F-002

進化計算により知識を自己獲得・利用するエージェントアーキテクチャ

Agent Architecture that Self-accuriers and Applies Knowledge with Evolutional Computation

森田 信吾 †　**加藤 昇平** †
Shingo Morita　Shohei Kato

1　はじめに

近年，認知アーキテクチャの研究が盛んに行われている [1, 2]．認知アーキテクチャは，認知機能をモデル化するフレームワークとして用いられている．認知アーキテクチャは，特定の応用に限定されず，問題解決，プランニング，学習などのさまざまな機能を提供している．代表的な認知アーキテクチャとして Soar [3, 4, 5, 6] や ACT-R [7] などが挙げられる．認知アーキテクチャはサブゴールを再帰的に作成・達成し，問題を解決する．また認知アーキテクチャはサブゴールを達成する毎に新たな知識を獲得する．Taatgen の研究 [8] や Ritter らの研究 [9] において，認知アーキテクチャが問題解決過程で新たな知識を獲得し，解決までの時間を漸進的に短縮することが示されている．

人工知能のテーマの 1 つに自動計画がある．自動計画はプランニングに用いられる．プランナーは問題の初期状態，目標状態および選択可能なアクション集合を入力とする．また，初期状態から目標状態を達成する一連のアクションの系列を出力とする．自動計画において，アクションは事前条件と事後条件から成る．事前条件が満たされたアクションの実行により，事後条件の状態変化が起こる．

本研究では，認知アーキテクチャのフレームワークに基づくエージェントアーキテクチャを提案する．提案アーキテクチャは知識を利用した自動計画によりサブゴールを生成する．また，問題解決には進化計算手法の一つである Genetic Network Programming（以下，GNP） [10, 11] を用いる．なお，本研究においてエージェントとは，与えられた問題環境において GNP により実際に問題を解く主体を指す．また，提案アーキテクチャは問題解決の過程において，エージェントが蓄積したデータをマイニングすることで，新たな知識を自己獲得する．本稿では，迷路問題を用いた実験により，提案アーキテクチャの有効性と環境変化に対する頑健性を検証する．

2　提案アーキテクチャ

図 1 に本稿で提案するエージェントアーキテクチャの概要を示す．提案アーキテクチャはプランナー，SolverTree，知識抽出器，知識プールの 4 つの要素から構成される．なお本研究において，知識は確定ホーン節の集合とみなす．以下に知識の例を示す．

$$\text{have(key1)} \leftarrow \text{at(key1)} \wedge \text{find(key1, movable)}.$$

この知識では，知識の後件部および前件部に記された述語は該当する問題の目標（ゴール）状態と副目標（サブゴール）状態をそれぞれ表現する．また，前件部の知識は左から順に評価される．本研究で用いるエージェントはメモリ機能を持ち，環境からの観測情報を状態としてメモリに時系列順に格納する．このとき，格納された状態をラベルとして持ち，それまでに保持されていた自身のメモリ内容とともにデータとして蓄積する．なお初期状態については，ラベルのみからなるデータとして蓄積する．エージェントは自身の GNP グラフに従いノード遷移を繰り返し，ゴール状態がエージェントの持つメモリに格納されたとき，タスクを達成したとみなしてノード遷移を終了する．具体的な提案アーキテクチャの動作は，プランナーによる問題分割，分割された部分問題をノードとした木構造をもつ SolverTree の作成，各 Solver におけるエージェント群による GNP を用いた部分問題の解決，問題解決過程で観測したデータからの新たな知識の獲得という流れが基本となる．

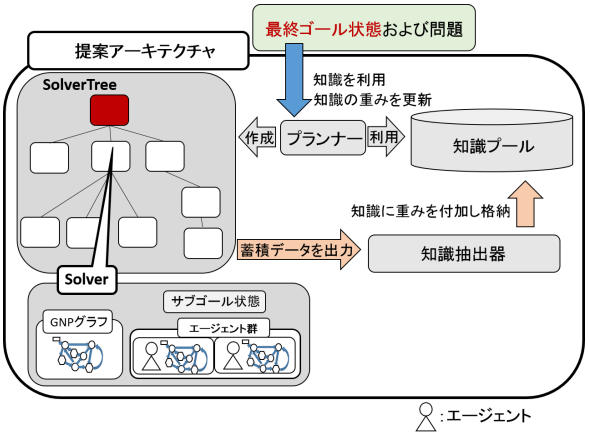


図 1　アーキテクチャ概要

2.1　プランナー

プランナーは与えられた問題およびゴール状態が現在の知識プールに存在する知識を用いて分割可能であるかを判断する．このとき，与えられたゴール状態を後件部に持つ知識を知識プールから検索する．さらにその前件部を後件部として持つ知識を深さ優先探索で繰り返し検索する．この過程で前件部が真である知識（単位節）を検索できれば問題は分割可能である．あるいは単位節を発見できなくとも，葉 Solver が問題の初期状態をサブゴール状態として持っている場合は問題分割が可能である．また，同じ後件部を持つ知識が複数ある場合，知識の重みに基づき知識を選択する．問題分割に成功した場合，プランナーは後述する SolverTree をトップダウンで作成する．図 2 に問題分割途中の例を示す．

