人　工　知　能 92－7
（1994．1．19）

時間系列を伴う例からの抽象化に基づく戦略学習

松浦　聰　　上原　邦昭　　前川　禎男
神戸大学工学部情報知能工学科

本研究の目的は，競合する複数のエージェントが存在するゲームの領域において，過去の問題解決の経験から，戦略プランを学習するシステムを構築することである．このような問題領域では，それぞれのエージェント間の相互作用を無視できないため，パズルゲームなどの単一エージェントの問題領域に比べて戦略の学習が困難である．

本稿では，ゲームの状態を抽象表現するために基本プランという概念を導入する．さらに，基本プランによって表現されたゲームの各状態を，時間的な因果関係を利用して解析することにより，演繹的な学習手法を用いて戦略プランが獲得できることを示す．また，獲得された戦略プランを実際の問題解決に適用したときの失敗の経験を通して，戦略プラン適用を制御する制約条件の学習を行なうことにより，さらに高度な戦略が利用できるシステムを構築できることを示す．

Strategy Learning from Abstract Past Experiences

Satoshi Matuura　Kuniaki Uehara　Sadao Maekawa
Department of Computer Science and Systems Engineering
Kobe University
Nada, Kobe, 657 Japan

This paper discusses an algorithm to extract strategic plans from past problem experiences in competitive multi-agent domains. In such a dmain with more than one agent learning strategic plans is much more difficult than the single agent dmains, because there is interaction between each agent.

In this paper, we will introduce a concept called simple plan for abstracting the game situation. We will show the system which can deduce strategic plans by analizing the abstract game situation. We will also show the algorithm to learn the conditional knowledge that control use of strategic plans.

1 はじめに

パズルゲームなどの単一エージェントの領域においては，これまでに問題解決の経験から問題解決のための戦略を学習する多くのシステムが提案されている．しかしながら，二人ゲームのような競合する複数のエージェントが存在する領域では，各々のエージェントの相互作用が生じるため，戦略の学習が困難である．

本稿では，ゲームの各盤面状態を問題に依存する性質や構造を用いて抽象化し，これを時間的な因果関係を利用して解析することにより，戦略プランを演繹的に学習する手法を提案する．また，本学習手法の有効性を示すために，二人ゲームである連珠[1]に本手法を適用した結果について述べる．

2 基本プランに基づいた抽象化

連珠のような複数のエージェントが存在する領域では，

(1) 観測可能な行動の組合せが非常に多いため，与えられた例記述のままでは学習はほとんど進行しない

(2) ある1つの盤面状態に対して達成可能なゴールが複数存在する

という問題が起こる．(1) の有力な解決法として，与えられた例記述を問題に依存する性質や構造を示す抽象度の高い記述に写像する抽象化手法 [2] がある．また，(2) に対しては，それぞれのゴールを導くプランの集合として盤面状態を表現するという方法が考えられる．本研究では，これらの要求を満たすために基本プラン [3] と呼ばれる概念を導入する．

基本プランとは，ゴールを導くことのできる行動の順序なし集合である．連珠を例にすると，同種の石で占領可能な5マスの並びがある場合，この5マスを占領するための行動の集合が基本プランということになる．また，基本プランを構成する属性は，各基本プランの識別子とゴールを達成するのに必要な行動数 (深さ) の2つからなり，個々の基本プランは述語 bp(プラン識別子，プランの深さ) を用いて表現される．

簡単のために 図1 に表す三目並べを例にして基本プランの表現形式を説明する．この例の場合，先手の×が達成可能なゴールは {1,2,3}，{7,8,9} の1行目と3行目，{1,4,7}，{3,6,9} の1列目と3列目である．ここで，これらのゴールに向かう基本プランの識別子をそれぞれ bp1, bp2, bp3, bp4 とする．基本プラン bp1 は盤面の 2 と 3 の2箇所に×をおけば，ゴールを達成することができるので，深さを表す 2 番目の引数は 2 ということになり，bp(bp1,2) と表現できる．この様にして，図1 の盤面状態を基本プランによって表現すると {bp(bp1,2), bp(bp2,3), bp(bp3,2), bp(bp4,3) } となる．

