U.D.C. 658.513：681.3

清水建設研究報告
第51号平成2年4月

生産システムにおける知的スケジューリング手法に関する研究

池田佳則
(技術研究所)
加藤暁朗
(技術研究所)
小西康之
(技術研究所)

§1. はじめに

製造業において，近年生産システムに課せられている課題は「注文に応じて多品種の製品を迅速に生産すること」である．消費者のニーズが多様化する中で，製品の品種数が増加し，ライフサイクルは短くなり，またリードタイムに対する顧客の要求が強くなっていることがその背景にある．

この課題に対して，ハードウェア技術が急速に進展しつつある．そして一方で，複数の自動機械をバランスよく運用して，多品種少量生産を可能にするソフトウェア面での技術革新が急がれている．

生産スケジューリングに関する研究は，生産システムの課題にソフトウェア面から取り組もうとするものである．近年研究が盛んな知識工学応用技術は，これまでシステムに取り込みにくかった経験に基づく人間の知識をコンピュータに記憶させることにより，人間が行なう複雑な判断を自動化しようとするもので，多くの知識を必要とするスケジューリング問題への適用が期待されている．

本報告では，生産スケジューリング問題の構造を整理し，知識工学を応用することを前提として開発されたコンピュータ言語，ツールを用いて作成したスケジューラのプロトタイプを紹介し，インテリジェントなスケジューラの開発上の課題について述べる．

§2. 生産スケジューリング問題

2.1 生産スケジューリング問題

生産スケジューリング問題を，本稿では次のように定義する[1]．

「機能の異なる複数の機械（または設備，装置，作業者）からなる工場で，多品種の製品（または部品）を製造する場合に，各機械がどの部品をいつ加工するかという手順を決定する問題」

生産スケジューリング問題を，次に分類したライン形態に従って考える[2]．

(1)ジョブを処理する機械の順序がオーダー毎に決まっている場合：ジョブショップ型

(2)ジョブを処理する機械の順序がすべて同一の場合：フローショップ型

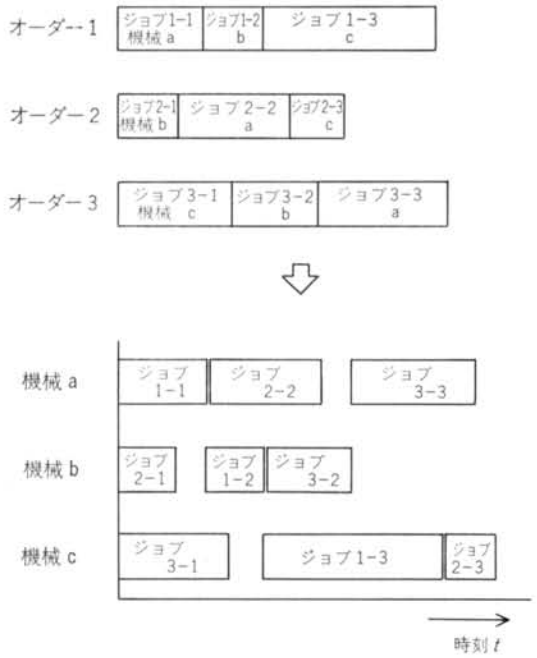
(3)ジョブを処理する機械の順序が自由な場合：オープンショップ型

(4)ジョブを処理する機械の順序がすべて同一でラインバランスがとれている場合：フローショップの特殊型

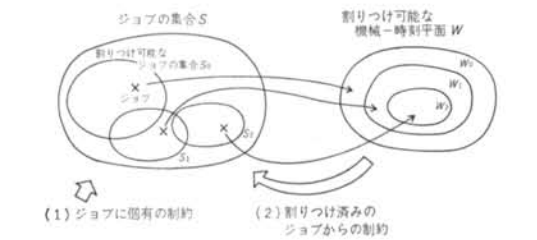
生産スケジューリングとは，機械毎に計画を与えることである．したがって，少品種の量産工場に見られる上記(4)に該当するラインの場合，部品は一連の流れに沿って一定時間後に完成するので，機械毎の計画を作成する必要はない．また，(3)に該当するラインの場合，空いている機械に順にジョブを割り付ければ良いので，計画作成の必要性は少ない．生産スケジュールが必要なラインはおもに(1)，(2)と考える．また，(2)フローショップ型は(1)ジョブショップ型のスケジュール作成において，すべてのジョブの処理順序を同一にして考えればよいので，ジョブショップ型の特殊型として扱うことができる．したがって，本稿ではジョブショップスケジューリングに焦点を当てて，議論を進めることとする．