2.2　SolverTree

SolverTree は Solver の持つ GNP グラフを一方向連結リストとして保持し，ボトムアップで各 Solver が GNP グラフを連結し問題解決を図る．図 3 に SolverTree の例を示す．この例では，根 Solver の連結リストには 15 個の GNP グラフが格納される．プランナーは問題分割が成功した場合に SolverTree を作成する．

2.2.1　Solver

Solver はある一つのゴール（サブゴール）状態とそのゴール（サブゴール）状態を達成するための GNP グラフ，さらに GNP におけるエージェント群および子 Solver 群を持つ．Solver は後述する GNP の機能を備えている．エージェント群および GNP グラフは各 Solver 毎に独立したものであるが，判定ライブラリと処理ライブラリについては共通のものを扱う．Solver は子 Solver 群が作成した GNP グラフのリストを連結する．その後，GNP を用いて自身に与えられたゴール（サブ

† 名古屋工業大学, Nagoya Institute of Technology

ゴール）状態を達成する．ゴール達成エージェントが獲得した中で最も適応度の高いグラフを 1 つ選択し，Solver の GNP グラフを更新する．

2.2.2 SolverTree 作成アルゴリズム

(i) プランナーにより根 Solver にゴール状態が割り当てられる．

(ii) 根 Solver はゴール状態を後件部に持つ知識の前件部をプランナーを介して取得し，前件部の状態それぞれをサブゴール状態とした子 Solver 群を作成する．

(iii) 子 Solver は (ii) と同様にプランナーを介して自身のサブゴール状態を後件部として持つ知識の前件部を取得する．取得した前件部に応じて以下の処理を行う．

(a) 前件部がゴール状態ではない述語を持つ場合，それらの述語をサブゴール状態とした子 Solver 群を作成し，それぞれについて (iii) を繰り返す．

(b) 前件部が真または述語を持たないの場合，Solver は子 Solver 群を持たない葉 Solver となる．

すなわち，任意の内部 Solver はひとつの知識（ホーン節）の後件部を達成し，その子 Solver 群はその知識の前件部を達成することとなる．SolverTree が完成した後，深さ優先探索により，各 Solver の GNP を用いてゴール（サブゴール）状態を達成していく．SolverTree による問題解決には以下の 2 種類のフェイズが存在する．

- ゴール達成フェイズ
- 最適化フェイズ

なおプランナーが問題分割に失敗した場合，SolverTree は子 Solver を持たない単一の Solver として作成される．入力された問題のゴール状態を達成する単一の GNP グラフを作成し，問題解決する．

2.2.3 ゴール達成フェイズ

ゴール達成フェイズでは，エージェントが知識利用により作成された各 Solver のゴール（サブゴール）状態を実際の問題環境で GNP を用いて達成する．選択された Solver は以下に示すアルゴリズムを繰り返す．

(i) サブゴール状態が達成されていない子 Solver を持つ場合，子 Solver を 1 つ選択する．

(ii) サブゴール状態が達成されていない子 Solver を持たない場合，自身のゴール（サブゴール）状態を GNP により達成し，子 Solver が作成したグラフリストを連結する．その後グラフリストに自身の GNP グラフを連結させ，親 Solver を選択する．自身が根 Solver の場合，グラフリストを暫定的な解としてアルゴリズムを終了する．

2.2.4 最適化フェイズ

最適化フェイズではゴール達成フェイズに作成された暫定的な解の最適化および環境や問題の変化への対応を行う．得られた GNP グラフリストのゴール（サブゴール）状態がそれぞれ達成できているかの確認を行った後，以下に示すアルゴリズムを実行する．アルゴリズムは問題規模に応じて設定された規定世代数に達すると停止する．

(i) ゴール（サブゴール）状態の達成に失敗した場合，SolverTree は環境や問題が変化したとみなしてゴール達成に失敗した Solver の GNP グラフを新たに作り直す．

(ii) SolverTree のグラフリストがゴール（サブゴール）状態の達成に成功した場合，グラフリストの中から最も達成にステップ数を要する GNP グラフを選択し改良する．GNP グラフの改良の際には，該当する Solver の GNP グラフを初期値として持つエージェント群を新たに生成する．

