スケジューリング問題への制約プログラミングの適用
― ILOG Solver/Scheduler を実例として―
Application of Constraint Programming for Scheduling Problems
-- Case Studies by ILOG Solver/Scheduler --

野末 尚次
(株)数理モデリング研究所

Naotugu Nozue
Math-Model Research Inc.
nozue@math-model.co.jp

Abstract This lecture explains how constraint programming(CP) can be used to solve scheduling problems and describes essentials of CP, such as declarative approach, constraint propagations, global constraints and search algorithms, and finally shows some case studies using ILOG Solver/Scheduler.

1 はじめに

最近の企業を取り巻く環境は、サプライチェーンの発展によるダイナッミクな多品種少量生産の実現、資源の有効利用、生産方式の効率化、環境負荷の軽減など、厳しい状況になってきました。

この流れに適応して行くためには、資源の調達、製造、物流を結合した効率的な計画の立案が不可欠ですが、現状では、個々の計画に対して、シミュレーション的な手法により、実行可能な実施案を立案しているケースが多く、効率の良い計画を作成する段階には到達していません。

これは、我が国のソフトウェア開発企業が、計画系のソフトウェアの知識やパッケージに対して、十分な知見を持っていないためだと思います。

また、我が国の計画系の研究者が、実務的なレベルの複雑性と規模を持った問題にあまり興味を示さないという点も影響していると思われます。

この１０年間に、計画系のソフトウェア・パッケージは、大幅な性能向上と低価格化と同時に、安価なパソコンで実行可能となっており、中小企業でも十分利用可能な段階になっています。

代表的なソフトウェア・パッケージとしては、「(整数) 線形計画法」と「制約プログラミング」が、大きな成果を挙げています。

このソフトウェアを利用する際の特徴は、従来のソフト開発のように、計画の作成手順のプログラミング（手続的アプローチ）を行う必要がないことです。計画の制約条件を記述（宣言的アプローチ）すれば、後はパッケージの計算エンジンが解を探索してくれます。

この宣言的アプローチは、システムの開発・保守の面からも非常に優れています。

線形計画法は、比較的よく知られていますが、制約プログラミングは、あまり知名度が有りません。しかし、制約プログラミングは、リソース制約のあるスケジューリング問題のように、組合せ問題に非常に有効です。

この講演では、制約プログラミングのサーベイではなく、この概念のバックグランドを理解していただき、スケジューリング問題への応用が見通せるように大雑把な話を致します。

尚、今回の講演では、私が開発に使用しております ILOG 社の制約プログラミングのパッケージ（ILOG Solver と ILOG Scheduler）をベースにお話しますのでご了承ください[文献１、２]。

理論的に厳密な話や他の制約プログラミング・パッケージに興味のある方は、文献３、４のサーベイを参照してください。

2 宣言的アプローチ

ソフトウェアの開発法は、次の２種があります。

- **手続的アプローチ**：対象とする問題に固有な解法手順（How）をプログラミングして、ソ

フトを開発する。

- **宣言的アプローチ**：問題の制約条件の定義（What）をプログラミングして、解法は汎用アルゴリズム（パッケージ）を使用して開発する。

図１に、鶴亀算を例に、両者の違いを示します。宣言的なアプローチでは、ソフトウェアの開発の信頼性が高く、また、時間の経過に伴う仕様変更（百足も加わったら？）に対するメンテナンスも容易ですが、「連立方程式の解法」という汎用のソフトウェアを知っていないと適用できません。制約プログラミングも同様で、「制約充足理論」という知識がないと宣言的なアプローチは適用できません。

つる･かめ算の計算

鶴と亀が足して4匹、足の合計が１２本の時、鶴と亀の数は？

手続的アプローチ	**宣言的アプローチ**
1) 全部鶴と仮定すると、足は8本	1) 鶴と亀の数を X、Yとする
2) 余った足の数 １２－８＝4本	2) X＋Y＝4　2X＋4Y＝１２ (定義)
3) 亀の数 ＝ 4本／2 ＝2匹	3) 連立方程式を解く (探索)
4) 鶴の数＝4－2＝2羽	4) X＝2　Y＝2

図1　開発アプローチ

3　制約充足理論入門

制約プログラミングは、問題の与えられた変数に対して制約条件を満たす解を「バックトラック法」により探索します。

バックトラック法では、各変数に順番に値を設定して、制約違反（Fail）となった場合は次の値を設定します。その変数の値を全て調べてしまった場合には、直前に設定していた変数から繰り返し、解空間を虱潰しに調べます。