2.2 ジョブショップスケジューリング問題の構造

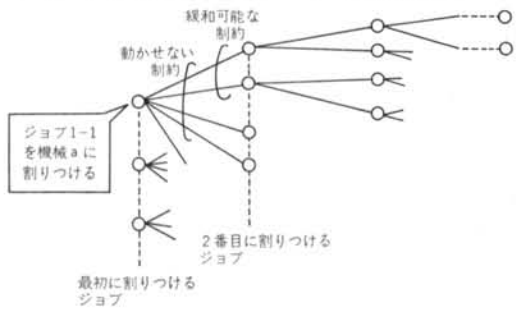
ジョブショップスケジューリング問題は，複数のジョブからなるオーダーを加工順序に従って，ジョブ単位に各機械の時刻座標上に割り付ける問題である（図—1）．各ジョブには使用する機械のほか，使用する治工具，作業担当者などの制約があり，各オーダーには材入日，納期などの制約がある．問題を解くには，これらの制約の範囲で「納期遅れの総和を最少にする」，「機械の平均稼働率を最大にする」などの要求を満たすようにジョブの割り付けを行なえば良い．



図—1　ジョブショップスケジューリングの例



図—2　ジョブショップスケジューリング問題の構造



図—3　木構造で表現したジョブショップスケジューリング問題

割り付けを行なう各機械とその動作が可能な時間を座標軸とする平面を機械—時刻平面Wとし，割り付けの対象となるジョブの集合をSとすると，Sからジョブを選び，平面W上の該当する位置に配置する問題として扱える（図—2）．このとき，Sからジョブを選ぶには，(1)ジョブに固有の制約(材入日など)，(2)割り付け済みのジョブからの制約（使用する機械の空き状況など）から割り付け可能なジョブの集合 S_i を設定し，さらにこの中から評価軸に従って1つのジョブを決める．一方，平面Wへの配置は，選ばれたジョブの制約の範囲で許容されるように行なう．このとき，選択したジョブを配置する平面Wは，割り付けを進める毎に狭くなっていき，割り付け可能なジョブを決める制約も変化する．

ジョブの選択方法と機械—時刻平面Wへの配置方法を決めることにより，組み合せ問題として木構造で表現することができる（図—3）．ジョブの枝は割り付けを進める都度，制約の範囲で割り付け可能な方法の数だけ展開される．得られる解は枝の道筋で表わされ，評価の高いスケジュールをこの中から選ぶことになる．展開される枝の数を決める制約には，加工順序や機械・治工具の占有などのように動かせない制約のほか，機械の1日の加工可能時間，納期など，ある程度緩和が可能な制約とがある．したがって，最終的に展開される枝の数，すなわち解の数は後者の制約の与え方によって膨大な数になることもあり，逆に解が得られない場合もあり得る．

2.3　問題解決の戦略

スケジューリングを行なう上での戦略を，ジョブを選択し機械—時刻平面に配置するための戦略と，組み合せ問題とした上で解を得るための戦略に分けて考える．本稿では，前者を「割り付け戦略」，後者を「処理戦略」と呼ぶこととする．

割り付け戦略は，次に割り付けるジョブの選択規則と選択したジョブの機械—時刻平面への割り付け方法によって与えられる．前者は機械の平均稼働率，納期遅れ，納期ずれ，仕掛品量などの制約および評価の結果として得られ，後者は割り付けを行なう時刻の方向および空き時間内での位置決めを行なう手順として与えられる．前者の例としては，SPT (Shortest processing time；最小加工時間規則)，LWKR (Least work remaining；最小残り作業量規則)，TWORK (Total work；総作業量規則) などがあり，これまでに比較評価が行なわれている[3]．後者の例には，(1)納期からさかのぼって割り付ける方法（バックワード割り付け），(2)現時点から将来に向けて割り付ける方法(フォーワード割り付け)がある．

一方，処理戦略は大きく分けて，(1)最適化解法，(2)発見的解法の2通りが考えられる[4]．前者は，考えられるすべての組み合わせについて割り付けパターンを試みる全解探索を基本として，これを効率化する方法である．

後者は，評価関数によって割り付けるジョブを絞り込むシミュレーションに似た方法である．全解探索によって最適解を得ることは理論的に可能であるが，現在のところコンピュータの処理能力の制約から難しく，明らかに最適解につながらない道筋を棄却する，いわゆる枝刈りによって探索の効率化を図ることになる．したがって，最適化解法では，枝刈りの方法および処理時間が課題となる．また，発見的解法では逐次的に評価した枝を延ばして行くことから，最終的に最適解につながる枝を捨ててしまう危険性がある．したがって，最適解に近い近似解を導くための逐次評価の方法が課題となる．

