修士論文

題目

動的問題における強化学習に関する研究

指導教授

所 真理雄 客員教授　・安西 祐一郎 教授

慶應義塾大学大学院

理工学研究科 計算機科学専攻

1997年度

89624288

桑折 隆之

論文要旨

専攻	計算機科学	学籍番号	89624288	氏名	桑折 隆之

(論文題目)

動的問題における強化学習に関する研究

強化学習とは，報酬や罰などといった強化信号に基づき適切な行動系列を学習する手法である．著者らは，強化学習の一手法であるQ学習において，問題の適切な分割 構造化を自動的に行なうAMQL(Automatic Modular Q-Learning)アーキテクチャを提案し，動的問題に対する有効性を示した．また，松林はAMQLアーキテクチャのモジュールセットの獲得能力の弱さを指摘し，GAによる広域探索の手法を適用することで，より適切なモジュールセットを獲得するAMQL-GA(AMQL with Genetic Algorithm)アーキテクチャを提案した．しかしAMQL-GAアーキテクチャは，問題変化毎に既学習内容を初期化するため，問題変化が小さい場合には効率的でない．逆に，従来のQ学習では，緩やかな問題変化に対しては追従できるが，急激な変化に対しては，追従するために非常に時間がかかり，適切に学習できない場合がある．これは過学習などが原因として考えられる．

本論文では，動的問題において効果的に学習を行なう機構であるAMQL-LL(AMQL with Lamarckian Learning)アーキテクチャを提案する．これは，AMQL-GAアーキテクチャの枠組で，既学習内容をモジュール単位で保持する機構を導入したものであり，問題変化に対して適切なモジュールを選択することによって効率良く学習することができる．ここでは追跡ゲームを用いた実験を行ない，問題変化が小さい場合には，有効な既学習モジュールを適用することにより，AMQL-GAアーキテクチャに比べて学習収束速度が向上することを示す．また，問題が大きく変化し，過学習が影響する場合にも，新たに生成されたモジュールを適用することにより，適切な行動方針が学習可能であることを示す．

慶應義塾大学理工学研究科修士課程

◎ 内容の要旨は約600字

Summary of Master's Thesis

Course	Student No.	SURNAME, Firstname
Computer Science	89624288	Takayuki Kohri

Title

A Study on Reinforcement Learning for Dynamically Changing Problems

Reinforcement learning is a technique for an agent to learn suitable action policies that maximize his utility, by the means of reinforcement signals: *reward* or *punishment*. Earlier, we proposed *AMQL(Automatic Modular Q-Learning)* architecture for reinforcement learning, that enables agents to obtain a suitable set of modules by themselves, and we showed AMQL is effective for dynamically changing problem. Matsubayashi improved AMQL's drawback of inability of obtaining a set of modules, and offered *AMQL-GA(AMQL with Genetic Algorithm)* architecture that obtains a suitable set of modules by using Genetic Algorithms. Although AMQL-GA is able to treat dynamically changing problems, it does not seem efficient, since it has to start learning from the beginning at each problem change. Conventional Q-learning is able to adapt to slowly-varying non-stationary problems. However, Q-learning is not effective for radically-varying problems. The drawback is caused by over-learning.

In this paper, we propose a new architecture called *AMQL-LL(AMQL with Lamrckian Learning)*, that learns dynamically changing problem efficiently. This architecture is an improvement of AQML-GA by holding action policies that have been obtained as modules, and obtains suitable sets of modules in response to problem changes. As an example of a learning problem, we take the *Pursuit Problem*. Through a number experiments, we show that agents with AMQL-LL can adapt themselves to small problem changes efficiently, by using suitable modules which learned suitable action policies in advance. We also show that this architecture solves the over-learning problem by adapting new modules in radically-varying problems.

Graduate School of Science and Technology, KEIO UNIVERSITY

◎ The contents of English summary should be written in about 300 words.

目 次

図目次

表 目 次

第 1 章

序論

1.1 はじめに

人間を始めとする多くの生物は，学習を行なうことで未知なる環境においても適切に振舞うことができる．さらに環境が変化した場合，既存の学習内容を修正，もしくは新たに学習を行なうことで柔軟に適応することができる．

このように環境の変化に対して柔軟に適応する機構を，開放型環境において行動する自律的エージェントの構築に応用することは極めて有効であると考えられる．これは，ある環境に対して適切に振舞うように予めエージェントのすべての行動を記述することは，環境が動的に変化する場合には不可能であり，環境との相互作用から適切な行動方針を獲得する必要性が必然的に生ずるからである．

ここでエージェントとは，独自の判断基準で行動決定を行ない，個々の問題を解決する自律的な個体のことを指し，具体的には計算機環境におけるソフトウェアエージェントや，実世界におけるロボットエージェントなどが挙げられる．未知なる環境において自律的に問題解決を行なうエージェントを構築するためには，教師との情報交換を通じて学習を行なう機械学習とは異なる枠組を必要とし，その一つとして強化学習が注目されている．強化学習とは，本来動物心理学あるいは動物行動学の分野で用いられた用語であり，人工知能，機械学習における強化学習の研究は，古くはSamuelのChecker Player[31]にまで遡ることができる．強化学習を用いて学習を行なうエージェントは，行動あるいは一連の行動系列により環境から報酬や罰などの強化信号が与えられるという過程を繰り返すことにより，その行動パターンが徐々に強化され，報酬を得るために適切な行動を学習する．この強化学習は，外部から直接的な解答を教えることなく，強化信号という弱い情報源のみによってエージェントが適切な行動系列を学習することが可能であるという特徴を持つ．

機械学習の分野において強化学習を用いた研究としてQ学習[38]，$TD(\lambda)$法[35]，分類子システム[19]など様々な学習手法が提案されてきた．特に強化学習の代表的手法であるQ学習は，問題領域としてマルコフ決定過程を仮定した場合，学習終了後には期待報酬獲得の最大化が保証されることなどの理由により多くの研究で用いられ，改良手法や応用方法が提案されてきた．

Q学習を用いて学習する際，収束するまでに時間がかかるという問題点に対しては，問題をより単純な複数の副問題に分割し，モジュール化を行なう機構である構造化Q学習がWhiteheadによって提案された[40]．しかし，構造化Q学習では，エージェントが遭遇する問題に対して適切なモジュールセットを設計者が予め与えなければならない．このため，動的に変化する問題に対して柔軟に対応できない．このような問題点に対し

て，著者らは環境との相互作用を通じて構造化を自動的に獲得する AMQL(Automatic Modular Q-Learning) アーキテクチャを提案した [25]．さらに実験により，モジュールセットを自動的に獲得する機構の動的問題における有効性を示した．

また松林は，AMQL アーキテクチャにおけるモジュールセット獲得能力の弱さを指摘し，広域探索の手法である遺伝的アルゴリズムを組み合わせることで，より効果的なモジュールセットを獲得する AMQL-GA(AMQL with Genetic Algorithm) アーキテクチャを提案した [6]．AMQL-GA アーキテクチャは，広域に探索するという性質上，問題変化の際に既存の学習内容を破棄し，初期化された状態から学習し直すという枠組を提供している．このため，いわば広域探索の必要がない，問題変化が少なく既存の学習内容が部分的に適用できるような場合には非効率的であるなど，問題の変化に対する追従性については疑問が残る．

従来の基本的な Q 学習の場合，問題変化が小さい場合には，既存の行動方針を修正することにより追従することができる [22]．しかし問題の大きな変化に対しては，特定の問題に対する過学習等の原因により，追従するまでに非常に時間がかかり，適切な行動方針を学習できない場合がある．動的問題における強化学習機構を構築する際には，問題の変化が少ない場合は既存の学習内容を利用，修正し，問題が大きく変化する場合には最初から学習し直す戦略が有効であると考えられる．

1.2 目的

本論文では，AMQL-GA アーキテクチャの枠組において，問題の動的な変化の度合に応じて適切に対応する AMQL-LL(Automatic Modular Q-Learning with Lamarckian Learning) アーキテクチャを提案し，評価を行なう．評価には分散人工知能のテストベッドである追跡ゲームを用い，動的問題における学習性能の比較，考察を行ない，AQML-LL アーキテクチャの有効性を示す．

1.3 本論文の構成

本論文の構成は以下の通りである．まず第 2 章では背景として強化学習，構造化 Q 学習，AMQL アーキテクチャについて説明し，さらに動的に変化する問題に対して提案された従来の学習機構について述べる．第 3 章では本論文の提案する AMQL-LL アーキテクチャについて説明し，第 4 章で評価・検討を行なう．第 5 章では関連研究との比較や今後の課題について述べる．最後に第 6 章で結論を述べる．

第 2 章

背景

本章ではまず強化学習の背景について述べ，特に本研究で用いるQ学習について説明する．また，本論文で提案するAMQL-LLアーキテクチャの基となるAMQLアーキテクチャ，AMQL-GAアーキテクチャの機構について詳しく述べる．さらに強化学習における動的問題に対する研究について幾つか紹介する．

2.1 強化学習 (Q学習)

強化学習は，ソフトウェアエージェントからロボットエージェントまで，未知なる環境で振舞う自律的エージェントの学習手法として多く用いられている．ここでは，強化学習と，その手法の一つであるQ学習についての特徴を述べる．

2.1.1 強化学習とエージェント

強化学習とは，強化信号という環境からの特別な入力を手がかりにして環境に適応するという学習手法であり，適応的なエージェントを生み出す手法の一つとしてこれまでに様々な研究が行われている．

一般的な強化学習のモデルを図2.1に示す．各時間ステップ毎にエージェントは現在の状態 s における入力 i を受け取り，出力として行動 a を実行する．行動することにより状態が変化し，その状態遷移に応じて強化信号を受け取る．具体的にモデルは以下の要素からなる．

- 状態空間 S
- 行動の集合 A
- スカラー値の強化信号 (一般に $\{0, 1\}$)

図2.1における I はエージェントが現在の状態をどのように認識するかを決める関数である．また，R は強化信号を与える関数である．

エージェントの目標は，ある時間長さにわたる報酬の和を最大化することである．エージェントの行動方針の性能は一般に，将来にわたる割り引き累加強化信号，すなわち効用により決定される．その効用は，次の式で表される．

$$V_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{2.1}$$

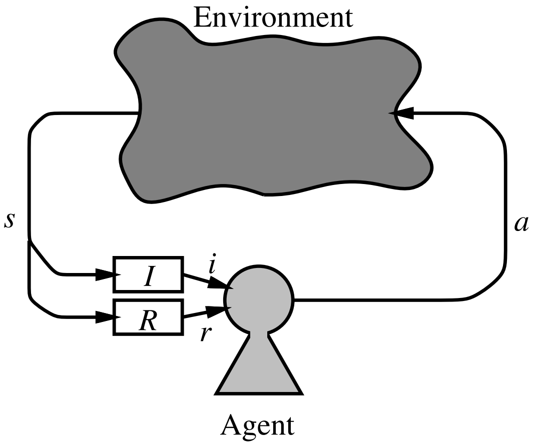


図 2.1: 強化学習のモデル

ここで V_t は時間 t から将来にわたる割り引き累加強化信号であり，r_t は時間 t から $t+1$ の遷移後に受け取る強化信号である．また，$\gamma(0 \leq \gamma \leq 1)$ は割引率であり，これは長期にわたる行動の結果の重要性を調節するものである．γ が 1 に近いほど遠い未来の報酬まで考慮し，0 に近いほど目先の報酬のみを考えた貪欲な戦略をとる．

強化学習の手法は，学習結果としてなるべく大きい報酬を得る行動方針を獲得するという最適性と，学習途中でもなるべく報酬を得続けるという効率性の二つの側面から分類することができる [4]．最適性を重視するアプローチは環境同定型 (exploration oriented) 学習，効率性を重視するアプローチは経験強化型 (exploitation oriented) 学習と呼ばれる．環境同定型学習の例としては，TD(λ) 法 [35]，Q 学習 [38] などが挙げられる．Q 学習は TD(λ) 法の発展型として提案されたもので，TD(λ) 法では各状態について評価を行なうのに対し，Q 学習では状態と行動の対を評価する．また経験強化型学習には主に分類子システム (Classifier System)[20] の枠組の中で研究されてきたバケツリレー (bucket brigade) 法 [19] や利益共有 (profit sharing) 法 [17] などがあるが，これらは行動の最適性を保証しない．以下では本研究で用いる Q 学習について説明する．

2.1.2 Q 学習の枠組

Q 学習は強化学習の手法として広く研究されている．その基本概念は次のような Q 関数と呼ばれる評価関数をエージェントが構築していく事である．

$$Q(state, action) \rightarrow utility$$

この Q 関数は状態と行動の組に対して，Q 値と呼ばれる効用を算出するものである．学習を重ねることにより，報酬を得る状態に近付くほど Q 値が大きくなるように Q 関数が構築されていく．

エージェントが環境中を行動することにより，Q 関数は徐々に更新されながら最適な関数に近づいてゆく．状態 x において行動 a を選択した結果，状態 yとなり強化信号 r が得られたとすると，$Q(x,a)$ の値は次式によって更新される．

$$Q(x,a) \leftarrow Q(x,a) + \beta(r + \gamma \max_b Q(y,b) - Q(x,a)) \tag{2.2}$$

ここで$\beta (0 \leq \beta \leq 1)$ は学習率である．また，$\max_b Q(y,b)$ は，状態 yにおける最大の Q 値である．γ は割引率である。

Q 関数を作る最も単純な方法は，エージェントの遭遇し得る状態と行動に対応する全ての Q 値を表として格納する Q テーブルを用いるというものである．また，状態と行動を入力として Q 値を出力するニューラルネットワークを用い，誤差逆伝播アルゴリズムによって修正していく Q-net[26, 28] を用いる方法なども挙げられる．

強化学習ではエージェントの行動選択が経験の内容を左右する．Q 関数がまだ十分学習された状態でなければ，ある状態 x で $Q(x,a)$ を最大にするような行動が最善な行動とは限らない．このため，エージェントが環境の同定 (exploration) を重視するか，学習中の報酬獲得 (exploitation) を重視するかで行動選択戦略が異なる．ここでは二つの行動選択法の例を述べる．

Watkins は次のような Boltzmann 分布に基づく確率的な行動選択法を提案している [38]．すなわち，状態 x において行動 a を選択する確率を

$$P(a|x) = \frac{exp(Q(x,a)/T)}{\sum\limits_{b \in possible\, actions} exp(Q(x,b)/T)} \tag{2.3}$$

と定義する．ここで T は温度係数であり，値が大きいほど行為はよりランダムになり，積極的な探索を行なう．逆に Tが 0 に近いほど Q 値を最大にするような行動がとられやすくなり，保守的に振舞う．つまり Tの値によって環境同定と報酬獲得のどちらを重視するかを決めることができる．

また，極端に環境同定を指向した例として宮崎は k-確実探査法を提案した [1]．k-確実探査法は，状態遷移の確率過程をできるだけ少ない回数で同定することを目的とする行

動選択手法である．これは状態遷移確率の統計的推定の正しさがルールの試行回数によって単調増加することを利用している．ここでのルールとは，ある状態における行動を指す．あるルールの選択回数が k 回以上であることを k-確実とし，各ルールの選択回数のバラツキを極力小さくするために，k-確実に達しないルールを積極的に選択する．学習器は，すべてのルールが k-確実となったとき k を 1 増加させる．k が十分大きくなって統計的推定が正しくなったとき、最適な行動が保証される．k-確実探査法は，確率的に行動を選択する手法に比べて少ない試行数で最適方針を獲得することができるが，学習中の報酬獲得を一切考慮していない．

2.1.3 Q 学習の問題点

Q 学習は，学習終了時には最適政策の獲得が保証されるが，最適政策を獲得するまでに膨大な行動数を要する．これはエージェントが遭遇し得る状態数や選択できる行動数に比例する．このため，これまでに Q 学習の学習の効率化に関する多くの研究が行なわれている．その一つとして状態空間を分割し，部分的な問題に分けてそれぞれを学習する方法がある．

Shingh は環境モデルの階層的強化学習を行う H-DYNA[32] を提案した．H-DYNA は階層プランニングにおける問題空間モデルの階層化を強化学習に応用したものであり，これにより従来の強化学習より速く学習を行うことを示した．

Lin も，シミュレーション中のロボットが充電器にたどり着くナビゲーションタスクを強化学習によって学習する際に問題を副問題に分割する手法を用いた [27]．Lin は問題を，壁に沿って進行，充電器へのドッキングなどといった副問題に分割し，各副問題を解決するための学習を行なう低い階層と，現在の状況に適切な副問題解決器を選択することを学習する上位の階層を用いる手法を提案し，これにより複雑な問題でも強化学習により効率よく学習できることを示した．

Dayan らも階層的な強化学習の手法を用いることで単体構造の Q 学習よりも速く学習が収束することを示している [15]．

2.2 構造化 Q 学習と AMQL

2.2.1 構造化 Q 学習の枠組

問題を分割，構造化を行う手法の一つとして，Whitehead らはエージェントが直面する問題を分割して強化学習を行うな構造化 Q 学習 (Modular Q-Learning) アーキテクチャ

を提案した[40]．Whitehead らの研究では，複数のゴールを持つような問題に対して，このアーキテクチャを用いることにより Q テーブルを縮小し，学習時間の向上を実現した．

構造化 Q 学習では，複数のゴールを各々モジュールに割り当て，ゴール毎に独立して学習を行なう．各々のモジュールは割り当てられたゴールにのみ着目して行動方針を提案し，それを上位の決定者 (arbiter) がある指針 (多数決など) の基にエージェントの行動を決定する．構造化 Q 学習は，エージェントが目指すゴールが増えたときはモジュールを増やすだけで済むという拡張性を備えている．一方，各モジュールは自分の目標以外の目標を考慮しないため単体構造 (monolithic) の Q 学習に比べて学習の質が劣るという欠点が指摘されている．

多重ゴール問題

Whitehead らは複数のゴールが存在し，エージェントがこれらのゴールへ到達するための行動系列の学習を行なう問題を扱った．Whitehead はグリッド上の異なる場所に水 (water) と食糧 (food) という二つのゴールを置き，グリッド上を動くエージェントがこれらのゴールに到達する方法を，到達した時に得られる報酬を基に学習するという実験を行った (図 2.2参照)．さらにこれらのゴールは時間により報酬が与えられるかどうかが変化する．つまり水と食糧の両方を獲得したい状態，水か食糧のどちらか一方のみを獲得したい状態などが時間により変化する．このためエージェントは様々な状況に応じて適切な行動系列を学習する必要が生ずる．

まず，エージェントに対し n 個のゴール $(\Gamma^1, \Gamma^2, \ldots, \Gamma^n)$ が存在する．各ゴールに対応する状態は一つであり，エージェントはあるゴール状態を満たしたときにのみ報酬を獲得する．ゴールΓ^i に対応する状態を $G^i \subset S$とすると，報酬関数は，

$$R^i(x) = \begin{cases} c^i & \text{if } x \in G^i \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

となる．ただし，Sはエージェントのとる状態の集合，c^iはゴールΓ^iを満たしたときの報酬の大きさを表す．

この時点ではゴールは静的なものとして扱っているが，時間的に変化するゴールを扱うために，ゴールの活性化という概念を導入する．ここでは，ゴールの状態を g^i_tで表し，ある時間 t において，ゴールΓ^iが活性化していれば $g^i_t = 1$ とし，不活性の場合は $g^i_t = 0$ とする．各時間における各ゴールの状態は，以下のように活性ベクトル$\overline{g}$として扱うことにする．

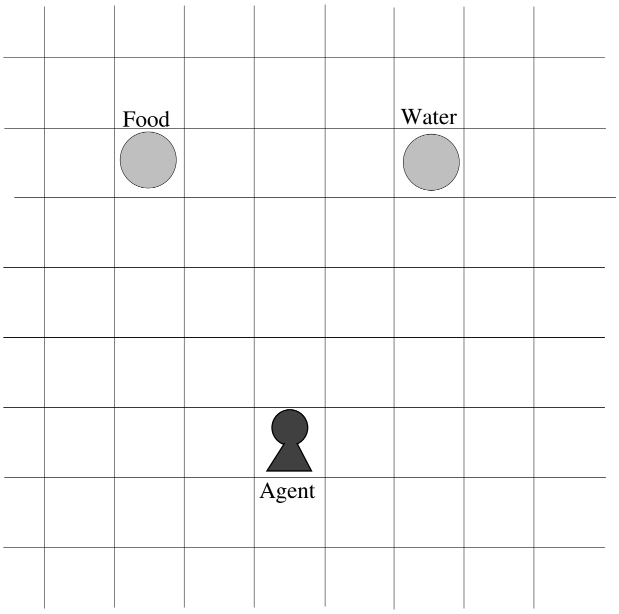


図 2.2: 多重ゴール問題の例

$$\overline{g} = (g^1, g^2, g^3, \cdots, g^n) \tag{2.5}$$

これにより報酬関数を以下のように表すことができる．

$$R(x, \overline{g}) = \sum_{i=1}^{n} R^i(x) g^i \tag{2.6}$$

Whitehead の実験では $g_t^i = 1$ のとき，エージェントがゴールΓ^iにたどり着けば次の時刻 $t+1$ では $g_{t+1}^i = 0$ となり，また 0 となった g_t^iはある確率で 1 になるように設定している．

状態空間の目標による分割

Q テーブルのサイズは単純に考えると $|S|$(エージェントがとり得る状態数) $\cdot$ $|A|$(エージェントが可能な行動数) で表される．しかし Whitehead らが扱った問題では，状態としてグリッド上のエージェントの位置とゴールの活性化状態の両方を考慮する必要がある．よってエージェントの遭遇する状態数は $|S| \cdot |\overline{G}|$ となる．Sはエージェントがとることが可能な位置の集合であり，$\overline{G}$はゴール活性化ベクトルの集合である．つまり，n 個の

