Vol.2012-GI-27 No.11
2012/3/2

連続状態行動空間での木探索によるオンライン協調行動プランニング

秋山英久[†1]

本稿では，連続状態行動空間において複数エージェントによるオンライン協調行動プランニングを可能にするための探索フレームワークを提案する．従来は，状態行動空間が連続，かつ，実時間性が要求されるタスクに対して木探索手法を適用することは困難であった．しかしながら，近年の計算機能力の向上に伴い，オンラインでの探索アプローチが可能となってきた．そこで，本研究では RoboCup サッカー 2D シミュレーションを実験環境として，木探索による協調行動プランニングを実現する．実験結果より，探索がチームパフォーマンスに与える影響について考察する．

Online Cooperative Action Planning using Tree Search in a Continuous Action State Space

Hidehisa Akiyama[†1]

In this paper, we propose a framework to search action sequences in order to enable an online cooperative action planning in multiagent systems. It was difficult to apply a tree search methodology to tasks that the action-state space is continuous and requires realtimeness. However, it has become possible to apply such an approach since the computational resources became more powerful today. We applied a tree search method to the RoboCup soccer 2D simulation environment and analyzed its effectiveness by evaluating the team's performance.

†1 福岡大学工学部
Faculty of Engineering, Fukuoka University

1. はじめに

サッカーのようにチームで対戦するゲームでは，複数のプレイヤによる協調行動を実現しなければ，チームとしてのパフォーマンスを向上させることは難しい．協調行動を実現するには，ある目標状態に到達するために複数のプレイヤによる行動の連鎖をプランニングする必要がある．このようなプランニングを実行するには，適切な行動の列をオンラインで生成・評価し，決定されたプランをプレイヤ間で共有しなければならない．しかしながら，状態空間・行動空間が連続なだけでなく，意思決定に実時間性が求められるタスクでは，従来のゲーム木探索のような計算機の能力を駆使するアプローチの適用は困難であった．この問題に対して，近年の計算機の性能向上に伴い，探索に基づいたプランニングを実時間性を損なうこと無く実行することが可能となってきた．

本稿では，連続状態行動空間における複数エージェントによる行動連鎖をオンラインで生成・評価する探索フレームワークを提案する．このフレームワークを用いることでオンラインでの協調行動プランニングを実現し，チームパフォーマンスの向上を試みる．探索によるオンライン協調行動プランニングの有効性を示すために，RoboCup サッカー 2D シミュレーション環境を用いた評価実験を行う．実験では，提案するフレームワークを実装したエージェントプログラムを用いてシミュレーションによる評価実験を行う．実験結果から，探索の設定変更がチームパフォーマンスに与える影響について考察する．

2. 従来研究

RoboCup サッカー 2D シミュレーション[2] はマルチエージェントシステムのテストベッドとして知られている．2D サッカーシミュレータは，さまざまな機械学習手法の適用だけでなく，戦略，戦術の枠組みに関する研究のためのプラットフォームとしても利用されている．

RoboCup サッカー 2D シミュレーションにおける従来の取り組みでは，強化学習研究に関連して，プレイヤエージェントの協調的な意思決定あるいは最適な個体制御を獲得させる研究事例が多い．特に，未知の行動ポリシーを持つ敵対エージェントが存在する環境における意思決定性能改善が目指されている．Stone らは強化学習のテストベッドとして Keepaway というサッカーのサブタスクを提案している[6]．また，Gabel らは敵エージェントの攻撃的行動に対する 1 対 1 の守備タスクを設定し，実用的な性能を持った意思決定能力を強化学習によって獲得することに成功している[7]．強化学習の枠組みにおいては，エージェントが

意思決定する機会において最良の行動選択を行う能力の獲得が期待できる．個体制御や状況を限定したタスクでの性能向上に関しては，強化学習は極めて有望なアプローチと言えるだろう．しかしながら，強化学習のようなボトムアップのアプローチでは，実際のサッカーの試合のように多数のエージェントが存在する動的な環境下において，戦略・戦術的な意思決定能力を獲得することは現実的には困難である．