2.2.5 GNP

GNP [11] は進化計算手法の 1 つであり，初期ノード，判定ノードおよび処理ノードから構成される有向グラフネットワークを利用する．GNP は判定ライブラリと処理ライブラリという 2 つのライブラリを持つ．これらのライブラリには判定または処理の内容が複数記述されている．また判定ノードおよび処理ノードの遷移には設計者により遅れ時間が設定される．エージェントがノード遷移する度に遅れ時間が加算され，その値が一定を超えたとき 1 ステップとする．図 4 に GNP グラフ例およびノードの例（判定ノード）を示す．GNP は他の進化計算手法と同様にエージェント群に対して遺伝的操作を適用し，GNP グラフを進化させる．以下に GNP のアルゴリズムを示す．

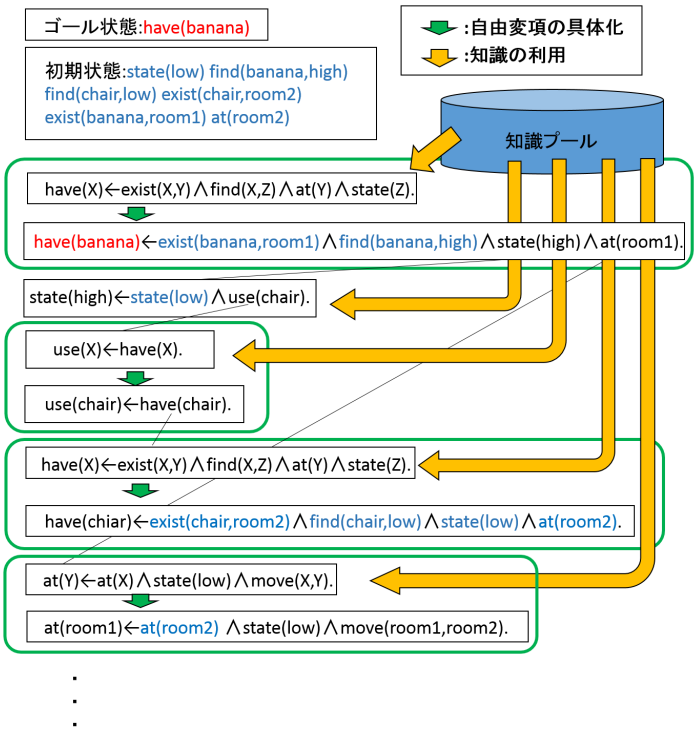


図 2　知識利用による問題分割の例

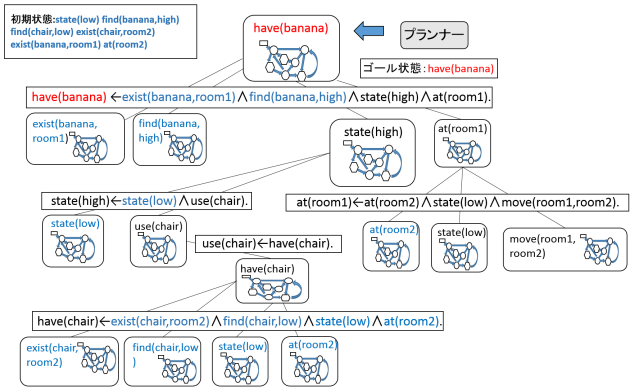


図 3　SolverTree の例

2.2.6 GNP のアルゴリズム

(i) 処理ノード，判定ノードをランダムに組み合わせて GNP グラフを初期化する．
(ii) エージェントは自身の GNP グラフを用いて規定ステップ数に達するまでノード遷移およびノードの実行を行い適応度を決定する．
(iii) （ii）で求めた適応度に応じて，選択アルゴリズムにより親個体を決定し，遺伝的操作を行う．
(iv) 規定世代数に達するまで，(ii) を繰り返す．

判定ノードは if-then 型の条件判定を行う．判定ノードはライブラリのインデックスを持つ．エージェントは該当する判定ライブラリの内容を実行し，自身あるいは環境の状態を判定する．その後，判定結果に応じて次に遷移するノードを決定し遷移する．また，処理ノードは自身の処理ライブラリのインデックスに応じてエージェントの行動を決定し，遷移する．GNP はノード遷移とそれぞれのノードでの処理を繰り返し行うことによってタスクの達成を目指す．また，GNP のノード遷移は初期ノードから開始するが終端ノードは存在せず，初期ノードへのノード遷移は存在しない．ノード遷移はあらかじめ定められたステップ数に到達するかタスクを達成した際に終了する．

