卒業研究
プランニング問題のドメイン非依存なモデリング手法: 近年のソルバ高速化手法を考慮して

氏名：堀江 慧
指導教官：福永 Alex
学生証番号：08-132021
所属：教養学部 学際科学科 総合情報学コース

2015 年 1 月 4 日

目次

表目次

1 序論

1.1 研究の概要

本研究で扱うプランニング問題は，現実の意思決定をモデル化した問題であり，人工知能分野の主要な問題のうちの一つとされ，かつてから広く研究の対象として扱われているものの一つである。この問題が解決された暁には，現時点の情報を入力として，目的を達成するまでのアクション列を自動で得ることができるようになる。

直接的な応用としてはロボットの自律的な行動決定が想像されるが，行為者は必ずしも物理的な実態を持つものに限らない。例えばゲームのプレイヤーは現状に対して行為の意思決定を求められるが物理的な実態は必ずしも必要としない。このように物理的な実態を持たない行為者のことアバターといい，ロボットとアバターとを合わせた抽象的な行為者をエージェントという。

人工知能の備える’知能’の性質にふさわしいものとして汎用性が挙げられる。与えられた問題に対して，ドメイン特化の知識を人間が与えれば，よりよいパフォーマンスを挙げることができることができるであろう。(Kautz and Selman98 [8] など) 一方で，そのような人間の’知能’を介助を得ること無くその問題解決を達成するような人工知能は，より望ましいものであるといえる。そして，そのような知能の実現は当然より難しいものとなる。

本研究では汎用モデルとそのソルバーを利用し，プランニング問題を解くドメイン非依存な手法について取り扱う。また，そのようなモデルとして充足可能性問題, 数理計画問題, 制約問題を扱う。

ここで言うドメイン非依存性とは，プランニング問題の枠組みで表現できる問題であればどのような問題インスタンスに対しても同様の処理で解を得ることが出来る性質をいう。プランニング問題自身の柔軟性，実用性からこのドメイン非依存性は有用な性質である。ドメイン特化の知識を利用して解を得る手法の場合，プランニング問題のドメインごとにその処理を変更する必要がある。一方，ドメイン非依存な手法を用いてプランニング問題の解を得ることができれば，それは一般のプランニング問題について解を得ることと同値である。

続いて，制約プログラミングのモデルにおける global constraint に着目する。global constraint とは制約プログラミングの分野で近年特に研究の進んでいる特定の制約群である。

これらは，解を求めるために利用できるような良い性質を持っていること，特定の制約ではあるが，ある程度の汎用性のある制約であること，をその性質として有している。

そしてこれらの制約に対しては，ソルバ内の処理として，より効率的な探索アルゴリズムが開発されているものも多くある。これらの制約をうまく取り込んだプランニング問題の定式化を行えば，素朴な制約のみから作成したモデルよりも効率的に解を得ることができる可能性があると考えられる。また，表現力豊かな制約を利用することで，モデリング自体の困難さを緩和し，等価なモデルをよりシンプルに表現することができるであろうと考えられる。

まとめると，本研究では制約プログラミングのモデルでプランニング問題を扱う際に，global constraint を

利用した有効な定式化な手法について述べる。

提案手法の有効性の検証として，充足可能性モデルの SATPLAN2006 [7]，数理計画モデルの optiplan [13] プランナや，制約モデルの GP-CSP [5] といったプランナに対して，global constraint を利用した本研究の手法による定式化を追加したプランナとの比較を行う。

1.2 背景

正確な定義は後に述べるが，プランニング問題とは例えば以下のような問に応える問題である。エアコンの電源を入れるにはどうすればよいか，部屋を綺麗にするにはどうすればよいか，洞窟の中から脱出するにはどうしたらよいか... また，スライディングタイルやハノイの塔といったパズルの類もこの枠組で表現することができる。

このように，この問題は現実の行動計画，意思決定の直接的な定式化である一方で，また，多くの現実的な問いをこの枠組みで表現することができる。

迷路の問題を元に考えてみる。迷路内で行える行為として，右に向き換えること，左に向きを換えること，向いている方向に直進すること，の３つの行為が可能であるものとする。この時に，迷路内のある地点にあるゴールにたどり着くようなプランを考える。ゴールに辿り着けるプランが存在するのであれば，実行可能な行為を全て列挙してしまえばゴールに辿り着くようなプランが得られるであろう。このように，初期状態から行為を実行していき，各行為の効果を書く時点での状態に反映させていくことを繰り返せば，探索によってプランを得ることができる。一方で，その探索空間の大きさはプランの深さに対して指数的に増加していくこととなる。

