人 工 知 能 72－1
（1990. 9. 12）

スケジューリング問題へのＡＴＭＳの適用

湯上 伸弘　　原 裕貴　　吉田 裕之

富士通研究所

本稿では、スケジューリング問題における探索の効率化の手法として、ＡＴＭＳの利用に関する検討を行う。仮説として、２個のジョブの前後関係をとり、各ジョブの開始時刻に関する不等式系の可解性をチェックすることによって環境の矛盾・無矛盾を調べるシステムを提案する。このシステムにより、連続値をとる時刻を扱うスケジューリング問題をＡＴＭＳの枠組みで解くことができる。このシステムの有効性について、例を用いて議論する。

Solving Scheduling Problems with ATMS

Nobuhiro Yugami　　Hirotaka Hara　　Hiroyuki Yoshida

FUJITSU LABORATORIES

1015 Kamikodanaka Nakahara-Ku Kawasaki 211, Japan

In this paper, we discuss a use of an ATMS for efficient searches in the scheduling problems. We propose a system in which an assumption is a pair-wise order of jobs and the consistency of an environment can be checked as a solvability of inequalities. This system can solve a scheduling problem ,which treats continuous variables "time" , within the framework of ATMS. We use a simple example to discuss an effectiveness of this system .

1　はじめに

我々は現在スケジューリング問題向けのエキスパート・シェルの研究をおこなっている。スケジューリング問題は組み合わせ（最適化）問題であるから、いかに効率的に探索を行うかが重要である。本稿では、探索の効率化の手法としてＡＴＭＳの利用の検討をおこなう。

まず、次の２、３、４節でスケジューリング問題とその特徴について検討し、なぜＡＴＭＳを使うのかについて述べる。５節では、スケジューリング問題を仮説推論で扱うために、仮説としてなにをとればよいかについて、６節では、仮説集合（環境）の無矛盾性を不等式系の可解性に対応させる方法[1]について述べる。７節では、ＡＴＭＳを利用する問題解決器について、８節では、ＡＴＭＳの利用による探索の高速化について議論する。最後に９節で、問題の拡張について述べる。

2　スケジューリング問題とその特徴

スケジューリング問題とは、処理されるべきジョブの集合が与えられたとき、各ジョブを処理する資源と、各資源におけるジョブの処理順序を、さまざまな制約を満足するように決定する問題である。スケジューリング結果である計画は、図１のような、縦軸として資源、横軸として時刻をとるGantt Chartで表わされる。Gantt Chartは、どの資源が、何時から何時まで、どのジョブを処理しているかを示している。時間は連続値を取り得る場合と離散値に制限される場合があるが、本稿では連続値をとるものとする。

ここで“資源”とは、計画の対象であって、ジョブをある有限の時間をかけて処理する。ただし一般には、あるジョブを処理できる資源は複数存在し、資源によってジョブの処理時間が異なる。

スケジューリング問題が、難しい理由には次の２点がある。第１に、スケジューリング問題は組み合わせ（最適化）問題であるから、探索空間が非常に大きく、その結果解を実用時間内に求めることが難しいという点である。第２に、制約や目的関数が複雑だったり、不明確である場合が多いということである。

スケジューリング問題は、ＯＲ（整数計画）などで解かれることが多かったが、定式化が難しく、また前述の２番目の点をうまく扱うことができない。さらに、条件を変えて（ジョブ、制約の追加・削除など）解く場合などに関する柔軟性が乏しいなどといった理由から、近年、エキスパートシステムを使った問題解決が盛んに行われている。ただし、エキスパートシステムを使っても、第１の特徴である探索空間の大きさからくる処理スピードの低下は依然として問題であり、効率的な探索をおこなう必要がある。

3　探索の効率化

従来のスケジューリング型エキスパートシステムでは、この問題をヒューリスティックスを使うことによって回避してきた。しかし、実用規模の問題を実用時間で解くためには、バックトラックをほとんど不要とするよな強力なヒューリスティックスを用いる必要がある。逆に、このような強力なヒューリスティックスを見つけることがシス