2.2.7 適応度

GNP における適応度はエージェントの持つ GNP グラフにより求められる．設計者は GNP を適用したい問題と達成させたいタスクに応じた適応度関数，遺伝的操作に用いる各種確率，さらにその問題の中でエージェントが取ることができる行動内容を判定ライブラリあるいは処理ライブラリに記述する必要がある．

2.2.8 選択

GNP において遺伝的操作を行うためにエージェント群から親個体を選択する必要がある．本アーキテクチャでは間普ら [12] と同様に親個体の選択方法はトーナメント選択を用いる．

2.2.9 交叉

GNP の交叉は選択されたエージェント 2 体の間で行われる．エージェントは GNP グラフのすべてのノードに対してあらかじめ定められた交叉確率で交叉するか否かの判定を行う．交叉される場合，それぞれの GNP グラフの同じインデックスの位置にあるノードを入れ替える．

2.2.10 突然変異

GNP の突然変異はエージェント群の中から選ばれたエージェント 1 体に対して行われ，ライブラリのインデックスとノード遷移先について 2 通りの突然変異が存在する．突然変異は，交叉と同様にエージェントの持つ GNP グラフのすべてのノードに対して行われ，あらかじめ定められた突然変異率に従い 2 種類の突然変異それぞれが発生するか否かの判定が行われる．突然変異が発生する場合，それがライブラリのインデックスの突然変異であればノードの種類に応じたライブラリのインデックスをランダムに 1 つ選択してライブラリのインデックスの書き換えを行う．遷移先の突然変異が発生した場合には，自身の GNP グラフから 1 つのノードを選択して新たな遷移先ノードとする．なおノード遷移先が複数存在するノードは各遷移先について突然変異が発生するかどうかの判定を行う．

2.3 知識抽出器

知識抽出器はエージェント群が問題を解く過程で得た蓄積データを蓄積データ集合として保持する．蓄積データはラベルにより部分集合に分けられる．知識抽出器は蓄積データ集合をマイニングし，新たな知識を獲得する．マイニングされた知識はホーン節で表現され，ホーン節に含まれる述語は各エージェントが自身の GNP グラフに従い行動することで環境から観測した状態を表す．プランナーが問題分割に失敗した場合，

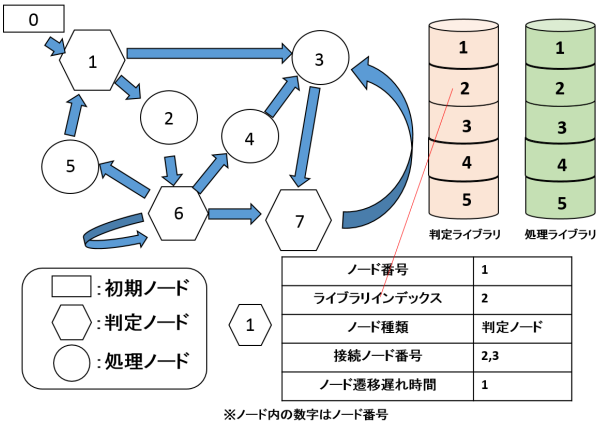


図 4 GNP グラフおよびノードの例（判定ノード）

その問題は既存の知識では解決できない問題であり，提案アーキテクチャはそのような状況においては単一の Solver による GNP での問題解決を図る．この過程で蓄積したデータを基に知識抽出器が新たな知識を獲得することで次回以降の問題解決に利用する．

2.3.1 マイニングアルゴリズム

ゴール状態をラベルとして持つ蓄積データには，必ず問題を解く上で必要であった状態の系列が含まれている．加えて，エージェントはメモリに状態が追加されると時系列順に状態を保持していくので，蓄積データに含まれる状態をそれぞれ順に達成していくことで必ずゴール状態を達成する．したがって蓄積データに含まれる状態はその観測された順序が重要である．よって，蓄積データは述語の出現順序をもつ順序付きの述語集合となる．知識抽出器は，ゴール状態をラベルとして持つ知識の前件部全てを知識化するまで以下のアルゴリズムに従い知識を抽出する．

(i) 蓄積データ集合全体からゴール状態をラベルとして持つ蓄積データの集合 D を抽出する（D のサイズを L とする）．この際，それぞれの要素には集合に追加された順にインデックスが割り当てられる．
(ii) 抽出した D の中から最も述語数の少ない，最小蓄積データ d_{min} を選択する．このとき d_{min} となる候補が複数存在する場合，最もインデックスの小さいものを d_{min} とする．
(iii) 任意の蓄積データ $d_l \in D\backslash\{d_{min}\}$ について，d_{min} の内のそれぞれ述語との間に $d_{min} \not\subset d_l$ が成立するならば，その蓄積データ d_l を獲得知識集合 S に追加する．さらに S に d_{min} 自身を加える．
(iv) 獲得知識集合 S 内の要素 s_k それぞれについて，知識を作成する．なお，この知識は s_k のラベルを後件部として持ち，s_k のラベルを除く述語全てを前件部として持つ．また，作成された知識は知識プールに格納される．その後，作成された知識を集合 K（K のサイズを G とする）に加えた後以下の操作を行う．
  (a) 作成された知識 $k_g \in K$ の前件部が述語を持つ場合，これらをラベルとして持つ D を蓄積データ集合全体の中から抽出し，（ii）以降を繰り返す．
  (b) k_g の前件部が述語を持たない場合，バックトラックを行う．

