タイムラグ付き RCPSP/τ に対するヒューリスティックな解法

草部博輝[†] 中森眞理雄[†]

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/τ モデルに，タイムラグの概念を追加した拡張モデル，RCPSP/τ+ モデルを取り扱う．本稿において，我々は RCPSP/τ+ モデルの解法として，タブーサーチアルゴリズムを提案し，ILOG CPLEX から得られた最適解と比較することにより，解の精度を評価する．

A Heuristic Algorithm for the RCPSP/τ with Time Lags

Hiroaki Kusakabe[†] and Mario Nakamori[†]

Resource-constrained project-scheduling problem (RCPSP) is a general model of several classical scheduling models. In this paper, we suggest the scheduling model RCPSP/τ+, which is added the time windows to the model of RCPSP/τ 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 tabu search algorithm for the RCPSP/τ+ and evaluate the solution accuracy comparing the oplitmal solution of ILOG CPLEX.

1. はじめに

20 世紀初頭からの，生産活動の大規模化に合わせ，生産資源としてのハードウェアも大きな進歩をとげた．これらのハードウェアも，非常に複雑な仕様を持つようになってきており，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不可欠である．たとえば，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．半導体に回路パターンを焼き付ける露光装置のスケジュールを作成する問題は，そのような例の 1 つである．

集積回路は 20 世紀の中頃に考案され，その後半導体製造技術の進歩により，回路規模および性能が向上してきた．今日，集積回路はコンピュータの CPU やメモリといった電子部品の構成には必要不可欠である．これらは，半導体露光装置を用いてつくられる．半導体露光装置とは，レーザ光を照射することによりシリコンウエハ上へ回路を焼き付ける装置である．これは，回路パターンの原画にあたるレチクル，実際に回路を焼き付けられるウエハおよび回路パターンを光学的に収束させるためのレンズ群により構成される．シリコンウエハへの回路の焼付けは，次の a から h までのステップをたどることにより行われる．

a. ウエハの洗浄
b. 成膜
c. 感光剤の塗布
d. 露光，現像
e. エッチング
f. イオンの注入

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

g. 層間膜および配線の形成
h. 平坦化，洗浄

1 つのシリコンウエハには，複数の回路パターンが層をなして焼き付けられる．工程 h から再び工程 c に移行し，レチクルを変えて異なる回路パターンを焼き付ける．この際，感光剤の塗布から露光までに時間がかかりすぎてしまうと，感光剤としての働きが失われてしまう．また露光の前準備として，レンズ群を通して回路パターンの像がシリコンウエハ上に，光学的に収束するかをチェックする必要がある．このチェックにも有効期間があり，チェック終了後から時間が経過しすぎると，信頼性が失われる．さらに，工程 f および g の完了後，ウエハ全体としての定着を待つ必要がある．このように，順序を定められたある作業間において，タイムラグが必要になる．回路パターンの像の収束についても，確認が必要であり，ある道具を用いて行われる．確認の最中，必要になる道具数が変化する．今用いてるものが 4 つ，次の瞬間には 3 つ，その次には 5 つといった具合である．確認に必要な道具の最大数を事前に用意するという考え方もあるが，ある瞬間に用いられていない道具を使い回すことによりなるべく無駄を減らしたいという要望がある．また，露光に用いられるハードウェアは，起動後に定期的にメンテナンスを行う必要があり，つねに一定量が稼動可能であるとは限らない．このように，作業に必要な資源量が変化し，かつ利用可能な資源量も変化することを考慮に入れなければならない．

実際の生産現場では，解の精度と計算速度の両立が求められる．仮に非常に優良なスケジュールを作成できたとしても，装置の故障や予定外の緊急作業が入ってくると，早急な再スケジューリングが必要になる．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）の変形モデルおよびその解法を提案する．本稿で提案するアルゴリズムから得られる解の精度は，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 章では，資源制約付きプロジェクトスケジューリング問題，および本稿で扱う変形モデルの概要を述べる．3 章では，本稿で扱う変形モデルを 0-1 整数計画問題として定式化する．4 章では，タブーサーチを用いたヒューリスティックなアルゴリズムを示す．5 章では，数値実験の結果を示す．

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/τ モデルおよび本稿で扱う変形モデルである RCPSP/τ+ 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．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．アクティビティの実行のために，一般的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．各資源は一般的に，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．一般的に，アクティビティ処理のために，各資源は種類ごとに複数個占有される．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，各アクティビティごとに異なる．また，各アクティビティは，それぞれに処理時間が与えられており，処理の中断は許されない．ただし，ダミーであるアクティビティ 0，$n-1$ の処理時間は 0 である．ある 2 つのアクティビティ間には，作業順序が定められることがある．この作業順序に関する制約を先行制約と呼ぶ．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．

2.2 RCPSP/τ

変形モデルの 1 つに，RCPSP/τ がある．このモデルは，文献 1) で提案されており，RCPSP と次のような点が異なる．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．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．これらの変化の様子は既知である．

2.3 RCPSP/τ+

本稿では，RCPSP/τ モデルの先行制約にタイムラグの概念を導入した，RCPSP/τ+ モデルを提案する．各資源には，時刻ごとに利用可能量が与えられている．各アクティビティが要求する資源量は処理開始からの経過時間によって変化する．2 つのアクティビティ $j=0$，$n-1$ は処理時間 0 のダミーである．さらに，作業順序が与えられたアクティビティ間に，2 種類のタイムラグに関する制約を追加する．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 に先行するという作業順序が与えられている．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，半導体露光装置のスケジューリングだけではなく，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．
- 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．

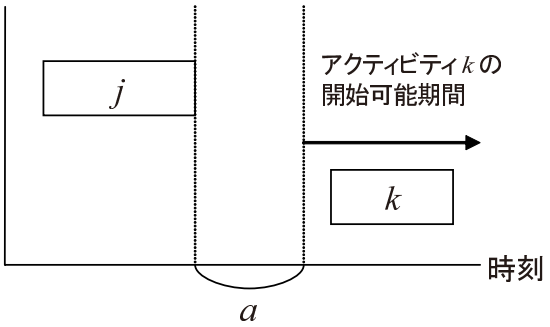

図 1 待機制約
Fig. 1 after constraint.