本稿では，処理戦略について述べることとし，最適化解法および発見的解法の双方について，各々に適したコンピュータ言語を用いてプロトタイプの作成を試みる．割り付け戦略は，一般によく用いられる機械の稼働率最大化，または納期遅れ最小化を評価関数とするフォーワード割り付けを基本として用いている．

§ 3. ジョブショップスケジューリング問題の解法

3.1 最適化解法

最適化解法とは，「制約を満たすジョブの割り付け方の組み合わせを可能な限り生成し，その中から最も要求を満足するスケジュールを見付け出す」方法である．ジョブの割り付け方の組み合わせをすべて求めるには，たかだか数台の機械に数個のジョブを割り当てる場合でも組み合わせの数は数十万通りにも達し，いわゆる組み合わせ的爆発を起こすことになる．本稿では，例題を用いてすべての組み合わせを求める全解探索を用いた解法と，“枝刈り”を用いて組み合わせ的爆発を回避する解法について説明する．

まず，割り付けの制約を加工順序と加工時間だけに限定して，機械の稼働率を最大にするような割り付け方を求める場合について考える．図—4は，機械a，b，cに対して，矢印の順で加工を行うオーダーを示している．a(4)，b(6)，……は個々のジョブを，また（ ）内の数字は加工時間を意味する．

12個のジョブを，機械の稼働率が最大となるように機械a，b，cにフォワード割り付けによって割り付けを行なう．このとき，全解探索では次のような手順でスケジュールを生成する．

オーダー1 …… a(4) → b(6) → c(5)
オーダー2 …… b(2) → c(7) → a(6)
オーダー3 …… a(3) → c(6) → b(4)
オーダー4 …… c(4) → a(5) → b(8)

図—4 ジョブショップスケジューリング問題の例題

手順1：12個のジョブの中から機械a，b，cに割り付け可能なジョブをすべて調べる．この例では，a(4)，b(2)，a(3)，c(4) の4つのジョブが割り付け可能である．

手順2：手順1で求めた割り付け可能なジョブから，最初に機械に割り付ける組み合わせをすべて求める．この例では，最初の割り付けの候補となる組み合わせは，〔a(4)，b(2)，c(4)〕と〔a(3)，b(2)，c(4)〕の2通りがある．

手順3：手順2で求めた組み合わせの各々について，残りのジョブから割り付け可能なジョブの集合を抽出し，この中から1つのジョブを割り付ける．

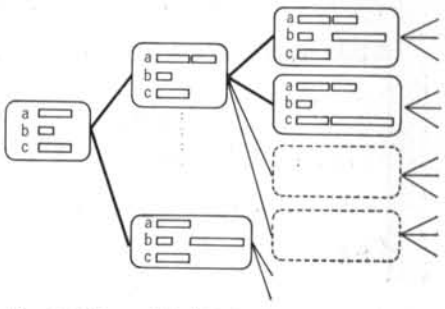
手順4：手順3の操作をすべての場合について，割り付け可能なジョブが無くなるまで繰り返す．

スケジュール生成のプロセスを図—5に示す．

3.1.1 縦型全解探索による最適化解法

割り付け方をすべて求めるための解法には，基本的な戦略として縦型全解探索と横型全解探索の二つがある[5]．縦型全解探索とは，なんらかの方法（例えば，オーダー番号順）に従って割り付け可能なジョブを順番に割り付け，一通り最後まで割り付けたところで，別の割り付け方を最後に割り付けたジョブからさかのぼって逆順に調べていく方法（バックトラックという）である．

エンジニアリングワークステーション（Sun-3 160 C）上で論理型言語（Quintus-Prolog Release 2.2）を用いてこの縦型全解探索を試みたところ，この例題では縦型全解探索によりすべてのスケジュールを生成し，稼働率の最も高いスケジュールを得ることができた．論理型言語が保有するバックトラック機能を利用することによって，縦型全解探索が容易に実現できている．しかし，この程度の例題でも組み合わせの数は膨大であり，論理型言語の中では，処理速度がトップクラスの言語処理系（Quintus-Prolog）を用いてもかなりの処理時間がかか



図—5 スケジュール生成のプロセス

る．現実の問題では，機械，オーダーとも数十個を扱う必要があるため，現在のコンピュータ技術ではこの問題に対して縦型全解探索では対応できないことが確認できた．

3.1.2 横型枝刈り探索による最適化解法

縦型全解探索では，最も稼動率の高いスケジュールを求めるために，実行可能なスケジュールをすべて求めたが，その中には実際に人間（専門家）が一目見れば明らかに稼働率が低いものが含まれている．こういった明らかに目標を大きくはずれるスケジュール案は，初めから探索の対象としない方が効率が良い．そこで，割り付け途中のスケジュールの稼働率を評価して，稼働率の低いスケジュールを除去し，探索の枝の数を削減する横型枝刈り探索を試みた．