このようにして知識抽出器は知識獲得を行う．図 5 に知識獲得の例を示す．図 5 の場合，最小蓄積データ $d_{min} = d_0$ となる．このとき，$d_{min} \not\subset d_2$ のみが成り立つので，d_0 と d_2 の知識を獲得する．最小述語数の蓄積データのみを知識化するのではなく，（iii）のような関係を満たす蓄積データ d_2 を知識化する

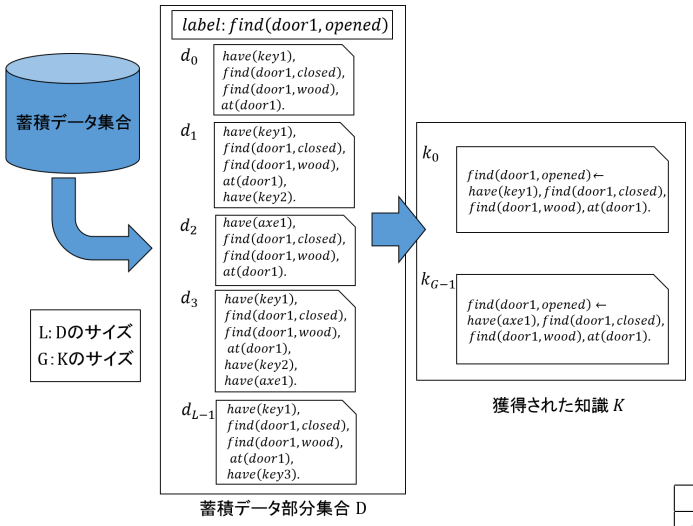
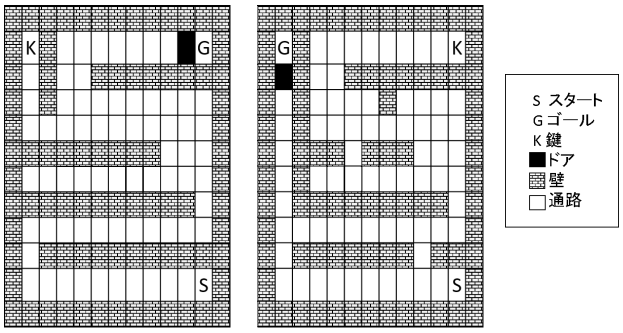


図 5　知識獲得例



[a] 迷路 1　　[b] 迷路 2

図 6　迷路環境

ことで，知識の多様性が得られると考える．この後 k_0,k_1 の順に，前件部一つ一つをラベルとして持つ蓄積データ部分集合が作成されていく．アルゴリズムが停止した後，獲得した知識に知識の重みが付与される．

2.4　知識プール

知識プールには知識抽出器が獲得した知識が格納される．知識プールに格納される知識は問題毎に消去されず，他の問題を解決する際にも利用可能である．知識プールに格納する知識には，前件部，後件部以外に知識の重みが記述される．この知識の重みはプランナーが知識利用により SolverTree を作成する際，同一の後件部を持つ知識が複数候補になった場合の知識選択に用いられる．また，知識の重みはプランニングの成否に応じて更新される．

3　実験

3.1　実験概要

実験では迷路問題を用いた 3 通りの実験を行った．図 6[a] に本実験で用いた迷路環境を示す．

本実験では，エージェントは S のスタートから探索を行い，G のゴールにたどり着くことを目標としている．G のゴールの前にはドアが存在し，K の鍵を取得した後，ドアを開かなければエージェントはゴールに到達することができない．1 ステップ制限時間が 5 なのでエージェントは 1 ステップの間に「5 回の判定ノードの実行」または，「4 回以下の判定ノードの実行と 1 回の処理ノードの実行」が可能である．エージェントは，最大ステップ数を経過するかゴールした時に適応度を更新する．全てのエージェントが適応度の更新を終えた後，エージェント群に対し遺伝的操作を適用し，世代交代を行う．表 1 に本実験で提案アーキテクチャが用いる GNP のパラメータを示す．