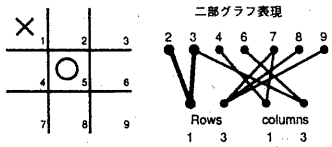

図 1: 三目並べの例

また本稿では，基本プランの組合せとして戦略プランを表すために， 図1 に示すような二部グラフによる表現を用いている．この二部グラフの上側のノードは現在の状況でとることのできる行動 (座標) によってラベル付けし，下側のノードは基本プランの識別子によってラベル付けする．そして，行動と基本プランとの対応づけをアークによって示す．例えば，図1 の二部グラフの太線で表した部分は，2 と 3 に×をおくことでゴールを達成できる基本プラン (すなわち，先の bp1) が1行目に存在することを示している．

3 戦略プランの抽出

連珠のような二人零和有限完全情報確定ゲームでは，ゲームの終盤においては勝者がある必勝のパターンにしたがって行動を決定している場合が多い．したがって，領域知識を用いて対戦履歴から再現された状態を解析した結果，必勝状態[2]であることが導かれた場合には，この状態に導いた行動の集合を戦略プランと考えることができる．

戦略プランを抽出するために，対戦履歴から再現される各々の状態を解析する場合，連珠盤の 15 × 15 の盤面に存在するすべての基本プランを考

[1] 詳しいことは文献 [1] を参照されたい

[2] 本研究では必勝状態を止めなければならない点が2箇所以上ある状態と定義する

慮にいれるのは効率的ではない．本研究が扱っている問題では，ゴール達成までの状態変化が既知であると仮定しているので，基本プランの時間的な因果関係に基づいて，ゴール状態から順に遡って盤面を解析することにより，ゴール達成に関係のある必要最小限の基本プランに絞って解析を進めることが可能となる．このように，基本プランの時間的な因果関係を考慮して，解析を行なうためには，時間的に次のターンの状態を表す述語が必要になる (図 2)．

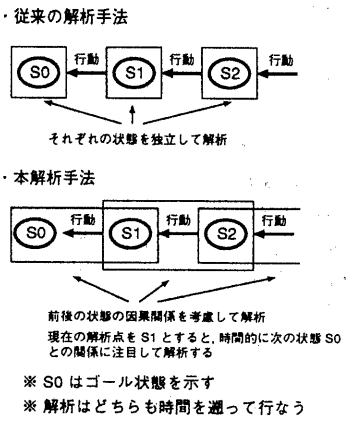

図 2: 本解析手法のイメージ

しかしながら，次ターンの状態を表す述語の真偽は，実際の問題解決の時点では判断できないので，解析時に現れる次ターンの状態を表す述語は，問題解決時には実行不可能な述語になる．また，解析の結果として得られる解析木は，基本プランの識別子や行動を識別するための盤上の座標の情報を含んでいるため，解析木をそのままの状態で異なる問題の解決に利用することはできない．

このため本研究では，解析の結果として得られる解析木から

(1) 次ターンの状態の事実を表す述語を時間的な因果関係に基づいて現在の事実を表す述語に書き換える

(2) 各対戦履歴に固有の情報を削除して一般化する

ことで，解析によって得られた情報を問題解決時に利用可能な知識に変換している．(1) の未来の事実を表す述語には，次ターンの状態の解析結果を表す next_state(State, Depth, Agent) と次にとった行動を表す next_action(Point, Agent) の 2 種類がある (図 3参照)．また，(2) の一般化には，解析木に含まれる情報のうち，基本プランの深さを表す引数以外をすべて変数化する方法を用いている．

3.1 状態解析手法について

本節では，人間同士の連珠の対戦履歴の解析を例に用いて説明する．状態の解析は，図 4に示すアルゴリズムを用いて行なう．このアルゴリズムでは，解析の結果として得られる解析木が，問題解決時に実行可能な述語のみで表せる場合に，現在行なっている状態の解析を終了し，2 ステップ遡った次ターンの状態の解析に移る．また，システムが利用可能な述語論理概念として図 3に示すものが与えられているとする．

- bp(BPLabel,Depth,Line,Agent)
  BPlabel が Line にある深さ Depth の Agent の基本プランであることを示す．
- action(BPLabel,Point)
  基本プラン BPLabel が行動 Point を持つことを表す．
- must_defense(Point,Depth,Agent)
  Agent に相手が阻止しなければならない深さ Depth の行動 Point が存在することを示す．
- current_state(State,Depth,Agent)
  Agent が State に示す深さ Depth の状態であることを表す．
- next_state(State,Depth,Agent)
  時間的に 1 つ後の状態の Agent の状態が深さ Depth の State であることを表す．
- next_action(Point,Agent)
  Agent の次の行動が Point であることを表す．
- action_flow(ActionList,Agent).
  Agent の行動の実行順を ActionList に示す．