このように，この問題は一般には解の長さに対して指数的に探索空間の大きさが増えていき，最悪の場合，有限多項式時間で解を求めることはできない。一方で，充足可能性問題, 数理計画問題, 制約プログラミングなどの問題も汎用な枠組み，困難な問題であることが示されていながらも，近年では比較的大規模な実用的問題でも利用されるようになってきている。このことを背景に，上に挙げた汎用モデルでプランニング問題を表現し，各モデルのソルバでプランを得るアプローチが研究されている。

先行研究としては，充足可能性問題を利用したものとして SATPLAN, 数理計画のものとして optiplan, 制約プログラミングのものとして GP-CSP などが開発されており，現在も引き続きより高速な手法が考案されて続けている。

モデルによるプランニング問題のアプローチの近年の成果として，Graphplan で初めて実装されたプランニンググラフを利用する方法がある。この手法では，与えられたプランニング問題からまずプランニンググラフを生成する。プランニンググラフは完全性を損なわない範囲で比較的大きく解空間を縮めることのできる問題の表現であり，このプランニンググラフ上の縮小された探索空間から解を発見する。プランニンググラフの構成と探索空間の縮小自体は，実際の探索と比べて小さいオーダーの実行時間で行うことが出来るのがポイントである。

この手法は充足可能性モデルで実装された後，その有効性が認められ，数理計画モデル，制約モデルにも持ち込まれている。

1.3 研究の目的

本研究はプランニング問題についてモデル表現とソルバを利用した方法によるアプローチについての研究である。プランニング問題は PSPACE 完全であることが知られている。(Bylanedr91 [3]) この問題の解を得るためのアプローチとしてモデルを使ったアプローチが研究されている。このアプローチではソルバの性能向上がプランニングシステムの性能に直結し，各モデルのソルバは日々進展している。しかしながら，近年の研究はモデルの中でも充足可能性モデルに関するものが多くを占めている。

この状況に対し，本研究では制約モデル，数理計画モデルでの近年のソルバの性能向上がそれぞれのプランニングシステムの性能向上にどのくらい影響を及ぼしているのかを再調査する必要があると感じた。

また，特に制約モデルの近年の性能向上に大きな影響を与えている global constraint について，それを効果的に織り込んだプランニングの定式化を示すことを目的としている。

1.4 本論文の構成

2 章では，まず，プランニング問題とはどのようなものかを正確に表現するための用語の定義を行う。続いてプランニング問題を形式的に表現し，これを書き表すための形式的な言語 PDDL の説明を例を交えながら行う。その後，各モデルでプランニング問題の表現がどのようなものとなるかを考察し，最も素朴な制約充足問題での表現を具体例を交えながら追う。

3 章では，充足可能性問題，数理計画問題，制約問題がどのようなものであるかをそれぞれ説明する。

4 章では，プランニング問題を先に述べた 3 つのモデルでそれぞれ表現することができることを示す。また，それぞれのモデルの表現力の包含関係についても説明を行う。続いて，プランニング問題の素朴な表現での無駄を省略するためのプランニンググラフを利用した探索空間の縮小について説明する。その後，本研究で扱った optiplan で用いられているプランニング問題の数理計画による表現のためのモデリングである state change model について説明する。

5 章では，SATPLAN2006 プランナと optiplan プランナの IPC ベンチマーク問題集による比較実験について述べる。まずはそれぞれのプランナの特徴について説明した後，実験データを掲載する。

6 章では，制約モデルでのプランナの実装について述べる。本研究の実装で利用した Gecode について説明した後，プランニング問題の定式化において有効な global constraint について述べる。

7 章では，6 章で述べた global constraint を利用して実装した制約モデルのプランナと，SATPLAN2006, GP-CSP, optiplan との比較実験について述べる。

2 プランニング問題

プランニング問題の設定としては，マルチエージェント環境やリアルタイム性を考慮したものなど多くのものが挙げられているが，本研究では最も広く扱われている古典的プランニング問題を扱う。

2.1 古典的プランニング問題の定義

以下，プランニング問題の古典的な表現である古典的プランニング問題の定式化についての説明を行う。古典的プランニング問題とは’ 十分に観察可能で，決定論的で，有限で，静的（エージェント自身が行為を実

行した場合にのみ変化する）で，かつ（時間，行為，対象，効果が）離散的な環境のみを扱う’問題である。(エージェントアプローチ人工知能第 2 版 381 ページ [12]) 十分に観察可能であるとは，プランニング問題の中で扱う命題に関してはその真偽が定まることである。決定的とはある状態に置いて行為を行うと至る状態が一意に定まることである。静的であるとはエージェントの行為の効果のみが環境を構成する命題の値を変化させることを述べる。離散的であるとは，扱うオブジェクトが非連続的であることを述べる。有限であるとは，命題，行為などのプランニングに関連するオブジェクトか高々有限個しか存在しないことを述べる。