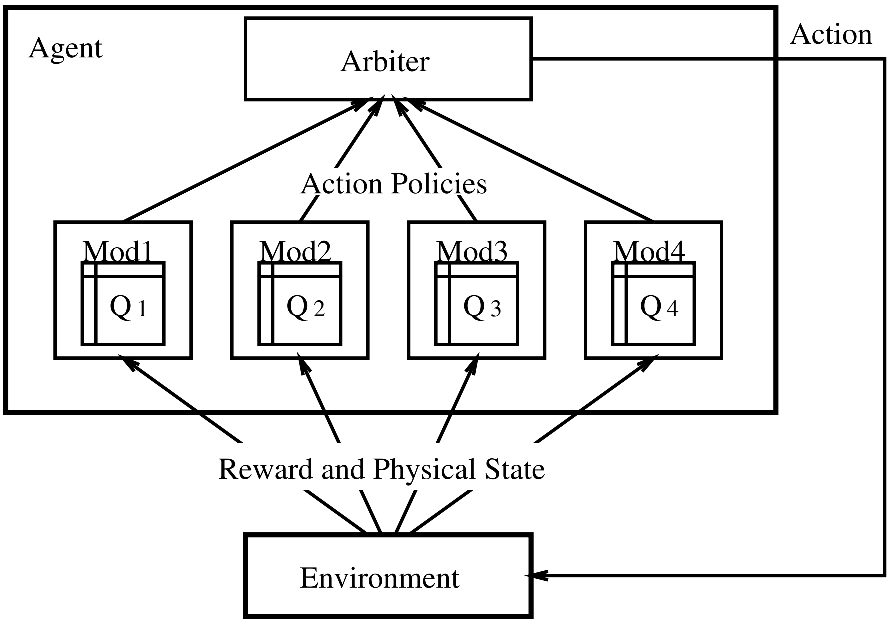


図 2.3: 構造化 Q 学習アーキテクチャ

ゴールがあるとき Q テーブルのサイズは $|S| \cdot 2^n \cdot |A|$ となりゴールが増える毎に指数関数的に増大してしまう．

そこで Whitehead らは 1 つのゴールのみを考慮するモジュールをゴールの数だけ作り，各モジュールにはその特定のゴールのみに注目した行動系列を学習させた．つまり状態空間をゴールを基に分割した．構造化 Q 学習アーキテクチャは図 2.3のようになる．モジュールは特定のゴールのみを考慮した $|S| \cdot |A|$ のサイズの Q テーブルを持ち，それぞれ独立に学習する．

行動決定の戦略

単体構造の Q テーブルを持つ Q 学習では，一つの Q テーブルによって得られる Q 値を基に行動を決定する．これに対し構造化 Q 学習アーキテクチャでは Q テーブルが複数存在するため，これらを考慮した行動方針を選択する戦略が用いられる．Whitehead らは構造化 Q 学習アーキテクチャに用いることができる単純な戦略として最大行動価値優先戦略 (nearest neighbor，以下 NN) と，行動価値総和戦略 (greatest mass merging，以下 GM) を定義している．

NN 戦略はある状態における複数の Q テーブルの各行動を選択した場合の Q 値を比較し，最も大きい値に対応する行動を選択する．NN 戦略における行動方針 f_{nn}は以下の式

で定義される．

$$f_{nn}(x,\overline{g}) = argmax_{a\in A}Q_{nn}(x,\overline{g},a) \tag{2.7}$$

ここで n 個のゴールに対し Q_{nn} は以下の式で表される．

$$Q_{nn}(x,\overline{g},a) = max_{i\in[1\ \ldots n]}[Q^i(x,a)\cdot g^i] \tag{2.8}$$

NN 戦略では，現在の状態で最高の Q 値を持つ 1 つのモジュールのみを考慮するため，その他のモジュールの行動方針が反映されないという欠点を持つ．

これに対し，GM 戦略はエージェントの選択可能な行動全てに対しアクティブな Q テーブルの Q 値を合計し，その合計が最大の行動を取る．いわば各行動に対して多数決をとり，最も希望の多かった行動を選択するため，全てのモジュールの行動方針が反映される．GM 戦略における行動方針 f_{gm} は以下の式で定義される．

$$f_{gm}(x,\overline{g}) = argmax_{a\in A}[Q^i_{gm}(x,\overline{g},a)] \tag{2.9}$$

ここで Q_{gm} は以下の式で表される．

$$Q_{gm}(x,\overline{g},a) = \sum_{i=1}^{n}[Q^i(x,a)\cdot g^i] \tag{2.10}$$

両戦略共に行動は f_{nn} ，f_{gm} により一意に決定され，確率的に探索を行なうような要素は考慮されていない．

このように状態を分けることにより，単体構造の Q テーブルでは $|S|\cdot 2^n\cdot |A|$ のサイズが必要であるのに対し，エージェントの持つ Q テーブルは n 個の $|S|\cdot|A|$ のサイズの Q テーブルへと分割され，一つの Q テーブルのサイズは劇的に減少する．またシングルタスクの場合でも，エージェントの入力が多次元のベクトルで構成されている場合には，入力を基に状態空間を分割して Q テーブルを作成することにより，同様の方法が適用できる．このように Q テーブルを分割することにより単体構造の Q テーブルを持つアーキテクチャと比較して，得られる行動方針が最適ではないものの学習速度が非常に速いことが実験により確認されている．

2.2.2 構造化 Q 学習のマルチエージェントシステムへの応用

構造化 Q 学習をマルチエージェント環境におけるエージェントの学習に応用し，エージェント間の協調的行動を学習することが期待される．小野は，エージェントの協調的行動の学習として，構造化 Q 学習を実装したエージェントによる追跡ゲームを研究した [5]．

追跡ゲーム

追跡ゲームは Benda らによって提案された分散人工知能のためのテストベッドである [11]．2 次元格子 (グリッド) 上に，4 つの追跡エージェントと 1 つの逃亡エージェントが初期配置される．各エージェントは 1 ステップ毎に隣接する四方のいずれかの方向へ移動するか，もしくは静止していることが可能である．4 つの追跡エージェントの目的は，ランダムに移動する逃亡エージェントを四方から取り囲み，捕獲することである．追跡問題において追跡エージェントは単独では逃亡エージェントを捕獲するという目標を達成することは出来ない．目標を達成するためには各追跡エージェントは他の追跡エージェントの行動も考慮し，協調的な行動をとらなければならない．

構造化 Q 学習における追跡ゲーム

小野は，追跡ゲームにおける追跡エージェントに構造化 Q 学習を実装した．これにより，状態空間の組合せの爆発を回避すると共に，それらに分業や群行動を伴う協調的な行動方針を獲得させた．具体的には次のように実現されている．

$n \times n$ のグリッド上の環境に四つの追跡エージェントと一つの逃亡エージェントがランダムに配置される．環境は東西の両端，南北の両端がつながっているトーラス状であり，論理的な端は存在しない．追跡エージェント群が逃亡エージェントを捕獲したら，これらのエージェントは再びランダムに配置され追跡と逃亡を開始する．

追跡エージェントと逃亡エージェントは，各時間ステップ毎に隣接した四方のグリッドへの移動か，またはその場に留まる，という五つの候補から行動を選択する．仲間の追跡エージェント同士は同じ位置を占めることができるが，追跡エージェントと逃亡エージェントは同じグリッドを占めることはなく，それぞれが同じ位置を占めるような行動を選択した場合には，その場に留まる．また，逃亡エージェントは毎回ランダムに行動を選択するのに対し，追跡エージェントはその視界に入る情報に応じて行動を選択する．各エージェントには追跡エージェント 1 ~ 4，逃亡エージェントというように識別子が与えられている．各追跡エージェントは $m \times m (m \leq n)$ の自分を中心とする視界を持ち，

他のエージェントが追跡エージェントの視界に入った場合にはその識別子と自分との相対位置を入力として得る．

追跡エージェントへの構造化 Q 学習の適用

この問題における追跡エージェントの各々を行列形式の Q テーブルを持つ Q 学習器として実装することは原理的には可能である．ある状態における入力を以下のような 4 次元ベクトルで表現し，単体構造の Q テーブルを作れば良い．

$$i^4 = (i^4_1, i^4_2, i^4_3, i^4_E)$$

ここで i^4 は追跡エージェント 4 の入力であり，i^4_1 ,i^4_2 ,i^4_3 は各追跡エージェントに関する入力である．i^j_k は番号 k の追跡者が番号 j の追跡者の視界に入っていればその追跡者の相対座標 (番号 j および k の追跡者の絶対座標の $x-$成分および $y-$成分の各差分の組) を表し，そうでなければ視界に入っていないことを表す特別な記号である．i^j_E は同様に逃亡エージェントの位置に関する情報である．

この方法では，視界内の状態数 N に対し，扱う全状態数は $(N+1)^4 = O(N^4)$ となる[1]．例えば追跡者の視界が 5×5 の場合でさえ，遭遇しうる状態数は $(5^2+1)^4 = 456976$ となり，このような表現を用いて単純に Q 学習を利用するだけでは膨大な記憶領域と学習時間を要することになる．

そこで状態空間を分割し，構造化 Q 学習アーキテクチャを用いる場合を考えてみる．小野は一つの追跡エージェントと逃亡エージェントの情報のみを認識する三つのモジュールを追跡エージェントの学習モジュールとして用意した．これにより視界内の状態数が N の場合 $3(N+1)^2 = O(N^2)$ となり，学習者が扱うべき状態数は劇的に減少する．たとえば，追跡者の視界の広さが 5×5 の場合，各学習モジュールは $(5^2+1)^2 = 676$ 個の状態を扱うだけで済む．

このような枠組で実験を行なった結果，どの追跡エージェントが逃亡エージェントのどの方向から捕獲するかの役割分担ができ，協調関係が生まれた．各モジュールは部分的な情報しか持たないにも関わらず，他のエージェントを考慮した，優れた捕獲行動が学習された．

2.2.3 構造化 Q 学習の問題点

問題を複数のモジュールに分割し，学習を行なうことにより学習の収束速度が向上することを示す研究はこれまでに数多く行われてきており，Whitehead らによって提案さ

---

[1] ここで左辺の「+1」は，視界に入っていない状態を指す．

れた構造化 Q 学習アーキテクチャについても，マルチタスク問題やマルチエージェントの分野において強化学習を行う際の有効性が示唆されている．

しかし構造化 Q 学習を始め，従来の構造化を行なう手法のほとんどは，タスクを分割し，モジュールを作成することが設計者に委ねられていた．このためエージェントは，ある環境でのみ都合の良い，固定したモジュール群を持つことになり，動的に変化する問題に柔軟に対応できないという欠点が存在した．

動的に環境の構造を獲得する手法として Thrun らは SKILLS アルゴリズム [37] を提案している．これは動的に環境の構造を発見し，その構造における複雑な行動方針を一つの SKILL という行動方針にまとめることにより，行動方針に関する記述を少なくし記憶資源を節約することができるというものである．しかし，SKILL アルゴリズムにおいて，環境の構造である SKILL を獲得するためには従来の単体構造による強化学習以上の収束時間を要するという問題点も指摘されている．

構造化を行う目的の一つは，単体構造のアーキテクチャで学習を行うには学習が困難，または学習に時間のかかる問題に対して，より速く効率的に学習を行うことである．構造化を自動的に行うことで，単体構造の強化学習よりも学習の収束までに時間がかかるようでは意味がない．このため，より複雑な問題を学習するためには，適切な構造化を自動的に行い，さらに構造化の利点である学習の収束速度の向上を実現する機構が必要となる．

2.2.4 AMQL アーキテクチャ

上記のような問題に対し，筆者らはエージェントの遭遇し得る状態空間を自動的に複数のモジュールに分割する AMQL(Automatic Modular Q-Learning) アーキテクチャ[25] を提案した．Whitehead，小野らの構造化 Q 学習における問題分割は設計者が行なっていたのに対し，AMQL では問題の適切な分割をエージェント自身が自動的に行なう．筆者らは追跡ゲームを用いたシミュレーションにより，AMQL を実装したエージェント自らが効用を最大化するための構造化を自動的に行ない，さらに問題や環境の変化に対して柔軟に対応できることを示した．AMQL の具体的な機構を以下に示す．

2.2.5 AMQL アーキテクチャの枠組

モジュールが注目する要素の選択

まず，エージェント内には決められた数のモジュールが存在する．ここでエージェントは環境を n 個の要素[2]で認識するとする．ある状態 s におけるエージェントの入力は以下のように表される．

$$i = \{i_1, i_2, \cdots, i_n\}$$

各モジュールはエージェントが環境から受け取る入力 i のうち，ランダムに $1 \sim m(1 \leq m < n)$ 個の要素を選択し，その要素からなる部分状態の Q テーブルを作成する (図 2.4 参照)．これにより，状態空間は複数のモジュールに分割される[3]．

モジュールへの適応度の割り当て

ある行動によってエージェントが報酬を獲得した際に，その行動に貢献したモジュールには適応度が割り当てられる (図 2.5参照)．つまり，適応度は報酬獲得に対するモジュールの貢献度を表す．

まずエージェントは，Whitehead の論文における行動価値総和優先 (GM) 戦略を用いて行動方針を決定する．ある状態 s において行動 a を実行した結果として報酬が得られた場合，行動 a を選択するにあたって各モジュールがどのような行動を希望したかを調べる．各モジュール中の，状態 s において選択可能であった全ての行動に対する Q 値を比較し，その中で行動 a の Q 値が一番大きいモジュール，つまり行動 a を希望したモジュールの適応度を 1 増加させる．適応度は，報酬獲得に貢献する度に加算され，以下に述べるモジュールの選択周期毎に 0 にリセットされる．

モジュールの選択

エージェントは，選択周期 T 毎にモジュールの評価を行なう．評価の際には各モジュールの適応度に注目し，ある閾値を越えたモジュールは報酬獲得に貢献したとみなして残すが，それより低いモジュールは新たに注目する要素を選択し，Q テーブルを作り直す (図 2.6参照)．選択における適応度の閾値 k は以下の式で示される．

$$k = (\text{報酬獲得回数}) \times p$$

---

[2] 自律ロボットの各センサ入力等に対応する．

[3] ただし，全ての要素がいずれかのモジュールにカバーされるとは限らない．

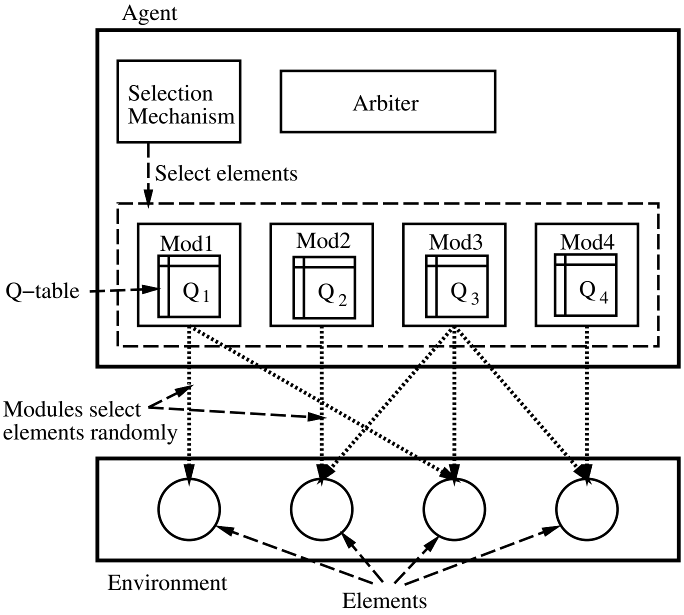


図 2.4: AMQL アーキテクチャ (要素の選択)

ここで $p(0 < p < 1)$ は，報酬獲得回数のうち何割以上報酬獲得に対して適切な行動をしたモジュールが生存するかを決める定数である．得られるモジュールの質は p に依存し，1 に近づける程より貢献度の高いモジュールに対する選択性が強まることになる．しかし p を大きくし過ぎると，偶然報酬を獲得できなかった場合などのミスがモジュールの淘汰に大きく影響するため，環境における状態遷移が決定的でない確率過程の場合にはモジュール群が安定しない可能性がある．

2.2.6 AQML アーキテクチャの問題点

筆者らは前述した AMQL アーキテクチャを用いて実際にシミュレーションを行ない，AMQL アーキテクチャが自動的に適切なモジュール群を獲得できることを示した．さらに環境が変化した際には，環境の変化にあわせてモジュールを変化させることで従来の固定モジュールを持つ構造化 Q 学習よりも柔軟に適応できることを示した．

しかし，AMQL アーキテクチャの評価を行った追跡ゲームは，モジュールの探索においては比較的容易である．松林は，追跡ゲームにおいて 4 つの追跡エージェントと逃亡エージェントの他に，報酬獲得とはまったく関係なく，ランダムに移動するノイズエージェントを加えた実験を行い，AMQL アーキテクチャのモジュール探索能力がそれほど高くないことを示した [6]．この理由として以下のことが考えられる，各モジュールへの

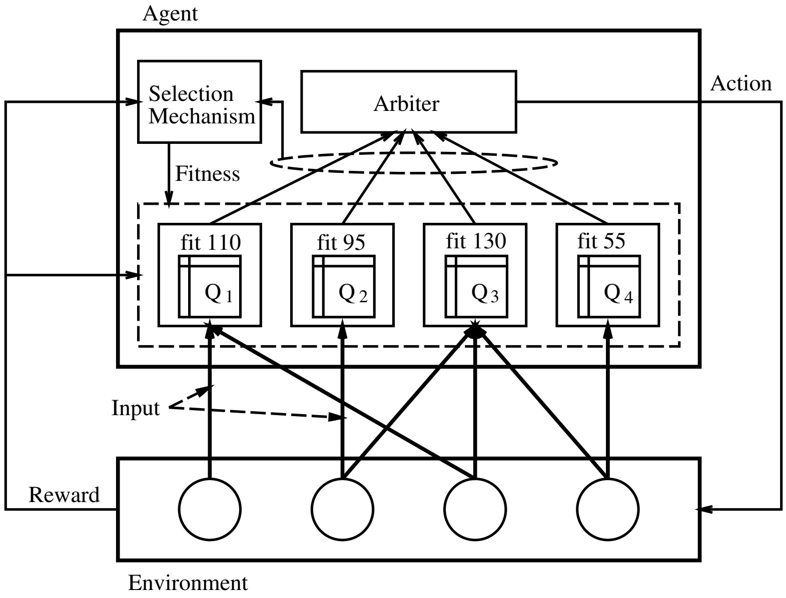


図 2.5: AMQL アーキテクチャ (適応度の割り当て)

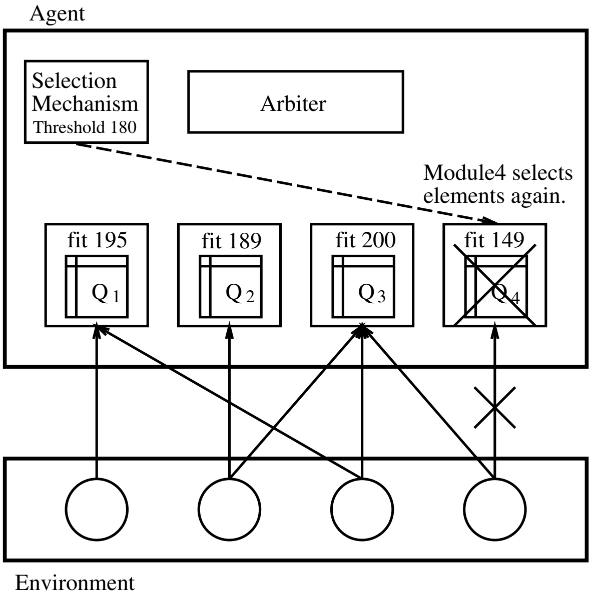


図 2.6: AMQL アーキテクチャ (モジュールの選択)

適応度の割り当てにおいて，AMQL アーキテクチャでは報酬を獲得した時点で適切な行動方針を持つモジュールのみに適応度が与えられる．強化学習が用いられる多くの問題では，報酬を獲得する時点における行動だけでなく，報酬を獲得する状態に至るまでの行動系列を適切に学習する必要があるが，AMQL アーキテクチャでは報酬を獲得するまでの行動系列が評価されていない．また AMQL アーキテクチャでは，モジュールの評価の際に適応度がある閾値を越えた場合にそのモジュールは残される．つまり全てのモジュールの適応度が閾値を越えるようになった時点でそれらを適切なモジュール群とみなし，次に環境の変化などにより適応度が閾値以下になるまで安定する．適切なモジュールセットの獲得とは，モジュールの全ての組合せを解空間とする探索に相当する．しかし AMQL アーキテクチャの方法では，ある時点における全ての組合せのうち一点についてのみ評価を行なっている．この場合，特に探索を行う空間が非常に広い場合に，局所解に陥ってしまい，最適なモジュールを獲得できない場合がある．

2.2.7 AMQL-GA アーキテクチャ

上記の AMQL アーキテクチャの問題点に対し，松林は適切なモジュールセットの獲得に遺伝的アルゴリズム[4]の手法を用ることで，改良を図った．図 2.7に AMQL-GA アーキテクチャを示す．遺伝的アルゴリズムを用いて適切なモジュールセットを獲得するために，各モジュールが注目する環境の要素はビット列の染色体として表される．例えば，追跡ゲームにおいて追跡エージェント 4 が $i_1^4, i_2^4, i_3^4, i_E^4$の全てを入力として単一構造の Q テーブルを作成する場合は ”1111” という染色体で表される．また，小野が用いたモジュールセット (逃亡エージェントとそれぞれ異なる追跡エージェントに注目する 3 つのモジュールからなる) を染色体で表現すると ”1001 0101 0011” となる．

エージェントはモジュールセットの基となる染色体を複数持ち，それぞれを順番に選択し，染色体を基に作成したモジュールを用いて実際に学習させることにより適応度を定める．全ての染色体について適応度を定めた後に，適応度を基に交叉，突然変異などの遺伝的操作を行い次の世代の遺伝子集団を構築する．具体的なアルゴリズムを以下に示す．

---

[4] Appendix 参照

1. ランダムに染色体を生成する．
2. 各染色体の評価を行なう．
    - (a) 染色体に基づきモジュールセットを生成，Q 値の初期化を行なう．
    - (b) j(一つの染色体の評価回数) $:= 1$，$trial$(試行回数) $:= 0$．
    - (c) 試行を行なう
        - i. $trial$を 1 増加する．
        - ii. エージェントを初期状態に配置する．
        - iii. 状態 s を更新する．
        - iv. GM 戦略に基づき行動を選択し，行動する．
        - v. Q 値を更新する．
        - vi. 遷移した状態がゴールであるか，もしくは一回の試行の終了条件が満たされれば試行を終了する．
        - vii. 2(c)iii へ戻る．
    - (d) $trial < T$(最小試行回数) であれば 2(c) へ戻る．
    - (e) $E_j := ((trial - T)$ 試行目から $trial$試行目までの間の一試行に要した時間ステップの平均)．
    - (f) 以下の条件を満たしている場合，jを 1 増加し 2(c) へ戻る．
        - $E_j < E_{j-1}$[a]
        - $E_j < E_{tar}$(平均時間ステップの目標値)
    - (g) F(適応度) $:= E_{max} - E_{j-1}$[b]．
3. Fに基づき二つ染色体を選択する．
4. 染色体に対してモジュール交換，突然変異を行い次世代の染色体を作る．
5. 次世代の染色体数が集団サイズに達していなければ 3 へ戻る．
6. 2 へ戻る．

---

[a] E_{i-1}の初期値は E_{max}(E_jの取り得る最大値) とする．
[b] 評価時における Eの最小値を意味する．

松林は，幾つかのノイズエージェントを含む追跡ゲームにおいて，AMQL-GA アーキテクチャの評価を行った．これにより，AMQL アーキテクチャでは適切なモジュールを獲得できないような困難な問題でも，AMQL-GA アーキテクチャを用いることにより適切なモジュールが獲得出来ることを示した．しかも最終的に得られたモジュールは，設計者によって作られた固定モジュールセット (小野らのものと同様 [30]) よりも収束後の学習性能が優れていた．

