修 士 論 文

動的な環境に対応するプラン再生成システム

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

陳　歷峰

2014 年 3 月

修 士 論 文

動的な環境に対応するプラン再生成システム

指導教員　東条　敏 教授

審査委員主査　東条　敏 教授
審査委員　島津　明 教授
審査委員　Nguyen Minh Le 准教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

1210032 陳　歷峰

提出年月: 2014 年 2 月

概 要

プランニングは人工知能研究の黎明期から研究対象の中心の一つであるが、現在ではプランナによってシステム行動を制御することが多い。従来のプランナには設計者があらかじめ考えられる状況や行動を組み込んでいるため、あたかもそのシステムが状況に合わせて知的にプランを生成しているように見える。しかし、状況が想定範囲を外れた途端、このシステムは破綻してしまう。このアプローチの限界は数多くの研究者によって指摘されており、能動的にプランを再生成する機能が必要とされている。本研究は、パタン認識に基づいて動的変化する環境に対応するプランを作成するためのシステムを構築する。

本研究のシステムでは、目標達成のために三つのブロック状態空間を利用する。状況や目標の変更があれば、現在の状況を考慮した上で達成すべき新目標を定める。具体的には、目標を達成するために現在の状態空間と目標状態空間の間に類似した部分がある場合には、現在の状態と比較し新たなプランを再構成し、バーチャル状態で再び新しい初期状態から探索を始める。その結果、システムもある程度に環境に対応できるものとなる。

目 次

第1章　はじめに

人間がロボットを発明して以来、ロボットへより効率的なコントロール方法を見つけようとしているが、マニュアルによるプログラミング技術がまだ主なロボットコントロールの方法である。しかし、人工知能のプランニングによって、多様で変化の多いの環境の中で、環境変化に適応的に合わせるような自律メカニズムがある程度に実現されている。

現在、自律的なプランニングアルゴリズムが世界中のいろんな分野で活躍している。例えば、自動家庭掃除ロボットは、あらゆる住環境で任せて、自律的な掃除プランを生成することができる。一方、火星探索器においては、電波は地球から火星まで届くには7秒間が掛かるので、地面から即時的にコントロールができず、探索器に自律プランニングアルゴリズムを搭載させることを求める。

この論文で紹介するプランニングシステムは、「ブロックの世界」とよばれる人工的な実験を対象としている。ブロックの世界は、一定の数のブロックとテーブル、ロボットアームからなる限定された世界である。ブロックの世界などを対象としたプランニングを行うシステムは、1970年代から、数多く作られているが、最も古典的なプランニングシステムはSTRIPS [1]であり、その他のシステムはほぼ目標状態から初期状態に至る道筋を探索するアルゴリズムを採用している。

ブロックの世界は、人工知能分野では過去何度か実験の対象に選ばれ、その中ではウィノグラードのSHRDLUという研究が最も有名である。ウィノグラードのブロックの世界では、ブロックの他にプラミッドや箱等もあり、「一番大きいブロックを箱にいれよ」といった命令を解釈実行したり、「ブロックをピラミッドの上に載せよ」といった命令に対し、「ピラミッドの上に載せることはできません」と返答してきたりもする。しかし、SHRDLUをそのままに応用では効かない、任意の環境に対応するプランの系列の生成するプログラム論理的にはなっていない。

プランニング生成に関する先行研究であるHelmet[2]は、プランニング問題のいくつかのクラスを解析し、その結果、GRAPHPLAN[3]とSATPLAN[4]のような制約ベースアプローチが最良であることを示した。しかし、これらのプランニングでは、環境は静的である，または環境変化がプランナに完全既知であることが想定されている。しかし、実世界の環境は一般に動的に変化するものであり、動的環境ではプランナに順応性が求められる。近年、HTN（2006）[4]のプランニングは、効率性や知識表現力が優れているだけでなく、動的環境に対応できるという利点があるが、状況変化で実行中のプランが無効になる問題がある。すなわち、大規模な問題においては、状況が変わる時点で割り込まれた新しいプランによって元のプランが無効になれ、効率が低下することになってしまう。

本研究でシステムでは、目標達成のためにパタン認識を利用する。状況や目標の変更があれば、現在の状況を考慮した上で達成すべき新目標を定める。具体的には、目標状態と現在の状態に類似した部分がある場合には、大規模の問題を分割し、数個の子問題を作成する。再び新しい初期状態から探索を始める。その結果、システムもある程度に環境に対応できることになり、大規模な問題を探索時間を大幅に軽減することができる。

第2章　関連研究

2.1　STRIPS

動作プランニングは、様々な問題領域で使用されるアルゴリズム――STRIPS を用いる。STRIPS は、目標状態のリストを用いるプランニングアルゴリズムであり、作用素を用いて、問題に適用できる手続きの順番を記述である。環境モデルにおいて、作用素には、作用素の条件リストを満足すると、削除リストと追加リストを適用し、操作の結果として、世界モデルに変化の影響を与える。例えば、pickup(A) の例を以下に示す。

pickup(A)

前提条件：clear(A), ontable(A), handempty
削除リスト：ontable(A), handempty
追加リスト：holdng(A)

前提条件の記述は A の上には何もなく、かつ A がテーブルの上に置いてあり、かつアームが何も掴んでいない状態である。この作用素が適用されると、追加リストのアームは A を掴んでいるという状態を環境モデルに追加し、削除リストの A がテーブルの上にあり、かつアームは何も掴んでいない状態を環境モデルから削除する。

STRIPS は、初期状態の対象世界を与えられた目標状態に変化させることを達成するためのプランを見つけるために、環境モデルの状態空間を探索する。その略図を図 2.1 に示す。対象世界の初期状態と目標状態は事前に入力されており、プランニング部が起動される。プランニング起動すると、初期状態の記述を初期状態スタックに、目標状態の記述を目標状態スタックに読み込む。目標状態スタック作成ルーチンの流れを図 2.2 に示す。