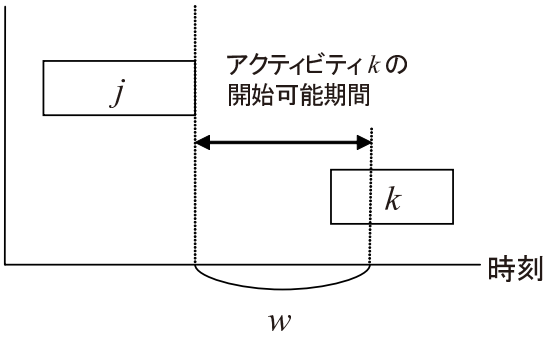

図 2 猶予制約
Fig. 2 within constraint.

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 間に作業順序が課せられていない場合，(j,k) と表記する．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) 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，作業順序関係，猶予時間および待機時間はすべて既知である．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/τ+ は，半導体露光装置のスケジューリングだけではなく，先述の例にある鉄鋼業の延鉄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できる．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．

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 2) および 5) がある．文献 2) 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 5) 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 3) があげられる．さらに，ヒューリスティックな手法に関する研究として，文献 6) があげられる．この文献は，マルチモードモデルに RCPSP/τ の特徴を取り入れ，さらに特殊な先行制約を用いることにより，非常に多様な表現力を持つ応用性の高いモデルを扱っている．解法として提案されているのは，タブーサーチアルゴリズムである．文献 1) 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 4) 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．

RCPSP/τ+ は一見，文献 6) のモデルに包含されるようであるが，制約条件を課するほどにモデルの特殊性が高まっている．モデルの特徴を最大限に活用して現実のスケジューリング問題に応用する，という面では我々のモデルの方が，その対象が広い．半導体露光装置のスケジューリングは，文献 6) のような拡張を必要とせず，利用可能資源量の変化，アクティビティが要求する資源量の変化，およびタイムラグの制約を用いてモデル化が可能である．本稿では，これらの制約にのみ着目し，モデル化および解法の提案を行う．

3. 定 式 化

RCPSP/τ+ は，次のような 0-1 整数計画問題に定式化される．

minimize

$$z = \max_j \left\{ \sum_{t=0}^{t_{max}-1} t x_{jt} + p_j \right\} \tag{1}$$

subject to

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

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

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

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

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

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.
- t_{max}：プロジェクト期間.
- T：離散で定義される時刻の集合.
  $T = \{0, ..., t_{max} - 1\}$
- p_j：アクティビティ j の処理時間.
- 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.

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)，(6) は，ノンプリエンプティブを示す制約である．アクティビティ j，k に対し $\langle j,k\rangle$ であり，その他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．

4. アルゴリズム

本章では，RCPSP/τ+ に対するアルゴリズムを示す．本アルゴリズムは，初期解構築フェイズと改善フェイズから構成される．事前の準備として，2 つのアクティビティ j，k に対し $\langle j,k\rangle$ かつ $w_{jk}=0$ である場合，j，k を合わせて 1 つのアクティビティとして扱うよう，結合処理を行う．たとえば，4 つのアクティビティ i，j_1，j_2，s を考える．これらは $[i,j_1]$，$\langle j_1,j_2\rangle$ かつ $w_{j_1j_2}=0$，$[j_2,s]$ であるとする．また，$p_{j_1}=3$，$p_{j_2}=2$，$d_{j_1r}=[1,2,3]$，$d_{j_2r}=[4,5]$ であるとする．アクティビティ j_1，j_2 を結合することによって，新たなアクティビティ k を生成する．アクティビティ k は，$p_k=p_{j_1}+p_{j_2}=5$，$d_{kr}=[d_{j_1r0},d_{j_1r1},d_{j_1r2},d_{j_2r0},d_{j_2r1}]=[1,2,3,4,5]$ となる．また，結合処理以降は j_1，j_2 の代わりに k を用い，$[i,k]$，$w_{ik}=w_{ij_1}$ および $[k,s]$，$w_{ks}=w_{j_2s}$ を新たに課す．アクティビティ k は，$\langle j_1,j_2\rangle$ かつ $w_{j_1j_2}=0$ を満たした j_1，j_2 を代替するので，すべての実行可能解において，j_1，j_2 の代わりに k を用いることができる．この結合処理を行うことにより，アクティビティ数を減らすことができる．