判定ノードは 1 種類であり，「進行方向に隣接するグリッドの状態を判定」し図 6[b] 迷路 2 の右に示す 6 状態を判定する．処理ノードは「前進」，「右 90 ° 回転」，「左 90 ° 回転」，「(現在のグリッドに存在する）アイテムの取得」の 4 種類である．

エージェントは K の位置で処理ノード「(現在のグリッドに存在する）アイテムの取得」を実行することで鍵を取得することができ，エージェントは鍵を取得している状態でドアに向かって進むことでドアを開くことが可能である．本研究においては，オブジェクトの性質と状態をエージェントはオブジェクトそのものからアフォーダンスとして受け取ることが出来るとする．例えば，エージェントが鍵を所持していない状態でドアのグリッドに到達した際，そのドアが閉まっていることやそれがドアであるという情報は，ドアというオブジェクト自身から与えられるものとしている．本実験ではエージェントは表 2 の状態を感知できるものとする．状態において，オブジェクト O とはそのオブジェクト固有の名前を差し，カテゴリ C はオブジェクトの種別を指す．例えば，door1 はオブジェクト名であるのに対し，そのカテゴリは door である．

表 1　パラメータ設定

世代数	3000
エージェント個体数	300（エリート:1，交叉:120，突然変異:179）
ノード数	60（判定ノード:40，処理ノード:20）
最大ステップ数	300
トーナメントサイズ	6
交叉確率	0.1
突然変異率	0.01
適応度	最大ステップ数-ステップ数:ゴールした個体 -1 :ゴールできなかった個体
目標状態	at(goal)
最適化フェイズ 規定世代数	300
ノード遷移遅れ時間	判定ノード:1，処理ノード:5
1 ステップ制限時間	5

表 2　エージェントの取りうる状態

状態	内容
at(O)	エージェントはオブジェクト O が存在するグリッドにいる
find(O,C,S)	エージェントはオブジェクト O がカテゴリ C に属し，状態が S であることを発見している
exist(O,C,X:Y)	エージェントはカテゴリ C に属するオブジェクト O が座標（X，Y）に存在することを確認している
have（O,C）	エージェントはカテゴリ C のオブジェクト O を所持している

3.2　実験 1：知識獲得実験

実験 1 では，提案アーキテクチャにおいてプランナーが知識プールを利用した SolverTree の作成に失敗した場合の知識獲得実験を行う．実験環境は図 6[a] に示した迷路を用いる．実験に用いるパラメータは表 1 に示したパラメータを用いる．またエージェントに与えられる初期状態はなく，メモリは空であるとする．以下に実験 1 で獲得された知識を示す．

at(door1) ←.

find(door1,door,closed) ← at(door1).

at(key1) ← .

find(key1,key,movable) ← at(key1).

exist(door1,door,9:1) ← at(door1)
∧ find(door1,door,closed).

exist(key1,key,1:1) ← at(key1)
∧ find(key1,key,movable).

have(key1,key) ← at(key1) ∧ find(key1,key,movable)
∧ exist(key1,key,1:1).

at(goal) ← find(key1,key,movable)
∧ exist(key1,key,1:1) ∧ have(key1,key)
∧ exist(door1,door,9:1)
∧ find(door1,door,closed).

find(door1,door,opened) ← find(key1,key,movable)
∧ exist(key1,key,1:1) ∧ have(key1,key)
∧ at(door1) ∧ find(door1,door,closed)
∧ exist(door1,door,9:1).

find(door1,door,opened) ← find(key1,key,movable)
∧ exist(key1,key,1:1) ∧ have(key1,key)
∧ find(door1,door,closed) ∧ exist(door1,door,9:1)
∧ at(door1).

迷路問題のゴールに到達するためには find(door1,door,opened) という状態が必要であるという知識が獲得できており，また，その状態を達成するためには have(key1,key) が必要であるという知識も獲得できていることが確認できる．

また，状態 find(door1,door,opened) を後件部に持つ知識は二つ獲得されている．これは，at(door1) の出現順が異なるので，別の知識として獲得されているためである．

3.3 実験 2：知識利用の可否による性能比較

実験 2 では提案手法と GNP のみによる手法との比較によりプランナーが知識を利用可能な場合とそうでない場合での性能比較実験を行う．パラメータや迷路については実験 1 と同様で行う．また，提案手法は実験 1 で獲得した知識を用いて問題を分割可能であるが，分割したサブゴールをそれぞれ同時に進化させると GNP のみの手法との間で GNP グラフを進化可能な回数に差異が生じる．したがって，提案手法はグラフを進化させる際に，その時点で最もサブゴール達成にステップ数が必要なグラフを一つ選び，進化を行う．進化対象となるグラフはパラメータに示した最適化フェイズ規定世代数に到達する度選び直される．図 7 に実験結果を示す．赤の実線で示されているのが提案手法，青破線で示されているのが GNP のみによる手法である．緑破線で示されるのは実験 2 における最適解を示す．

