汎用スケジューラ
—RCPSPによるアプローチ—

野々部　宏司，茨木　俊秀

1. はじめに

スケジューリング問題は，応用において重要であるだけでなく，計算の複雑さの理論という観点からも興味深いことから非常に多くの研究がなされています．オペレーションズ・リサーチの分野でも，1950年代以降，多くのモデルが提案され，その計算の複雑さの解明や効率の良いアルゴリズムの開発がなされてきました [1, 6]．しかし一口にスケジューリングと言っても，生産スケジューリング，時間割作成，配送計画，要員計画など様々なものが考えられ，実際の応用では，個々の状況に応じた特殊な制約なども考慮しなくてはなりません．このため，我々が直面するスケジューリング問題を包括的に扱うことを目的とする研究はほとんどなく，個々のスケジューリング問題に対して，その問題専用のアルゴリズムを開発するという研究が大部分でした．

しかし，アルゴリズム開発には通常，多くの人手や時間が必要となりますので，いろいろなスケジューリング問題に手軽に適用できる，汎用スケジューラなるものがあれば非常に有用であることになります．このことから，広い範囲のスケジューリング問題を扱うことのできるソフトウェアも様々な分野において実用化されています．しかしそれらのパッケージは，問題の規模や種類，計算時間などの面では実用に耐え得ても，最適化という観点からは必ずしも満足できる性能を持つとは言えないようです．

我々は，現実のスケジューリング問題に適用できる，汎用性と最適化の機能を備えたスケジューラの実現を目指しています．もちろん厳密な最適解を求めるという意味での最適化ではなく，近似解法を実装したものです．本稿では，このような考えに基づいてこれまでに我々が開発したスケジューラを紹介します．

ののべ　こうじ　京都大学大学院情報学研究科
いばらき　としひで　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
e-mail: {nonobe,ibaraki}@amp.i.kyoto-u.ac.jp

2. 汎用モデル: RCPSP

世の中に現れるスケジューリング問題は多種多様であり，それらすべてを統一的に定式化することは困難です．しかし，単にスケジューリング問題と言えば，多くの場合，

> 与えられた複数の仕事を限られた台数の機械上で処理する際に，各仕事をいつ，どの機械で処理するかを決定する問題

を意味すると考えられます．もちろん，機械や仕事の特性，スケジュールの評価基準などに応じて様々なモデルが考えられます．ここで，「機械」をより一般的に，「人」・「原材料」・「予算」など仕事を処理するのに必要な「資源」であると解釈すれば，上の問題は，

> 限られた資源の下で与えられた仕事を処理する際に，資源配分および作業の開始時刻を決定する問題

と考えることができます．このような問題は，**資源制約付きスケジューリング問題** (resource constrained project scheduling problem, RCPSP) と呼ばれ，現実に現れるいろいろなスケジューリング問題を含む一般的な枠組みとして研究されています [7]．

我々のスケジューラでは，問題を記述する汎用モデルとして RCPSP を用いています．すなわち，ユーザが準備すべきことは解きたい問題を RCPSP とし

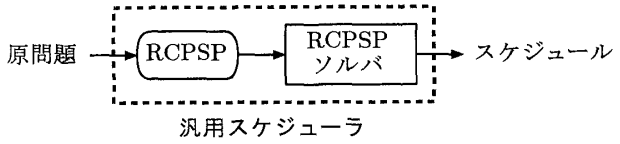

図 1: システムの概要

て記述することであり, その後はスケジューラの中の**RCPSP**ソルバがスケジュールを求めてくれるというわけです (図 1).

しかし, 従来の RCPSP モデルでは段取り替え作業が扱えないなど, 記述可能な問題がやや限られていました. そこで我々は, 従来の RCPSP を拡張し, システムの適用範囲を拡げました. 以下, 本システムの RCPSP モデルについて説明します.

2.1 資源・作業

RCPSP の基本要素は, 資源と作業です. さらに資源は,

(i) 機械や人, 作業場所など, 各単位時間 (例えば 1 日) あたりに使用できる量が予め与えられている資源,

(ii) 原材料や予算など, 1 日あたりではなく, 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資源,

の 2 種類に分けられます[1]. (i) のような資源は, 前日までの消費量に関係なく, 毎日ある定められた量だけ使用可能になることから**再生型資源**と呼びましょう. これに対し, (ii) のような資源を**非再生型資源**と呼びます.