＜各引数の説明＞
BPLabel: 基本プランの識別子
Depth: ゴールまでに必要な最小の行動数
Line: プランの存在するラインの識別子
Agent: エージェント名
Point: 行動を識別するための盤上の座標
State: 状態の解析結果で defenceless (防御不能) または defensible (防御可能) をとる
ActionList: 行動の実行順を示したリスト

図 3: システムに用意される述語

Algorithm 状態の解析
begin
例をゴール状態から2ステップ遡り，$i=1$ とする;
repeat begin
状態 S_i を領域知識を用いてボトムアップに解析する;
while 解析が進行 **do begin**
部分解析木を $\mathcal{PT}_i$ とする.
if $\mathcal{PT}_i$ に実行不可能な述語 $\mathcal{NOP}_{i,j}$ が存在
then begin
$\mathcal{NOP}_{i,j}$ を実行可能な述語に書き換える;
$\mathcal{PT}_i$ を一般化したものを領域知識に追加する
end
end;
解析結果を C とする;
例を2ステップ遡り，i++
end until C = 'defenseless'
end.

図 4: 状態解析アルゴリズムの概要

- 領域知識:

must_defense(P1,D1,A), must_defense(P2,D2,A),
D1 ≤ D2 → current_state(defenseless,D1,A). … (R1)
next_action(P,A), next_state(defenseless,D1,A),
D2 is D1+1 → must_defense(P,D2,A). … (R2)
ゴール達成状態 S0 は,
current_state(defenseless,0,Agent) とする. … (R3)

(R1) は阻止しなければならない行動が2つあれば，防御不能な状態になることを定義している．また，(R2) は次の行動 P によって，次ターンの状態が防御不能な状態になるならば，行動 P を阻止しなければならないことを定義している．

- 対戦履歴の例:

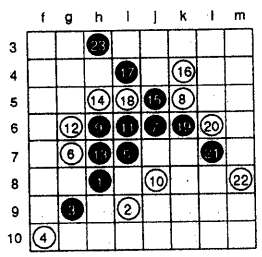

図 5: 対戦履歴

まず，領域知識を用いて，図5のゴール達成状態 S0 から対戦履歴を2ステップ遡った状態 S1 を解析する．領域知識を用いて

next_action(h3,black), next_state(defenseless,0,black)
→ must_defense(h3,1,black).

が生成される．この部分解析木は,「次の先手の行動 h3 によりゴールが達成されるので，先手の行動 h3 は阻止しなければならない行動である」ということを意味している．ここで，下線部に未来を表す述語 next_action/2, next_state/3 が用いられているので，これらの述語を時間的な因果関係に基づいて現在の基本プランを表す述語に書き換えると

bp(bp1,1,l1,black), action(bp1,h3)
→ must_defense(h3,1,black).

となる．これを一般化した

bp(X,1,L,A),action(X,P,A)
→ must_defense(P,1,A). …(R4)

を領域知識に追加する．(R4) は,「深さ1の基本プランの行動は直ちに阻止しなければならない」ことを意味しており，領域知識の (R2) の定義を具体化したものになっている．

さらに，(R4) を用いて，状態 S1 を再び解析することにより，以下に示す解析木が得られる．

bp(bp1,1,l1,black), action(bp1,h3)
→ must_defense(h3,1,black).
bp(bp2,1,l1,black), action(bp2,m8)
→ must_defense(m8,1,black).
must_defense(h3,1,black), must_defense(m8,1,black)
→ current_state(defenseless,1,black).

この解析木は，問題解決時に実行可能な述語のみで表現されているので S0 の解析を終了する．

S1 が defenseless な状態であるので，さらに，領域知識を用いて対戦履歴を2ステップ遡った状態 S2 を解析する．このとき得られる解析木は

bp(bp3,1,l2,black), action(bp3,l6)
→ must_defense(l6,1,black).
next_action(l7,black), next_state(defenseless,1,black)
→ must_defense(l7,2,black).

となる．S1 のときと同様にして下線部を書き換えることにより

bp(bp1,2,l2,black), action(bp1,h3),
action(bp1,l7), bp(bp2,2,l2,black),
action(bp2,l7), action(bp2,m8),
→ must_defense(l7,2,black).