また，ここで述べるもの以外にも，プランニングの表現としては多くのものが提案されていることを述べておく。

まず，世界のうちの個別の事象の状態を表現するような種々の命題というものが存在するとする。命題 p に対して，その否定を $\neg p$ と表現する。命題とその否定を合わせてリテラルと呼ぶ。また，前者を正のリテラル，後者を負のリテラルと呼ぶこともある。

p と $\neg p$ はある命題の成否と対応するリテラルで，その真偽値は同時には真にならない。

また，$p \vee \neg p$ であるものとする。(閉世界仮説)

ここで，現実の世界のうちどの事象を考慮の対象とすることで問題を十分に表現しうるか，という問題はフレーミングの問題として知られており，人工知能分野の古くから知られている難問のうちの一つで有ることを注意しておく。しかしながら，今，問題を定義するための命題の集合

を定めたとすると，この命題集合の上でプランニング問題を定義することができる。

プランニング問題は以下の 4 つ組で定義される。

$$\{P, I, G, A\}$$

- $P = \{p_i \mid i = 0, \cdots, n\}$ は命題集合
- $I(\subseteq P)$ は初期状態
- $G(\subseteq P)$ は目的状態
- $A(\subseteq P \times P \times P)$ は行為集合

ここで，世界の状態 S とは，

$$S = \{(p_i, b) \mid \forall p_i \in P, b \in \{true, false\}\}$$

であり，命題のすべての要素への値の割り当てを表現しているものとする。

S の要素のうち (p, b) を (p, b') に置き換える操作を $S[p] \setminus b'$ と表すこととする。また，S の要素 (p, b) が存在するとき，$S[p] = b$ と定める。

行為は以下のように定義される。

$$a = (Pre, Add, Del)(a \in A, Pre \subseteq S, Add \subseteq S, Del \subseteq S)$$

Pre は前提条件であり，この行為が実行される際にその真偽値が真となっていることを必要とする命題の集合である。一方，Add と Del は行為の効果であり，行為の実行後に真となっているべき命題，偽となっているべき命題の集合である。Add を正の効果，Del を負の効果という。

S に対して実行可能な行為 α とは，α の前提条件集合 Pre に対して $Pre \in S$ が，成立するようなもののことである。

この行為を実行すると，世界の新たな状態 S' が得られる。

$$S' = \{(p_i, b') \mid b = \begin{cases} true & p_i \in Add \text{ のとき} \\ false & p_i \in Del \text{ のとき} \\ S[p_i] & \text{それ以外のとき} \end{cases} \}$$

すなわち S' においては，Add に入っている命題はその真偽値が真であり，Del に入っている命題はその真偽値が偽であり，いずれにも入っていない命題は S の真偽値を保存する。

ここで，Add と Del には同じリテラルは含まれないことを仮定していることに注意すると，S の要素の書き換えはどの順序で行っても結果が変わらないことに注意する。また，この仮定は Add と Del の意味を考えると適当なものである。

これを状態 S に行為 α を適応するという意味で $S' = \alpha(S)$ と書くこととする。

また，初期状態 I は全ての命題への真偽値の割り当てを表現する。ここでは閉世界仮説を採用していることから，初期状態で正の値を割り当てられる命題の集合を S とすれば偽が割り当てられる命題は明らかである。

$$I \subseteq S$$

すなわち，開始時点の状態 S_0 では I の要素となっている命題には真が，入っていない要素には偽が割り当てられていることを要求する。

$$\forall_{(p,b) \in S_0} b = \begin{cases} true & p \in I \text{ のとき} \\ false & \text{それ以外のとき} \end{cases}$$

続いて，目的状態 G はプランニング問題のゴール，最終的に要求される状態である。

$$G \subseteq S$$

ここで，初期状態では集合 I に入っていない要素には偽が割り当てられることを要求していたが，目的状態 S_T では集合 G に入っていない状態に関してはなんの要求もしないことに注意する。すなわち，初期状態は状態集合の値の割り当てを一意に特定するが，目的状態では興味のある命題にのみ値の割り当てを要求する。

$$\forall_{(p,b) \in S_T \wedge p \in G} b = true$$

このことから，初期状態は I から一意に定まるが，目的状態に関しては複数の状態がそれを満たしうることがある。

プランニング問題の解，すなわちプランとは，初期状態 S_0 に順次適応することで目的状態 S_T へと至るような行為の列である。ここで，目的状態に至ったかどうかの判定はその時点での状態集合が目的状態を全て含んでいるかどうかで行われることに注意する。

すなわち，

$$\begin{cases} S_T = \alpha_n(\alpha_{n-1}(\cdots \alpha_1(\alpha_0(S_0)))) \\ S_T \supseteq G \end{cases}$$

の時に

$$(\alpha_0, \alpha_1, \cdots, \alpha_{n-1}, \alpha_n)$$

がプランである。

2.1.1 プランニング問題の例

プランニング問題の例として以下の素朴な問題を考えてみる。

２つの部屋 A,B が存在する。現在，持ち手を２つ備えたロボットが部屋 A にいるものとする。ロボットに可能な行為は以下の３つである。

- 現在の部屋から他の部屋に移動すること
- 空いている持ち手でロボットのいる部屋にあるボールを拾うこと
- 持ち手にあるボールをロボットのいる部屋にボールを落とすこと