次に, 各作業の処理時間や, 各作業がどの資源をどれだけ必要とするのかは予め分かっているものとします. さらに, 作業の処理は一旦開始されたら, 完了するまで中断することは許されないものとします. しかし, 例えば,

> 通常 1 人で 2 日かかる作業が, もう 1 人臨時に雇うことで 1 日でできる. ただし, そのときコストが発生する,

など, 処理時間や必要な資源量が一意でない場合も現実にはよくあります. そこで, 作業の処理方法を表すモードというものを考えます. すなわち, 各作業は有限個のモードを持ち, 各モードに対して処理時間, および必要資源量が定められているものとします. 先の例では, 作業は, 処理日数が 2 日であり「人」という再生型資源を 1 人必要とする「通常」モードと, 処理日数 1 日, 必要「人」資源量 2 人で, さらに「予算」という非再生型資源を必要とする「特急」モードの 2 つを持つことになります. よって, スケジューリングにあたっては, ユーザが与えた制約を満たすよう

> 「各作業をいつ, どのモードで処理するか」

を決定することが問題となります.

2.2 資源制約

では, スケジュールが満たすべき制約として, どのようなものが記述できるのでしょうか. 以下, 制約について見ていきます. なお, 以下では作業 j の開始時刻, 完了時刻をそれぞれ s_j, c_j で表すことにします. また, 各作業の処理時間, 開始時刻 および 完了時刻はすべて非負の整数であるという前提で議論します.

再生型資源制約

> 各単位時間において, 処理中である作業が必要とする資源 r の総量がその供給量を越えてはならない.

通常の機械における資源制約は, 同時に複数の作業を処理することができないわけですから, 供給量, 消費量ともに常に 1 である特殊なケースと考えられます (図 2 参照). 一般に,

$$c_i \le s_j \text{ または } c_j \le s_i$$

が成立しなければならないという離接的な制約は, このタイプの再生型資源制約で記述できます.

また, 本 RCPSP モデルでは, 各資源の供給量は時間に関し一定でなくてもよいものとします. これにより,

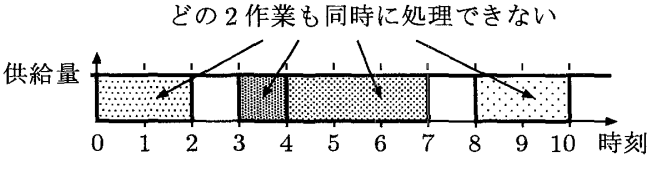

図 2: 供給量, 消費量ともに常に 1 である再生型資源.

[1] この他にも様々なタイプの資源が考えられますが, ここでは言及しません.

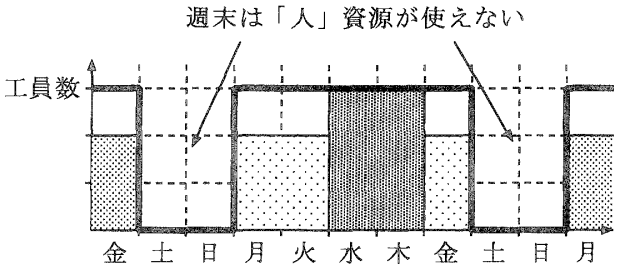

図 3: 供給量が一定でない再生型資源.

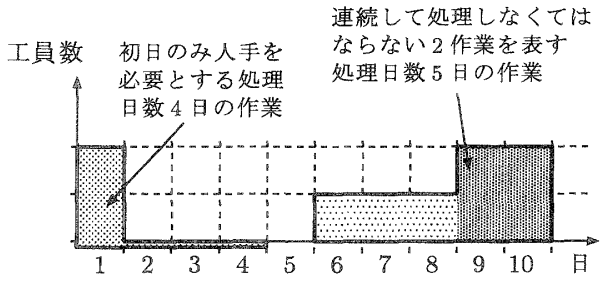

図 4: 資源消費量が一定でない作業.