横型枝刈り探索では，縦型全解探索のように最適解を得る保証はなくなるが，探索空間の急激な膨張を回避できる．ここでは，機械5台(a, b, c, d, e)に各々5個のジョブからなる7個のオーダーを割り付ける例について，残す枝の数を変えて試行を繰り返し，処理時間の測定を行なった．その結果，稼働率の高いほうから順に50個の枝を残した場合6分程度の処理時間で，手作業で行なうスケジュールと比較して同等か，それ以上と評価できる結果が得られた（図—6）．

論理型言語（PROLOG）を用いた最適化解法のためのスケジューラの試作を通じて，①ジョブの選択にパターンマッチ機能が有効に機能すること，②割り付け結果の保存にリスト表現が有効であること，③バックトラック機能を用いて縦型探索を容易に実現し得ることが論理型言語を用いる有利な点として明らかになった．反面，横型探索を行なうときに頻繁に行なう情報のデータベース空間における書き換え操作に時間がかかることが不利な点として挙げられた．これは，PROLOG が診断型の問題解決を行なう後向き推論用に開発された言語であるためで，本来の目的と異なる使い方をしているために生じる問題と考えられる．

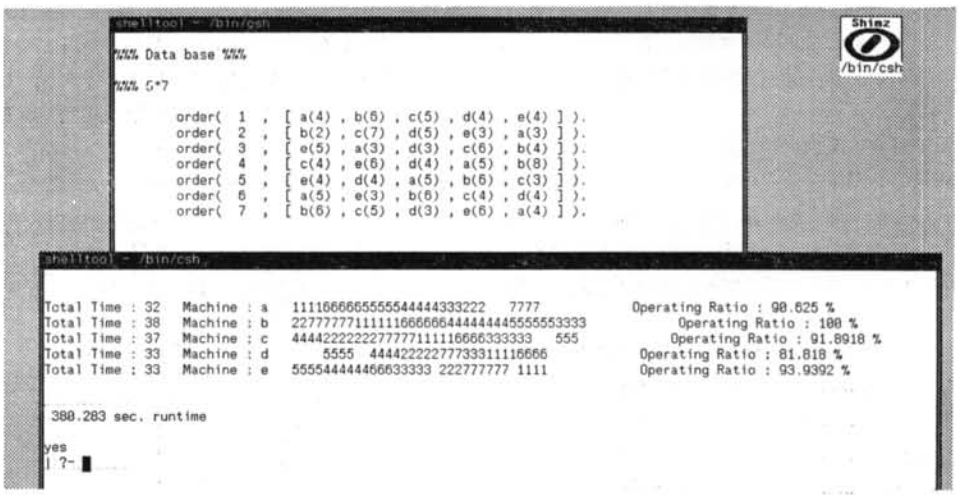
そこで，次に計画型の問題解決を行なう前向き推論用に開発された言語 OPS 83（カーネギーメロン大学 C. L. Forgy が考案した人工知能用言語）を用いて横型枝刈り探索を試みた．

OPS 83 は，ワーキングメモリーと呼ばれるパターンマッチを行なう一時記憶領域を持っており，割り付け途中のスケジュールを記憶させ，内容を高速に書き換えることができる．OPS 83 で試作したスケジューラでは，機械4台にオーダー5個の割り付けを行なう例について，残す枝の数を50個として実行したところ，処理はおよそ1分で終了した（NEC-PC 9801 VM を使用）．しかし，ワーキングメモリーの容量には制限があり，同じプログラムを用いて機械5台にオーダー7個の割り付けを試みたところ，メモリー容量をオーバーして解が得られなかった．PROLOG では，処理時間がボトルネックとなって残す枝の数を制限したが，OPS 83 ではワーキングメモリーがボトルネックとなった．

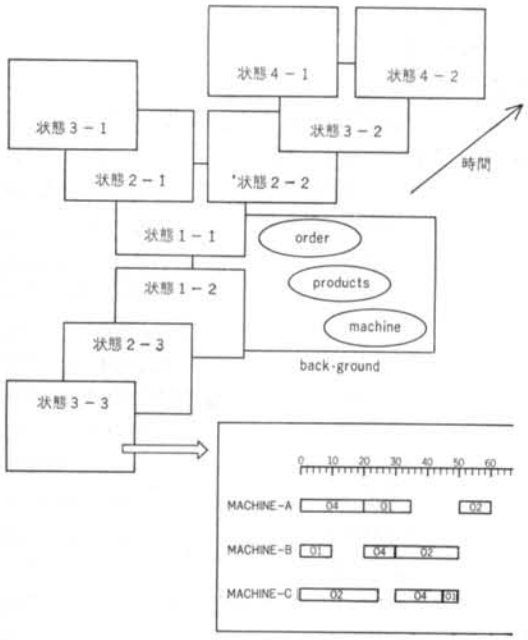
いずれの場合も，最適化解法を用いる場合は，実現可能な範囲で残す枝の数を決める方法が重要な課題となるといえる．