4.1 初期解の構築

初期解の構築には，タブーリスト付き priority rule（priority rule with tabu list: PRTL）を用いた．この手法は，RCPSP における，priority rule を用いた実行可能解構築法に，(j,st_j) を属性とするタブーリストを併用したものである．ただし，$j\in J$ であり，st_j はアクティビティ j の処理開始時刻である．J^+ を，すでに処理開始時刻を与えられたアクティビティの集合とする．$|J^+|<n$ の場合，J^+ によって構築されている不完全なスケジュールを，部分スケジュールと呼ぶことにする．アクティビティに処理開始時刻を与えることを，ディスパッチするということにする．まず，各アクティビティに何がしかの優先度を与える．優先度の高い順にディスパッチする権利を与える．今，アクティビティ j が権利を持っているとすると，j に対して，$J^+\cap\{j\}$ が制約条件を満たすような，最小の処理開始時刻の付与を試みる．j がディスパッチされたされた場合は，集合 J^+ に要素 j を加える．ディスパッチの成功，失敗にかかわらず，優先度が 1 つ低いアクティビティに権利を譲渡する．これを，最も優先度が低いアクティビティまで行う．ここまでの手続きを，n 回繰り返す．ただし，実行可能スケジュールが生成されたら，その時点で終了する．この手続きを dispatching_in_PRTL として，**図 3** に示す．c_j はアクティビティ j の処理完了時刻を表し，$c_j=st_j+p_j$ である．P_j は，j の先行アクティビティの集合である．また，$P_j^w\subset P_j$ および $P_j^a\subset P_j$ をそれぞれ，アクティビティ j を先行アクティビティとして持ち，j と猶予制約，待機制約を課されたアクティビティの集合とする．途中でコールされるサブルーチン in_TL は，アクティビティ j と時刻 t を引数として与え，(j,t)

```
procedure initialize()
begin
  st_0 := 0;
  dsp_0 := 0;
  h := 1;
  for j := 1 to n − 1 do
   st_j := NOT_START;
end.
procedure dispatching_in_PRTL(J)
begin
  initialize();
  for itr := 1 to n − 1 do
   for i := 0 to n − 2 do
   begin
    j := priority(J \ {0}, i);
    if (P_j ⊂ J^+) and not(j ∈ J^+) then
    begin
     t_from := max_k{c_k + a_jk | k ∈ P_j^a};
     if |P_j^w| ≥ 1 then
      t_to := min_k{c_k + w_jk | k ∈ P_j^w};
     else
      t_to := t_max − 1;
     for t := t_from to t_to do
      if not(in_TL(j,t)) then
       if satisfy(j,t) then
       begin
        st_j := t;
        dsp_h := j;
        h++;
        J^+ := J^+ ∪ {j};
        break;
       end
    end
   end
   if feasible(J^+) then
    break;
end.
```

図 3 PRTL で用いるディスパッチの手続き
Fig. 3 Dispatching procedure used in PRTL.

がタブーリストに記録されていれば真を，そうでなければ偽を返す．satisfy は，$st_j = t$ とすることで，J^+ の全要素間で制約条件を満たすなら真を，そうでなければ偽を返す．priority は，引数としてアクティビティの集合 J と整数 i を与え，集合 J の要素から，i 番目の優先度を持つアクティビティを返す．また，feasible は，アクティビティの集合 J を引数として与え，J の全要素が実行可能なスケジュールを構築していれば真を，そうでなければ偽を返す．

この手続きを実行することにより，(部分) スケジュール，およびディスパッチされたアクティビティの順序を表す配列 $[dsp]$ が得られる．dsp_h は，h 番目にディスパッチされたアクティビティの番号が格納されている．

n 回の繰返し後，あるアクティビティ $j \in J$ がディスパッチできなかった場合，その原因が P_j の要素のいずれかにあると判断する．このとき，決められた手続きにより P_j からアクティビティを 1 つ選択し，その番号と処理開始時刻をタブーリストに記録する．ただし，$P_j = 0$ である場合は，例外的に S_0 の中からディスパッチされているアクティビティを 1 つ選択し，その番号と処理開始時刻をタブーリストに記録する．

J^- を，先行アクティビティがすべてディスパッチされており，かつ自分自身がディスパッチされなかったアクティビティの集合とする．$|J^-| \geq 2$ である場合，J^- の要素から最も高い優先度を持ったアクティビティを選択する．今，アクティビティ $j^- \in J^-$ を選択したと仮定し，タブーリストに記録するアクティビティの決定手続きを get_tabu_activity として，**図 4** に示す．サブルーチン create_set_K は，引数としてディスパッチされなかったアクティビティ j^- を与え，$P_{j^-} \neq \{0\}$ の場合にコールされる．これは，$|P^w_{j^-}| \geq 1$ であれば，集合 P_{j^-} の要素のうち，式 (7) の値を持つアクティビティの集合を，それ以外の場合では，$P^a_{j^-}$ の要素のうち，式 (8) の値を持つアクティビティの集合を K とする手続きである．サブルーチン exists は，引数で与えた制約が存在するならば真を，そうでなければ偽を返す．