- ある期間機械を停止しなくてはならない,
- 週末は平日よりも出勤する工員が少ない,
- 毎週決められた曜日にしか処理できない作業がある,

など, より現実的な状況を記述することが可能になります (図 3 参照). さらに, 各作業が必要とする資源量も, 処理中一定である必要はありません. したがって,

> 作業を開始するのに人手を必要とするが, 後は機械が勝手に処理してくれる,

といった状況にも対応できます. また, 間を開けずに連続して処理しなくてはならない 2 作業 i, j (すなわち, $c_i = s_j$) を, 1 つのまとまった作業として扱うことも可能です (図 4 参照).

非再生型資源制約

> 全作業による資源 r の総消費量が, 与えられた供給量を越えてはならない.

すなわち, 非再生型資源制約は, 各単位時間あたりではなく, スケジュール全体での消費量に関する制約であることに注意して下さい. また, 各作業が必要とする非再生型資源の量は, その開始時刻によらず, モードにのみ依存します. すなわち, 非再生型資源制約は作業のモードに関する制約であり, 複数のモードを持つ作業に対してのみ, 意味を持ちます.

2.3 先行制約

先行制約 $i \to j$

> 作業 i が完了するまで, 作業 j の処理を開始できない. すなわち, $c_i \leq s_j$.

本 RCPSP モデルではより一般的に, 任意の整数 (負数も可) $\delta_{i,j}$ を用いた制約

$$c_i + \delta_{i,j} \leq s_j$$

(時間制約, temporal constraint などと呼ばれる) を記述することができます[2]. ただし, 先行関係 $\to$ は半順序であるとします.

再生型資源 r 上の排他的先行制約 $i \Rightarrow_r j$

> 作業 i は j に先行し ($i \to j$), さらに, i, j がともに資源 r を消費するならば, i の処理完了後, 資源 r を消費する他の作業 k は, j よりも先に開始してはならない. すなわち, $s_k < c_i$ または $s_j \leq s_k$. (図 5 参照).

例えば資源 r が図 2 のような機械であれば, $i \Rightarrow_r j$ は, i の次に j を処理することを義務づける制約となり, 作業の切替えに伴う段取り替え作業を扱うことが可能になります (2.4 節で述べます).

ところで, この排他的先行制約は機械資源以外に対しても有用です. 紙面の都合上詳細は省きますが, 人為的な資源, および作業を導入することで,

- 作業 i と j は同時に開始しなくてはならない,
- 作業 i が完了後, 直ちに j の処理を始めなくてはならない[3],

などの制約を記述することもできます. また, モードと組み合わせることで,

> 作業 i は, 作業 j, k, ℓ のいずれかと同時に開始しなくてはならない,

[2]先行制約だけでは「$c_i = s_j$」という制約を記述することはできません.

[3]再生型資源制約のところで述べたように, i, j を 1 つの作業 k と見なしてしまっても構いません. しかし, i, j がともに複数のモードを持つ場合, k のモード数は多くなります. よってこのような場合, i, j を個々の独立した作業と見なす定式化の方が RCPSP ソルバの効率は高まります.

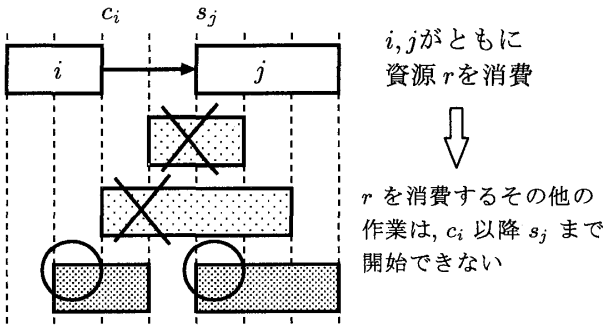

図 5: 再生型資源 r 上の排他的先行制約 $i \Rightarrow_r j$

といった制約も記述できます.

2.4 段取り替え作業

現実の問題では, ある機械 r 上で作業 i から 作業 j に移行する際, 段取り替え作業 $setup(j)$ が必要となることがあります. このとき段取り替え作業 $setup(j)$ は, 当然, 機械 r 上において作業 j の直前に処理されなくてはならず, j を開始するまでの間, 機械 r 上で他の作業を処理することはできません. このような制約は, 前節の排他的先行制約

$$setup(j) \Rightarrow_r j$$

を用いて記述できます[4].

