 51 | 51.32 | 52.00 | 56 | 56 |
| 16 | 105 | 103 | 103 | 103 | 105 | 105 |
| 17 | 104 | 101 | 101 | 101 | 101 | 101 |
| 18 | 119 | 63 | 68.22 | 75.56 | 64 | 75.44 |
| 19 | 77 | 75 | 75 | 75.01 | 76 | 76 |
| 20 | 98 | 59 | 59.07 | 63.18 | 59 | 63.21 |

**表 3**　$n = 32$ での結果
Table 3　Results on $n = 32$.

| inst. | 最適値 | R | LR* | LPR | 分割 | 分割 LPR |
|---|---|---|---|---|---|---|
| 21 | 71 | 61 | 64.49 | 64.63 | 61 | 64.64 |
| 22 | 179 | 119 | 122.66 | 136.67 | 119 | 136.67 |
| 23 | 139 | 113 | 115.89 | 117.03 | 139 | 139 |
| 24 | 124 | 97 | 102.22 | 102.59 | 98 | 103.31 |
| 25 | 83 | 83 | 83 | 83 | 83 | 83 |
| 26 | 84 | 84 | 84 | 84 | 84 | 84 |
| 27 | 184 | 148 | 148 | 166.15 | 148 | 166.15 |
| 28 | 92 | 78 | 78.06 | 79.01 | 83 | 83 |
| 29 | 162 | 154 | 156.87 | 156.92 | 154 | 156.92 |
| 30 | 182 | 136 | 136 | 146.86 | 136 | 146.86 |

- OS：Windows XP SP2
- 言語：C++
- CPU：Pentium4 2.4 GHz
- メモリ：768 MB

“inst.” 欄はインスタンスの番号を，“最適値” 欄は各インスタンスの最適値を示す．“R”，“LR*”，“LPR” および “分割” はそれぞれ，R，LR*，LPR，待機制約の追加による手法から得た下界値を示す．また “分割 LPR” は，“分割” 手法から下界値を取得してラグランジュ乗数を更新した後，再度 “分割” 手法で下界値を得るという繰返しから得た下界値を示す．ラグランジュ乗数とステップ長の更新は，4.3 節と同様に行った．“LR*” および “分割 LPR” での繰返し回数の上限は 100 回である．“LR*”，“分割”，“分割 LPR” それぞれにおいて，下界値の暫定値が最適値と一致した時点で，アルゴリズムは終了する．また，線形計画問題を解く場合に，GNU Linear Programming Kit（GLPK）4.9 を用いた．

**表 4**　$n = 42$ での結果
Table 4　Results on $n = 42$.

| inst. | 最適値 | R | LR* | LPR | 分割 | 分割 LPR |
|---|---|---|---|---|---|---|
| 31 | 181 | 181 | 181 | 181 | 181 | 181 |
| 32 | 130 | 114 | 117.83 | 118.18 | 130 | 130 |
| 33 | 132 | 84 | 90.88 | 91.50 | 110 | 110 |
| 34 | 150 | 150 | 150 | 150 | 150 | 150 |
| 35 | 135 | 89 | 93.43 | 94.30 | 92 | 94.39 |
| 36 | 103 | 103 | 103 | 103 | 103 | 103 |
| 37 | 183 | 179 | 179 | 179 | 179 | 179 |
| 38 | 173 | 110 | 110 | 118.25 | 110 | 118.25 |
| 39 | 147 | 132 | 132 | 132 | 132 | 132 |
| 40 | 149 | 117 | 117 | 117 | 117 | 117 |

**表 5**　誤差の平均 [%]
Table 5　Gap average [%].

| $n$ | R | LR* | LPR | 分割 | 分割 LPR |
|---|---|---|---|---|---|
| 12 | 12.94 | 10.91 | 10.54 | 10.14 | 8.70 |
| 22 | 19.13 | 17.80 | 16.26 | 13.56 | 11.49 |
| 32 | 15.31 | 13.80 | 11.20 | 12.81 | 9.12 |
| 42 | 15.30 | 14.16 | 13.54 | 11.88 | 11.23 |