3.1.3 多重世界機能を用いた全解探索

スケジューリングの途中，あるいは終了した時点で，スケジューリングの過程に戻ってその状態を参照できれ



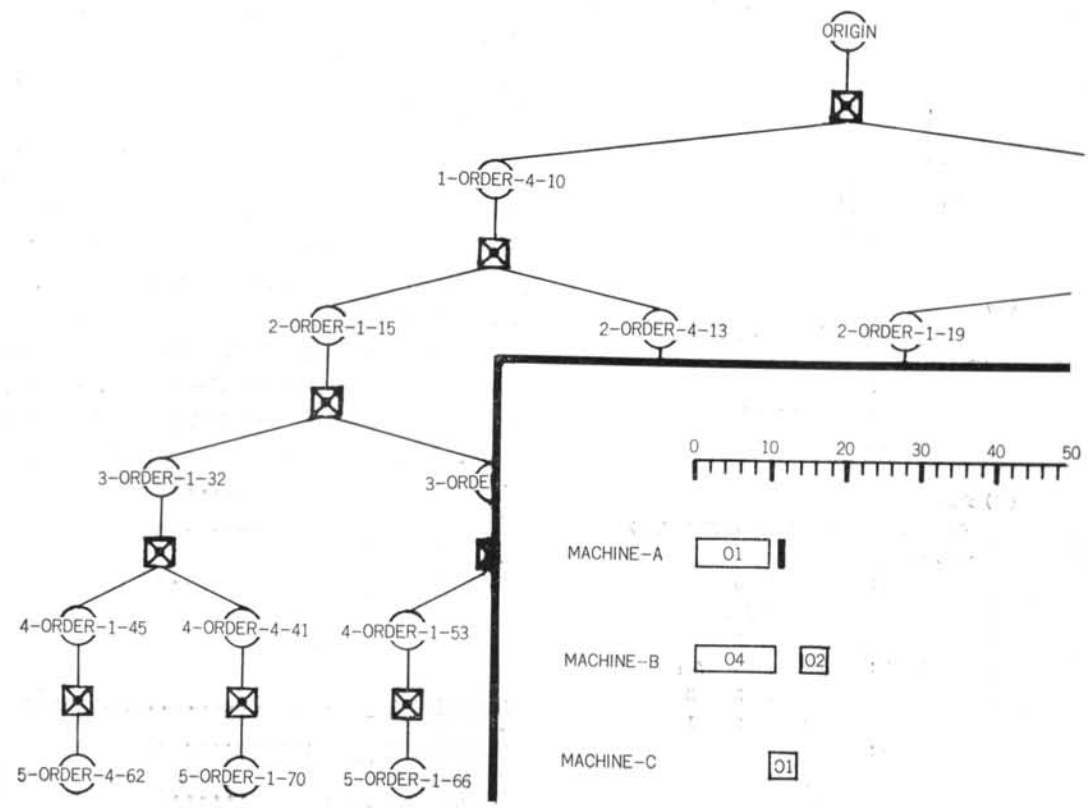
図—6 横型枝刈り探索によるスケジューリング結果



図―7　多重世界機能の概念

ば，変更，修正を行なう上で有利である．本稿では，KEE（米国 Intelli Corp 社が開発した AI ツール）による多重世界（ワールド）機能を用いて，その実現の可能性を検討した．多重世界とは，初期状態を保存したまま，状態が変わる都度生成される新しい状態を保存して複数の状態―世界を展開する機能である（図―7）．

図―8は，ジョブを割り付ける都度，新しいワールドを生成していく様子を示している．プロトタイプの作成を通じて，多重世界機能によりスケジューリング過程を保存することによって，割り付けルールの妥当性を確認することができた．しかし，KEE が診断用システム開発のためのツールとして開発され，後に計画型システム開発のための機能を付加していることから，処理時間が遅い，ルールの記述が行ないにくい等，いくつかの問題が認められた．今後，スケジュールの修正，変更を行なうエディター機能が重視される実用レベルの開発を行なうに当たっては，多重世界機能をシンプルに表現する方法を検討する必要がある．