を得る．これを一般化した

bp(X,2,L,A), bp(Y,2,L,A), action(X,P), action(Y,P)
→ must_defense(P,2,A). … (R5)

を領域知識に追加する．(R5) は，「同一ライン上に存在する深さ 2 の 2 つの基本プランに 1 つの共通な行動が存在するならば，この行動を阻止しなければならない」ことを意味している．このようにして，後手にとって防御不能な状態 defenceless が続く限り解析を続けていく．

3.2 解析木から戦略プランへの変換

must_defence/3 は相手にとって阻止しなければならない行動を示しているが，逆に攻撃側がこの行動を実行すれば，相手が防御不能な状態を実現できる．したがって，問題解決時には行動を決定するための戦略としてこの概念を用いることができる．しかしながら，must_defence/3 はただ一つの行動を提示するのみであり，ゴールまでの一連の行動計画を戦略プランとして提示することはできない．

本節では，解析時に得られる解析木と行動の履歴を用いて，ゴールまでの一連の行動計画を戦略プランとして提示する知識を生成する手法について説明する．本手続きの大まかな流れは，

1. 行動履歴の解析時に，既に解析の終了した Agent の行動を，述語 action_flow(ActionList, Agent) の引数 ActionList に記述しておく．
2. 部分解析木 must_defence/3 の条件部と action_flow(ActionList, Agent) を連結する．
3. 述語 action_flow/2 の引数 ActionList に含まれない行動に関する述語を条件部から削除し，Depth 以外の引数の定数を変数化することにより一般化する

となる．このような変形操作によって，与えられた問題と条件部のマッチングを行なうことによって，ゴールまでの行動計画を戦略プランとして導出できる知識を生成することができる．

例えば，図 5 の対戦履歴の解析中に得られる解析木

bp(bp1,2,l2,black), action(bp1,h3),
action(bp1,l7), bp(bp2,2,l2,black),
action(bp2,l7), action(bp2,m8),
→ must_defense(l7,2,black).

の条件部を action_flow/2 と連結すると，

bp(bp1,2,l2,black), action(bp1,h3),
action(bp1,l7), bp(bp2,2,l2,black),
action(bp2,l7), action(bp2,m8),
→ action_flow([l7,h3],black).

となる．ここで，条件部の action(bp2,m8) は action_flow([l7,h3],black) の引数 [l7,h3] に含まれていない行動を表す述語であるため削除し，さらに Depth 以外の引数の定数を変数に置き換えて一般化すると

bp(B1,2,L,A), action(B1,P1), action(B1,P2),
bp(B2,2,L2,A), action(B2,P2)
→ action_flow([P2,P1],A).

となる．この知識は，ゴールまでの行動計画を導出する知識として利用できる．以下では，このような知識のことを戦略プランと呼ぶ．

4 戦略プランの利用

本章では，3 章での手法によって獲得された戦略プランを，どのようにして問題解決時に利用するかについて述べる．

戦略プランの分割

戦略プランは，対戦履歴を解析するごとに領域知識に追加されていく．その結果，多くの戦略プランが追加されるにつれ，戦略プラン利用時の探索量が増大するためにマッチングコストが高くなってしまうという問題が生じる．また，戦略プランの条件部は，深さが増すにつれて大きくなり，一度にすべてのマッチングを行なうことができなくなる．

本研究では，これらの問題の部分的解決のために，縦，横，斜めの同一ライン上に存在する基本プランごとに戦略プランを分割して，それぞれの部分ごとにマッチングを行なうという方法を用いる．また，一般に深さの浅い基本プランほど存在確率が低いので，ある解決すべき問題が与えられた場合に，深さの浅い部分グラフから順にマッチングしていくことで，マッチング可能な戦略プランの候補を絞り込むことができる．

図 6 は，図 5 に示す対戦履歴の解析によって最終的に得られる戦略プランを二部グラフで表現したものである．この二部グラフの上側のノードは実行可能な行動を示しており，下側のノードは異なるゴール状態を導く基本プランを示している．また，上側と下側のノードを結ぶアークは各基本プランと行動の対応づけを示している．さらに，こ

の二部グラフの行動を示す変数 P1, P2, P3, P4, P5 の上に書かれた○印の数字は行動の実行順を示し，この順に行動を実行していけばゴールを達成できることを表している．