また, $setup(j)$ の処理時間などは, 通常, その直前に機械 r 上で処理されていた作業 i によって異なります. 本スケジューラでは, $setup(j)$ のモードを (すなわち, 処理時間および資源消費量を), i に応じて定めることができます[5].

2.5 評価基準

次に, スケジュールの評価基準について考えましょう. 多くのRCPSPモデルでは, 資源制約および先行制約の下で, 最大完了時刻 (makespan) を最小化することを目的としています. しかし現実のスケジューリング問題においては, 滞留時間和 (total flow time) 最小や納期遅れ作業数最小, 納期ずれ時間和最小など, 状況に応じて様々な評価基準が考えられますので, 汎用モデルとしては評価基準を予め定めておくことは望ましくありません. そこで, 各作業 j のモードを表す 0-1 変数

$$x_{j,m} = \begin{cases} 1, & \text{作業 } j \text{ がモード } m \text{ で処理される,} \\ 0, & \text{その他,} \end{cases}$$

を導入し, $x_{j,m}$および開始時刻 s_j に関する線形不等式

$$\sum_j \sum_m \alpha_{j,m} x_{j,m} + \sum_j \beta_j s_j \leq \gamma \qquad (1)$$

を制約として記述することにします ($\alpha_{j,m}$, β_j および γ は係数). そして, 与えられた線形不等式制約の違反度 (ペナルティ) によりスケジュールの評価を行うのです. すなわち, このタイプの制約は, 必ず満たさなくてはならない制約 (絶対制約) ではなく, 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) として扱われます. そして, 再生型資源制約や先行制約などの絶対制約[6]を満たした上で, 考慮制約に対するペナルティを最小化することが目的となります. なお, 制約違反に対するペナルティとしては,

(i) $\max\{(\text{左辺}) - (\text{右辺}), 0\}$,

(ii) 制約が満たされているならば 0, 満たされていないならば 1,

のどちらかを選ぶことができます. また, 考慮制約は複数個あっても構いません. その場合, 各考慮制約に重要度を表す非負の重みが与えられ, それらの重み付きペナルティ和を最小化することになります.

ではここで, しばしば用いられる目的関数がどのように記述されるのか簡単に見てみましょう. まず, 最大完了時刻最小化についてです. sink を, 全ての作業に後続する処理時間 0 の仮想的な作業としますと, sink の完了時刻 c_{sink} を最小化することが目的となります. よって, 線形不等式制約

$$c_{sink} \leq 0$$

を与え, ペナルティを $\max\{c_{sink}, 0\}$ とすることで, 最大完了時刻最小化が記述できます. 同様に, 滞留時間和最小化は,

$$\sum_j c_j \leq 0$$

で記述できます[7]. また, 納期遅れ作業数最小化は, 作業 j の納期を d_j とすれば, それぞれの作業 j に対して制約

[4] 段取り替え作業 $setup(j)$ 完了後, 直ちに作業 j を開始するのであれば, $setup(j)$ と j をまとめて 1 つの作業と見なしても構いません.

[5] この結果, 原理的には, 巡回セールスマン問題 (traveling salesman problem, TSP) を扱うことも可能になりますが, 実際に TSP を効率良く解けるわけではありません.

[6] 非再生型資源制約は全て (1) の形で記述され, 考慮制約と見なされます.

[7] 作業 j の完了時刻 c_j は, モード m の処理時間を p_m とすれば, $s_j + \sum_m p_m x_{j,m}$ で表せます.

$$c_j \leq d_j$$

を与え, ペナルティを (ii) のタイプ (満たされていれば 0, 満たされていなければ 1), さらにすべての制約の重みを 1 とすることで記述可能です.

