修士学位論文

連続空間の階層的分割を用いた
マルコフ決定過程における
行動選択に関する研究

2017年度

広域科学専攻 広域システム科学系

31-156861

大渡 勝己

論文要旨

連続空間の階層的分割を用いた
マルコフ決定過程における
行動選択に関する研究

広域科学専攻 広域システム科学系
31-156861
大渡 勝己

本研究ではマルコフ決定過程におけるプランニング課題を扱った．マルコフ決定過程は時間について離散的にとらえた環境モデルであるが，現実世界でのリアルタイム性の低い課題に対しては有効なモデル化である．

本研究ではマルコフ決定過程の中でも，より現実の課題への応用が期待される「状態空間が連続であり，状態遷移に不確定性がある環境」でのプランニング課題を対象とした．

離散の空間に対しては，環境のモデルを利用したシミュレーションを繰り返し行うことで行動選択を行う「モンテカルロ木探索」という手法が成果を残している．

本研究ではこの手法をベースとし連続な状態空間を扱うことを可能にするアルゴリズムを提案した．

本研究の提案アルゴリズムでは，連続な状態空間を階層的に分割し，分割したそれぞれの状態集合にシミュレーション中に得た報酬の知識を記録していくことで

- 連続な状態空間において「近い」状態においては行動の価値も近いという経験則を活用する
- シミュレーションを高速に行える

の 2 点をいずれも満たすものである．

提案アルゴリズムは，低次元空間におけるベンチマーク課題において，モンテカルロ木探索を連続な状態空間に適用した既存手法よりも効率よくプランニングを行えることを実験にて実証することができた．

また最大 30 次元の状態空間を持つカーリング課題にも提案アルゴリズムを適用し，高次元の課題に対しても提案アルゴリズムがはたらくことを確認できた．

目 次

第 1 章　はじめに **1**

第 2 章　連続空間 MDP におけるプランニング **4**
2.1 モンテカルロ法によるプランニング 5
2.2 モンテカルロプランニングにおける方策 5
2.3 バンディット問題 6
2.4 離散空間におけるモンテカルロ木探索 7
2.5 連続状態空間におけるバンディット問題 9
2.6 連続状態空間におけるモンテカルロ木探索 9
2.6.1 Progressive Widening 10
2.6.2 Double Progressive Widening 10
2.6.3 Weighted UCT 12

第 3 章　連続空間を階層化して価値を近似するプランニング **13**
3.1 木構造による状態分割 13
3.2 状態空間の分割法 14
3.3 状態木の作成単位 14
3.4 状態木のノードに保存するデータ 14
3.5 状態木の展開 14
3.6 状態木からの楽観的行動価値の推定 15
3.7 状態木を利用したシミュレーション 16
3.8 再帰的状態分割を利用したモンテカルロ木探索 16
3.9 計算量 19
3.9.1 時間計算量 19
3.9.2 空間計算量 20

第 4 章　提案手法の簡単なプランニング問題への適用 **21**
4.1 Trap Problem への適用 21
4.1.1 課題設定 21
4.1.2 実験条件 22
4.1.3 実験計画 23
4.1.4 実験結果 24
4.2 Treasure Hunt Problem への適用 24
4.2.1 課題設定 24
4.2.2 実験条件 25
4.2.3 実験計画 26

4.2.4 実験結果 27

第 5 章 提案手法のカーリングへの適用 34
5.1 カーリング 34
5.2 デジタルカーリング 34
5.3 カーリングの行動決定における既存研究 36
5.4 デジタルカーリングの問題定式化 36
5.5 デジタルカーリングに対する提案手法の適用詳細 37
5.5.1 状態木の実装 37
5.5.2 シミュレーションの実装 38
5.5.3 モンテカルロ木探索の実装 40
5.5.4 その他の実装上の工夫 43
5.6 対戦実験計画 43
5.7 実験結果 43

第 6 章 考察 45

第1章　はじめに

現代社会において計算機による計算は人間の生活に欠かせないものとなっている．金融の処理など人間が短時間では不可能な計算を計算機が行うことで大規模な社会システムが形作られており，検索エンジンにおける自然言語からの情報の検索は日常の様々な場面で役に立つものである．このように社会の様々な側面で計算機は重責を担って，計算機が現在より広い範囲の課題を扱えるようになればさらに便利になることが期待される．

計算機で今後実現したい課題は多数あるが，本研究では連続状態空間のマルコフ決定過程（Markov Decision Process; MDP）におけるプランニング課題を扱う．MDP に関しては後ほど詳細な定義を述べるが，これは大まかには計算機にあるタスクを与え，それを解決するための方法を発見させるという課題であり，それを実世界のような連続的な空間内で扱えるようにすることを目標とする．

これまで離散空間におけるいくつもの課題で，計算機によるプランニングでは人間プレイヤの能力を上回ることが示されてきた．その一方で，より現実的な環境におけるプランニングにおいては人間の能力に追いついていない分野も多い．

連続空間は整数の金額の処理などとは異なり，計算機で厳密に値を表現することができない．また連続なパラメータのある状態空間は有限の範囲においても非可算無限個の状態を取りうるので，可能なパターンを全て列挙して制御するということが小規模な問題であっても不可能であることも，課題を難しくしている．

本研究の将来において解決したい課題として，例えば塗装の問題がある．塗装は均一に色を塗るというゴールのはっきりとした課題である．にもかかわらず，皿のような小さいものから，重機のような大きなものであっても，熟練された人間が塗装を行ったほうが質が高いことが多くある．

さらに，より身近な例で言えば，洋服をたたむこと，部屋の片付けをすることなど，人間が可能であるが手間のかかる作業がある．今後ロボティクスと計算機によるプランニングが結びつくことで自動化が期待される課題の一つである．

これらの課題における行動プランの作成と制御は短時間で終了するに越したことはない．しかし，リアルタイム性の要求は車の自動運転課題などに比べれば低いと考えられる．そういった場合には時間を離散化した上でモデル化を行いプランニングを行うことができる．このような環境のモデル化としてマルコフ決定過程は広く用いられている．

マルコフ決定過程（MDP）の一部を図で表すと，図 1.1 のように表現される．

図 1.1 において記号の添字は時刻（$0, 1, 2, ...$）を表す．

$s_0, s_1, s_2, ...$ はそれぞれの時刻での状態を表す．ここで状態とは，エージェントが各時刻に観測可能な情報とする．

$A_0, A_1, A_2, ...$ はエージェントがそれぞれの時刻取った行動を表す．エージェントは各時刻において観測した状態を入力として行動を選択し出力する．

$R_0, R_1, R_2, ...$ はエージェントがそれぞれの時刻で環境から受け取る報酬を表す．即時報

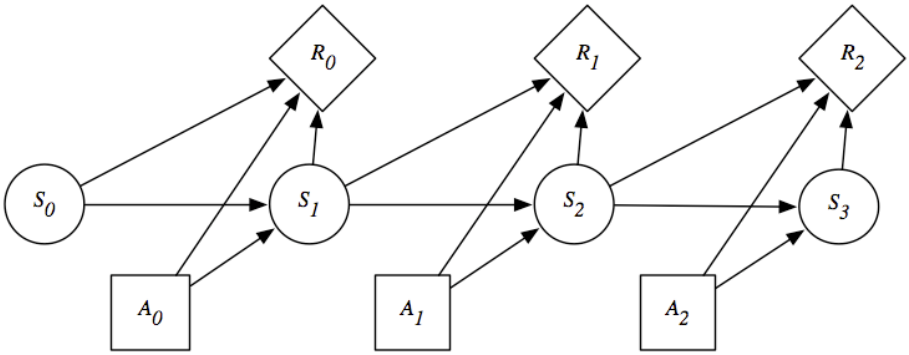

図 1.1: マルコフ決定過程（MDP）[1] より引用

酬とはエージェントが行動を起こした時刻に即座に手に入る報酬を指す．報酬はそれぞれの状態にてエージェントが行動を選択して実行し，次の状態の観測と同時にフィードバックされるものとする．

図 1.1 中の矢印は，MDP における各要素の間の因果関係を示している．たとえば s_1 は一つ前の状態 s_0 と，そのときに実行した行動 A_0 によって定まる．ただし決定的に定められるとは限らない．この状態の変化を状態遷移と呼ぶ．報酬 R_0 は前の状態 s_0，後の状態 s_1，行動 A_0 によって定まる．ただしこちらも決定的とは限らない．また状態遷移と報酬の別の定式化として，(前の時刻の状態, 行動）から（遷移後の状態, 即時報酬）が定まるということもでき，図 1.1 では遷移後の状態に対しての条件付き確率を矢印で表しているという見方もできる．本研究では基本的に後者の定式化を前提として考える．

以下，本研究では状態を小文字の x，行動を小文字の a，即時報酬を小文字の r で表す．

MDP 全体は，状態の集合 X，行動の集合 A，遷移関数 P の組として記述されるモデルである．エージェントが状態 $x \in X$ を観測し，そこで行動 $a \in A$ を選択すると，遷移関数に従って次の状態 x' と即時報酬 r が与えられる．

このように MDP を定義したが，MDP においてエージェントが最大化すべき目的関数は一般には累積報酬（収益）と呼ばれ，以下の式で与えられる．

$$r_{sum} = \sum_{t=0}^{\infty} \gamma^t r_t \tag{1.1}$$

ここで γ （$0 \leq \gamma \leq 1$） は割引率と呼ばれ，$\gamma < 1$ では将来受け取る報酬であればあるほど報酬の価値を下げていくような効果がある．$\gamma = 1$ では即時報酬の単純和となる．本研究では，割引率は最大化したい目的関数に含まれるパラメータとしてトップダウンで与えられるものとする．

本研究において MDP におけるプランニング課題とは，エージェントが毎時刻今後の収益を最大化する行動を選ぶ課題である．ただし，行動を起こした結果何が起こるか全くわからない状況ではプランニングを行いようがないので，エージェントは状態遷移関数のモデル，つまり状態遷移関数を保持していることを前提とする．

特に本研究では状態遷移が非決定的な場合を対象とする．この条件下では，エージェントは自分がある行動をとったときの次の状態と即時報酬は一意に定まらない．服をたたむ例でいえば，ロボットで力をかけたけれども実際に衣類がどう変形するかはやってみなれ

ばわからない，という場合である．ただし「どんな確率で」「どう変形するか」の分布に従った変形をエージェント側で試すことは可能であるという問題設定とする．

本稿は以下のように構成した．まず第 2 章において，離散 / 連続空間 MDP におけるプランニングの先行研究の概要をまとめた．次に第 3 章にて本研究で提案するプランニングアルゴリズムを提示した．提案するアルゴリズムの性能検証として，第 4 章にて小規模ないくつかの課題への適用を行い，第 5 章ではより高次元な空間における現実的な課題として氷上のスポーツであるカーリングのシミュレーションに適用した結果を記した．最後に第 6 章にて実験結果から示唆された提案手法の課題と今後より幅広く利用するために検証すべき点を議論した．

第2章　連続空間MDPにおけるプランニング

MDP においてエージェントが適した行動選択を行うためには，大まかには 2 つの点が重要である．一つ目は「事前に環境についての知識を習得しておくこと」であり二つ目は「行動決定時に深く思慮すること」である．

この 2 つがいずれも高い性能を持ち，両者を上手く組み合わせることができれば，囲碁 [2] や将棋で人間の最高峰のプレイヤに勝利するなど，特定の課題では優れたプランニングを行えることがこれまでに分かっている．

特定の目的に特化したシステムではこの 2 つの点のバランスが取れた最良の組み合わせを目指して開発を行うことができる．

一方でより汎用なシステムを目指す上ではそのレベルでの調整は難しく，扱う課題によっては行動選択に対して冗長な計算が行われることもやむを得ない．ただし，そういった場合であっても 2 つの側面それぞれの質が上がっていけば，解決できる課題の幅が広がり有用であると考えられる．そのため，どちらか一方の側面に着目した研究も十分に意義があると考えられる．本研究は後者の「行動決定時の思慮」に着目する．

