数理解析研究所講究録
1120巻 1999年 88-97



# 資源制約付きスケジューリング問題の定式化と近似解法

## Formulation and Tabu Search Algorithm for the Resource Constrained Project Scheduling Problem (RCPSP)

野々部 宏司　　茨木 俊秀
Koji NONOBE　Toshihide IBARAKI

京都大学大学院情報学研究科
〒606-8501 京都市左京区吉田本町
{nonobe,ibaraki}@kuamp.kyoto-u.ac.jp

**摘要:** 資源制約付きスケジューリング問題 (RCPSP) は, 多くのスケジューリング問題を定式化できるという汎用性の高さから最近改めて注目を集めている. しかし現実の応用分野では, スケジュールの評価基準が複雑かつ多様であるなど, 従来のRCPSPの枠組では扱うことのできない問題も少なくない. そこで, 広範なスケジューリング問題に対応できるよう, RCPSPの定式化を拡張するとともに, タブー探索に基づく近似解法を提案する. また, 大きな構造物を扱う工場におけるスケジューリング問題を例にとり, 本RCPSPソルバの現実問題への適用について述べる.

**キーワード: 資源制約付きスケジューリング問題, 汎用アルゴリズム, 近似解法, タブー探索.**

## 1 はじめに

資源制約付きスケジューリング問題 (Resource Constrained Project Scheduling Problem, RCPSP) は, フローショップ問題やジョブショップ問題をはじめ, 多くのスケジューリング問題を定式化できるという汎用性の高さから, 最近改めて注目を集めている [1, 2, 4, 5]. しかし現実の応用分野では, スケジュールの評価基準が複雑かつ多様である, 平日と休日とでは使用できる資源の量が異なる, 段取り替え作業を伴うなど, 従来のRCPSPの枠組では扱えない問題も少なくない. 本研究の目的は, 多くのスケジューリング問題を扱うことのできる, 汎用RCPSPソルバを開発することである. そのために, まず, より広範なスケジューリング問題を定式化できるようRCPSPを拡張し, その拡張RCPSPに対しタブー探索の適用を行う. このRCPSPソルバを用い, ベンチマーク問題を解いたところ, 多くの問題例に対して, これまでの最良解を更新することに成功した. また, 幾つかの現実問題に対しても計算実験を行った. 本論文では, その一例として, 大規模な構築物を対象とするスケジューリング問題に対する適用について述べる.

## 2 定式化

本論文では, RCPSPを以下のように定式化する. 資源集合 $R = R^{re} \cup R^{non}$, 作業集合 $J$ が与えられている. 資源集合 $R$ は, 再生可能 (renewable) 資源集合 $R^{re}$ と再生不可能 (nonrenewable) 資源集合 $R^{non}$ に分類される. 再生可能資源 $r \in R^{re}$ は, 各単位時間 $(t-1, t]$ $(t = 1, 2, \ldots)$ に利用できる量がそれまでのスケジュールに依らないのに対し, 再生不可能資源 $r \in R^{non}$ は, 各単位時間当たりではなく, スケジュール全体を通して利用できる量が決まっている. 例えば, 機械, 人, 作業場所などが再生可能資源, 原材料などが再生不可能資源に相当する. また, 各作業 $j$ に対して, その処理方法を表す処理モードの集合 $M_j$ が与えられる. 作業 $j$ はその処理モード集合 $M_j$ に属するいずれか一つの処理モード $m_j$ で処理され, その際の処理時間 $p_{m_j}$, 処理に必要な資源集合 $R^{re}_{m_j} \cup R^{non}_{m_j}$, およびそれらの消費量が与えられている.

スケジュールは, 各作業 $j$ の処理モード $m_j \in M_j$, および 開始時刻 $s_j$ (完了時刻 $c_j = s_j + p_{m_j}$) に基づいて, $(\boldsymbol{m}, \boldsymbol{s}) = ((m_j \mid j \in J), (s_j \mid j \in J))$ で表

される.

### 2.1 資源制約

(i) 再生可能資源制約: 各単位時間 $(t-1,t]$ $(t=1,2,\ldots)$ における再生可能資源 $r \in R^{re}$ の総利用量が, その供給量を越えてはならない. なお, 資源 $r$ の供給量, および利用量は一定でなくてもよい.

代表的な再生可能資源の例として, 機械が挙げられる. 同時に高々1 つの作業しか処理できない機械は, その供給量, 利用量ともに, 常に 1 である再生可能資源として扱うことができる.

(ii) 再生不可能資源制約: 各資源 $r \in R^{non}$ に対して, その総利用量が与えられた供給量を越えてはならない. 各作業が必要とする再生不可能資源の量は, その開始時刻によらず, 処理モードにのみ依存する. よって, 複数の処理モードを持つ作業に対してのみ, 再生不可能資源制約は意味をなす.

### 2.2 先行制約

(i) 先行制約 $j_1 \prec j_2$: 作業 $j_1$ が完了するまで, 作業 $j_2$ の処理を開始できない. すなわち,

$$c_{j_1} \leq s_{j_2}. \tag{1}$$

(ii) 資源 $r \in R^{re}$ 上の直前先行制約 $j_1 \prec\prec_r j_2$: 作業 $j_1$ は $j_2$ に先行し, さらに, $j_1, j_2$ が共に資源 $r$ を消費するならば, $j_1$ は $j_2$ の直前先行作業でなくてはならない. すなわち,

$$\begin{array}{l} c_{j_1} \leq s_{j_2} \text{ かつ} \\ s_{j_2} \leq s_{j'} \text{ for all } j' \text{ s.t. } r \in R^{re}_{m_{j'}},\ c_{j_1} \leq s_{j'}. \end{array} \tag{2}$$

この直前先行制約は, 先行制約 (i) の一般化であり, 段取り替え作業をはじめ, $j_1$ と $j_2$ の処理は同時に開始されなくてはならない, $j_1$ が完了後, 直ちに $j_2$ の処理を始めなくてはならない, などの制約を (仮想作業を導入することで) 記述することができる. 段取り替え作業については, 次節でやや詳しく述べる.