実験結果からわかるように，提案手法のほうが GNP のみの手法に比べより良い解を示していることがわかる．さらに，GNP のみの手法は初期の世代ではゴールが達成できず，適応度が-1 を示しているのに対し，提案手法では初期の世代からゴールを達成している．また，収束速度においても提案手法は GNP のみの手法を上回っていることがわかる（750 世代付近）．これは GNP のみの手法は一連の状態を全て達成するノード遷移を一つの GNP グラフのみで完成させなければならないのに対し，提案手法はプランナーによりサブゴール状態の分割がなされることで GNP グラフの作成が容易であったことが原因として考えられる．また，GNP のみの手法と比べ，提案手法においてはグラフが階段状となっていることがわかる．これは，上でも述べたように提案手法は問題分割をしたあとそれらのサブゴールを達成する GNP グラフを並列的に作成せずあえて一つのみを選択して進化させているためである．したがってこの階段状の変化は最適化フェイズ規定世代数である 300 世代毎に発生していることがわかる．今回は進化可能な回数を 2 手法で統一するためにこのような設定で実験を行ったが，提案手法においてサブゴールを達成する GNP グラフを同時に進化させることによりこの階段状の変化は指数関数的な変化になると考える．

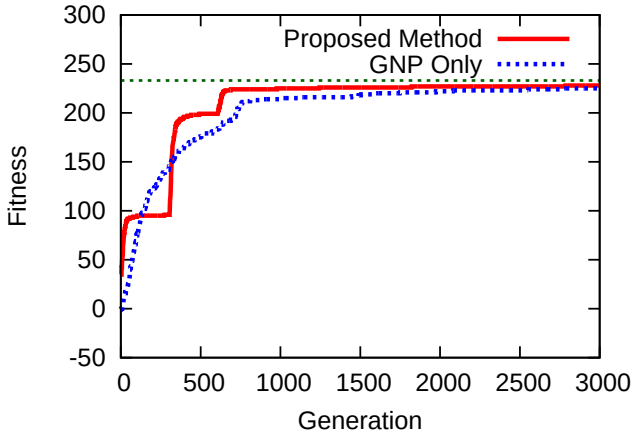


図 7　知識利用の可否による性能比較実験

表 3　実時間比較

	提案手法		GNP のみの手法
	合計時間	SolverTree 作成時間	
最小時間（ms）	40,318	310	55,124
最大時間（ms）	53,951	1,057	74,682
平均時間（ms）	46,201	569	60,288

3.3.1 提案アーキテクチャの時間的性能

提案アーキテクチャでは，プランナーが知識プールに存在する知識を利用して SolverTree を作成する．問題分割を行う性質上，解の質や収束速度が GNP のみの手法以上になるのはもとより，問題解決にかかる時間も GNP のみの手法と同等以上となることが望ましい．ここではプランナーが知識利用により問題を分割し，SolverTree を作成するのにどの程度のオーバーヘッドがかかるかということと，提案手法による問題解決の時間的な能力を検証するための実時間比較を行う．パラメータや迷路は実験 1 と同様である．また，実験は独立 50 試行で行う．表 3 に計測結果を示す．表 3 からわかるように，SolverTree を作成するオーバーヘッドは提案手法の平均終了時間の 1% ほどである．また，SolverTree の作成にかかるオーバーヘッドを考慮しても，提案手法のほうが GNP のみの手法と比較して 3000 世代を終了するまでの時間が短くなっている．これは，GNP のみの手法は一つの GNP グラフのみで問題を解決するので，GNP による進化の途中でのエージェント一体の平均ステップ数が問題が分割できている提案手法よりも多くなることが原因であると考える．