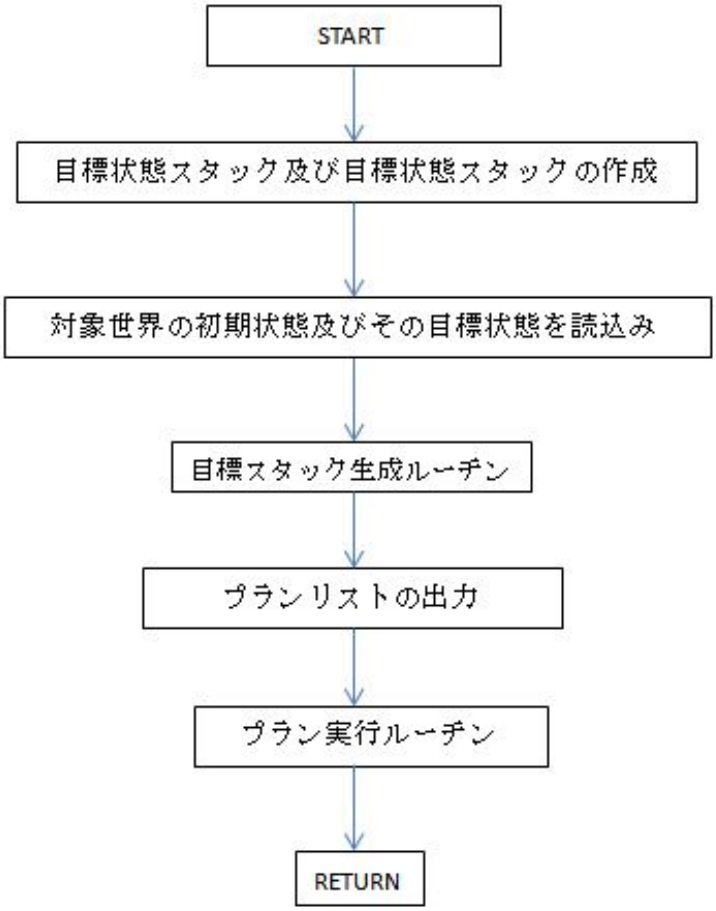


図 2.1: プラニング部の流れ

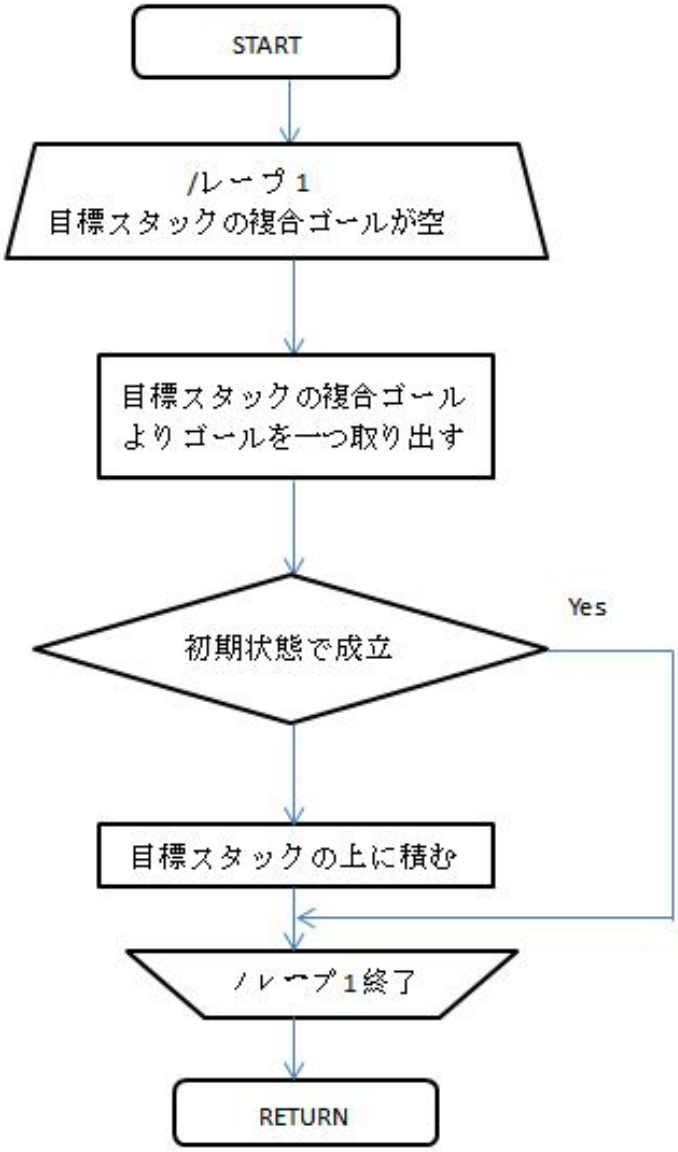


図 2.2: 目標スタック生成ルーチン

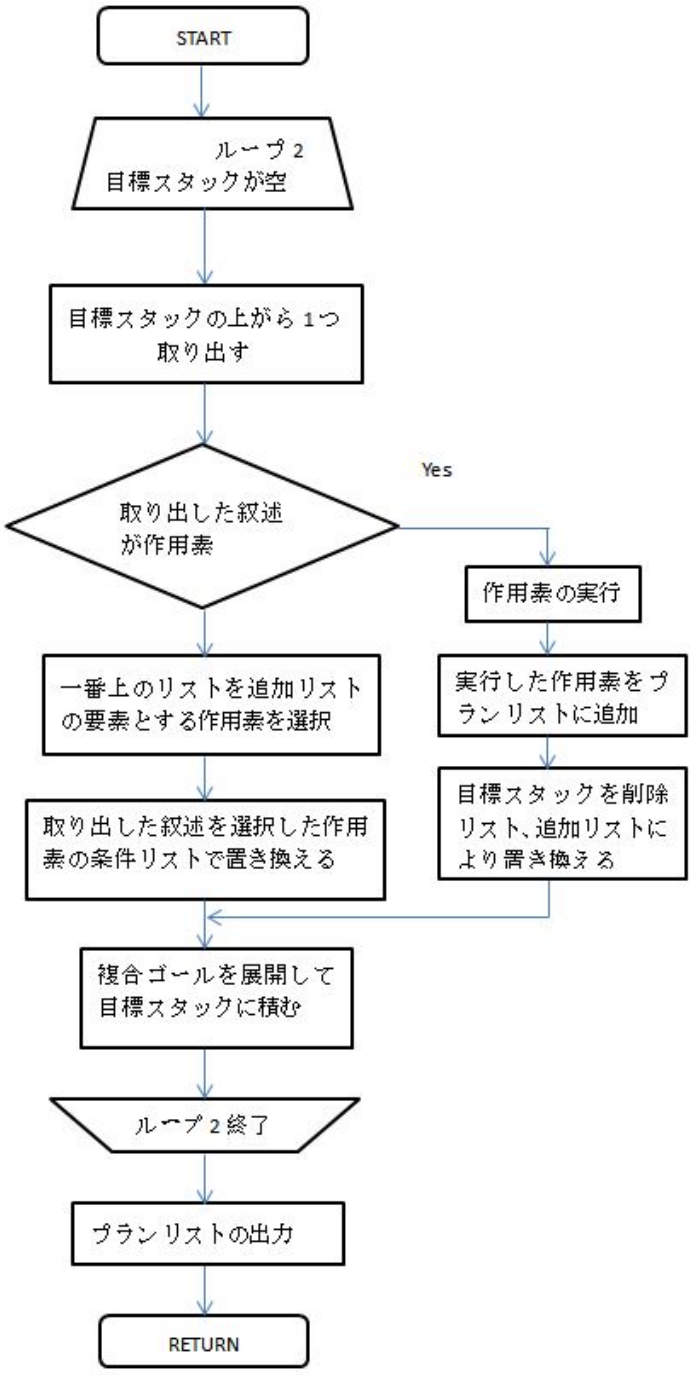


図 2.3: プランリスト生成ルーチン

この目標状態スタックを上から順に読み出し、作用素であれば実行しプランリストに追加する。作用素でなければ、その記述を追加リストの要素とする作用素を選択し、選択した前提条件で置き換え、目標スタックに積む。この操作を繰り返し目標スタックが空になった状態で処理が終了する。終了時のプランリストが求めるプランである。プランリスト生成ルーチンの流れを図 2.3 に示す。

2.2 バックトラック法

バックトラック法（backtraking）あるいは後戻り法とは、問題の解を見つけるため、解の候補を全て調べることを組織的かつ効率よく行うための技法である。難しい組み合わせの問題を解くための技法であり。応用範囲も幅広い。　数多くの組み合わせの中から、ある条件を満たすものを見つけたり、評価値の最もよいものを見つけたりする問題を考えることに踏まえ、問題の構造を解析し、方程式を解くような解法があればよういが、ほとんどの組み合わせの問題においては、解を求めるための方程式は存在せず。そのため、解の候補になりうるところを片端から調べ上げるしかない場合があり、このようなときに、役に立つアルゴリズムがバックトラック法である。

2.2.1 エイト・クイーンパズル

「8 クイーン」はコンピュータに解かせるパズルの中でも特に有名な問題です。8 クイーンは 8 行 8 列のチェスの升目に、8 個のクイーンを互いの利き筋が重ならないように配置する問題です。クイーンは将棋の飛車と角をあわせた駒で、縦横斜めに任意に動くことができます (図 2.4)。解答の一例を図 2.5 に示す。

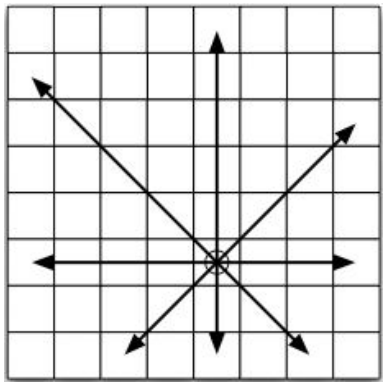


図 2.4: クイーン移動の例

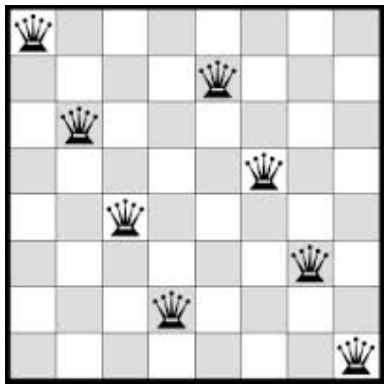


図 2.5: エイトクイーンパズル

2.3 プランニングシステム

問題解決のゴールが複数あって、それらのゴールが互いに干渉しあう場合の解決手順を推論するプランニングシステムについて紹介する。

2.3.1 はじめに

プランニングの手法の目的は、与えられたゴールを達成するための手続きの列を生成することである。プランニングの応用分野としては、問題解決や自動プログラミングなどが挙げられる。

問題解決の方法では、初期状態から出発して目標状態に到達するめでの経路を探索するテクニックが重要な手法となる。最良優先探索やαβ刈り等がその例である。問題自体が小規模な場合は、縦型探索や横型探索のような、悉皆的な探索も十分有効な手法となる。しかしより複雑な問題においては、探索空間が膨大になって、演算時間の面から実用的な解決は得られなくなってしまう。例えばチェスのようなゲームにおいて、次に打てる手が平均10通りあるとしよう。すると、ある局面の１手先は10通り、２手先は100通り、３手先は1000通りとなる。コンピュータのスピードが10倍になっても同じアルゴリズムでは、同じ時間で１手余分に先読みできるだけである。

従って、問題を分割し、分割された副問題を解決し、最後に副問題の解決策を組み合わせて最終的解決を得る、という手法の開発が重要なものとなる。そうした分割を行う際に一番大きな課題となるのが、分割した問題間の干渉いかに調整するかといった、副問題の間の相互干渉である。この、相互干渉の課題さえ解決できれば、問題の分割は、大規模な問題に対する有効な手段となる。

本研究に使用しているプランニングシステムは、「ブロックの世界」図 2.6 とよばれる人工的な実験室を対象にしている。ブロックの世界は、いくつかのブロックとテーブル、ボロッとアームからなる限定された世界である。

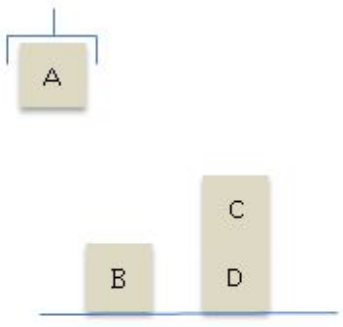


図 2.6: ブロックの世界の一例

ブロックの世界は、人工知能分野では過去何度か実験の対象に選ばれてきた。その中[1]ではWinogradのSHRDLUという、自然言語理解に関する研究が最も有名である。Winograd

のブロックの世界では、ブロックの他にプラミッドや箱等もあり、「一番大きいブロックを箱にいれよ」といった命令を解釈実行したり、「ブロックをピラミッドの上に載せよ」といった命令に対し、「ピラミッドの上に載せることはできません」と返答してきたりもする。

さて、ブロックの世界などを対象としたプランニングを行うシステムは、1970 年代から、数多く作られているが、その多くは目標状態から初期状態に至る道筋を探索するアルゴリズムを採用している。そうしたプランニングシステムに必要な機能として、次の 4 項目が挙げられいる。

1. 次に適用すべき最適の手続きの選択

   現在の状態と目標とされる状態をなんらかの形で記述して、その差を縮めるような手続きを選択する。

2. 手続きの適用と結果の計算

   各手続きを、手続きが適用されうる状態、手続きの適用により世界から消去される状態記述、手続きの適用により世界に追加される状態記述の 3 つの要素と一緒に記述しておくことで達成される。例えば、ロボットアームがブロックを掴むには、「ブロックの上に何も置かれていない」「ロボットのアームが空いている」という前提状態が必要である。ブロックを掴み上げる手続きを実行すれば、「ロボットのアームが空いている」という記述や掴み上げられたブロックと他のブロックとの位置関係は消去され、「ロボットのアームは、あるブロックを掴んでいる」という状態記が追加される。

3. ゴールが達成されたことの検出

   最初のゴール集合と状態検出とのマッチングが成功すれば、ゴールは達成されたものとみなす。

4. 不適当な道筋を検出して切り捨てる

   手続きの適用により矛盾した状態が生成された場合や、もとのゴールに戻ってしまった場合を検出する。矛盾した状態とは、一つのブロックの上に 2 つのブロックが載っているとか、ロボットのアームが掴んでいるブロックがテーブルの上にもあるというような場合である。

図 2.7 のようなブロックに対して、「ブロック C をテーブルの上に、ブロック B をブロック C の上に、ブロック B をブロック C の上に、ブロック A をブロック B の上に移動せよ」という事例を考える。ず 3.2 の左側が初期状態で、右側が目標状態ということになる。まず問題を「ブロック C をテーブルの上に移動する」、「ブロック B をブロック C の上に移動する」、「ブロック A をブロック B の上に移動する」という三つの副問題に分けよう。すると既にブロック A はブロック B の上にあるので、「ブロック C をテーブルの上に移動する」と「ブロック B をブロック C の上に移動する」の方を解決することにす

る。ブロックCをテーブルの上に移動するのは簡単である。しかし、ブロックBをブロックCの上に移動するには、まずブロックBをつかまなければならず、そのためにはブロックAを一旦他へ移さなければならない。この時点で最初に解決していたはずの「ブロックAをブロックBの上に移動する」という副問題が未解決状態になってしまった。これが副問題の相互干渉の一例である。

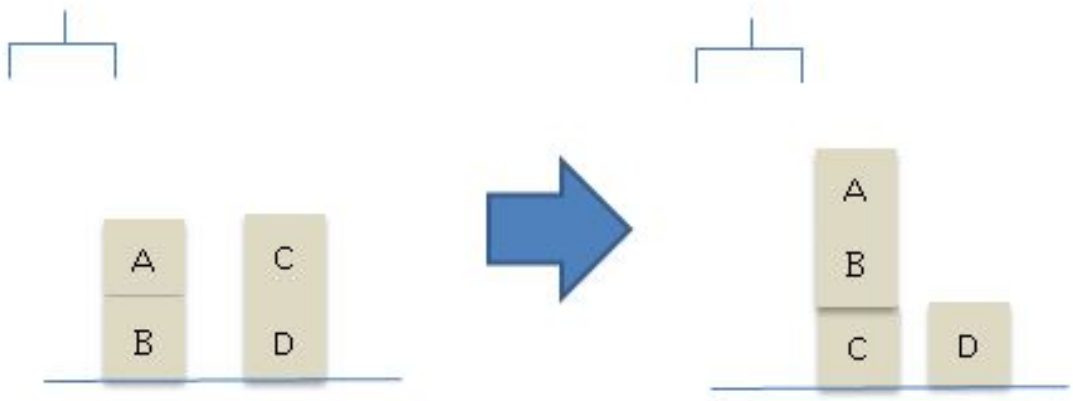


図 2.7:

2.3.2 目標スタックを用いたプランニングシステム

以上のような、複数の目標間の相互干渉を解決する手法の一つとして考え出されたのがこの章で紹介する目標スタックを用いたプランニングシステム、STRIPSだる。STRIPS自体は、1971年にFikesが提案した問題解決システムである。このころは、様々な問題に対し共通に適用することのできる「一般的な問題解決手続き」の研究がまだ盛んだったころで、STRIPS自体も、ブロックの世界だけでなく様々な領域の問題に共通に使うことのできるプラン生成アルゴリズムとして作成された。

作用素

様々な問題領域に対応するため、 STRIPSでは図3.3に挙げるような作用素（オペレータ）を用いて問題に適用できる個々の手続きを記述する。条件リストは、問題空間の現在の状態が条件リストとマッチすれば、その作用素が適用可能であることを示している。削除リストは、その作用素を適用したときに問題空間の記述集合から取り除かれる記述である。同様に追加リストは、その作用素を適用したときに問題空間の記述集合に追加される記述である。プランニングシステムは、現在の問題空間の状態の状態記述と目標とされる問題空間の状態記述との差を埋めるための作用素の系列を見つけ出すシステムと言い替える事ができる。今回のシステムは、次のようなアルゴリズムをとっている。