一方，戦略や戦術に基づいた集団制御の実現に関しては，トップダウンなアプローチが取られている例が多い．トップダウンなアプローチとして，Situation Based Strategic Position[4] や Locker Room Agreement[3] などの，事前知識として戦略や戦術をエージェント間で共有しておく手法が効果的であることが知られている．Situation Based Strategic Position は，戦略のデザインをチームのフォーメーションとして捉え，状況に応じたエージェントの配置を事前に決めておくアプローチである．Locker Room Agreement では，特定の条件下で実行すべき戦術を事前知識として静的に共有しておくことで，環境の状態が条件に合致した際に固定的なプランが遂行される．これらの手法によって，複数エージェントの協調的な振る舞いを見かけ上は実現可能である．しかし，実行される行動列や目標状態が固定的であるために柔軟性に乏しいといった問題がある．

トップダウンなアプローチとしては，環境を俯瞰して観察できるコーチエージェントを用意し，部分情報しか観測できないプレイヤエージェントへコーチエージェントからアドバイスを与えるという取り組みもなされている。サッカーにおける戦略・戦術記述言語としては，Reis らによる Coach Unilang が提案されている[5]．2D サッカーシミュレータには，コーチによるアドバイスを実現するためのコーチ言語が実装されており，サッカーのアドバイスに特化した言語仕様が策定されている[1]．Coach Unilang の仕様の一部はこのコーチ言語に反映されている．コーチ言語は実際の競技会においても使用可能である．しかしながら，コーチエージェントを利用した戦略や戦術のアドバイスが有効に機能している事例はまだ十分に報告されていない．これは，プレイヤエージェントの意思決定におけるプランニング能力に十分な柔軟性を持たせられておらず，コーチエージェントからのアドバイスが適切であったとしても，プレイヤエージェントがそれらを反映できないことが原因と考えられる．従来のアプローチの問題を解決するには，オンラインでの協調行動プランニングを実現し，動的な環境や未知の対戦相手へより柔軟に対応する必要がある．

3. 木探索による協調行動プランニング

本稿では，連続状態行動空間におけるオンライン協調行動プランニングを実現するための探索フレームワークを提案する．本稿では，複数エージェントによって一定時間継続して実行される一連の行動連鎖を戦術と定義し，この戦術を協調行動としてプランニングする．提案するフレームワークでは，木探索によって有効な行動の連鎖を生成・評価することで複数エージェントによる協調行動のプランニングを実現する．

3.1 行動連鎖探索フレームワーク

提案するフレームワークは，自分と他者を含めた複数のエージェントによって実行されうる行動（サッカーの場合はパス，ドリブル，シュートなど）を生成し，探索木にノードとして格納していくことで有効な行動連鎖の探索を実行する．図 1 にサッカーにおける行動連鎖のイメージ図を示す．この図では，10 番のプレイヤを起点とする 4 つの行動の連鎖を表している．本稿では，このようなサッカーにおけるボールを扱う行動の連鎖を扱う．

提案するフレームワークには，行動および行動を実行後の状態を生成，評価するメカニズムが用意されている．このメカニズムは，以下の 3 つのクラスによって実現される．

- CooperativeAction クラス
- ActionGenerator クラス
- FieldEvaluator クラス

これらはオブジェクト指向における抽象クラスを意味する．フレームワークを利用する場合，要求されるタスクに応じて，これらのクラスから派生した具象クラスを実装しなければならない．

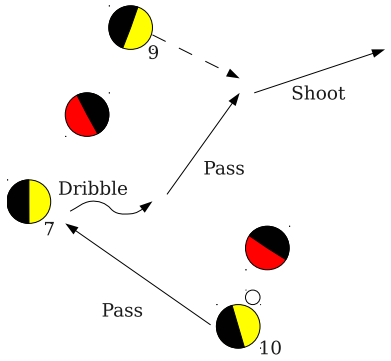


図 1 サッカーにおける行動連鎖のイメージ図．10 番から 7 番へのパス，7 番によるドリブル，7 番から 9 番へのパス，9 番によるシュート，という 4 つの行動の連鎖を表す．