3.4 実験 3：環境変化による影響

実験 3 では，世代の途中で問題環境が変化した場合（図 6[b]）について提案手法と GNP のみによる手法の性能比較実験を行

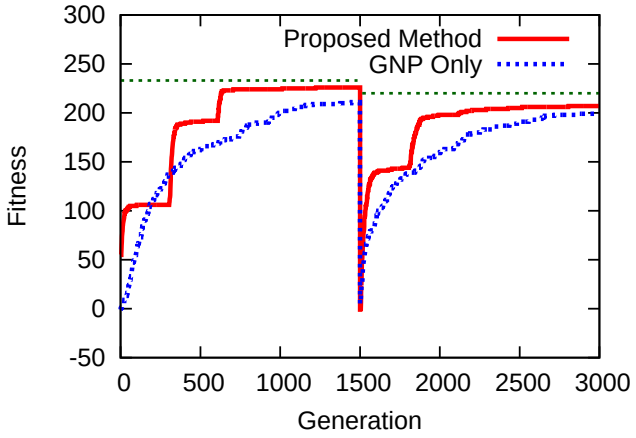


図 8　環境変化による性能実験

う．世代数が 1500 世代になると迷路が変化し，両手法ともに GNP グラフを再構築する必要がある．実験結果を図 8 に示す．実験 2 と同様に赤の実線は提案手法，青破線は GNP のみの手法を示す．緑破線は最適解を示すが，迷路環境が変化した際に最適解も変化する．
図 8 の実験結果からもわかるように，実験 1 と同様に提案手法の方が良い解を獲得できている．また，環境変化前の世代においては，GNP のみを用いた手法は解が収束しきっていないにも関わらず，提案手法は 700 世代ごろから解の収束が見られる．環境変化後の世代においても同様に，提案手法は環境変化から 500 世代ほどで解の収束がみられる．提案手法の収束速度について，環境の変化後により顕著に現れている．これは提案手法は変化前後の環境で共通する GNP グラフを再利用できるため，これらの GNP グラフの作成コストが削減された結果であると考える．

4　おわりに

本稿では，知識利用が可能な問題では自動計画を行い GNP を用いて問題を解決し，知識利用が不可能な問題では問題解決した後，蓄積データをマイニングすることで新たな知識を獲得するエージェントアーキテクチャを提案した．また，GNP との比較実験を行い知識利用による収束速度の向上や解の質の向上を確認した．この結果から，知識獲得および知識利用により問題分割を行う提案アーキテクチャの有効性を確認した．
今後の課題として，説明に基づく学習 [13] などにも見られるような，獲得した知識の一般化が挙げられる．また，現在提案アーキテクチャのプランナーは問題に対し，知識の適用を静的にかつ適用が完全に行える場合しか知識利用ができない．したがって今後は部分的な知識獲得や動的な自動計画が行えるような改良を行うとともに Soar や ACT-R などの認知アーキテクチャとの比較も行っていきたい．

謝辞

本研究は，一部，文部科学省科学研究費補助金（課題番号 25280100，および，25540146）の助成により行われた．

参考文献

[1] 森田：“文献紹介認知アーキテクチャを利用したスキル獲得に関する最近の研究の紹介”, 認知科学, **15**, 4, pp. 699–704 (2008).
[2] F. E. Ritter, 森田：“認知モデリングにおける二つのフロンティア：感情とユーザビリティ (< 特集 > 認知科学におけるモデルベースアプローチ)”, 人工知能学会誌, **24**, 2, pp. 245–252 (2009).
[3] J. E. Laird, A. Newell and P. S. Rosenbloom: “Soar: An architecture for general intelligence”, Artif. Intell., **33**, 1, pp. 1–64 (1987).
[4] 池田：“Soar と知識処理 (< 小特集 >「soar プロジェクト」)”, 人工知能学会誌, **9**, 4, pp. 487–496 (1994).
[5] 奥乃：“Soar アーキテクチャ (< 小特集 >「soar プロジェクト」)”, 人工知能学会誌, **9**, 4, pp. 479–486 (1994).
[6] 開：“Soar における学習：認知科学的側面 (< 小特集 >「soar プロジェクト」)”, 人工知能学会誌, **9**, 4, pp. 497–504 (1994).
[7] J. R. Anderson, B. D., M. D. Byrne, S. Douglass and Y. Lebiere, C. & Qin: “An integrated theory of the mind.”, Psychological Review, **111**, pp. 1036–1060 (2004).
[8] N. A. Taatgen: “Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills.”, Cognitive Science, **29**, pp. 421–455 (2005).
[9] P. A. Ritter, E. F. & Bibby: “Modeling How, When,and What is Learned in a Simple Fault-Finding Task.”, Cognitive Science, **32**, pp. 862–892 (2008).
[10] S. Sendari, S. Mabu and K. Hirasawa: “Two-Stage Reinforcement Learning on Credit Branch Genetic Network Programming for Mobile Robots”, IEEJ Transactions on Electronics, Information and Systems, **133**, 4, pp. 856–863 (2013).
[11] 片桐, 平澤, 胡, 村田：“Genetic network programming とそのマルチエージェントシステムへの応用”, 電気学会論文誌, **122**, 12, pp. 2149–2156 (2002).
[12] 間普, 平澤, 古月, H. JINGLU：“強化学習を用いた遺伝的ネットワークプログラミングとそのエージェントの行動生成における性能評価”, 情報処理学会論文誌, **46**, 12, pp. 3207–3217 (2005).
[13] 川上：“説明に基づく学習”, 日本ファジィ学会誌, **11**, 1, pp. 64–69 (1999).