1. 目標とされる問題空間の状態記述にある記述を、目標スタックの初期状態として設定する。この目標スタックが空になるまで、2、3 の手続きを繰り返す。

2. 目標スタックの先頭から、ブロックの世界の状態記述とマッチする記述を取り除く。

3. ブロックの世界とマッチしない記述に出会ったら、その記述を追加リストの要素として持つ作用素を見つけ出し、その作用素の条件リストを目標スタックの一番上に付け加える。

stack(X,Y)

条件リスト：clear(Y) and holding(X)
削除リスト：clear(Y) and holding(X)
追加リスト：armempty and on(X,Y

unstack(X,Y)

条件リスト：on(X,Y) and clear(Y) and armempty
削除リスト：on(X,Y) and armempty
追加リスト：holding(X) and clear(Y)

pickup(X)

条件リスト：clear(X) and ontable(X) and armempty
削除リスト：ontable(X) and armempty
追加リスト：holding(X)

putdown(X)

条件リスト：holding(X)
削除リスト：holding(X)
追加リスト：ontable(X) and armempty

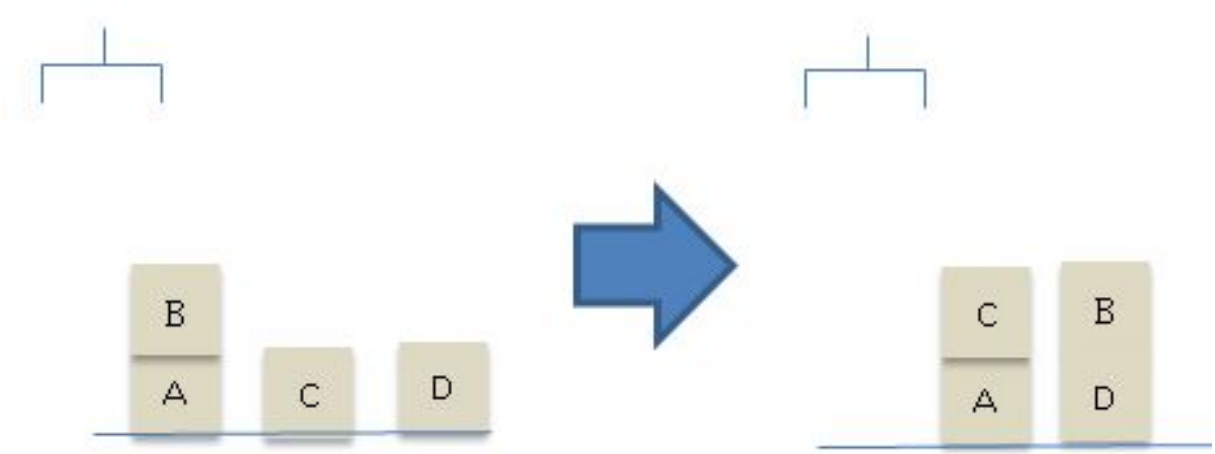
stack(X,Y)： ブロック Y の上に X を積む
unstack(X,Y)： ブロック Y の上からブロック X を取る
pickup(X)： ブロック X をテーブルの上から取る
putdown(X): ブロック X をテーブルの上に置く
clear(X)： ブロック X の上に何もない
holding(X)： ロボットの腕がブロック X を持っている
on(X,Y)： ブロック Y の上にブロック X がある
ontable(X)： ブロック X はテーブル上にある
armempty： ロボットの腕は何も持っていない

アルゴリズム

具体的な例を用いて、そのアルゴリズムを追ってみよう。アルゴリズムが複雑なため、実際に次の節でインプリメントするプログラムの実行経過に沿って説明する方が分かりやすいであろう。図 2.8 の左側がブロックの世界の初期状態で、右側が目標状態であるとしよう。作成されるべきプランのリストは

unstack(b,a)
stack(b,d)
pickup(c)
stack(c,a)

となるはずである。



初期状態：
on(b,a)
ontable(a)
ontable(c)
ontable(d)
clear(b)
clear(c)
clear(d)
armempty

目標状態：
on(c,a)
on(b,d)
ontable(a)
ontable(d)

図 2.8: 初期状態と目標状態

目標スタックの初期状態は、

on(c,a) and on(b,d) and ontable(a) and ontable(d)

となっている。すなわちブロックcがブロックaの上に、ブロックbがブロックdの上に、ブロックaとブロックdはテーブルの上にある状態である。ブロックの世界の初期状態は

on(b,a) and
ontable(a) and
ontable(c) and
ontable(d) and
clear(b) and
clear(c) and
clear(d) and
armempty

と記述することができる。目標スタックの一番上は4つの単一ゴールがand結合したものだが、このゴールは成立しない。その場合はand結合したゴールのうち成り立っていない部分を、目標スタックの上に積む。この場合

ontable(a)
ontable(b)

の2つは既に成り立っているので、新しいスタックは

on(c,a)
on(b,d)
on(c,a) and on(b,d) and ontable(a) and ontable(d)

となる。ここで、一つの副目標が達成が、既に達成されたはずの他の副目標の達成を取り消してしまう場合を検出するために、and結合したゴールが設定される。例えば最初に
on(c,a)
を解決して、次に
on(b,d)
を解決したときに
on(c,a)
が再び未解決の状態に戻ってしまった場合を、このand結合ゴールで検出することができるのである。
次にスタックの一番上の記述「ブロックcはブロックaの上にある」＝ ”on(c,a) ”を、追加リストの要素とする作用素を選択する。条件に当てはまる作用素は、「ブロックcをブロックaの上に積む」という意味の
stack(c,a)
だけである。そこで、stack(c,a)の条件リスト

clear(a) and holding(c)

で、on(c,a) を置き換える。新しい目標スタックは clear(a) and holding(c)

stack(c,a)
on(b,d)
on(c,a) and on(b,d) and ontable(a) and ontable(d)

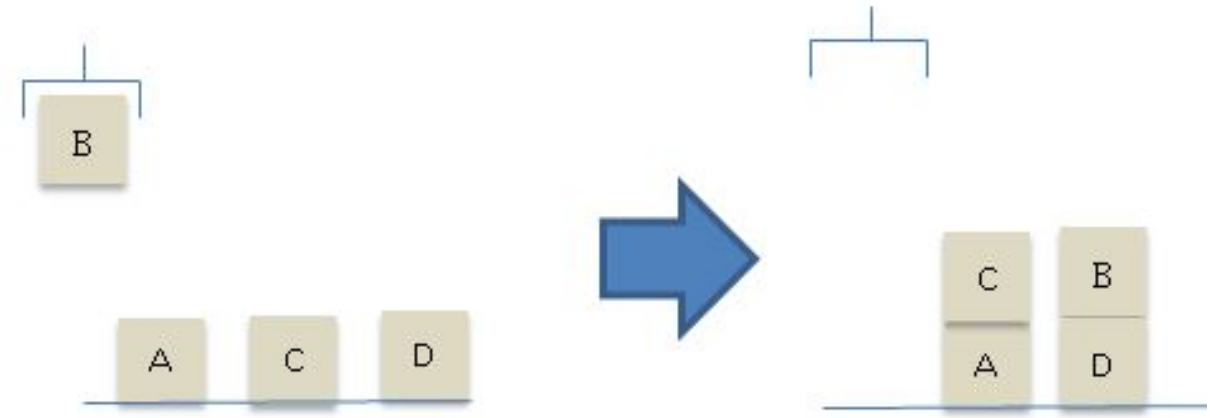
となる。次に複合ゴールが展開されて、目標スタックは

clear(a)
holding(c)
clear(a) and holding(c)
stack(c,a)
on(b,c)
on(c,a) and on(b,d) and ontable(a) and ontable(d)

となる。この展開の順番は、一般的に成立させやすい目標から順にスタックに積んでいく。現在のスタックの一番上の記述である clear(a) は成立していないので clear(a) を追加リストとして持つ作用素 unstack(b,a) を選択し、先ほどと同様に積み上げていく。新しい目標スタックは

on(b,a)
clear(b)
armempty
on(b,a) and clear(b) and armempty
unstack(b,a)
holding(c)
clear(a) and holding(c)
stack(c,a)
on(b,d)
on(c,a) and on(b,d) and ontable(a) and ontable(d)

となる。スタックの上から 3 つの記述と 4 つ目の and 結合したゴールは、全て成立するのでスタックから取り出せる。するとスタックの一番上は unstack(b,a) となる。スタックの一番上が作用素となった時は、その作用素を次際に適用する。この例ではこれが最初の作用素の適用となっている。unstack(b,a) を実行することで、すなわち unstack(b,a) の削除リストを取り除き追加リストを加えることで、ブロックの世界の記述は図 2.9 のようになる。



初期状態：
ontable(a)
holding(b)
ontable(c)
ontable(d)
clear(b)
clear(c)
clear(d)
armempty

目標状態：
on(c,a)
on(b,d)
ontable(a)
ontable(d)

図 2.9: unstack(b,a) を実行したところ

この時点での目標スタックは

holding(c)
clear(a) and holding(c)
stack(c,a)
on(b,d)
on(c,a) and on(b,d) and ontable(a) and ontable(d)

となっている。目標スタックの一番上の holding(c) は成り立っていない。そこで holding(c) を成り立たせるような作用素をさがす。holding(c) を追加リストとして持つ作用素は、pickup(c) と unstack(c,X) の 2 つがある。適用可能な作用素が 2 つ以上ある時は、競合解消戦略を用いて、その後有利になる作用素を選択しなければならない。今回インプリメントしたシステムは、この競合解消を領域に依存したルールによって行う。元となったシステムでは、領域に依存しないアルゴリズムを用いていたが、そのためにシステムが複雑になり、融通が利かなくなっている。今回は、この競合解消ルールをプランニングシステムの適用領域に合わせて作成すれば、細部にわたって最適化を行うことができる。

さて、pickup(c) と unstack(c,X) の競合解消戦略はごく簡単なもので、ブロック c がテーブルの上に載っていれば pickup(c) を選択し、ブロック c が他のブロック X の上に載っていれば unstack(c,X) を選択するというものである。今回の場合、ブロック c はテーブル上にあるので、pickup(c) が作用素として選択される。この時点での目標スタックは

ontable(c)
clear(c)
armempty
ontable(c) and clear(c) and armempty
pickup(c)
clear(a) and holding(c)
stack(c,a)
on(b,d)
on(c,a) and on(b,d) and ontable(a) and ontable(d)

となる。スタックの上から 2 つのゴールは、この時点で成立しているので取り出せる。スタック一番上は armempty となり、今度は armempty を追加リストとして持つ作用素を選択することになる。その条件を満たす作用素には stack(b,X) と putdown(b) の 2 つがある。stack(b,X) と putdown(b) の競合解消ルールは、初期目標に on(b,X) というものがあれば stack(b,X) を選択し、なければ putdown(b) を選択する、というものである。この例では on(b,d) が初期目標として設定されているので、X を d とした stack(b,d) を作用素として選択する。この時点での目標スタックは

clear(d)
holding(b)
clear(d) and holding(b)
stack(b,d)
ontable(c) and clear(c) and armempty
pickup(c)
clear(a) and holding(c)
stack(c,a)
on(b,d)
on(c,a) and on(b,d) and ontable(a) and ontable(d)

である。あとは、目標スタックが

on(b,d)
on(c,a) and on(b,d) and ontable(a) and ontable(d)

となるまで stack(b,d)、pickup(c)、stack(c,a) を実行していく。この時点で on(b,d) も armempty を成立させるためにとった stack(b,d) という作用素の副作用で成立しているので、on(b,d) も取り除くことができる。最後に残った

on(c,a) and on(b,d) and ontable(a) and ontable(d)

もう一つの目標を達成するために他の目標を崩していなければ成立しているはずである。今回は、たまたまそういった干渉がなかったので、最後の複合ゴールも成立しスタックから取り除かれる。
目標スタックが空になった時に、プランニングが終了したと判断され、それまでに実行された作用素がプランとして提出される。この例でプランは

unstack(b,a)
stack(b,d)
pickup(c)
stack(c,a)