ボールが部屋 A にあり，ロボットが部屋 B にいるものとするとき，ボールを部屋 B に移動させるようなプランを求める。

2.2 PDDL

プランニング問題を記述するための標準的な形式として Planning Domain Definition Language(PDDL) という形式が広く利用されている。

PDDL を構成するファイルには問題ドメインについての共通の情報を扱うドメインファイルと個別の問題ファイルがある。

ドメインファイルにはドメイン名，述語の定義，行為の定義などが含まれる。行為の定義では各行為の前提条件と効果が述べられる。

問題ファイルではオブジェクトの一覧と，初期状態，終了状態が含まれている。

また，プランニング問題の解も PDDL 形式で出力されることがある。これは出力されたプランを並べたものである。

2.2.1 PDDL の例

プランニングの例で述べた gripper ドメインの問題を表現する PDDL の例を挙げる。以下の PDDL ファイルは fastdownward 付属の benchmarks の gripper ドメインの domain.pddl, prob01.pddl を元に著者がボールの個数を減らし，コメントを付与したものである。

まず，ドメインファイルは以下のようになる。(なお; 以降はコメントである。)

```
;gripper-strips のドメインファイルを定義
(define (domain gripper-strips)
   ; 述語の定義
   (:predicates (room ?r) ;r は部屋であるか
(ball ?b)          ;b はボールであるか
(gripper ?g)       ;g は持ち手であるか
(at-robby ?r)      ;r にロボットがいるか
(at ?b ?r)         ;b が r にあるか
(free ?g)          ;g が空いているか
(carry ?o ?g))     ;g に o を持っているか
```

; 行為 move の定義
```
(:action move           ;from から to に移動する
    :parameters  (?from ?to)
    :precondition (and  (room ?from) (room ?to) (at-robby ?from))
                        ; 前提条件は
                        ; from,to が部屋であり,from にロボットがいること
    :effect (and  (at-robby ?to)
  (not (at-robby ?from))))
                        ; 効果は
                        ; ロボットが to にいるようになること（正の効果）
                        ; ロボットが from にいないようになること（負の効果）
```

; 行為 pick の定義
```
(:action pick           ;obj を room で gripper が拾う
    :parameters (?obj ?room ?gripper)
    :precondition  (and  (ball ?obj) (room ?room) (gripper ?gripper)
 (at ?obj ?room) (at-robby ?room) (free ?gripper))
            ; 前提条件は
            ; obj がボールであり,room が部屋であり,
            ; gripper が持ち手であり,obj が room にあり,
            ; ロボットが room にいて,gripper が空いていること
    :effect (and (carry ?obj ?gripper)
 (not (at ?obj ?room))
 (not (free ?gripper))))
                ; 効果は
                ; gripper に obj を持っているようになること（正の効果）
                ; obj が room にあるようになること（負の効果）
                ; gripper が空いているようになること（負の効果）
```

; 行為 drop の定義
```
(:action drop           ;obj を room で gripper から落とす
    :parameters  (?obj  ?room ?gripper)
    :precondition  (and  (ball ?obj) (room ?room) (gripper ?gripper)
 (carry ?obj ?gripper) (at-robby ?room))
            ; 前提条件は
            ; obj がボールであること,room が部屋であること
            ; gripper が持ち手であること,gripper に obj を持っていること
            ; room にロボットがいること
    :effect (and (at ?obj ?room)
```

```
    (free ?gripper)
    (not (carry ?obj ?gripper))))
                ; 効果は
                ; obj が room にあるようになること（正の効果）
                ; gripper が空いているようになること（正の効果）
                ; gripper に obj を持っていないようになること（負の効果）
)
```

問題ファイルは以下のようになる。

```
;strips-gripper の問題ファイルを定義
(define (problem strips-gripper-x-1)
   (:domain gripper-strips)
   (:objects rooma roomb ball left right)
                        ; オブジェクトとして rooma,roomb,ball,left,right が存在する
   (:init (room rooma)
          (room roomb)
          (ball ball)
          (at-robby roomb)
          (free left)
          (free right)
          (at ball rooma)
          (gripper left)
          (gripper right))
                        ; 初期状態は
                        ; rooma,roomb は部屋である,ball はボールである
                        ; ロボットは rooma にいる
                        ; left,right は空いている
                        ; ball は rooma にある
                        ; left,right は持ち手である
   (:goal (and (at ball roomb)))
                        ; 目的状態は
                        ; ball が roomb にあるようになることである
)
```