3.1.1 CooperativeAction クラス

CooperativeAction は，探索実行時に最小単位となる行動を表す抽象クラスである．サッカーの場合であれば，パス，ドリブル，シュートなどの，サッカープレイヤとしてある程度意味のある行動を具象クラスとして実装することになる．

このクラスは，行動のタイプ，目標状態，目標状態に到達するまでに要する時間などの情報を保持する．エージェントはこれらの情報を参照することで最終的な自身の制御を行う．

3.1.2 ActionGenerator クラス

ActionGenerator は，エージェントが観測した現在の環境の状態あるいは予測した環境の状態を入力とし，その状態において取りうる CooperativeAction を生成する抽象クラスである．通常，ある入力状態に対して複数の CooperativeAction の候補を生成することができる．例えば，ボールを所有するプレイヤがパスのアクションを実行しようとする場合，無数のパスコース候補が存在する．生成したい行動のタイプや問題の性質に応じて，適切な具象クラスを設計・実装する必要がある．

成功が予測される CooperativeAction が生成されると，その結果の予測状態が同時に生成され，ActionStatePair が作られる．ActionStatePair は，CooperativeAction と予測状態とを単純に組にしたものである（図 2）．実際に探索木のノードに格納されるのは，この ActionStatePair である．

ActionGenerator は，既に探索木に入れられている ActionStatePair を入力として，さらに ActionStatePair を生成することができる．首尾良く生成された ActionStatePair は新しいノードとして探索木に加えられる．このとき，新規に生成されたノードは，入力となった親ノードの子ノードとなる．

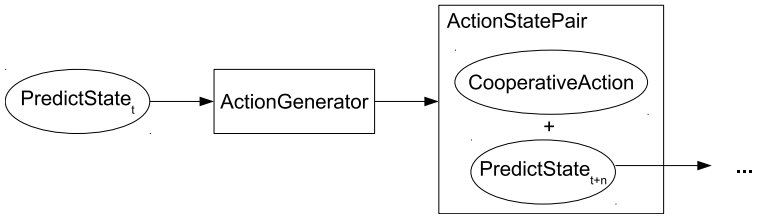


図 2 ActionGenerator の模式図．時刻 t の予測状態において実行可能な行動と，その実行結果となる時刻 $t+n$ での予測状態を生成する（n は行動完了まで要する時間とする）．生成された予測状態はさらに次の ActionGenerator で利用可能となる．

3.1.3 FieldEvaluator クラス

FieldEvaluator は，新規に生成された ActionStatePair の評価を担当する．実際の利用時には，評価値を返すメソッドを定義した具象クラスを実装する必要がある．

評価を実行する際には，新規作成されたノードからルートノードへ繋がる ActionStatePair 列全体が FieldEvaluator への入力となる．FieldEvaluator は評価値として実数を返し，入力となった ActionStatePair 列はこの値を保持する．探索木の走査終了後，エージェントは生成された ActionStatePair 列の中からもっとも高評価のものを選択する．現在状態においてエージェントが実際に実行するのは，ルートノードで生成された CooperativeAction である．

3.2 探索木の構造と走査アルゴリズム

採用する探索木の模式図を図 3 に示す．図で示すように，フレームワークでは多分探索木によって探索処理を実行する機能が提供されている．図中のノードには，それぞれ一つの ActionStatePair が格納される．木のルートノードから葉ノードまでのノード列をつなげると，ある行動連鎖が得られる．

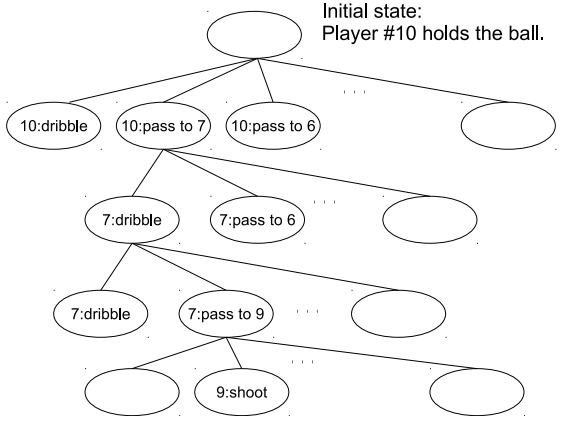


図 3 フレームワークで採用する探索木の模式図．多分探索木を採用する．各ノードは ActionStatePair を格納する．各ノードが持つ予測状態が子ノードへの入力となる．