**表 6**　平均実行時間 [sec]
Table 6　Computation time average [sec].

| $n$ | R | LR* | LPR | 分割 | 分割 LPR |
|---|---|---|---|---|---|
| 12 | 0.70 | 1.20 | 5.26 | 0.75 | 9.46 |
| 22 | 1.60 | 3.42 | 42.10 | 1.71 | 145.61 |
| 32 | 5.54 | 11.35 | 201.03 | 5.59 | 862.81 |
| 42 | 10.54 | 18.50 | 441.33 | 10.97 | 1294.86 |

**表 5** に，$n = 12$，22，32，42 それぞれのインスタンスにおける誤差の平均を示す．

**表 6** に，$n = 12$，22，32，42 それぞれのインスタンスでの平均実行時間を示す．“分割 LPR” 手法の平均実行時間が非常に長くなっている．これは，1 インスタンスにおいて多く

の線形計画問題を解かなければならないためである．このため，“分割 LPR” 手法の実行速度は，解く線形計画問題の数および 1 回解く速度に大きく依存する．

“LR*”，“LPR”，“分割” および “分割 LPR” はそれぞれ，緩和問題 R を解くことから始まる．それゆえ，R の結果と比較してどれだけ下界値を改善したかを見ることによって，各手法の性能を評価できる．これらの結果から，下界値の精度と計算時間の両面で良好な結果となったのは “分割手法” であることが分かった．しかし，アクティビティ数の増加にともない，R から下界値を改善したインスタンス数は少なくなっている．これは，一時的に待機制約を課されるアクティビティが 1 組のみであるため，$n$ が大きくなると追加された制約が及ぼす効果が小さくなることが原因だと思われる．これを改善するためには，たとえば R の解において制約 (4) を違反するアクティビティのペアを複数選択し，待機制約も複数加えるといった方法が考えられる．どのような待機制約を追加すれば下界値が上昇しやすいかという点についても今後調査が必要である．たとえば，$lb_j$ が $lb_{n-1}$ に近いアクティビティのペアという基準が考えられる．これは，処理完了時刻が遅くなりそうなアクティビティほど，アクティビティ$n-1$ の完了時刻に強く作用しそうだという予想に基づく．分割手法によって下界値が改善するインスタンスには，何がしかの共通した特徴があるかもしれない．たとえば，2 つのアクティビティのオーバラップによって，資源制約 (4) の違反が起こりやすいといった点である．しかし，分割手法は R から取得した解に対して実行するので，R からどのような解が得られるかという点に大きく依存する．このため，分割によって下界値が改善するかという見積りに関しては，さらに調査が必要である．また表 5 から，緩和問題 R から得た下界値の改善という点では，平均的には LR* および LPR ともに成功したことが分かる．緩和問題 LR* と LPR を比較すると，多くのインスタンスで LPR の方が良い精度を出力することが分かる．これは，LPR には待機制約が課せられているが，LR* では緩和によって取り除かれているという違いに起因する．特に，表 5 における $n = 32$ での結果では，分割手法を上回る精度である．また表 3 においては，LPR が分割手法を上回るインスタンス数は，半数以上の 6 である．$n = 42$ においては，表 4 および表 5 ともに分割手法に劣る．しかし，先述のように $n$ が大きくなるにつれて分割の効果が小さくなっていることを考えると，$n$ がより大きいインスタンスにおける性能を評価，比較する必要がある．

LR*，LPR および分割 LPR に関しては，ラグランジュ乗数の更新に用いるステップ長の更新方法が妥当であるかの調査も必要である．本稿の RCPSP/$\tau$+ モデルのように，目的関数がメイクスパンの最小化である場合，アクティビティ$n-1$ 以外はどのような時刻に処理が完了しても，目的関数に直接は関連しない．しかし，本稿で提案した手法のうち，R 以外については，JIT モデルのようなメイクスパン最小化以外の目的関数に応用可能である．たとえば，作業順序が定められることがあり，各作業の納期ずれにともなうコストの和を最小化するようなモデルへの応用は容易である．これらを今後の課題とする．