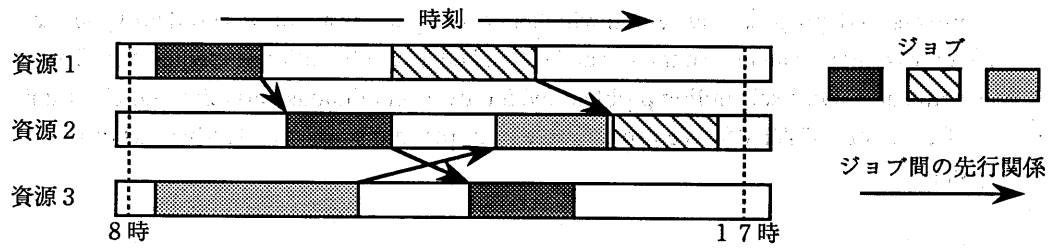

図１　Gantt Chart の例

テム開発において非常に重要なポイントであった。ところが、このようなヒューリスティックスは、問題に大きく依存し、汎用性があまりないため、ジョブ数、制約などの変化、また、資源の追加などによる問題の変化にうまく対応できない場合がある。また、専門家へのインタビュー等により発見することも難しい。そのため、現実的に獲得しやすいヒューリスティックス、すなわち、それほど強力ではないけれども、より汎用性のあるヒューリスティックスを使って問題解決を行うためには、探索自体の高速化をおこなう必要がある。

代表的な探索の効率化の手法としては、ＴＭＳ（Truth Maintenance System）[2]やＡＴＭＳ（Assumption based TMS）[3]などがある。解を一つだけ求めればよい問題にはＴＭＳが効果的で、すべての解を求めたり、最適解を求めたりする場合にはＡＴＭＳが向いていると言われている。また、単解探索におけるＡＴＭＳの利用としてＡＤＤＢ（ATMS-based DDB）[4]をはじめとする巡回型ＡＴＭＳが提案されている。すなわち、ある程度時間をかけて最適性を求める場合にはＡＴＭＳが、また制約充足解を１個求めればよいならばＴＭＳが有効であると思われる。本稿では、ＡＴＭＳの利用について述べるが、以下の議論する枠組みで、ほとんど変更なしにＴＭＳを利用することができる。

4　問題の定義

前節で述べたように、スケジューリング問題は、ジョブの資源への割り当てと、各資源に関するジョブの処理順序の決定の２つフェーズからなるが、以下では議論を簡単にするために、スケジューリング問題の特徴である時間を扱う部分である処理順序の決定の部分のみを考える。すなわち、

全てのジョブについて、処理する資源が既定である

とする。なお、ジョブの資源への割り当て問題については９節で述べる。

この結果、ジョブの処理時間は定数になる。また、制約としては、以下の３種類を考える。ここで、T_iはジョブｉの開始時刻である。

(a) ジョブの最早開始時刻
$$Const. \leq T_i$$
(b) ジョブの最遅開始時刻
$$T_i \leq Const.$$
(c) ジョブ間の先行関係
$$T_i + Const. \leq T_j$$

通常(b)の制約はジョブの締切時刻、すなわちジョブの終了時刻の上限値で与えられる場合が多いが、ジョブの処理時間が既定であるため、開始時刻の上限値に書き直すことができる。また、(c)の制約は、２個のジョブの相対的な関係であり、一方のジョブの終了時刻と他方の開始時刻の関係で与えられる場合が多いが、(b)と同様の理由で、ここでは開始時刻間の関係として表わすことができる。

5　スケジューリング問題における仮説

スケジューリング問題において、仮説として何をとればよいだろうか。スケジューリング問題の解であるジョブの処理順序が仮説集合となるように仮説をとるのが望ましい。このような仮説には、例えば以下のようなとりかたがある。

1 “ジョブｉはｎ番目に処理される”
2 “ジョブｉはジョブｊの直前に処理される”
3 “ジョブｉはジョブｊよりも前に処理される”

上のどの仮説も、その集合としてジョブの処理順序を表現できるが、本稿では３の仮説を採用し、簡単のため

“ジョブｉはジョブｊよりも前に処理される”