鶴亀算では、変数の値域は 0～4 の整数とします。

変数：X[0.. 4]　Y[0.. 4]

制約：X+Y＝4、2X+4Y=12

この問題のバックトラック法による履歴を図２に示します。この探索により、X=2、Y=2 という解が得られました。

3.1　制約伝播の導入

バックトラックにより原理的には解を得ることが出来ますが、制約違反となる組合せを繰り返し

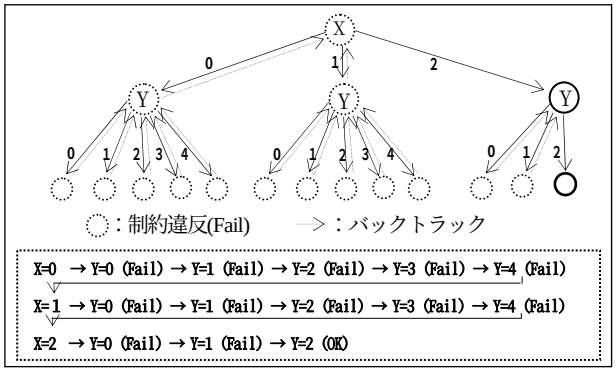

図2　バックトラックによる探索法

調べており、あまりにも効率が悪いため実用には使えません。

これを解消する手段として、制約違反となる変数の値を事前に削除することにより、バックトラックによる無駄な数え上げを回避する方法が、**制約伝播法**として提案されています。

例として、変数の取る値の上限値と下限値を、各制約式に基づいて縮小する**境界制約伝播法**の例を示します。

変数の値域：X[Lx..Ux]、Y[Ly..Uy]、整数値

制約伝播式：x+y=4

RUx-1: x ≦ (4 － Ly)
RLx-1: x ≧ (4 － Uy)
RUy-1: y ≦ (4 － Lx)
RLy-1: y ≧ (4 － Ux)

制約伝播式：2x+4y=12

RUx-2: x ≦ (12 － 4*Ly)/2
RLx-2: x ≧ (12 － 4*Uy)/2
RUy-2: y ≦ (12 － 2*Lx)/4
RLy-2: y ≧ (12 － 2*Ux) /4

初期値：Lx=0、Ux=∞、Ly=0、Uy＝∞

初期制約伝播：

RUx-1:　x≦ (4－Ly) =4　　=>　Ux=4
RUy-1:　y≦ (4－Lx) =4　　=>　Uy=4
RUy-2:　y≦ (12－2*Lx)/4 = 3　　=>　Uy=3
RLy-2:　y≧ (12－2*Ux)/4 = 2　　=>　Ly=2
RUx-1:　x≦ (4－Ly) =2　　=>　Ux=2
RLx-1:　x≧ (4－Uy) =1　　=>　Lx=1
RUy-2:　y≦ (12－2*Lx)/4 = 2.5　　=>　Uy=2
RLx-1:　x≧ (4－Uy) =2　　=>　Lx=2

伝播結果：X[2..2]、Y[2..2]で、X=2、Y=2 の解が得られました。

図3　境界制約伝播

この制約伝播により、今回は数え上げを行わずに解が得られました。

制約伝播により動的に不要な解候補を削除して探索空間を狭めて、効率的に探索する方式が、制約プログラミングの基本です。

3.2　制約伝播のレベル

制約伝播を徹底的に行えば、原理的には、数え上げを行わずに解を得られますが、不要な要素を取り除くアルゴリズムは、一般的に NP となり、

指数関数的に処理時間が掛かってしまい実用的ではありません。

このため、制約伝播は不十分でも、処理時間の短いアルゴリズムが提案されており、これらを用いて解空間を縮小しながら、バックトラック法で探索します。

Queen の配置問題を例に説明します。４×４のボード上にチェスの Queen を４個を次の条件を満たすように配置（図４参照)します。

(1) 各列には、１個の Queen を配置
(2) 各行には、１個の Queen を配置
(3) 各対角線上には、最大１個の Queen を配置

この問題は、各行に置く Queen の位置を変数とすることにより、図５に定式化されます。

	1	2	3	4
1		**Q**		
2				**Q**
3	**Q**			
4				**Q**

図4 Queen 配置例

定式化:各行のQueenの位置を変数
変数: x1、x2、x3、x4
制約条件:
1)縦: x1≠x2、x1≠x3、x1≠x4、
x2≠x3、x2≠x4、x3≠x4
2)右上: x2＋1 ≠x1
x3＋2 ≠x1、x3＋1 ≠x2、‥
3) 右下: x2－1 ≠x1
x3－2 ≠x1、x3－1 ≠x2、‥