探索木の走査アルゴリズムとして，本稿では単純な最良優先探索を用いる．ノードとして ActionStatePair が生成される際に，FieldEvaluator によってノードの評価値が得られる．

この評価値は，最良優先探索のためのヒューリスティック値としても利用される．新規ノードが追加されるごとに，評価値に基づいてノード列を格納する優先順位付きキューが更新される．ノードの走査は優先順位付きキューでの格納順に実行される．

探索木の走査では，葉ノードに到達した時点で探索の終了条件を満たしているかどうかを確認する．本稿では，以下を終了条件とする．

- 走査したノード数があらかじめ設定した最大数を越えた場合．
- CooperativeAction が新規生成されなかった場合．

葉ノードにおいて以下の条件のいずれかを満たす場合は，その葉ノードでの新規子ノードの生成は行われない．

- 深さがあらかじめ設定した木の深さの最大数を越えた場合．
- 入力された予測状態から CooperativeAction を生成できなかった場合．
- 行動連鎖の終了と設定されている CooperativeAction(例えばシュート) が格納されている場合．

3.3 サッカーエージェントへの実装

3.3.1 CooperativeAction

本稿の実験で使用するサッカーエージェントには，CooperativeAction として以下の行動を実装した．いずれもボールを扱う行動のみで，エージェントの移動のみの行動はここでは対象としない．

- Dribble：ボールを保持しつつ，自分自身とボールを移動させる．行動生成時にドリブルによる移動距離と移動方向が設定されている．
- Hold：その場にとどまってボールを保持する．ドリブルとは異なり，自分自身は移動しない．行動生成時にボールの移動位置は設定されない．
- Pass：他の味方へボールをパスする．行動が生成された時点で，パスが出される位置，ボールの初速度，パスの受け手，パスを受ける位置などが設定されている．
- Shoot：敵ゴールへシュートする．行動生成時にシュートコースとボールの初速が設定されている．シュートは行動連鎖の終了と設定されており，シュート実行後の予測状態から新しい行動は生成されない．

3.3.2 ActionGenerator

本稿の実験で使用するサッカーエージェントには，計算資源節約のために大きく分けて二種類の ActionGenerator を用意する．ひとつは結果の予測計算を厳密に行うもの，もうひとつは簡易な計算で予測を行うものである．具体的には，探索木の深さ 1 の行動生成時，すなわち現在状態からの行動生成には厳密計算を採用し，深さ 2 以上では簡易計算を採用する．これは，深さ 2 以上では予測状態の精度が粗く，厳密に計算する意味が乏しいためである．現在の実装では，予測状態の精度は極めて粗く，ボールに関わらないプレイヤの位置情報は予測されていない．そのため，厳密計算と簡易計算では予測結果の一貫性は保たれていない．

実装されている ActionGenerator を以下に示す．

- OmniDribble（厳密計算）：全方位移動を利用した短距離のドリブルを生成する．移動距離と移動方向が異なる最大 24 パターンのドリブルを生成する．
- ShortDribble（厳密計算）：移動方向へ体を向けてから直進するのみの短距離ドリブル．移動距離と移動方向が異なる最大 48 パターンのドリブルを生成する．
- LongDribble（厳密計算）移動方向へ体を向けてから直進するのみの長距離ドリブル．移動距離と移動方向が異なる最大 300 パターンのドリブルを生成する．
- SimpleDribble（簡易計算）：ドリブルの成功判定を簡易にしか行わない．移動距離と移動方向が異なる最大 24 パターンのドリブルを生成する．
- Hold （厳密計算，簡易計算）：ボールを保持する行動．1 パターンのみ．
- StrictCheckPass（厳密計算）：現在状態で実行して成功が予測されるパスを生成する．敵プレイヤの移動予測まで厳密に計算する．最大で 2250 パターンのパスを生成する．
- Cross (厳密計算)：敵ゴール周辺に存在する味方プレイヤへ向けたパスを生成する．最大で 300 パターンのパスを生成する．
- DirectPass (簡易計算) ： 最大で 10 パターンのパスを生成する．
- VoronoiPass（簡易計算）： 敵プレイヤの配置に基づいたボロノイ図の辺上にレシーブ位置を設定し，パスを生成する．最大パターン数は未定義．
- Shoot（簡易計算）：最大 10 パターンのシュートを生成する．