$$\min_{k \in P_{j^-}} \{c_k + w_{kj^-}\} \tag{7}$$

$$\min_{k \in P^a_{j^-}} \{st_k\} \tag{8}$$

この手続きから得られた $(j_{tabu}, st_{j_{tabu}})$ をタブーリストに記録し，再度 priority rule によってディスパッチを試みる．ただし，$j^* \in J$ に対し，(j^*, st_{j^*}) がタブーリストに記録されている場合，アクティビティ j^* の処理が時刻 st_{j^*} に開始されるのを禁止する．この繰返し処理は，実行可能解が得られるか，もしくは繰返し回数が事前に与えられた値に達した場合に終了する．PRTL で用いる優先度は，式 (9) で得られる値の昇順に従う．

```
procedure create_set_K(j⁻)
begin
  K := φ;
  minimum := INFINITY;
  if |P^w_{j⁻}| ≥ 1 then
  begin
   minimum := min{c_k + w_{kj⁻} | k ∈ P_{j⁻}};
   for each k ∈ P_{j⁻} do
   begin
    value := c_k + w_{kj⁻};
    if value = minimum then
     K := K ∪ {k};
   end
  end
  else
  begin
   minimum := min{st_k | k ∈ P^a_{j⁻}};
   for each k ∈ P^a_{j⁻} do
   begin
    value := st_k;
    if value = minimum then
     K := K ∪ {k};
   end
  end
end.
procedure get_tabu_activity(j⁻)
begin
  if P_{j⁻} ≠ {0} then
  begin
   create_set_K(j⁻);
   j_tabu := priority(K, 0)
  end
  else
  begin
   j_tabu := priority(S_0 \ {j⁻}, 0)
   for i = 0 to |S_0| − 1 do
   begin
    k := priority(S_0 \ {j⁻}, i)
    if exists([0, k]) and a_{0k} > 0 then
    begin
     j_tabu := k;
     break;
    end
   end
  end
end.
```

図 4 タブーリストに記録されるアクティビティの決定手続き
Fig. 4 Decision of actibity to be stored in tabulist.

```
procedure get_no_dispatch(J⁻)
begin
 for k = 1 to n − 1 do
 begin
  j := priority(J, k);
  if j ∈ J⁻ then
   return j;
 end
end.
```

図 5 集合 J^- からのアクティビティの選択
Fig. 5 Selection of activity from set J^-.

```
procedure PRTL()
begin
 for itr = 1 to ITR_MAX do
 begin
  set_priority(J);
  dispatching_in_PRTL(J);
  if not feasible(J) then
  begin
   j⁻ := get_no_dispatch(J⁻);
   record_tabulist(j⁻);
  end
 end
end.
```

図 6 PRTL
Fig. 6 PRTL.

$$\sum_{r=0}^{m-1}\sum_{u=0}^{p_j-1} d_{rju}, \quad j \in J. \tag{9}$$

式 (9) は，アクティビティ j の資源総要求量を表す．2 回目以降のディスパッチでは，優先度は課された猶予制約数の和の降順に従う．PRTL は，(部分) スケジュールと，数列 $[dsp]$ を出力する．集合 J^- を引数とし，J^- の要素から優先度が最も高いアクティビティを選択して返す処理を，get_no_dispatch として**図 5** に，PRTL の手続きを**図 6** に示す．ただし，手続き set_priority は，引数で与えられたアクティビティの集合 J の全要素に，優先度を与えるものである．record_tabulist は，j^- を引数として (j^-, st_{j^-}) をタブーリストに記録する手続きである．

以下のインスタンスを用いて，初期解構築の例を示す．各アクティビティのデータおよび再生型資源量は以下のとおりである．ただし，$d_{jr} = [d_{jr0}, \cdots, d_{jrp_j-1}]$ および $l_r = [l_{r0}, \cdots, l_{rt_{max}-1}]$ である．

アクティビティ数，資源種類数，プロジェクト期間

$n = 6$, $m = 1$, $t_{max} = 15$

再生型資源量

$l_r = [3,3,3,3,1,3,3,3,3,3,3,3,3,3,3,3]$

アクティビティ

- $j = 0$: $p_0 = 0$
- $j = 1$: $p_1 = 2, d_{11} = [1,2]$
- $j = 2$: $p_2 = 3, d_{21} = [1,2,2]$
- $j = 3$: $p_3 = 3, d_{31} = [1,2,2]$
- $j = 4$: $p_4 = 4, d_{41} = [3,3,2,2]$
- $j = 5$: $p_5 = 0$

猶予，待機制約

- $[0,1]$: $a_{01} = 0$
- $[0,2]$: $a_{02} = 0$
- $[1,3]$: $a_{13} = 3$
- $\langle 2,4\rangle$: $w_{24} = 2$
- $[3,5]$: $a_{35} = 0$
- $[4,5]$: $a_{45} = 0$

アクティビティ 0，5 は処理時間 0 のダミーである．これらは図のガントチャートには表記されない．

式 (9) に従い，アクティビティを優先度の降順に並べると，$(4,2,3,1)$ の順になる．$J^+ = \{0\}$ である．$P_4 \subset J^+$ を満たさないので，権利をアクティビティ 2 に譲渡する．$P_2 = \{0\} \subset J^+$ であるので，アクティビティ 2 のディスパッチを試みる．

$\max\{st_k + p_k + a_{kji} | k \in P_2\} = 0,$
$\min\{t_{max} - 1, st_k + p_k + w_{jik} | k \in P_2^w\} = 15$

であるので，$0 \le t \le 15$ かつ J^+ の要素間との猶予および待機制約と資源制約を満たす，最小の t を st_2 に与える．この場合，$st_2 = t = 0$ となる．また，$J^+ = \{0,2\}$ となる．次に，アクティビティ 3 は，$P_3 \subset J^+$ を満たさない．さらに次のアクティビティ 1 は，$P_1 \subset J^+$ を満たすので，ディスパッチを試みる．アクティビティ 1 は時刻 2 にディスパッチされる．$st_1 = 2$，$J^+ = \{0,1,2\}$ となる．再度，最大の優先度を持つアクティビティ 4 に権利を譲渡する．しかしアクティビティは 4 は，アクティビティ 1 との猶予制約と，資源制約を同時に満たす処理開始時刻が存在しない．ここで 1 回目のディスパッチが終了する．このときのガントチャートを，**図 7** に示す．アクティビティ 4 の先行アクティビティ 2 とその処理開始時刻，$(2,0)$ をタブーリストに記録する．

2 回目以降のディスパッチでは，アクティビティの優先度を，課せられた猶予制約数の降順で与える．優先度順に並べ替えると，$[2,4,1,3]$ となる．$J^+ = \{0\}$ とし，再度ディスパッチの手続きを実行する．同様の処理を行うと，タブーリストに $(2,0)$ が記録されているので，アクティビティ 2 は時刻 1 にディスパッチされる．次にアクティビティ 4 が権利を取得する．アクティビ

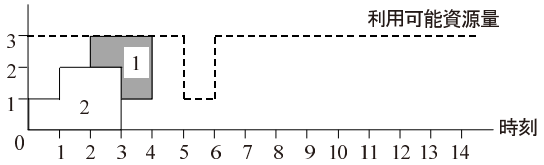

図 7 ディスパッチの失敗
Fig. 7 Failure of dispatching.