図5 定式化

この問題で、１つの Queen を図６の位置に配置した場合の制約伝播を考えます。

1) **前方検査法**:「一つの変数の値が決まったら、他の変数の無効な値を削除する。」

現在の配置により、不可能となった他の Queen の位置を削除する（図７参照)。

	1	2	3	4
1	**Q**			
2				
3				
4				

図6 Queen の配置

	1	2	3	4
1	**Q**			
2	X	X		
3	X		X	
4	X			X

図7 前方検査

2) **アーク整合性法**:「任意の2変数の組に対して、可能性のない値を削除する。」

図８の２行目の A の位置に Queen を置くと、３行目の Queen は置く場所がないので、A は削除。４行目の B の位置に Queen を置くと、３行目の Queen は置く場所がないので、B は削除。この結果、図９の状態となります。この段階で、４行目の C の位置は、２行目の最後の場所（４列目）と競合してしまうため、置くことが出来ません。この結果、C の位置が削除されるので、４行目の Queen の配置場所が無くなり、Fail します。

この結果より、１行目の Queen を１列目に置くことが Fail して、バックトラックにより、次の探索位置は、１行目の２列目になります。これにより、X2、X3、X4 の探索が省かれます。

	1	2	3	4
1	**Q**			
2	X	X	A	
3	X		X	
4	X		B	X

図8 位置の削除

	1	2	3	4
1	**Q**			
2	X	X	X	
3	X		X	
4	X	C	X	X

図9 位置の削除

制約伝播アルゴリズムとしては、さらにレベルの高いもの（Path-整合性、ｋ－整合性等）がありますが、計算時間の面で、現実の問題への適用はあまり役に立ちません。

後で述べますが、制約伝播が効率的に行える制約条件－全ての変数が異なる値を持つという制約、同一の値を取る変数の数の上限・下限に対する制約、容量制約、・・・－が発見されていて、これらを用いて制約伝播を強化する方が現実的です。

3.3 制約充足の基本的な構成

制約充足問題の基本的な構成を図１０に示します。制約伝播の結果、制約違反がある場合には、或る変数の領域が空になる（取るべき値が無くなる）ことにより検出されます。

3.4 変数の選択順序と値の選択順序

図１０で、変数を選択する順序が非常に探索効率に影響します。 次の指針が有向です。

(1) 最も探索空間が縮小する変数を選択
その時点で、最も候補の数の少ない変数
(2) 最も探索空間が縮小しない値を選択
(3) 探索の初期段階の選択が重要

実際に問題を解く場合に、始めの段階が悪いと探索木が深いところでいくらバックトラックしても、始めの方の選択に対するバックトラックが起こりません。このためにリスタートを複数回行い変数の領域削減効果を測定して、変数を決める方式(インパクトと呼ばれる)も提案されています。