第3章　3-BLOCKSパタンによるブロック世界の問題を分割するモデル

本章では、動的な環境に対応するシステム中の最も中心となる部分「3-BLOCKSパタンによるブロック認識仕組み」を紹介する。古典的な方法では、ブロック世界の全て可能な組合せを探索空間とし、前向き方法もしくは後ろ向き方法で問題を解決していくので、ブロック数が多い場合に、計画中のオペレータ同士干渉が既に第二章に述べたように考慮済みが、実際に探索の効率が大幅に低下することになる。大規模のブロック世界プランニング問題で、プランニングは最悪の状況で、プランニング時間は指数的に増加する。その膨大な時間の増加にならないように、大規模の問題を小規模の問題に分割する思想に踏まえ、パタン認識の機能を導入する。パタン認識の導入するため、様々な盲目的に動作を試すことを最小限に抑え、特に大規模の問題に役に立つと予想する。
そこで、本研究では、3 - BLOCKSパタン認識を利用して、バーチャル状態空間とリアル状態空間の互い作用によって、状態空間を有機的に分離することができる。

3.1　システムの構成

本研究のシステムの設計は図 3.1 に示すように、3 - BLOCKSパタン認識部、バーチャル状態生成部、子状態分離部という三つのシステムから構成される。

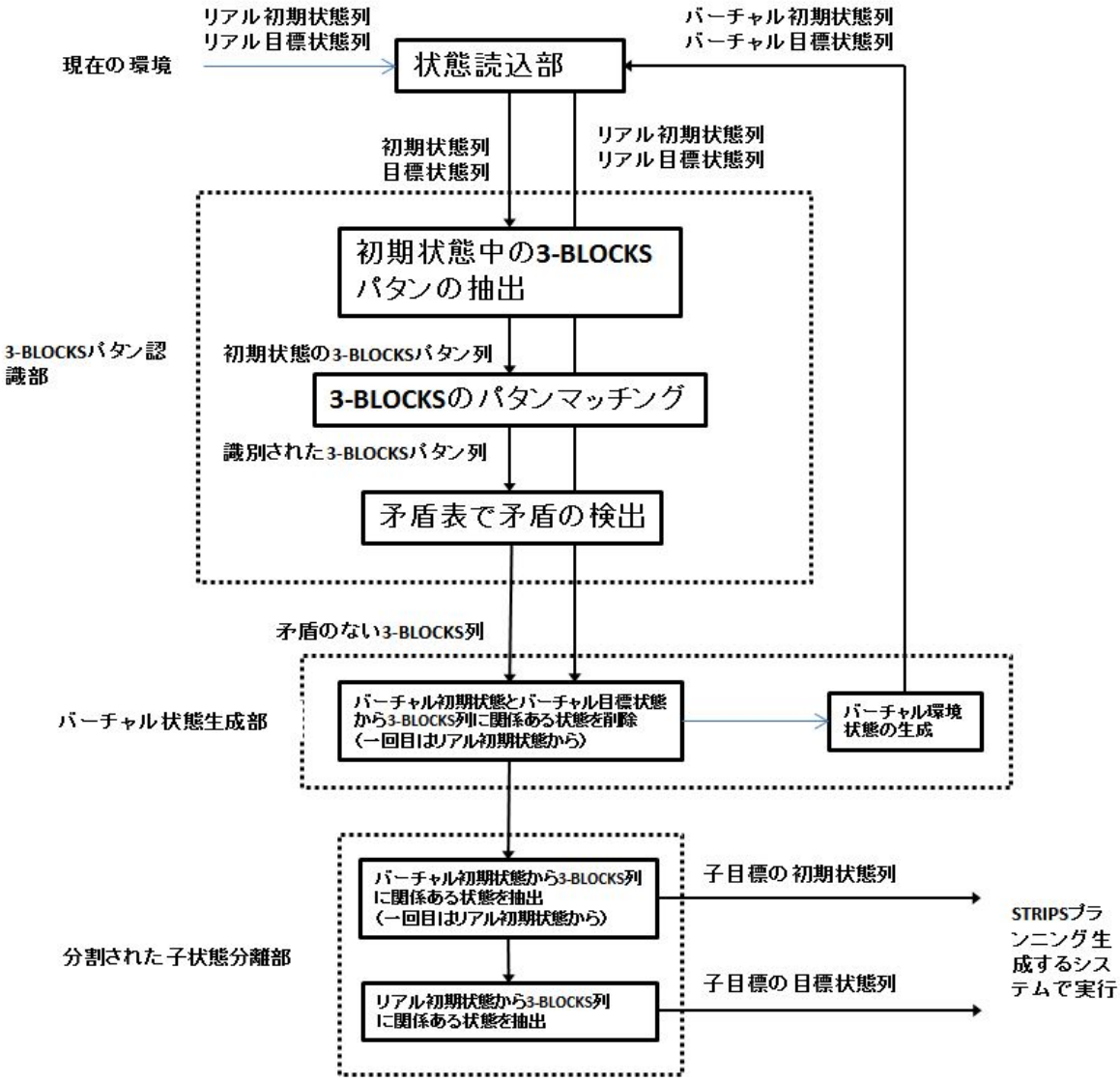

リアル初期状態列
リアル目標状態列
現在の環境
状態読込部
バーチャル初期状態列
バーチャル目標状態列
初期状態列
目標状態列
リアル初期状態列
リアル目標状態列
初期状態中の3-BLOCKS
パタンの抽出
3-BLOCKSパタン認
識部
初期状態の3-BLOCKSパタン列
3-BLOCKSのパタンマッチング
識別された3-BLOCKSパタン列
矛盾表で矛盾の検出
矛盾のない3-BLOCKS列
バーチャル状態生成部
バーチャル初期状態とバーチャル目標状態
から3-BLOCKS列に関係ある状態を削除
（一回目はリアル初期状態から）
バーチャル環境
状態の生成
バーチャル初期状態から3-BLOCKS列
に関係ある状態を抽出
（一回目はリアル初期状態から）
子目標の初期状態列
分割された子状態分離部
リアル初期状態から3-BLOCKS列
に関係ある状態を抽出
子目標の目標状態列
STRIPSプラ
ンニング生
成するシス
テムで実行

図 3.1: システムの構成

3.2　3 - BLOCKS パタン認識

3.2.1　パタン認識の仕方

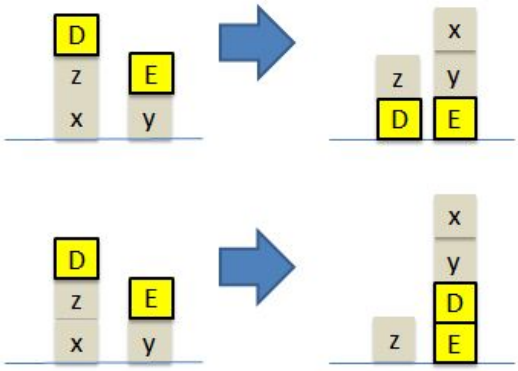


図 3.2: 同様のパタンがある例

初期状態と目標状態の中に、ブロックの組合せの同じ部分があれば、それらのパタンを最優先に移動する。そのため、必ず初期状態の一番上にあるパタンと目標状態の一番下にあるパタンでなければならない。図 3.2 の例に、初期状態の一番上にあるブロック B と E が目標状態の一番下のある場合ならば、B と E という 2 つのブロックを最優先に移動し、目標状態の一番下になる。

3.2.2　3-BLOCKS パタン

ブロックの世界において、1 個や 2 個、3 個及びそれ以上のブロックの組み合わせがあり、これらの組合せの中に、
一個ブロックの状態空間の表現は図 3.3 に示す

図 3.3: 1 個ブロックの状態空間

二個ブロックの状態空間の表現は図 3.4 に示す

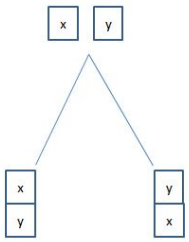


図 3.4: 3 個ブロックの状態空間

三個ブロックの状態空間の表現は図 3.5 に示す

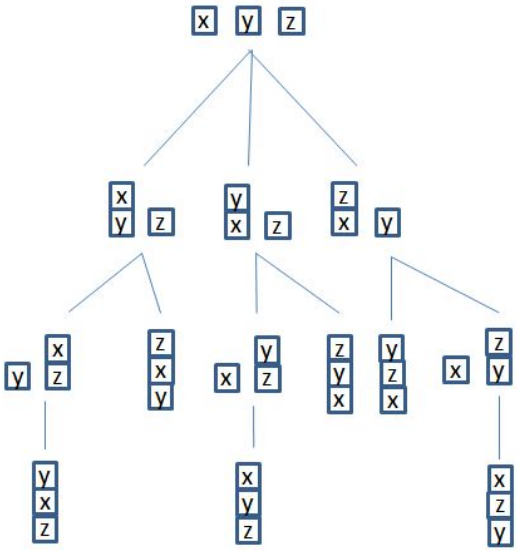


図 3.5: 一個ブロックの状態空間

一個ブロックのパタンの場合に、状態空間は大きさは 1 であるため、ブロックの世界に対してはもっとも柔軟的なパタンであるはずである。しかし、図 3.6 に示す環境状態の間に、灰色のブロックはすべて、1 個ブロックの状態空間に満足しているが、対象となる目標状態に灰色ブロックとマッチングできるブロックが一個見出すこともない。原因は、一個のブロックの状態空間は深度がただ一なので、目標状態の中に、下から二層目もブロックに至ることができない。
同じように、二個ブロックの状態空間を用いて探索すると、同じ問題が発生してしまう。三個ブロックパタンを探索すると、左の図にあるブロック B, K, x と右の図にある B, K, x をマッチングできた。

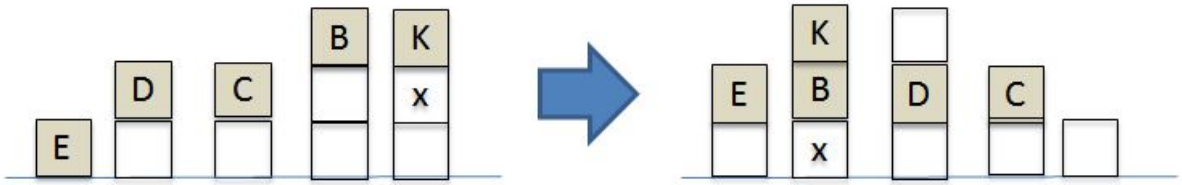


図 3.6: ブロックの状態空間でパタン認識の一例

三つのブロックの状態空間が最優である。一つのブロックの縦、横方法を全部に探索することになれる。
それ以外、四つのブロック世界も同じように、縦と横となる部分を含むことができるが、探索空間が 144 になり。そのような膨大な探索空間で探索することで、多い無駄なマッチング時間が要する、返って探索効率が低下していく。実際に、三つブロックの探索空間が四つのブロックの探索空間を含んでいる。それは図 3.7 ように示す。

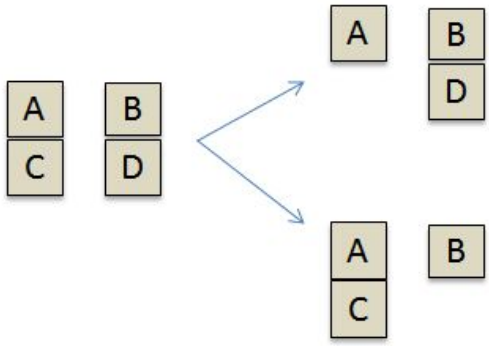


図 3.7: 4 つブロックパタンと 3 つブロックパタンの関係

上に述べたように、本研究で、一つのブロック周囲に接しているブロックを万遍する最もコンパクトな状態空間は 3-BLOCKS 状態空間である。そして、パタン認識を三つブロックからなる 3-BLOCKS パタンで認識を行う。

3.2.3 3-BLOCKS マッチング部

本節で、3-BLOCK マッチング部について説明する。ブロック世界には、初期状態となるブロックの組み合わせに一番上になる三つブロックの組み合わせは下の三つの式で記述する。

clear(x) and clear(y) and clear(z)
clear(x) and clear(y) and on(z,x)
clear(x) and on(x,y) and on(y,z)

目標状態となるブロックの組み合わせに一番上になる三つブロックの組み合わせは下の三つの式で記述する。

ontable(x) and ontable(y) and ontable(z)
ontable(x) and ontable(y) and on(z,x)
ontable(x) and on(x,y) and on(y,z)