## 6. おわりに

本稿において，我々は RCPSP/$\tau$+ モデルを定式化し，ラグランジュ緩和法，LP 緩和および子問題への分割に基づく下界値の計算方法を提案した．提案した手法は，ラグランジュ緩和法，PSISTC への緩和および待機制約の追加による緩和問題の分割であった．$n = 12$ から 42 のインスタンスを用いて，最適値との比較により，得られた下界値の精度を評価した．その結果，精度の平均と計算時間の両面で “分割” 法が優れた結果を残すことが分かった．最適値との差の平均は 10%前後という結果を得た．これは，離散最適化問題の解法としては，良好な結果と考えられる．ラグランジュ緩和と分割を併用した計算法である “分割 LPR” は，精度の平均では分割手法を上回るものの，計算時間の面で劣る結果となった．下界値の精度と計算速度は，分枝限定法に実装した場合，その性能に直結する．このため，精度と計算速度の両面から，“分割” 手法が優れているといえる．ラグランジュ緩和を用いた手法は，ラグランジュ分解調整法への応用が考えられる．ラグランジュ緩和を用いた手法から得た解を利用して実行可能解を構築できれば，精度の良い実行可能解を取得することが期待できる．厳密解の取得が目的ではなく，ある程度の計算時間を確保できるのであれば，ラグランジュ分解調整法の利用も考えられる．

## 参考文献


1) Blazewicz, J., Lenstra, J.K. and Kan, A.H.G.R.: Scheduling subject to resource constraints: classification and complexity, *Discrete Applied Mathematics*, Vol.5, Issue 1, pp.11–24 (1983).
2) Brucker, P., Drexl, A., Möhring, R., Neumann, K. and Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods, *European Journal of Operational Research*, Vol.112, Issue 1, pp.3–41 (1999).
3) Hartmann, S.: *Project Scheduling under Limited Resources: Models, Methods and Applications*, Springer-Verlag Berlin, Heidelberg (1999).
4) Hoitomt, D.J.: A Practical Approach to Job-Shop Scheduling Problems, *IEEE Trans. Robotics and Automation*, Vol.9, pp.1–13 (1993).
5) Stinson, J.P., E.W.D. and Khumawala, B.M.: Multiple Resource-Constrained Scheduling Using Branch and Bound, *AIIE Trans.*, Vol.10, No.8, pp.252–259 (1978).

6) Kolish, R.: Efficient priority rules for the resource-constrained project scheduling problem, *Journal of Operations Management*, Vol.14, pp.179–192 (1996).
7) Kolish, R. and Sprecher, A.: PSPLIB — A project scheduling problem library, *European Journal of Operational Research*, Vol.96, pp.205–216 (1996).
8) Christofides, N., R.A.-V. and Tamarit, J.M.: Project shceduling with resource constraints : A branch and bound approach, *European Journal of Operational Research*, Vol.29, pp.262–273 (1987).
9) Möhring, R.H., Schulz, A.S., F.S. and Uetz, M.: On project scheduling with irregular starting time costs, *Operations Research Letters*, Vol.28, pp.149–154 (2001).
10) Chaudhuri, S., R.A.W. and Mitchell, J.E.: Analyzing and Exploiting the Structure of the Constraints in the ILP Approach to the Scheduling Problem, *IEEE Trans. Very Large Scale Integration* (*VLSI*) *Systems*, Vol.2, pp.456–471 (1994).
11) 草部博輝，中森眞理雄：タイムラグ付き RCPSP/$\tau$ に対するヒューリスティックな解法，情報処理学会論文誌：数理モデル化と応用，Vol.48, No.SIG 15 (TOM 18), pp.34–46 (2007).
12) 草部博輝，中森眞理雄：資源制約と先行制約を拡張したプロジェクトスケジューリング問題とその下界値の計算方法，電子情報通信学会論文誌 A，Vol.J91-A, No.4 (2008). (印刷中)

## 付　　録

### A.1　インスタンスの生成

本稿の数値実験で用いたインスタンスは，PSPLIB で公開されている RCPSP のインスタンスに対して，資源制約，待機制約および猶予制約をランダムに拡張して追加した．ここでは各制約の生成法を示す．

#### A.1.1　猶予制約と待機制約の生成

猶予，待機制約の生成には，**図 10** に示された処理 extend_precedence を用いた．これは，2 つのアクティビティ $j$，$s \in S_j$ を引数とし，オリジナルの RCPSP インスタンスにおいて課せられていた先行制約を，あらかじめ定めた確率に従い，猶予，待機制約のいずれかに変更する．途中で用いられるサブルーチン set は，引数で与えられた猶予制約と猶予時間，もしくは待機制約と待機時間を，生成する RCPSP/$\tau$+ インスタンスに課する手続きである．また，サブルーチン random は，0 以上 1 未満の乱数を返す．