3 モデル

以下，上述のように定義されたプランニング問題を他のモデルに変換することを考えていく。そのため，本研究で扱った他のモデルである充足可能性モデル，数理計画モデル，制約モデルの３つをまずは正確に定義する。

3.1 充足可能性問題

充足可能性とは，命題論理式が与えられた時に，その論理式を充足する，すなわち値を真とするような命題変数への値が存在するかどうかを判定する問題である。

論理式は命題のリテラルと論理和，論理積の論理演算子から作られる。また，リテラルの論理和を節という。

定義には $\Rightarrow$ 論理演算子を含まなかったが，$p \Rightarrow q = \neg p \vee q$ でありモデルの表現力にはこの演算子を含んでも変化がないことに注意する。

3.2 数理計画問題

数理計画問題は目的関数，変数集合，不等式制約集合から構成される。目的関数は変数集合内の変数の任意の関数である。不等式制約集合は変数集合内の任意の変数の一次結合を左辺，右辺を定数とする不等式の集合である。一般の数理計画問題では不等式制約として任意の不等式を許すこともあるが，高速に解を求める事ができる線形不等式制約が広く使われる。この研究でも，不等式制約としては線形不等式のみを認めた。

数理計画問題の全ての変数に対して，全ての不等式制約を満たすような値の割り当てを数理計画問題の実行可能解という。実行可能解の集合を実行可能領域という。数理計画問題を解くとは，実行可能領域内に含まれる変数の割り当てのうち，目的関数を最大化するような変数の割り当てを得ることをいう。

ここで，目的関数 $f(x)$ の最大化は $-f(x)$ の最小化と同値であること，任意の等式制約を不等式制約に変換できること，最小値，最大値，絶対値といったものを含む不等式の一部を線形不等式に変形できることなどが知られていることを注意しておく。線形計画問題の表現能力とその具体的な変形方法については最適化の数学 [16]，東京農工大学宮代先生の数理計画メモ [15] などにまとまった記述がある。

この変数への割り当てを数理計画問題の最適解といい，この時得られる目的関数の値を最適値という。また，数理計画問題において単に解，といった場合，実行可能解ではなく最適解を表すことが多い。

線数計画問題と充足可能性問題の応用上の大きな違いとして，目的関数の存在がある。前者にはモデルに自然に解の評価値が与えられており，変数の値の任意の関数を目的関数として利用できることから，解の選好を自然に扱うことが出来る。生成されるプランの長さを最小化することは充足可能性問題においても行為に対応する命題の個数を最小化する，という形でモデル内で表現可能だが，実際の応用では資源の最小化，エネルギーの最小化，体積の最小化など様々な目的関数を用いて解の良さを定量化し，これをモデルの中で最適化出来ることは数理計画モデルの長所である。

また，モデルの変換の章でも述べるように，充足可能性問題は一般に数理計画問題に変換が可能である。

3.3 制約問題

制約問題は，変数 $\{x_i\}(i = 0...n)$，制約 C から定義される。ここで全ての変数には定義域が明示的に与えられているものとする。また，制約は全ての変数の関係として与えられる。すなわち，変数 $x_0, x_1, \ldots, x_n$ とするとき，

$$X = x_0 \times x_1 \times \cdots \times x_n$$

に対して

$$C \subseteq X$$

である。

ここで，全ての変数に値を割り当てたもの

$$A = (v_0, v_1, \cdots, v_n)(x_i = v_i \text{という割り当てを表す})$$

もまた $A \subseteq X$ であることから，A が C の要素の部分集合となっているかどうかによってこれを 2 分割することができる。この時，補助関数 $\mathrm{is}\_true(A)$ を導入し，この値を A が C に属するならば真，そうでないならば偽と定める。

$$is\_true(A) = \begin{cases} 1 & A \subseteq C \text{ のとき} \\ 0 & \text{それ以外のとき} \end{cases}$$

制約問題の解とは，真であるような変数への値の割り当てである。このような解を見つける問題を制約充足問題という。

また，制約問題には目的関数を必ずしも含まないが，目的関数を利用することもできる。目的関数を利用し，解のうち目的関数を最大化するものを求める問題も含む，という意味合いで本論文では制約問題という。

制約として，不等式制約，論理制約を共に扱えることから，充足可能性問題，数理計画問題は一般に制約問題に変換が可能である。

4 プランニング問題のモデリング

プランニング問題を先に述べた 3 つのモデルに変換することができる。

制約モデルは，その制約として論理制約，不等式制約を利用することができるので，他の 2 つのモデルで表現することのできる問題を表現することはできる。

また，任意の論理式を論理式として同値な連言標準系に変換できることはよく知られた事実である。(例えば Handbook of satisfiability [1] を参照) ここで，その連言標準系の各節内の変数の真偽値を 1,0 値として持つ変数 x_l を考え，全ての節 L_i に対して

$$\sum_{l \in L_i} x_l \geq 1$$

という制約を加えることで任意の充足可能性モデルの問題は数理計画モデルの中で表現可能である。

以上に述べたことから，3 つのモデルのうち，充足可能性モデルでプランニング問題を表現することが出来たならば，数理計画モデル，制約モデルでもプランニング問題でも表現できることがわかる。