図―8　多重世界を用いたスケジューリング例

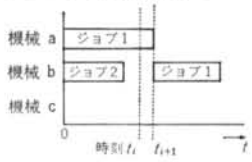
3.2 発見的解法によるスケジューリング

発見的解法によるスケジューリングでは，木構造によって表現されたジョブの枝をなんらかの方法で1本に絞り，枝を繰り返し延ばしていく手順を踏む．発見的解法では，候補となる枝の1本を残して残りのすべてを棄却するので，最適化解法のように組み合わせ的爆発を起こすことはない．したがって，実用に耐える大規模なモデルを扱うことが可能で，最適化解法に比べ現実的な方法といえる．発見的解法を，実用性を重視した近似解を得る方法と割り切って考えれば，スケジューラ開発上の課題は，人間が行なう複雑なノウハウをメンテナンスが容易な形でプログラム上に表現する方法に重点をおいて考えることができる．

ここでは，メンテナビリティの高いデータ，制約，戦略の表現を行なうことを配慮して LISP，OPS 83 で開発したスケジューラのプロトタイプを紹介し，スケジューラ開発で取り得る処理戦略について述べる．

3.2.1 時刻データの扱い方による処理戦略の違い

スケジューリングを進めるに当たって，単位時間毎に時刻を進め，その時刻に可能な割り付けを行なう方法と，単位時間を決めずにジョブ毎に割り付けを進めていく方法がある．本稿では，前者を時刻の進行に従って処理を進めることから時間駆動型（JIS Z 8121 でいう定間隔時間制御方式）と呼び，後者をジョブの割り付けによって処理を進めることからジョブ駆動型と呼ぶことにする（図—9）．



（i）時間駆動型の場合

〈時刻 t_{i+1} でジョブ1が割りつけ可能となる〉

——データ表現——

機械 a：空き状態

機械 b：ジョブ1を加工中

機械 c：空き状態

（ii）ジョブ駆動型の場合

〈ジョブ1を選択して，時刻 t_{i+1} に割りつける〉

——データ表現——

機械 a：加工開始時刻　0

　　加工終了時刻　t_{i+1}

　　加工ジョブ　1

機械 b：加工開始時刻　0　t_{i+1}

　　加工終了時刻　3　8

　　加工ジョブ　2　1

機械 c：加工開始時刻

　　加工終了時刻

　　加工ジョブ

図—9　時間駆動型とジョブ駆動型

OPS 83 で開発したプロトタイプは，時間駆動型を用い，ワーキングメモリー上に時刻の進行を管理するフレーム（OPS 83 ではエレメントと呼ぶ）を用意して，ある時刻で発火すべき割り付けルールが無くなったときに時刻進行のルールが発火して，すべての要素の現在時刻を進めている．この方法によると，時刻断面で関連する複数の要素の状態がモニターできるので，特に多くの制約が絡む場合や，扱う機械やジョブの数が多い場合にプログラムの開発が容易に行なえる．反面，すべての割り付けのルールを時刻を進める都度チェックするので，処理系によっては時間がかかる欠点をもつ．また，時刻を進める都度状態を更新するために，現在時刻より前にさかのぼって割り付けを行なうことができないことも欠点の一つである．

LISP で開発したプロトタイプは，ジョブ駆動型を用いている．この方法では，ジョブを割り付ける都度，割り付けた機械に加工開始時刻と加工終了時刻，およびジョブ番号を追加記入していく．ジョブ駆動型は，考え方が人間がバーチャートを引く作業に似ていることから，人間の思考プロセスをプログラムに容易に表現し得る．しかし，いくつかの要因が絡むと割り付けの手順が複雑になり，デバッグが困難になることもある．図—10は，納期を優先して，オーダー4を先に割り付け，後にオーダー3を割り付けた例である．このように，ジョブ駆動型では割り付けを行なう時刻さえ与えれば，必ずしも時刻順に従わなくても割り付けを進めることができる．

3.2.2 データの表現方法

スケジューリングのデータは，機械，ジョブなどのスケジューリング要素に付随するデータであり，その多くはスケジューリングの進行とともにその内容が書き換えられる．また，スケジューリングの途中経過は最終結果

```
MACHINE_A ___···········4
MACHINE_B ___44444444444
MACHINE_C ___···········44
```

⇩

```
MACHINE_A ___···········4··333333333
MACHINE_B ___4444444444333
MACHINE_C ___333333·····44
```

図—10　ジョブ駆動型による納期優先割り付け

```
ORDER_1
"       " ORDER_1
"       " (OPERATION_SEQUENCE (MACHINE_A) (MACHINE_C) (MACHINE_B))
"       " (NEXT_OPERATION)
"       " (FINISH_TIME 18)
"       " (MATERIAL_IN 0)
"       " (DUE_TIME 20)
"       " (STATUS DONE)
"       " (REMAIN_PROCESS_TIME 0)
"       " (PROCESS_TIME (10) (4) (4))
MACHINE_A
"       " MACHINE_A
"       " (ORDER_SEQ ORDER_1 ORDER_2 ORDER_3 ORDER_4)
"       " (BEGIN_TIME 0 10 15 24)
"       " (END_TIME 10 15 24 25)
"       " (WAITING_ORDER)
"       " (OPERATION_TIME 25)
"       " (OPERATION_RATE 100.0)
```