もし初期状態の中に、初期状態の組み合わせ式の中の三つの式をいずれか満足すれば、目標状態に、目標状態の組み合わせ式の中にいずれか満足するならば、そのパタンマッチングが成功した。計算機中のマッチング部の機能をより直観的に表現すると、図 3.11 のように示す。

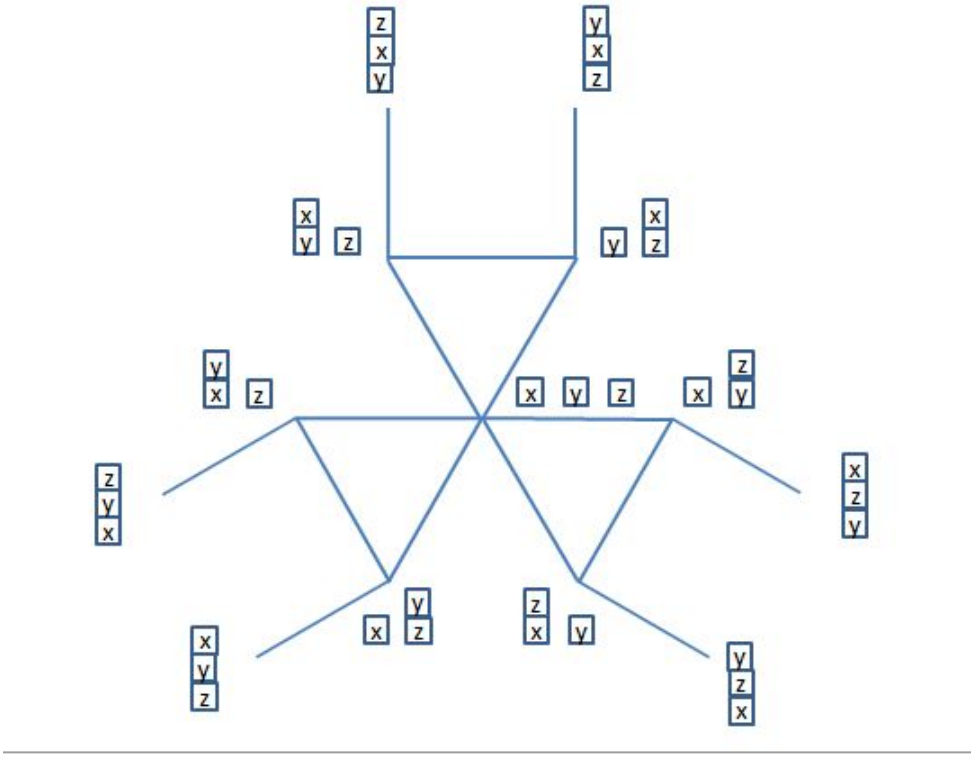
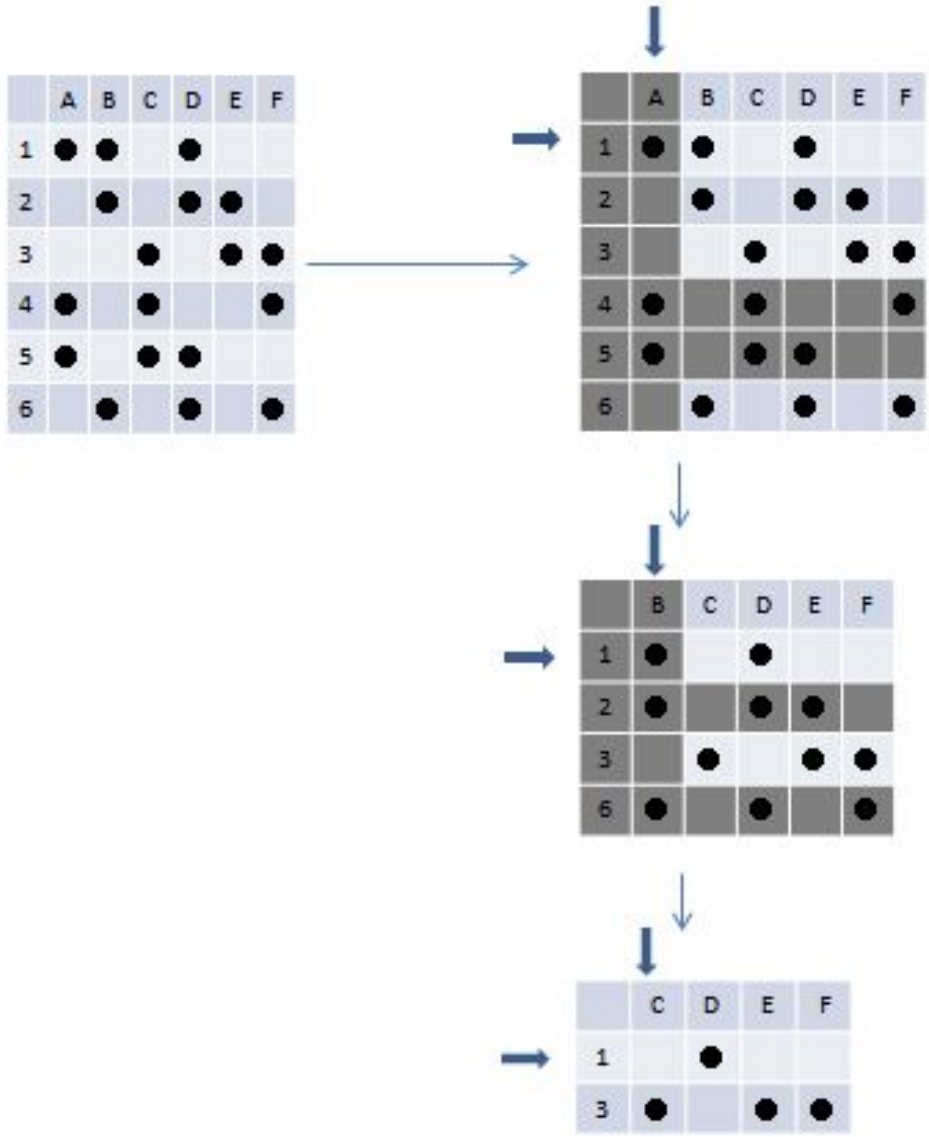


図 3.8: コンパクト的な 3 個ブロックの状態空間の表現

3.2.4 矛盾表による矛盾の検出

10 個以下のブロックの世界には、2 つ以上の 3-BLOCKS パタンを検出することが少ない、さらに、10 個以上ブロックの環境では、2 つ、3 つ、さらに 100 つ以上の 3-BLOCKS パタンを検出したことも多いのである。これらのパタンの間に、共有した部分があるし、

もしくは、すべて違うパタンもある。効率的なプランニングシステムを構築するため、一回でできる限りに多くのブロックをパタン毎に移動することを考慮する。そのあめ、矛盾検出機能を導入する。
矛盾検出機能のアルゴリズムは、「エイト・クイーンパズル」のアルゴリズムを簡略化されてきたものである。「エイト・クイーンパズル」においては、横、縦及び斜めにクイーンを置くことができない点から発想し、、矛盾表で、ただ縦だけ「クイーン」を置くことができない。しかし、基盤の大きさは 8X8 の正方形が、矛盾表の横と縦の長さが決められていないため、縦の方向から優先に見ていくことにする。図 3.9 に示したように、
図 3.9 は、仮に１～６までのパタンを「3-BLOCKS 認識方法」によって識別され、
3-BLOCKS パタン部で生成されたパタンの列を「矛盾の検出部」に導入された様子である。まず、1 番目のパタンから矛盾を検出しはじめる。A 列の中に、矛盾が発生してしまうパタンはパタン 4 とパタン 5 である。そのため、パタン 4 とパタン 5 を表から削除し、その上に、第一列を削除する。新しい表が生成されて、再帰アルゴリズムによって、再びに矛盾検出しはじめ、同じように、矛盾が発生するパタンはパタン 2 とパタン 6、同様に、パタン 2 とパタン 6 及び第一列を削除し、矛盾検出を再開し、矛盾がなくなるまで終わる。その例で、上述の再帰アルゴリズムで検出された矛盾がないパタンはパタン１とパタン 2 である。
そして、パタン 1 に基づく矛盾検出が終了し、併せて 2 つ矛盾がないパタンを検出した。そして、パタン 1 の一行をまるごとに削除し、パタン 2 を一番上にし、矛盾検出部に導入され、同じように再帰アルゴリズムで実行し、最後に得られた矛盾のないパタンは 1 つだけである。全てのパタンの矛盾検出が終了すると、最も多い数のパタンの列を移動できるブロック群れとして定義する。これらのパタンにあるブロックのネームを次の節に紹介する「バーチャル状態生成」により処理し、本来のリアル状態からバーチャル状態を派生することによって処理される。



パタン 1 に基づく矛盾検出の結果：1、3 のパタンは矛盾がないことを検出した。

図 3.9: 再帰アルゴリズムで矛盾検出の実行例

3.3　バーチャル状態生成部

ブロックの世界のプランニングの認識の性能を向上するために、一回目 3-BLOCKS パタン認識することで終わるわけではなく、ブロックの移動することに伴い、新たなブロックが一番上に現し。新しく現れた上面のブロックと、新しい一番下にあるブロックの中に 3-BLOCKS パタン認識を再開するために、変化されたブロックの状態を直接に読み込んではなく、必ず真実の環境と異なり、しかし関連があるバーチャル環境を生成しなければならない。そのバーチャル状態、いわゆる仮状態、を生成するには、図 3.11 ような例を持って示す。

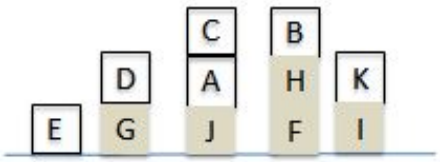


図 3.10: パタン認識されたのブロック（白色のブロック）

図 3.11 の中に、パタン認識によって認識されたブロックを白色で塗りつぶし、それらのブロックを最優先に移動する。そして、移動された後、白いブロックをテーブルの上に置いてある状態であるけれども、この新しいブロックの世界の状態をそのままに新たな状態として受け入れることをしない。理由は、認識されたブロックをテーブルの上に置いてあるが、次にパタン認識するときに、それらのすでに認識されたパタンを再認識することになり、システムは無限のロープになってしまい、正しいパタン認識に干渉してしまう。そうならないため、バーチャル状態の生成機能が要する。

図 3.11: 目標状態からバーチャル状態の生成の例

バーチャル状態生成部の役は、すでに識別されたパタンと識別されていないパタンを分離することである。初期状態に対しては状態分離が簡単で、ただ上のブロックを消し、バーチャルの状態が生成する。しかし、目標状態の状態分離には時間がかかる。原因は、ブロックを消すことだけではなく、「重力」の原因で下のブロックが消すと、上のブロックが沈んでいくということが発生する（図 3.11）。そして、バーチャル状態生成することに、重力を考えに含め、流れは図 3.12 に示す。