以下ではまず，Kautz と Selman [9] で示されたプランニング問題の制約充足モデルの表現方法について解説する。続いて，素朴な問題エンコーディングに加えて，ソルバがより高速に解を求めるための工夫として大きな効果を発揮するプランニンググラフ [2] について説明する。

さらに，今回の実験で使用した optiplan で採用されているプランニング問題のエンコーディング方法 state change model [14] について説明する。

4.1 制約充足問題によるプランニング問題の表現

プランニング問題を制約充足問題で表現するにあたって，論理命題に状態を対応させる必要がある。論理命題と状態を単純に一対一対応させると時間の表現を含まなくなってしまうため，あるステップにおいてある状態に対応する命題が真であるということを SAT 命題を一つ使って表現する。すなわち，各ステップにおいて命題の真偽値を保持する SAT 命題を定義する。

ここまずで問題となるのは，事前にはプランの存在するステップの深さがわかっていないことである。論理式を構成するには関連する命題論理式を列挙出来る事が必要である。

そのため，まずは最大ステップ数を 1 として論理式を構成し，その論理式からプランが得られるならばそこで終了し，そうでないならば最大ステップ数を増やして論理式を作成し，プランが得られるかを検証，この手順をプランが得られるまで繰り返すこととする。

ステップ t での命題の真偽値と対応する命題の集合を P_t と表すこととする。

また，同様に，あるステップで行為が行われるかどうかに対応する命題も最大ステップ数以下の全てのステップについて定義する。すなわち，ステップ t での行為の実行の有無と対応する命題の集合を A_t と表すこととする。

プランニング問題を SAT に変換するには以下の条件を論理式で表現すれば必要かつ十分である。

- 初期状態
- 終了状態
- 次状態公理
- 前提条件公理
- 排他行為公理

初期状態は $t = 0$ での世界の状態を定める。(ここで p_s_t はプランニング問題の状態 s に対応するステップ t での命題である。すなわち $p\_s\_t \in P_t$ である。)

$$\bigwedge_{s \in I} p\_s\_0 \wedge \bigwedge_{s \notin I} \neg p\_s\_0$$

終了状態 t=T での命題変数が真となっていることを要求する。

$$\bigwedge_{s \in G} p\_s\_T_{max}$$

次状態公理は，ある命題に対して，その命題が真となっているならば，直前のステップでその命題を真とする行為が実行されること，あるいは直前のステップでその命題は真でありかつその命題を偽とするような行為が実行されないことを要求する。(ここで p_a_t はプランニング問題の行為 a に対応するステップ t での命題である。すなわち $p\_a\_t \in P_t$ である。また，ADD_s，DEL_s はそれぞれ正の効果，負の効果に s を含むような行為の集合である。)

$$\forall_{t=0 \cdots T_{max}-1} \quad \bigwedge_{s \in S} p\_s\_t{+}1 \Rightarrow \Big(\bigvee_{a \in ADD_s} p\_a\_t \Big) \vee \Big(p\_s\_t \wedge \bigvee_{a \in DEL_s} \neg p\_a\_t \Big)$$

前提条件公理は，ある行為を実行するならば，その直前で行為が前提条件とする命題の割り当てが満たされていることを要求する。(ここで PRE_a は a の前提条件集合である。)

$$\forall_{t=0 \cdots T_{max}-1} \quad \bigwedge_{a \in A} p\_a\_t{+}1 \Rightarrow \bigwedge_{s \in PRE_a} p\_s\_t$$

排他行為公理は，あるステップではひとつの行為までしか実行されないことを要求する。

$$\forall_{t=0 \cdots T_{max}} \quad \bigwedge_{a1, a2 \in A} (a1 \neq a2 \Rightarrow \neg a1 \vee \neg a2)$$

以上の公理群を表現する命題を節とし，これらの論理積からなる論理式の真偽値はあるステップ数までの長さの直列プランが存在することと一対一対応を持つ。

4.2 プランニンググラフ

上に述べた定式化の問題点として，不要な命題が多く存在していることがあげられる。例えば，初期状態において偽である命題を前提条件として含む行為は t=0 で決して実行されることは無いため不要である。

問題の構造から効率的にこれらのような不要な命題を除くことを可能にする工夫として，プランニンググラフを利用することが挙げられる。プランニンググラフの構成自体の実行時間は多項式オーダーで行うことができ，定式化の完全性を損なわない範囲で多くの不要な命題を除くことができるといった有効性から，この手法は充足可能性モデルにとどまらず，数理計画モデル (optiplan) や制約モデル (GP-CSP) といった他のモデルにも応用され，広く使われている。