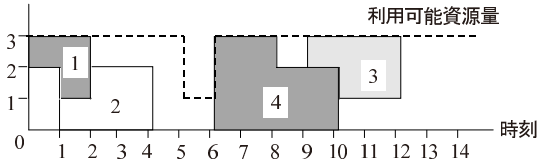

図 8 ディスパッチの完了
Fig. 8 End of dispatching.

ティ 2 との猶予制約を満たす処理開始時刻は，4，5，6 のいずれかであるが，このうち資源制約を満たす時刻は 6 であるので，$st_4 = 6$ となる．続けて，$st_1 = 0$，$st_3 = 9$ となり，実行可能なスケジュールが生成される．このときのガントチャートを，**図 8** に示す．ディスパッチされた順序を表す数列は，$[dsp] = [2, 4, 1, 3]$ となる．これらを出力し，PRTL は終了する．

4.2 解の改善

次に，PRTL で得られたスケジュールの改善を図る．PRTL で得られるスケジュールは，ディスパッチの順序に依存するため，この順序を並べ替え，再度解を構築する．ディスパッチ順序の変更を近傍操作と定義し，4.3 節において詳しく述べる．

近傍操作を行った後，再度ディスパッチを行う．ここでは，使用するディスパッチの手続きを，PRTL を基に次のように変更する．アクティビティの優先度に代わってディスパッチの順序を入力として与え，順序を変更してのディスパッチは許可しない．つまり，あるアクティビティがディスパッチされない限り，次のアクティビティにディスパッチの権利を譲渡しない，というものである．また，途中でコールされる get_no_dispatch を，集合 J^- から，ディスパッチの順序が最も早いものを選択するように変更する．その他は PRTL と同様である．この手法を，sequence rule with tabu list（SRTL）と呼ぶ．SRTL が実行可能解を構築できずに終了した場合は，式 (10) から得られる値を目的関数値として扱う．

$$\sum_{j=0}^{n-1} t_{max} \left(1 - \sum_{t=0}^{t_{max}-1} x_{jt} \right) \tag{10}$$

式 (10) は，t_{max} とディスパッチされなかったアクティビティ数との積を表す．実行可能解が構築された場合の目的関数値はつねに t_{max} 未満である．しかし，もしディスパッチされなかったアクティビティが存在する，すなわちすべての $t \in T$ に対して $x_{jt} = 0$ となるような j が存在する場合，式 (10) の値は t_{max} 以上となる．式 (10) を用いることにより，実行可能解が構築できなかった場合，ティスパッチされなかったアクティビティの数が少ないものを，より良い解として採用できる．

```
procedure dispatching_in_SRTL([dsp])
begin
  st_0 := 0;
  J^+ := φ;
  for itr := 1 to n − 1 do
   for h = 1 to n − 1 do
    if P_{dsp_h} ⊂ J^+ then
    begin
     t_from := max_k{c_k + a_jk | k ∈ P_{dsp_h}};
     t_to := min_k{c_k + w_jk | k ∈ P^w_{dsp_h}};
     for t := t_from to t_to do
      if not in_TL(dsp_h, t) then
       if satisfy(dsp_h, t) then
       begin
        st_{dsp_h} := t;
        J^+ := J^+ ∪ {dsp_h};
       end
    end
    else
     break;
   if feasible(J^+) then
    break;
end.
```

図 9 SRTL で用いるディスパッチの手続き
Fig. 9 Dispatching procedure used in SRTL.

SRTL で用いるディスパッチの手続きを dispatching_in_SRTL として，**図 9** に示す．ただし，ディスパッチの順序を示す数列 $[dsp]$ を入力として与えるものとする．

4.3 近傍

ここでは，RCPSP/τ+ の近傍を 3 つ定義する．本稿で提案するアルゴリズムで用いる近傍は，ディスパッチ順序を変更することによって定義される．i 番目にディスパッチされるアクティビティを $\pi(i)$ で表す．また，アクティビティ j のディスパッチ順序を $\sigma(j)$ と表すことにする．

1 つ目の近傍は，挿入近傍である．この近傍操作により，$\pi(i_1)$ のディスパッチ順序は，$i_2 (i_1 \neq i_2)$ へと変更される．それにともない，$i_1 < i_2$ であれば，$i_1 < k \leq i_2$ を満たすすべての k において，$\sigma(\pi(k))$ を 1 減らす．$i_1 > i_2$ であれば，$i_2 \leq k < i_1$ を満た

```
procedure neighbourhood_insert(i1, i2)
begin
  j' := π(i1);
  k := i1;
  if i1 < i2 then
   for k = i1 to i2 − 2 do
    π(k) := π(k + 1);
  else
   for k := i1 downto i2 + 1 do
    πk := πk−1
  π(i2) := j';
end.
```

図 10 挿入近傍による近傍操作
Fig. 10 Neighborhood operation with insertion neighborhood.