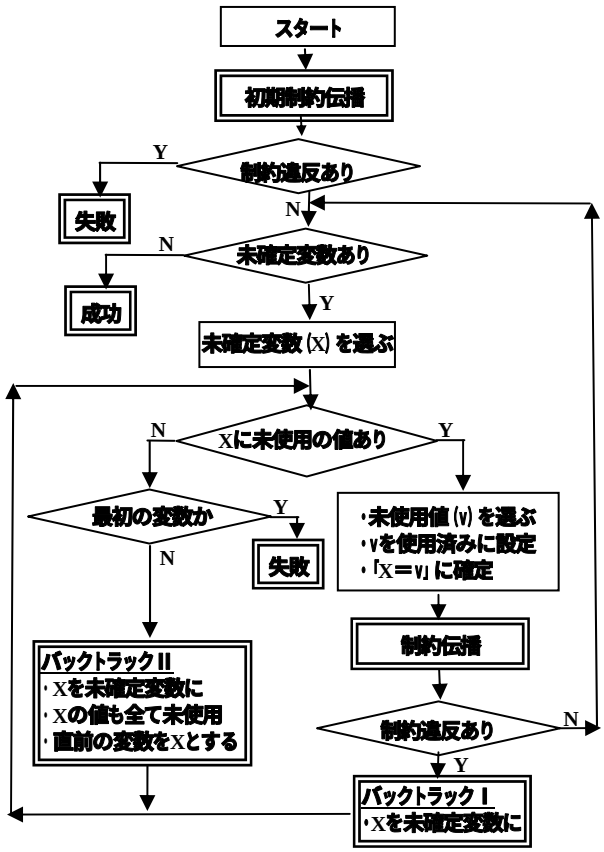

図10　制約伝播を利用した探索方式

4　制約プログラミング

4.1　論理プログラムミングの性格

CP は、元々は、論理プログラミングから発達したため、論理的な制約を書くことが可能です。

例えば、勤務作成で、X を勤務の配列とします。

「連続する７日の間に、必ず休暇を与えなければならない。」は、次のように表現可能です。

$$\sum_{k=0,6} (X[t+k] == \text{休暇}) \geqq 1$$

また、CP は、多重プロセスの考えで出来ており、次のような動的な制約も可能です。

「ｔ日目の作業が夜勤なら、ｔ＋１日目の勤務は明けである。」は、次のように表現可能です。

IfThen(X [t] == 夜勤, X[t+1] == 明け)

4.2　動的な制約の追加

この様に複雑な論理的制約条件を表現できますが、さらに利用者が、制約の伝播方式を定義することにより、新たな制約を定義できます。図１１は、X が Y より小さいという制約の表現です。

これは、現実問題の複雑な制約の論理表現を行ったのでは、制約伝播が困難な場合やヒューリスティックスで解空間を絞り込む時に不可欠です。

```
新しい制約条件 Less:: 意味は、 X ＜ Y
  整合性の条件： max(X) ＜ max(Y)、 min(X) ＜ min(Y)
  制約伝播用関数の定義
  ReduceX() { X.setMax(Y.getMax() － 1);}
  ReduceY() { Y.setMin(X.getMin() ＋ 1);}
実行時の制約伝播条件: 変数Ｘの値の集合の変化条件を登録:
   X.whenDomain(ReduceY):   値が一つでも欠けたら
   X.whenRange(ReduceY):    上限か下限が変化したら
   X.whenValue(ReduceY):    値が確定したら
使用法:  Less(U,V)
```

図11　利用者による制約の定義

4.3　大域的制約

CP では、制約伝播は図３に示したように、個々の制約に対する制約伝播として行われますが、変数全体を対象とした制約伝播のアルゴリズムが開発されています。これを**大域的制約**と呼びます。

例えば、「X1、X2、X3 が異なる値を取る。」という制約は、次のように表現されますが、個別の制約では、解が無いことを検出できません。

```
状況:   X1 ∈{0, 1}  X2 ∈{0, 1}  X3 ∈{0, 1}
制約条件： X1 ≠X2  X1 ≠X3  X2 ≠X3 ＝＞不整合は、未検知
大域的制約: allDifferent( X1,X2,X3 ) ＝＞ 不整合を検知
```

図12　大域的制約の有効性

しかし AllDifferent と呼ばれる大域的制約では検出されます。このアルゴリズムは、変数群と値群をマッチング問題に定式化することにより制約伝播を行っています。