プランニンググラフとは

プランニンググラフは命題から余分なものを除くかつ必要なものを除かないことから，プランニンググラフのなかで目的命題が全て活性化されるステップ T_{min} はプランの存在するステップ数の下限となる。この性質を利用すると，プランニングの定式化は $T = T_{min}$ から論理式の構成を開始することができる。

ここで，プランニンググラフによって活性化済みの命題の数は単調に増加すること，また，元のプランニング問題の命題の数は有限であることから，プランニンググラフの展開は有限回でレベルオフすることがわかる。一方，レベルオフして，かつゴール命題の全てが活性化されている場合にはその時点から有限回の展開によって解が得られることが示されている。（要検証）

4.3 state change model

本研究で扱ったプランナ optiplan では state change model という定式化を利用してプランニング問題を数理計画にエンコードしている。

state change model は Vossen99? [14] で提案された定式化で，

state change model ではプランニング問題の命題を直接的に論理変数とせず，それらの状態遷移に対応する変数を使って間接的に命題を表現している。

state change model のうえで初期状態，終了状態，次状態公理，前提条件経理，無矛盾制約を満たす制約を記述すると以下のようになる。

5 実験: optiplan のモデルの再現実装と SATPLAN2006, GP-CSP との性能比較

国際会議 ICAPS と共に行われるプランニングシステムのコンテスト International Planning Competition(以下 IPC とする) の過去のベンチマーク問題集 (fastdownward システムに付属しているもの) のうち，本研究で実装したプランナの対象とする問題である全てのドメイン (表 1 に一覧した) に対して本研究で実装したプランナと比較対象のプランナ群 (表 2) を実行した。これらの問題集合はプランニング問題の標準的な形式である Planning Problem Definition Langage(PDDL) にしたがって記述されたドメインファイルと問題ファイルの組で与えられる。

プランナはそれらのファイルを入力とし，実行時間の制限 n 分以内に実行可能なプランを出力出来たものをその問題を解けた，とみなし，その実行時間を記録した。

表 1　実験に使用したドメイン一覧

大会	ドメイン名
ipc98	gripper, logistics98, blocks, grid, mprime, mystery, movie
ipc00	miconic
ipc02	depot, driverslog, zenotravel, satellite, rovers, freecell
ipc04	airport, psr-small, pipesworld-tankage
ipc06	tpp, storage, pathways, trucks-strips
ipc08	elevators-sat08-strips, elevators-opt08-strips, parcprinter-08-strips

表 2　実験に使用したプランナ一覧

プランナ	詳細
SATPLAN2006	http://www.cs.rochester.edu/ kautz/satplan/のもの
optiplan	gurobi での再実装，1 コア，プランニンググラフなし
optiplan	gurobi での再実装，1 コア，プランニンググラフあり
optiplan	gurobi での再実装，4 コア，プランニンググラフなし
optiplan	gurobi での再実装，4 コア，プランニンググラフあり
GP-CSP	http://rakaposhi.eas.asu.edu/gp-csp.html のもの

実行時間の計測は入力ファイルからモデルを生成する時間と，生成したモデルを元にソルバが求解を行う時間を分けて計測した。また，出力されたプランの正しさの検証には VAL [6] というツールを利用した。

また，実行時に生成される各ソルバの入力サイズを (algorithm) で圧縮したもののファイルサイズを比較した。この比較から，問題を各モデルに変換したソルバ入力ファイルの情報量の大きさを近似的に見積もることかできるであろう。

gurobi のバージョン 5.6.3, SATPLAN2006 の linux binary, Gecode のバージョン 4.3.2 を利用し，実験はなんちゃらの ubuntu14.04 上で行った。

5.1　optiplan

optiplan プランナは Kautz と Selman [13] によって実現されたプランナであり，数理計画モデル，ソルバによって実装されているプランナにして初めて International Planning Competition に参加したプランナである。

本研究では，近年の数理計画ソルバの進歩による効果実証を行うため，optiplan 論文にしたがってそのモデルを再実装した。その際に，optiplan 現論文では商用ソルバ CPLEX(version.) [4] を利用した実装となっていたが，本研究の実装では近年のベンチマーク [10] で好成績を上げている商用ソルバ Gurobi Optimizer(version.) [11] を利用して実装を行った。また，実装言語としては C++ を利用した。

さらに，当時の環境と比べ，ソルバの並列実行に対する対応が進んでいることが考えられるため，単一コア，複数コアでの計測を行うこととした。

optiplan の定式化の特徴としてはプランニンググラフしていることが挙げられる。また，state change model として Vossen の研究で述べられた定式化を工夫して IPC のドメインで繰り返し現れる状態遷移を直接的に表現できるよう改良されたモデルを利用して実装していることが挙げられる。