前者の「事前の知識の習得」に着目した研究としては，課題に特化しないニューラルネットワークのモデルアーキテクチャによって，仮想の一人プレイ環境にて多数の課題を解決する研究（[3][4][5] 等）の発展が目覚ましい．これらの研究では後者の「行動決定時の思慮」を行わず事前の学習のみによって，複数の課題で人間を上回る行動選択能力を示している．文献 [3][4] は Arati2600 フレームワーク上での画面の RGB 値を入力として用いるものであり，現実の課題と比較すると低次元な課題を扱っているが，文献 [5] は強化学習手法を進歩させて 3D 物理シミュレータ MuJoCo[1] 上での制御タスクも効率よく学習できることを示している．また同じく end-to-end の行動選択モデルとして，MRP（マルコフ報酬過程; 行動選択のない MDP）の状態遷移や報酬の構造を抽象的に構築するモデルベースのアーキテクチャを使うことでより効率的な学習が行えるとする研究 [6] もあり，今後さらなる発展が期待される．

一方で後者の「行動決定時の思慮」についても研究が進められている．一つの試みとして，プランニング課題のルール（環境のモデル）を課題開始時に初めて受け取り，一切の事前知識の無い状態で探索的に最適行動を探していく取り組み [7] が挙げられる．このフレームワーク上において汎用のプランニングアルゴリズムとして成果を上げているのが，「モンテカルロ法」ベースの手法である．この手法は大まかには，環境モデルによって実際に状態を進めてみる手法であり，状態や行動の価値の事前知識が無い場合には特に適した手法であると考えられる．そのため本研究はこのモンテカルロ法ベースの手法にて連続状態空間 MDP の課題に適用することを検証する．

[1] http://www.mujoco.org/

2.1 モンテカルロ法によるプランニング

本研究が対象にしている連続状態空間でのプランニングを困難にする技術的課題はいくつかある．本節ではこの困難さをいくつか指摘し，モンテカルロ法と呼ばれる手法群がこれらの困難さに対処できることを述べる．

1 点目は，行動後の状態の価値についての知識が少ないことである．仮に状態の価値を高精度で表現するモデルがある場合には現在の状態で行動を選択し，即時報酬と次状態の価値を見るだけで行動価値を見積もることができるが，次状態の価値がわからず，即時報酬もあてにならない場合には質の高い行動選択はできない．

こういった場合には，環境モデルを使って先の先の状態をエミュレートすることで，よりよい価値推定を得て行動を選択することが有効である．何からの方策 (policy) を用いて状態を進めていくことで，先の報酬まで考慮した価値推定を行う手法は「モンテカルロ法」と呼ばれる．終端状態がそれなりに近くにあることが分かっている場合には終端状態まで進めてみることが考えられる．逆に終端がわからない場合にはステップ数で区切ったり，割引率がある場合には累積割引率が一定以下になるまでといった条件も用いられる．この，一方向に状態遷移を進めてみる一連の処理のことを以下「シミュレーション」と呼ぶ．

もう 1 点困難となりうるのは，状態遷移が確率的である場合である．この場合ある状態からある行動を取った場合の結果となる状態は 1 通りに定まらず，特に遷移先の状態が無限に多くのパターンを取る場合には全ての行動後の状態について調べ尽くすことは不可能である．

そのため限定した有限個数の状態のみサンプリングして期待値を推定することが求められる．この際，状態遷移の確率分布からランダムに状態遷移を発生させ，サンプリング数を増やすことで漸近的に期待値を計算できる．これも実際に環境モデルによって状態を遷移させることで達成できるため，モンテカルロ法によって対処できる．

2.2 モンテカルロプランニングにおける方策

方策によって行動を選択し，状態を進めていくモンテカルロ法において，推定される価値の妥当性に大きな影響を与えるのが方策の性質である．全ての可能な行動の中から一様ランダムに行動を選ぶ方策を用いることはできる（これは原始モンテカルロ法と呼ばれる）が，実際に将来ある状態が実現した場合にその状態で自分/味方エージェント/敵エージェントが選ぶ行動の分布が一様ランダムから大きく乖離しているのであれば，一様ランダムな方策を用いて計算した推定価値の妥当性は低い．

ここでは，各エージェントは出来る限り良い行動をとることを想定し，自分自身の期待報酬が大きい方策を良い方策と考える．このとき，よりよい価値推定を行うためのアプローチは大別して 2 つ考えられる．

1 つは，事前の知識によってより現実的な方策を得ておく手法である．

2 つ目は，事前知識の有無に関わらず，プランニング中に得た知識により方策を改善していく手法である．

モンテカルロ法において状態を入力とするシミュレーション用方策関数を別途定義した場合の方策の改善方法としては Adaptive Playout[8] が挙げられ，囲碁で成果を挙げてい

る．この手法ではシミュレーションを繰り返す中で，シミュレーション中の行動と報酬を記録し強化学習により方策を改善する．ただし文献 [8] においては使用された特徴量と方策の初期パラメータが元々囲碁に特化したものであったため，より一般の課題にて適切な方策関数を設計できるのかという別の問題がある．

その一方で，環境の状態空間から方策への関数を用意できない場合であっても，より直接的にシミュレーション中で訪れた各状態で良い行動を選べるようにする手法もある．こちらは「モンテカルロ木探索」[13] と呼ばれる手法群である．この手法はシミュレーション中の各状態での行動決定を「バンディット問題」として扱う手法であり，代表的な手法として UCT [14] がある．次節でバンディット問題について，次々節にてモンテカルロ木探索について詳しく述べる．

2.3 バンディット問題

バンディット問題 [9] は一般に以下のように定義される．報酬の分布がわからない複数のスロットマシーンがある状況を考える．プレイヤは複数のスロットマシーンのうちどれかを毎ターン選ぶことができ，選んだ台から報酬が与えられる．バンディット問題は時間ステップの長さが 1 の MDP でのプランニング課題と等価である．バンディット問題におけるスロットマシーン（選択肢）は MDP における行動と捉えることができる．

プレイヤは行動の選択を繰り返し，無限回の試行後に，または予め与えられた回数制限の中でより多くの報酬を得ることを目的とする．

このときプレイヤは 2 つの目標を両立しながら行動を選ばなければならない．

1. 探索　どの行動の報酬の平均値が高いのか知るために，色々な行動を選びたい
2. 活用　最終的な報酬を多くするために，報酬平均が大きいと予想されている行動を優先して選びたい

この 2 つは相反した目標であるが，どちらか一方のみを目指すと最終的な報酬の期待値は小さくなる．この事象は「探索と活用のジレンマ」と呼ばれる．そのため 2 つの目標をバランス良く満たす戦略を取ることが求められ，この戦略決定問題がバンディット問題における課題となる．

バンディット問題が抱えるジレンマは，モンテカルロプランニングにおけるシミュレーション割り振りの決定問題にもあてはまる．

1. 探索　どの行動が良いのかを知るために，色々な行動を選びたい
2. 活用　ルート状態なら，最も良い行動がどれかの統計的正確さの向上のため，遷移先の状態のシミュレーション回数を増やすことで妥当な値が返りやすくなるようにという理由から，良さそうな行動を優先的に選びたい．　ルート以外の状態なら，ルートの評価としてより妥当な値が返るように，良いと予想されている行動を優先的に選びたい．

そのため，シミュレーションにおけるルートとそれ以外のいずれの行動の選び方にもバンディット問題を応用することは自然である．

計算時間のコストを抑えながらバンディット問題を扱える手法としてはThompson Sampling[10]やUCB1,UCB1-tuned, UCB2[11] などがある．以下代表的な手法として，Thompson SamplingとUCB1について説明する．

Thompson Sampling[10] は確率的な手法で，各行動 a の報酬平均の推定分布 D を計算し，行動の集合 A から以下の式によって行動を決定する．

$$a_{TS} = \underset{a \in A}{\operatorname{argmax}}\, v_a \sim D_a \tag{2.1}$$

v_a は各行動について報酬分布 D_a からランダムに出力された値であり，それゆえにThompson Samplingは確率的なアルゴリズムである．試行が少ないうちは各行動のサンプルが少ないので，報酬平均推定の分散が大きく，行動選択がばらけるが，試行を重ねると分散が小さくなるため良いと推定した行動へ割り振るようになる．文献 [10] においては報酬が $0 \sim 1$ 区間にあると仮定して平均報酬の推定にベータ分布を用いているが，これを正規分布などの定義域の広い関数に置き換えて平均と標準誤差を計算することで報酬の値域が未知の場合にも適用することが可能である．

一方 UCB1[11] は決定的に腕を選択しながら探索と活用のジレンマを扱う手法である．行動 a のシミュレーション回数を N_a，行動 a で得た報酬の和を R_a とするときUCB1値 v_a の計算と行動の選択は

$$v_a = \frac{R_a}{N_a} + C_{UCB}\sqrt{\frac{2\ln\sum_{a' \in A} N_{a'}}{N_a}} \tag{2.2}$$

$$a_{UCB1} = \underset{a \in A}{\operatorname{argmax}}\,(v_a) \tag{2.3}$$

に従って行う．C_{UCB1} は正の定数であり，C_{UCB1} が大きいほどこれまでシミュレーションを割り振った回数が少ない行動が優先的に選択される．

UCB1の式は，報酬の平均に報酬についての情報が少ないほど大きいバイアスを足しており，楽観的な行動価値を計算していると捉えることができる．一般のバンディット問題においてもこのような手法が有効であることが知られており,「不確かな時は楽観的に」（optiminizm in the face of uncetrainty; OFU）の原則と呼ばれる [12].

2.4 離散空間におけるモンテカルロ木探索

モンテカルロ木探索 [13] はモンテカルロ法においてシミュレーションを繰り返し行う際に同一の状態に複数回到達することを利用して，それぞれの状態において各候補行動ごとにシミュレーションで得られた報酬を記録し，その情報を利用してシミュレーションの質を動的に向上させる．

モンテカルロ木探索の概念図を [13] より引用して図2.1に掲載した．

モンテカルロ木探索は，モンテカルロ法の一種であり，ルートの状態から終端状態の間のシミュレーションを繰り返すことを基本とするが，すでにある一定回数訪れた状態に対してはノードを作成し（図2.1の左から2番目; Expansion），各行動に対して，それぞれの行動を選んでシミュレーションを進めた場合の収益を記録しておく（図2.1の最も右側; Backpropagation）．そしてノードが作成された状態においてはバンディット問題によって行動を選ぶ（図2.1の最も左側; Selection）．訪れた全ての状態でノードを作成し

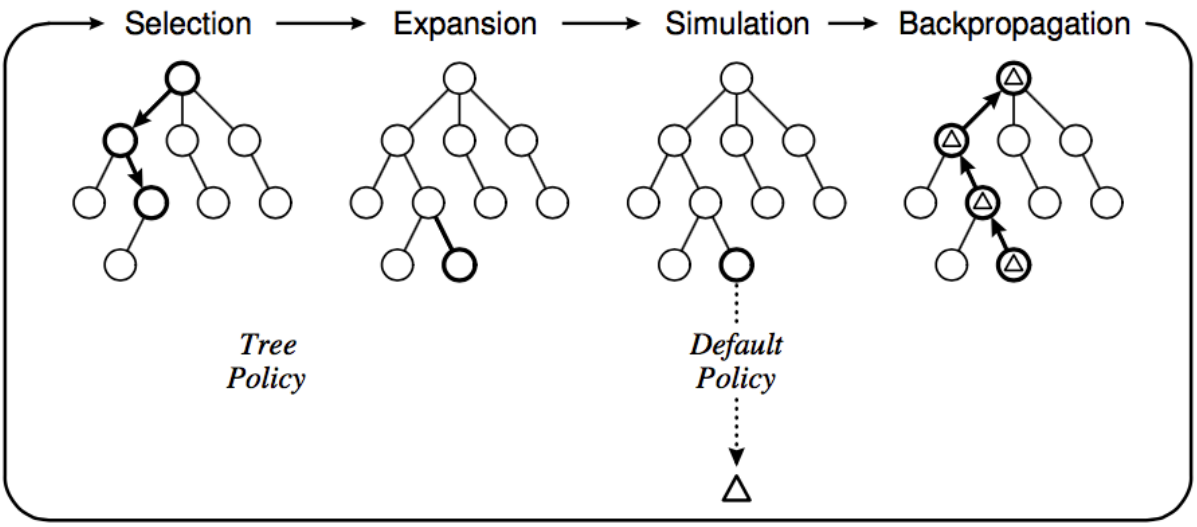

図 2.1: モンテカルロ木探索（[13] より引用）

バンディット問題を解く手法がある一方で，図 2.1 ではノードの展開に訪問回数などの制限を設けて，ノードが展開されなかった場合には通常のモンテカルロ法と同じシミュレーションを行っている（図 2.1 の右から 2 番目; Simulation）．