大域的制約は、他にも色々開発されていますが、実際のシステム開発では、これらを制約表現として取り込めるか否かが、成否の鍵となります。

5　スケジューリング問題への適用

5.1　スケジューリングの表現

スケジューリング問題の簡単な例として、以下の問題を考えます。

1）**作業**:図１３に示す A1～B3 の６個の作業がある。括弧内の数字は、作業時間です。

2）**資源**：同時には１作業にのみ利用可能な機械が２種類(R1、R2)あります。

3）**時間制約**：全体の終了時刻(makespan)の設定

4）**順序制約**：作業間には、矢印で示す順序関係があります。

５）**リソース制約**：各作業に必要な機械は矢印で示されています。

６） **目的**：全体の終了時刻(makespan)の最小化

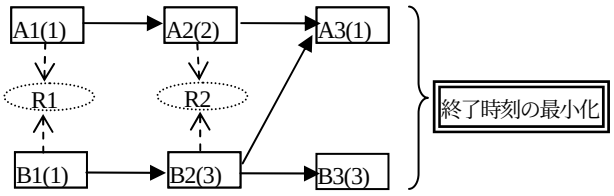

図 13 簡単なスケジューリング問題

この問題を ILOG のパッケージに似せた表現をすると、次のようになります。オブジェクト指向で開発されているため、実際の問題と非常に近い形で表現できているのが分かると思います。

```
制約プログラミング(ILOG 風)による表現
 (1)  Activity の定義:作業の定義で、括弧内は作業時間
      Activity A1(1),A2(2),A3(1),B1(1),B2(3),B3(3);
 (2)  Resource の定義:資源の定義で、括弧内は同時に利用可能な数
      Resource R1(1),R2(1);
 (3)  時間制約の定義:
      変数の定義:終了時刻用の変数(makespan)の導入
      IntVar makespan;
      終了時刻の制約定義:
      A3.endsBefore(makespan);
      B3.endsBefore(makespan);
 (4)  順序制約の定義:
      A2.startsAfterEnd(A1); A3.startsAfterEnd(A2);
      B2.startsAfterEnd(B1); B3.startsAfterEnd(B2);
      A3.startsAfterEnd(B2);
 (5)  リソース制約の定義:
      A1.requires(R1); B1.requires(R1);
      A2.requires(R2); B2.requires(R2);
 (6)  目的関数の定義:
      Minimize(makespan);
```

図 14 制約プログラミングによる表現

5.2 解の探索

ILOG の Scheduler では、容量１の資源は、Unary-Resource と呼ばれています。この場合には、同一のリソースを利用する作業は、全順序が付きますので、これを決めるアルゴリズムがあります。

これは、各作業の順序関係、開始時刻の制約、終了時刻の制約等に基づいて、最早開始時刻や最遅終了時刻を計算して、制約伝播としては、必然的な作業順序を決めるアルゴリズム（Edge Finder と呼ぶ）をベースに、作業の順序を決定します。

また、容量が２より大きい場合に対しては、各作業に対して、必要な資源量×処理時間を、最早開始時刻～最遅終了時刻の間に分配して、任意の時刻に対して、その時刻に資源が分配されている作業の分配量を積算して、最も容量の面で厳しいと思われる時刻を抽出し、その時刻で競合する作業の順序を決定する（Texture Measurement と呼ぶ）アルゴリズムが提供されています。

解の探索は、次のステップで行います。

```
(7)  探索法の指定:UnaryResource の効率的な利用順序決定をベースとした探索法(Ranking)を選択
(8)  探索の実行:
     Search( Ranking( makespan ) );
```

図 15 制約プログラミングによる探索

この探索の結果、以下に示す makespan が９の最適解が得られます。

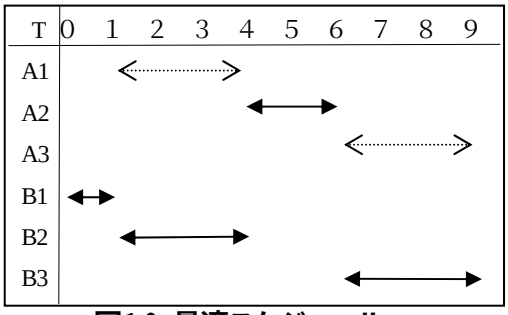

図16 最適スケジュール

この解では、A1 と B1、及び、A2 と B2 は、資源の競合のため、作業時間帯のオーバーラップが回避されています。また、B1=>B2=>A2=>B3 がクリティカルパスとなっており、makespna＝９は、最適解である事が判ります。

制約プログラミングでは、ユーザは制約の定義を行い、求解は、システムの探索エンジンに依頼する形態となり、基本的には、探索ロジックは書かないで良いことを理解していただけたと思います。

6 実用問題への適用

私が開発に参加した２システムについて、モデル化の基本的な考え方を紹介します。

6.1 グループ循環型の交代勤務計画

日勤と夜勤の交代勤務があり、各勤務とも４名要員（資格者１名を含む）で実施している。これに対して、「日勤-夜勤-明け-休暇」というサイクルをベースとした４グループと予備人員グループで循環型の割当を実施しているが、このサイクルだと休暇が７～８回／月となり不足するし、休暇の希望も有るため、サイクルを修正する必要がある。