### 2.3 段取り替え作業

現実の問題では, ある機械 $r$ 上で作業 $j_1$ から 作業 $j_2$ に処理を移行する際, $j_1$ と $j_2$ に依存した段取り替え作業 $setup(j_1, j_2)$ が必要となることが多い. このとき, 段取り替え作業 $setup(j_1, j_2)$ は, 機械 $r$ 上において作業 $j_2$ の直前に処理されなくてはならず, その間, 機械 $r$ 上で他の作業を処理することはできない.

本定式化においては, 以下のように段取り替え作業を扱う. $j_2$ の段取り替え作業に対応する作業 $\sigma(j_2)$ を導入し, 直前先行制約 $\sigma(j_2) \prec\prec_r j_2$ を加える. また, $\sigma(j_2)$ の処理モードは, 機械 $r$ 上で直前に処理される作業 $prev(\sigma(j_2))$ に依存して決まる. 本アルゴリズムでは, $prev(\sigma(j_2))$ に応じて $\sigma(j_2)$ がどの処理モードをとるか, 予め指定しておくことにより, 常に, 段取り替え作業 $\sigma(j_2)$ が適切な処理モードをとるようにしている. なお, 処理モードの決定以外においては, 段取り替え作業も通常の作業と同様に扱われる.

### 2.4 目的関数

現実のスケジューリング問題では, その評価基準は, 最大完了時刻最小, 総納期遅れ時間最小など, 状況に応じて変化するため, 予め目的関数を定めておくことは好ましくない. また, より複雑な評価基準を用いることも多い. 本定式化では, 上述の制約に加えて, 各作業 $j$ の処理モード $m_j$, 開始時刻 $s_j$, 完了時刻 $c_j$, および処理時間 $p_{m_j}$ に関する任意の付加制約を許し, それらの違反度により解 (スケジュール) の評価を行う. 現時点での我々のコードでは, 以下の線形不等式制約

$$\sum_{j \in J} \sum_{m_j \in M_j} \alpha_{j,m_j} x_{j,m_j} + \sum_{j \in J} \beta_j s_j \leq \gamma \tag{3}$$

が利用可能である. ここで, $\alpha_{j,m_j}$, $\beta_j$ および $\gamma$ は係数であり, $x_{j,m_j}$ は, 作業 $j$ が処理モード $m_j$ で処理される (されない) ならば 1 (0) をとる 0-1 変数である. (作業 $j$ の処理時間, および完了時刻はそれぞれ $\sum_{m_j} p_{m_j} x_{j,m_j}$, および $\sum_{m_j} p_{m_j} x_{j,m_j} + s_j$ で表される.) 資源制約, 先行制約, およびこれらの付加制約は, 必ず満たさなくてはならない制約 (絶対制約) と満たすことが望ましいが必ずしも満たさなくてもよい制約 (考慮制約) の 2 種類に分類され, 絶対制約を満たす (以下, 実行可能であると呼ぶ) 範囲で, 考慮制約の満足度を最大化する. これを実現するために, 各考慮制約 $C_\ell$ に対して, それが満たされる (満たされない) ならば 0 (正の値) をとるペナルティ関数 $p_\ell(\boldsymbol{m}, \boldsymbol{s})$, および $C_\ell$ の重要度を示す非負の重み $w_\ell$ を導入する. これらを用いて, 全体としての重み

つきペナルティ関数

$$p(\boldsymbol{m},\boldsymbol{s}) = \sum_{\ell} w_\ell p_\ell(\boldsymbol{s},\boldsymbol{m}) \qquad (4)$$

を最小化する.

この定式化において, どの制約を絶対制約として記述するか, 各考慮制約 $C_\ell$ の重み $w_\ell$ をどのように設定するかなどは, ユーザにより決定される. これらの決定は注意深く行われなくてはならないが, この柔軟な枠組みにより, 多様な問題を扱うことが可能になる.

ここで, しばしば用いられる目的関数の例を 2 つ示す.

最大完了時刻最小化: 最大完了時刻 (makespan) 最小化は, 作業 $sink$ の完了時刻 $c_{sink}$ に対する以下の不等式制約

$$c_{sink} \leq 0 \qquad (5)$$

を考慮制約として記述することで実現できる. ここで作業 $sink$ は全ての作業に後続する処理時間 0 の仮想的な作業であり, この場合, 値 $c_{sink}$ がペナルティとなる.

納期ずれ最小化: 作業 $j$ の納期を $d_j$ とし, 納期ずれに対するコスト関数を $cost\_due(c_j, d_j)$ とする. 任意の関数 $cost\_due(c_j, d_j)$ に対して, その最小化は, 以下のような再生可能資源制約 $r_{\mathrm{due}}(j)$ を考慮制約として導入することで記述できる. 資源 $r_{\mathrm{due}}(j)$ は, 作業 $j$ にのみ消費され, その量は, 最後の単位時間のみ $k = \max_t cost\_due(t, d_j)$, それ以外は 0 とする. また, $r_{\mathrm{due}}(j)$ の供給量 $K_{r_{\mathrm{due}}(j),t}$ $(t = 1, 2, \ldots)$ を, $k - cost\_due(t, d_j)$ とする. これにより納期ずれのコストは資源 $r_{\mathrm{due}}(j)$ の不足量, すなわちペナルティとして表される.

# 3 アルゴリズム

これまでに, RCPSP に対して多くの近似解法が提案されており, 中でも局所探索法, およびその変形・拡張であるメタ解法に基づくアルゴリズムの有用性が示されている [1, 2, 4]. 本論文でも, RCPSP ソルバの枠組みとして, メタ解法の一つであるタブー探索を用いる.