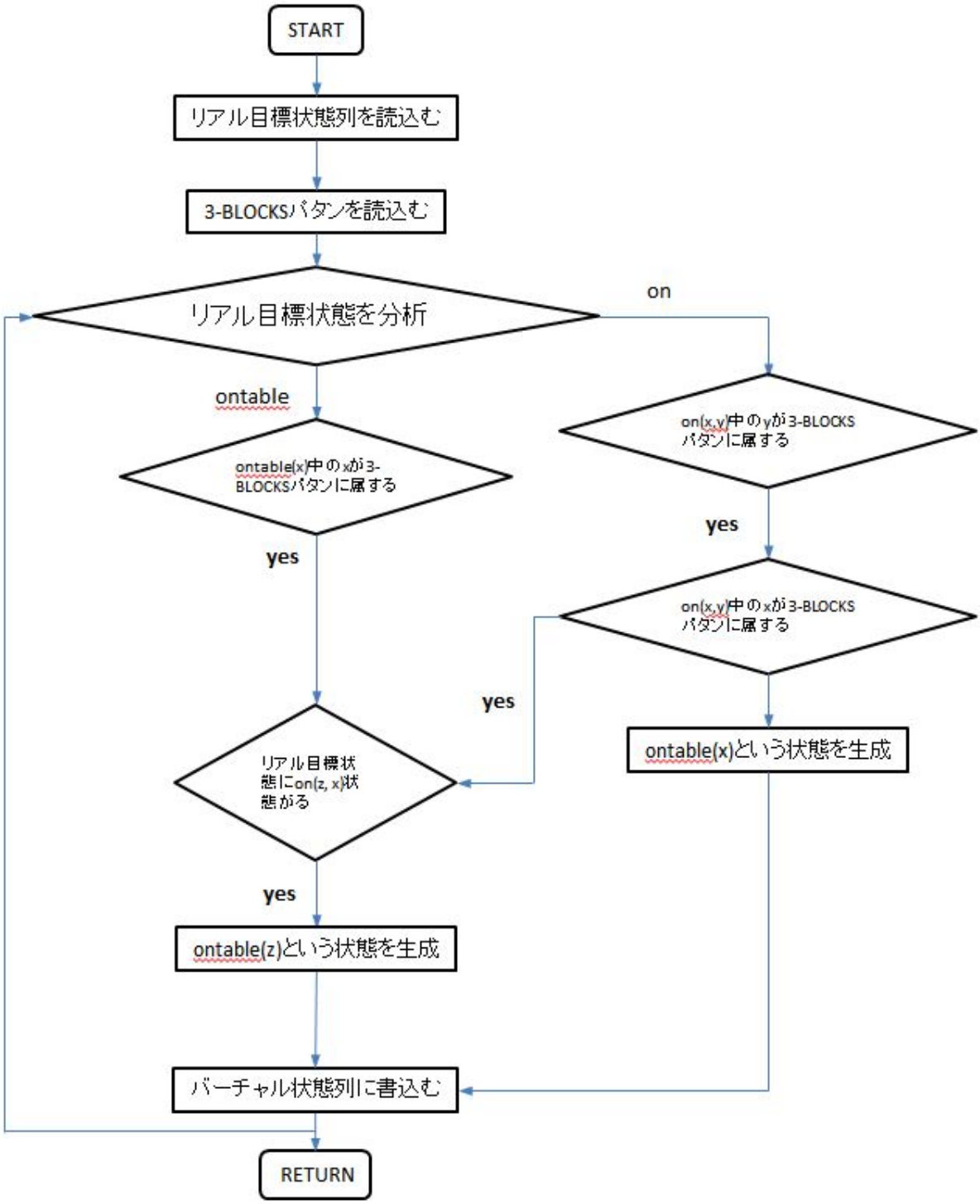
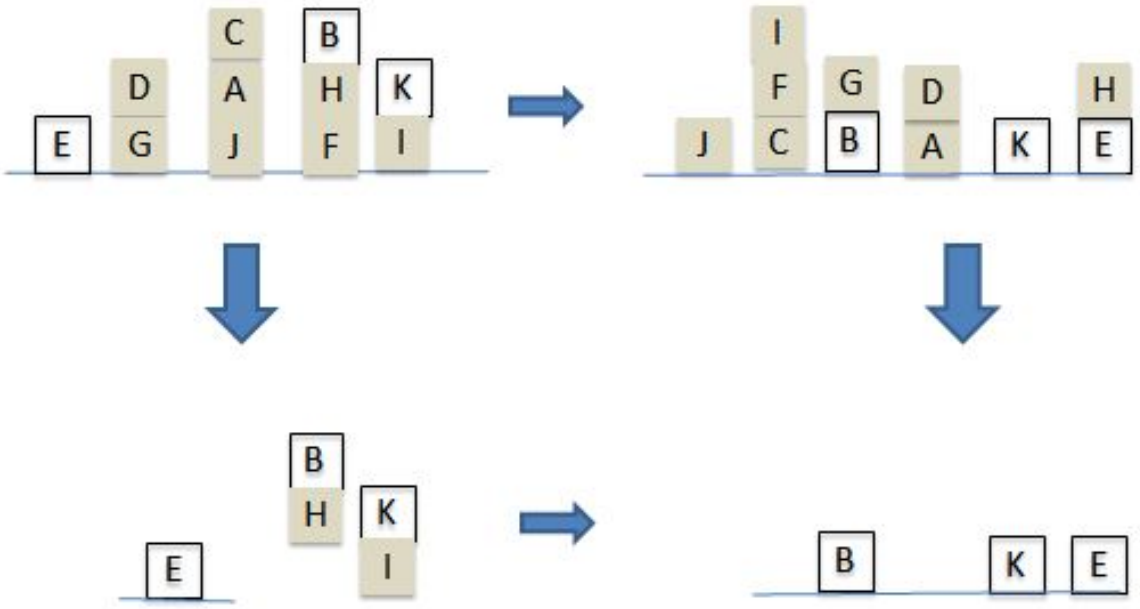


図 3.12: バーチャル目標状態の生成ルーチン

3.4　子目標を分離部

本システムの重要な出力の一つは、STRIPS の書式のファイルである。前節で紹介した「バーチャル状態生成部」との違いのは、「バーチャル状態生成部」のやり方は「引き算」だが、子目標の分離部は「足し算」で実行する。
STRIPS の特徴は、局所プランニングができることである。言い換えると、必要だけの情報を STRIPS に教えるだけで十分である。たとえば、図 3.13 の示したように、



子初期状態：
ontable(E)
clear(E)
on(B,H)
clear(B)
on(K,I)
clear(K)

子目標状態：
ontable(E)
ontable(K)
ontable(B)

図 3.13: バーチャル目標状態の生成ルーチン

図 3.13 の中に、白色のブロック E, B, K が前節に述べたパタン認識による識別されて、そのパタンをどのようなオペレータの系列で、目標状態の中のブロック B, K, E の状態に到達することを計算する前に、これらのブロックの状態を STRIPS プランニングシステムに告知しなければならない。もちろん、初期状態を丸ごとに STRIPS に告知することは古典的な方法で実行し、その中にブロック B, K ,E に関係があるオペレータを抽出するのは楽だが、その方法は実際に意味がなく、古典的な方法と違いがないから。その問題を解決するため、ブロック B, K ,E と関係があるブロックだけを抽出し、STRIPS による実行されるのは、実行の時間を節約する。

子目標の分離部の中に、中心となる思想は、3-BLOCKS パタン認識による識別されたブロックとこのブロックのしたに接しているブロックだけの状態を、リアル初期状態とリアル目標状態から抽出する。例としては、図 3.13 の子目標の分離部に抽出されたブロックの状態を STRIPS に入れて実行されると、次のオペレータの系列を生成した:

unstack(B,H)
putdown(B)
unstack(K,I)
putdown(K)

第4章　評価実験

4.1　評価の目的

本研究に対して、評価実験を実行する。任意のブロック世界の環境を「3-BLOCKS パタンによるブロック世界の分割するシステム」に与えるとき、そのブロック世界の
その評価の目的は、3-BLOCKS パタン認識によって、現在の状態空間の認識率や、分離された子状態の総実行時間と古典的な方法を比較し、時間の節約率を算出する。

4.2　評価の方法

「3-BLOCKS パタンによるブロック世界の問題を分割するシステム」を C 言語でプログラミングし、可視化するために、ブロック名を一文字で表示ことにする。そのため、すべてを ASCII の印字可能文字を利用して、その中に、WINDOWS システムのファイル名として使えない文字を除去した後、併せて 80 個、そして、多くとも 80 個のブロックの世界の生成することを実現した。
STRIPS は、第 2 章のプランニングアルゴリズムに参照し、PROLOG 言語でプログラミングする。その上に、ループ検証機能を導入した。そして、プランニングに無駄なステップを最小限に抑える。無限ループのない PROLOG のプログラムを作った。その中に記述しているオペレータ記述は以下のようになる：

stack(X,Y)

条件リスト：clear(Y) and holding(X)
削除リスト：clear(Y) and holding(X)
追加リスト：armempty and on(X,Y

unstack(X,Y)

条件リスト：on(X,Y) and clear(Y) and armempty
削除リスト：on(X,Y) and armempty
追加リスト：holding(X) and clear(Y)

pickup(X)

条件リスト：clear(X) and ontable(X) and armempty
削除リスト：ontable(X) and armempty
追加リスト：holding(X)

putdown(X)

条件リスト：holding(X)
削除リスト：holding(X)
追加リスト：ontable(X) and armempty

子状態は3-BLOCKSパタンによる生成され、子目標の初期状態と目標状態をPROLOGファイルに書込まれた。

システムの評価は実行時間の計測することでする。C言語での3-BLOCKSパタン認識の総時間とPROLOGでプランを生成することにかかる総時間を足し算で計算する。

4.2.1 ランダムブロック世界の発生器

図 4.1: ランダム初期状態の例

図 4.2: ランダム目標状態の例

現在の環境について、ブロック世界の3-BLOCKSパタン認識の実験のせいかく率を向上するため、ランダムブロック環境生成システムでランダムにブロックの世界を生成する。生成可能なブロック世界のブロック数は5個から80個までである。生成されたブロック環境に、一列で重ねることと一行でテーブルの上に並べることにならないように、ブロック世界を発生するプログラムで、ブロック世界のテーブルと接する幅N_Bを一定の範囲にしなければならない。I_Bはユーザが設定したブロックの総数

$$N_B = \lambda I_B (0.2 \leq \lambda \leq 0.8)$$

本研究の実験は、10個から80個までのブロック環境で各5回実行することにし、ブロック環境のブロック数は10個ごとに増える。併せて40回の実験をする。

4.2.2 実験システムの設計

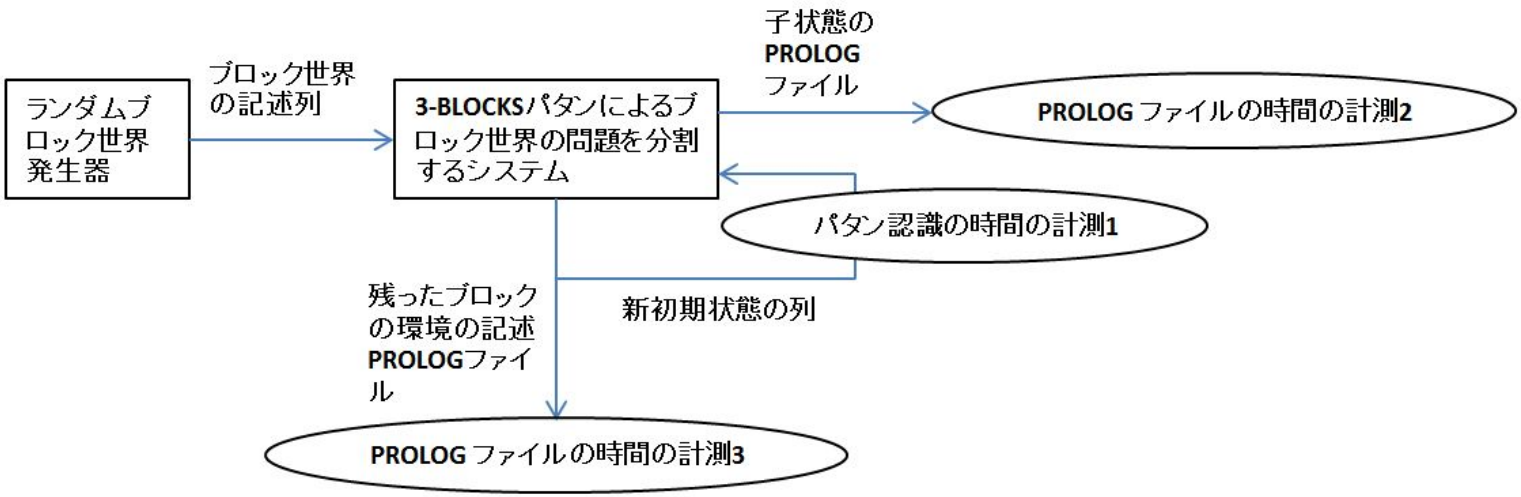


図 4.3: 評価システムの設計

4.3 実験

4.3.1 認識率

3 - BLOCKS パタン認識の認識率とは、ブロックの総数の中に、併せてどれほどブロックがパタン認識されることである。これは、本研究のシステムに置く一番重要な参考量である。R_B は識別されたブロックの総数。表 4.3 は認識率計算式により計算した 10 個〜80 個ブロックの世界の認識率である。認識率 f の計算式は：