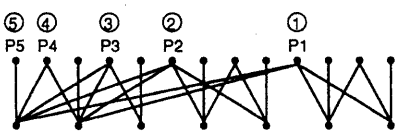

図 6: 図 5 の解析から得られる戦略プラン

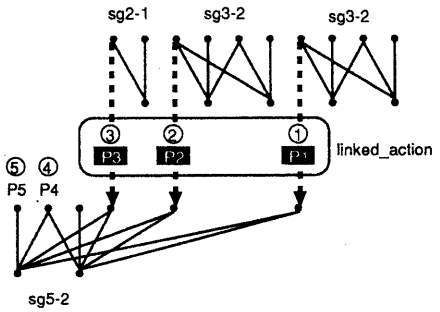

図 7: 分割された戦略プラン

例えば，この戦略プランを同一の並び (ライン) に存在する基本プランごとに分割すると図 7 のようになる．ここで，分割することによって得られる部分グラフを，その部分グラフを構成する基本プランの深さ D と 数 N によって sgD-N と表すことにする．例えば，図 7 の部分グラフ sg3-2 は深さ 3 の 2 つの基本プランから構成されていることを表している．このようにして，戦略プランを部分プランに分割して表現することにより，戦略プランが限られた種類の部分グラフから構成されていることが分かる．このような特徴を利用することにより，ある戦略プランがマッチングに失敗しても，その部分グラフのマッチング結果を，次の候補となる戦略プランのマッチングに利用することを可能としている．

ここで，分割された部分プランを述語論理表現するために，述語 linked_action(Point, PPKind, Line, Agent) を導入する．この述語は，各部分グラフに共通の行動を示すために用いている (図 7 の点線矢印の白黒反転で示した行動)．この述語は「Line に存在する行動 Point は Agent の PPKind で示す種類の部分プランに含まれる行動である」ことを示している．なお，引数 PPKind は sg2-1 などの部分プランの種類を表す定数をとる．

図 7 に示す分割された戦略プランを述語論理表現すると以下のようになる．

```
bp(X,2,L,A),action(X,P)
→ linked_action(P,'sg2-1',L,A).   (1)
bp(X1,3,L,A),bp(X2,3,L,A),
action(X1,P),action(X2,P)
→ linked_action(P,'sg3-2',L,A).   (2)
bp(X1,5,L1,Agent),bp(X2,5,L1,Agent),
action(X1,P5),action(X1,P4),action(X2,P4),
action(P1,X1),action(P1,X2),action(P2,X1),
action(P2,X2),action(P3,X1),action(P3,X2),
linked_action(P1,'sg3-2',L4,Agent),
linked_action(P2,'sg3-2',L3,Agent),
linked_action(P3,'sg2-1',L2,Agent)
→ action_flow([P1,P2,P3,P4,P5],Agent).   (3)
```

戦略プランの利用

次に，戦略プランの利用方法であるが，上記の述語論理表現された戦略プランを図 8(a) の問題に適用する場合について説明する．

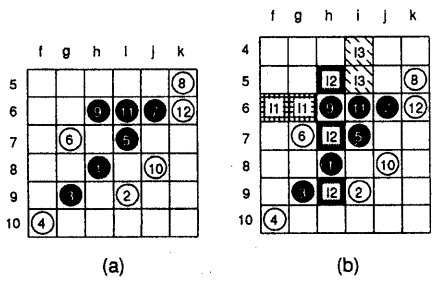

図 8: 詰め連珠の問題

まず，部分プラン (1), (2) を用いることにより，以下の事実が導出される (図 8(b) 参照)．

```
linked_action(f6,'sg2-1',l1,black).   …(※)
linked_action(g6,'sg2-1',l1,black).
linked_action(h5,'sg3-2',l2,black).
linked_action(h7,'sg3-2',l2,black).
linked_action(h9,'sg3-2',l2,black).
linked_action(j4,'sg3-2',l3,black).
linked_action(j5,'sg3-2',l3,black).
```

ここで，(※) は「ライン l1 に存在する部分プラン sg2-1 の行動 f6 は他の部分プランと共通の行動である」を意味している．

また，部分プラン (3) は，「P1, P2, P3 は同一ラインに並ぶ行動でなければならない」ことを示しており，これらの条件を満たす行動の組を探索すると {g6, h5, i4} が見つかる．

さらに，部分プラン (3) の linked_action/4 以外の述語とのマッチングを試みることにより，

action_flow([i4,h5,g6,f7,i4],black)

が戦略プランの候補として導出される．もし，部分プラン (3) のマッチングに失敗しても，マッチング途中に生成された事実 linked_action/4 を他の候補となる戦略プランとのマッチングに利用できるため，マッチング時のバックトラックを少なからず抑えることが可能である．また，戦略プラン action_flow/2 に含まれる基本プランが相手の行動によって実行できなくなった場合には，その戦略プランを放棄し，その時点で再び戦略プランの適用を行なうことにより，ゴールまでの行動計画が導き出される．