(a) LISP によるフレーム表現の例

```
(order
    order_no = !order_1!;
    baket_value = 3;
    baket_no = 1;
    product_name = !small!;
    status = !null!;
    remain_t = 0;
    priority = 1;
    clock = 0;)
```

```
(process
    name = !narabe!;
    machine_no = 1;
    status = !vacant!;
    remain_t = 0;
    occupation = !null!;
    clock = 0;
    running_time = 0;)
```

(b) OPS83 によるフレーム表現の例

図—11 フレーム表現の例

```
(defun release22(machine_list)
   (unless (eq (all_finish_check (sum_next_order machine_list)) 'finished)
      (setf order_list (sum_next_order (search_machine machine_list)))
      (cond ((null (pre_ope_finish_check (mrl_list_check order_list)))
             (setf order_list (ope_ready order_list)))
            (t (setf order_list (pre_ope_finish_check
                                 (mrl_list_check order_list)))))
      (setf order (car (search_order_order1 order_list)))
      (make_schedule (valuegl order 'next_operation)
                     order)
      (release22 machine_list)))
```

ジョブを1つ選ぶ部分

```
(defun search_order_order1(order_list)
   (cond ((null order_list) nil)
         (t (let ((priority_list))
              (dolist (order order_list)
                (let ((machine (valuegl order 'next_operation)))
                 (unless (equal (valuegl order 'status) 'done)
                    (push (list order
                                (cal_yoyu1 machine order))
                          priority_list))))
              (setf return
                 (twin_head (equal_list (sort priority_list 'cadr) nil)))
))))
```

```
(defun cal_yoyu1(machine order)
     (setf return (- (valuegl order 'due_time)
                     (max
                         (bool_int(valuegl1 machine 'end_time))
                         (bool_int(valuegl order 'finish_time)))
                     (valuegl order 'remain_process_time))))
```

(1) 材入日を満たすジョブを関数 mrl-list-check で調べ，(2) 機械の空き状態を関数 pre-ope-finish-chek で調べ，(3) 関数 search-order-order1 で，納期までの余裕時間を計算する関数 cal-yoyu1 を用いて，ジョブを1つに絞り込んでいる。

図—12 LISP を用いたプロトタイプでのジョブの選択

の一部分であり，実行中に制約として利用するものでもある．したがって，使用するデータは初期条件，途中経過，最終結果を区別することなく，機械，ジョブなどのスケジューリング要素毎にまとまっていることが望ましい．OPS 83 では，保有するフレーム表現，エレメントを用いて機械，ジョブ，作業者のデータ表現を行ない，LISP ではフレームのデータ操作を行なう関数を予め用意してフレーム表現を実現している（図—11）．

ジョブ駆動型のスケジューラでは，割り付け済みの時間帯を次の割り付けの制約とするために，機械毎に状態別の時刻を集合として保有できると都合がよい．LISP で作成したプロトタイプではこれを実現するために，データ数を予め決めずに集合として定義できるリスト表現が有効に機能した．OPS 83 で同様の表現を実現するには，スケジューリング要素に直接関係のないダミーのエレメントを作る必要があり，メンテナビリティの点でリスト表現が扱えるか否かは，処理戦略をも制約する重要な要因といえよう．

3.2.3 ルールの記述方法

ルールの記述は，2.3 で述べた「ジョブを選択し，機械—時刻平面への割り付け」をプログラム上に表現することである．ジョブの選択のためのプログラム記述は，各々のプロトタイプについて次のような方法で行なった．

LISP を用いたプロトタイプでは，リスト表現されたジョブの集合から，制約毎にジョブの絞り込みを行なう「ふるい」の役割を果たす関数を組み合わせて，制約を満たすジョブの集合を抽出し，評価関数によって一つのジョブを選んでいる（図—12）．OPS 83 を用いたプロトタイプでは，同様の手順をルールの条件部に並列に記述している．これは，ワーキングメモリー上から条件を自在に組み合わせてエレメントを抽出する OPS 83 のパターンマッチ機能を利用したもので，メンテナビリティの高いルール記述ができた．

機械—時刻平面への割り付けは，対象となるジョブと機械の双方の状態を書き換えることにより行なう．OPS 83では，パターンマッチ機能により両者の組み合わせを同時に抽出できることから，ルールを簡略化して記述できた（図—13）．いずれの場合も，選択のルールが制約に対応した形で記述されており，FORTRAN などのような手続き型言語に比べて，ルールの追加，修正が容易に行なえる．