## 3.1 探索空間

2章で述べた通り, 解は各作業 $j$ の処理モード, および開始時刻を表すベクトル $(\boldsymbol{m},\boldsymbol{s})$ で表される. しかし, 一般に, ある実行可能解 $(\boldsymbol{m},\boldsymbol{s})$ から, 一つの作業 $j$ の開始時刻 $s_j$ や処理モード $m_j$ を変化させて得られる解 $(\boldsymbol{m}',\boldsymbol{s}')$ は, 実行可能であるとは限らない. そのため, $(\boldsymbol{m}',\boldsymbol{s}')$ から, 再び実行可能解 $(\boldsymbol{m}'',\boldsymbol{s}'')$ を得るためには, 他の作業の開始時刻の変更を含めた複数の局所的変更が必要となる. したがって, 局所的変更の定義を拡張して, 一回の変更で実行可能解 $(\boldsymbol{m}'',\boldsymbol{s}'')$ が得られるとすれば, より効果的な局所探索が期待できる. そこで本論文では, 絶対制約のうち, 再生可能資源制約, 先行制約, 直前先行制約, および段取り替え作業の処理モードに関する制約を全て満たす解 (以下, 疑似実行可能解と呼ぶ) に限定して局所探索を行うことを考える. そのために, 全作業の順列 $\boldsymbol{\pi} = (\pi(1), \pi(2), \ldots, \pi(|J|))$ を用意し, $\boldsymbol{\pi}$ に従って開始時刻 $s_{\pi(i)}$ を前から順に決定していくことによりスケジュールを構成する. 以下, 簡単化のため直前先行制約, 段取り替え作業はないものと仮定して, その手順を説明する. (実際には, 直前先行制約, および段取り替え作業の処理モードに関する制約も満たすよう実装している. 詳しくは文献 [8] を参照されたい.) 各作業 $\pi(i)$ の開始時刻 $s_{\pi(i)}$ を, $s_{\pi(1)}, s_{\pi(2)}, \ldots, s_{\pi(i-1)}$ はすでに決定しているという状況で, 疑似実行可能性を満たしつつ最早時刻に定めていく.

**CONSTRUCT**
入力: $(\boldsymbol{m},\boldsymbol{\pi})$, 出力: $(\boldsymbol{m},\boldsymbol{s})$.

**ステップ** 0 **(初期設定)** : $i := 1$.

**ステップ** $i$ : 作業 $\pi(i)$ に対し, 疑似実行可能となる最早開始時刻を計算し $s_{\pi(i)}$ とする. $i < |J|$ ならば $i := i+1$ としてステップ $i$ へ. さもなければ終了.

なお, 順列 $\boldsymbol{\pi}$ から CONSTRUCT により疑似実行可能スケジュールを得るために,

$$\pi(i_2) \not\prec \pi(i_1), \qquad 1 \leq i_1 < i_2 \leq |J| \qquad (6)$$

を仮定する. 条件 (6) を満たす順列 $\boldsymbol{\pi}$ の集合を $\Pi$ と記す. 本タブー探索では, 探索空間として

$$\left\{(\boldsymbol{m},\boldsymbol{\pi}) \mid m_j \in M_j,\ \boldsymbol{\pi} \in \Pi\right\} \qquad (7)$$

を用い, 再生可能資源制約, 先行制約以外の絶対制約は, 十分大きな重みを持つ考慮制約として扱い, それらを考慮に入れたペナルティ関数を目的関数とする. もちろん, 探索の途中, それらの絶対制約を満たさない解を訪問することもあるが, 最終的に得られる暫定解 (探索中に見つかった最良解) はそれらを満たしていると期待できる.

ここで, 探索空間を (7) とすることで探索効率を高めている反面, 問題例によっては最適スケジュールを求めることが原理的に不可能である場合も存在することに注意する必要がある.

### 3.2 近傍

以下の 3 種類の局所的な変更のいずれかを, 解 $(\boldsymbol{m}, \boldsymbol{\pi})$ に適用することによって得られる解の集合を $(\boldsymbol{m}, \boldsymbol{\pi})$ の近傍 $N(\boldsymbol{m}, \boldsymbol{\pi})$ とする:

(i) ある作業 $j$ の処理モード $m_j$ を他の処理モード $m'_j \in M_j$ に変更する,

(ii) 順列 $\boldsymbol{\pi}$ において, ある作業 $\pi(i_1)$ を作業 $\pi(i_2)$ $(i_1 < i_2)$ の直後に移す,

(iii) 順列 $\boldsymbol{\pi}$ において, ある作業 $\pi(i_2)$ を作業 $\pi(i_1)$ $(i_1 < i_2)$ の直前に移す.

ただし (ii) においては, 新しい順列 $\boldsymbol{\pi}'$ が $\boldsymbol{\pi}' \in \Pi$ を満たすように, $\pi(i_1) \not\prec \pi(i_2)$ でなくてはならず, さらに, $i_1 < i' < i_2$, $\pi(i_1) \prec \pi(i')$ となる作業 $\pi(i')$ が存在する場合, $\pi(i')$ も $\pi(i_2)$ より後ろに移動する. (iii) についても同様である.

ここで, 局所探索中, 近傍 $N(\boldsymbol{m}, \boldsymbol{\pi})$ 内の解全てを候補に探索を行うことは, 各解の評価に時間がかかることを考慮すると非効率的である. そこで, 現在の解 $(\boldsymbol{m}, \boldsymbol{\pi})$ が満足していない制約 $C_\ell$ に注目し, そのような $C_\ell$ の少なくとも一つを満たす効果があると思われる局所的変更のみを試みる. 例えば, 考慮制約 $C_\ell$ が再生可能資源制約 $r \in R^{re}$ であり, ある単位時間 $(t-1, t]$ で, 消費量が供給量を越えているとする. このとき, $(t-1, t]$ において $r$ を消費している作業 $j$ に対して, $\pi(i_1) = j$ $(\pi(i_2) = j)$ として変更 (ii) (変更 (iii)) を適用する. また, $C_\ell$ が再生不可能資源制約など, 処理モードに関わる制約であれば, 変更 (i) を試みる. さらに, ある作業 $j'$ の開始時刻を早めることが求められている場合, 以下のような変更を試みることも効果的である: 解 $(\boldsymbol{m}, \boldsymbol{\pi})$ に対して,

$$A_P = \{(j_1, j_2) \mid j_1 \prec j_2 \text{ かつ } c_{j_1} = s_{j_2}\},$$

$$A_R = \left\{ (j_1, j_2) \;\middle|\; \begin{array}{l} \text{作業 } j_2 \text{ の開始時刻が, 作業 } j_1 \\ \text{との資源の競合により遅れた} \end{array} \right\}$$

とする. 有向グラフ $(J, A_P \cup A_R)$ において, 上述の作業 $j'$ に至る有向路上にあり, $(j_1, j_2) \in A_R \setminus A_P$ である枝 (作業の組) に対して (ii) を適用する.