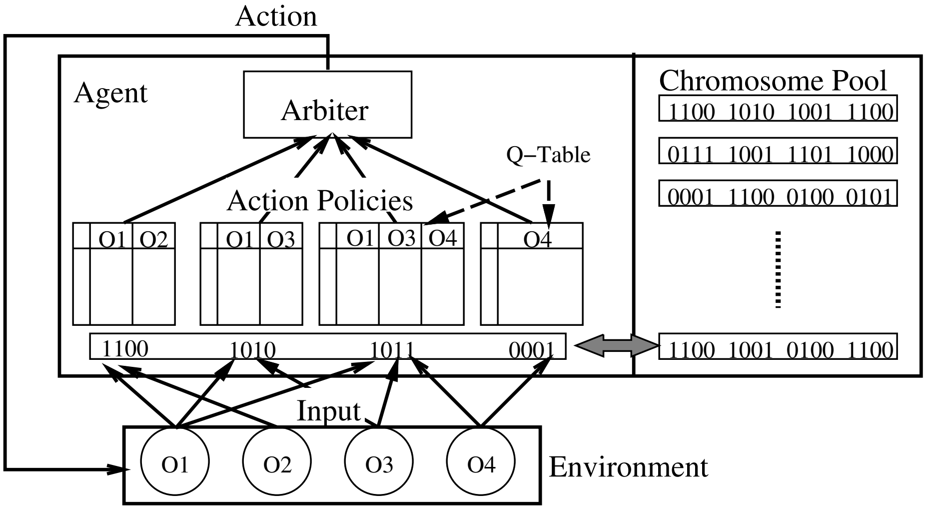


図 2.7: AMQL-GA アーキテクチャ

AMQL-GA アーキテクチャでは，問題が変化して既存の行動方針が適切でなくなると，$E_j < E_{tar}$ が満たされなくなることがトリガーとなって再度探索を行うという興味深い枠組を与えてはいるものの評価などは行われていない．

2.3 問題の変化と学習

未知なる環境で適切に振舞うエージェントの構築や，環境が既知であったとしても解決方法を予めエージェントに与えることができないような問題の解決手段としてさまざまな学習機構が提案された．これらの多くの研究は静的な問題を仮定している．これは，学習機構の基本的性能や学習の振舞いを分析するためには静的な問題を仮定することが適切であるからと考えられる．しかし実世界は動的に変化し，静的な問題を仮定した学習機構が有効に動作する保証はない．

ここでは動的環境に対して強化学習を用いた研究例についていくつか述べ，またそれらの問題点を指摘する．

2.3.1 問題の変化と強化学習

強化学習において問題のクラスを分類する場合には，まずマルコフ問題と非マルコフ問題に分けることができる．強化学習に関する従来の研究の多くは，マルコフ決定過程 (Markov decision process，MDP) を対象としている．これは環境を確率状態遷移グラフでモデル化したとき，環境の状態遷移確率が一定かつ不変な性質を指す．マルコフ的な

環境は静的であり，このような環境に対してQ学習などの環境同定型の強化学習方法は最終的に最適行動方針を得ることを保証している[39]．

非マルコフ問題としては様々なものがあり，静的な問題と動的な問題がある．代表的なものとしては，環境がマルコフ的に状態遷移しているとしても，感覚が不完全なために異なる状態が区別できず，エージェントにとっては非マルコフ的に見える場合がある．これは隠れ状態問題(hidden state problem)，perceptual aliasing，部分観測マルコフ決定過程(Partially Observable Markov Decision Process, POMDP)などと呼ばれている．隠れ状態問題に対するアプローチとしては，リカレントコネクションを利用して過去の情報から予測モデルを構築する方法などがある[13, 14, 28]．

動的問題としては，状態遷移確率やゴールが動的に変化する問題が考えられる．Whiteheadが構造化Q学習のテストベッドとして利用した多重ゴール問題[40]ではゴール状態が動的に変化するが，ここでのゴール状態の変化とはエージェントの内部で起こるものであり，エージェントはゴールの変化を知ることができる．エージェントが問題の変化を把握できる場合はエージェントの内部でモジュールを切り替えれば良いため，結局複数のタスクを別々に解決することと同様であるといえる．本論文において問題の動的変化を述べる際は，エージェントが変化を直接的に観測できないものとして扱う．

幾つかの研究では，緩やかな問題変化に対して既存の学習内容を部分的に修正することにより，強化学習の機構でも動的問題においても学習できることを示している．Suttonは，未訪問の場所をある程度積極的に訪問することにより，変化した問題に対してより適した行動方針を学習するDyna-Qシステムを提案した[34]．Dyna-Qでは，状態xにおいて行動aを選択した後にQ関数の$Q(x, a)$の値を更新するが，その際に以前に状態xで行動aを選択してから経た時間ステップに応じて探索ボーナス(exploration bonus)が与えられる．つまり，長らく取っていなかった行動を選択した場合に付加的な得点が得られ，これによってある程度の冒険をするようになる．Suttonは，グリッド上のエージェントがゴール地点にたどり着くことが目的であるナビゲーションタスクにおいて，最初の環境における最短経路上に動的に障害物が生じる問題(ブロッキングタスク)と，最初の環境においてあるグリッド上に存在した障害物がなくなることにより新たな最短経路が生じる問題(ショートカットタスク)に上記の機構を適用した．これにより環境の変化に対してある程度探索することで既存の学習内容を修正し，より良い経路を学習できることを示した．

また，徐々に変化する問題としてマルチエージェントにおける強化学習が挙げられる．各エージェントが他のエージェントの状態を基に行動決定するような枠組でエージェントが学習を行なう場合には，各エージェントの行動方針の変更が互いに影響し合うこと

により集団のダイナミクスが形成される．つまり，各エージェントが行動選択確率を徐々に修正することが，環境の状態遷移確率が徐々に変化することに対応している．Tan は複数の補食者が協力して被食者を捉える問題に強化学習を用いている [36]．Tan は学習に Q 学習を用い，感覚，エピソード，行動方針などを共有することで，各エージェントが独立に学習するより協調行動を効率的に学習できることを示した．しかし，通信などの補助的手段を用いて学習過程を制御する必要があり，枠組として不自然になりがちであるという指摘もある [3]．

また，断続的に変化する問題へのアプローチとして，環境の変動を，観測可能な情報から予測し，変化毎に環境モデルの文節化を行なう方法が提案されている．山口は，実ロボットが実環境下で行なう学習と，実環境を基に構築された仮想環境での高速学習を並列に行なう並列環境強化学習の枠組みにおいて，動的環境における従来の環境のモデル化手法の問題点と環境の変動を推定する方法について述べている [2]．山口は環境のモデルとして遷移観測マルコフ決定性 (TOMDP) を定義している．これはエージェントの観測が部分的であり，ある期間 i にエージェントが観測した環境のモデルを TOMDP(i) とすると真のマルコフ決定過程は学習エージェントから見て TOMDP の遷移としてモデル化可能であるというものである．つまり，観測可能な情報の変動から観測不可能な真の状態の遷移を推定する．また，これに応じて見かけの状態遷移確率の違いから環境モデルを分割し，変動する環境のモデルを構築する．しかし，課題として TOMDP モデルを用いた並列強化学習の実験が残されている．

2.3.2 動的問題における既存の強化学習機構の問題点

Q 学習などに代表される幾つかの強化学習の基本的な枠組では，マルチエージェントにおける強化学習の研究が示すように，問題の緩やかな変化に対して学習結果を適応的に追従させることが可能である．しかし，急激な変化が起こる場合には，以前に学習した内容に学習性能が大きく依存し，初期状態から学習するよりも時間がかかるかもしくは局所解から抜け出せないことにより適切な行動が学習できない場合がある．この理由として学習内容が特定の問題に特化しすぎる過学習が考えられる．

異なる問題を学習する手法としては，Whitehead の提案した構造化 Q 学習に代表されるようにモジュール化を行ない，問題毎に別々のモジュールで学習を行なう方法がある．急激に問題が変化する場合にも，複数のモジュールを用意し，問題が変化する度に適切なモジュールを用いることにより対応する方法が考えられる．ただし，問題ごとに別々のモジュールで学習を行なうには，エージェントが問題の変化を知り，問題に適したモジュールを獲得する必要が生ずる．山口は環境の観測可能な状態遷移確率の時系列的な

変動から環境の断続的な変化を推定する手法について述べている．しかしエージェントが，問題がどのように変化したかを知ることは非常に困難である．このため，実際に環境の中で学習を行い評価を行なうことにより適切なモジュールを特定する必要が生ずる．

著者らの提案した AMQL アーキテクチャは，現在の問題に適切なモジュールセットを環境との相互作用によって自動的に獲得することにより，動的環境への有効性を示した．また，AMQL-GA アーキテクチャは遺伝的アルゴリズムを用いることにより，ノイズエージェントがいるようなより高い探索能力が必要な場合でも，適切なモジュールセットについて広域探索を行なうことができる．AMQL-GA アーキテクチャは適切なモジュールセットの獲得能力が優れているだけでなく，環境の変化をモジュールセットの適応度の変化から自分で判断する望ましい枠組を提供している．しかし，探索能力の向上の重きが置かれており，選択淘汰の際にモジュールセットの構成のみを継承し，強化学習によって学習した内容は継承されないといったように，動的性質には疑問が残る．AMQL-GA アーキテクチャでは，モジュール獲得に遺伝的アルゴリズムを用いているが，遺伝的アルゴリズム自体は，問題の動的に変化する場合には集団が以前直面した問題の解付近に収束するために多様性が失われ，新しい問題に対して局所解に陥りやすいという問題が指摘されている [8]．また，強化学習によって獲得する行動方針を考えた場合，問題が急激に変化する場合は，学習内容が継承されないために特定の問題における過学習による弊害はないが，問題の変化が小さく，以前に獲得した行動方針を部分的に修正すれば良いような場合には，効率的でない．

Q 学習など従来の強化学習手法は，問題変化が小さい場合には適応できるが，変化が大きい場合には過学習の影響により変化に追従できない場合がある．逆に，AMQL-GA アーキテクチャは過学習の影響はないが，問題変化が小さい場合にはあきらかに効率が悪い．現在の問題が過去に学習した問題に類似した場合には過去の学習内容を適用し，問題が急激に変化する際には最初から学習しなおす戦略が有効であると考えられる．

第 3 章

AMQL-LL アーキテクチャの提案

3.1 AMQL-LL アーキテクチャ

本論文では，AMQL-GA アーキテクチャの枠組においてエージェントが学習内容をモジュール単位で保持し，既学習モジュールを問題の類似度に応じて適用する機構であるAMQL-LL(Automatic Modular Q-Learning with Lamarckian Learning) を提案する．

AMQL-LL アーキテクチャは，松林の提案した AMQL-GA アーキテクチャを基に構築したものである．両アーキテクチャは，共に Whitehead が提案した構造化 Q 学習 [40] を枠組とし，選択淘汰のメカニズムを用いて自動的に適切なモジュールセットを獲得することができる．つまり，エージェントの中の複数のモジュールセットを評価，選択淘汰を行ない，最終的に現在の問題に対して適切なモジュールセットを獲得する．ここで，エージェントは選択淘汰による適切なモジュールセットの獲得と，各モジュールにおける，強化学習を用いた行動方針の学習というレベルの異なる二つの学習を行なう．AMQL-GA アーキテクチャは，選択淘汰の際にモジュールの基本構造に関する情報のみを継承し，行動方針についてはモジュールセットの評価毎に，初期化された状態から学習を行なう．これに対し，AMQL-LL アーキテクチャでは，モジュールセットの選択淘汰の際に，これまでに学習した行動方針を継承する．これは，AMQL-GA アーキテクチャが後天的に獲得した形質を継承しないというダーウィンの進化論の立場を取るのに対し，AMQL-LL アーキテクチャは獲得形質の遺伝という点でラマルク的立場を取ると言える．AMQL-GA アーキテクチャは，学習内容を継承しないという性質上，問題変化が小さく，過去の学習内容を部分的に修正するだけで済むような場合でも，まったく初期化された状態から学習を行なわなければならない．これに対し，AMQL-LL アーキテクチャでは，問題変化が小さい場合には過去の学習内容を利用し，効率良く学習を行なうことができる．また，突然変異として新たに初期化されたモジュールが生成されることにより，問題が大きく変化し，従来の Q 学習では過学習による弊害が出るような場合にも，新しいモジュールを利用することで適切な行動方針を学習することができる．

まず，AMQL-LL アーキテクチャのアルゴリズムの概要について説明する．AMQL-LL アーキテクチャは以下の手順を繰り返すことによって適切なモジュールセットを獲得する．

- モジュールセット群の生成
- 各モジュールセットの評価・適応度の割り当て
- モジュールセットの選択淘汰

まずエージェントは図 3.1に示すように複数のモジュールセットを持つ．ここでのモジュールとは環境の部分入力について学習を行う Q 関数であり，部分的な問題について学習を

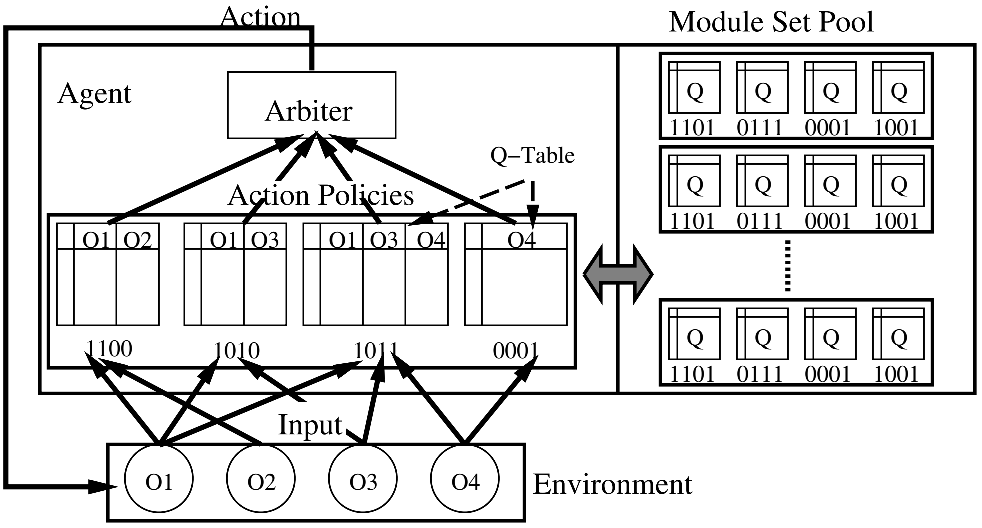


図 3.1: AMQL-LL アーキテクチャ

行なう．エージェントは複数のモジュールをモジュールのセットとして組み合わせることにより一つの問題に対して学習を行なう．

AMQL-LL アーキテクチャは，エージェント内に存在する複数のモジュールセットに対して，それぞれ順番に評価を行なう．評価の際には，モジュールセットを用いて実際に行動，学習を行ない，行動の結果を基に適応度を割り当てる．

全てのモジュールセットの評価を終えた後に，選択淘汰のメカニズムを用いて新たなモジュールセットを作る．まず適応度を基に二つのモジュールセットを選択し，交叉を行なう．これにより優れたモジュールがモジュールセットの集団に繁栄することになる．また，突然変異によって新たに初期化されたモジュールが生成される．これにより新たに遭遇した問題に対しても過学習に影響されることなく学習する可能性を確保する．具体的な機構を以下に示す．

3.1.1　モジュールセット群の生成

AMQL-LL アーキテクチャにおいて，エージェントは $N_{mod\,all}$ 個のモジュールを保持する．モジュールとは，環境の部分状態に関して学習を行なう Q 関数であり，ここでは Q テーブルを用いている．エージェントは環境を n 個の要素 $(O_1, O_2, ..., O_n)$ に分割して認識する．ここで要素とは，ロボットエージェントにおける各センサや，マルチエージェント環境における他のエージェントの状態などに相当する．要素 O_j に関する入力を i_j としたとき，エージェントの入力 i は以下のように表される．

$$i = (i_1, i_2, \cdots, i_n)$$

まず，エージェント内の全てのモジュールは環境の入力要素から，ランダムに$1 \sim m(1 \leq m < n)$個を選択する．例えば，四つ要素がありO_1とO_2を選択する場合，どの部分状態に注目するかという情報は，注目する要素を1，注目しない要素を0として”1100” で表現される．次に全てのモジュールは，この注目する要素によって構成される部分空間に関するQテーブルを作成する．AMQL-GAアーキテクチャでは，モジュールセットの評価毎に注目する部分入力の情報を基にQテーブルを作成するが，AMQL-LLアーキテクチャではQテーブルを予め作成しておき，淘汰されるまでなくならない．

各モジュールは評価の単位であるモジュールセットに属する．モジュールセットはN_{mod}個のモジュールからなる．エージェントは$N_{mod\,set}$個のモジュールセットを保持し，その一つ一つのモジュールセットが選択淘汰のメカニズムにおける個体に相当する．各モジュールは一つのモジュールセットにのみ属すため，以下のような関係が成り立つ．

$$N_{mod\,all} = N_{mod} \cdot N_{mod\,set}$$

3.1.2　モジュールセットの評価・適応度の割り当て

生成されたモジュールセットは，評価のため順番にエージェントの学習に用いられる．まず，一つのモジュールセットが選択される．エージェントは，各時間ステップ毎にWhiteheadの論文における行動価値総和優先(GM)戦略を用いて行動方針を決定する．

$$f_{gm}(x) = argmax_{a \in A} \sum_{i=1}^{N_{mod}} Q^i(x, a) \tag{3.1}$$

ここで，f_{gm}は状態xにおけるエージェントの行動方針である．可能な行動全てに対しそれぞれN_{mod}個のモジュールのQ値を合計し，合計値が最大の行動方針を選択する．

初期状態から報酬獲得まで，もしくは終了条件(最大時間ステップ)を満たすまでを一試行とした場合，最低試行回数であるT試行まで同一のモジュールセットを用いて実行する．ここで，T試行間に要した時間ステップをTで割ったもの，つまり一試行に要した時間ステップの平均をEとすると，Eが以下の条件を満たした場合には，次のT試行も同じモジュールが用いられる．

$$\begin{cases} E_j < E_{j-1} \\ E_j < E_{tar} \end{cases} \tag{3.2}$$

ここで T試行を一回の評価の単位とすると，j回目の評価における Eの値を E_jとする．E_0は Eの取り得る最大値 E_{max}とする．3.2 式における上部の条件は，最近の平均時間ステップが以前の結果と比較して良くなっている，つまり学習が進んでいる場合には続けて評価を行なうことを示す．E_{tar}は目標平均時間ステップを示す．下部の条件は，平均時間ステップが目標値に達した場合は，モジュールセットの使用を継続することを表し，適切なモジュールセットが獲得された状態とは常にこの条件が満たされる場合を指す．

E_jを評価する際に 3.2 式の条件を満たせなくなった時点でこのモジュールセットに適応度を割り当て，次のモジュールセットの評価を行なう．適応度は最大時間ステップ E_{max}から，これまで計算された E_jの最小値を引いた値とする．具体的には以下の式で表される．

$$F := E_{max} - E_{j-1} \tag{3.3}$$

ここで，E_{j-1}は E_jの最小値を示す．これは，モジュールセットに適応度を割り当てる際には，常に $E_{j-1} < E_j$の関係が成り立つからである．3.3 式により，短い時間で報酬を獲得できるほど，大きな適応度が割り当てられることになる．

$N_{mod\,set}$個のモジュールセット全ての評価を終えた後にモジュールセットの選択淘汰を行なう．

3.1.3 モジュールセットの選択淘汰

評価を基に適応度を与えられたモジュールセットに対し選択，交叉，突然変異などの操作を行ない，新たなモジュールセット群を構築する．AMQL-GA においては，各モジュールが注目する要素をビット列で表し，ビット単位で交叉，突然変異を行ったが，AMQL-LL ではモジュール単位で交換，突然変異を行なう．

まずエリート保存戦略を用い，適応度の一番高いモジュールセットを次世代に残す．残りのモジュールセットは以下に述べるように選択的に構成する．はじめに親となる二つのモジュールセットを適応度比例戦略[1]を用いて選択し，選択された二つのモジュールセット間でお互い同数のモジュールを交換する (図 3.2参照)．この操作は遺伝的アルゴリズムにおける交叉に相当する．ただし，幾つのモジュールを交換するかは 0 ~ $(N_{mod} - 1)$の間でランダムに決定する．また，遺伝的アルゴリズムにおける遺伝子座に相当するも

---

[1] Appendix 参照

のは存在せず，両モジュールセットで，どのモジュールを交換するかをそれぞれ決定し，交換する．交換により生成されるモジュールセット中のモジュールは，親の学習内容を継承する．

次に，モジュール交換により新たに生成されたモジュールセットに対し，突然変異を行なう．具体的にはモジュールセット中のモジュールを，ある確率で新しく生成されたモジュールに交換する操作を行なう(図 3.3 参照)．新たに生成されるモジュールは注目する環境の部分をランダムに選択し，Q テーブルを作成する．以上の操作が新たなモジュールセットの数が N_{modset}を満たすまで繰り返される．

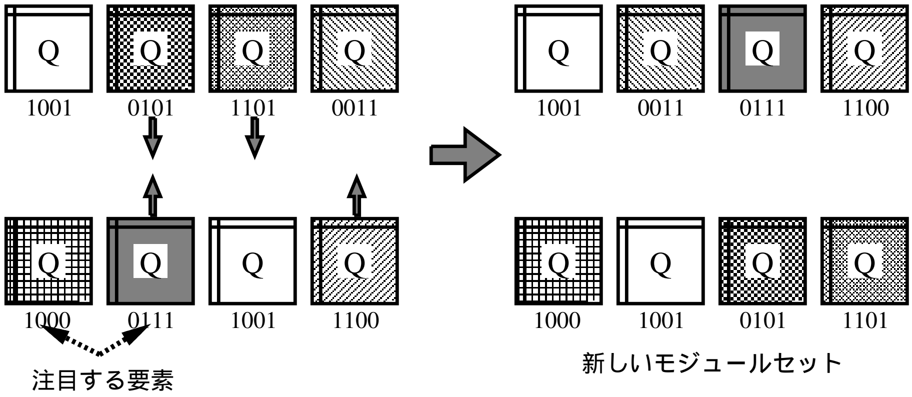


図 3.2: モジュールの交換

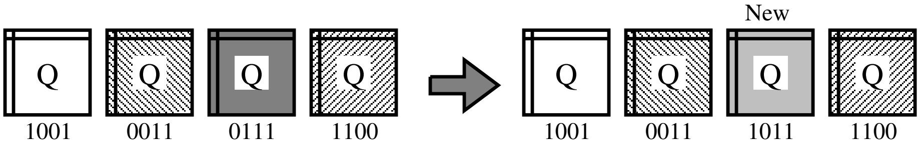


図 3.3: モジュールの突然変異

3.2　アルゴリズム

AMQL-LL のアルゴリズムを以下にまとめる．

1. $N_{mod\,all}$個のモジュールをランダムに生成し，これらを N_{mod}個のモジュールからなる $N_{mod\,set}$個のモジュールセット集団とする．