2.6 絶対制約と考慮制約

ところで, 資源制約や先行制約を絶対制約ではなく, 考慮制約として扱いたいこともあるでしょう. 逆に, 線形不等式制約 (1) を絶対制約として扱いたいこともあると思います. 本モデルでは, 再生型資源制約を考慮制約として扱うこともでき[8], 先行制約は, (1) の形で記述することで考慮制約となります (排他的先行制約は絶対制約としてのみ記述可能). ただし, 線形不等式制約 (1) として与えられるその他の制約 (非再生型資源制約を含む) を絶対制約とすることはできません. 絶対制約として扱いたい場合は, 十分大きな重みをかけることで対処することになります.

このように本モデルでは, どの制約を絶対制約として記述するか, 各考慮制約の重みをどのように設定するかなどはユーザが決定しなくてはなりません. しかし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わけです.

3. RCPSP ソルバ

次に, RCPSP を解くアルゴリズムについて考えましょう. まずはじめに, RCPSP は解くことが非常に困難であることを強調しておきます. ただし, ここで言いたいのは RCPSP が NP 困難だからということだけではありません. 近年, NP 困難な問題であっても, 理論的保証を要求しなければ, メタ戦略 (metaheuristics) などの近似解法を用いて良質の近似解を求めることが可能になってきています. RCPSP (あるいは, ジョブショップ問題を含め, スケジューリング問題全般) は, そのような近似解法を用いても, 実際に良い解を求めることが困難なのです. このような意味での問題の難易度を厳密に比較することは不可能でしょうが, 文献などでよく用いられるベンチマーク問題の規模が, 例えば集合被覆問題 (set covering problem) では変数, 制約の数がそれぞれ 1000, 10000 以上, TSP では都市数が数千から数万にもなるのに対して, RCPSP では作業数が数十から百程度であることからも分かると思います. つまり現段階では, 複雑な制約を伴う現実のスケジューリング問題に対して満足のできる最適化を行うことができるのは, 作業数がせいぜい数百程度の問題までででしょう.

3.1 最適化の仕組み

ジョブショップ問題やRCPSPなどのスケジューリング問題では, 従来, 短時間でスケジュールを生成することができるリストスケジューリング (list scheduling) がよく用いられてきました. すなわち, 作業をある順番に並べたリストを用意しておき, その順序にしたがって処理を行っていく方法です. このとき, どのように作業を並べてリストを作るかが問題となります. この目的に, 処理時間が最短の作業を優先するなど, 様々な優先規則 (priority rule あるいは dispatching rule) が研究されてきました. しかし, どのような問題に対しても良質のスケジュールを生成してくれる都合の良い優先規則はないと考えられますので,

> 標準的な優先規則を幾つか用意しておき, それらに基づいて生成されたスケジュールの中から最良のものを出力する,

というのも一つの方法でしょう. その際, 必要に応じてユーザが優先規則を追加・変更できるようにしておくことも必要かと思われます.

しかし, この方法では (計算時間は短くても) 最適化はあまり期待できません. 最適化を行うには, 膨大な数のリストの中から, 良いスケジュールを生成するリストを効率よく探索する必要があります. 我々のRCPSP ソルバは, リストの探索をタブー探索で行うアルゴリズムです. 本稿では詳しい説明は省かせて頂きますが, アルゴリズムの詳細について興味をお持ちの方は文献 [4] を御覧下さい.

3.2 ベンチマーク問題による性能評価

我々の RCPSP ソルバのような, 解の精度に関して理論的保証のない近似解法の性能評価は, 多くの場合, web 上などで一般公開されているベンチマーク問題を用いた計算実験によりなされます. PSPLIB (project scheduling problem library)[2] はそのようなベンチマーク問題集の 1 つであり, ベンチマーク問題が, これまでの最良値などとともにホームページ[9] から入手できます. 我々もこの問題集を解いてみました. 詳しくは述べませんが, j60.sm, j90.sm, j120.sm,

[8] 総消費量が供給量を越えたとき, その超過量がペナルティとなります.

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

j30.mm という 4 タイプ[10], 計 2110 個 (それぞれ, 480 個, 480 個, 600 個, 550 個) の問題例のうち, 計 1341 個 (それぞれ, 371 個, 369 個, 219 個, 382 個) の問題例に対して, 平均計算時間 約 30 秒 (j60.sm) ～10 分 (j120.sm) の間に最良解を求めることができました (ワークステーション Sun Ultra2 (300MHz) を使用). その中には, これまでの最良値を更新したものもあり, 1999 年 12 月現在, 計 235 個 (それぞれ, 5 個, 15 個, 89 個, 126 個) の問題例の最良解が, 本 RCPSP ソルバにより発見されたものです.