を

$$i < j$$

と表記することにする。3の仮説を使う理由は、次節で述べるように、環境の矛盾・無矛盾を簡単に調べることができるからである。

ここで、資源aで処理されるジョブの数をN_aとすると、各資源につき、それぞれ${}_{N_a}P_2$個の仮説があるから、全体の仮説の数は、

$$\sum_a N_a (N_a - 1)$$

個である。ただし、これらの仮説がすべて独立ではないことに注意しなければならない。例えば

$$i < j \quad \wedge \quad j < k$$

であれば、明かに

$$i < k$$

である。

また、

$$i < j \quad \vee \quad j < i$$

がかならず真であるから、nogoodとなる環境を管理するときに以下の様にリゾリューションを使うことにより、新たなnogoodを導くことができる。

$i < j \quad \wedge \quad A$ is nogood
$j < i \quad \wedge \quad B$ is nogood
↓
$A \quad \wedge \quad B$ is nogood

ここで、A，Bは任意の仮説集合である。

6 環境の矛盾・無矛盾

推論を進めるうえで、ある環境（仮説の連言）が無矛盾かどうかを調べる必要がある。本稿では、仮説として2個のジョブの前後関係をつかっているので、環境を以下のように、等価な1次不等式系に変換し、その不等式系が解を持つかどうかをチェックすることにより、環境が無矛盾であるかどうかを調べる。

環境に含まれる仮説“$i < j$”に対して、つぎのように同値な不等式を与えることができる。

$$T_i + L_i \leqq T_j$$

ここで、T_i，T_jはジョブi、ジョブjの開始時刻、L_iはジョブiの処理時間である。この不等式は、ジョブiの終了時刻（左辺）がジョブjの開始時刻よりも小さいことを示しているから、仮説“$i < j$”と同値である。

環境に含まれる全ての仮説に対して、上のようにして、同値な不等式をもとめ、その集合に、制約として与えられた不等式を加えた不等式系を求める。この不等式系が解をもてば、対応する環境は無矛盾であり、解をもたないなら、対応する環境は矛盾していることがわかる。一般に不等式系における解の存在性を調べるためには、各変数の下限値と上限値を求めることにより調べることができる。一般の1次不等式系に関して、そのためのアルゴリズムがSup-Inf法[5]として知られている。しかし、本稿で扱う不等式系に含まれる不等式は

$$T_i \leqq \text{Const.}$$
$$\text{Const.} \leqq T_i$$
$$T_i + \text{Const.} \leqq T_j$$

の3種類しかないから、より簡単な方法で下限値、上限値を計算することができる。この計算は、変数の数の3乗のオーダーの計算時間で計算できることが知られている[6]。　この下限値、上限値

の計算（による環境の無矛盾のチェック）は、推論中頻繁に行われるから、できうる限り高速に行う必要がある。この観点からすると、変数の数の３乗というのは遅すぎる。ここでは、特に、全ての変数の下限値、上限値が既知である不等式系に、新たに不等式を加えたときの下限値、上限値の変化を計算する場合が多いので、その場合の計算法を考察する。

ある不等式系C（変数T_iの下限値I_i，上限値S_i）に不等式$T_i+A_{ij}\leqq T_j$を加えることを考える。このとき、下限値の計算は以下の述語をlower_bound([(Tj,Ii+Aij)],[Ti+Aij≦Tj|C],NewC)で呼び出すことにより計算することができる。ここで、NewCは新しく計算された下限値を含む不等式系である。

```
lower_bound([],C,C):-
    !.
lower_bound([(Ti,Vi)|Values],C,NewC):-
    member(Ii≦Ti,C),
    Vi≦Ii,
    !,
    lower_bound(Values,C,NewC).
lower_bound([(Ti,Vi)|Values],C,NewC):-
    member(Ii≦Ti,C,RestC),
    Vi>Ii,
    setof((Tj,Vj),new_boundary(Ti,Vi,Tj,Vj,C),NB),
    append(NB,Values,NewValues),
    !,
    lower_bound(NewValues,[Vi≦Ti|RestC],NewC).

new_boundary(Ti,Vi,Tj,Vj,C):-
    member(Ti+Aij≦Tj,C),
    Vj is Vi+Aij.
```