こういった手法の群全体がモンテカルロ木探索と呼ばれる．代表的な手法として，各状態で UCB1 を計算して行動選択を行う UCT[14] がある．遷移先状態が有限な場合の UCT の擬似コードはアルゴリズム 1 のようになる．ただし $transition(statex, actiona)$ を遷移関数，ここでは状態 x で行動 a を選択した場合の次の状態と即時報酬を返す関数とし，γ を割引率とする．なおノードの展開制限はなく訪れたノード全てに対して UCB1 を計算する場合としている．

Algorithm 1 UCT におけるシミュレーション

$N_{x,a}$ 状態 x 内の状態で行動 a を取った回数
$R_{x,a}$ 状態 x 内の状態で行動 a を取ったときの報酬の総和
N_x 状態 x 内への到達回数（$= \sum_{a \in A_x} N_{x,a}$）
function $simulationInUCT(x)$
if x が終端状態 **then**
  return x の終端報酬
end if
$a_{sim} \leftarrow \mathrm{argmax}_{a \in A_x} \left(\frac{R_{x,a}}{N_{x,a}} + C_{UCB} \sqrt{\frac{2 \ln N_x}{N_{x,a}}} \right)$
$x', r \leftarrow \mathrm{transition}(x, a_{sim})$
$r_{sim} \leftarrow r + \gamma \times \mathrm{simulationInUCT}(x')$
$N_x \leftarrow N_x + 1$, $N_{x,a_{sim}} \leftarrow N_{x,a_{sim}} + 1$, $R_{x,a_{sim}} \leftarrow R_{x,a_{sim}} + r_{sim}$

最終的にはルート状態 x_0 にてシミュレーションを行った回数 $N_{x_0,a}$ が最大の行動を選択する．

2.5 連続状態空間におけるバンディット問題

モンテカルロ法によって同じ状態に複数回到達する場合には，ベーシックなバンディット問題を各状態での行動選択で用いることで，漸近的に最適な行動が選ばれるようになる．一方で，連続空間における不確定な状態遷移により同じ状態に到達し得ない場合には，UCT 等をそのまま適用してもシミュレーション中の行動が良くなっていかないために質の高い行動決定はできない．

そのため，まず 1 ステップの MDP，かつ初期状態が確率的に与えられる環境で行動を選ぶ課題を考える．この問題設定においては，連続行動空間でのバンディット問題にガウシアンカーネルを利用した GP-UCB[15] にて状態の情報も利用した [16] や，行動が離散の場合に，過去の結果にカーネルを用いた重み付けを行って UCB1 値を近似的に計算する Weighted UCB (WUCB) [17] がある．

カーネル関数を $k((state, action), (state, action))$ として，WUCB は以下のように行動 a を選択する．行動が離散的な場合や行動空間内での価値の類似性を用いない場合には，「$a_1 \neq a_2$ ならば $k((s_1, a_1), (s_2, a_2)) = 0$」とすれば状態空間のみに対してのカーネルと考えることができる．結果の保存プール P には (状態 y, 行動 b, 報酬 r) のタプルを保存しておく．

$$n(x, a) = \sum_{y,b,r \in P} k((x, a), (y, b)) \tag{2.4}$$

$$r(x, a) = \sum_{y,b,r \in P} k((x, a), (y, b)) \times r \tag{2.5}$$

$$n(x) = \sum_{a \in A} n(x, a) \tag{2.6}$$

$$v_a = \frac{r(x, a)}{n(x, a)} + C_{UCB} \sqrt{\frac{2 \ln n(x)}{n(x, a)}} \tag{2.7}$$

$$a_{UCB1} = \underset{a \in A}{\operatorname{argmax}} (v_a) \tag{2.8}$$

長期的なプランニングにおいては，r は即時報酬ではなく収益を指す．この手法を用いることで，連続値のコンテキストが与えられた場合にも似たコンテキストでの結果の情報を効率的に用いることができる．

2.6 連続状態空間におけるモンテカルロ木探索

本節では連続空間 MDP かつ状態遷移が非決定的である場合，その中でも特に状態空間の連続性に対処したモンテカルロ木探索ベースでの行動決定手法の先行研究を挙げる．Double Progressive Widening と Weighted UCT という手法は第 4 章での実験において比較対象として用いるので，本節で詳細を述べる．

2.6.1 Progressive Widening

連続な状態空間を扱う手法の前提として，まず離散状態空間において行動の候補の数が多い場合や，行動集合が無限集合である場合に行動候補を少しずつ増加させていく Progressive Widening (PW) という手法について説明する．行動空間が連続である場合，行動集合 A は本来無限集合である．A が無限集合であれば全ての行動に対して UCB1 値を計算することができない．そのため Progressive Widening では行動を少しずつサンプリングして候補の数を段々と増やしていく．行動空間に PW を適用した UCT（UCT + PW）の擬似コードがアルゴリズム 2 である．ここで $newAction(statex)$ は状態 x から新たな候補行動を生成する関数であり，α は Progressive Widening において行動候補の数を増やす速さの減衰の度合いを決める定数である．

Algorithm 2 UCT + PW におけるシミュレーション

A_x 状態 x で選択される行動候補の集合（探索開始時点では全ての x に対して空集合）
function $simulationInUCT - PW(x)$
if x が終端状態 **then**
  return x の終端報酬
end if
if ${N_x}^{\alpha} \geq \#A_x$ **then**
  $A_x \leftarrow A_x \cup newAction(x)$
end if
$a_{sim} \leftarrow \mathrm{argmax}_{a \in A} \left(\frac{R_{x,a}}{N_{x,a}} + C_{UCB} \sqrt{\frac{2 \ln \sum_{a \in A} N_{x,a}}{N_{x,a}}} \right)$
$x', r \leftarrow \mathrm{transition}(x, a_{sim})$
$r_{sim} \leftarrow r + \gamma \times$ simulationInUCT-PW(x')
$N_x \leftarrow N_x + 1$, $N_{x,a_{sim}} \leftarrow N_{x,a_{sim}} + 1$, $R_{x,a_{sim}} \leftarrow R_{x,a_{sim}} + r_{sim}$

PW において行動候補数を制御する関数は一般に ${N_x}^{\alpha}$ のような形とされる．ただし状態への到達回数より行動の数がゆっくり増加しなければ，各行動に対するシミュレーション回数をだんだんと増やしていくことができないので，$0 < \alpha < 1$ を前提としている．

本研究では主に状態空間の連続性をどう扱うかに主眼を当てることとし，連続な行動空間の扱いについては PW によって候補を増やしていくことで適用可能であるとしそれ以上の内容は扱わない．

2.6.2 Double Progressive Widening

連続空間中で状態遷移に乱数が加えられる想定でのモンテカルロ木探索の実装の既存研究としては Double Progressive Widening（DPW）[18] がある．この手法は，本来連続的に分布する遷移後の状態を離散的にのみ生成し，その数を増やしていくことで，1 つ 1 つの遷移先の状態への到達回数を増やしつつ漸近的に全ての状態遷移を試す手法である．

DPW はこの PW を状態遷移に対しても適用する手法である．そのために Double Progressive Widening と名付けられている．DPW は UCT 以外のモンテカルロ木探索手法をベースとすることもできるので，UCT + DPW と表記するべきではあるが，DPW を提案した文献 [18] において UCT をベースとしたアルゴリズムを DPW として紹介している

ため，本稿でも以降 UCT + DPW を単に DPW と呼ぶ．DPW の擬似コードをアルゴリズム 3 に掲載した．[18] では即時報酬についての記述は無かったがアルゴリズム 3 では即時報酬がある場合を考慮に入れている．状態空間で PW を行うために，行動空間での PW のパラメータである α に対応して状態空間での PW の速さの減衰度合いを決定する定数 β の設定の必要がある．

Algorithm 3 DPW におけるシミュレーション

$M_{x,a}$ 状態 x で行動 a を取った際の (遷移先状態, 即時報酬) の組の集合（探索開始時点では全ての (x, a) に対して空集合）
$N_{x,a,(x',r)}$ 状態 x で行動 a を取った際の遷移先状態が x'，即時報酬が r であった回数
function $simulationInDPW(x)$
if x が終端状態 **then**
  return x の終端報酬
end if
if ${N_x}^{\alpha} \geq \#A_x$ **then**
  $A_x \leftarrow A_x \cup$ newAction(x)
end if
$a_{sim} \leftarrow \mathrm{argmax}_{a \in A} \left(\frac{R_{x,a}}{N_{x,a}} + C_{UCB} \sqrt{\frac{2 \ln \sum_{a \in A} N_{x,a}}{N_{x,a}}} \right)$
if ${N_{x,a}}^{\beta} \geq \#X'_{x,a}$ **then**
  $Mx, a \leftarrow M_{x,a} \cup$ transition(x, a_{sim})
end if
$x', r \leftarrow M_{x,a}$ から各 $\frac{N_{x,a,(x',r)}}{\sum_{x',r \in M_{x,a}} N_{x,a,(x',r)}}$ の確率で選ぶ
$r_{sim} \leftarrow r + \gamma \times$ simulationInDPW(x')
$N_x \leftarrow N_x + 1$, $N_{x,a_{sim}} \leftarrow N_{x,a_{sim}} + 1$, $R_{x,a_{sim}} \leftarrow R_{x,a_{sim}} + r_{sim}$, $N_{x,a,(x',r)} \leftarrow N_{x,a,(x',r)} + 1$

遷移先の状態を Progressive Widening で増やしていくことで，ナイーブな UCT では同じ状態に 2 度到達しないという困難を解決している．そのために，遷移先の状態を確率的に選ぶ際に「これまでに到達した回数が多い状態ほど多く」遷移する点が「数多くの遷移先状態を探索したい」「モンテカルロ木探索として機能させるために同じ状態に多数回到達したい」というジレンマへの対処になっており，仮にこれを逆にして「これまでに到達した回数が少ない状態ほど多く」などとすると性能が大きく落ちることを筆者は確認している．

DPW は可能な行動の集合が状態ごとに全く異なっていても適用可能であり，非常に汎用な手法であるといえる．各状態で可能な行動の集合が無限にあることを前提として設計されてはいるが，PW は有限の行動集合に対しても用いられるため行動空間が離散であってもそのまま適用が可能である．

一方で，現実的な問題においては近い状態で同じ行動が可能であり，なおかつそれらの間の行動価値が近いということも考えられ，特に大規模な課題に対してはそういった性質を利用しなければ現実的なシミュレーション回数内で有効なプランニングが行えない事態も考えられる．次に述べる WUCT は WUCB を用いることで,「近い状態で近い行動は価値が近い」という性質を利用し，より少ないシミュレーション回数で効率的に探索することを可能にする．

2.6.3 Weighted UCT

DPW は探索中に出てくる全ての状態を完全に別個に扱うため，状態ごとに可能な行動が異なる場合など幅広い問題設定に適応できることが特徴である．一方で，現実的な課題においては，似た状態では行動の価値が近いという性質を持っていることもある．この場合 DPW に比べてより効率よく環境の構造を利用することも自然な発想である．

そういったモチベーションによる先行研究として，WUCB を探索に利用した Weighted UCT (WUCT) [17] が挙げられる．まず WUCB では，状態空間中に距離カーネルを定義し，シミュレーション中に UCB1 値を計算する際に，他の状態での行動と報酬の記録を重み付けすることで，新奇の状態に対しても UCB1 値を計算することを可能にする．

WUCT におけるシミュレーションの擬似コードをアルゴリズム 4 に示す．DPW と異なり，他の状態での行動結果を使う以上状態によって可能な行動がある程度重なっている必要か，行動空間の連続性を考慮したカーネルを利用する必要がある．ただし本研究では状態空間の連続性のみに重きをおくため，アルゴリズム 4 は行動は有限でありすべての状態で A という同じ行動集合が候補となることを前提とした処理になっている．