2. 各モジュールセットの評価を行なう．

   (a) まだ評価を行っていないモジュールセットを選択する．

   (b) j(選択されたモジュールセットの評価回数) $:= 1$，$trial$(試行回数) $:= 0$．

   (c) 試行を行なう．

      i. $trial$を 1 増加する．

      ii. エージェントを初期状態に配置する．

      iii. 状態 s を更新する．

      iv. GM 戦略により行動を選択し，行動する．

      v. Q 値を獲得した報酬に基づき更新する．

      vi. 遷移した状態がゴールであるか，もしくは 1 回の試行の終了条件が満たされれば試行を終了する．

      vii. 2(c)iii へ戻る．

   (d) $trial < T$(最小試行回数) であれば 2(c) へ戻る．

   (e) $E_i := ((trial - T)$ 試行目から $trial$試行目までの間の一試行に要した時間ステップの平均)．

   (f) 以下の条件を満たしていれば，jを 1 増加し 2(c) へ戻る．

      - $E_j < E_{j-1}$
      - $E_j < E_{tar}$(平均時間ステップの目標値)

   (g) F(適応度) $:= E_{max} - E_{j-1}$

3. Fを基に二つのモジュールセットを選択する．

4. 選択されたモジュールセットについてモジュール交換，突然変異を行ない新たなモジュールセットを生成する．

5. モジュールセット集団が N_{modset}に達していなければ 3 へ戻る．

6. 2 へ戻る．

第 4 章

AMQL-LL アーキテクチャの評価

4.1 実験の枠組

本論文では，モジュールセットの獲得の難しさ，報酬獲得の難しさという点から追跡ゲームを用いて，動的に変化する問題に対するAMQL-LLアーキテクチャの追従性についての評価を行なう．まず，基本的な追跡ゲームを用いた実験により，基本的な学習性能について従来のアーキテクチャと比較を行なう．次に，問題の変化が小さく，過去の学習内容が利用できる場合におけるAQML-LLアーキテクチャの収束性についてAMQL-GAアーキテクチャとの比較を行なう．さらに，変化後の問題において過学習が妨げとなる場合における学習能力について，従来の固定されたモジュールを持つアーキテクチャと比較を行なう．

4.1.1 追跡ゲームの基本設定

まず，本論文における追跡ゲームの基本的な設定について述べる．

- グリッドサイズ 10×10．東西の両端および南北の両端はそれぞれ接続されており，理論的な端は存在しない(トーラス)．
- 追跡エージェントおよび逃亡エージェントの初期位置はランダムに決定される．
- 逃亡エージェントと追跡エージェントは，各時間ステップ毎に東西南北のいずれかの方向に一歩進むか，現在の場所に留まるかの五種類の候補の中から行動を一つ決定する．しかし，追跡エージェントと逃亡エージェントは同じグリッドに位置することはできない．このため逃亡エージェントが追跡エージェントの存在する位置に移動する行動を選択した場合は，現在の場所に留まる．逆に追跡エージェントが逃亡エージェントの位置への移動を試みる場合も同様である．ただし，複数の追跡エージェント同士は同じグリッドを占めることができる．
- 逃亡エージェントはランダムに行動を選択する．
- 追跡エージェントの視界は，自分を中心とする 5×5 のグリッド内に限られる(図4.1参照)．また，各エージェントには互いを区別する識別子が付けられており，追跡エージェントはその視界内にいる他のエージェントの識別子と自分との相対位置を入力として得る．
- 追跡エージェントが逃亡エージェントを捕獲するか，捕獲できずに2000時間ステップ経った時点で1回の試行を終了する．

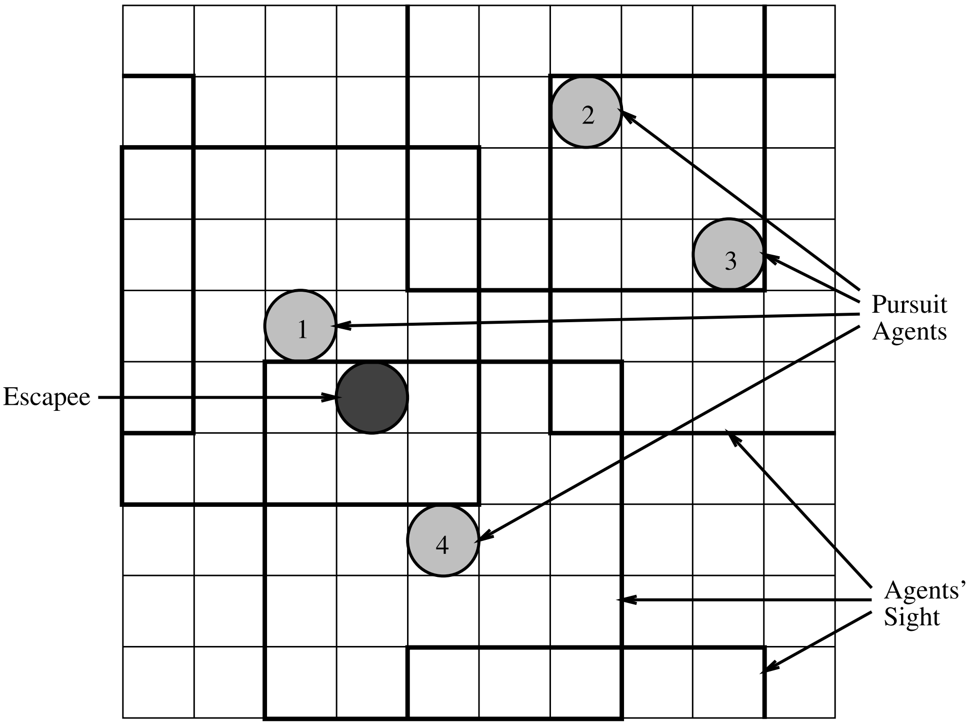


図 4.1: 本実験の追跡ゲームの例

4.1.2 Q 学習に関する設定

各モジュールの Q テーブルは，2.1.2式に基づいて Q 値を更新する．ここでは全ての実験において，定数は表 4.1のように設定した．各追跡エージェントは逃亡エージェントを捕獲した時のみ報酬を獲得する．

表 4.1: Q 学習における定数

	値
学習率 β	0.1
割引率 γ	0.9
初期 Q 値	0.1
報酬	1

4.2 基本問題における学習性能の評価

まず基本的な追跡ゲームの枠組を用い，AMQL-LL アーキテクチャの収束速度や収束後のパフォーマンスなどについて評価を行なった．また，ここでは従来の機構として単体構造 (monolithic) の Q 学習，構造化 Q 学習アーキテクチャ，AMQL-GA アーキテクチャなどとの比較を行なった．各アーキテクチャについて 80000 試行までの実験を 30 回行なった．

4.2.1 各学習アーキテクチャの設定

追跡ゲームにおける追跡エージェントは，自分以外のエージェントの状態を入力として受け取る．例えば追跡エージェント 4 の場合，以下の要素を入力とする．

$$\{i_1\ ,i_2\ ,i_3\ ,i_E\}$$

ここで 1 ,2 ,3 は自分以外の追跡エージェントの識別子で，i_1は番号 1 の追跡エージェントが視界に入っていれば，その追跡者の相対座標 (追跡エージェント 4 および追跡エージェント 1 の絶対座標の $x-$成分および $y-$成分の各差分の組) を表し，入っていなければ追跡エージェント 1 が視界に入っていないことを表す特別な記号である．一方，i_Eは同様に逃亡エージェントについての位置に関する入力である．

本実験における各学習アーキテクチャの設定の詳細を以下に記す．

AMQL-LL アーキテクチャの設定

AMQL-LL アーキテクチャにおいてモジュールを生成する時，1 ~ 3 個の自分以外のエージェントをランダムに選択し，これらの状態を入力とする Q テーブルを作成する．また AMQL-LL アーキテクチャにおける各パラメータを表 4.2に記す．

AMQL-GA アーキテクチャの設定

AMQL-GA アーキテクチャの追跡ゲームに対する設定は，第 2.2.7節で述べたものと同様である．AMQL-GA アーキテクチャにおける各パラメータは表 4.3のように定める．表 4.2におけるモジュールセット数 $N_{mod\,set}$とモジュールセットの持つモジュール数 N_{mod}は，それぞれ表 4.3における染色体数 $N_{chromosome}$と染色体の持つモジュール数 N_{mod}に対応する．

表 4.2: AMQL-LL アーキテクチャのパラメータ

	値
全モジュール数 $N_{mod\,all}$	40
モジュールセット数 $N_{mod\,set}$	10
モジュールセットの持つモジュール数 N_{mod}	4
最低試行回数 T	80
目標平均時間ステップ E_{tar}	50
突然変異率	0.2

表 4.3: AMQL-GA アーキテクチャのパラメータ

	値
染色体数 $N_{chromosome}$	10
染色体の持つモジュール数 N_{mod}	4
最低試行回数 T	80
目標平均時間ステップ E_{tar}	50
突然変異率	0.05
交叉率	0.6

固定アーキテクチャの設定

単体構造のQ学習は，エージェントの取り得る全ての状態に関するQテーブルを作成する．また，構造化Q学習の場合，エージェントは三つのモジュールを持ち，各モジュールは逃亡エージェントと互いに異なる他の追跡エージェントの二つに注目したモジュールとする．これは小野の使用したものと同様である[30]．

4.2.2 結果と考察

図4.3～図4.6に，各アーキテクチャを用いた場合の各試行における捕獲までに要する時間ステップを示す．結果は30回の実験の平均である．ただし，図4.4に関しては収束が非常に早いため，3000試行までの結果のみを示す．また，表4.4に各学習アーキテクチャにおける収束までに要する試行数及び収束後の捕獲に要する平均時間ステップを示す．追跡ゲームでは各試行において捕獲に要する時間ステップが初期位置や逃亡エージェントの動きなどランダムな要素に左右されるため一定の値に収束しない．このため，最後の79901試行から80000試行までの平均時間ステップを収束値として用いた．また，各

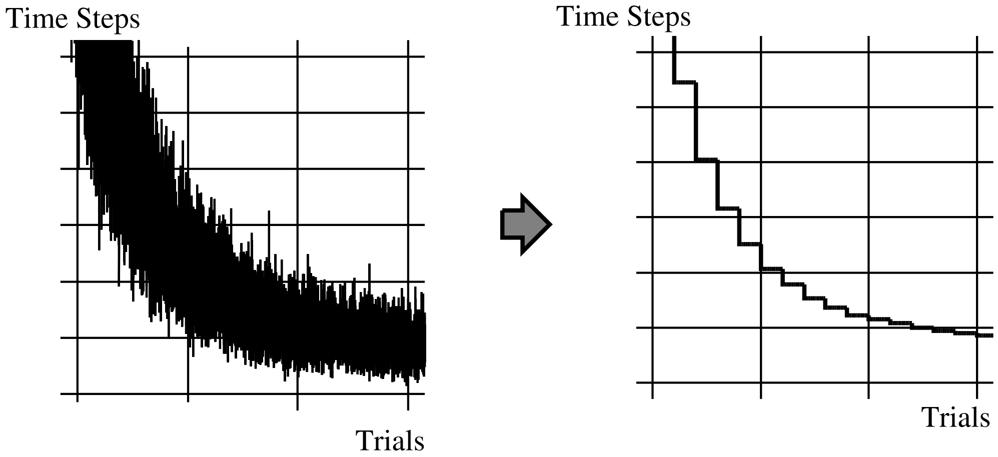


図 4.2: 時間ステップの平均化

実験における捕獲に要する時間ステップのグラフを 100 試行毎に平均化し (図 4.2参照)，最初に上記の方法で求めた収束値に達した試行を収束までに要する試行数として用いた．

まず図 4.3の単体構造アーキテクチャの結果と他の結果とを比較すると，単体構造アーキテクチャでは最初はうまく捕獲することが出来ず，短い時間ステップで捕獲できるようになるまでに多くの試行数を要するのに対し，他のモジュール化を行なったアーキテクチャでは最初から比較的短い時間ステップで捕獲できるようになることがわかる．表 4.4を見ても単体構造の Q 学習に比べ，他の三つのモジュール化によるアプローチの収束速度が向上していることがわかる．これは，問題空間をモジュールに分割することにより，一つの Q テーブルが扱う状態空間の数が減少し，Q テーブルを適切なものに更新するまでの時間が比較的短くて済むからである．特に，固定モジュールのアーキテクチャの収束速度は非常に速い．これは AMQL-GA，AMQL-LL アーキテクチャの場合，適切なモジュールを獲得するまでに時間がかかるのに対し，固定モジュールの場合はあらかじめ設計者により適切なモジュールセットが与えられているからである．しかし，表 4.4 における収束値を比較すると，設計者によって予め与えられたモジュールセットよりも，モジュールセットを自動的に獲得するアーキテクチャの方が優れたモジュールセットを獲得していることがわかる．

図 4.5と図 4.6を比較すると，AMQL-GA アーキテクチャの方が，収束するまでの間の捕獲に要する時間ステップの平均が大きいことがわかる．これは，AMQL-GA アーキテクチャの場合，各染色体の評価を始める際に毎回行動方針を初期化した状態から行なうからである．これに対し AMQL-LL アーキテクチャでは選択淘汰の際に行動方針も継承するため，モジュールセットの評価を始める際は途中から学習を行なうことになり，学

表 4.4: 収束値と収束までに要する試行数

	収束値	収束までに要する試行数
単体構造	67.44	35720
MQL	34.95	2727
AMQL-GA	32.64	19613
AMQL-LL	33.04	14320

習途中の行動方針もAMQL-GAアーキテクチャに比べて良くなる．AMQL-LLアーキテクチャの収束速度が優れているのも，あるモジュールを2回目に評価を行なう時はすでにある程度学習された状態から行なうためと考えられる．

表4.4を見ると収束値に関してはAMQL-GAアーキテクチャの方が若干優れていることがわかる．適切なモジュールセットを探索する際に，AMQL-GAアーキテクチャはあるモジュールが注目する他のエージェントに関して交叉，突然変異を行なうのに対し，AMQL-LLアーキテクチャはモジュール単位で交換，突然変異を行なう．このためAMQL-LLアーキテクチャのモジュールは注目する部分状態が途中で変更できず，より洗練されたモジュールセットを構築するのが困難なためであると考えられる．

以上の結果より，AMQL-LLアーキテクチャを用いることにより設計者があらかじめ作成した固定モジュールのアーキテクチャより収束後の性能が優れていることを示した．また収束速度に関しても，単体構造のQ学習よりも速いことを示した．

4.3　変化が小さい問題における適応性の評価

次に，問題の変化が小さく，既存の学習内容(既存のQテーブルにおけるQ値)を利用，修正した方が有利であるような問題について評価を行なう．具体的には追跡エージェントが報酬を獲得する状態(図4.7参照)を動的に変更する．問題Aにおけるゴール状態は通常の追跡ゲームにおけるものと同様である．問題Bにおけるゴール状態は追跡エージェントがX字型に逃亡エージェントを取り囲む．問題Bは問題Aのゴールにおける追跡エージェントの位置が一グリッド分だけずれた状態であり，同一の逃亡エージェントを追跡するという点で問題の変化としては小さい．ここでは比較対象として単体構造のQ学習，構造化Q学習アーキテクチャ，AMQL-GAアーキテクチャを用いる．構造化Q学習アーキテクチャは第4.2節で用いたものと同様である．問題Aと問題Bを50000試行毎に入れ換え，200000試行まで行なった．

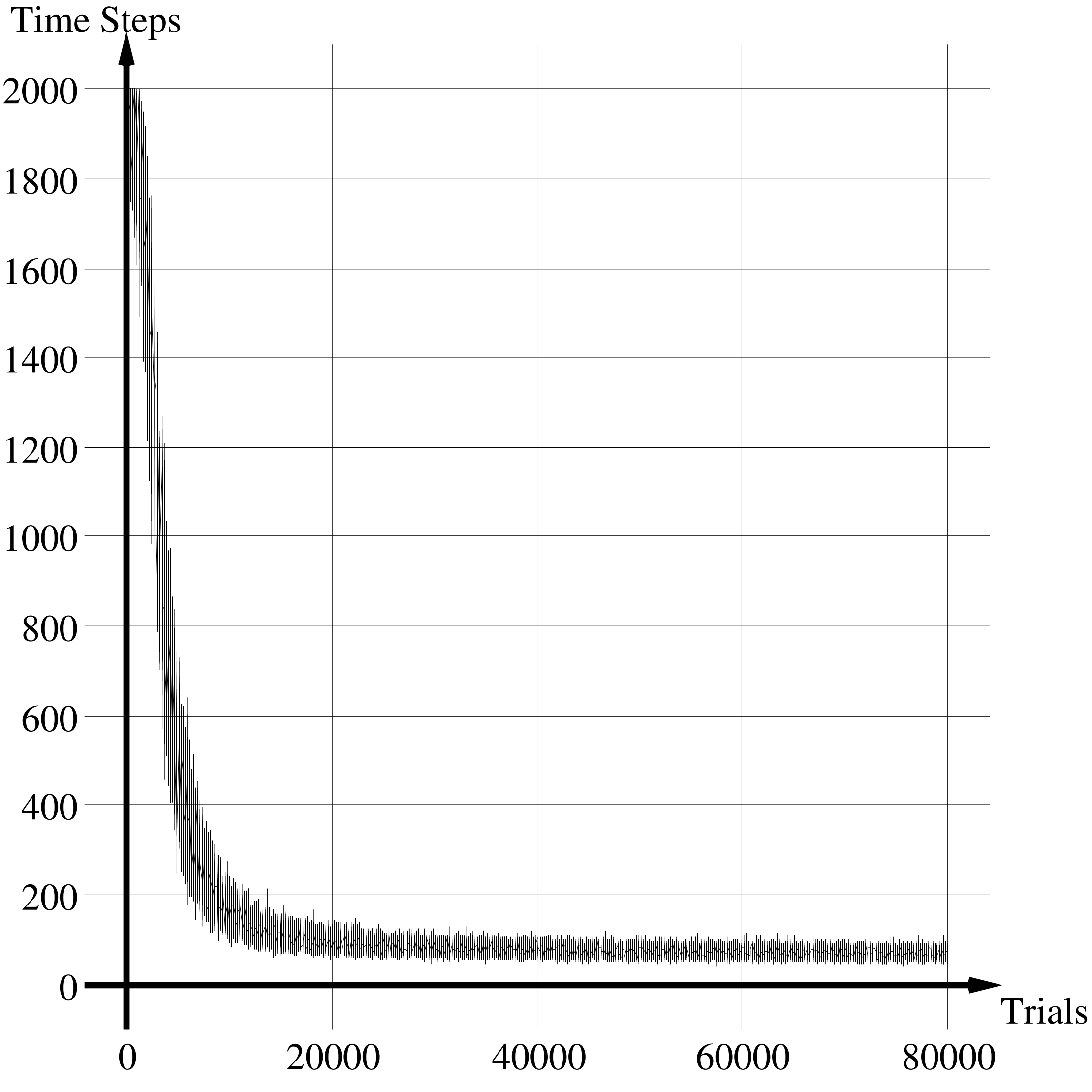


図 4.3: 捕獲までの時間ステップ (単体構造アーキテクチャ)

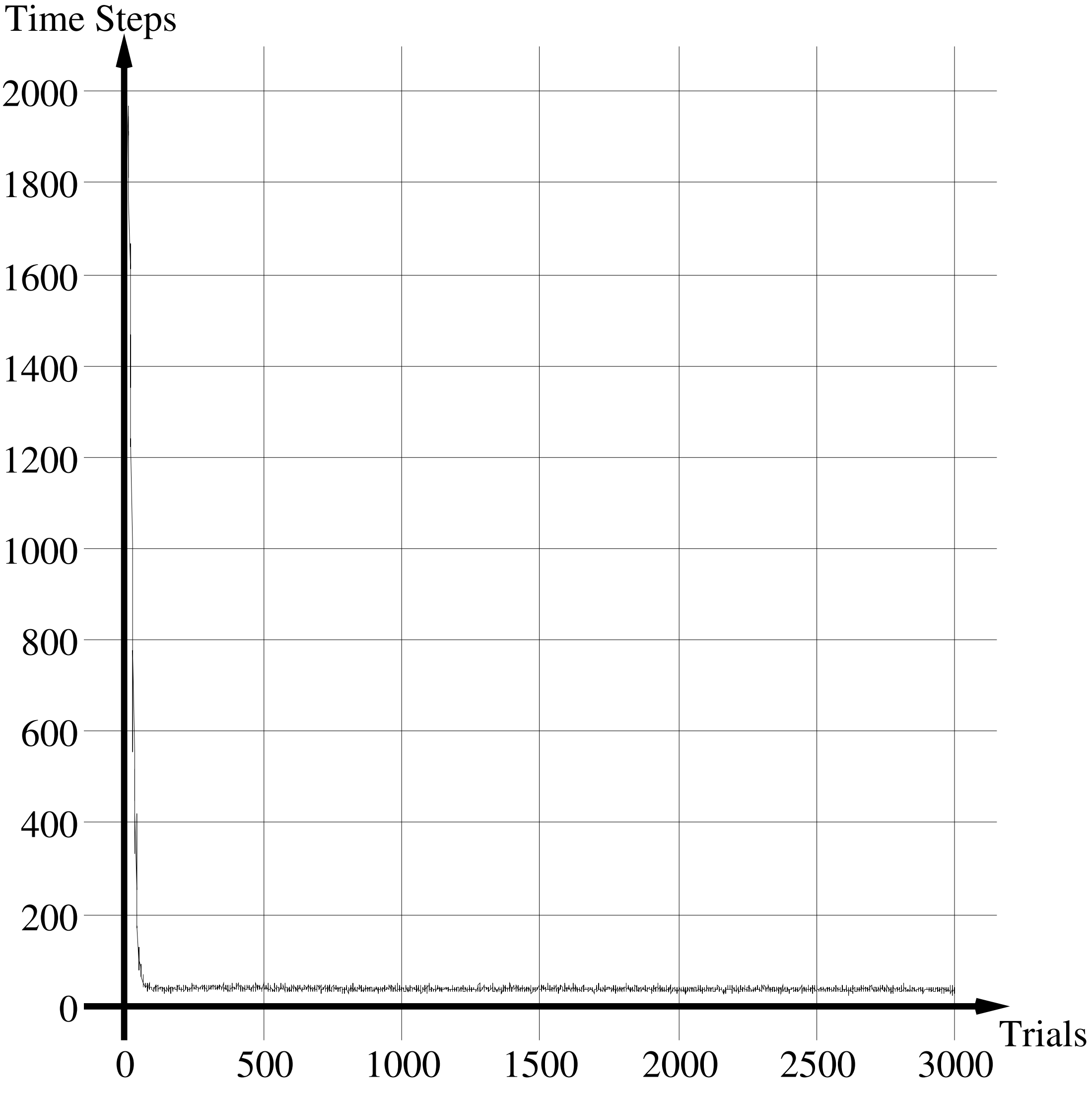


図 4.4: 捕獲までの時間ステップ (固定モジュールアーキテクチャ)