上限値も全く同様にして計算できる。すなわち、計算は、基本的には次の２種類の計算を繰り返しおこなうことで実現される。

下限値の計算：

$$I_i\leqq T_i,\ T_i+L_i\leqq T_j \Rightarrow I_i+L_i\leqq T_j$$

上限値の計算：

$$T_i+L_i\leqq T_j,\ T_j\leqq S_j \Rightarrow T_i\leqq S_j-L_i$$

また、不等式系が解をもたないこと、すなわち環境の矛盾は

$$I_i\leqq T_i,\ T_i\leqq S_i,\ S_i<I_i \Rightarrow NG$$

から導かれる。

この計算法では、加える不等式の影響の伝播を計算しているので、通常は局所的な計算で終了するため、比較的高速に下限値・上限値を計算することができる。ただし、最悪の場合は非常に非効率的になる。計算量が最悪となるのは、例えば次の場合である。

$$T_i+A_{ij}\leqq T_j$$
$$T_j+A_{jk}\leqq T_k$$
$$T_k+A_{ki}\leqq T_i$$

ただし、定数、A_{ij}，A_{jk}，A_{ki}は以下の条件を満たす。

$$0<A_{ij}+A_{jk}+A_{ki}=\varepsilon<<1$$

この例では、明かに解が存在しないが、そのことを発見するのにO［（S_i-I_i）／ε］の述語呼び出しが必要である。すなわち、εが小さいとき、計算量は非常に大きくなる。ただし、現実的にはεが非常に小さいことはあまりない。また、不等式系に解が存在するときは上のような計算量の増大はおきない（上の例で解がある場合、すなわち$\varepsilon\leqq 0$のときは３回の計算で終了する）から、再帰呼び出しの回数がある一定数をこえる場合には“解なし”とすることにより実用時間で、ほぼ正確に解の存在性を判定することができる。

7　問題解決器

前節までの考察結果をつかうと、図２のようなスケジューリング問題解決システムを考えることができる。システムは、実際に推論をおこなう問題解決器（ＰＳ）とＡＴＭＳとからなる。ＰＳは次に調べる環境を選択し、その環境のもとで、各ジョブの開始時刻の下限値、上限値を計算し、その経過・結果をＡＴＭＳに通知する。

もっとも単純なＰＳは、以下の述語Schedulerをつかって実現できる。

```
scheduler([],E,E).
scheduler([(i,j)¦Pairs],E,Answer):-
    calculation([i<j],E),
    scheduler(Pairs,[i<j¦E],Answer).
scheduler([(i,j)¦Pairs],E,Answer):-
    consistent(E),
    calculation([j<i],E),
    scheduler(Pairs,[j<i¦E],Answer).
```

ここで、第１引数Pairsは、順序関係を決めなくてはいけないジョブの組で、初期値は同一資源で処理されるジョブの組全てである。ここではリストの第一要素から順序を決めていくが、適当な評価関数を用いて次に処理するペアを選択することによりヒューリスティックを組み込むことができる。第２引数Eは、現在着目している環境で初期値は［］である。また、述語consistent(E)は、環境Eが既知のＮＧを含むかどうかを調べる述語で、含まないとき真となる。述語calculation(A,E)は、環境Eに対応する不等式系に仮説集合Ａと同値な不等式の集合を加えたときに各変数の上限値・下限値を計算する述語で、新しい不等式系が解を持つとき真となる。この計算の経過・結果は、下限値＞上限値からの矛盾の導出を含めて、ＡＴＭＳに通知される。すなわち、矛盾が検出されたとき、ＡＴＭＳは、その下限値と上限値をデータとしてもつノードのラベルから矛盾する環境を求めるので、ＮＧとして得られるのは、ＰＳが現在着目しいる環境（[i<j¦E] or [j<i¦E]）ではない。