Algorithm 4 WUCT におけるシミュレーション

P (状態, 行動, 収益) の結果を保存するプール
function $simulationInWUCT(x)$
if x が終端状態 **then**
  return x の終端報酬
end if
$n \leftarrow$ 行動から実数へのマップ 各要素の値を 0 で初期化
$r \leftarrow$ 行動から実数へのマップ 各要素の値を 0 で初期化
for $y, a, r' \in P$ **do**
  $n[a] \leftarrow n[a] + k((x,a),(y,a))$
  $r[a] \leftarrow r[a] + k((x,a),(y,a)) \times r'$
end for
$n_{num} \leftarrow \sum_{a \in A} N[a]$
$a_{sim} \leftarrow \mathrm{argmax}_{a \in A} \left(\frac{r[a]}{n[a]} + C_{UCB} \sqrt{\frac{2 \ln n_{sum}}{n[a]}} \right)$
$x', r \leftarrow \mathrm{transition}(x, a_{sim})$
$r_{sim} \leftarrow r + \gamma \times$ simulationInWUCT(x')
$P \leftarrow P \cup (x, a_{sim}, r_{sim})$

このアルゴリズムはカーネル関数を設定すれば幅広く利用できる．一方でこのアルゴリズムのネックはシミュレーション中の 1 状態あたりの行動選択の計算時間である．シミュレーション中に到達した状態数に対して線形の計算時間がかかってしまうので，シミュレーションを多数行うことは難しく，こちらも課題の規模が大きくなれば有効性が低くなると予想される．

第3章　連続空間を階層化して価値を近似するプランニング

本節では，前節で述べた先行研究の課題に対処するために提案する手法について述べる．

3.1　木構造による状態分割

連続空間を扱った先行研究から改善が望まれる点として以下の2点があった．DPW [18] においては空間の連続性の情報を使っていなかった．一方で WUCT [17] においては，シミュレーション中の行動決定の時間計算量が観測した状態数に対して線形に増加するため，シミュレーションを数多く行うことが難しいという問題点があった．

本研究ではこれらの困難に対処するため，以下の手法を連続空間でのプランニングに適用することを提案する．

まず1点目の空間の連続性の情報についてであるが，こちらは「近い」状態を区別せずに同じ状態集合として扱うことで，モンテカルロ木探索の枠組みにおいて似た状態での行動と報酬との関係もシミュレーションに生かす事ができる．ただし，「似た」状態としてどういった状態集合をとるかが問題で，あらかじめ状態集合を有限の個数に分割して，それぞれの状態集合内で最適な行動が1つに定まるようなことがわかっているならば，そもそも連続空間上の問題として扱う必要はない．

本研究では有限の状態集合として分割できない場合にも対処するため，漸近的に無限個に状態集合を分割していくことを提案する．このときすでに分割された状態集合をさらに分割していくような再帰的な分割を行えば，プログラム言語で容易に実装ができ，汎用性があると考えた．

状態空間が一次元の場合の再帰的な分割の例を図 3.1 に示した．

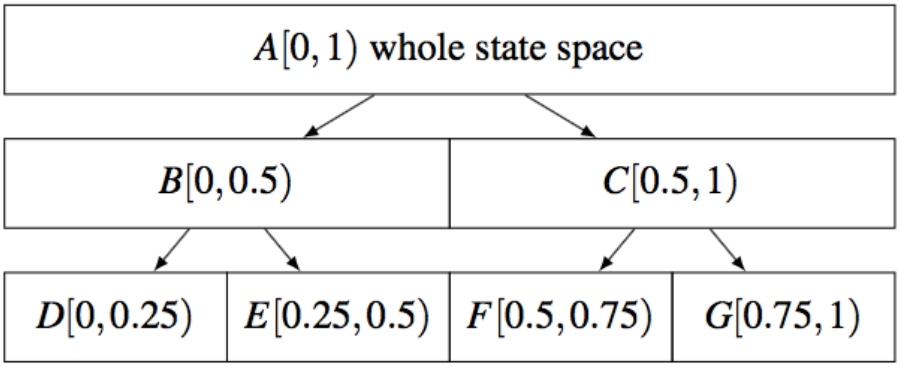

図 3.1: 一次元の状態木

図 3.1 では，$[0, 1)$ 区間で定義された状態空間を,「区間の中央で分割する」という最も簡単な分割規則で再帰的に二段階分割した様子を示している．

モンテカルロ木探索に適用する上では，この分割された状態集合を全てノードとして扱うことで，ある状態 x で選んだ行動とその報酬を，x を含む全ての状態集合ノードに記録する．

以下，この再帰的に分割された状態空間を木構造と見て,「状態木」と表記する．以降で提案手法について「ノード」と表記する場合には全て状態木のノード，つまり状態の集合を表し，一方で UCT などの手法におけるノード（状態を表す）は状態と記す．同じように「根ノード」は状態木の根ノード（状態木が表現する状態全体の集合）であり，探索開始時点の状態のことではない．後者は「ルート状態」と表記する．

3.2 状態空間の分割法

本研究において状態空間の分割方法は，3.1 で示したような，各次元の上界と下界の中心で分割する手法によって行う．このとき，状態空間が D 次元とすると 1 度の分割の際の分岐数は 2^D となり，D が大きいときには非常に横幅の広い木となるが，この手法をとる上での時間計算量，空間計算量が 2^D のオーダーとはならないことを後の節で述べる．また第 5 章では実際に状態空間が最大 30 次元の課題に対してこの分割手法を適用している．

3.3 状態木の作成単位

本研究においては，状態木は探索空間全体で 1 つには限定していない．実際に第 4 章での検証実験では行動ごとに状態木を作成する手法を使っており，この場合は状態木の各ノードは（状態集合, 行動）の組を表現している．第 5 章では行動では分けてはいないが，状態の性質が大きく変わるような離散的状態で分けて複数の状態木を作成している．

3.4 状態木のノードに保存するデータ

状態木の各ノードに保存するデータは，ノードが状態集合を表すか（状態集合, 行動）の組を表すかによって異なる．

前者の場合には，各行動 a に対してノードが表す状態集合 s に含まれる状態にて行動 a を選択した回数 $N_{s,a}$ とその際に得た累積報酬の和 $R_{s,a}$，さらにノードに到達した回数 N_s を保存しておく．

後者ではノードに行動 a は 1 つしか含まれないので，$N_{s,a}$ と $R_{s,a}$ の 2 つの値のみを保存する．

3.5 状態木の展開

連続的な状態空間を木構造で表現した場合，状態を細かく分割するためには無限の深さの木が必要である．しかしモンテカルロ木探索の特徴として,「多数回訪れた状態の状態

で良い行動を取ることが最優先であり，あまり訪れない状態での行動の改善は遅くても良い」という性質がある．

そのため何度も訪れる状態（ただしここでは連続な状態空間を考慮するので，完全に同一の状態でなくても，近い状態に多数回訪れる場合も同じように扱いたい）については，よりよい行動を選べるように，あまり訪れない状態に比較して状態を細かく見分けたいと考えるのが自然である．

そこで本研究での提案として，初期状態では根ノードのみ確保しておき，以降は状態木のあるノードが生成されてから，そのノードの表す状態集合に訪れた回数が一定以上となった場合にノードを展開することとした．以下この閾値を，状態木のノードの根ノードからの深さ（分割回数）を d として $th(d)$ と表記する．

ここで展開閾値を d の関数としているが，$th(d)$ を d に対して単調増加とし，木が深くなるにつれ展開が抑制されるようにすることで，全く同一の状態に訪れ続けた場合にも木の深さが線形に増加しないようにできる．これは事前実験にて実装して適用した結果効果があったため採用しており，また計算量の面でも優位性があることを後の節で述べる．

また，木を展開する際には，一度に子ノードを全て作成するのではなく，展開する際に訪れている状態を含むノードだけを生成することで，ノードの生成数を必要なものだけに留めることができる．この点は状態木の分岐数（あるノードの子ノードの数）が大きい場合には重要である．

3.6 状態木からの楽観的行動価値の推定

シミュレーション結果の報酬の逆伝播の際には，経験した状態を含む状態集合を葉ノードから根ノードまで辿り，全てのノードに行動の選択回数を 1 回分加算し，報酬和に観測された末端報酬を加える．

逆にシミュレーション中の任意の状態での着手の選択においては，状態木のうち目的の状態が含まれる葉ノードから根ノードに記録されている結果を合算して使う．このとき，より末端のノードの結果の寄与度を大きくする重み付けによって結果が合算されるように，状態木の根ノードからの分割回数 d のノードに記録されている結果に，d に対して単調増加な関数 $w(d)$ によって重み付けを行い，より細かく分割された情報集合に記録された結果を重視させる．

この $w(d)$ を用いて，状態 x における行動 a の仮の到達回数 $\tilde{n}(x,a)$ と仮の報酬和 $\tilde{r}(x,a)$ を以下のように求める．ただし S を状態木上で状態 x を範囲内に含んでいる状態集合 s の集合とし，$d(s)$ は状態木の根ノードから s に対応するノードまでの分割回数とする．$N_{s,a}$ と $R_{s,a}$ は状態木のノード上に記録されている行動 a の到達回数と報酬和を表す．

$$\tilde{n}(x,a) = \sum_{s \in S} w(d(s)) N_{s,a} \tag{3.1}$$

$$\tilde{r}(x,a) = \sum_{s \in S} w(d(s)) R_{s,a} \tag{3.2}$$

さらに状態 x への仮の到達回数は，状態 x で生成された行動全体の集合 A として

$$\tilde{n}(x) = \sum_{a \in A} \tilde{n}(x,a) \tag{3.3}$$

と求める．ここで，ある状態集合 s に含まれる状態 x_0，x_1 があり，状態 x_0 にて生成される行動集合と状態 x_1 で生成される行動集合が異なるという場合には，$\tilde{n}(x) = \sum_{s \in S} w(d(s))N_s$ とはできないことに注意されたい．

このように求めた $\tilde{n}(x,a)$ や $\tilde{r}(x,a)$ から状態 x における行動 a の UCB 値を計算することも可能であるが，状態木の展開が進むと重み付け $w(d)$ によって $\tilde{n}(x,a)$ は実際の到達回数より速いスピードで増加するという問題点がある．

この増加のスピードは状態 x 近傍への到達回数のみならず同じ状態木の離れた状態への到達回数によっても変化する．UCT における探索制御と異なり，各状態への到達の頻度や外乱の影響などドメインの性質によって左右されるため，定量的に扱うことは難しいかもしれないが，本研究では線形より増加スピードの遅い関数 $f(n)$ によって以下のように変換を掛けることで増加スピードを抑えることを提案する．

$$n(x) = f(\tilde{n}(x)) \tag{3.4}$$

この変換の際の割合を $\tilde{n}(x,a)$ と $\tilde{r}(x,a)$ にも掛けて UCB1 値の計算に用いる $n(x,a)$ と $r(x,a)$ を計算する．

$$n(x,a) = \tilde{n}(x,a)\frac{n(x)}{\tilde{n}(x)} \tag{3.5}$$

$$r(x,a) = \tilde{r}(x,a)\frac{n(x)}{\tilde{n}(x)} \tag{3.6}$$

$n(x)$，$n(x,a)$，$r(x,a)$ を用いて擬似 UCB1 値 [11] を計算し，この値が最大となる行動 a_{try} を選んでシミュレーションを進める．

$$a_{try} = \underset{a \in A}{\operatorname{argmax}} \left(\frac{r(x,a)}{n(x,a)} + C_{UCB}\sqrt{\frac{2\ln n(x)}{n(x,a)}} \right) \tag{3.7}$$

ただし C_{UCB} は UCB1 の設定パラメータとする．

3.7 状態木を利用したシミュレーション

前節で提案した，状態木を利用して楽観的な行動価値を推定する手法を利用したモンテカルロ木探索におけるシミュレーションの疑似コードをアルゴリズム 5 とアルゴリズム 6 に示した．なおアルゴリズム 5 では木は行動ごとに分けていない場合であり，一方アルゴリズム 6 では木は行動ごとに生成する場合である．