このように, 試行される局所的変更の定義は考慮制約 $C_\ell$ のタイプに依存して決定されるが, その方法は一意ではない. 我々の実装におけるこれらの詳しい定義については, [8] を参照されたい.

### 3.3 タブー探索

上述の探索空間, 近傍を用いてタブー探索 [3] を行う. タブーリストには, 局所的変更 (i) に対しては作業 $j$ を, (ii) に対しては作業 $j_1$, (iii) に対しては作業 $j_2$ をそれぞれ保持する. また, 本アルゴリズムでは, タブー探索の基本的要素に加え, 文献 [7] と同様の方法でパラメータ tabu tenure の自動調節を行っている.

## 4 ベンチマーク問題に対する計算実験

前章で述べた RCPSP アルゴリズムを用いて, ベンチマーク問題に対する計算実験を行った. なお, アルゴリズムは C 言語で記述し, 計算実験は全てワークステーション Sun Ultra2 (300MHz) 上で行った. PSPLIB[6] (Project Scheduling Problem LIBrary) には, RCPSP のベンチマーク問題が数多く用意されており, 問題例やこれまでの最良値などがホームページ[1] から入手できる. 計算実験では, その中から j60.sm, j90.sm, j120.sm, j30.mm の 4 タイプ, 計 2110 個の問題例を用いた. 各タイプのサイズ等は以下の通りである. また, 4 タイプとも先行制約付きであり, 最大完了時刻を最小化することが目的である.

| 問題例のタイプ | $\lvert J\rvert$ | $\lvert M_j\rvert$ | $\lvert R^{re}\rvert$ | $\lvert R^{non}\rvert$ |
|---|---|---|---|---|
| j60.sm | 60 | 1 | 4 | 0 |
| j90.sm | 90 | 1 | 4 | 0 |
| j120.sm | 120 | 1 | 4 | 0 |
| j30.mm | 30 | 3 | 2 | 2 |

[1] http://www.bwl.uni-kiel.de/Prod/psplib/index.html

表 1: PSPLIB に対する計算結果.

| タイプ | 問題例の数 | 最良値を求めた数[1] | 最大反復回数 | 平均計算時間 |
|---|---|---|---|---|
| j60.sm | 480 | 371 (1) | 10000 | 26.5 [秒] |
| j90.sm | 480 | 369 (8) | 30000 | 181.3 [秒] |
| j120.sm | 600 | 219 (50) | 30000 | 641.7 [秒] |
| j30.mm | 550 | 382 (57) | 10000 | 33.6 [秒] |

1. (): 最良値を更新した問題例の数.

最大反復回数を 1 万回, あるいは 3 万回とし, 各問題例に対して, 1 回ずつタブー探索を行った. その結果を表 1 に示す. 多くの問題例に対して最良値を求めることができ, さらに, 幾つかの問題例に対してはこれまでの最良値を更新することができた. また, 多くの計算時間をかけることで, さらに多くの問題例の最良値を更新することに成功した. 表 2に, 1999 年 9 月現在の状況を示す. 3 列目は, 我々が更新した最良値が, これまでに知られている下界値と一致した問題例の数を示す.

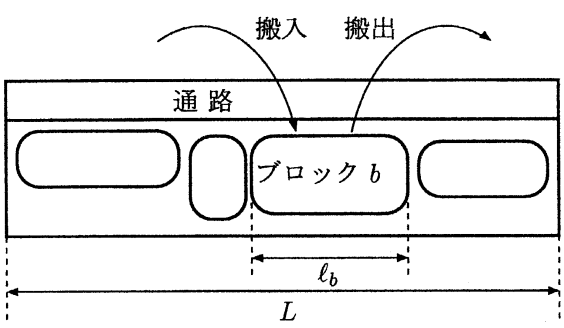


図 1: 工場概略図

表 2: 従来の最良値を更新した問題例の数.

| タイプ | 最良値を更新した問題例の数 | 最適値を求めた問題例の数 |
|---|---|---|
| j60.sm | 6 | 0 |
| j90.sm | 15 | 4 |
| j120.sm | 89 | 33 |
| j30.mm | 126 | 0 |

## 5 RCPSP ソルバの適用例

本章では, 我々が開発した RCPSP ソルバの現実問題への適用例について述べる. 例として, 大きな構造物を生産する工場におけるスケジューリング問題を考える. この問題では, 扱う構造物が大きいため, 一旦それらを工場内に配置すると, 作業が終了するまで移動することができない. そのため, 構造物の配置を含めてスケジューリングをする必要がある. さらに, 限られた数の工員で, 納期までに作業が完了するよう, 配員計画を行わなくてはならない. この問題は容易には直接 RCPSP として定式化することができない. そこで, 構造物の搬入日の決定・構造物の配置の決定・配員計画の 3 段階に分け, それぞれを RCPSP として解くというアプローチをとる.

一般に, 問題が与えられたとき, その都度個々の問題に対する専用アルゴリズムを開発することは手間がかかり, 現実的には容易なことではない. このことから, 既存の汎用アルゴリズムを用いて実用的な計算時間で良質の解を求めることができれば, 実用上有用な手段であると言える. 本アプローチはこの考えに基づくものであり, その一例を示すものである.