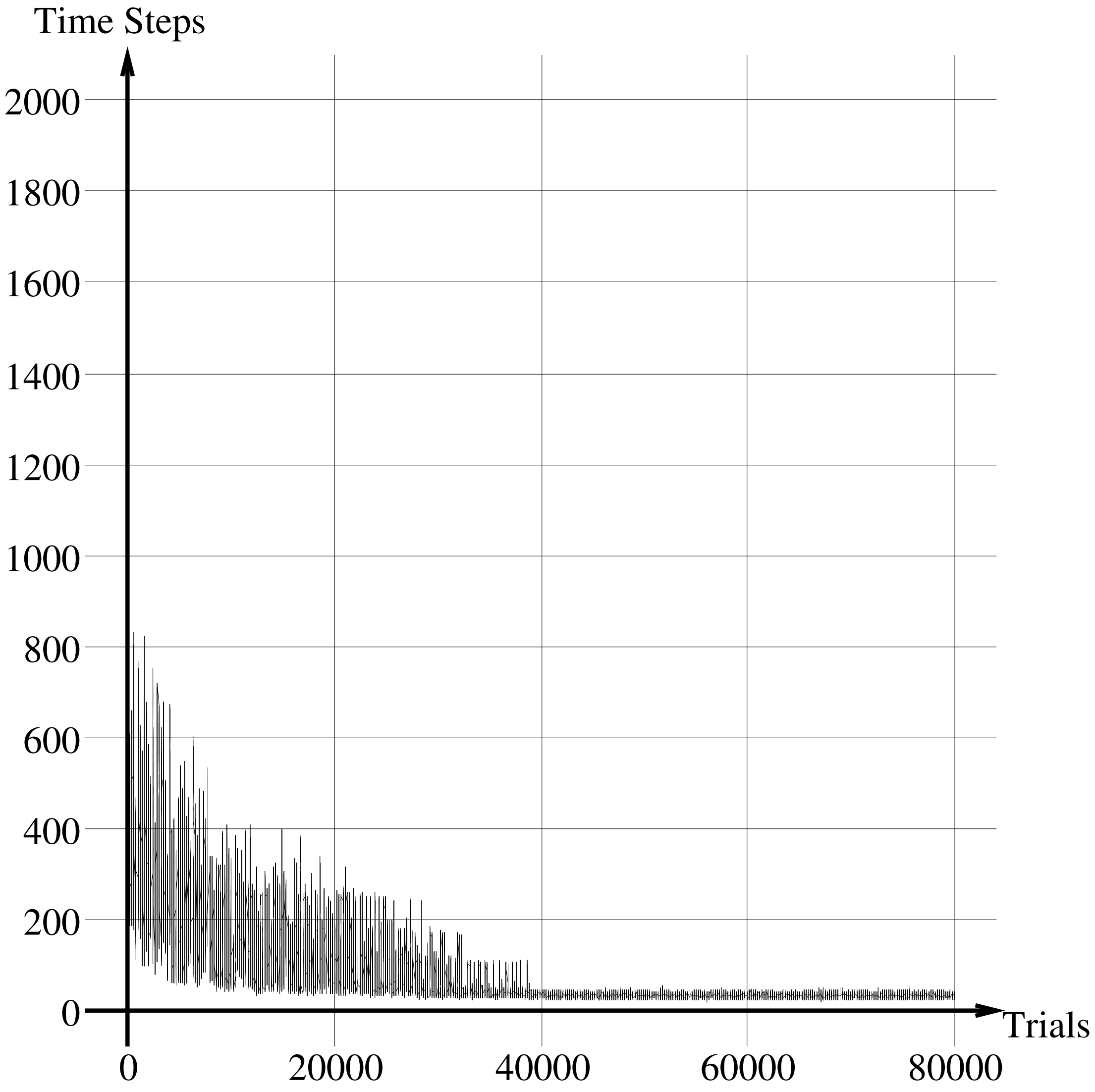


図 4.5: 捕獲までの時間ステップ (AMQL-GA アーキテクチャ)

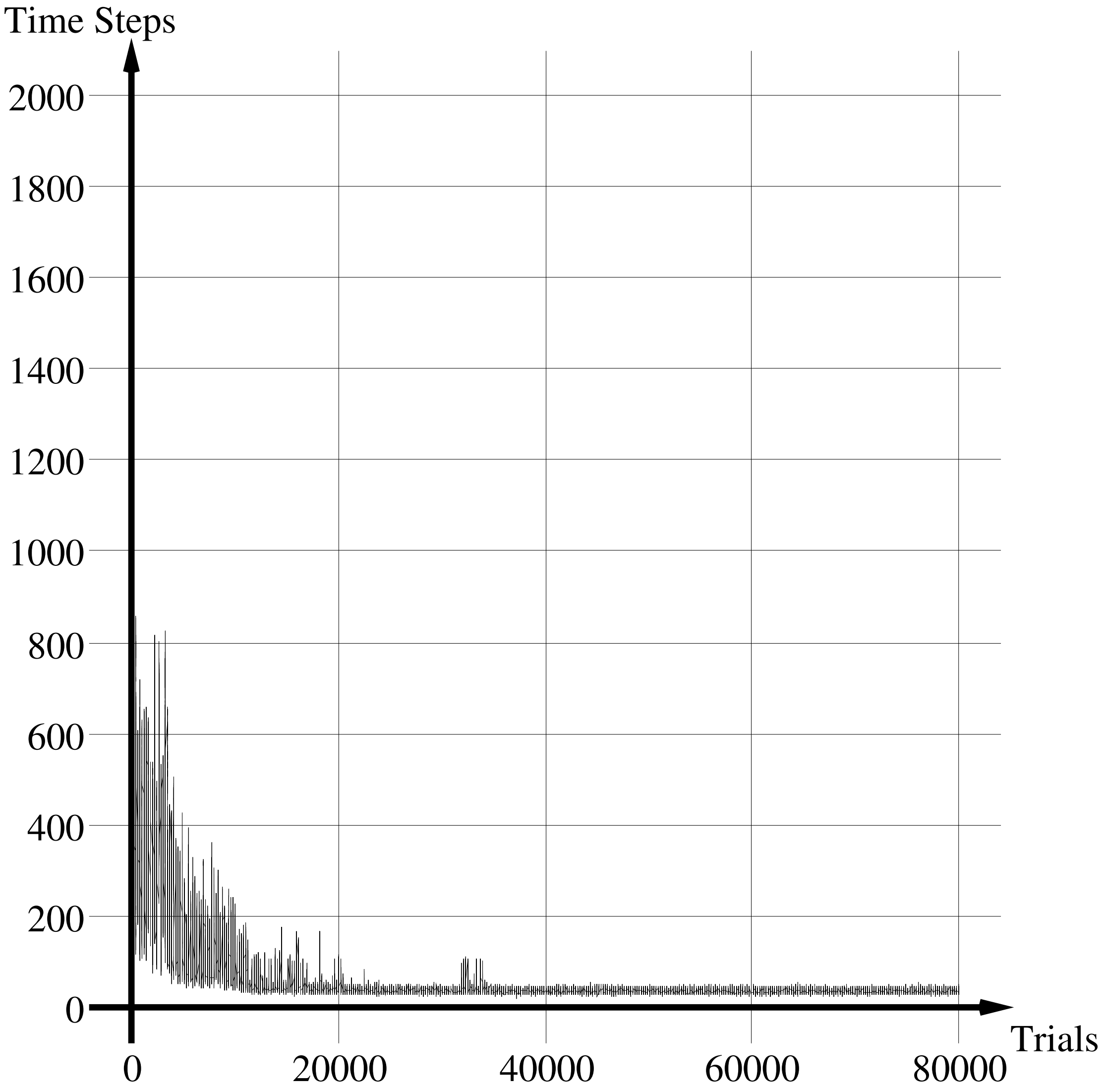


図 4.6: 捕獲までの時間ステップ (AMQL-LL アーキテクチャ)

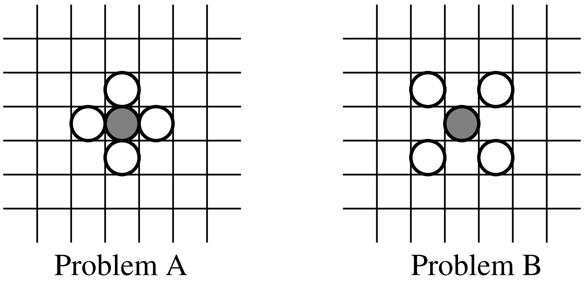


図 4.7: 問題 A，B のゴール状態 (捕獲位置の差)

4.3.1　結果と考察

図 4.8～図 4.11に，各アーキテクチャを用いた場合の各試行における捕獲までに要する時間ステップを示す．図 4.12に各学習アーキテクチャの，問題が切り替わってから収束までに要する試行数を示す．ここでは各実験の捕獲に要する時間ステップの結果を 100 試行毎に平均化し，変化の直前 100 試行を収束値とした．100 試行毎の平均が最初に収束値まで達した時点の試行を収束までに要する試行数とする．AMQL-GA 及び AMQL-LL アーキテクチャについては，モジュールが安定してから収束値まで達した時点の試行を収束値とした．各結果は 30 回の実験の平均である．また，図 4.13に AMQL-LL アーキテクチャが各問題において獲得したモジュールの継承元の割合を示す．ここで新しいモジュール (New Module) とは，各問題中に突然変異によって生成された未学習モジュールである．最初の問題 A(1～50000 試行)，問題 B(50001～100000 試行) を A1，B1，次の問題 A(100001～150000 試行)，問題 B(150001～200000 試行) をそれぞれ A2，B2 とし，A1，B1，A2 モジュールとは，それぞれ A1，B1，A2 の問題において獲得したモジュールを用いたことを示す．その他のモジュール (Other Modules) は，以前の問題においてモジュール集団中には存在したが，最終的に獲得した解としては使用されなかったモジュールを示す．

図 4.8を見ると，従来の固定 Q テーブルを持つアーキテクチャでは問題変化の直後でも短い時間ステップで捕獲できることが分かる．この実験における問題 A と問題 B では，4 つの追跡エージェントが逃亡エージェントを追跡し，取り囲むという点で非常に類似している．このため，最初の問題で逃亡エージェントを取り囲むことを学習したことが，変化後の問題における報酬獲得を速めたと考えられる．これにより，この問題における変化に対しては，既学習内容を若干修正することが有効であることが分かる．

また図 4.9を見ると，問題 A より問題 B の方が若干捕獲に要する時間ステップが多い．問題 A の場合，追跡エージェントが逃亡エージェントの四方を取り囲もうとすると，逃

亡エージェントは追跡エージェントのいる位置に移動できないため，ゴール状態付近で逃亡エージェントの動きを止めることができる．しかし，問題 B のゴールでは逃亡エージェントの動きを止めることができず，問題 B の方が若干難しくなり捕獲に余計に時間がかかるためと考えられる．

図 4.10を見ると，AMQL-GA アーキテクチャの場合，問題が変化するまで安定したモジュールセットを獲得できない場合が存在したことがわかる．これに対し，図 4.11を見ると AMQL-LL アーキテクチャは問題の変化毎にほぼ確実に安定したモジュールセットを獲得できることがわかる．

図 4.12を見ると，モジュール化を用いた機構では問題 A1 と比較してその直後の問題 B1 は，収束までにより多くの試行数を要することがわかる．これは，問題 B の方が問題 A よりも難しいことが理由として考えられる．しかし，問題 A2 と問題 B2 の結果を比較すると，AMQL-GA アーキテクチャは 1 度目の結果と同様に問題 B2 の方がより収束に時間がかかるのに対し，他の機構は問題 B2 の収束の方が速くなっている．この理由としては，固定アーキテクチャの場合には以前に問題 B について学習した結果が残っており，この既学習内容が影響することにより問題 B2 に対してより速く学習できたと考えられる．これと同様に AMQL-LL アーキテクチャも，以前に学習した内容をモジュールとして保持しており，これらを利用することでより速く学習している．図 4.13を見ると，各問題において既学習モジュールを用いている割合が，未学習モジュールよりも大きい．特に問題 B2 では，半分以上のモジュールが以前の問題で獲得したものである．AMQL-LL アーキテクチャの収束速度が構造化 Q 学習アーキテクチャよりも遅いのは，前述の基本問題と同様に適切なモジュールを獲得するまでに多くの試行数を要するためである．

以上の結果より，AMQL-LL アーキテクチャは問題の変化が小さく既学習内容の利用・修正が有効な問題変化に対して，既学習モジュールを適用することにより AMQL-GA アーキテクチャよりも効率的に収束することができることを示した．

4.4　変化が大きい問題における適応性の評価

最後に，問題の変化が大きく，以前に学習した内容が新たな問題における学習の妨げになるような問題について評価を行なう．具体的には基本的な追跡ゲームに以下の設定を加える．

- 四つの追跡エージェントと二つの逃亡エージェントが存在する
- 二つの逃亡エージェントはそれぞれ逃亡エージェント 1，逃亡エージェント 2 という識別子を持ち，お互いが区別される．

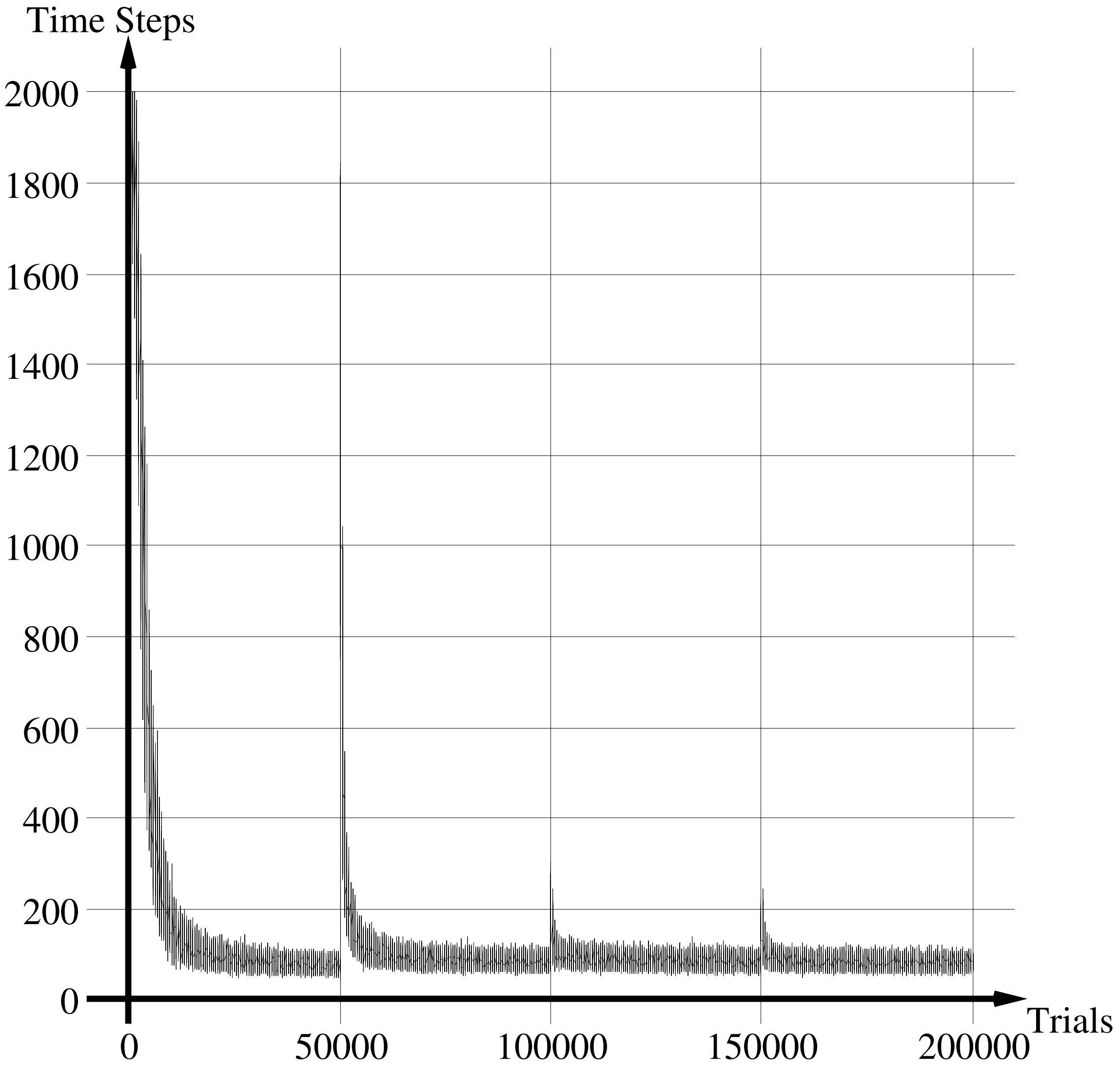


図 4.8: 問題変化が小さい場合の捕獲までの時間ステップ (単体構造アーキテクチャ)

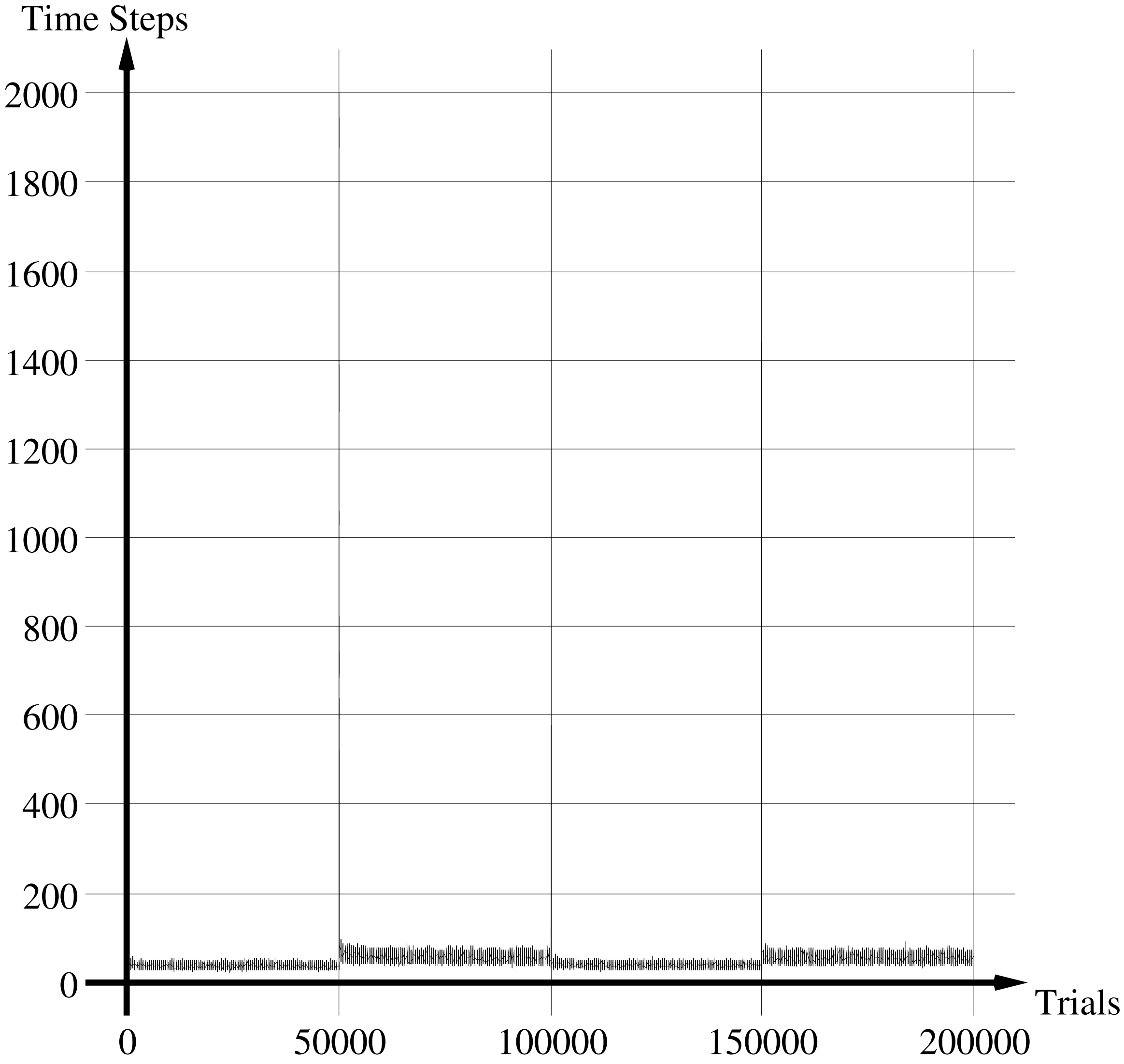


図 4.9: 問題変化が小さい場合の捕獲までの時間ステップ (固定モジュールアーキテクチャ)