これまでに述べたとおり，OPS 83 の保有するパターンマッチ機能は，ルール記述の簡略化に非常に有効である．しかし，状態変化の都度すべてのエレメントに対して行なうメンテナンスルールや統計データ計算のためのルールなど，ルールの発火順位を作為的に制御したい場合には，制御用のエレメントを作成して対応する必要があり，逆に煩わしい点もあった．LISP では，すべての制御関数を開発者が作成する必要があり，開発に時間が

```
rule assign_pre_chunyu
{
        &T(time_control);
        &P(process clock = &T.clock ;
                   name = pre_chunyu ; status = vacant);
        &O(order clock = &T.clock ;
                 baket_no = 1 ; status = narabe ; remain_t = 0);
         ~(order baket_no = 1 ; status = narabe ; remain_t = 0;
                priority < &O.priority );
        &W(worker clock = &T.clock;
                   status = not_busy);
        &D(products name = &O.product_name);
        &R(order_record order_no = &O.order_no ; baket_no = &O.baket_no)
-->
        write()!assign pre_chunyu * order_no = ! , &O.order_no ,
                ! baket = ! , &O.baket_no ,
                ! machine = !, &P.machine_no ,
                ! worker = ! , &W.name , '¥n';
        call assign_excute (&O , &P , &D.pre_chunyu_t);
        modify &W(status = busy ;
                   remain_t = &D.pre_chunyu_t);
        modify &R(pre_chunyu[1] = &T.clock ;
                   pre_chunyu[2] = &T.clock + &D.pre_chunyu_t;
                   pre_chunyu_m = &P.machine_no;
                   pre_chunyu_w = &W.name);
        call running_record (&P , &D.pre_chunyu_t);
        call running_record_W (&W , &D.pre_chunyu_t);
);
```

図—13 OPS 83 の割り付けルールの例

かかるが言語の持つ制御構造を意識する必要がなく，自由度の高い記述が可能である．

§ 4. おわりに

プロトタイプの作成を通じて，知識工学応用技術を生産スケジューリング問題に適用する上で明らかにすべき事項を検討した．その結果，知識工学応用技術を利用することにより，これまでコンピュータで扱いにくかったスケジューリングの木構造の探索を容易に実現し，人間が行なう処理手順をプログラムに表現する上で，メンテナビリティの高いデータ表現，ルール表現を可能にすることが確認できた．また，ハードウェアや言語処理系の能力の範囲で，処理し得る枝の数を決める方法，使用する言語を選択する上で考慮すべき重要な機能，人間が行なう処理手順をプログラムに表現する上での処理戦略など，現実のスケジューラを開発する上で要点となる課題を明らかにした．

今後は実用化を図る上で，現実の生産スケジューリング問題において必要となる機能をさらに調査する必要がある．現実に運用して効果を上げる知的なスケジューラには，本稿で扱った解法のほかに，スケジューリングの過程，またはスケジュールの完成後に人間が介入して修正，変更を行なうエディターの機能が重要な役割を果たすと思われる．また，現実のスケジューリング過程では制約が厳しいために解が得られない場合に，納期，加工時間，１日の機械の加工可能時間などを変化させてスケジュールの調整を行なっている．これは，時間値が曖昧さを含む数値として扱われているためで，近年研究が盛んなファジィの概念を用いることで，制約の緩和を行なうスケジューラが実現できると考えている．エディター機能の実現，制約の緩和については，今後の課題として取り組んでいく計画である．

＜参考文献＞

1) 田村担之："FMS スケジューリングにおける多目的評価と知識工学" 大阪大学知識科学研究会（1986年）
2) 石井博昭："スケジューリング理論の研究の復権をめざして" 日本OR学会秋期研究発表会アブストラクト集（1987年）
3) R. W. Conway, W. L. Maxwell & L. W. Miller : "Theory of Scheduling" 日刊工業新聞社（1971）
4) 田部勉："生産スケジューリングへのエキスパート・システムの利用" 日本経営工学会誌（1989年）
5) N. J. Nilsson : "Principles of Artiflcial Intelligence" Tioga Publishing (1980)
6) 森戸晋，相沢えり子："SLAM II によるシステム・シミュレーション入門" 構造計画研究所（1986年）
7) C. L. フォギー："人工知能用言語 OPS 83" パーソナルメディア社（1986年）
8) G. L. Steele, Jr., 他："Common LISP" 共立出版（1985年）
9) 池田佳則，他："機械ショップのスケジューリング問題における AI 的解法と課題" 第１回インテリジェント FA シンポジウム講演論文集（1987年）
10) 石井博昭，他："ファジィ処理時間を持つスケジューリング問題" 日本OR学会秋期研究発表会アブストラクト集（1988年）