4. 適用例

では最後に, ベンチマーク問題以外への適用例を 3 つ紹介したいと思います.

1 つ目は, 77 個の作業を 19 台の機械で処理する問題です. この問題は, 各作業がどの機械で処理されるのか決まっているため, 各機械に対して作業の処理順序を決める問題となります. 作業にはそれぞれ納期が与えられていて, 処理完了日が納期より早くても遅くてもコストがかかります. さらに, 作業の移行には段取り替え作業が必要で, どの作業からどの作業に移行するかによって, そのコストが変化します. また, 一日に段取り替え作業を行うことのできる機械は 4 台までで, さらに, 週末には段取り替え作業を行うことができません. この制約を満たした上で, 総コストを最小化することが目的となります. この問題に対して, 数秒から数十秒程度で最適解 (最適性は商用 MIP (混合整数計画問題, mixed integer programming) ソルバで確認) を求めることができました.

次の例は, 文献 [3] で取り上げられた, 装置産業におけるスケジューリング問題です. この問題は, 11 種類の製品を 4 台の機械で生産する際の, 1ヶ月のスケジュールを求める問題です (スケジュールは 1 日単位で作成される). この問題には, 「各製品の生産日数は決められているが, 超過・不足してもよく, 数回に分けて生産してもよい」, 「一旦ある製品の生産を開始したら, 3 日間は中断できない」, 「ある機械上で生産する製品を変更するには段取り替え作業を要する」, 「製品は 7 グループに分けられ, 異なるグループ間の処理移行には 2 日間の工事を要する」, 「休日には, 段取り替え作業, および工事は行えない」など, 多くの制約があります. 目的は, 段取り替え作業の回数, および生産日数の不足を最小化することです. この問題も MIP の形に書けるのですが, 商用 MIP ソルバが 1 時間かかって求めた解よりも良いスケジュールを, 本スケジューラを用いることで, 5 分程度で求めることができました.

また文献 [5] では, 大きな構造物を対象とした生産スケジューリング問題を扱っています. この問題では, 扱う構造物が大きいため, 一旦作業を開始したら完了するまで動かすことができないという制約があります. つまり, 作業日だけでなく, 作業場所も考慮してスケジューリングしなくてはなりません. さらにクレーン制約や納期制約などもあり, 問題の規模はそれほど大きくありませんが (作業数 58), 全ての制約を満たすスケジュールを, 数日かかって人が作成しているのが現状です. この問題に対しても, 本スケジューラを使って[11] 実質計算時間 1 分程度でスケジュールを求めることができました.

5. おわりに

本稿では, 我々がこれまでに開発した汎用スケジューラを紹介しました. ここまでお読みになって, 「結構いろいろなところで使えそうだ」と思って頂けましたでしょうか? 今後, ソースコードを特定領域研究「アルゴリズム工学」のデータベース[12] 上で公開していきますので, ダウンロードして実際に使ってみて下さい.

ところで, 本アプローチでは RCPSP を汎用モデルとしていますが, 現実のスケジューリング問題においては, いろいろな要因が重なるため何が制約で何を最適化すべきなのか判断すること自体が難しいと思われます. また, 適切なモデル化を行うことこそが最重要であるとも考えられます. 本アプローチでは, この「モデル化」という問題はユーザ側に残されたままです. しかし, 手元に解きたい問題があるとき, それが RCPSP として記述できるのであれば, とりあえず我々のスケジューラを試すことができます. もしその答えに満足できないのであれば, モデル化をし直し, 別のアプローチを試みる必要がありますが, ユーザがプログラムを記述することなく「とりあえず試してみ

[10] 60, 90, 120, 30 は作業数を表し, "sm", "mm" はそれぞれ, single mode (各作業は唯一つのモードを持つ), multi mode (各作業は複数のモードを持つ) を表します. なお, いずれも最大完了時刻最小化が目的.