3.3.3 FieldEvaluator

本稿の実験で使用する ieldEvaluator にはさまざまな評価項目が導入されており，人手による調整が施されている．使用されている評価項目を以下に挙げる．これらの項目に基づいて人手でルールを設計し，評価値に加点・減点することで最終的な評価値を決定する方式を採用している．

- ボール位置座標
- シュートコースの存在
- 敵のディフェンスラインとの位置関係

- 敵プレイヤとの距離
- 行動連鎖実行にかかる時間

これらの実装により，図 4 に示すような行動連鎖の生成が可能となった．図中では，右上で 9 番のプレイヤがボールを保持しており，9 番から 11 番へのパス，11 番から 6 番へのパス，6 番によるシュート，という行動連鎖が決定されている．

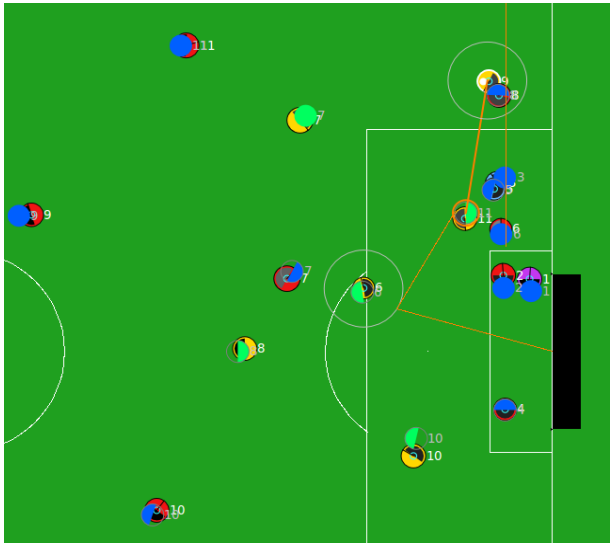


図 4 生成された行動連鎖の例．黄色の 9 番のプレイヤによって生成された行動連鎖を表している．図中の黄緑色と青色の円は，黄色の 9 番のプレイヤが内部状態として持っている他プレイヤの情報を表している．

4. 評価実験

4.1 実験設定

探索木の設定変更によるチームのパフォーマンス変化を調査するために，以下の各項目の組み合わせを用いて試合を実行する．チームのパフォーマンスの指標として得失点差を用いる．

- 木の深さの最大数：{ 1, 2, 3, 4, 5, 6, 7 }
- ノード走査の最大数：{ 10, 100, 1000, 10000, 100000 }
- ActionGenerator：{ 実際の競技会で使用されたもの，実際の競技会で使用したものよりも行動生成数を減らしたもの }
- FieldEvaluator：{ 実際の競技会で使用された評価関数，ボール位置のみに基づく単純な評価関数 }

実行するチームとして RoboCup2011 で準優勝した HELIOS を，対戦相手として agent2d-3.1.0[⋆18] を使用する．ただし，上記の設定変更に伴い，HELIOS は競技会に参加したものとは異なる挙動を示すようになる．実際の競技会で用いた設定では，木の深さの最大数は 4，ノード操作の最大数は 1000 である．

上記の各設定で 1 ハーフの試合を 20 試合ずつ，合計 2800 試合を実行する．ただし，木の深さやノード走査の最大数が極端に大きい場合，実時間での計算は間に合わない．そこで，今回の実験では，2D サッカーシミュレータに用意されている同期モードを用いることで計算時間を無視する．