5 戦略プランの適用条件の追加

一般に二人ゲームでは，後手の戦略 (以下では妨害プランと呼ぶ) が先手の戦略に影響を及ぼすことが知られている．このような妨害プランの影響を問題解決時にできるだけ小さくするためには，各々の状況に応じて適切な戦略プランを選択したり，行動を決定しなければならない．そのためには，戦略プランの選択や行動の実行を制御するための制約条件が必要になってくる．そこで本章では，戦略プランの実行に関する制約条件を，失敗の経験を通して領域知識に追加していくアプローチについて述べる．

本研究で対処する戦略プラン実行の失敗として，

(A) 相手の一時的な妨害に応答せずに戦略プランの実行を続けてしまう

(B) 相手の連続的な妨害によって，相手に先にゴールを達成されてしまう

という場合を考える．それぞれの失敗は逆転負けなどの結果を導くので，どちらの失敗もゲーム終了時に認識することができる．これらの失敗を区別するために，プラン実行度という指標を導入する．プラン実行度とは，自分の戦略プランに含まれる行動を実行した場合に +1，相手のプランに含まれる行動を実行した場合に −1 の値を持つ指標である．失敗を認識した場合に，自分のプラン実行度がゲーム了時まで +1 の場合は (A) の失敗，途中から −1 になっている場合は (B) の失敗として区別する．

失敗の原因となる行動は，プランの深さを併せて用いることで決定できる．(A) の失敗の場合は，後手のプランの深さが初めて先手のプランの深さよりも浅くなっている状態における行動，(B) の失敗の場合はプラン実行度が −1 に変わる直前の行動ということになる．

次に，制約条件の追加方法についての大まかな枠組について説明する．例えば，戦略プラン SP_i を実行した時に，システムが (A) の失敗を認識したとする． まず，失敗の原因となる行動 A_ij を見つけ出し，相手の一時的な妨害を除去する代替手 A'_{ij} を立てる．ここで，行動 A_{ij} をとった環境に存在する後手の妨害プランを，3 章で述べた方法と同様の解析によって求め，これを OP_k とすると

$$A_{ij} \cup OP_k \rightarrow A'_{ij} \qquad (1)$$

を制約条件として領域知識に追加する．この制約条件は，戦略プラン SP_i の行動 A_{ij} を実行しようとする場合に，もし妨害プラン OP_k が存在するならば，行動 A'_{ij} を実行しなければならないことを示している．なお，OP_k, A'_{ij} は座標や基本プランの識別子は変数に置き換えて一般化してあるものとする．また，後の対戦において，この A'_{ij} が失敗した場合には，同様の手続きによって，代替手 A''_{ij} を立てる．A'_{ij} が失敗したときの後手の妨害プランを解析によって求め，これを OP'_k とすると

$$A'_{ij} \cap OP'_k \rightarrow A''_{ij} \qquad (2)$$

を追加し，(1) と (2) の制約条件から

$$A_{ij} \cap OP_k \cap OP'_k \rightarrow A''_{ij} \qquad (3)$$

を追加する．(3) の制約条件により，SP_i の行動 A_{ij} を実行しようとする場合に，妨害プラン OP_k と OP'_k が同時に存在するならば，行動 A_{ij} を中止し，行動 A''_{ij} を実行しなければならないことが示される．

さらに，A''_{ij} が失敗し，代替手が存在しない場合には，システムは (B) の失敗を認識し，戦略プ

ラン SP_i の適用が誤っていたと考える．この場合には，戦略プランの適用を行なった状態を S とすると，先手にとって致命的となった妨害プランの S における状態を見つけ出す．これを OP_S とすると

$$SP_i \cap OP_S \rightarrow fail. \qquad (4)$$

という条件を追加し，戦略プランの選択時における同様の失敗を回避する．以上の枠組を学習システムに導入することにより，戦略プランが一般化され過ぎている場合でも，システムの知識全体で適切な一般化レベルを達成できる．

5.1 例 1. 一時的な妨害の回避

本節では，簡単な例として，先手が戦略プランの行動を実行して三をつくったときに，後手が四をつくった場合を考える (図 9)．この場合，制約条件を一切持たない初期状態では，先手は後手の四を止めずにさらに自分の戦略プランを進めようとするので，次のターンで後手に五を達成されてしまう．