5.2 SATPLAN2006

SATPLAN は SATPLAN2006 として第 5 回 IPC(2006 年) の Optimal Planning(Propositional Domains) 部門，SATPLAN2004 として第 4 回 IPC(2004 年) の Optimal 部門での優勝ソルバであり，Kautz らのホームページでバイナリファイルが公開されている。

特徴としては，プランニンググラフを利用した制約充足モデルのプランナであり，そのソルバとしてを利用している。

また，内部で局所的な探索を行うソルバ，大域的な探索を行うソルバなどを使い分けている点も特徴としてあげられる。前者のソルバでは解の最適性を保証することはできないが，確率的かつ貪欲な探索を行いうことで非常に高速に解を発見できる事がある。後者のソルバでは解の最適性を保証した探索が可能であり，解が存在するならば必ずその解を発見するが，問題によっては非常に多くの時間を消費する。以上の長所，短所を組み合わせ，局所的な探索と大域的な探索を切り替えて利用することでより高性能な探索が可能になっていることが特徴である。

5.3 結果

5.4 考察

6 制約モデル

一定の成果を挙げた制約モデルでのプランナの実装として GP-CSP [5] がある。これは GRAPHPLAN の成果を取り込んだプランニンググラフ解析を行う制約モデルプランナの実装である。制約モデル，数理計画モデルの成果と比べ，制約モデルのプランナは International Planning Competition に参加した実績がない。(本当？) 各モデルの優劣に対して決定的な理論付けがないままに制約モデルの研究が進展していない近年の状況に対して問題意識を持ち，本研究では充足可能性モデル，数理計画モデルで得られた知見を利用した制約モデルプランなの実装を行い，その性能を測定，比較した。

制約モデルのソルバとしては近年の制約プログラミングソルバのコンテスト Minizinc Conpetition で優秀な実績を収めている Gecode を利用した。

6.1 Gecode

Gecode は各プログラミング言語上で制約プログラミングを行うためのライブラリ集合である。この実装として C++ を利用し，optiplan のモデルと同様の state change model によって命題変数を表現しプランニンググラフを利用するモデルを制約モデル上で実装した。

6.2 global constraint

global constraint という

7 実験: global constraint を利用した制約モデルのプランナ実装の検証

7.1 結果

7.2 考察

8 結論

9 謝辞

右も左もわからない私に日々教育的視点からためになるアドバイス，課題を示してくださった福永先生には最も感謝しています。本研究の審査を務めていただきました金子先生，田中先生，福永先生，山口先生にも大変感謝しております。ミーティングで研究に関して有益なフィードバックを下さった福永研のみなさまにも感謝しています。

また，学際科学科１期生の諸君，特に学生控室で日々議論に付き合ってくれた陣内くん，僕にプログラミングの手引を与えてくれた田中くんを始め，多くを共に学ぶことのできたみなさまにも非常に感謝しています。

索引

参考文献

[1] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, Vol. 185. IOS Press, 2009.

[2] Avrim L Blum and Merrick L Furst. Fast planning through planning graph analysis. *Artificial intelligence*, Vol. 90, No. 1, pp. 281–300, 1997.

[3] Tom Bylander. Complexity results for planning. In *IJCAI*, Vol. 10, pp. 274–279, 1991.

[4] IBM ILOG CPLEX. 12.2 user's manual, 2010.

[5] Minh Binh Do and Subbarao Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into csp. *Artificial Intelligence*, Vol. 132, No. 2, pp. 151–182, 2001.

[6] Richard Howey, Derek Long, and Maria Fox. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*, pp. 294–301. IEEE, 2004.

[7] H Kautz, B Selman, and J Hoffmann. Satplan: Planning as satisfiability. *Booklet of the 2006 International Planning Competition*, 2006.

[8] Henry Kautz. The role of domain-specific knowledge in the planning as satisfiabuity framework. 1998.

[9] Henry A Kautz, Bart Selman, et al. Planning as satisfiability. In *ECAI*, Vol. 92, pp. 359–363, 1992.

[10] Bernhard Meindl and Matthias Templ. Analysis of commercial and free and open source solvers for linear optimization problems. *Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS*, 2012.

[11] Gurobi Optimization. Gurobi optimizer, 2013.

[12] Stuart Jonathan Russell, Peter Norvig, 康一, 古川. エージェントアプローチ人工知能. 共立出版, 2008.

[13] Menkes van den Briel and Subbarao Kambhampati. Optiplan: Unifying ip-based and graph-based planning. *J. Artif. Intell. Res.(JAIR)*, Vol. 24, pp. 919–931, 2005.

[14] Thomas Vossen, Michael O Ball, Amnon Lotem, and Dana Nau. On the use of integer programming models in ai planning. 1999.

[15] 宮代隆平. 整数計画法メモ. http://www.tuat.ac.jp/m̃iya/ipmemo.html.

[16] 木村俊房, 茨木俊秀. 最適化の数学. 共立出版, 2011.