### 5.1 問題定義

$\{1, 2, \ldots, B\}$ を構造物 (以下, ブロックと呼ぶ) の集合とする. 各ブロックはクレーンで工場内に搬入され, 取り付け・溶接等の作業がなされた後, 搬出される. これらのブロックは長さ $L$ の工場内に一列に配置される. 各ブロック $b$ には処理日数 $p_b$ が与えられており, その処理期間中, 長さ $\ell_b$ のスペースを占有し, 一旦配置されると, 搬出されるまで移動する

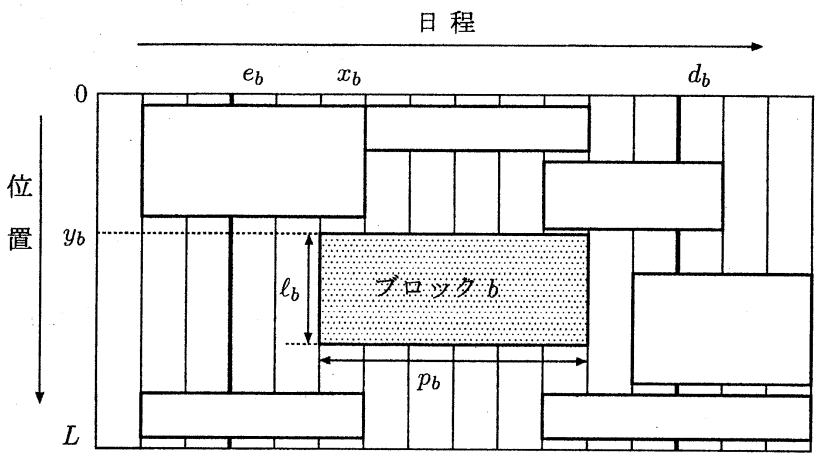


図 2: 矩形の詰め込み

ことはできない (図 1参照). また, 各ブロック $b$ には最早搬入日 $e_b$ および納期 $d_b$ が与えられており, その期間内に処理しなくてはならない. さらに, クレーンの運搬能力により, 1 日に高々 $M$ ブロックしか工場内に搬入することができない (すなわち, 各日において処理が開始されるブロックは高々$M$個). これらの制約を満たすように, 各ブロックの搬入日 (作業開始日) $x_b$ および位置 $y_b$ $(0 \le y_b \le L)$ を決定することが第一の目的である. これは図 2のように, 各ブロック $b$ に対応する (ブロック長 $\ell_b$) × (処理日数 $p_b$) の矩形を, (工場の長さ $L$) × (計画期間) の長方形内に, 互いに重ならないよう配置する問題と見なすことができる.

各ブロックに対してなされる作業は, 部品の取り付けと溶接に分けられ, 取り付けは溶接に先行する. すなわち, 溶接が行われる前日までに, 取り付け作業はすべて完了しておく必要がある. 以下では便宜上, 取り付け作業, 溶接作業をそれぞれ job1, job2 と呼ぶ. ここで, 各ブロック $b$ に対して, 搬入後 $u$ 日目 $(u = 1, 2, \ldots, p_b)$ に job $i$ を行う工員数を $w^i_{b,u}$ 人としたとき $(i \in \{1,2\})$, $(w^1_{1,1}, w^2_{1,1}, \ldots, w^1_{B,p_B}, w^2_{B,p_B})$ を配員計画と呼ぶ. ただし, ブロック $b$ に対する job2 の開始日を $u'_b$ とすると,

$$w^2_{b,1} = w^2_{b,2} = \cdots = w^2_{b,u'_b-1} = 0$$
$$w^1_{b,u'_b} = w^1_{b,u'_b+1} = \cdots = w^1_{b,p_b} = 0$$

でなくてはならない. また, ブロック $b$ に対して行う job1, job2 の仕事量は予め与えられており, それぞれを $W^1_b, W^2_b$ 人日とすれば,

$$\sum_u w^i_{b,u} = W^i_b, \quad i \in \{1,2\},$$

である. さらに, ブロック $b$ において, 同時に job $i$ を行うことのできる人数は $N^i_b$ 人であり,

$$w^i_{b,u} \le N^i_b, \quad i \in \{1,2\},\ u = 1, 2, \ldots, p_b$$

でなくてはならない. 配員計画 $(w^1_{1,1}, w^2_{1,1}, \ldots, w^1_{B,p_B}, w^2_{B,p_B})$ がこれらの制約を満たすとき, 実行可能であると言う. このとき, 1 日に job $i$ を行う工員総数の最大値

$$w^i_{max} = \max_t \sum_{\substack{b,u \ s.t. \\ x_b+u-1=t}} w^i_{b,u}$$

$(i \in \{1,2\})$ をできるだけ小さくするような実行可能配員計画を求めることが第二の目的である.

## 5.2 RCPSP アプローチ

本アプローチでは, ブロックの搬入日 (作業開始日) および配置の決定 (図 2における矩形の詰め込み) と配員計画を同時には扱わず, まず前者を解き, その後で後者を解く. これは, (1) 両者を分けて考えることで, 個々の問題が易しくなる, (2) 現実には, ブロック搬入の計画を予め長期間分作成しておく必要があるのに対して, 配員計画は比較的短期に分けて行わ