3.8 再帰的状態分割を利用したモンテカルロ木探索

アルゴリズム 5 またはアルゴリズム 6 を 1 回のシミュレーションとして，これを繰り返し行うことでモンテカルロベースのプランニング手法を提案する．この疑似コードを 7 に示した．

7 ではルート状態の行動のみを最終的に出力しているが，探索の終了後にも状態木を保持しておけば，連続して行動を選択することが可能になる．また行動によって状態を進め

Algorithm 5 状態木（行動ごとに木を分けない）を用いたシミュレーション

$N_{s,a}$ 状態集合 s 内の状態で行動 a を取った回数
$R_{s,a}$ 状態集合 s 内の状態で行動 a を取ったときの報酬の総和
N_s 状態集合 s に到達した回数
$nextState$(state, action) 状態遷移関数
w(depth) ノードの深さに応じた結果の重み付け関数
th(depth) 状態木のノード展開の閾値関数
f(frequency) 重み付け加算したシミュレーション回数の調整用関数
N_{prior} 行動回数の事前値, R_{prior} 報酬総和の事前値
x 状態
function $simulation(x)$
if x が終端状態 **then**
  return x の終端報酬
end if
$A \leftarrow$ 状態 x で可能な行動の集合
$\tilde{n} \leftarrow$ 長さ $\#A$ の配列 各要素の値を N_{prior} で初期化
$\tilde{r} \leftarrow$ 長さ $\#A$ の配列 各要素の値を R_{prior} で初期化
$s \leftarrow$ 状態 x を含む状態木の根ノード
$_, r_{sim} \leftarrow simulationOnStateTree(x, 0, s, A, \tilde{n}, \tilde{r})$
return r_{sim}

function $simulationOnStateTree(x, d, s, A, \tilde{n}, \tilde{r})$
$W \leftarrow w(d)$
for $i \leftarrow 0$ to $\#A - 1$ **do**
  $a \leftarrow A[i]$
  $\tilde{n}[i] \leftarrow \tilde{n}[i] + W \times N_{s,a}$, $\tilde{r}[i] \leftarrow \tilde{r}[i] + W \times R_{s,a}$
end for
if $N_s \geq th(d)$ かつ s の子ノードで状態 x を含むものが未作成 **then**
  状態 x を含む s の 子ノード s' を追加
end if
if s の子ノードで状態 x を含むものが未作成 **then**
  $\tilde{n}_{sum} \leftarrow$ summation of $\tilde{n}$
  $n_{sum} \leftarrow f(\tilde{n}_{sum})$
  $n \leftarrow$ 長さ $\#A$ の配列 i 番目の要素は $\tilde{n}[i] \times n_{sum}/\tilde{n}_{sum}$
  $r \leftarrow$ 長さ $\#A$ の配列 i 番目の要素は $\tilde{r}[i] \times n_{sum}/\tilde{n}_{sum}$
  $a_{sim} \leftarrow A[\mathrm{argmax}_{i \in 0..\#A-1} \left(\frac{r[i]}{n[i]} + C_{UCB} \sqrt{\frac{2 \ln n_{sum}}{n[i]}} \right)]$
  $x', r \leftarrow transition(x, a)$
  $r_{sim} \leftarrow r + \gamma \times simulation(x')$
else
  $s' \leftarrow$ 状態 x を含む s の子ノード
  $a_{sim}, r_{sim} \leftarrow simulationOnStateTree(x, d+1, s', A, \tilde{n}, \tilde{r})$
end if
$N_{s,a_{sim}} \leftarrow N_{s,a_{sim}} + 1$, $R_{s,a_{sim}} \leftarrow R_{s,a_{sim}} + r_{sim}$
$N_s \leftarrow N_s + 1$
return a_{sim}, r_{sim}

Algorithm 6 状態木（行動ごとに別の木）を用いたシミュレーション

$N_{s,a}$ 状態集合 s 内の状態で行動 a を取った回数
$R_{s,a}$ 状態集合 s 内の状態で行動 a を取ったときの報酬の総和
$nextState$(state, action) 状態遷移関数
w(depth) ノードの深さに応じた結果の重み付け関数
th(depth) 状態木のノード展開の閾値関数
f(frequency) 重み付け加算したシミュレーション回数の調整用関数
N_{prior} 行動回数の事前値, R_{prior} 報酬総和の事前値
x 状態
function $simulation(x)$
if x が終端状態 **then**
  return x の終端報酬
end if
$A \leftarrow$ 状態 x で可能な行動の集合
$\tilde{n} \leftarrow$ 長さ $\#A$ の配列 各要素の値を N_{prior} で初期化
$\tilde{r} \leftarrow$ 長さ $\#A$ の配列 各要素の値を R_{prior} で初期化
$s \leftarrow$ 状態 x を含み行動 a に対応する状態木の根ノード
$_, r_{sim} \leftarrow simulationOnStateTree(x, 0, s, A, \tilde{n}, \tilde{r})$
return r_{sim}

function $simulationOnStateTree(x, d, s, A, \tilde{n}, \tilde{r})$
$W \leftarrow w(d)$
for $i \leftarrow 0$ to $\#A - 1$ **do**
  $a \leftarrow A[i]$
  $\tilde{n}[i] \leftarrow \tilde{n}[i] + W \times N_{s,a}$, $\tilde{r}[i] \leftarrow \tilde{r}[i] + W \times R_{s,a}$
end for
if $N_{s,a} \geq th(d)$ かつ s の子ノードで状態 x を含むものが未作成 **then**
  状態 x を含む s の 子ノード s' を追加
end if
if s の子ノードで状態 x を含むものが未作成 **then**
  $\tilde{n}_{sum} \leftarrow$ summation of $\tilde{n}$
  $n_{sum} \leftarrow f(\tilde{n}_{sum})$
  $n \leftarrow$ 長さ $\#A$ の配列 i 番目の要素は $\tilde{n}[i] \times n_{sum}/\tilde{n}_{sum}$
  $r \leftarrow$ 長さ $\#A$ の配列 i 番目の要素は $\tilde{r}[i] \times n_{sum}/\tilde{n}_{sum}$
  $a_{sim} \leftarrow A[\mathrm{argmax}_{i \in 0..\#A-1} \left(\frac{r[i]}{n[i]} + C_{UCB} \sqrt{\frac{2 \ln n_{sum}}{n[i]}} \right)]$
  $x', r \leftarrow transition(x, a)$
  $r_{sim} \leftarrow r + \gamma \times simulation(x')$
else
  $s' \leftarrow$ 状態 x を含む s の子ノード
  $a_{sim}, r_{sim} \leftarrow simulationOnStateTree(x, d+1, s', A, \tilde{n}, \tilde{r})$
end if
$N_{s,a_{sim}} \leftarrow N_{s,a_{sim}} + 1$, $R_{s,a_{sim}} \leftarrow R_{s,a_{sim}} + r_{sim}$
return a_{sim}, r_{sim}

Algorithm 7 状態木を利用したモンテカルロ木探索

rootAction(state) ルート状態での行動を選択する関数 (ex. UCB1)
x_{root} ルートの状態
while 制限時間まで **do**
  $a_{root} \leftarrow rootAction(x_{root})$
  $x'_{root}, r_{root} \leftarrow transition(x_{root}, a_{root})$
  $r_{sim} \leftarrow r_{root} + \gamma \times simulation(x'_{root})$
  r_{sim} をルートでの a_{root} の結果として記録
end while
return ルートの行動として最高評価なもの

た後に再度探索を行う場合にも，生成された状態木を利用することで 1 からよりも効率良く探索を行える可能性もある．これらの点は第 4 章での実験で検証する．

3.9 計算量

3.9.1 時間計算量

以上のように状態木という構造を定義し，モンテカルロ木探索ベースのプランニングアルゴリズムを提案したが，このアルゴリズムの時間計算量と空間計算量について本節で議論する．

既存研究である WUCB にを利用した WUCT においては，シミュレーション中の行動選択の時間計算量は，重み付け加算を行う過去の（状態, 行動, 収益）数に比例し，これは先行研究の提案をナイーブに実装しシミュレーション中のこれまで全ての結果を参照した場合にはシミュレーション中の行動決定の回数を n，行動候補数を $\#A$ として $O(\#An)$ の時間計算量を必要とする．この場合，シミュレーションを n 回行う計算量は $O(n^2)$ となり，シミュレーションを多数回行うことの時間的コストは大きく探索の時間効率は段々と低下していく可能性がある．

対して提案手法ではシミュレーション中の行動選択の時間計算量は $O(\#A \times$状態木の深さ$)$ で抑えられ，以下のように考えることができる．

1. ノード展開閾値 $th(d)$ が定数のとき

このとき最大の木の深さの期待値は $O(n)$ である．到達しうる状態の分布の確率密度において，ある状態の周囲でどれだけ狭い範囲（ただし空集合でない）をとってもその範囲の状態が出現する確率が 0 でないような状態が存在するとき，最大の木の深さの期待値はそういった状態における挙動によって決まり，到達回数と線形に増える．よってこの場合は，MDP の構造によっては WUCT と同じオーダーの時間計算量になる．逆に状態の出現確率の分布にそういった特異点が出てこない環境であれば，線形よりは小さいオーダーで抑えることができる．

2. ノード展開閾値 $th(d)$ が線形オーダーの単調増加関数のとき

展開閾値をノードの深さ d の単調増加をする場合の一例として $th(d)$ が d の線形である場合を考える．この場合にも最大の木の深さの期待値の計算において支配的なのは上記のような特異点的な状態での挙動である．この場合の深さ d での展開速さは $\frac{1}{th(d)}$ のオーダー

であるため，木の深さはこれを n の線形だけ加算したオーダーであり，$O(logn)$ と同等である．$th(d)$ が線形よりも速く大きくなる場合にはさらに小さいオーダーになるが，その分木の展開が遅いことで状態を細かく見分けられなくなるので，環境に応じて調整すべきバラメータであると考えている．

3.9.2 空間計算量

一方で空間計算量は，WUCT と同じく最大で $O(n)$ 程度で抑えられる．時間計算量の議論では木の深さが問題となったが，空間計算量はノードの数によって測ることができる．

簡単のため各ノードの展開閾値が出現 1 回とする．このときシミュレーション中の全ての状態にてノードが展開されるため，ノードの数は線形な増加と言える．ノードの展開閾値が深さに依存しない定数のときはこの線形の増加より悪くなることはなく，ノードの展開閾値が深さに対して単調増加の場合にはノード数の増加スピードは遅くなるので，少なくとも線形で抑えることはできる．よってノード数は $O(n)$ で抑えることが可能であるといえる．

一方で，各ノードが保持しておくべきデータは 3.4 において述べたように，木が行動ごとに別の場合は $O(1)$，そうでない場合は $O(\#A)$ となる．木構造の実装の際に子ノードへのポインタを持つ実装も考えられ，その場合は子ノードの数に比例する空間計算量が必要であるが，本研究では状態の分割は各次元の上界と下界の半分で分けるという手法を取っているため，分割を何回行うか，分割されたどの集合に属したかによってハッシュ値を計算して，ハッシュマップ上に保存したノードにアクセスすることで，D 次元の空間，分割 d 回においてハッシュ値の計算に $O(Dd)$，アクセスに $O(1)$ の計算時間で済ませることができる．さらに，状態木の子ノードを辿っていく際にハッシュ値を差分計算することで，シミュレーション中の行動決定において状態木へのアクセスの計算時間量は全体でも $O(Dd)$ で行うことが可能である．

以上の時間計算量，空間計算量の見積もりにより，本研究の提案手法は先行研究の $WUCT$ より効率よくプランニングを行うことができることが期待される．次章から提案手法をベンチマークの課題に適用して検証する．

第4章　提案手法の簡単なプランニング問題への適用

本章では，低次元の連続空間内におけるプランニング課題として既存研究 [18][19] 等で比較実験に用いられている課題を利用し，提案手法の性能を検証する．なお既存研究 [18][19] の課題における行動空間は連続として与えられている．本研究では行動空間は離散，連続の双方を想定しているため，連続の場合は課題をそのまま，離散の場合は独自に離散化を設定した．