1）勤務者全員（縦軸）を対象にして、1 日分の勤務を割り当てるが、配分制約を用いて、必要な人員の大域的制約を表現している。

2）各作業者（横軸）には、休暇希望日の外に、一定期間内の公休の付与、月間の休暇数の確保、夜勤数の上限の設定等があり、これは 4.1 節で示した方法で対応している。

この問題の最も困難な点は、如何にしてグループの勤務形態を確保するかであるが、これを探索するためのアルゴリズムを開発している。

図 17 勤務計画の作成結果
(鉄道情報システム株式会社提供)

6.2 鉄道運転整理システムの開発

鉄道では、事故等で列車に遅延が発生すると、その遅延を回復するために、列車の運行順序や特急と普通の追越駅の変更で対応しますが、続行する列車間の安全確保、駅のホームの使用順序、駅間の線路容量、列車の折返し、列車の分割併合等の制約条件を満たす必要があるため、非常に困難な問題です。

従来は、時刻を少しずつ移動させながらシミュレーションして、回復ダイヤを作成する方式が採用されていますが、これには本質的な欠陥があります。図１８に示すように、列車 T2 が B 駅を出発できる条件は、未来の時点で A 駅に着くときにホームが空いていること、即ち先行列車 T1 が A 駅を出発していること、さらに先行列車 T1 が A 駅を出発するためには、列車 T3 が・・・と、T2 列車の発時刻は未来側の条件が決まらないと定まらないという循環に陥ります。

この問題に対し、次のモデル化で解決しました。

1）駅の出発列車の順序表（SOT）と列車の駅での出発順位表（POT）を使用して、回復ダイヤを表現する。

2）列車間の制約は、全て発着時刻間の不等式で表現する。

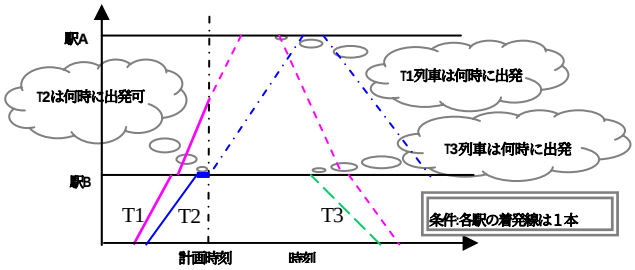

図 18 シミュレーション方式の欠陥

制約1：SOT と POT は逆関数の関係 SOT[POT[k]]＝k
　大域的制約：IloInverse で表現
制約2：駅の続行列車の時隔制約
　発時刻[SOT[k]]＋時隔 ≦ 発時刻[SOT[k＋1]]
制約3：列車kは、列車jより後に発車する。
　POT[j] ＋1 ≦ POT[k]

図 19 ダイヤモデルの基本制約型

このモデル化により、全列車の発着時刻が不等式系で表されており、列車の SOT が決まると全ての不等式が確定します。この不等式系の最小時刻を運転時刻として採用すると、安全基準や資源の使用制約を満たした回復ダイヤが得られます。

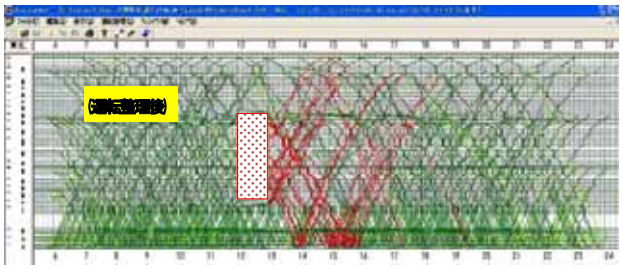

図 20 運転整理の実施結果

（ 共同研究：(株) JR 東日本情報システム、(株)ニューメディア総研、(株)数理モデリング研究所 ）

7 おわりに

制約プログラミングは、理論と同時に、現実問題に対するモデル化のセンスが重要です。ILOG には、多くの機能があり、サンプル プログラムで制約プログラミングのセンスを磨けます。

この分野に多くの方々が参加されて、わが国の計画技術が向上することを期待しています。

参考文献

[1] ILOG Solver User’s Manual 6.1, ILOG

[2] ILOG Scheduler User’s Manual 6.1, ILOG

[3] K. R. Apt, Principles of Constraint Programinng, Cambridge Univ. Press（2003）

[4] F. Benhamou, et al(Ed.), Trends in Constraint Programming, ISTE（2007）