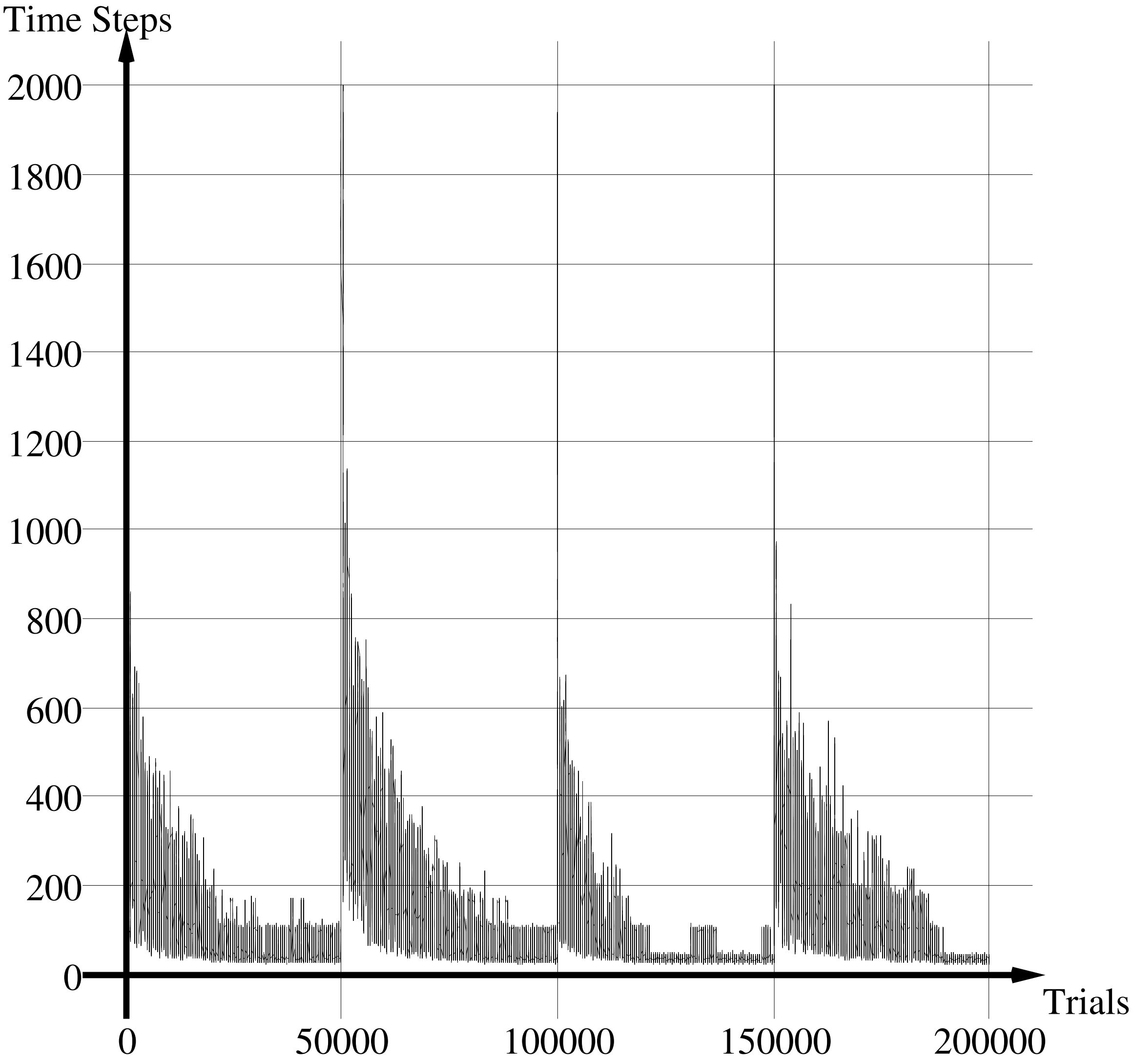


図 4.10: 問題変化が小さい場合の捕獲までの時間ステップ (AMQL-GA アーキテクチャ)

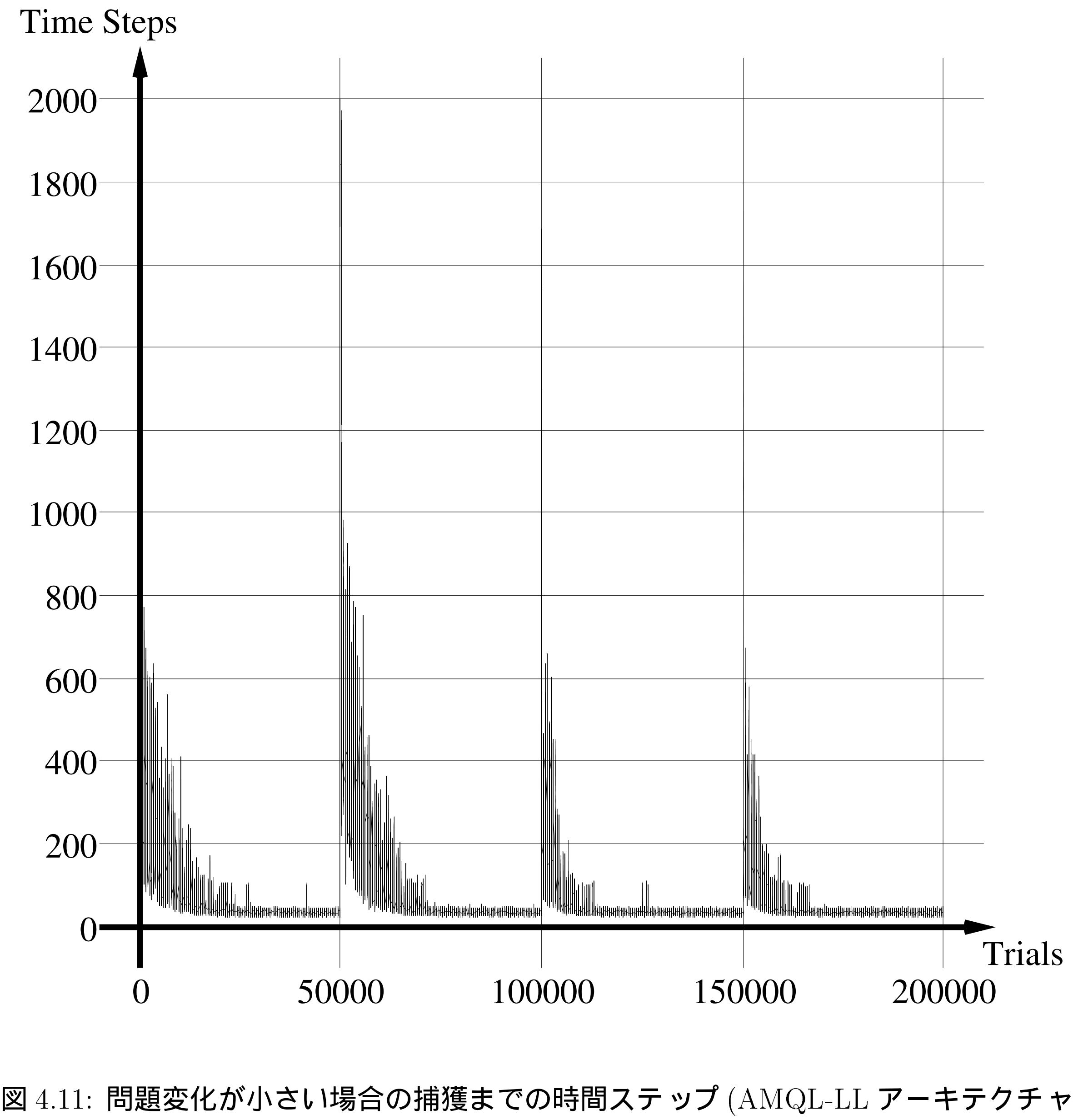


図 4.11: 問題変化が小さい場合の捕獲までの時間ステップ (AMQL-LL アーキテクチャ)

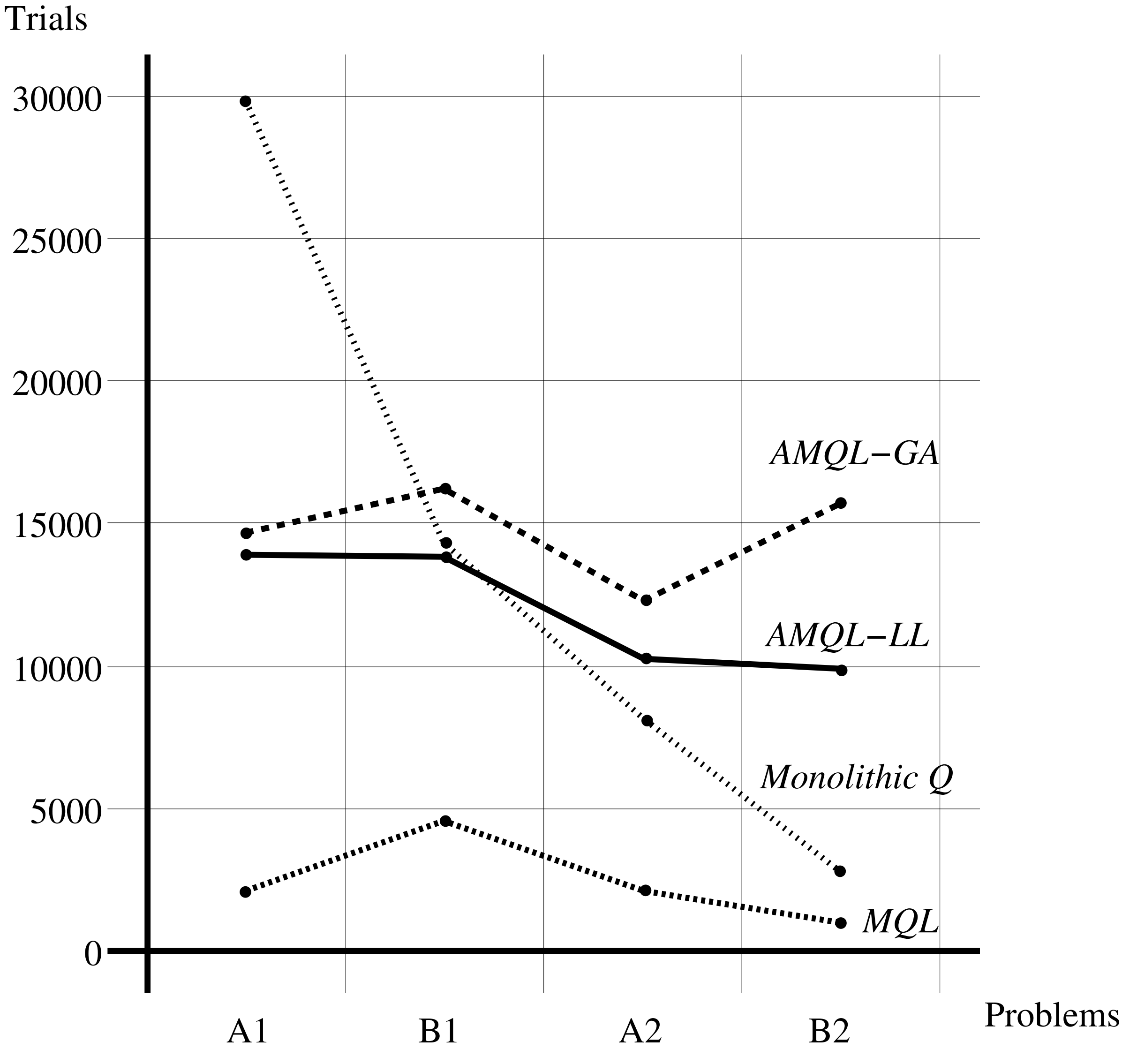


図 4.12: 各問題における収束に要する試行数

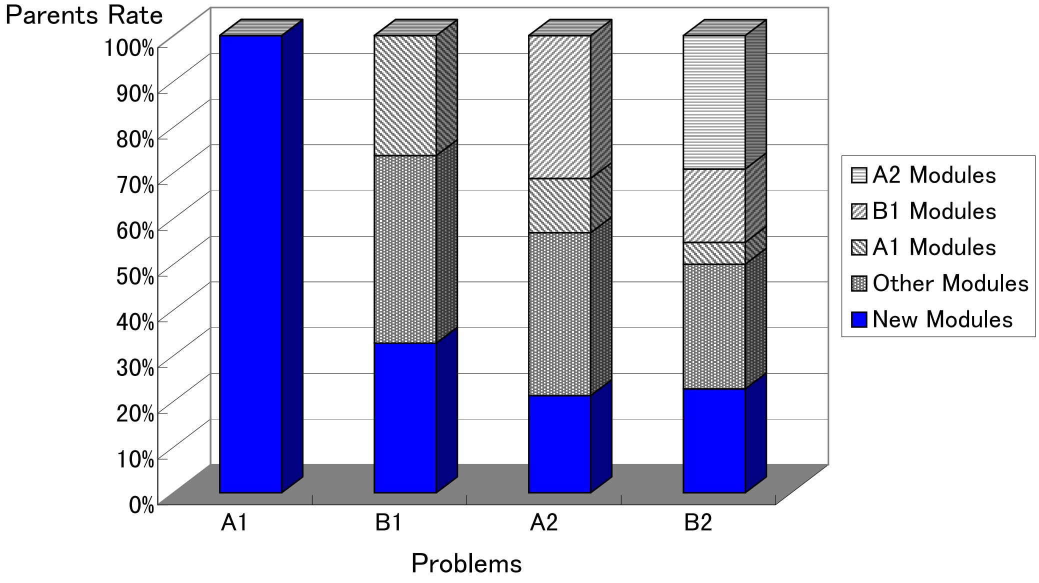
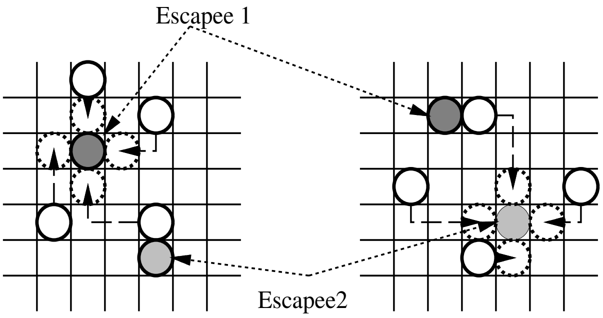


図 4.13: 問題変化が小さい場合の獲得モジュールの継承元の割合



(a) Problem A　(b) Problem B

図 4.14: 問題 A，B のゴール状態 (捕獲対象の差)

- 問題 A では逃亡エージェント 1 を四方から取り囲んで捕獲した時のみ追跡エージェントは報酬を獲得する (図 4.14(a) 参照)．逃亡エージェント 2 を捕獲しても追跡エージェントは報酬を獲得できない．
- 問題 B では逆に逃亡エージェント 2 を捕獲した時のみ追跡エージェントは報酬を獲得する (図 4.14(b) 参照)．
- 上記の二つの問題を 50000 試行毎に入れ換える．

ここでは環境におけるエージェントの数が多いため，単体構造の Q 学習における Q テーブルが大きくなりすぎ評価を行なうことが不可能である[1]．また，AMQL-GA アーキテクチャを用いた場合，問題の変化後に収束できない場合が生じたため，収束値に関する比較ができない．このため収束値に関する比較対象としては固定モジュールの構造化 Q 学習アーキテクチャを用い，問題の変化に対する収束性について評価を行なう．ここで用いる構造化 Q 学習アーキテクチャは，三つのモジュールを持ち，各モジュールは二つの逃亡エージェントと互いに異なる追跡エージェントに注目するものを用いた．

4.4.1 結果と考察

図 4.15 ~ 図 4.17に，各アーキテクチャを用いた場合の，各試行における捕獲までに要する時間ステップを示す．図 4.18には，AMQL-LL アーキテクチャと構造化 Q 学習アーキテクチャについて各問題毎に最後の 100 試行における捕獲に要する時間ステップの平均を求め，その推移を表す．また，図 4.19に AMQL-LL アーキテクチャが各問題において獲得したモジュールの継承元の割合を示す．

---

[1] 扱う状態数は $(5^2+1)^5 = 11881376$ となる．

図 4.15を見ると，固定モジュールアーキテクチャの場合には 100000 試行までは問題が変化しない場合とほぼ同じ時間ステップで捕獲することができるようになるが，100000 試行以降になるとそれ以前よりも捕獲により多くの時間ステップを要することがわかる．さらに，問題が変化する毎に収束に，より多くの試行数を要する．この問題では問題が変化する毎に別の逃亡エージェントを追跡する必要があるが，一つの Q テーブル内で逃亡エージェント 1，逃亡エージェント 2 に関する学習を行なうため，互いに干渉してしまう結果，適切な行動方針を獲得できない．また，一方の問題に対する過学習がもう一方の学習の妨げになっている．実際に個々の実験の結果を調べると，問題 A もしくは問題 B のどちらか一方のみの収束値が比較的良く，もう一方の問題に対しては非常に悪い結果である場合が多く見られた．

一方，図 4.17を見ると，AMQL-LL アーキテクチャの場合は問題が動的に変化しても最終的に適切に捕獲できるようになることがわかる．図 4.18における比較結果を見ると，構造化 Q 学習アーキテクチャでは変化する度に捕獲により多くの時間ステップを要するようになるのに対し，AMQL-LL アーキテクチャは逆により短い時間ステップで捕獲できるようになっていることがわかる．これは AMQL-LL アーキテクチャは突然変異によって内部に新しく初期化されたモジュールを保持し，過学習の影響をうけにくいためである．図 4.19を問題変化が小さい状況の場合の図 4.13と比べて見ると，未学習モジュールが各問題毎に獲得したモジュールの大きな割合を占めていることが分かる．これに対し，直前の問題で獲得したモジュールを利用する率は大きくない．これにより，AMQL-LL アーキテクチャが現在の問題に対して都合の良いモジュールを選択的に使用していることがわかる．

図 4.17の各問題における収束までの様子を見ると，収束速度に関して問題が変化する毎に若干早くなっていることが分かる．この二つの問題の場合，目標とする逃亡エージェントは異なるが四つの追跡エージェントが逃亡エージェントを取り囲むという共通点がある．このため，追跡エージェント同士がお互いに集まるという行動に関して，これまでに学習した内容が部分的に利用されているためと考えられる．さらに，図 4.19の問題 B2 の結果を見ると，直前の問題 A2 で獲得したモジュールよりも，以前に直面した同じ問題である問題 B1 で獲得したモジュールをより多く使用していることからも，過去の学習モジュールを選択的に利用していることが分かる．

図 4.16を見ると，AMQL-GA アーキテクチャは問題の変化に対して適切に収束出来ない場合が多いことがわかる．この理由として，4.3節における実験ではある追跡エージェントが注目すべき他のエージェントの数が 4 つであるのに対しこの実験では 5 つになり，モジュールセットの探索空間が大きくなることにより収束により多くの試行数を必要と

するようになったことが考えられる．また，最初の問題 A1 では多くの実験で収束しているのに対し，変化後の問題ではうまく収束しないことがわかる．この実験において，各問題において報酬を得られない逃亡エージェントは，注目しても無駄であり，むしろ注目することで無駄に Q テーブルのサイズを大きくするため学習の妨げになる．問題 A では逃亡エージェント 1 に，問題 B では逃亡エージェント 2 に注目する必要があり，問題が変化する毎に注目する部分入力の異なる染色体が選択される必要がある．しかし，AMQL-GA アーキテクチャで用いる遺伝的アルゴリズムには，選択淘汰を繰り返すことで集団が似たような個体に偏り，多様性を失うという特徴がある．このため AMQL-GA アーキテクチャでは，染色体が最初の問題に特化したものに偏ることで，変化後の問題に対してなモジュールセットを獲得しにくくなると考えられる．

以上の結果より，従来の固定モジュールのアーキテクチャでは動的に変化する問題に対して適切な行動方針を学習できないのに対し，AQML-LL アーキテクチャは新たに生成された未学習モジュールを利用することにより過学習に影響されることなく適切な行動方針を学習することができることを示した．さらに，未学習のモジュールを用いるだけでなく，過去に学習した内容を選択的に適用することにより，効率的に学習を進められることを示した．

4.5 まとめ

本章では追跡ゲームにおいて，追跡エージェントに AMQL-LL アーキテクチャを実装し評価を行なった．これまでに行なった実験をまとめると以下のようになる．

- AMQL-LL アーキテクチャは，モジュール獲得までに時間がかかるが最終的には適切なモジュールを獲得できた．また，学習が収束するまでに必要な試行数の面では，ラマルク的に学習内容を再利用するために AMQL-GA アーキテクチャに比べて少ない試行数で収束することができた．
- 問題の変化が小さく，既学習内容の若干の変更で済む問題では，AMQL-LL アーキテクチャは既学習内容を保持し，それに変更を加えることにより効率良く収束することができた．
- 問題の変化が大きく，既学習内容が新たな学習の妨げになってしまう問題では，新たなモジュールなどを適用することにより適切な行動方針を学習することができた．

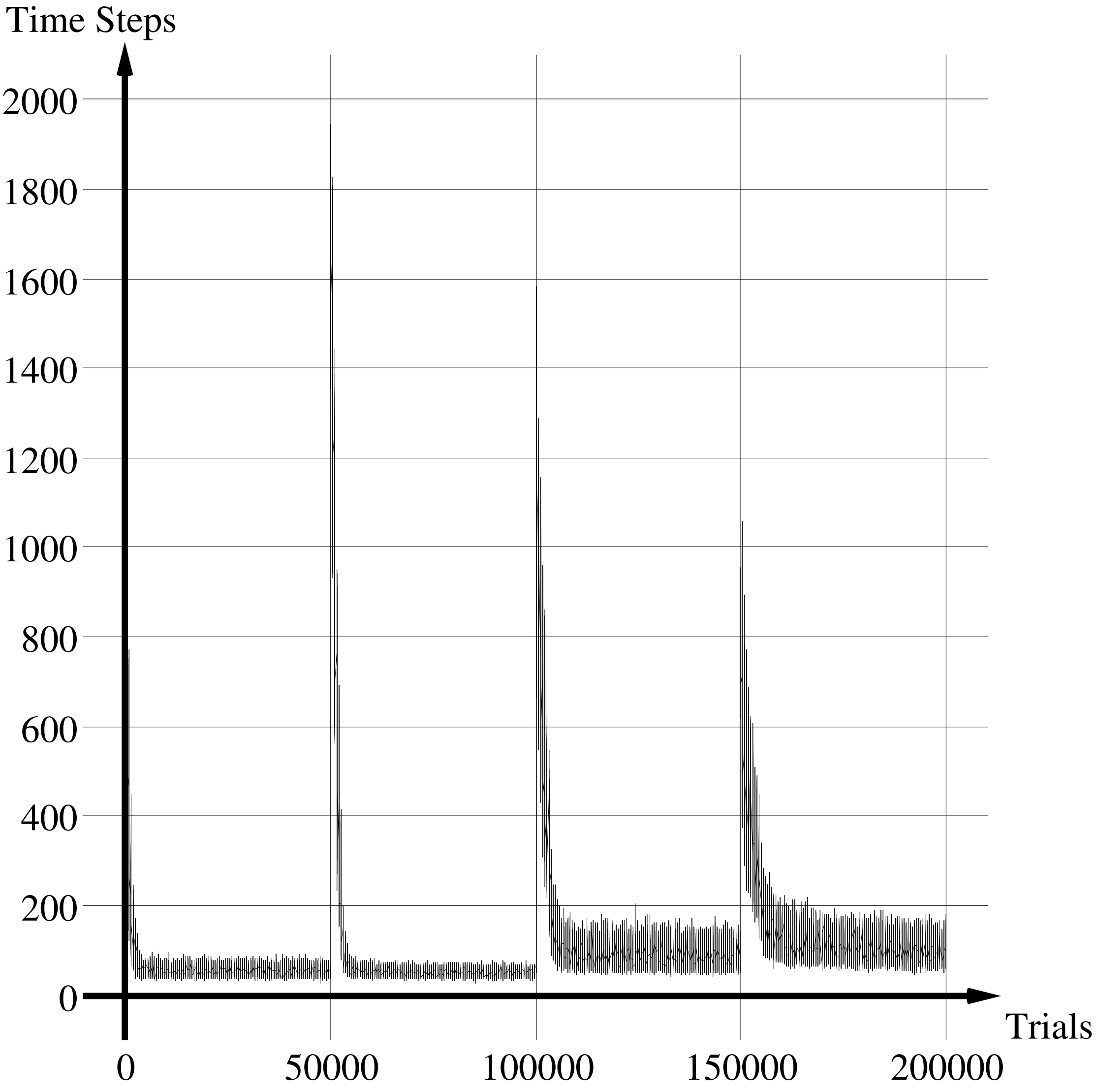


図 4.15: 問題変化が大きい場合の捕獲までの時間ステップ (固定モジュールアーキテクチャ)