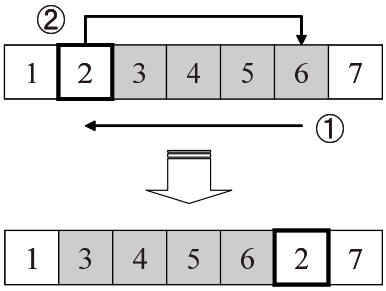

図 11 挿入近傍
Fig. 11 insertion neighborhood.

すすべての k において，$\sigma(\pi(k))$ を 1 増やす．挿入近傍操作を neighbourhood_insert とし，**図 10** に示す．入力として，i_1 および i_2 が与えられているものとする．

$i_1 = 2$，$i_2 = 6$ の場合の例を，**図 11** に示す．

2 つ目は，2-swap 近傍である．この近傍操作は，2 つのアクティビティを選択して，それらのディスパッチ順序を入れ替える．入力として，2 つのアクティビティのディスパッチ順序 i_1，i_2 が与えられているものとする．2-swap 近傍操作を neighbourhood_2-swap とし，**図 12** に示す．

$i_1 = 3$，$i_2 = 6$ の場合の例を，**図 13** に示す．

3 つ目は，3-opt 近傍である．この近傍操作は，3 つのアクティビティを選択して，ディスパッチ順序を変更する．入力として，3 つのアクティビティのディスパッチ順序 i_1，i_2，i_3（$0 < i_1 < i_2 < i_3 < n-1$）が与えられているものとする．3-opt 近傍操作を neighbourhood_3-opt とし，**図 14** に示す．

$i_1 = 2$，$i_2 = 5$，$i_3 = 7$ の場合の例を，**図 15** に示す．

```
procedure neighbourhood_2-swap(i1, i2)
begin
  j1' := π(i1);
  j2' := π(i2);
  π(i1) := j2';
  π(i2) := j1';
end.
```

図 12 2-swap 近傍による近傍操作
Fig. 12 Neighborhood operation with 2-swap neighborhood.

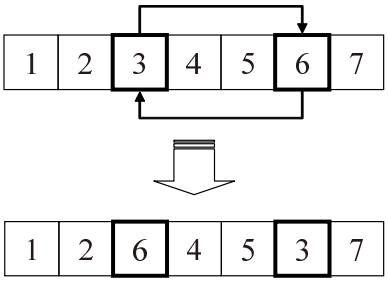

図 13 2-swap 近傍
Fig. 13 2-swap neighborhood.

```
procedure neighbourhood_3-opt(i1, i2, i3)
begin
  for i := 1 to n − 2 do
   ji := π(i);
  k := 0;
  for i := 0 to n − 1 do
  begin
   π(i) := jk;
   case k of
    i1 :
     k := i2 + 1;
    i2 :
     k := i3 + 1;
    i3 :
     k := i1 + 1;
   else
    k + +;
  end
end.
```

図 14 3-opt 近傍による近傍操作
Fig. 14 Neighborhood operation with 3-opt neighborhood.

4.4 タブーサーチ

4.3 節で導入した 3 つの近傍を用いたローカルサーチを基にした，タブーサーチアルゴリズム（以下，TS）を実装した．4.1 節で述べたように，初期解は PRTL から得る．各近傍に対して，使用するタブーリストの属性はそれぞれ異なる．挿入近傍で用いる属性は，挿入

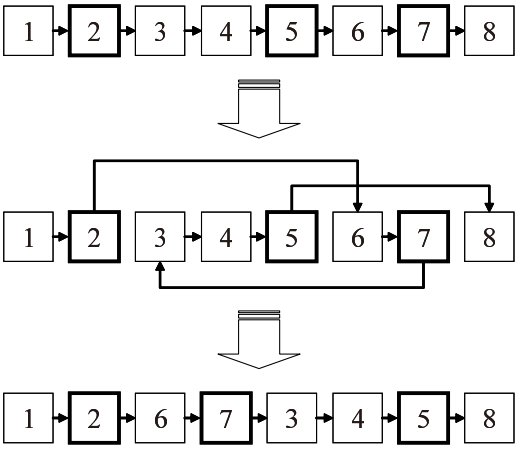

図 15 3-opt 近傍
Fig. 15 3-opt neighborhood.

```
procedure extend_precedence(j, s)
begin
  if 0 ≤ random() < 0.1 then
  begin
   w_js := ⌈(p_j + s_j) * random() * 3⌉;
   set(⟨j, s⟩, w_js);
  end
  else if 1 ≤ random() < 0.2 then
  begin
   a_js := ⌈(p_j + s_j) * random() * 3⌉ ;
   set([j, s], a_js);
  end
  else
   set([j, s], 0);
end.
```

図 16 猶予，待機制約の生成
Fig. 16 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
   if 0 ≤ random() < 0.4 then
    d_jru := d*_jr;
   else if 0.4 ≤ random() < 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 17 要求資源量の生成
Fig. 17 Generation of resource demand.