4.1.1 評価関数の設定

実験では 2 種類の評価関数を用意し，それぞれ試合を実行する．図 5 に，それぞれの評価関数による評価値を 2D サッカーシミュレータのフィールド上にマッピングした様子を示す．これは，フィールドを小さなグリッド ($0.5m \times 0.5m$) に区切り，各グリッド中央にボールが到達した状態の評価値に基づいて色分けをしたものである（フィールドの広さは $105m \times 68m$）．図中の赤色がもっとも評価が高い地点を表し，黒色に近づくにつれて評価値が低くなっていることを意味する．

一つ目の評価関数（図 5(a)）は，RoboCup2011 で準優勝した HELIOS が実際に競技会で使用したものである．さまざまな評価項目が導入されており，人手による調整が施されている．二つ目の評価関数（図 5(b)）は，フィールド上の位置座標と敵ゴールへシュートコースが存在するかどうかのみを評価項目としている．一つ目に比べて単純な評価関数となっている．

4.1.2 ActionGenerator の設定

ActionGenerator の設定を変えた 2 パターンの実験を行う．ひとつめの設定では，3.3.2 節で述べた ActionGenerator を用いる．ふたつめの設定では，これらを改変して行動の最大生成数をそれぞれ約半分に減らした ActionGenerator を用意する．

⋆1 http://sourceforge.jp/projects/rctools/ より入手可能

4.2 実験結果

各実験設定で実際に試合を行った結果の得失点差を図 6 と図 7 に示す．得失点差の値はすべて 20 試合の平均値である．

HELIOS の評価関数を使用した場合では，最大走査数が 10 の設定で明らかにパフォーマンスの低下が見られる．生成する行動の数に比べて明らかに走査数が足りていないため，充分な探索が行えずにパフォーマンスが低下したと考えられる．一方で，最大走査数が 100 以上の結果には顕著な差は見られない．最大走査数が 100 の場合が比較的好成績を収めており，ノードの走査数とチームパフォーマンスの間にあまり相関が見られない結果となっている．また，木の最大深さが 2 の場合が比較的良い結果となっている．このような結果になった原因については調査中である．

一方で，単純な評価関数を使用した場合では，探索木の設定変更によるチームパフォーマンスの変化がほとんど見られない結果となった．これは，使用した評価関数が最良優先探索におけるヒューリスティック関数として適していなかったことが原因と考えられる．

図 8 に最大走査数を 1000 に限定した場合の比較結果を示す．この結果からは，HELIOS の評価関数を用いた方が比較的良いパフォーマンスが得られていることが分かる．

4.3 考　　察

走査するノードの最大数が極端に小さい場合，ほぼすべての状況において深さ 1 の探索となってしまうため，生成される CooperativeAction の順序によってチームの特徴が決定してしまう．今回の実験では，パス行動が最初に生成されるように実装されていたため，最大走査数が 10 の場合にはほぼパスしか生成されなくなる．その結果，ほとんどパスしか実行しないチームとなり，パフォーマンスが低下したと考えられる．

一方で，人手で調整した評価関数を用いれば，ノードの走査数が一定の値を越えるとチームのパフォーマンスが安定してくることが分かる．これは，より多くのノードを走査することで，局所解に陥りにくくなるためであると予想される．しかし，今回の実験では，ノードの走査数を増やしても劇的にパフォーマンスが改善されることはなかった．探索の深さと得失点差との間に相関が見られていない点にもより詳しい分析が必要である．

今回の実験では各設定での試合数が 20 と少なかったために，得失点差のみでは有意な差が現れなかったと予想される．試合数を増やすだけでなく，チームパフォーマンスを測る指標を複数導入することで，探索が与える影響をより詳細に分析する必要があるだろう．