本章では 2 種類の課題にて提案手法の性能を検証した．いずれの課題においても割引率 $\gamma = 1$ であり，収益は即時報酬の和であることを前提とする．

4.1　Trap Problem への適用

Trap Problem は [18] において検証に利用された課題であり，連続した移動行動を決める短期のプランニング課題である．

4.1.1　課題設定

課題は一次元の空間上で行われる．大部分を既存研究 [18] に倣い以下のように定めた．ただし本研究では行動空間を離散化した課題を扱う．

状態 x を一次元の状態空間とし，エージェントは空間内で移動する．初期状態 $x_0 = 0$ とし，エージェントは時刻は t で距離 a_t 移動する．本研究では 8 個の行動が可能とした．具体的には整数 $0 \leq i < 8$ に対して $a^i = \frac{i+0.5}{8}$ とした．これは既存研究 [18] にて移動幅を $[0, 1]$ 区間から自由に選べる設定であったことによる．

ただし，行動の際に $[0, R)$ 区間の一様分布から生成されるノイズ d だけエージェントは強制的に位置を移動させられる．遷移後の状態 x' は $x' = x + a + d$ となる．報酬 r は状態遷移後の x の値に応じて与えられ，$x < 1$ で 0.7，$x \geq 1.7$ で 1 である．いずれも正の報酬とする．$l \leq x < h$ では報酬は 0 である．時刻 t は初期で $t = 0$ であり，$t = 2$ で終了とする．逆にそれ以外の終了条件はない．

ノイズのパラメータは，
$R = 0.01$ または $R = 0.1$
とした．

先行研究 [18] ではノイズの分布の幅 R は $R = 0.01$ のみであったが，本研究では $R = 0.1$ の条件も追加している．先行研究のノイズの幅ではノイズが無いものとして考えてもほとんど影響がないため，より連続空間探索のベンチマークとして有効になるようノイズの幅が大きい条件を加えた．

この課題では状態ごとに独立に扱う UCT では最適行動を選べないことが既存研究における検証 [18] で示されている．

この課題では最大の収益である 1.7 を獲得するためには，$t = 2$ で $x \geq 1.7$ となっておく必要があるため，$t = 0, t = 1$ の両方で a として大きめの値を選ぶべきであるが，$1 \leq x < 1.7$ で報酬 0 の範囲があるため，$t = 1$ の行動がランダムに選ばれるとすると $t = 0$ では初期状態からあまり動かずに収益 1.4 を狙うほうが安全である．そのためナイーブな UCT では収益 1.4 を狙う行動が選択されやすい．

このように，収益 1.4 を狙へ向かう局所解の方が容易に到達できるために，この局所解の「trap」を回避できるかによって探索能力の有無を検証する課題である．

4.1.2 実験条件

実験においては既存研究である DPW，WUCT と提案手法でそれぞれ条件を定めてプランニングを行った．この課題においては最善の行動が明らかであるため，行動空間は離散の場合のみに限定した．DPW は行動空間に対しても PW を行うことを前提に提案された手法のため，8 個の行動候補についても PW を適用した．

既存手法の DPW は以下のような設定で行った．

$C_{UCB} = 1$

$\alpha = 0.3$

$\beta = 0.25$

既存手法の WUCT は以下のような設定で行った．

$C_{UCB} = 1$

状態間のカーネルは以下の手法で計算した．まず，アルゴリズム 4 において (状態, 行動, 収益) の結果を保存するプール P を配列形式とし，シミュレーションの実施順に先頭から並べて保存しておく．同じシミュレーション内の結果の保存順は問わないとする．このとき P 内で i 番目の結果タプルにおける状態との間のカーネル関数の値を以下のように再帰的に定義する．ただしカーネルを計算する対象の状態 x, 行動 a，P 内で i 番目に記録されている結果の状態 x_i, 行動 a_i とする．

$$\sigma_{i,a} = 0.5 \times (1 + \sum_{j \in 0..i-1} k((x, a), (x_j, a_j)))^{-\frac{1}{2}} \tag{4.1}$$

$$k((x, a), (x_i, a_i)) = \frac{e^{-\frac{(x-x_i)^2}{2{\sigma_{i,a}}^2}}}{2\sqrt{\pi}\sigma_{i,a}} \tag{4.2}$$

$\sigma_{i,a}$ の計算における 0.5 は課題の状態空間の幅に対応して定めた値である．

この計算は行動の候補数が有限で固定の場合には O(#P) で行える．またカーネルの和は全ての過去の結果に対しての加算が終了すればアルゴリズム 4 中の $n[a]$ となるので，固定のカーネルを使うことに対する計算コストの増加は逐次計算を行う限りごくわずかである．このカーネルは，より過去のシミュレーションの結果に対しては幅が広く，新しい

シミュレーションの結果に対しては幅が狭くなる性質を持っている.「シミュレーションを重ねる上で段々と妥当な収益の結果が返ってくる点」と「シミュレーションを重ねると到達回数が増えるので近傍の状態の結果に相対的に大きな重み付けができる点」の双方を仮定して効率を上げることを目的としている.

提案手法の設定は以下とした.

$C_{UCB} = 1$

$th(d) = 8 \times 1.5^d$

$w(d) = (d+1)^2$

$f(n_{sum}) = \tilde{n}_{sum}^{\frac{2}{3}}$

状態木の根ノードの表す状態集合: $[-2R, 2+2R]$

またシミュレーションにおいて，状態木は行動ごとに生成するアルゴリズム 6 とした．これは行動候補数が可変の際に行動結果を各ノードで管理する必要がなくなる点で実装がシンプルになるメリットがある．事前の検証において，状態全体で木構造を作る手法に比べどちらの手法が優れているとは判断できなかった．これにより，状態木のノードの展開条件は状態集合 s への到達回数から状態集合と行動の対 (s, a) への到達回数に変更されている．

WUCT と提案手法においてはルート状態は 1 つに確定しているため,「似た」状態の結果を使用せずとも推定行動価値の統計的な精度を高めていくことが可能である。そのためルート状態（$t = 0$）に対しては独立に結果を記録し，シミュレーション中 $t = 0$ での行動選択においてはこの結果の記録から UCB1 を適用して行動を選択した，この際 UCB1 のパラメータは $C_{UCB} = 1$ とした．

実験に使用した計算機は Intel Corei7-4790K 4.00 GHz である．

4.1.3 実験計画

実験は各時刻ごとにシミュレーションの回数または計算時間を固定して行った．計算時間固定の場合には制限時間が来た時に行っているシミュレーションは終了し結果を反映させるまで継続させた．そのため，特に 1 回のシミュレーション時間の長い WUCT では制限時間を大幅に超過することが多かったが，結果に実際の使用時間を追記し，グラフ作成の際に実際の使用時間の平均の位置にプロットすることで対処した．

実験は $R = 0.01$，$R = 1$ のそれぞれで以下の条件で行った．シミュレーション回数の制限は 1, 10, 100, 1000, 10000 回の 5 通りとし，各条件 10000 試行を行った．計算時間の制限は 1, 10, 100, 1000ms の 4 通りで行った．1ms〜100ms では各条件 10000 試行，1000ms では各条件 1000 試行を行った．

4.1.4 実験結果

図 4.1 がシミュレーション固定条件における平均収益，図 4.2 が条件における平均収益である．それぞれ $R = 0.01$ 条件と $R = 0.1$ 条件での結果のプロットに分かれている．

図 4.1 より，WUCT と提案手法が DPW に比べて，全体として少ないシミュレーション回数でも高報酬を獲得している．全ての手法がシミュレーション 100 回では収益 1.4 の局所解あたりの収益を記録しているが，さらにシミュレーションを増やすことでより良い行動を発見できており，特に WUCT はシミュレーション 1000 回でほぼ満点の結果を出している．提案手法がそれに続いている．

図 4.2 でも同じ傾向であり，WUCT が各行動決定機会に 10ms 程度の思考時間で正解をほとんど選べているのに対し提案手法は 1000ms 程度で追いついている．DPW は短時間での性能がいずれの条件でも高かった．一方で収益の増加度合いが緩やかである．

$R = 0.01$ の結果と $R = 0.1$ の結果の比較では，WUCT は結果にあまり影響がなく，提案手法と DPW はノイズ幅が大きくなることで収益が相対的に低い値になっている．収益の下がり方は提案手法の方が DPW より小さかった．

4.2 Treasure Hunt Problem への適用

Treasure Hunt Problem は [19] において検証に利用された課題である．

4.2.1 課題設定

[19] から引用した課題図を図 4.3 に掲載した．

4.3 の図のうち左側が課題の全体図である．エージェント二次元空間 (x, y) 上に位置し，左下のスタート地点（「Start」）を $(0, 0)$ とする．エージェントの目的は右上に「Treasure」で示されたゴール範囲に到達することであり，迷路課題の一種である．位置空間は一辺の長さ D とする正方形内で定義され，$0 \leq x \leq 0 \wedge 0 \leq y \leq D$ である．ゴールゾーンは一片 G とし，$D - G \leq x \leq D \wedge D - G \leq y \leq D$ の範囲とする．エージェントがこの範囲に到達できれば報酬 $R_{treasure}$ を獲得しエピソードが終了する．中央に一辺 h の正方形穴が全体と平行に空いており，$\frac{D-H}{2} \leq x \leq \frac{D+H}{2} \wedge \frac{D-H}{2} \leq y \leq \frac{D+H}{2}$ の範囲に入ると負の報酬 R_{hole} を得てエピソードが終了する．エージェントは毎時刻，自分の位置 (x, y) を観測し，A の距離だけ移動する．エージェントは移動方向 θ を決定できる．移動しない，A より短い距離だけ移動するという選択肢は無い．エージェントの移動時にはノイズが加算される．ノイズは図 4.3 の右側に示されるように一辺 ϵ の正方形の中から一様分布で選ばれる．またゴールにへの早い到達を目指すことの促進として，状態遷移毎に負の報酬 R_{time} が加算され，時刻 t が $t = T$ でエピソードは強制的に終了する．

範囲外に出る行動が選択可能かどうか，ノイズを加えた移動の後に範囲外にいた場合はどうなるかについて文献 [19] には記されていない．本研究では「範囲外に出る行動は選択可能」「移動後に範囲外にいた場合は強制的に範囲の端に戻される」とした．具体的には，時刻 t にてエージェントが選択した行動とノイズの加算後のエージェントの位置が (x', y') のとき $x_{t+1} = max(0, min(x', D))$，$y_{t+1} = max(0, min(y', D))$ とした．そのため，行動が完全ランダムとすれば範囲の端への存在確率は範囲の内側より高い．また

$(0,0),(0,D),(D,0)$ の 3 つの状態には複数回到達する可能性があり，3.9.1 にて「特異点」と述べた状態が実際に存在する課題となっている．

各パラメータは以下のように設定した．

$D = 5$ または $D = 15$

$G = 1$

$h = D/2$

$A = 1$

$\epsilon = 1$

$R_{treasure} = 1, R_{hole} = -0.5, R_{time} = -0.001$

$T = D * 10$

既存研究 [19] においては穴がある場合には $D = 5$ の条件のみ（他の条件は上記と同じ）検証を行っていたが，本研究ではより長期的で難易度の高い課題として $D = 15$ の条件も加えた．

この課題では各時刻でマイナスされる報酬に対してゴールの報酬の方が大きいので，収益の比較の上では基本的には遅くてもゴールできたかどうかの比重が大きい．

4.2.2 実験条件

この課題では制限時間が十分に長いと考え，状態の観測時に時刻は無視してエージェントの座標のみで状態を見分けた．そのため WUCT と提案手法ではシミュレーション内の他の時刻の結果も利用したといえる．

行動空間が連続のため，DPW は提案通りの手法とし，WUCT と提案手法も行動空間全体に対して PW を行った．ただし DPW と異なり，WUCT と提案手法では状態空間全体に対しての行動候補を PW で増やしていく手法をとった．このときの PW のパラメータはともに $\alpha = 0.2$ とした．

DPW の C_{UCB} は，Treasure Hunt Problem のにおける報酬の分散が Trap Problem の場合より小さいとして $C = 0.3$ に変更した．一方 WUCT と提案手法の C_{UCB} は 1 のままとした．

また WUCT が用いる距離カーネルは，Trap Problem に対する設定と同じくガウスカーネルベースであり，Trap Problem に適用した式 4.2 において $(x - x_i)^2$ であったところを $(x - x_i)^2 + (y - y_i)^2$ と 2 次元上での距離を表すように変更した．また課題の幅が D に依