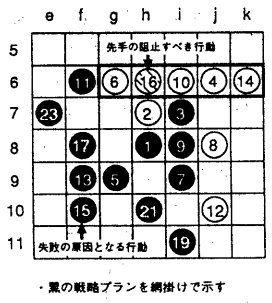

図 9: 一時的な妨害

状態	S1		S0	
順番	黒	白	黒	白
行動	f9	j9	f10	j7
プランの深さ	2	1	1	0
プラン実行度	+1	+1	+1	+1

表 1: 表を用いての解析

システムは表 1 を用いて，この失敗の履歴を解析し，状態 S0 においての行動 f10 が失敗の原因であることを見つけ出す．さらに，ゴール状態を遡って解析を行なうことにより，f10 を実行した状態での後手の妨害プランを見つける．

ここで，状態 S1 における先手と後手それぞれの状態と先手のとるべき行動を一般化することにより「先手が戦略プランを実行して三をつくったときに，後手が四をつくった場合には，戦略プランの実行を一時中止し，後手の四を止めなければならない」ことを意味する制約条件を領域知識に追加する．

```
/* 失敗した行動 */
  take_action(X1,A),
  bp(B1,2,L1,A),action(B1,X1),
/* 制約を起動する条件 */
  bp(B2,2,L2,A), bp(B3,2,L2,A),
  action(B2,X2), action(B3,X2),
  action(B2,X3), action(B3,X3), — (先手の三を表す)
  bp(B4,1,L3,B), action(B4,X4) — (後手の四を表す)
/* 代替の行動 */
  → take_action(X4,A). … (RC1)
```

5.2 例 2. 連続的な妨害の回避

次に戦略プラン選択に関する制約の学習例を示す．一般に，詰め連珠の問題に複数の戦略プランの適用が可能な場合がある．このような場合には，すべての戦略プランが成功するとは限らないので，状況に応じて適切な戦略プランを選択する必要がある．図 10 に示すような詰め連珠の問題を考える．図 10 の問題では適用可能なプランが複数存在している．

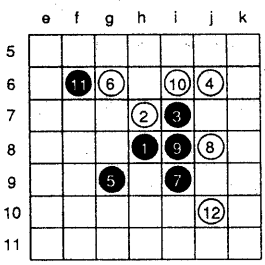

図 10: 詰め連珠の問題

相手の行動によって現在の戦略プランが阻止されてしまった場合には，システムの問題解決部はその状態で再び戦略プランとのマッチングを試み

る．もしマッチする戦略プランが見つかるならば，その戦略プランから導出される行動計画に基づいて以後の行動を決定する．マッチする戦略プランが見つからなければ，一つ前の戦略プランの選択が失敗であったと考えて，解析によって後手の妨害プランを見つけ，これと同じ妨害プランが含まれている状況では，再び，この戦略プランを選択しないように制約条件を付加する．

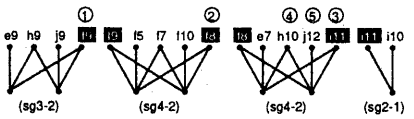

図 11: 適用可能な戦略 (a)

例えば，図 11 の戦略プランを選択した場合には，先手 f9，後手 h9，先手 f8 のあとで，先手が最善手を打ったとしても，後手 j7，j5，k4 の四追い[3]で後手の勝ちになる．このとき，先手は戦略プランの選択の失敗を自覚し，解析によって後手の妨害プランを見つけ出す (図 12)．

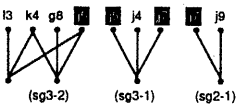

図 12: 妨害プラン

状態	S4		S3		S2	
順番	黒	白	黒	白	黒	白
行動	f9	h9	f8	j7	j9	j5
プランの深さ	2	2	2	1	2	1
プラン実行度	+1	-1	+1	+1	-1	+1
状態	S1		S0			
順番	黒	白	黒	白		
行動	j4	k4	l3	g8		
プランの深さ	2	1	2	0		
プラン実行度	-1	+1	-1	+1		

表 2: 失敗点の解析

システムは図 11 の戦略プランと図 12 の妨害プランを比較することにより表 2を得る．この表のプラン実行度から，先手 f8 の行動が失敗の原因と決定される．行動 f8 を実行した状況では一時的な妨害プランの回避は不可能であるため，図 11の戦略プランを選択したことが失敗であるとして，制約条件

SP(a), OP → fail … (RC2)

を追加する．ここで，SP(a) は図 11 の戦略プランの条件部の述語論理表現を指し，OP は図 12 の妨害プランの条件部の述語論理表現を指すものとする．

再び同じ問題が与えられた場合には，制約条件 (RC2) によって図 11 の戦略プランが適用できなくなるので，今度は 図 13の戦略プランが選択される．この後，先手は戦略プランと前節の例 1 で得られた制約条件により行動を決定していく．