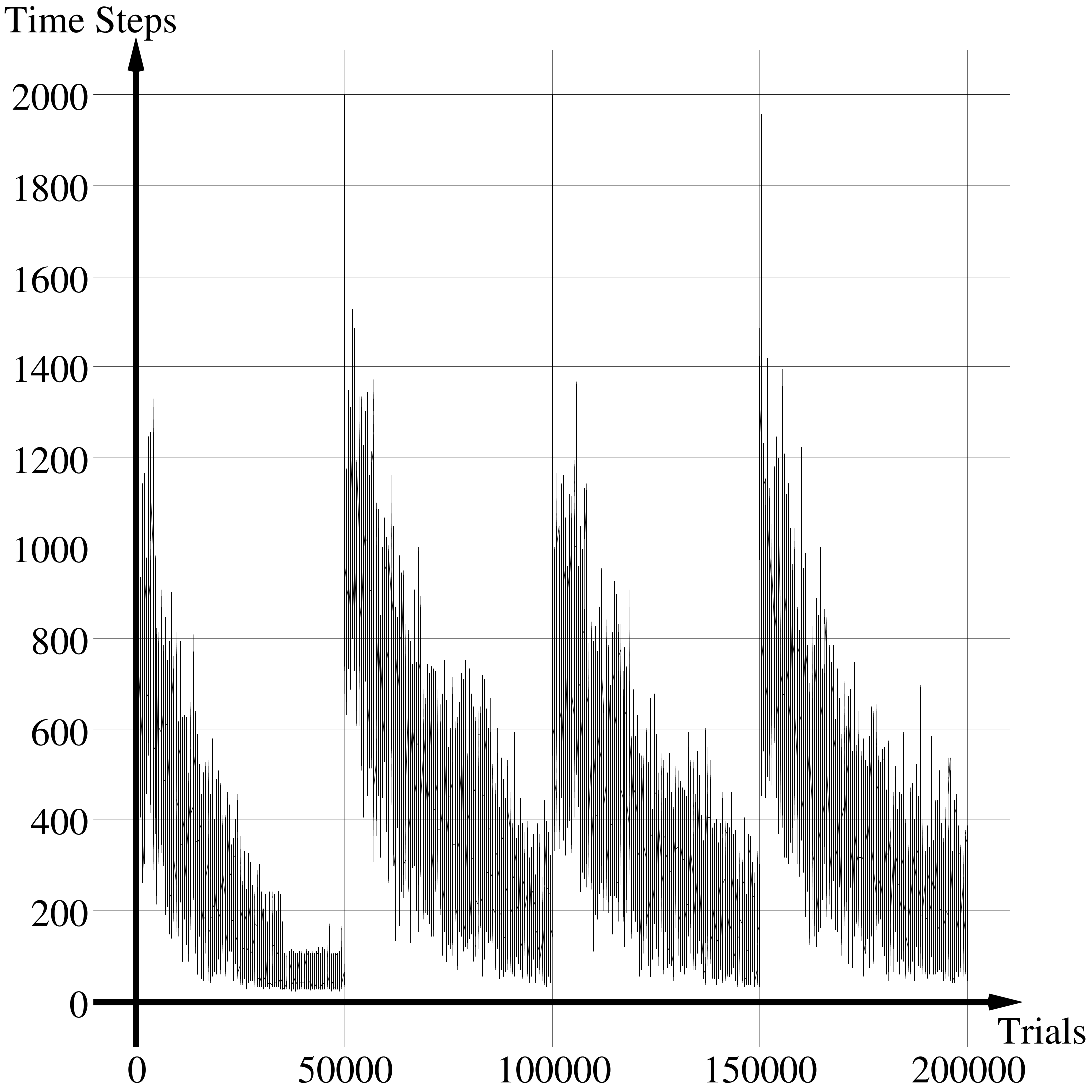


図 4.16: 問題変化が大きい場合の捕獲までの時間ステップ (AMQL-GA アーキテクチャ)

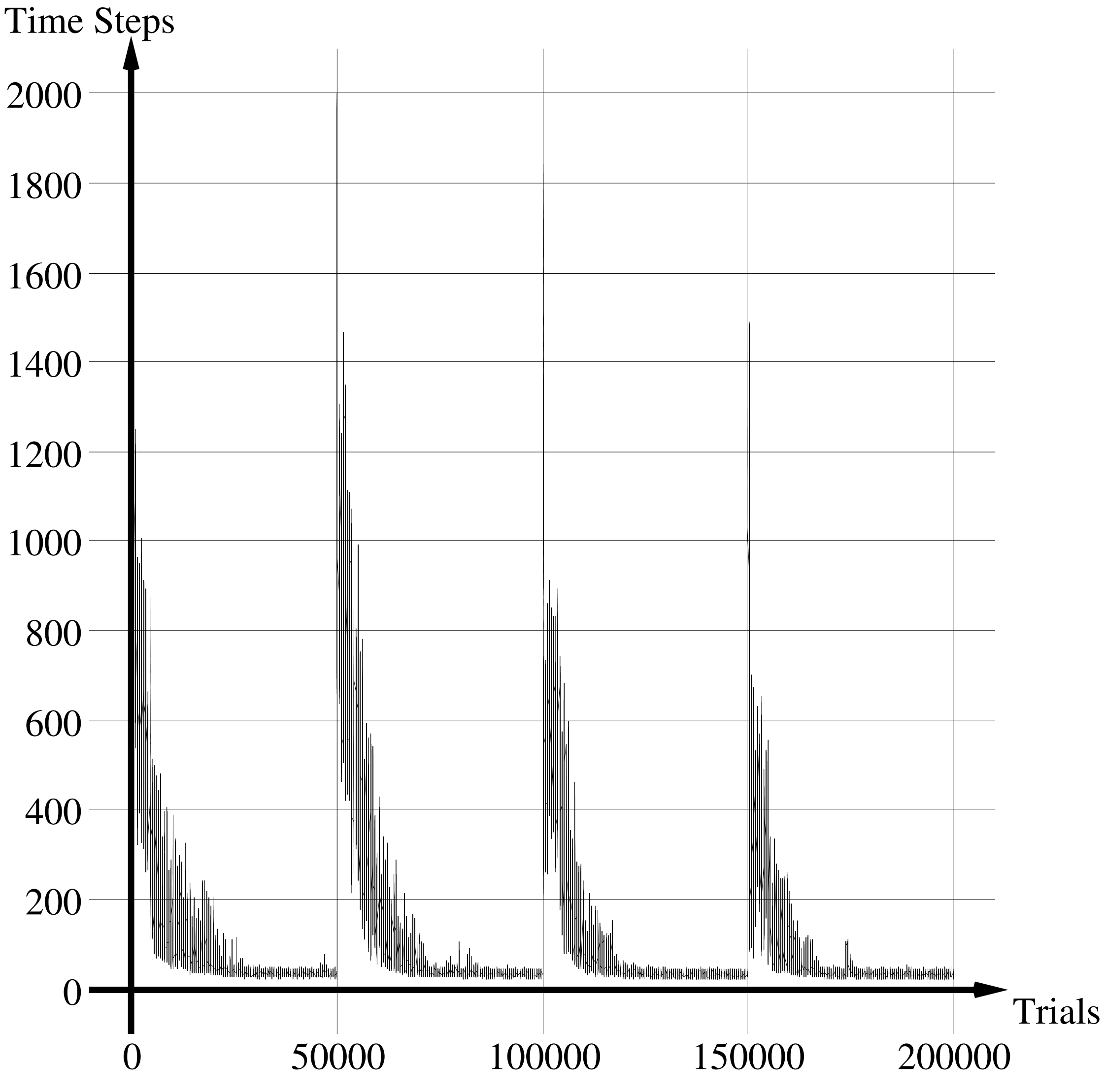


図 4.17: 問題変化が大きい場合の捕獲までの時間ステップ (AMQL-LL アーキテクチャ)

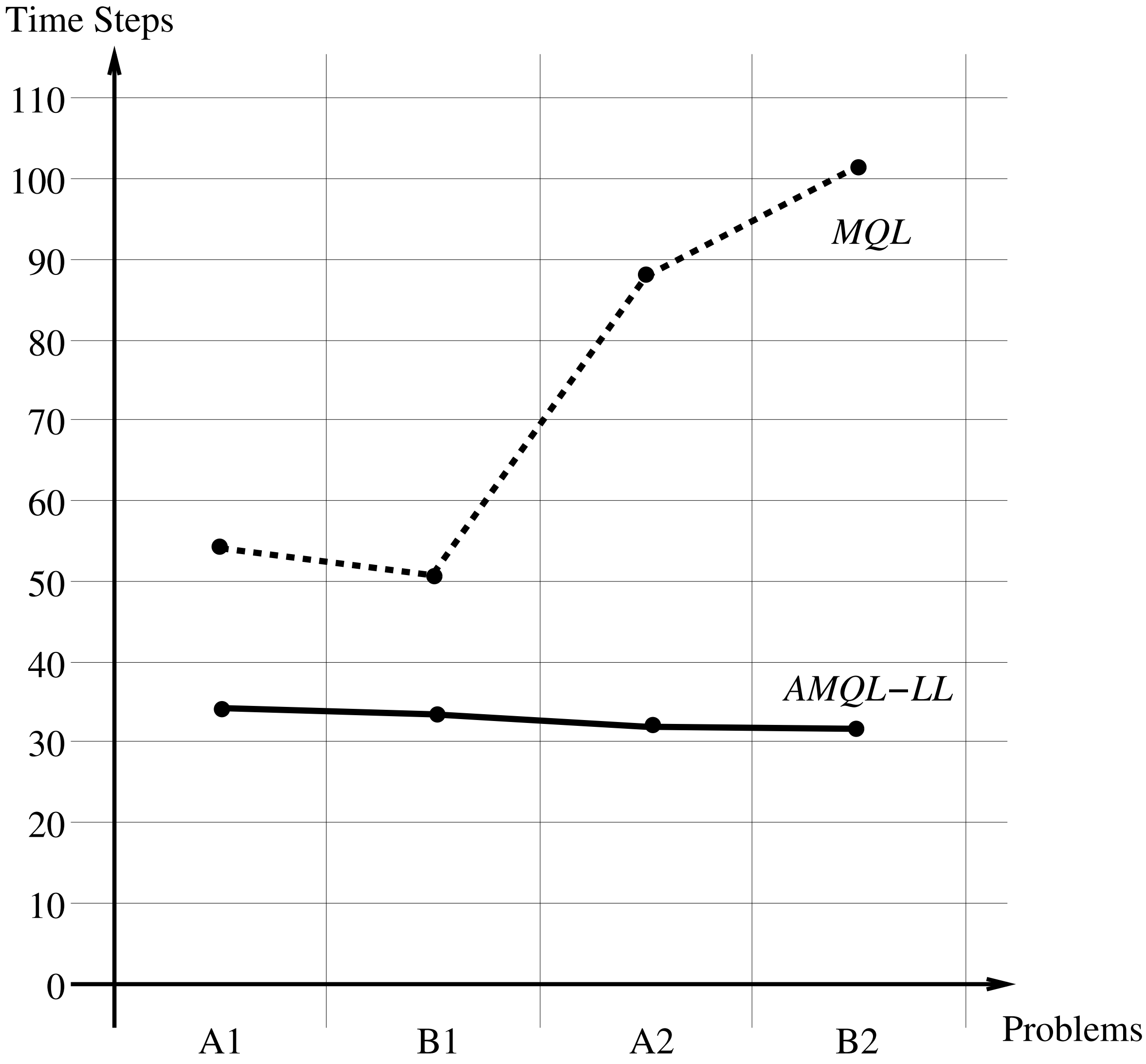


図 4.18: 各問題における平均捕獲時間ステップ

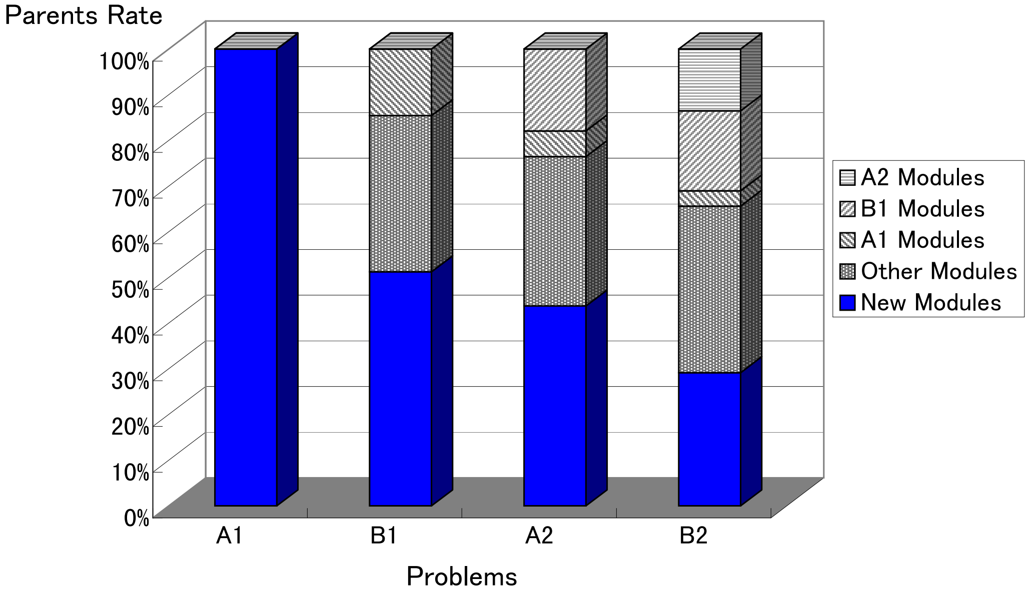


図 4.19: 問題変化が大きい場合の獲得モジュールの継承元の割合

第 5 章

議論と今後の課題

5.1 AMQL-GA アーキテクチャとの比較

AMQL-LL アーキテクチャは，松林が提案した AMQL-GA アーキテクチャ[6] を基に構築したものである．ここでは両アーキテクチャの比較を行う．

AMQL-LL アーキテクチャと AMQL-GA アーキテクチャは，共に構造化 Q 学習 [40] におけるモジュールセットを，環境との相互作用によって自動的に獲得する機構である．エージェント内にモジュールセットに関する情報を保持する点や，選択淘汰のメカニズムを利用した現在の問題に対する適切なモジュールセットの獲得など，アルゴリズムの点では多くの共通点がある．基本的な相違点は AMQL-GA アーキテクチャでは選択淘汰の際にモジュール構成の情報のみを継承するのに対し，AMQL-LL アーキテクチャでは学習により獲得した行動方針も継承することである．しかし，その継承内容の違いから，様々な特徴の違いが見られる．

- 学習内容の継承

  AMQL-LL アーキテクチャは選択淘汰の際に学習内容を継承する．つまりエージェントは，これまでに学習した内容をモジュール単位で保持することになる．このため，問題が変化しても以前に学習した問題と部分的な共通点や類似点がある場合は過去の学習内容を利用することにより効率良く学習することができる．これに対し，AMQL-GA アーキテクチャは学習内容を保持しないため，変化後の問題が以前に学習した内容と類似している場合にも最初から学習を行なう必要が生ずる．Q 学習は特に学習の初期において強化信号を得るまでに非常に時間がかかるという特徴がある．このため Lin は最初に学習の初期に数例の模範解答を人間が与えることにより学習速度が向上し，局所解の脱出をはかることができることを示している [27]．AMQL-LL アーキテクチャの場合は，過去に学習した問題と類似した問題に直面したときには，既学習内容を適用することにより強化信号の獲得を速めることが期待できる．しかも，外部から手本を与えるのではなく，エージェントが自動的に過去の経験を利用するため，動的に変化する問題に対して非常に有効であるといえる．

- 交叉・突然変異などの操作の粒度

  AMQL-GA アーキテクチャでは，各モジュールが注目する環境の部分要素をビット列で表し，一つ一つの部分要素の単位で交叉，突然変異を行なう．これに対し，AQML-LL アーキテクチャはモジュール単位で交換，突然変異を行なう．このため AMQL-LL アーキテクチャでは，一度生成されたモジュールの注目する部分要素が

変化することはない．このように遺伝的操作の粒度が粗いため，あるモジュールが最初の時点で報酬獲得に関係のない要素を選択した場合，無駄な状態空間について学習しなければならなくなる．このため AMQL-LL アーキテクチャの方がより洗練されたモジュールセットを獲得するのが困難であるといえる．実際に追跡ゲームの実験において，最終的に収束した時点における捕獲に要する時間ステップは AMQL-GA アーキテクチャの方が若干優れていた．エージェントが学習する問題がより困難になる場合，AMQL-LL アーキテクチャでもより洗練されたモジュールを獲得できる機構が必要となると考えられる．

- 記憶情報の量

  本論文では Q 関数を Q テーブルを用いて表現している．AMQL-GA アーキテクチャでは，内部にモジュールの構成情報のみを保持するのに対し，AMQL-LL アーキテクチャでは，各モジュール全てが学習内容を Q テーブルの形で保持している．このため学習内容の保持のために非常に多くの記憶容量を要する．例えば本論文で用いた追跡ゲームにおいてエージェントがモジュールを 40 個保持し，各モジュールが三つの他のエージェントの取る状態に関する Q テーブルを作る場合，全モジュールが保持すべき状態数は $(25+1)^3 \times 40 = 703040$ となる．これは他の注目するエージェントが四つの時の単体構造の Q テーブル $((25+1)^4 = 456976)$ よりも大きい．このためモジュールをあまり重複しないようにしたり，Q 関数にニューラルネットワークを用いるなどの工夫が必要になると考えられる．

5.2 強化学習と進化的機構との融合について

AMQL-LL アーキテクチャは，行動方針の学習に強化学習を使用し，現在の問題に適切なモジュールセットを獲得する手段として進化的手法を用いている．実際に強化学習やニューラルネットワークなど局所探索的な学習方法と，遺伝的アルゴリズムなどの広域探索手法とを融合した研究は多くの分野で行なわれている．ここでは，強化学習などの個体単位での学習方法と進化的手法を組み合わせた研究についていくつかの例を挙げ，AMQL-LL アーキテクチャとの比較を行なう．

強化学習と遺伝的アルゴリズムを組み合わせた例として分類子システムが挙げられる．分類子システムにおける行動選択機構は，一種のプロダクションシステムである．各ルールには強さが割り当てられており，条件部が満足された実行可能ルールの間の競合は強さを参考に解消される．この強さは，ある意味で Q 値のようなものであり，Q 学習の場合のように強さに比例した確率で実行ルールを選択することも考えられる．ある程度実

行が進み，ルールの強さに十分な変更がなされた時点で，強さを適応度とみなして遺伝的アルゴリズムを適用する．これにより，無駄なルールは淘汰され，有効なルールを組み合わせて新たなルールの生成が行なわれる．このような枠組はミシガンアプローチと呼ばれ CS-1 として発表された [20]．このように各ルールで淘汰するのと対照をなすものとして，ルールセットを個体としてルールセット同士の交叉を繰り返すことにより適切なルールセットを獲得するピッツアプローチなどがある [33]．AMQL-LL アーキテクチャにおけるモジュールはルールセットと見なす事ができ，複数のルールセットの中から現在の問題に適切なものを選択淘汰によって獲得するという点ではピッツアプローチと似ている．ただし，ピッツアプローチは報酬獲得のための行動方針は速く得られるものの最適方針が得られる保証はない．Q 学習はこれとは対照的に，最適方針の獲得が保証されるが，最適方針を獲得するまでに膨大な時間ステップを要するという欠点があるが，AMQL-LL アーキテクチャでは，Whitehead が提案した構造化の手法により学習時間の問題を解決しているため，より速く，適切な行動方針を獲得することが期待できる．

強化学習を用いた機構とは異なるが，ニューラルネットワークと遺伝的アルゴリズムを組み合わせることで，学習性能の向上やニューラルネットワークの構造の自動獲得を試みている研究が幾つか存在する．これらは遺伝的アルゴリズムを用いる目的や，バックプロパゲーションなどの学習を同時に用いるかなどの違いにより幾つかに分類することができる．

1. 遺伝的アルゴリズムのみを用いたニューラルネットワークの学習 [41]．
2. 遺伝的アルゴリズムの広域探索の能力とニューラルネットワークの局所探索の能力を利用した学習 [23]．
3. 遺伝的アルゴリズムを用いたニューラルネットワークの構造獲得 [12, 18, 29]．
4. 遺伝的アルゴリズムを用いたニューラルネットワークの構造と重みの獲得 [24]．

1. と 2. は両者とも遺伝的アルゴリズムを用いてウェイトトレーニングを行なうものである．ただし 1. は遺伝的アルゴリズムのみでトレーニングを行なうのに対し，2. はバックプロパゲーションなどの局所探索的学習機構を組み合わせたものである．また，3. は遺伝的アルゴリズムをニューラルネットワークのトポロジー構築に用いたものである．また，4. の例としては北野の提案したニューロジェネティックラーニング (以下 NGL) が挙げられる．NGL は遺伝的アルゴリズムを用いてニューラルネットワークのトポロジーと初期値として使用されるウェイトの両方を獲得する．AMQL-GA アーキテクチャや AMQL-LL アーキテクチャにおいて，それぞれ注目する部分入力の異なるモジュール

の組み合わせを決定する過程は上記の 3. における初期構造の獲得と共通点があるといえる．ただし，構造のみを進化的機構で獲得し，毎回初期状態から学習する点では 3. はむしろ AMQL-GA アーキテクチャと似ている．さらに，4. では構造のみならず適切な学習の初期状態を選択的に獲得するという点で AMQL-LL アーキテクチャと共通している．ただし，上記の 1. ~ 4. の手法は全て集団に対する学習であるため，シミュレーションでしか用いることができない．これに対し，AMQL-LL アーキテクチャは個体に対する学習手法であるためオンラインで用いることが可能である．

また，ニューラルネットワークにおけるウェイトはエージェントの行動方針と言える．つまり，上記の 1.，2.，4. は遺伝的アルゴリズムを用いて行動方針を学習している．これに対し，AMQL-LL アーキテクチャでは行動方針は Q 学習でのみ学習し，モジュールセット獲得は選択メカニズムを用いる，というように遺伝的アルゴリズムは行動方針の学習とは異なる次元で用いられている．本研究では Q 関数を構築する際に Q テーブルを用いたが，ニューラルネットワークを用いて構築する方法も存在するため，AMQL-LL アーキテクチャと北野の NGL などと融合することにより構造化の獲得と行動方針をより効率的に獲得する方法が考えられる．