存するため，4.1 式における $\sigma_{i,a}$ の計算にて Trap Problem で 0.5 であった箇所を $\frac{D}{2}$ としている．

提案手法における状態木の根ノードの表す状態の範囲は $([0, D], [0, D])$ と状態空間全体とした．

これ以外に各手法の実装やパラメータ設定において Trap Problem に対する実験の条件と大きな違いはないが，WUCT は特に以前の探索結果を保存する方式の場合，エピソードが長くなると計算時間が非常に長くなる．各時刻でシミュレーション 10 回程度でもかなりの長時間がかかることが事前にわかったため，WUCT については時間固定のみとした．さらに，WUCT と提案手法ではプランニング後に探索結果を保存して 1 ステップ後のプランニングに利用する場合と保存しない場合の双方の条件で実験を行った．

以降では，探索結果を保存する場合を「記憶あり」の WUCT/提案手法，保存しない場合を「記憶なし」の WUCT/提案手法と呼ぶ．記憶ありの場合には以前のステップでのプランニングに利用したデータを再度用いるので，1 回目のシミュレーションからある程度の質の高さが期待されるが，その分シミュレーションにかかる計算時間は段々と長くなるというジレンマがある．

また新たに追加した条件として，初期状態におけるプランニングのみでシミュレーションを行い，以降の状態においては初期状態におけるプランニングの過程で生成されたデータのみを利用可能とする条件も加えた。この条件においては提案手法は状態木の

その結果のみでエピソード末端までの行動決定を行う条件を追加した．近い状態の結果を利用しない DPW はこの方式では $t = 1$ 以降ランダムな行動をとらざるを得ないため，WUCT と提案手法でのみこの条件での実験を行った．

このとき初期状態以外で最終的に行動決定を行う基準は，到達回数（カーネルや状態木により擬似的に計算されたもの）ではなく平均収益によるものとした．これは，WUCT や提案手法ではカーネルを足し合わせたり，その近似的処理を行うため状態空間の中央近くでは状態空間の端よりも擬似的な到達回数が大きい値になりやすいという問題があり，状態空間の中央近くで選択されることの多い行動に正のバイアスがかかる可能性があり，到達回数では公平な比較が行えないと考えたためである．

4.2.3 実験計画

全ての状態で探索する条件では $D = 5$ と $D = 15$ の両方で実験を行った．双方でシミュレーション回数の制限は 1, 10, 100, 1000, 10000 回の 5 通りを用意し，1 回〜100 回では各条件 10000 試行，1000 回と 10000 回では各条件 1000 試行を行った．計算時間の制限は 1, 10, 100, 1000ms の 4 通りで行った．1ms, 10ms では各条件 10000 試行，100ms と 1000ms では各条件 1000 試行を行った．

初期状態での探索のみの条件では $D = 15$ のみで行った．シミュレーション回数の制限は 1, 10, 100, 1000, 10000 回の 5 通りを用意し，1 回〜100 回では各条件 10000 試行，1000 回と 10000 回では各条件 1000 試行を行った．計算時間の制限は 1, 10, 100, 1000ms の 4 通りで行った．1ms〜100ms では各条件 10000 試行，1000ms では各条件 1000 試行を行った．ただし WUCT は初期状態だけの探索でも時間がかかったので，シミュレーション回数 1000 回以上での実験は行えなかった．

4.2.4 実験結果

全ての状態で探索する条件において図 4.4 がシミュレーション回数を固定した場合の平均収益のプロット図 4.5 が一度の行動決定機会ごとに計算時間を固定した場合の時間-平均収益のプロットである．それぞれ $D = 5$ 条件と $D = 15$ 条件のプロットに分かれている．

一方，初期状態でのみ探索を行い以降は初期状態での探索結果のみを使う条件において図 4.6 がシミュレーション回数を固定した場合のシミュレーション回数-平均収益のプロット図 4.7 が一度の行動決定機会ごとに計算時間を固定した場合の時間-平均収益のプロットである．

図 4.4 において，記憶なしの提案手法いずれの条件でもシミュレーション回数の増加とともに平均収益を上げており，$D = 5$ と $D = 15$ のいずれにおいても最終的にほとんどの試行でゴールにたどり着いている様子が見られる．記憶ありの提案手法は少ないシミュレーション回数では高い収益を記録している．とくにシミュレーション 1 回であっても一定割合でゴールにたどり着いているのは他の 2 手法とは異なる．しかし $D = 15$ 条件においてはシミュレーション回数が増えた結果収益が下がる事態となっており，過去の記憶が必ずしも良い結果につながらない結果となった．DPW は $D = 5$ では提案手法には及ばないもののある程度はゴールへたどり着けている．一方 $D = 15$ 条件においては収益はほとんど伸びていない．

図 4.5 の時間固定の場合の結果も提案手法と DPW については同じ傾向がある．記憶なしの提案手法は記憶ありの提案手法より単位時間あたりのシミュレーション回数が多い分，1ms の短時間条件での記憶ありと同程度の結果であり，10ms では追い越している．一方 WUCT については記憶なしの方が $D = 5$ でも $D = 15$ でも良い結果となった．グラフでは $D = 5$ の短時間では DPW より下であるが，時間が伸びた場合には提案手法と DPW の間程度の性能であった．

また，特に $D = 15$ の 1ms 条件において記憶ありの WUCT は平均として制限時間を大幅に超過していることが見て取れる．各時刻ステップにおいて超過する分は最大シミュレーション 1 回分であるが，その計算時間が制限時間の 1ms より 10 倍以上長くかかりうるという結果となった．

最後に，初期状態でのみ思考可能とする条件の結果について述べる．図 4.6 にて，WUCT はシミュレーション 100 回までしか行えなかったが，シミュレーション 10 回と 100 回において提案手法の方がより高い収益が得られており，その後も収益は向上を続けている．図 4.7 時間あたりの制限であっても同じ傾向である．また図 4.4 と図 4.5 の $D = 15$ の結果と比較すると，提案手法と WUCT は初期状態でシミュレーション 1 回を行うだけでも DPW のシミュレーション 10000 回や思考時間 1000ms より良い結果を得られていることもわかる．