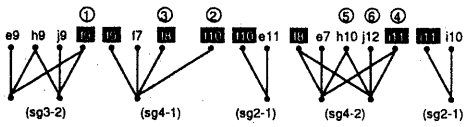

図 13: 適用可能な戦略 (b)

ところが，後手 f8 により，この戦略プランは阻止されてしまう．ここで，先手は再び戦略プランのマッチングを試み，図 14 のプランとのマッチングに成功する．

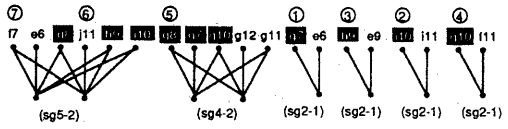

図 14: 適用可能な戦略 (c)

この状態から，先手は導出された行動計画にしたがって g7，i10，h9，g10，g8，j11 の四追いで五を達成することができる．このときの対戦履歴を図 15に示す．

[3]連珠では四の連続で詰めることを四追いと呼んでいる．

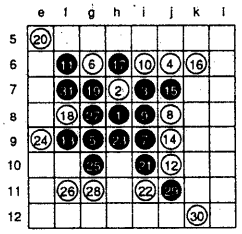

図 15: 問題の解決

以上のような枠組をシステムに提供することにより，対戦の経験にともなって行動を決定するための制約が増加していくので，経験の量に比例して対戦相手に対して知的に応答することが可能になる．

6 まとめと考察

本稿では，連珠を学習領域として取り上げ，特に連珠における終盤の戦略の学習について述べてきた．本研究は，基本的に解析によって学習が進行するので，大きな枠組では EBL による知識の洗練化の研究に位置付けられる．しかしながら，推論時間の短縮など高速化を主目的とはしていない点，および時間的な推論を通して事例を解析しながら，初期概念記述をもとにして具体的な概念定義を獲得する点で，他の EBL を利用したシステムとは異なっている．

また，戦略プランを獲得するだけでなく，戦略プランを利用して問題解決を行ない，失敗の経験を通して，戦略プランの選択や行動の決定を制御するための制約条件を追加する手法を導入した．この手法の導入によって，知識の抽出，利用，修正の統合された枠組を提供し，より高度な問題解決を可能にしている．

本研究と同様の抽象空間を学習に利用しているシステムには Epstein の HOYLE [3] などがある．Epstain は ゲームの状態を表すのに strategic map という二部グラフを用いている．また，学習の例として tic-tac-toe[4](基本プラン数: 3，行動空間: $3 \times 3 = 9$) や qubic[5](基本プラン数: 4，行動空間: $4 \times 4 \times 4 = 64$) を用いており，比較的小さいコストで strategic map を作成できる．ところが，連珠 (基本プラン数: 5，行動空間: $15 \times 15 = 225$) などになると，探索空間が比較的大きくなるために strategic map の作成にコストがかかってしまう．本稿では，戦略プランを図的に示すのに二部グラフを用いているが，strategic map に相当する空間全体を表現するグラフを作成する手法は採用しなかった．

[4]三目並べのこと．

[5]$4 \times 4 \times 4$ の立方体を利用した四目並べのこと．

7 今後の課題

本研究で提案したシステムでは，戦略プランが適用可能な状態からの問題解決のみが可能である．本稿では触れなかったが，戦略プランが適用可能な状態までどのように行動を決定していくかという問題も興味深い問題である．本システムによって序盤戦からの問題解決を可能にするためには，戦略プランにマッチする状態までの行動を決定するための戦略を学習する枠組が必要である．連珠では，序盤では先手，後手にそれぞれ定石があり，この定石を知らずして序盤を制することはほぼ不可能と考えられる．このような定石の獲得には多くの事例を扱うことが必要になるので，帰納的学習の手法が適用できると思われる．また，今後の課題として，本研究の枠組を他の領域の戦略学習に適用することを検討している．

参考文献

[1] 新井華石 : 連珠必勝法, 虹有社, 昭和 54 年 7 月 20 日 8 刷発行.

[2] Tenenberg, J. D. : Preserving Consistency across Abstraction Mappings, *Proc. of IJCAI-87*, pp. 1011–1014 (1987).

[3] Epstein, S. L. : Learining Plans for Competitive Domains, *Proc. of the eighth Int. Conf. on Machine Learning*, pp. 190–197 (1991).