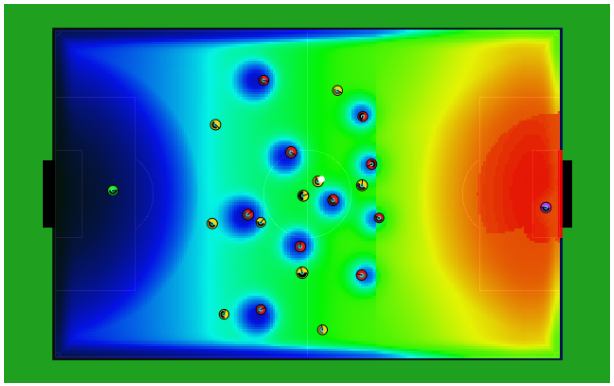
5. おわりに

本稿では，木探索を用いて行動連鎖を探索するフレームワークを提案し，これを用いることでオンラインでの協調行動プランニングの実現を試みた．RoboCup サッカー 2D シミュレーション環境において，フレームワークを実装したエージェントを用い，探索木の深さ，走査するノード数，評価関数，生成される行動の数の変更によってチームのパフォーマンスに現れる影響を測った．

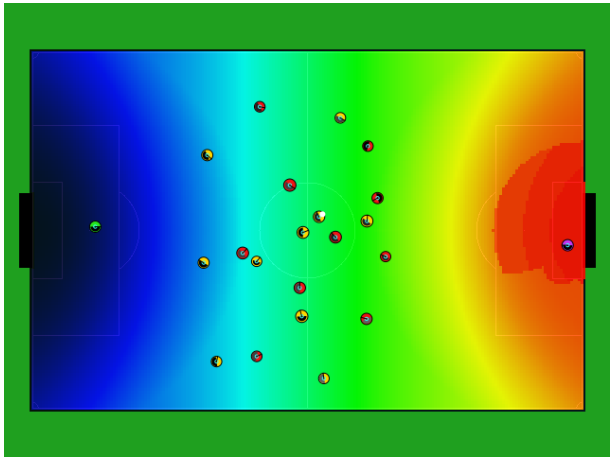
今後の課題として，適切な評価関数の設計とより効率的な探索アルゴリズムの開発が挙げられる．これらを実現するには，まずはより多様な評価基準に基づいてチームの性能分析を進める必要があるだろう．

参 考 文 献

1) The RoboCup Soccer Simulator Users Manual, http://sourceforge.net/projects/sserver/files/rcssmanual/
2) Itsuki Noda and Hitoshi Matsubara: Soccer Server and Researches on Multi-Agent Systems, Proc. of IROS-96 Workshop on RoboCup, pp. 1-7, (1996)
3) Peter Stone and Manuela Veloso: Task Decomposition, Dynamic Role Assignment, and Low-Bandwidth Communication for Real-Time Strategic Teamwork, Artificial Intelligence, 110(2), pp.241-273, (1999)
4) Luis Paulo Reis, Nuno Lau and Eugenio C. Oliveira: Situation Based Strategic Positioning for Coordinating a Team of Homogeneous Agents in Markus Hannebauer, Jan Wendler and Enrico Pagello Editors, Balancing Reactivity and Social Deliberation in Multi-Agent System From RoboCup to RealWorld Applications, Springer LNAI, Vol. 2103, pp. 175-197, (2001)
5) Luis Paulo Reis and Nuno Lau: COACH UNILANG - A Standard Language for Coaching a (Robo) Soccer Team, RoboCup-2001: Robot Soccer World Cup V, Springer Verlag LNAI, Vol. 2377, pp. 183-192, Berlin, (2002)
6) Peter Stone, Richard S. Sutton and Gregory Kuhlmann: Reinforcement Learning for RoboCup-Soccer Keepaway, Adaptive Behavior, 13(3), pp. 165-188, (2005)
7) Thomas Gabel, Martin Riedmiller and Florian Trost: A Case Study on Improving Defense Behavior in Soccer Simulation 2D: The NeuroHassle Approach. RobopCup 2008: Robot Soccer World Cup XII. pp. 61–72, (2008).
8) 秋山英久: ロボカップサッカーシミュレーション 2D リーグ必勝ガイド, 秀和システム, (2006)