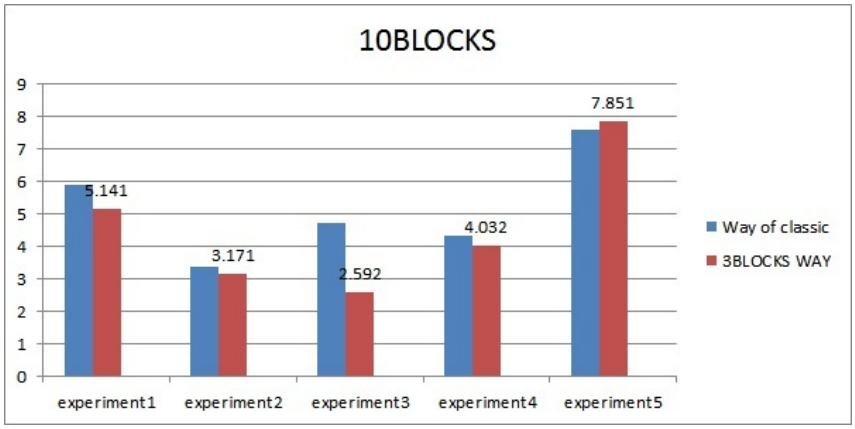
$$f = \frac{R_B}{I_B} \times \ 100\%$$

	実験 1	実験 2	実験 3	実験 4	実験 5
10BLOCKS	30.0%	30.0%	90.0%	90.0%	0.0%
20BLOKKS	90.0%	90.0%	90.0%	90.0%	30.0%
30BLOCKS	20.0%	70.0%	40.0%	100.0%	20.0%
40BLOCKS	97.5%	7.5%	97.5%	60.0%	97.5%
50BLOCKS	100.0%	96.0%	12.0%	42.0%	90.0%
60BLOCKS	100.0%	25.0%	100.0%	20.0%	100.0%
70BLOCKS	34.3%	98.6%	94.3%	60.0%	12.8%
80BLOCKS	97.5%	97.5%	97.5%	41.3%	97.5%

表 4.1: 10 個〜80 個ブロック世界に置く 3-BLOCKS パタンの認識率

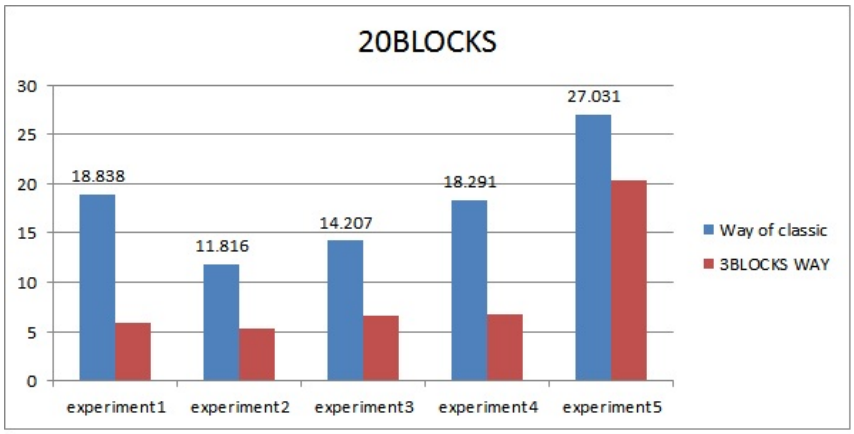
4.3.2 プランニング時間

図 4.4〜図 4.11 は 10〜80 個ブロック世界の実行時間である。図の中に、比較として、5 回の同じブロック数のブロック世界の古典の方法と 3-BLOCKS パタン認識の方法の実行時間を計測した。左の柱は古典的な方法で、右の柱は 3-BLOCKS パタン認識の方法である。



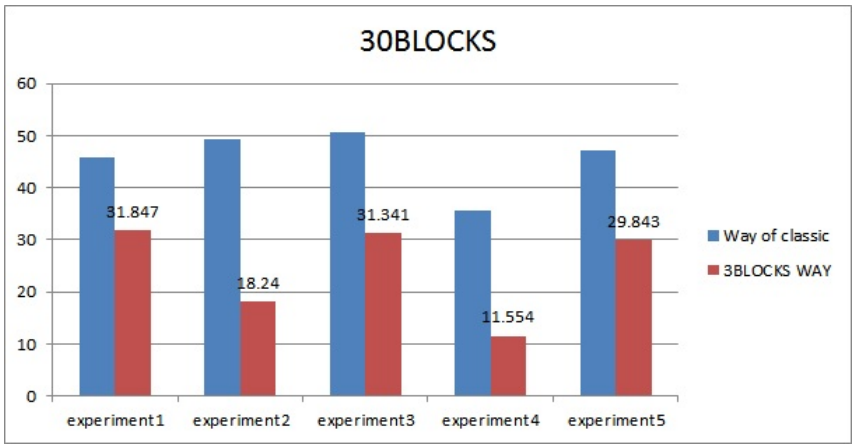
ブロック数は 10 個の場合に、3-BLOCKS パタン認識によるプランニング時間は従来のなプランニング時間とほぼ同じである。

図 4.4: 10 個ブロックのプランニング時間



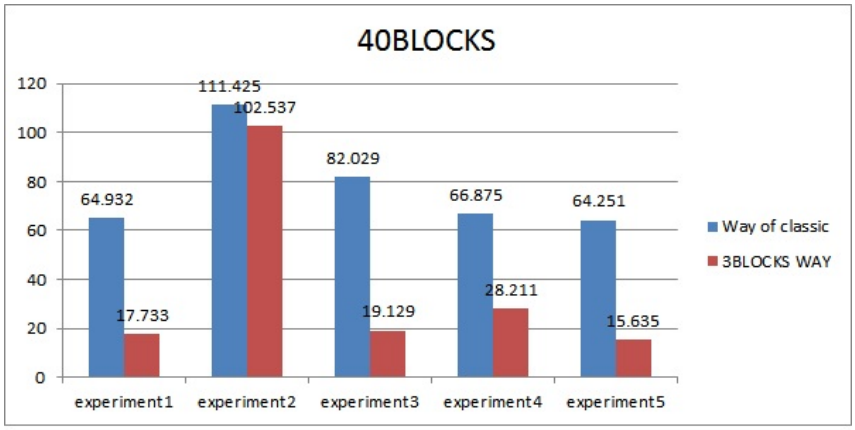
ブロック数は 20 個のとき、ブロックの認識率が高まり、プランニング時間を 20%〜80%上下に削減した。

図 4.5: 20 個ブロックのプランニング時間



30 個のブロックの時、ブロック世界の構造が複雑になるから、認識率が 20 個ブロックのときより下回った。そして、プランニングの平均の時間節約率は 20 個のときより悪くなった。

図 4.6: 30 個ブロックのプランニング時間

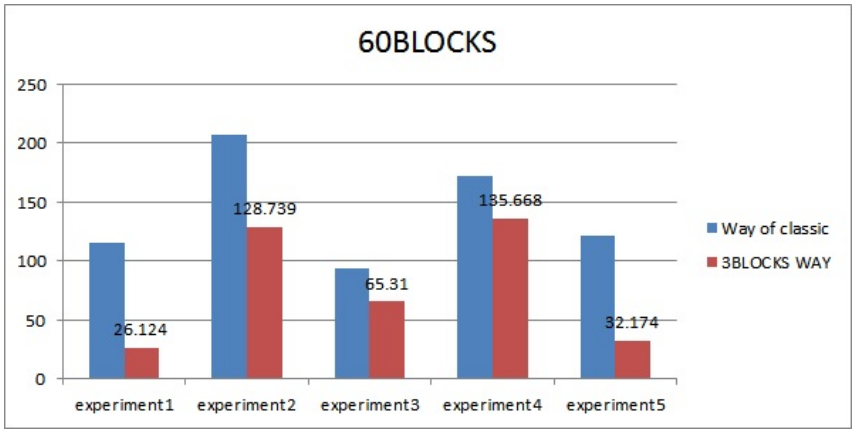


40 個のブロック世界の全体は、30 個のブロックの世界と比べると、テーブルと接するブロック数が多くなる。いわゆるブロック世界の幅が増大する。そのため、40 個のブロック世界の複雑さが高まった一方、認識率がよくなり、時間の節約率も高くなった。

図 4.7: 40 個ブロックのプランニング時間

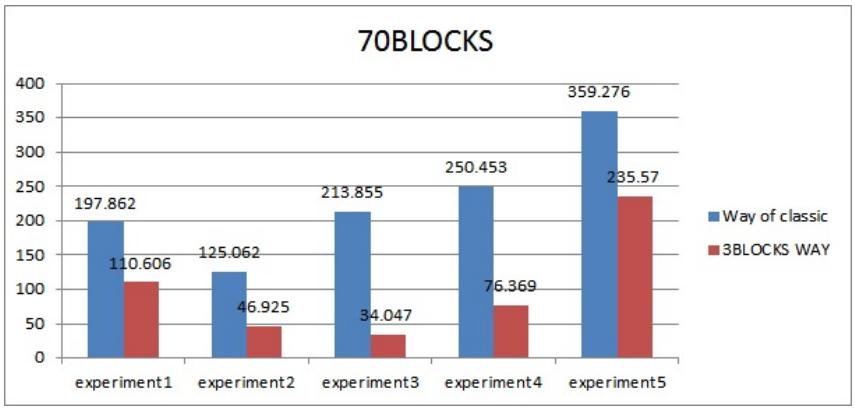
50 個の場合、認識率は 40 個のブロック世界とほぼ同じなので、時間の節約率は 40 個の場合と同じレベルである。

図 4.8: 50 個ブロックのプランニング時間



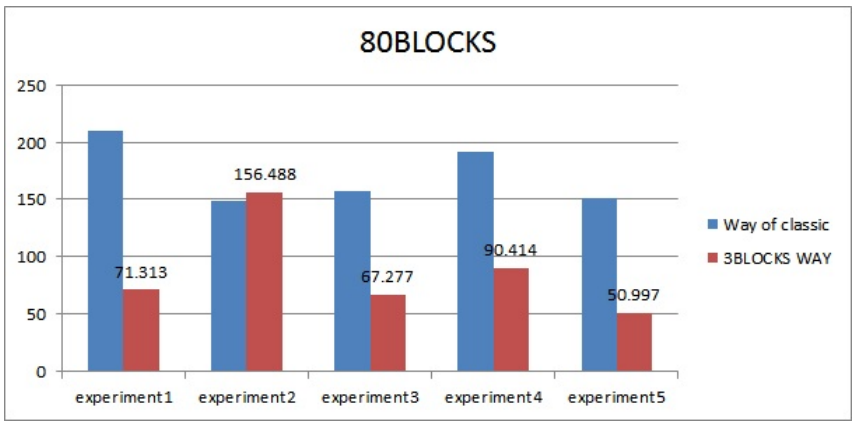
50 個ブロック世界と同じレベルの認識率が出たが、ブロックの組み合わせが複雑なので、時間節約率が 50 個の場合より少し低下した。

図 4.9: 60 個ブロックのプランニング時間



ブロックの組み合わせの複雑さの増加にしたがって、認識率は 60 個の場合より悪くなったが、時間の節約率は逆によくなった。

図 4.10: 70 個ブロックのプランニング時間



80 個のブロック世界になると、すべての実験に一番高い平均認識率を得た。しかし、ブロックの組み合わせがさらに複雑になる原因で、プランニングの平均の時間節約率は 20 個ブロックの場合と同じレベルである。