5.3 他の構造化学習手法との比較

第 2.1.3節において述べたように，強化学習では問題を分割し，構造化を用いる手法が幾つか提案されている．これらの多くは，ある時点では一つのモジュールしか用いない．これに対し，AMQL-LL アーキテクチャでは，一つのモジュールは部分的な問題に注目して学習を行ない，エージェントは複数のモジュールを同時に用いることにより問題解決を行なう．このため，ある特定のモジュールについての評価を行なうことは困難であるが，ある問題において学習したモジュールを別の問題で用いることによる問題変化に対する柔軟性が期待できる．

構造化学習では現在の状態に対してどのモジュールを適用すべきかを学習する必要がある．Lin の研究では，ロボットはタスク毎にモジュールを用意して Q 学習を用いて学習し，さらにその上位の学習としてある状態においてどのモジュールを用いるべきかを強化学習を基に学習する [27]．しかし Lin の機構では，まず各モジュールが部分問題について学習を行ない，十分学習した時点で上位のモジュール選択の学習を行なう．つまり適切なモジュール選択を学習するには，モジュールの学習内容が適切であることを仮定しているこれに対し，AMQL-LL アーキテクチャはモジュールが学習を行ないながら適切なモジュールを選択するという点で異なる．

また，ニューラルネットワークによるパターン認識の分野において，Jacobs らはモジュラーニューラルネットワークを提案している [21]．これは，パターンの種類と位置を識別するという二つのタスクを学習する際に，複数のモジュールを持つネットワークを用いて，各タスクに適切なモジュールを自動的に選択し，同時に学習を行なうものである．個々のモジュールの学習と，問題に対して適切なモジュールの選択を同時に，かつ自動的に行なうという点において，AMQL-LL アーキテクチャとの共通性が見出せる．しかし，Jacobs らの機構において適切なモジュールを選択するためには，バックプロパゲーションによる教師を必要とする．さらに，AMQL-LL アーキテクチャで用いたような報酬獲得に遅延を伴う問題では用いることができない．

5.4 今後の課題

ここでは AMQL-LL アーキテクチャにおける今後の課題について述べる．

5.4.1 パラメータについて

AMQL-LL アーキテクチャでは幾つかのパラメータを設定する必要がある．ここでは AMQL-LL アーキテクチャの評価の際に用いられたパラメータについて，その特徴ついて述べ，また改良すべき点などについて触れる．

最低試行回数 T

Tは，モジュールセットの適応度とモジュール獲得までに要する試行回数に影響する．本論文における実験ではエージェントの初期位置や逃亡エージェントの行動はランダムに決定されるため，追跡エージェントが捕獲に要する時間ステップは試行毎に異なり，モジュールセットの正確な評価を行なうのが困難である．これに対しTを大きくすれば，試行毎の揺らぎの影響が平均化され，より正確なモジュールセットの適応度を求めることができる．ただし，Tを大きくしすぎると，一つのモジュールセットを評価するのに必要以上の試行数を要することになり適切なモジュールを獲得するまでに時間がかかる．また，Tを小さくすると，モジュールセットの適応度が，試行毎の報酬獲得に要する時間ステップの揺らぎに大きく依存してしまう．この場合，モジュールセットが適切であったとしても，偶然にも首尾良く報酬を獲得できなかった場合には急激に適応度が下がることがあるため，モジュールが安定しない．このためTは環境のノイズなど，ランダムな要素による性能の揺らぎの影響を受けない程度の値を選択する必要がある．

目標平均時間ステップ E_{tar}

E_{tar}は，獲得されるモジュールセットの質を決めるパラメータである．E_{tar}が大きい場合には，比較的簡単にモジュールセットが目標を達成しやすいため，より速くモジュールセットが安定するが，その分あまり適切でないモジュールセットでも安定してしまう可能性がある．このため E_{tar}は問題の難易度に応じて決定すべきである．また，エージェントが解決する問題の最適解が予測できない問題では，E_{tar}の値を学習時に動的に決定できる方法が必要であると考えられる．

モジュールセットの持つモジュール数 N_{mod}

モジュールに関するパラメータとして，モジュールセットが持つモジュール数 N_{mod}がある．本論文における実験では四つに固定した．追跡ゲームの実験においてモジュール数を少なくした場合には，適切なモジュールを獲得するのが困難であった．AMQL-LL アーキテクチャにおいて，モジュールは部分的な問題に関する行動方針であり，複数のモジュールを組み合わせることによりエージェントの全体の行動方針とする．この際モジュールの数が少なすぎると，たとえ適切な行動方針を学習できるモジュールが選択されても，それぞれ部分的な入力しか扱えないために，エージェント自体の行動方針として適切なものを得るのに不十分であり，このために適切なモジュールセットを獲得するのが困難になると考えられる．また，モジュールが多すぎる場合もあまり良い行動方針を獲得することができない．これは，たとえばノイズ的な入力に注目する冗長なモジュールが存在することにより，適切でない行動方針を導きだす可能性があることが理由として考えられる．実際に松林の行った追跡ゲームの実験においても，報酬獲得に全く関係のないノイズエージェントに注目するモジュールが多く存在すると，学習が収束するのが困難になるという結果が示されている [6]．さらに冗長なモジュールの存在は記憶資源を圧迫するという欠点もある．適切なモジュールの数は状態空間の広さやエージェントの持つセンサーの数など問題に依存する面がある．適切なモジュール数がエージェントが遭遇する問題毎に異なる場合には，モジュールの数を自動的に調節する機構が必要である．遺伝的アルゴリズムでは，お互いに異なる長さの染色体同士で交叉を行なう Messy GA などが提案されている [16]．AMQL-LL アーキテクチャにおいても，Messy GA のようにお互いに異なる数のモジュールを交換する方法が考えられる．また，エージェントが保持するモジュール群の中で予め固定されたモジュールセットを決めるのでなく，評価を行なう毎にモジュール群の中からランダムな数のモジュールを選択し，評価を行なう方法も考えられる．

モジュールセットの数 $N_{mod\,set}$

$N_{mod\,set}$はモジュール獲得までにかかる試行数などに影響する．$N_{mod\,set}$は遺伝的アルゴリズムにおける個体数に相当し，大きくなるほど優れたモジュールセットを獲得することが期待される．ただし AMQL-LL アーキテクチャは一つのエージェントの学習であり，モジュールセット集団を並列に評価を行なうことを仮定せず，評価は逐次的に行なう．このため $N_{mod\,set}$が大きくなることによりモジュールセットの評価に要する試行数も大きくなる．このため $N_{mod\,set}$には，モジュールセットの探索に十分で，かつできるだけ少ない数が要求される．

5.4.2 学習内容の保持について

AMQL-LL アーキテクチャでは，部分問題に対する行動方針をモジュールの形で保持する．エージェントは複数のモジュールを持ち，その中から適切なモジュールセットを遺伝的操作によって選び出す．この際，現在の問題に対して適切でないモジュールは淘汰される．つまり，現在使われることのない学習内容については忘れる．しかし，エージェントが複数の問題を同時に解決する場合には，ある問題を学習することで他の問題の学習内容を失うのは効率的でない．このためエージェントが複数の問題を同時に扱うような場合，ある問題に対して学習した内容を明示的に保存することが必要であると考えられる．遺伝的アルゴリズムの分野において，複数の問題を同時に扱う方法として免疫システムに着想を得たモデルがある [10]．これは各問題に対する最適の解を明示的に保存し，以前に学習した問題に再び遭遇した時にすぐに適切な解を得ることができるものである．複数の問題を同時に扱う場合には，このような枠組を AMQL-LL アーキテクチャに用いることが有効であると考えられる．ただし記憶資源が限られているため，やはりある程度の忘却の要素は必要であると思われる．

5.4.3 AMQL-LL アーキテクチャの評価方法について

本論文では追跡ゲームを用いて AMQL-LL アーキテクチャの評価を行なった．しかし，AMQL-LL アーキテクチャの有効性を示すには，より多くの問題で評価を行なう必要がある．しかし，これまでに強化学習の研究では様々なテストベッドが用いられているが，その多くは静的な問題である．今後強化学習の分野において動的問題を扱うにあたり，より現実に即した評価方法が要求される．

5.5 まとめ

これまで議論されたように，現時点のAMQL-LLアーキテクチャには幾つか解決すべき点や改良できる点が挙げられる．以下に今後の課題についてまとめる．

- より洗練されたモジュールの獲得方法．
- 一つのモジュールセットにおける適切なモジュール数の自動獲得．
- 学習記憶の保持によるマルチタスクへの適用．
- 様々な問題を用いた評価．

第 6 章

結論

従来のQ学習は，変化が小さい問題には追従できるものの，問題の急激な変化に対しては過学習などの理由により適切に学習できない場合がある．また，以前著者らは，Q学習をモジュール化により高速化させた構造化Q学習のモジュール化を自動化させたAMQLアーキテクチャを提案した．また，松林はAMQLアーキテクチャにおけるモジュール探索能力の弱さを改良したAMQL-GAアーキテクチャを提案したが，問題変化毎にこれまでの学習内容を初期化するため，動的問題に対して効率的には学習ができない．このような問題点に対し，本研究では，部分問題について学習するモジュール単位で既学習内容を保持し，問題の変化に応じて適切なモジュールセットを適用する機構であるAMQL-LL(Automatic Modular Q-Learning with Lamarckian Learning)アーキテクチャを提案した．また，シミュレーションにおける実験により，問題の変化に応じて既学習モジュールと未学習の新しいモジュールの中から適切なモジュールを選択し，問題変化の度合に応じた効果的な適応が可能なことを示した．

具体的には，分散人工知能のテストベッドである追跡ゲームを用い，動的問題における学習性能について従来のアーキテクチャとの比較を行なった．まず，追跡ゲームの基本的な枠組において評価を行ない，問題に対して適切なモジュールセットを獲得できることを示した．また，設計者によって与えられた固定モジュールを持つアーキテクチャよりも優れた行動方針を獲得できることを示した．次に，問題の変化が小さく，既存の学習を修正するという方法が効果的な場合における学習の収束速度について評価を行ない，AMQL-GAアーキテクチャとの比較を行なった．AMQL-LLアーキテクチャを用いた場合，問題の変化に対して既学習モジュールを選択・適用することにより効率的に学習を行なうことを示した．最後に，問題変化が大きく，特定の問題への過学習が変化後の問題の学習の妨げになる場合について評価を行なった．この場合，従来の固定されたQ学習アーキテクチャでは問題が動的に変化した際に，以前に学習した内容が問題変化後の学習に影響し，適切な行動方針を学習できなくなる．これに対しAMQL-LLアーキテクチャを用いた場合には，新しく生成されたモジュールなど変化後の問題に対して適切なモジュールを選択・適用することにより，適切な行動方針を学習できることを示した．

開放型環境において振舞うエージェントを構築するには，問題の動的変化を考慮した学習機構が必要である．本研究において提案したAMQL-LLアーキテクチャは，パラメータの設定やマルチタスク問題への応用などに課題が残されているが，今後開放型環境を構築するエージェントに応用したいと考えている．

謝辞

本研究の機会を与えられ，また日頃から懇切なご指導を頂きました慶應義塾大学理工学部客員教授 所 真理雄博士に心から深く感謝致します。

本研究に多大なるご指導と貴重な助言をくださった慶應義塾大学理工学部未来開拓プロジェクト 松林 慶氏に深く感謝致します．

日頃のディスカッションを始め，貴重な助言をいただいた本研究室博士課程 佐々木 貴宏氏，日頃から共に研究した本研究室修士課程 池谷 直樹氏，勝野 恭治氏，北脇 晴之氏，久曽神 宏氏，祢宜 知孝氏，三輪 泰孝氏，矢崎啓之氏，Pornchai LAOHAKULVATHIT 氏，お忙しいところ本論文の査読を引き受けて下さった慶應義塾大学理工学部助手 Joachim BACZEWSKI 博士，そして本研究室諸氏に心から感謝致します．

最後に学業を始めさまざまな面で支援して頂いた両親に，そして心のささえとなってくれた江木 恭子氏を始め，多くの友人に感謝致します．

1998年2月

Appendix A

遺伝的アルゴリズム

A.1 遺伝的アルゴリズム

遺伝的アルゴリズム (GA) は，生物進化 (選択淘汰・突然変異) の原理に着想を得たアルゴリズムであり，確率的探索・学習・最適化の一手法と考えることができる．これは，Holland の”Adaptation in Natural and Artificial Systems(1975)” において提唱された．

遺伝的アルゴリズムを説明する前に，まず基礎的な用語について説明する．染色体 (chromosome) が，遺伝情報を伝える実体として存在する．染色体の各位置にどのような遺伝情報が記述されているかが決まっているが，このような位置を遺伝子座 (locus) と呼ぶ．この各遺伝子座に対してその形質を決定するコードを遺伝子 (gene) と呼ぶ．そして，遺伝子の組合せのパターンを遺伝子型 (genotype)，遺伝子型に基づいて形成された個体を表現型 (phenotype) という．

遺伝的アルゴリズムでは，対象とする問題の解をビット列もしくは配列変数などで表現したものを遺伝子として扱う．この解の集団に対して Generate-and-Test 型のアルゴリズムにより処理を行なう．解に対しては，選択，交叉，突然変異などの遺伝的操作を施すことにより最適解を求める．遺伝的アルゴリズムの処理手順は以下のようになる．

1. 初期集団の生成
2. 適応度の評価
3. 評価時における適応度を基に 2 つの親を選択
4. 選択された 2 つの親を交叉
5. 交叉を終えた解の突然変異
6. 次の世代の解集団が生成された後に 2 へ戻る．

ここで，2 ~ 6 のループが問題の終了条件が満たされるまで繰り返される．

A.1.1 遺伝的操作

初期集団が生成されると，各々の個体に対して適応度の評価を行なう．評価の方法は問題によって異なる．一般に個体に対して適応度を導き出す評価関数などにより評価を行なう．

スケーリング

適応度が決定されても，この値をそのまま選択時の確率として用いるのではなく，何らかの関数を導入し，適応度の違いを拡大または縮小させることがある．このような関数を導入することをスケーリング (scaling) と呼ぶ．基本的な手法としては，(1) 線形スケーリング，(2) シグマ切断，および (3) べき乗スケーリングがある．

選択

選択とは次世代の親となる個体を選び出すことである．基本的に適応度の高い個体がより多く選択される機構となる．これまでにいくつかの方法が提案されているが，ここでは AMQL-LL アーキテクチャにおいて用いている 2 つの方法について説明する．

- 適応度比例戦略

  適応度比例戦略は，ルーレット・モデルまたはモンテカルロ・モデルとも呼ばれ，各個体の適応度に比例した確率で子孫を残せる可能性があるモデルである．ある個体 i が，各々の選択で選ばれる確率 P_{select_i} は，次式で表される．

  $$P_{select_i} = \frac{f_i}{\sum_{j=1}^{n} f_i} \tag{A.1}$$

  ここで，f_iは個体 i の適応度を示す．選択確率が与えられた時，$0 \sim 1$ の区間で乱数を発生させ，その値によってどの個体と，どの個体とを交配させるかを決定する．選択確率の大きな個体は複数回の交配に選択されるので，その遺伝子は集団中に広がっていく．

- エリート保存戦略

  エリート保存戦略は，集団中で最も適応度の高い個体をそのまま次世代に残す方法である．この方法を用いる場合には，その時点で最も良い解が交叉，突然変異によって壊されないという利点がある．ただしエリートの遺伝子が集団中に急速に広がる可能性が高いため局所解に陥る危険もある．エリート保存戦略は，適応度比例戦略などとの組合せで使われる．

交叉

適応度比例戦略などによって二つの親が選択された後に，両者の染色体の組変えによって新しい染色体を生成する．この組変えを交叉と呼ぶ．その方法には次のようなものがある．

- 単純交叉

  染色体を一点で切り，それ以降の遺伝子を交換する．

- 複数点交叉

  染色体を複数点で切り，遺伝子の交換を行なう．

- 一様交叉

  交叉時にマスクをかけ，それによってどちらの親の遺伝子を受け継ぐかを決定する．AMQL-GA アーキテクチャではビット毎のマスクを用いて交叉を行なう．また，AMQL-LL アーキテクチャではモジュール単位で交叉を行なうが，遺伝子座という概念がないため二つの親毎に交換するモジュールの数が同数となるようなマスクを用いていると言える．

突然変異

交叉を行なった後，各々の個体に対して突然変異の操作を行なう．一般的に，突然変異は固定された確率で各遺伝子が変化するように設定するが，変異率を動的に変化させる手法もある．突然変異率を大きく設定するとランダムサーチと化してしまうが，突然変異が無い場合には初期集団の遺伝子の組合せ以外の空間を探索することができないため解の質にも限界が出てくる．

参考文献

[1] 宮崎和光，山村雅幸，小林重信．$k-$確実探査法：強化学習における環境同定のための行動選択戦略．人工知能学会誌，10(3):454–463，1995．

[2] 山口智浩．仮想個体と実ロボットとの並列環境強化学習システムにおける諸問題．第 *5* 回マルチエージェントと協調計算ワークショップ *(MACC'97)*，1997．

[3] 山村雅幸，宮崎和光，小林重信．エージェントの学習．人工知能学会誌，10(5):683–689，1995．

[4] 山村雅幸．用語解説：強化学習．人工知能学会誌，8(6):833–834，1993．

[5] 小野　典彦，池田　治．構造化 Q 学習エージェント群による協調的行動の組織化．第 *4* 回マルチエージェントと協調計算ワークショップ *(MACC'95)*，1995．

[6] 松林慶，桑折隆之，所真理雄．構造化 Q 学習におけるモジュールの自動獲得．知能情報・高度情報処理シンポジウム，1997．

[7] 畝見達夫．強化学習．人工知能学会誌，9(6):830–836，1994．

[8] 那須正明，所真理雄．遺伝的アルゴリズムに関する研究．修士論文，慶應義塾大学，1992．

[9] 北野宏明．遺伝的アルゴリズム．産業図書，1993．

[10] 和田健之介，和田佳子．山登り飛び虫の進化と免疫システム論について．数理科学，(353):12–23，1992．

[11] M. Benda, V. Jagannathan, and R. Dodhiawalla. *On Optimal Cooperation of Knowledge Sources.* Technical Report BCS-G2010-28, Boeing AI Center, 1985.

[12] Egbert J.W. Boers and Herman Kuiper. *Biological metaphors and the design of modular artificial neural networks.* Master's thesis, Leiden University, 1992.

[13] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting Optimally in Partially Observable Stochastic Domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1023–1028, 1994.

[14] Lonnie Chrisman. Reinforcement Learning with Perceptual Aliasing: The Perceptual Distinctions Approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 183–188, 1994.

[15] Peter Dayan and Geoffrey E. Hinton. Feudal Reinforcement Learning. In *Advances in Neural Information Processing Systems 5*, pages 271–278, 1993.

[16] D. Goldberg, B. Korb, and K. Deb. *Messy Genetic Algorithms : Motivation, Analysis, and First Results.* Technical Report TCGA Report 89003, Department of Engineering Mechanics, The University of Alabama, Tuscaloosa, AL, 1989.

[17] John J. Grefenstette. Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms. *Machine Learning*, 3:225–245, 1988.

[18] Steven Alex Harp, Tariq Samad, and Aloke Guha. Towards the Genetic Synthesis of Neural Networks. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 360–369, 1989.

[19] John H. Holland. Escaping brittleness. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning II*, pages 593–623, Morgan Kaufmann, 1986.

[20] John H. Holland and J. Reitman. Cognitive systems based on adaptive algorithms. In D. Waterman and F. Hayes-Roth, editors, *Pattern directed inference systems*, pages 313–329, Academic Press, 1978.

[21] Robert A. Jacobs and Michael I. Jordan. Task Decomposition Through Competition in a Modular Connectionist Architecture: The What and Where Vision Tasks. *COGNITIVE SCIENCE*, 1991.

[22] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research 4*, 237–285, 1996.

[23] Hiroaki Kitano. Empirical Studies on the Speed of Convergence of the Neural Network Training by Genetic Algorithms. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 789–795, 1990.

[24] Hiroaki Kitano. *Neurogenetic Learning: An Integrated Method of Designing and Training Neural Networks Using Genetic Algorithms.* Technical Report, Carnegie Mellon University, 1992.

[25] Takayuki Kohri, Kei Matsubayashi, and Mario Tokoro. An Adaptive Architecture for Modular Q-learning. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 820–825, 1997.

[26] Long-Ji Lin. *Reinforcement Learning for Robots Uning Neural Networks.* PhD thesis, Carnegie Mellon University, 1992.

[27] Long-Ji Lin. Scaling Up Reinforcement Learning for Robot Control. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 182–189, 1993.

[28] Long-Ji Lin and Tom M. Mitchell. Reinforcement Learning With Hidden States. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 271–280, 1993.

[29] Geoffrey F. Miller and Peter M. Todd. Designing Neural Networks using Genetic Algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384, 1989.

[30] Norihiko Ono and Kenji Fukumoto. Multi-agent Reinforcement Learning: A Modular Approach. In *Proceedings Second International Conference on Multi-Agent Systems*, pages 252–258, 1996.

[31] A. L. Samuel. Some Studies in Machine Learning using the Game of Checkers. *IBM J. on Research and Development*, 3:210–229, 1959.

[32] Satinder P. Singh. Reinforcement Learning with a Hierarchy of Abstract Models. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 202–207, 1992.

[33] S. F. Smith. *A Learning System Based on Genetic Adaptive Algorithms.* PhD thesis, University of Pittsburgh, 1980.

[34] Richard S. Sutton. Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, 1990.

[35] Richard S. Sutton. Learning to predict by the methods of temporal differences. In *Machine Learning 3*, pages 9–44, 1988.

[36] Ming Tan. Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. In *Proceedings of the Tenth International Conference on Machine Learning*, 1993.

[37] Sebastian Thrun and Anton Schwartz. Finding Structure in Reinforcement Learning. In *Advances in Neural Information Processing Systems 7*, pages 385–392, 1995.

[38] C.J.C.H. Watkins. *Learning from Delayed Rewards.* PhD thesis, University of Cambridge, 1989.

[39] C.J.C.H. Watkins and Peter Dayan. Technical note: Q-Learning. *Machine Learning*, 8(3):279–292, 1992.

[40] Steven Whitehead, Jonas Karlsson, and Josh Tenenberg. Learning Multiple Goal Behavior via Task Decomposition and Dynamic Policy Merging. In *ROBOT LEARNING*, Kiluwer Academic Press, 1993.

[41] Darrell Whitely and Thomas Hanson. Optimizing Neural Networks Using Faster, More Accurate Genetic Search. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 391–396, 1989.