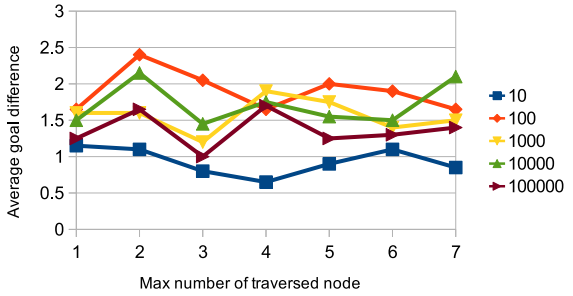


(a) HELIOS が実際に競技会で使用した評価関数．さまざまな評価項目を組み合わせ，人手で調整したものを用いる．

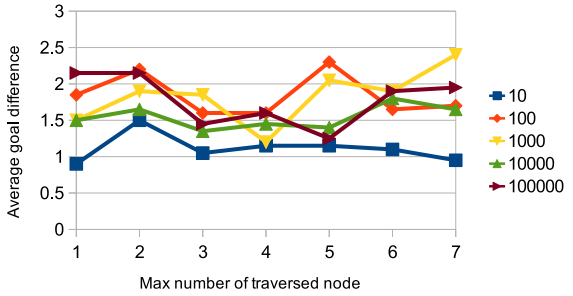


(b) 単純な評価関数．フィールド上の位置座標とシュートコースの判定のみを評価項目とし，敵プレイヤの配置などは考慮されない．

図 5 実験で使用した評価関数をフィールド上にマッピングした例．

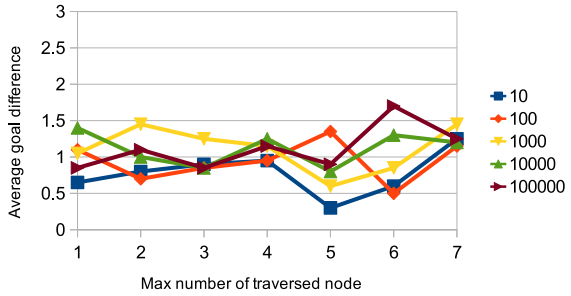


(a) HELIOS の評価関数を使用．

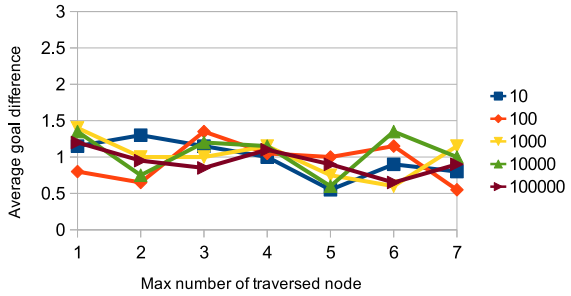


(b) HELIOS の評価関数を使用（行動生成数を抑制）．

図 6 探索木の設定変更によるチームパフォーマンスの変化（実際の競技会で用いた評価関数を使用した場合）．グラフの各線は走査するノードの最大数の設定に対応している．横軸は探索木の深さの最大数，縦軸は 20 試合の得失点の平均値を表す．



(a) 単純な評価関数を使用.



(b) 単純な評価関数を使用（行動生成数を抑制）.

図 7 探索木の設定変更によるチームパフォーマンスの変化（単純な評価関数を使用した場合）．グラフの各線は走査するノードの最大数の設定に対応している．横軸は探索木の深さの最大数，縦軸は 20 試合の得失点の平均値を表す.

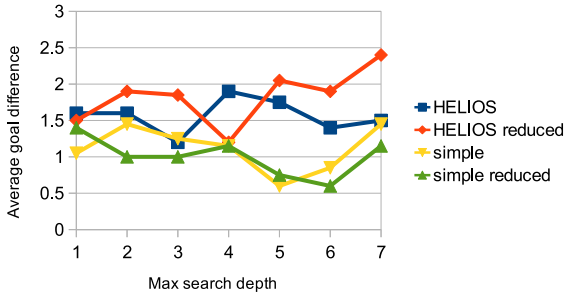


図 8 探索木の最大走査数を 1000 に限定した場合の比較．グラフ中の各線は，HELIOS は HELIOS の評価関数を使用したもの，simple は単純な評価関数を使用したものを意味する．また，reduced は行動生成数を抑制した場合を意味する.