図 4.11: 80 個ブロックのプランニング時間

4.3.3 時間節約率

ブロックの数の増加に従って、認識されたパタンや、プランニングの時間も増える。古典的なプランニング方法と比べ、どれほど時間を節約したのは意味がある。下の表 4.3 は本実験で得られた古典的な方法の実行時間と 3-BLOCKS パタン認識の方法でプランニング実行時間の差を計算し、したの式で時間の節約率を計算する。

$$t_B = \frac{O_t - N_t}{O_t} \times \ 100\%$$

	実験 1	実験 2	実験 3	実験 4	実験 5
10BLOCKS	12.4%	6.0%	44.9%	6.6%	-3.9%
20BLOKKS	69.1%	54.8%	53.1%	63.4%	24.8%
30BLOCKS	30.3%	62.9%	37.9%	67.5%	36.5%
40BLOCKS	72.7%	8.0%	76.5%	57.8%	75.5%
50BLOCKS	74.7%	47.0%	21.8%	56.2%	79.5%
60BLOCKS	77.3%	37.6%	30.6%	21.0%	73.4%
70BLOCKS	44.1%	62.5%	84.1%	69.5%	34.4%
80BLOCKS	66.1%	-5.3%	57.3%	52.9%	66.4%

表 4.2: 10 個〜80 個ブロック世界の 3-BLOCKS パタン方法で時間節約率

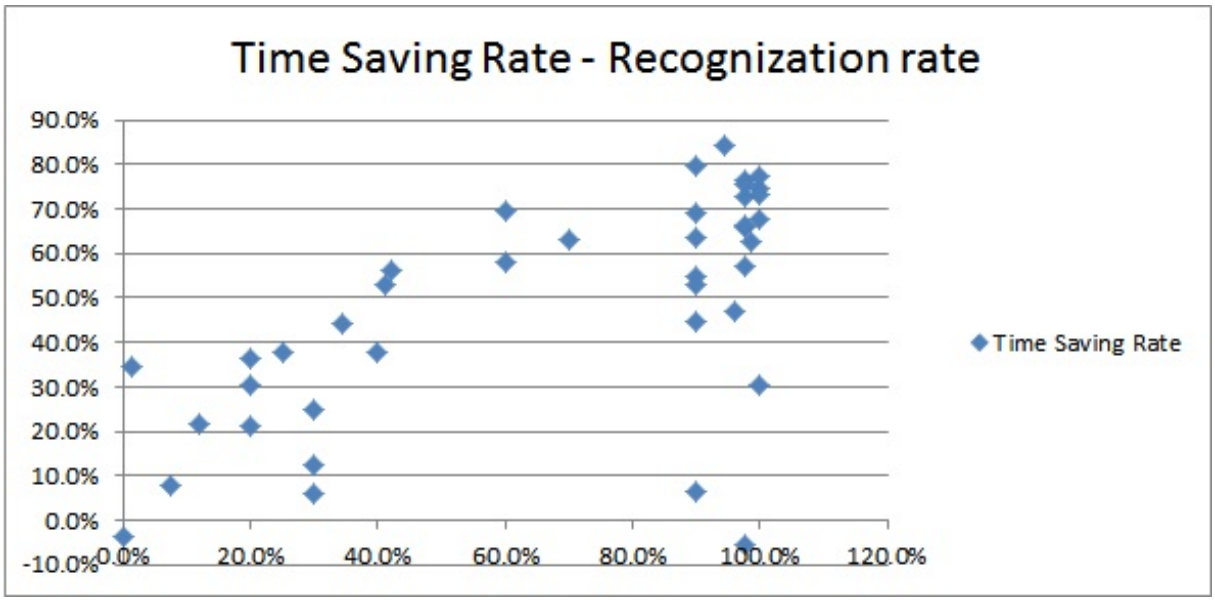


図 4.12: 認識率と時間節約率の分布

4.4 結果の評価

10 個ブロックの場合（実験の結果は図 4.4 に示す）に、実験 5 のブロック環境（図 4.13 と図 4.14）で「3-BLOCKS パタン」が 0 個識別されて、そのため、パタン認識システムの時間を含め、かえって実行の時間が古典的な方法より多かった。しかし、ブロック数が少ない場合にこの発生する確率が小さいのである。多くの 10 個ブロックのパタンで、1 から 3 までの 3-BLOCKS パタンを識別できた。

図 4.13: 10BLOCKS の実験 5 の初期状態

図 4.14: 10BLOCKS の実験 5 の目標状態

	10BLOCKS	20BLOCKS	30BLOCKS	40BLOCKS
平均の時間節約率	13.2%	53.0%	47.0%	58.2%
	50BLOCKS	60BLOCKS	70BLOCKS	80BLOCKS
平均の時間節約率	55.8%	40.0%	58.9%	47.5%

表 4.3: 10 個〜80 個ブロック世界の 3-BLOCKS パタン方法で平均の時間節約率

	10BLOCKS	20BLOCKS	30BLOCKS	40BLOCKS
で平均のパタン認識率	48.0%	78.0%	50.0%	72.0%
	50BLOCKS	60BLOCKS	70BLOCKS	80BLOCKS
で平均のパタン認識率	68.0%	69.0%	57.7%	86.3%

表 4.4: 10 個〜80 個ブロック世界の 3-BLOCKS パタン方法で平均のパタン認識率

図 4.12 で示したように、認識率が高まることに従って、時間の節約率が高まる。認識率が 60%に近づくときに、プランニング時間を 50%〜90%節約ことができた。大規模的なブロック世界の問題では、表 4.3 のように、ブロックの数が 20 個以上に超えると、平均の時間節約率は 50%あたりを上下している。
ブロック数が多ければ多いほど、少しだけのパタン認識によって認識されたブロックを移動した後、残ったブロックのプランニング時間もを大幅に削減した。たとえば、70 個

ブロック環境の実験 5 の結果（図 4.10 に示す）そのブロックの環境は（図 4.15 と図 4.16 に示す。

図 4.15: 70BLOCKS の実験 5 の初期状態

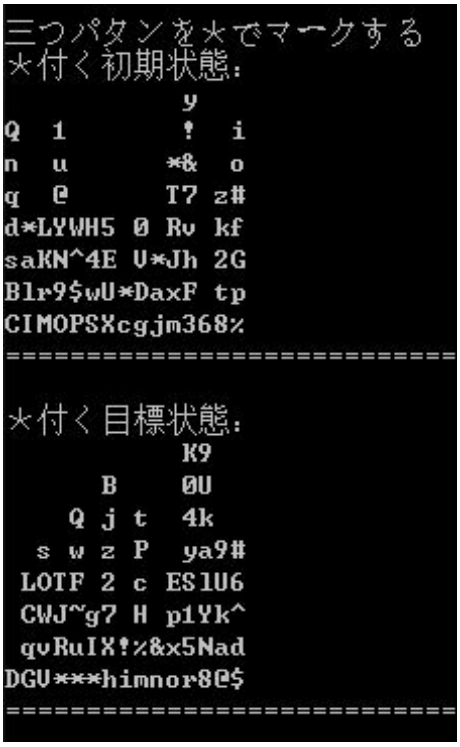


図 4.16: 70BLOCKS の実験 5 の目標状態

図 4.15 のようなブロック環境でブロック名 Z、b、e を第一回目パタン認識で識別されて、第二回目でブロック名 T、J、R が識別されて、三回目で A、i、o が識別され、それ以上息別されたパタンがなくなった。それで、あわせて、3 つのパタンを識別され、識別されたブロックの数は 9 で、70 個ブロックのわずか 12．8%だが、プランニング時間の節約率は 34.4%を得た。

第5章　結論

5.1　現状のまとめ

本研究のシステムでは、目標達成のために三つのブロック状態空間を利用する。状況や目標の変更があれば、現在の状況を考慮した上で達成すべき新目標を定める。具体的には、目標を達成するために現在の状態空間と目標状態空間の間に類似した部分がある場合には、現在の状態と比較し新たなプランを再構成し、バーチャル状態で再び新しい初期状態から探索を始める。その結果、システムもある程度に環境に対応できるものとなる。
今までの進捗は、「3 - BLOCKS パタン認識によるブロック世界の問題を分割するシステム」の研究は、パタン認識方法で、複数回の実験で、特に大規模のブロック世界の問題をバーチャル状態と子状態に分割することで、3-BLOCKS パタンの認識率が 100％に接近する実験の回数の半分ぐらいを占めていて、ブロック世界のプランニングのスピードが高速化されたことがわかった。
本研究は、プランニングスピードを高速化させることで、ブロック世界の環境の変化を対応させていくので、その中に、たまたまにブロック世界の重ね方により、ブロックの認識率が低下する場合もある。その状況で、高速的に環境変化に対応することができず、プランニング効率が悪くなる。しかし、大規模のブロック世界問題では、プランニング時間がブロックの数によって指数的に増加することから見ると、わずか少しパタンを認識されでも、プランニング時間を 10％〜20％減らし、プランニング時間節約に貢献できる。

5.2　今後の課題

今後の課題としては、より柔軟的に環境変化に対応するため、変化後の環境と変化する前の環境の違い部分を抽出して分析する。その違い部分に「3-BLOCKS パタン認識」で変化されていない部分から有機的に分割することで、環境変化に対応するスピードをさらに高速化することを期する。
プランニングスピードを高速化させることで、ブロック世界の環境の変化を対応させていくので、その中に、たまたまにブロック世界の重ね方により、ブロックの認識率が低下する場合もある。その状況で、高速的に環境変化に対応することができず、プランニング効率が悪くなる。その状況を解決するために、パタン予測機能が有すれば、より効率的にパタン発見することができると予想する。具体的には、ある 3-BLOCKS パタンの組合せが発見され、矛盾表で矛盾がないパタンの組合せを数個見出せば、どちを実行す

ることを決定するシステムを導入する。そのシステムで、ある 3-BLOCKS パタンを実行したあと、もっと多い 3-BLOCKS パタンを現せれば、そちの 3-BLOCKS パタンを実行することに決定する。

謝辞

本研究を進めるにあたり，指導していただいた東条敏教授に深く感謝いたします。佐野勝彦助教には、研究室の先輩の皆様，日頃から様々なことで気にかけて下さり，とても励みになりました。不慣れなことで私に的確な助言と助けをして下さった先輩と同級生に謝意を表し、心からお礼を申し上げます。

参考文献

[1] R. Fikes and N. Nilsson STRIPS: a new approach to the application of theorem proving to problem solving Artificial Intelligence 2:189-208.

[2] Helmert, M., On the complexity of planning in transportation domains In Cesta, A. Barrajo, D. (Eds.) Sixth European Conference on Planning (ECP-01) Toledo, Spain. Springer-Verlag, 2001

[3] Kautz, H., Unifying SAT-based and Graph-based Planning, Proceedings of IJCAI-99, pp.318-325, 1999

[4] 鍋島 英知、宋 剛秀、井上 克巳、岩沼 宏治, 「効率的な SAT プランニングと SAT スケジューリングのための補題再利用」電子情報通信学会. AI, 人工知能と知識処理, 106(38), pp.19-24, 2006

[5] 国立国語研究所, 分類語彙表, 秀英出版, 1964.

[6] Hayashi, H., Tokura, S., Hasegawa, T., and Ozaki, F., An Incremental Forward-Chaining HTN, Planning Agent in Dynamic Domains, Springer, 2006, pp.171-187

[7] 佐藤 敦史、石川 悟、大森 隆司、山内 康一郎, 「動的環境下における人の適応的プランニングの計算モデル化」, 電子情報通信学会. NC, ニューロコンピューティング 107(542), pp.295-300, 2008

[8] Richard E.Fikes, Peter E. Hart and Nils J.Nilsson 「Learning and Executing Generalized Robot Plans I」, Stanford Research Institute. NC, Menlo Park, California 94025, Artificial Intelligence 3 (1972), 251-288

[9] 市瀬龍太郎　ダニエル　シャピロ　　パット　ラングリー「行動履歴からの構造的プログラムの学習法」, 電子情報通信学会論文誌　D-I Vol. J87-D-I No.6 2004 年 6 月

[10] 松原　繁夫　　石田　亨「実時間探索に副目標生成機構を組み込んだ実時間プランニング」, 人口知能学会誌　Vol. 12 No. 1 1995

[11] D. Shapiro and P. Langley 「Separating skills from preference: Using learning to program by reward」, Proc. Nineteenth International Conference on Machine Learning, Sydney, 2002

[12] 三浦純 「ロボットにおけるプランニング」, 人工知能学会誌 16 巻 5 号（2001 年 9 月）