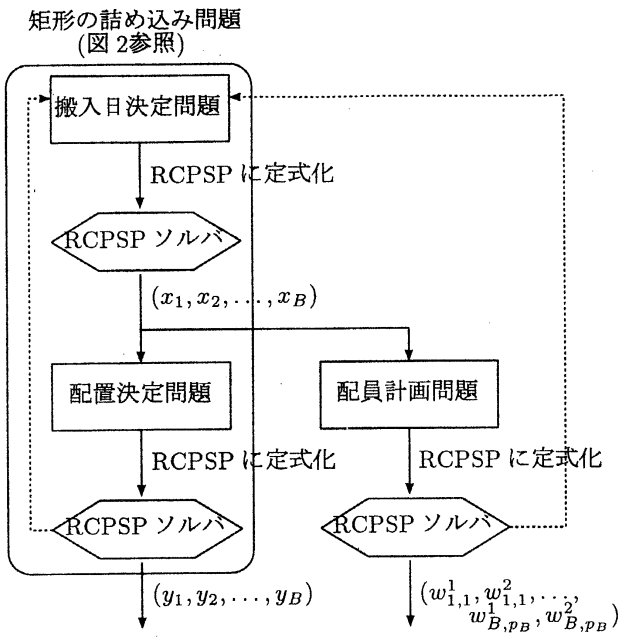


図 3: RCPSP アプローチの流れ

れるからである. さらに前者については, 搬入日の決定と配置の決定の 2 段階に分けて解く. この流れを図 3に示す. 搬入日決定問題・配置決定問題・配員計画問題は, それぞれ RCPSP に定式化され, RCPSP ソルバを 3 回用いることによって解かれる. ただし, 配置決定問題, および配員計画問題は, 搬入日決定問題を解いて得られたスケジュール $(x_1, x_2, \ldots, x_B)$ の下で, それぞれ, ブロックの配置 $(y_1, y_2, \ldots, y_B)$, および配員計画 $(w^1_{1,1}, w^2_{1,1}, \ldots, w^1_{B,p_B}, w^2_{B,p_B})$ を求める問題であり, 良質の解が得られない場合, 必要に応じて搬入日決定問題に戻り, 異なるスケジュール $(x'_1, x'_2, \ldots, x'_B)$ の生成を試みることになる.

以下, それぞれの問題をどのように RCPSP として定式化するかについて述べる.

### 5.2.1 搬入日決定問題

各ブロック $b$ を処理時間 $p_b$ の作業とみなし, 以下の制約をすべて満たすような, 各作業 (ブロック) $b$ の開始時刻 (搬入日) $x_b$ を求めることが本段階での目的である.

1. (クレーン制約)

   各日 $t$ の利用可能量が $K_{r^c,t} = M$ である資源 $r^c$ を導入し, その必要量を

   $$k_{b,r^c,u} = \begin{cases} 1, & u = 1, \\ 0, & u = 2, 3, \ldots, p_b, \end{cases} \quad \text{for all } b,$$

   とする.

2. (長さ制約)

   各日 $t$ の利用可能量が $K_{r^\ell,t} = \rho L$ ($\rho$ は $0 \leq \rho \leq 1$ を満たす定数) である資源 $r^\ell$ を導入し, その必要量を $k_{b,r^l,u} = \ell_b$ $(b = 1, 2, \ldots, B, u = 1, 2, \ldots, p_b)$ とする.

3. (最早搬入日・納期制約)

   $$e_b \leq x_b \leq d_b - p_b, \quad \text{for all } b.$$

ここで, 2 の長さ制約において乗数 $\rho$ が導入されているのは, 長さに対する資源制約を必要条件より強めて

$$\sum_{b \in J_{r^l,t}} \ell_b \leq \rho L, \quad t = 1, 2, \ldots$$

とするためである. すなわち, $\rho < 1$ を用いることで次段階において実行可能な配置を求め易くしているのである. ただし, $\rho < 1$ を用いると, 実行可能な配置が存在するスケジュール $(x_1, x_2, \ldots, x_B)$ を排除してしまう可能性がある.

### 5.2.2 配置決定問題

前段階と同様, 各ブロック $b$ を作業とみなす. ただし本段階では, ブロックの座標軸を RCPSP の時間軸に対応させる. すなわち, 各作業 $b$ の作業時間はブロック長 $\ell_b$ であり, 開始時刻 $y_b$ が搬入位置となる (図 4参照).

まず, 同日に処理されるブロックが互いに重ならないようにするため, 各日 $x$ に対して, 利用可能量が $K_{r^x,t} = 1$ $(t = 1, 2, \ldots)$ である資源 $r^x$ を導入し, $x_b \leq x \leq x_b + p_b - 1$ を満たす各ブロック $b$ に対して, $k_{b,r^x,u} = 1$ $(1 \leq u \leq \ell_b)$ とする. 次に, 時間制約

$$s_{sink} - s_{source} \leq L$$

を導入する. ここで source (sink) は, すべての作業に先行 (後続) する処理時間 0 の仮想的な作業であり, source の開始時刻 $s_{source}$ は便宜上 0 であるとする.

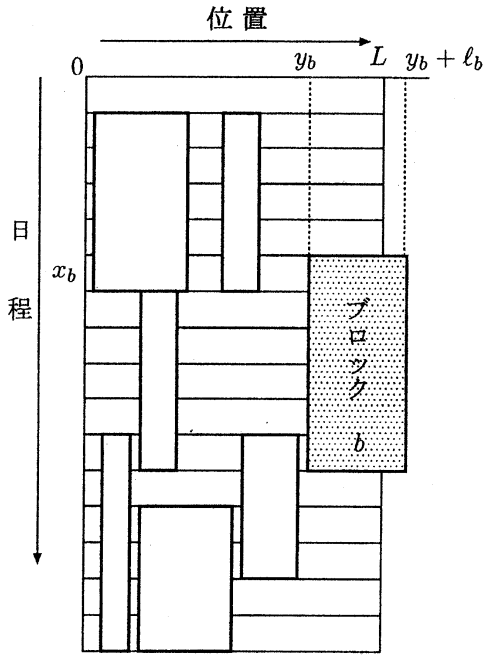


図 4: 配置決定問題

これにより, 最大完了時刻 $\max_b y_b + \ell_b$ は $L$ 以下に制約され, 実行可能スケジュール $(y_1, y_2, \ldots, y_B)$ は, ブロックが互いに重ならない配置となる.

なお, RCPSP の定式化において時刻 $t$ は整数値をとるため, ブロックの座標軸を適当な長さで離散化しておく必要がある.