#### A.1.2　資源制約の生成

$d_{jru}$（$j \in J$，$r \in R$，$u = 0, \ldots, p_j - 1$）は，**図 11** に示された処理 extend_resource_demand を用いて決定した．途中で用いられる $d^*_{jr}$ は，オリジナルの RCPSP イ

```
procedure extend_precedence(j, s)
begin
 if 0 ≤ random(0,1) < 0.4 then
 begin
  rnd := random(0,1);
  if rnd < 0.5 then
  begin
   w_js := ⌈(p_j + s_j) * random(0,1) * 1.5⌉;
   set(⟨j, s⟩, w_js);
  end
  else
   a_js := ⌈(p_j + s_j) * random(0,1) * 1.5⌉ ;
   set([j, s], a_js);
  end
 end
 else
  set([j, s], 0);
end.
```

**図 10**　猶予制約と待機制約の生成
Fig. 10　Generation of within and after constraints.

```
procedure extend_resource_demand(j)
begin
 for r := 0 to m − 1 then
  for u := 0 to p_j − 1 then
  rnd := random();
  if 0 ≤ rnd < 0.4 then
   d_jru := d*_jr;
  else if 0.4 ≤ rnd < 0.7 then
  begin
   if u = 0 then
    d_jru := d*_jr;
   else
    d_jru := d_jru−1;
  end
  else
   d_jru := ⌈d*_jr * random()⌉;
end.
```

**図 11**　要求資源量の生成
Fig. 11　Generation of resource demand.

```
procedure extend_resource_limit()
begin
  for r := 0 to m − 1 then
   for t := 0 to t_max − 1 then
    rnd := random(0, 1)
    if 0 ≤ rnd < 0.35 then
     l_rt = l*_r;
    else if 0.35 ≤ rnd < 0.65 then
     if t = 0 then
      l_rt = l*_r;
     else
      l_rt = l_rt−1;
    else if 0.65 ≤ rnd < 0.95 then
     l_rt = l*_r ∗ random(0.7, 1.2);
    else if 0.95 ≤ rnd ≤ 1 then
     l_rt = 0;
    end
end.
```

**図 12**　利用可能資源量の生成
Fig. 12　Generation of resource limit.

ンスタンスにおいて，アクティビティ $j$ が要求する資源 $r$ の量である．この処理はアクティビティ $j$ を引数とし，あらかじめ定めた確率に従い，各 $j$，$r$，$u$ に対して $d^*_{jr}$ を基に $d_{jru}$ を生成する．

$l_{rt}$ ($r = 0, \ldots, m-1$, $t \in T$) は，**図 12** に示された処理 procedure extend_resource_limit によって決定した．途中で用いられる $l^*_r$ は，オリジナルの RCPSP インスタンスにおける資源 $r$ の使用可能量である．この処理は引数を持たず，あらかじめ定めた確率に従い，各 $r$，$t$ に対して $l^*_r$ を基に $l_{rt}$ を生成する．

(平成 19 年 11 月 21 日受付)
(平成 19 年 12 月 27 日再受付)
(平成 20 年 2 月 1 日再受付 (2))
(平成 20 年 2 月 27 日採録)

**草部　博輝**

昭和 51 年生．平成 15 年東京農工大学大学院工学府（後期課程）電子情報工学専攻学生．スケジューリング問題の研究に従事．

**中森眞理雄**（正会員）

昭和 52 年東京大学大学院工学研究科計数工学専攻博士課程修了．工学博士．同年東京農工大学工学部講師．現在，同大学教授．アルゴリズム，データ構造，数理計画法，情報処理教育カリキュラムの研究に従事．情報処理学会 MPS 研主査（平成 7〜10 年）・CE 研主査（平成 18 年〜）・情報処理教育検討委員会幹事（平成 3〜6 年）．日本オペレーションズ・リサーチ学会理事（平成 9〜10 年，17 年〜）・フェロー．