このschedulerでは、同一資源で処理される２個のジョブの前後関係を１つ１つ決めていくことにより計画を進めている。すなわち、ＰＳで用いられる仮説とＡＴＭＳで用いられる仮説が一致している。しかし、一般にはこの２つは一致している必要はない。例えば、以下のようにすれば前詰め法（ある資源で最初に処理するジョブから順番に選択していく方法）を実現できる。

```
scheduler2([],E,E):-
    !.
scheduler2(Jobs,E,Answer):-
    member(i,Jobs,Rest),
    consistent(E),
    A=[∀(i<j), s.t. member(j,Rest) ],
    calculation(A,E),
    append(A,E,NewE),
    scheduler2(Rest,NewE,Answer).
```

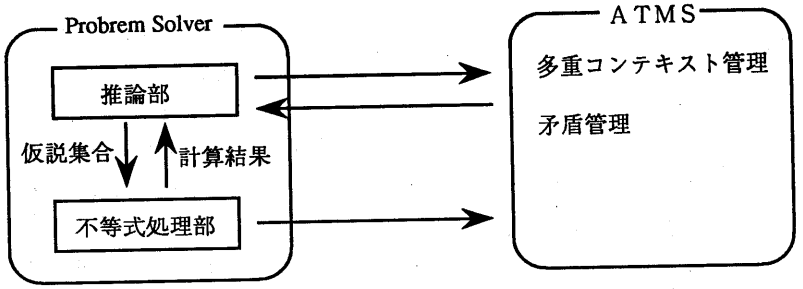

図２　ＡＴＭＳを備えた問題解決機構

ここで、scheduler2の第１引数は“未配置”のジョブの集合であり、初期値は全てのジョブの集合である。第２のEはschedulerと同様である。上で述べたように前詰め法は、初めに処理するジョブ、２番目に処理するジョブ、…という順番で選択していく方法であるから、選択されたジョブは、選択されていないジョブよりも（時間的に）前に処理されることを仮定することになる。すなわち、前詰め法では、ＰＳでつかう仮説は、ＡＴＭＳの仮説の連言になる。上記の述語で、member(i,Jobs,Rest)のかわりに、適当な評価関数をもちいて次に配置するジョブを選択することによりヒューリスティックが実現される。なお、このscheduler2は資源が１個の場合の述語であるが、資源が複数の場合への一般化は容易である。

ここであげた２つのＰＳの他にも、ほとんど同様にして、同一資源で処理されるジョブ間の隣接関係を決めていく探索や、ジョブ集合の部分集合の全順序に他のジョブを割り込ませていくような探索[7]を実行することができる。また、ここにあげた例は基本的にＡＤＤＢをおこなうＰＳであるが、より一般的な巡回型のＰＳを使うこともできる。

8　例

この節では、簡単な例題を用いることにより、ＡＴＭＳを備えた問題解決機構の動きを説明する。例題としては、以下の制約をもつ３個のジョブを１個の資源で処理するとき、その処理順序を求めることとする。

	処理時間	開始時刻の条件
ジョブ１	3	$2 \leqq T_1 \leqq 9$
ジョブ２	4	$0 \leqq T_2 \leqq 6$
ジョブ３	6	$1 \leqq T_3 \leqq 8$

ＰＳとして、前節で与えたschedulerを用いる。このとき、探索は以下のようにして進む。図３はこのときの探索木、図４は探索途中で発見されるＮＧである。

```
1 scheduler([(1,2),(1,3),(2,3)],[])
      calculation([1<2],[])
2     scheduler([(1,3),(2,3)],[1<2])
        calculation([1<3],[1<2])
3       scheduler([(2,3)],[1<2,1<3])
          calculation([2<3],[1<2,1<3])
                →fail   [1<2,2<3] is nogood
          consistent([1<2,1<3])
          calculation([3<2],[1<2,1<3])
                →fail   [3<2] is nogood
        consistent([1<2])
            →fail
      consistent([]),
      calculation([2<1],[])
2'    scheduler([(1,3),(2,3)],[2<1])
```

ここで、consistent([1<2])が失敗するのは、その前に発見された２個の矛盾する環境からＡＴＭＳが、次のようにリゾリューションをおこなうからである。

```
[1<2,2<3] is nogood
[3<2]     is nogood
2<3 ∨ 3<2
      ↓
[1<2]     is nogood
```