### 5.2.3 配員計画問題

本段階では, 各ブロック $b$ に対して行われる仕事量 $W_b^i$ 人日の job $i$ ($i \in \{1,2\}$) を, $W_b^i$ 個の単位作業に分割し, それぞれを RCPSP における作業とみなす. すなわち, すべての作業 $j$ に対して処理時間 $p_j = 1$ 日である. ブロック $b$, job $i$ に対応する単位作業の集合を $J_b^i$ とする. このとき, 配員計画は

$$w_{b,u}^i = |\{j \in J_b^i \mid s_j - x_b + 1 = u\}|$$

($i \in \{1,2\}$, $b = 1,2,\ldots,B$, $u = 1,2,\ldots,p_b$) で与えられる. ただし,

$$x_b \le s_j \le x_b + p_b - 1, \quad j \in J_b^1 \cup J_b^2,$$

$$s_i + 1 \le s_j, \quad (i,j) \in J_b^1 \times J_b^2,$$

でなくてはならず, これらの制約は時間制約として記述される. また, ブロック $b$ において同時に job $i$ を行う人数を $N_i^b$ 人以下に抑えるため, 資源 $r_b^i$ を導入し, その利用可能量と必要量をそれぞれ $K_{r_b^i,t} = N_b^i$ ($t = 1,2,\ldots$), $k_{j,r_b^i,1} = 1$ ($j \in J_b^i$) とする. これらの制約の下で, 1 日に job $i$ を行う工員総数の最大値 $w_{max}^i$ を最小化することが目的となる. そのために, 本アプローチでは目的関数固定法を用いる. すなわち, まず十分大きな $K^i$ に対し $w_{max}^i \le K^i$ ($i \in \{1,2\}$) となる配員計画の作成を試みる. これは, 利用可能量 $K_{r^i,t} = K^i$ ($t = 1,2,\ldots$), 必要量 $k_{j,r^i,1} = 1$ ($j \in J_b^i$, $b = 1,2,\ldots,B$) である資源 $r^i$ ($i \in \{1,2\}$) を導入することにより, 資源制約で記述できる. そして, $w_{max}^i \le K^i$ ($i \in \{1,2\}$) となる配員計画が得られる度に $K^i$ を減らしていくのである.

## 5.3 計算実験

実データに基づく問題例に対して計算実験を行った. この問題例は, ブロック数 $B = 58$, 工場の長さ $L = 119$ m であり, クレーンで工場内に搬入できる 1 日あたりのブロック数は $M = 2$ 個である. また, 処理日数 $p_b$, ブロック長 $\ell_b$ の平均はそれぞれ 9.64 日, 20.4 m であり, 計画期間は 158 日である.

以下では, 1 回の探索の計算時間を 3 分とし, 3 分以内に実行可能スケジュールが得られた場合, その探索は成功であると言う.

まず, $\rho L = 119, 116, \ldots, 104$ とした搬入日決定問題に対して, タブー探索を, 乱数系列を変えて 30 回ずつ行い, その後, 30 回の探索中成功した際に得られた各実行可能スケジュール $(x_1, x_2, \ldots, x_B)$ に対して, 配置 $(y_1, y_2, \ldots, y_B)$ を求める探索を 1 回ずつ行った. その計算結果を表 3に示す. 表 3には, 各 $\rho L$ に対して, 搬入日決定問題, および配置決定問題に対する探索が成功した回数と, 実行可能スケジュールが得られるまでの平均計算時間 (秒) が記されている. $\rho L$ が小さくなるにつれて, 搬入日の実行可能スケジュールを求めることが困難になっていく一方, 配置決定問題に対する成功率は増していく傾向があることが分かる. ただし, 実用上, $\rho L$ をどの値に設定すればよいか予め判断することは難しく, 予備的実験が必要となる.

この計算実験で得られたスケジュールの例を図 5 に示す (計画期間 158 日中, 40 日間分のみを図示).

表 3: 搬入日・配置決定問題に対する計算結果

| $\rho L$[m] | 搬入日の決定[1] | 配置の決定[1] |
|---|---|---|
| 119 | 30/30<br>0.07[秒] | 5/30<br>(66.1[秒]) |
| 116 | 30/30<br>0.11[秒] | 4/30<br>(71.2[秒]) |
| 113 | 30/30<br>0.59[秒] | 16/30<br>(61.9[秒]) |
| 110 | 30/30<br>2.87[秒] | 30/30<br>43.8[秒] |
| 107 | 30/30<br>22.4[秒] | 30/30<br>35.7[秒] |
| 104 | 0/30<br>— | —<br>— |

1. 上段は探索の成功割合. 下段は実行可能スケジュールが得られるまでの平均計算時間 (実行可能スケジュールを求めることに失敗した探索がある場合は, 成功した探索に対する平均).

なお, 現状では, この規模の問題例に対して日程計画者がスケジュールを作成するのに数日以上かかっている.

次に, 図 5 に示されたスケジュールに基づいて配員計画問題に対する計算実験を行った. 計画期間は図 5 に図示された 40 日間である (配員計画の対象となるブロックの数は 24 個). job1, job2 の総仕事量 $\sum_b W_b^1$, $\sum_b W_b^2$ は, それぞれ 393 人日であり (よって, RCPSP に定式化した際の作業数は 786), 各ブロック $b$ に対して, 同日に作業できる仕事量は $N_b^1 = N_b^2 = 8$ であるとする. この問題例に対して, $w_{max}^1, w_{max}^2$ の上限値 $K^1, K^2$ を変えながら, タブー探索を 30 回ずつ行った. その結果を表 4に示す. 計算実験で得られた最良解は $(K^1, K^2) = (11,11)$ であった. なお, 実用上, $(K^1, K^2) = (13,13), (12,12)$ に対してはそれぞれ 1 回の探索で十分である.

ここで扱った配員計画問題は, 変数 $w_{1,1}^1, w_{1,1}^2, \ldots, w_{B,p_B}^1, w_{B,p_B}^2$ に加えて, ブロック $b$ に対する job1 の作業が搬入後 $u$ 日目までにすべて完了していれば (いなければ) 1 (0) をとる 0-1 変数 $z_{b,u}$ $(b = 1,2,\ldots,B, u = 1,2,\ldots,p_b)$ を導入すれば, 容易に整数計画問題 (IP) として定式化できる. 商用 IP ソルバを用いて同じ問題例を解いたところ, 1 分以内で $(K^1, K^2) = (11,11)$ の解を求めることに成功した. しかし, RCPSP アプローチでは, 問題の規模が job1, job2 の総仕事量 $\sum_b W_b^1$, $\sum_b W_b^2$ に大きく依存するのに対して, IP アプローチでは問題の規模がブロックの処理期間 $p_b$ に依存するため, IP アプローチの方がどの問題例に対しても有効であるとは断言できない.