操作を行うアクティビティの，番号および挿入後のディスパッチ順序である．タブーリストに，$(j, \sigma(j)), j \in J$ が記録されている場合，ディスパッチ順序が $\sigma(j)$ であるアクティビティ j が，挿入近傍操作の入力とされることを禁止する．2-swap 近傍の場合では，ディスパッチ順序の交換を行う，2 つのアクティビティの番号である．タブーリストに，$(j_1, j_2), j_1, j_2 \in J$ が記録されている場合，2-swap 近傍操作の入力として，$\sigma(j_1)$，$\sigma(j_2)$ の組合せを用いることを禁止する．3-opt 近傍の場合は，近傍操作の入力として与えられる 3 つのアクティビティの，ディスパッチ順序を用いる．タブーリストに，(i_1, i_2, i_3) が記録されている場合，3-opt 近傍操作の入力として i_1，i_2 および i_3 の組合せを用いることを禁止する．近傍内の解への移動戦略は，即時移動戦略を用いた．すなわち，探索中に目的関数値を改善する実行可能解が見つかった場合，即座にその解に移動する．ただし，近傍内に改善解が発見できない場合は，最良移動戦略を用いる．すなわち，改悪解しか発見できない場合なので，移動前と移動後との目的関数値差が最小となる解に移動する．終了条件は，近傍操作繰返し回数の，規定数超過とする．規定数はあらかじめ与えられる．

5. 数 値 実 験

4 章で述べたアルゴリズムから得られる解の精度を，最適解と比較することによって検証する．数値実験に用いた試行対象は，250 個の RCPSP/τ+ インスタンスである．これらのインスタンスは，*PSPLIB* に掲載されているベンチマーク用インスタンスを基に，ランダムに生成したものである．猶予，待機制約の生成には，**図 16** 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/τ+ インスタンスに課す手続きである．また，サブルーチン random は，0 以上 1 未満の乱数を返す．

$d_{jru} (j \in J, r = 0, \ldots, m-1, u = 0, \ldots, p_j - 1)$ は，**図 17** 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

```
procedure extend_resource_limit()
begin
  for r := 0 to m − 1 then
   for t := 0 to t_max − 1 then
    rcv_t := 0;
   for t := 0 to t_max − 1 then
    if 0 ≤ random() < 0.5 then
    begin
     ρ := ⌈l*_r ∗ random()⌉
     if t + 5 < t_max then
     begin
      rcv_{t+5} += ρ;
      l_rt = l*_r − ρ + rcv_t;
     end
     else
      l_rt = l*_r − ρ;
    end
end.
```

図 18 使用可能資源量の生成
Fig. 18 Generation of resource limit.

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 18** に示された処理 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．

このうち 200 題は，$n = 12$，22，32，42 のいずれかであり，これらを set1 と呼ぶことにする．また，残り 50 題は，$n = 62$ であり，これらを set2 と呼ぶことにする．いずれも，n は，ダミーアクティビティを含む数である．また，すべてのインスタンスに対し，$t_{max} = 400$ である．最適解の取得には，ILOG CPLEX 8.0 を用いた．ただし各インスタンスに対して，計算時間の上限を 50,000 秒としており，これを超える場合は，最適性の保障は得られていない．これらの条件から得られた解を，比較対象として用いる．set1 の 200 インスタンス中，11 インスタンスは実行可能解を持たなかった．残り 189 インスタンス中，ILOG CPLEX によって最適性を保障されたインスタンスは 175 個であった．set2 のすべてのインスタンスは，実行可能解を持つ．

PRTL および SPMT2 における繰返しの上限回数はそれぞれ 500 である．各近傍を用いた TS において，本実験で用いたタブーリストの長さはそれぞれ，挿入および 2-swap 近傍の場合は 50，3-opt 近傍の場合は 10 である．

各近傍を単体で用いた場合の実行結果を，set1 を試行対象として，**表 1** に示す．本稿で提案される各アルゴリズムの実行環境は以下のとおりである．

- OS　：Windows XP SP1
- CPU　：Pentium4 2.6 GHz
- メモリ：1 GB
- 言語　：言語 C

表 1 中の “実行可能” 列は，各アルゴリズムから得られた実行可能解数を示す．また，“最適” 列は，得られた最適解の数を示す．“誤差” 列は，最適解との誤差 [%] の平均値を示す．“PRTL” 行は，PRTL のみの実行結果である．TS は実行されていない．“挿入” 行は，挿入近傍を用いた TS を示す．同様に，“2-swap” および “3-opt” 行はそれぞれ，2-swap および 3-opt 近傍を用いた TS を示す．各 TS の終了条件となる返し規定数は，100 である．

表 2 の各列については，表 1 と同様である．各行における “挿入”，“2-swap” および “3-opt” は，それぞれの近傍を用いた TS を示す．また，“→” は，近傍を変更して TS を続行したことを示す．たとえば “挿入 → 2-swap” は，挿入近傍による TS 実行後，2-swap 近傍による TS を実行したことを示す．近傍の変更を行う場合は，各近傍に対して，終了条件となる繰返し上限数を 50 としている．

表 2 から，2-swap → 3-opt および 3-opt → 2-swap による TS が，良い精度の解を出力することが分かる．

set1 を試行対象とした実行時間の平均を，アクティビティ数ごとに**表 3** に示す．単位は秒である．

表 3 における “CPLEX” 行は，ILOG CPLEX での実行結果を示す．表 3 から，各アルゴリズムともに，ILOG CPLEX に比べ，高速に動作することが分かる．次に，set2 を試行対象とした，2-swap → 3-opt およ

表 1 初期解と TS の比較
Table 1 Comparison of initial solution and each TS.

	実行可能	最適	誤差 [%]
PRTL	188	51	11.45
挿入	189	111	3.67
2-swap	188	123	2.54
3-opt	189	123	2.70

表 2 2 つの近傍を用いた TS の実行結果
Table 2 Tabu search using two neighborhoods.