[11] 正確には, この問題を作業日決定問題と作業場所決定問題の 2 段階に分けて, それぞれを RCPSP に定式化します. すなわち, 本スケジューラを 2 回適用することになります.

[12] http://www-or.amp.i.kyoto-u.ac.jp/algo-eng/db/index.html

る」ことができるという手軽さは実用上有用であると考えています.

いま,「手軽」という表現を使ってしまいましたが, 実際に多くの方に「手軽」に使って頂くには, GUI などユーザ・インタフェースの充実が必要でしょう. ですが, 現段階ではタブー探索の過程をガントチャートとして視覚的に見ることはできますが[13], 入力はファイルから読み込むことになっています. しかし少しでもユーザの手間を減らすため, (当り前のことでしょうが) 入力ファイルとしては, 図 6 のように人が読んで意味が分かるようなフォーマットにしています.

我々は今後, バッファや在庫, ロット化など, より現実的な要素を扱うことのできる, より実用性の高い汎用スケジューラの開発を目指していきたいと考えています. そのためには, どのような制約を持った, どのくらいの規模のスケジューリング問題を解くことが現実には求められているのか知ることが重要となってきます. これに関して, 読者の方々よりご助言頂ければ幸いです.

参考文献

[1] P. Brucker, *Scheduling algorithms*, Springer Verlag (1998).

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

[3] 森戸 晋, 今泉 淳, 朴 宰完, "厳しい制約を有するスケジューリングに対する数理計画アプローチ", 生産スケジューリング・シンポジウム '96 講演論文集 85-90.

[4] K. Nonobe and T. Ibaraki, "Formulation and tabu search algorithm for the resource constrained project scheduling problem (RCPSP)", Technical Report #99010, Department Applied Mathematics and Physics, Kyoto University (1999).

[5] 野々部 宏司, 茨木 俊秀, "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", スケジューリング・シンポジウム '99 講演論文集 (1999) 53-58.

[6] M. Pinedo, *Scheduling: theory, algorithms, and systems*, Prentice Hall (1995).

[7] J. Weglarz (ed.), *Project scheduling: recent models, algorithms, and applications*, Kluwer Academic Publishers (1999).

[13] Unix 上の限られた環境にしか対応してません.

```
# Example: small instance (3 jobs, 2 machines)
PROBLEM example

# Resources
RESOURCE machine[1] =
   {amount:(1)*inf  weight:(inf)*inf}
RESOURCE machine[2] =
   {amount:(1)*inf  weight:(inf)*inf}

# Activities
ACTIVITY activity[1][1] =
   {mode:{time:5  resource:machine[1] (1)*5}}
ACTIVITY activity[1][2] =
   {mode:{time:8  resource:machine[2] (1)*8}}

ACTIVITY activity[2][1] =
   {mode:{time:3  resource:machine[2] (1)*3}}
ACTIVITY activity[2][2] =
   {mode:{time:7  resource:machine[1] (1)*7}}

ACTIVITY activity[3][1] =
   {mode:{time:2  resource:machine[2] (1)*2}}
ACTIVITY activity[3][2] =
   {mode:{time:5  resource:machine[1] (1)*5}}

# Precedence constraints
PRECEDENCE job[1] =
   {activity[1][1] -> activity[1][2]}
PRECEDENCE job[2] =
   {activity[2][1] -> activity[2][2]}
PRECEDENCE job[3] =
   {activity[3][1] -> activity[3][2]}

# Soft constraints
CONSTRAINT makespan =
   {weight:1
   expression:[completion_of sink] <= 0}
```

図 6: 3 仕事 2 機械のジョブショップ問題の入力ファイル. "#" から始まる行はコメント文. また, 例えば "(1)*5" は "1" が 5 個並んだ列 "1,1,1,1,1" を表す. 2 つの機械資源 machine[1], machine[2] は, 供給量が常に 1 ("(1)*inf") で, 絶対制約 (重みが無限大 "inf"). 各作業は, 唯一つのモードを持ち, 資源 machine[1] あるいは資源 machine[2] を, 処理期間中 1 ずつ消費する. 同じ仕事の作業間には先行制約があり, 最大完了時刻 (sink の完了時刻 c_{sink}) 最小化が目的.