表 4: 図 5のスケジュールの下での配員計画問題に対する計算結果

| $(K^1, K^2)$ | 成功割合 | 平均計算時間[1] |
|---|---|---|
| (13,13) | 30/30 | 12.9 [秒] |
| (12,12) | 30/30 | 32.0 [秒] |
| (11,11) | 11/30 | (108.6 [秒]) |
| (10,11) | 0/30 | — |
| (11,10) | 0/30 | — |

1. 実行可能スケジュールが得られるまでの平均計算時間. 実行可能スケジュールを求めることに失敗した探索がある場合は成功した探索に対する平均.

## 6 まとめと考察

本論文では, 広範なスケジューリング問題を扱うことができるよう, RCPSP の定式化を拡張し, タブー探索に基づく RCPSP ソルバを開発した. その有用性を確かめるために, ベンチマーク問題を解いたところ, 多くの問題例に対して, これまでの最良値を更新することができた. さらに, 現実問題に対する適用例として, 大きな構造物の生産スケジューリング問題に対する RCPSP アプローチについて述べ, 計算実験を行った. 搬入日・ブロック配置決定問題に対しては, 従来, 日程計画者が数日以上かかって作成していたスケジュールよりも良質のものを短時間で求めることに成功した. この結果と, 計算のために準備する必要があるのは原問題を RCPSP に変換するフィルタのみであることを考えれば, 本アプローチは十分実用的であると言える. このように, 既存の汎用アルゴリズムを利用することにより, 実用上,

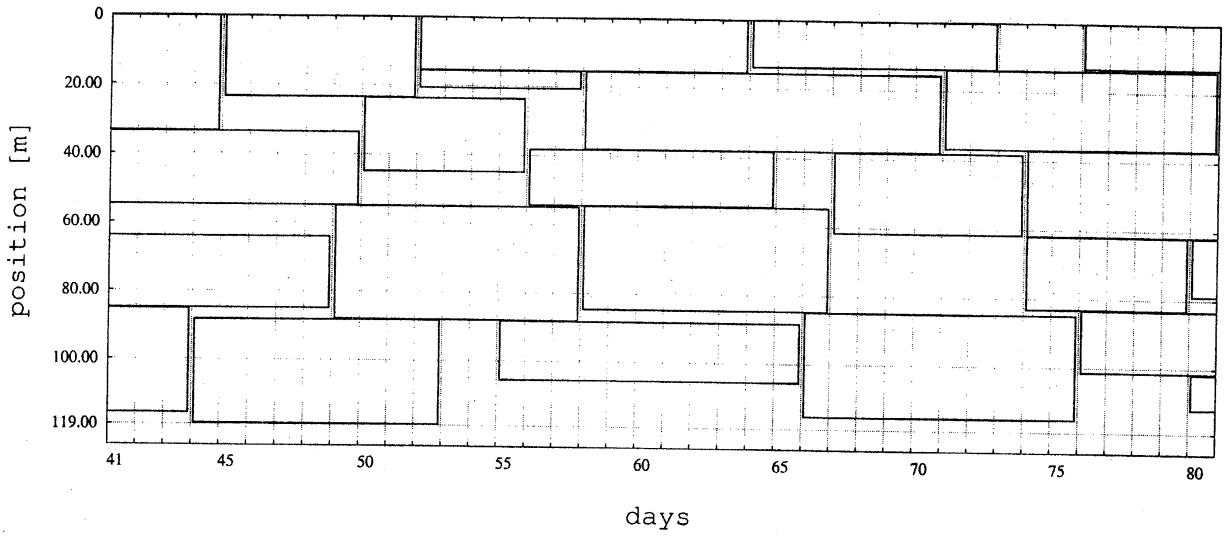


図 5: 計算実験で得られたスケジュールの例 (計画期間 158 日中, 40 日間分).

満足のいく結果が得られる他の問題も少なくないと思われる. 今後, より広範囲の問題を扱うことのできる汎用アルゴリズムを開発していくことは重要であり, 今後の課題の一つである.

また一方で, 与えられた問題を解くために, どのアルゴリズムをどのように活用すれば効率的であるか判断することは容易ではない. 5.3章で述べたように, どの定式化が望ましいか判断することは, 現段階では経験的知識や予備的実験に依ることが多く, その指針を与えることも重要であると思われる.

## 参考文献

[1] P. Brucker, A. Drexl, R. Möhring, K. Neumann and E. Pesch, "Resource-constrained project scheduling: Notation, classification, models, and methods", it European Journal of Operational Research 112 (1999) 3-41.

[2] T. Baar, P. Brucker and S. Knust, "Tabu search algorithms and lower bounds for the resource-constrained project scheduling problem", in: S. Voss, S. Martello, I. Osman and C. Roucairol (eds.): *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, (1998) 1-18.

[3] F. Glover, "Tabu search - Part I", *ORSA Journal on Computing* 1 (1989) 190-206.

[4] S.Hartmann, "A competitive genetic algorithm for resource-constrained project scheduling", *Naval Research Logistics* 45 (1998) 733-750.

[5] W. Herroelen, B. De Reyck and E. Demeulemeester, "Resource-constrained project scheduling: A survey of recent developments", *Computers and Operations Research* 25 (1998) 279-302.

[6] R. Kolisch and A. Sprecher, "PSPLIB – A project scheduling library", *European Journal of Operational Research* 96 (1997) 205-216.

[7] K. Nonobe and T. Ibaraki, "A tabu search approach to the CSP (Constraint Satisfaction Problem) as a general problem solver," *European Journal of Operational Research* 106 (1998) 599-623.

[8] K. Nonobe and T. Ibaraki, "Formulation and tabu search algorithm for the resource constrained project scheduling problem (RCPSP)", to be submitted (1999).