この結果

```
3'      scheduler([(2,3)],[1<2,3<1])
```

は呼び出されず、探索は

```
2'    scheduler([(1,3),(2,3)],[2<1])
```

に進む。

このように、ＡＴＭＳの矛盾管理機能を利用することにより、無駄な探索を避けることができる。

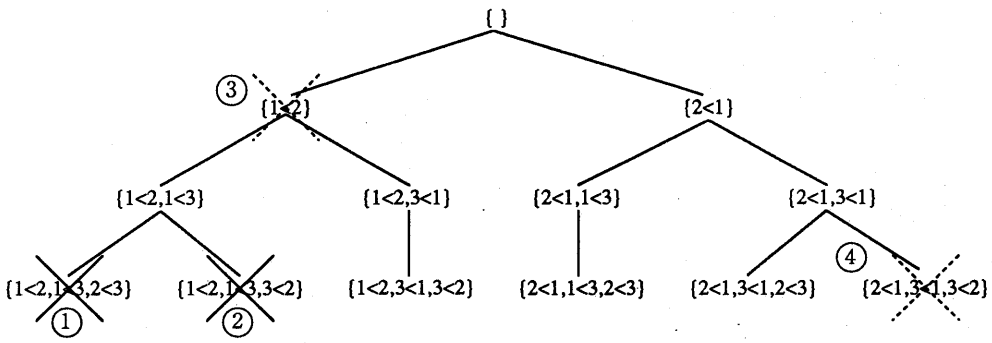

図3　探索木の例

①と②をチェックすることにより③,④が矛盾することがわかる

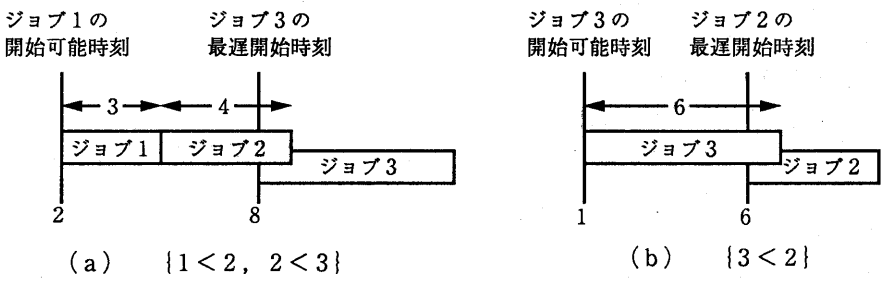

図4　矛盾する環境

ただし、実際の問題では、資源1個当りの処理するジョブの数は10ないしそれ以上であるから、仮説の数は数百程度になり、巨大な環境束を管理する必要がある。そのためのオーバーヘッドとATMSを利用することによる探索の効率化のどちらが実際の探索時間に効くかは問題の性質に大きく依存する。

9　問題の拡張

この節では、いままで議論を簡単に進めるために省略した部分について述べる。

（1）ジョブの資源への割り当て

本稿ではここまで、ジョブを処理する資源は決まっているものとして議論を進めてきた。しかし、通常のスケジューリング問題では、1つのジョブを処理できる資源は複数存在し、どの資源で処理するかによって処理時間が異なったりする。このジョブの資源への割り当てを扱うためには、基本的には、次の2通りの方法がある。

最も簡単な方法は、ジョブの資源への割り当てを最初に全ておこない、その後に各資源について、割り当てられたジョブの処理順序を決定するという方法である。この場合は8節までの議論は全く変更を受けない。ただし、一般にはジョブの資源への割り当てが、全体の計画に非常に大きな影響を与え得るから、あまり望ましい方法ではない。

2番目の方法は、逆に、資源への割り当てと処理順序の決定を同時におこなう方法である。この

ときは、仮説としてジョブの前後関係の他に、“ジョブｉは資源m_iで処理される”という仮説を加える必要がある。また、環境の無矛盾性を調べるとき、扱う不等式系は一般に以下のようになる。