	実行可能	最適	誤差 [%]
挿入 → 2-swap	189	127	2.04
挿入 → 3-opt	189	127	2.37
2-swap → 挿入	188	132	1.84
2-swap → 3-opt	188	134	1.68
3-opt → 挿入	189	126	2.34
3-opt → 2-swap	189	136	1.83

表 3 平均実行時間
Table 3 Average cpu time.

algorithm	$n = 22$	$n = 32$	$n = 42$
CPLEX	193.65	6410.30	8543.76
PRTL	0.01	0.02	0.02
挿入	1.37	18.37	13.89
2-swap	0.35	7.06	5.71
3-opt	4.07	45.64	39.40
挿入 → 2-swap	1.07	14.21	10.68
挿入 → 3-opt	2.81	33.49	27.73
2-swap → 挿入	12.64	72.25	109.81
2-swap → 3-opt	1.56	25.80	23.64
3-opt → 挿入	2.79	32.67	27.76
3-opt → 2-swap	2.34	27.34	23.17

表 4 set2 を対象とした実行結果
Table 4 Result on problem set2.

	実行可能	実行時間 [sec]	誤差 [%]
CPLEX	44	23558.61	-
2-swap → 3-opt	50	354.46	0.65
3-opt → 2-swap	50	351.97	0.16

び 3-opt → 2-swap による TS の実行結果を，**表 4** に示す．比較対象は，ILOG CPLEX での実行結果である．“実行時間” 列は，計算時間の平均値 [sec] を示している．“実行可能” および “誤差” の各列は，表 1 および 2 と同様である．

set2 の 50 インスタンス中，ILOG CPLEX は 44 インスタンスに対して実行可能解を出力した．残り 6 インスタンスに対しては，実行可能解を発見する前に計算時間が 50,000 秒を超えたため，計算が打ち切られた．また，実行可能解を得られていても，最適性の保証が得られなかったインスタンスがいくつか存在した．それに対し TS は，50 インスタンスすべてに対して実行可能解を出力した．$n = 62$ であっても，TS は良い精度の実行可能解を出力することが分かる．

6. 今後の課題

本稿で提案したアルゴリズムには，ディスパッチの手続きを繰り返す回数やタブーリストの長さなど，いくつかのパラメータを与える必要がある．これらのパラメータは，アルゴリズムの性能に，少なからず影響を与えると思われる．よって，より良いパラメータの設定が必要である．解の改善フェイズに用いる SRTL は，インスタンスによっては，いかなるディスパッチ順序を与えても最適解を出力できない可能性がある．本稿では，比較的アクティビティ数が少ないインスタンスによって数値実験を行ったが，現実の生産現場から得られるインスタンスは，非常に大規模であると思われる．よって，より大規模なインスタンスでの検証およびアルゴリズムの高速化が必要になるであろう．また，近傍操作の後，再度スケジュールの構築を試みるので，スケジュールに対して直接変更を加えることに比べると，計算時間が多くなる．これらの点を，今後の課題とする．

7. ま と め

これまでの生産活動の発展の様子から，モデルはより複雑化し，規模はより大きくなっていくことが考えられる．そのような状況では，解の精度と計算時間の両立が，これまで以上に大きな要求として現れてくるだろう．精度の良い解を得るために必要な点として，現場固有の制約条件を取り入れる，つまりモデル化の際になるべく簡略化をしないことがあげられる．今後，これまでの古典的モデルではモデル化できないスケジューリング問題に関する研究は大きな広がりを見せると思われる．このようなモデルの 1 つとして，我々は本稿において，RCPSP/τ+ モデルを提案し，0-1 整数計画問題として定式化した．また，解の精度と計算時間の両立を実現する手法の 1 つとして，近年，メタヒューリスティクスが大きな発展をとげている．本稿は，メタヒューリスティクスの 1 つである，タブーサーチアルゴリズムを実装し，その性能を評価した．初期解の構築法として，PRTL を提案した．数値実験の結果より，PRTL によって，多くのインスタンスで実行可能解を得られることが分かった．解の改善における局所探索に用いる近傍は，アクティビティをディスパッチする順序を変更することにより挿入，2-swap，および 3-opt の 3 種類を定義した．これは，2 つのアクティビティの処理開始時刻を交換するといった，解の直接操作が難しいことに起因する．与えられたディスパッチ順序に従ってスケジュールを構築する手法として，SRTL を用いた．実験により，これらの近傍および SRTL を実装したタブーサーチにより，PRTL から得られた解が大きく改善されることが分かった．

参 考 文 献

1) Hartmann, S.: *Project Scheduling under Limited Resources Models, Methods and Applications*, Springer-Verlag Berlin, Heidelberg (1999).
2) Joel, P., Stinson, E.W.D. and Khumawala, B.M.: Multiple Resource-Constrained Scheduling Using Branch and Bound, *AIIE Trans.*, No.8, pp.252–259 (1978).
3) Kolish, R.: Efficient priority rules for the resource-constrained project scheduling prob-

lem, *Journal of Operations Management*, Vol.14, pp.179–192 (1996).
4) Kolish, R. and Sprecher, A.: PSPLIB — A project scheduling problem library, *European Journal of Operational Research*, Vol.96, pp.205–216 (1996).
5) N. Christofides, 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).
6) Nonobe, K. and Ibaraki, T.: Formulation and tabu search algorithm for the resource constrained project scheduling problem, *Essays and Surveys in Metaheuristics* (*MIC'99*), pp.557–588 (2002).

(平成 18 年 11 月 22 日受付)
(平成 19 年 1 月 12 日再受付)
(平成 19 年 2 月 19 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系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．