$A_i \leqq T_i$
$T_i \leqq B_i$
$T_i + C_{ij} \leqq T_j$
$A_i \in \{A_{i\,m_i}\}$
$B_i \in \{B_{i\,m_i}\}$
$C_{ij} \in \{C_{i\,m_i\,j\,m_j}\}$

すなわち、５節では定数とすることができた、A_i，B_i，C_{ij}がジョブｉ、ｊがどの資源で処理されるかにより複数の値をとるようになる（どの資源で処理するかが決まってしまえば定数になる）。この条件式系では、解の存在性を調べるのが難しい。そこで、この条件式系を通常の不等式系に緩和して、緩和された問題について解の有無を調べる。

$A_i \leqq T_i$
$T_i \leqq B_i$
$T_i + C_{ij} \leqq T_j$
$A_i = \mathrm{min}\ \{A_{i\,m_i}\}$
$B_i = \mathrm{max}\ \{B_{i\,m_i}\}$
$C_{ij} = \mathrm{min}\ \{C_{i\,m_i\,j\,m_j}\}$

ただし、m_i，m_jが決まったら、不等式系にA_i，B_i，C_{ij}に$A_{i\,m_i}$，$B_{i\,m_i}$，$C_{i\,m_i\,j\,m_j}$を代入した不等式

$A_{i\,m_i} \leqq T_i$
$T_i \leqq B_{i\,m_i}$
$T_i + C_{i\,m_i\,j\,m_j} \leqq T_j$

を加えればよい。この方法では、扱う不等式の形は、やはり

$T_i \leqq \mathrm{Const.}$
$\mathrm{Const.} \leqq T_i$
$T_i + \mathrm{Const.} \leqq T_j$

に限られるので、５節の議論は変更する必要はない。ただし、あくまでも緩和された不等式系の解の有無であるから、実際には矛盾している環境の矛盾をチェックできない場合がある。しかし、実際の探索にはあまり影響はないと思われる。

（２）資源の切り替え時間

一般のスケジューリング問題では、連続する２個のジョブの間に一定の休止時間（切り替え時間）D_{ij}が要求される場合がある。このときは、各仮説に対応する不等式を

$T_i + L_i \leqq T_j$

のかわりに

$T_i + L_i + D_{ij} \leqq T_j$

とすればよい。ただし、推論の単調性、すなわち、２個の環境Ｅ，Ｅ′に関して、

$E \supset E'$，E' is nogood → E is nogood

を保証するためには、以下の条件を満足しなければならない。

$D_{ij} + D_{jk} \geqq D_{ik}$

１０　まとめ

本稿では、スケジューリング問題における探索の効率化のために、ＡＴＭＳの利用について検討をおこなった。その結果、同一資源で処理される２個のジョブの前後関係を仮説とし、各ジョブの開始時刻の下限値と上限値を計算することにより

環境の矛盾・無矛盾を調べるという枠組みで、ＡＴＭＳを利用できることを示した。

ＡＴＭＳの有効性が発揮されるのは、基本的にバックトラックが多発するような場合であるから、一般に解が求まるまでにある程度の時間が必要となる。これは、問題自身が難しい（解空間が小さい）場合や、ある程度の最適性が求められる場合、また有効なヒューリスティックが発見されていない場合などである。実際に、探索空間全体のうちどの程度の部分を探索する場合に有効となるかに関する検討・評価は今後の課題である。

参考文献

[1] 湯上、原：　スケジューリング問題の解法の検討，第４回人工知能学会全国大会

[2]　Doyle,J.：A truth maintenance system, Artificial Intelligence 24(1979)

[3] de Kleer,J.：An assumption-based Truth maintenance System, Artificial Intelligence 28(1986)

[4] de Kleer,J.：Back to Backtracking:Controlling the ATMS, Proc. of AAAI-86

[5] 溝口文雄　他：　制約論理プログラミング，共立出版

[6] Dechter,R.：Temporal Constraint Networks, Proc. of 1st KR(1989)

[7] 湯上、原：　スケジューリング問題における探索方式，第４１回情報処理学会全国大会