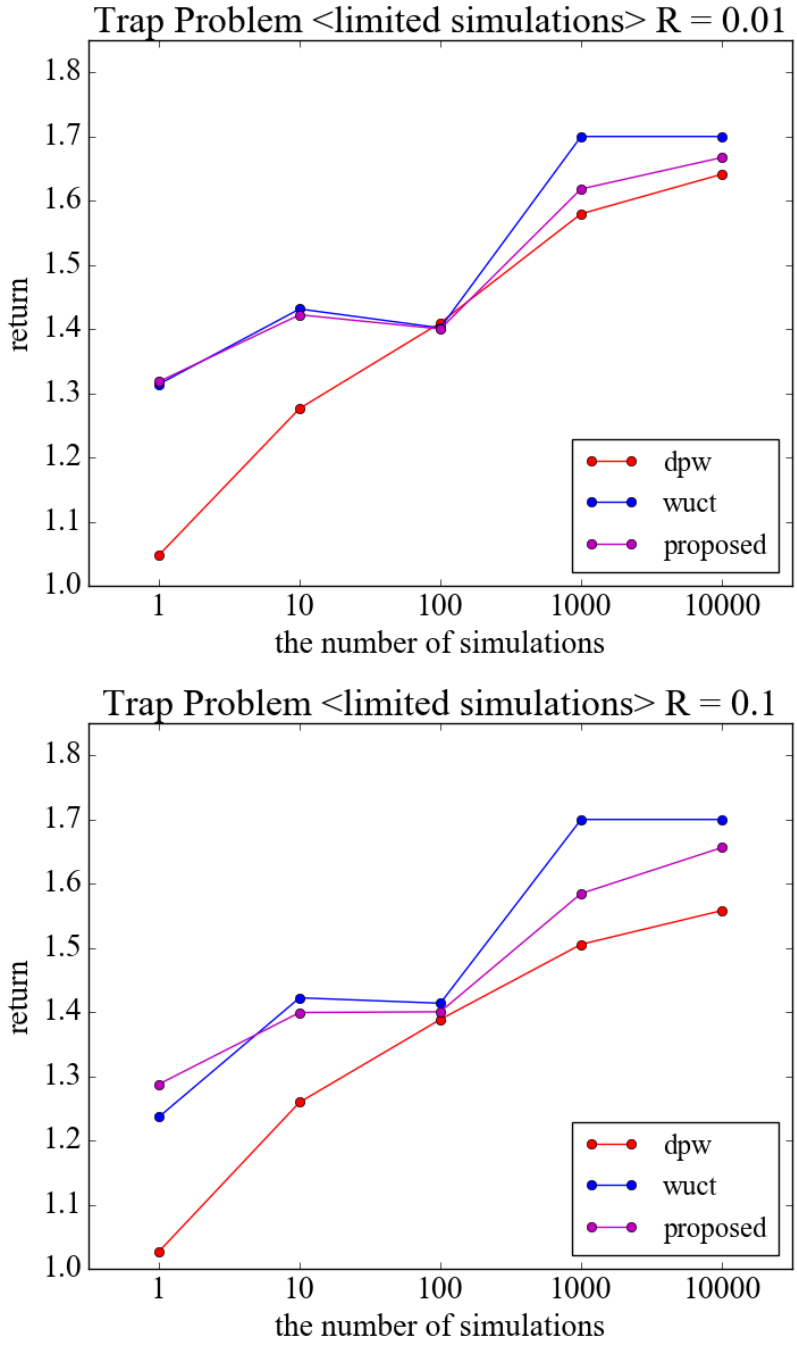

図 4.1: Trap Problem（シミュレーション回数固定）平均収益

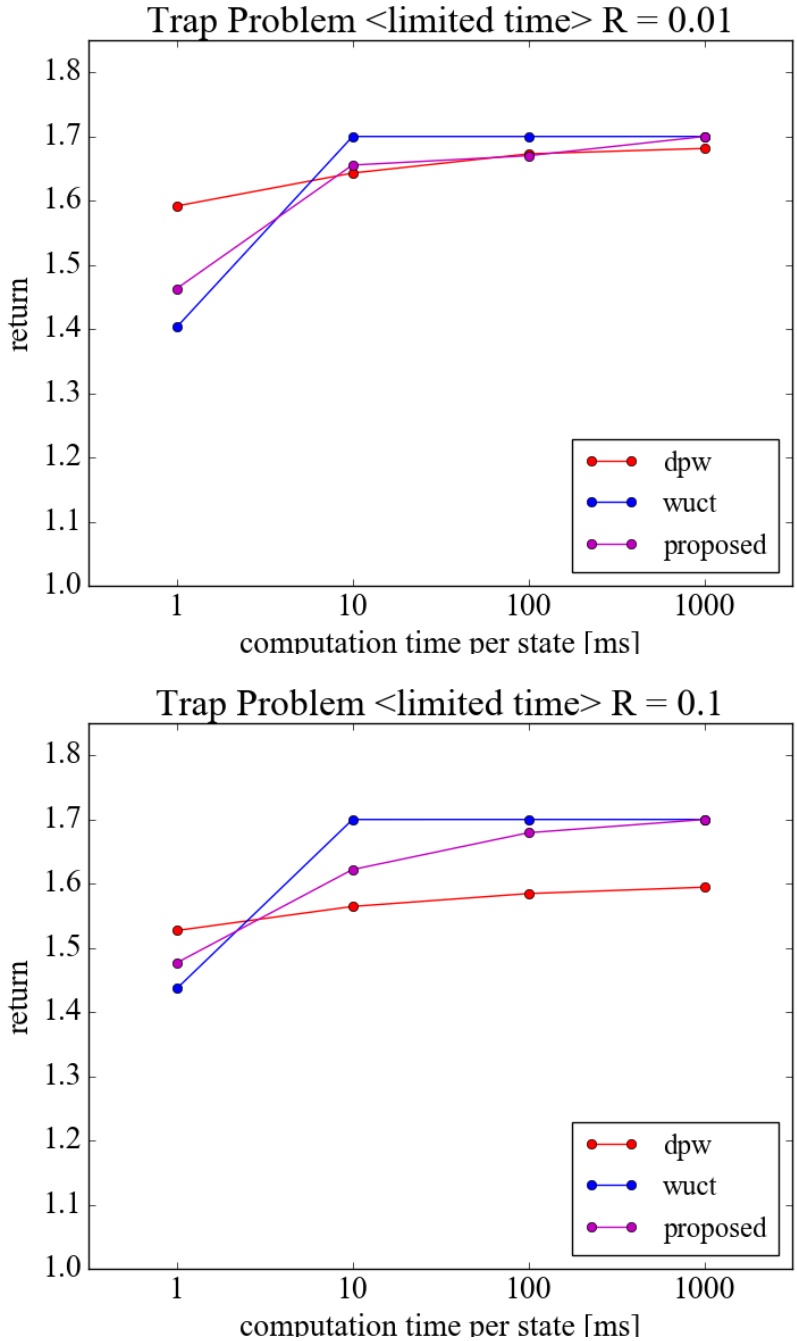

図 4.2: Trap Problem（計算時間固定）平均収益

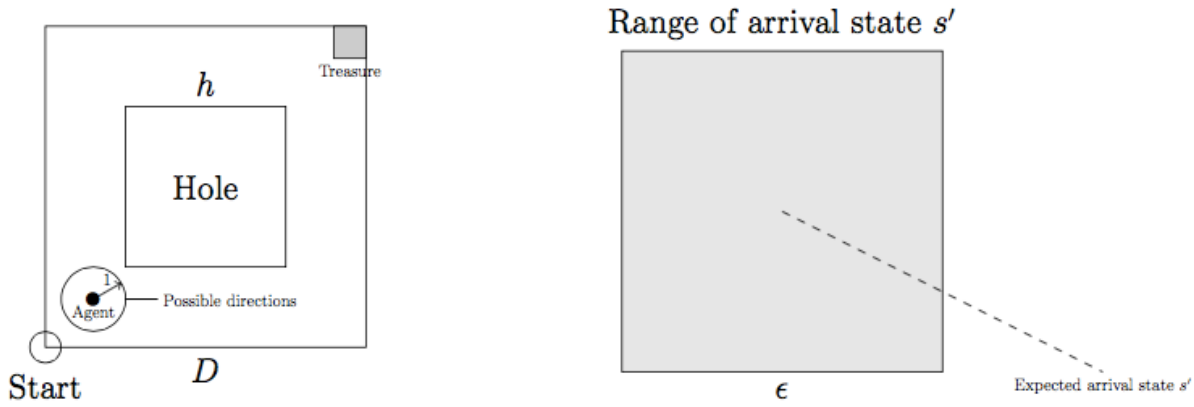

図 4.3: Treasure Hunt Problem（[19] より引用）

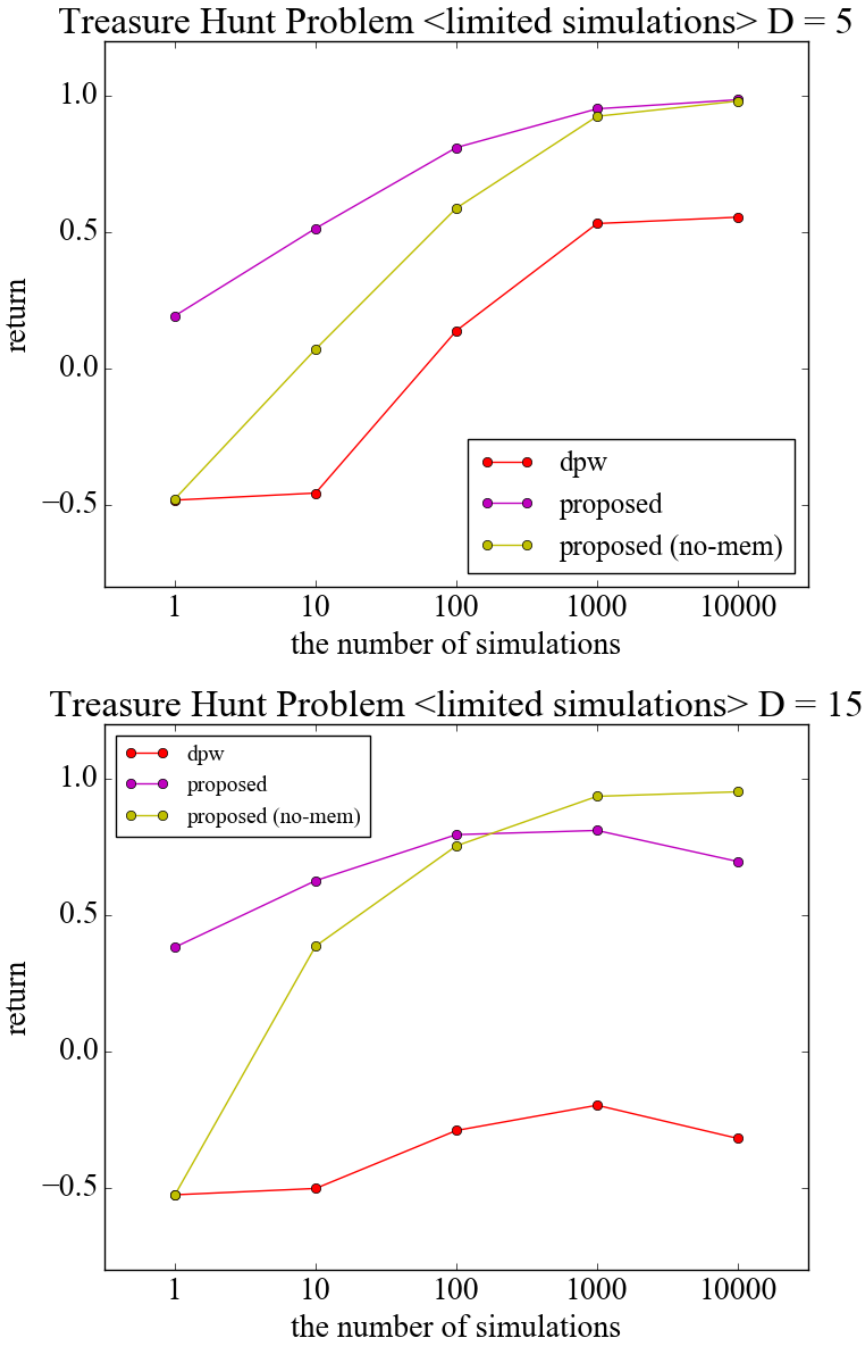

図 4.4: Treasure Hunt Problem（シミュレーション回数固定）平均収益

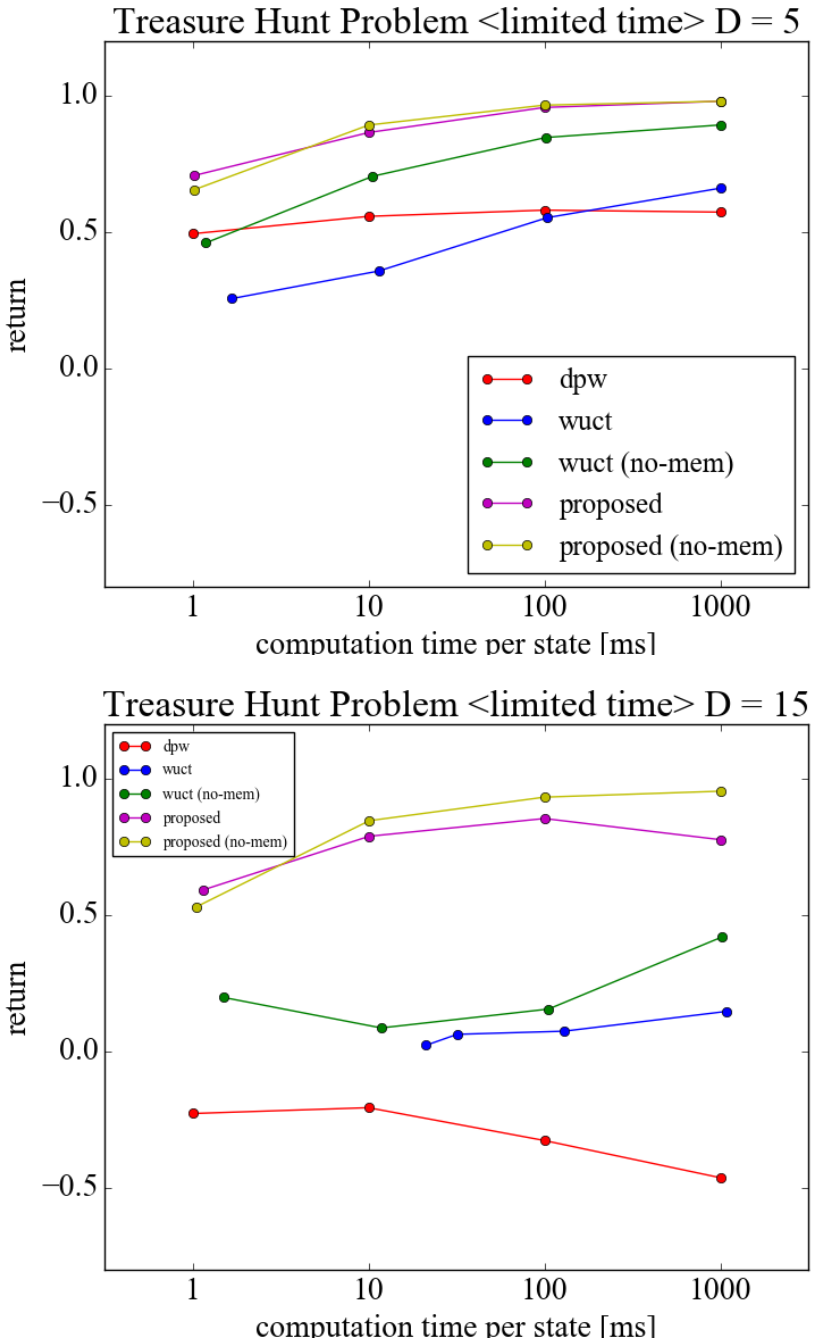

図 4.5: Treasure Hunt Problem（計算時間固定）平均収益

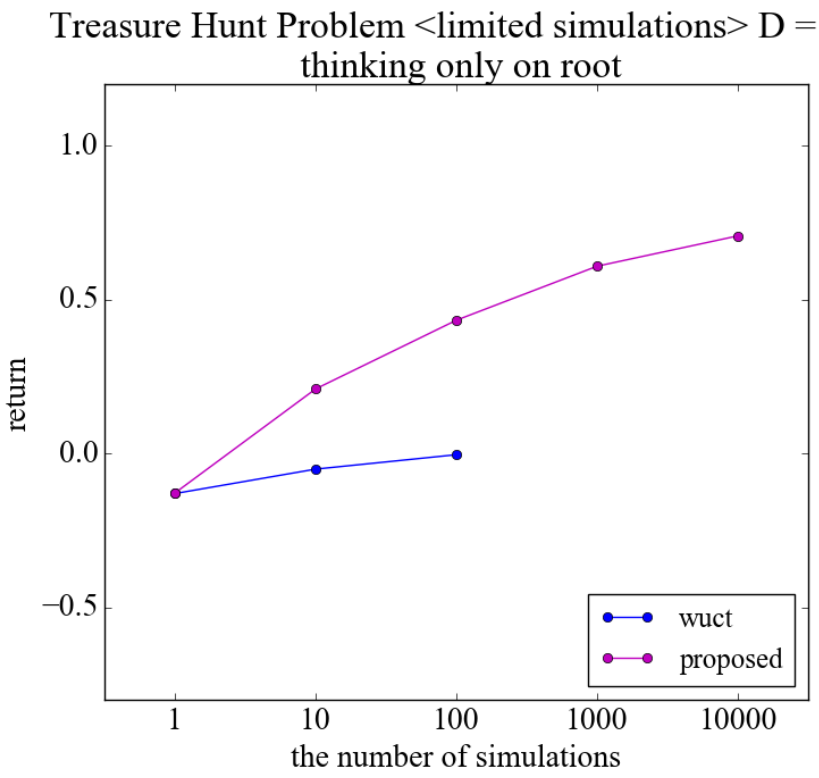

図 4.6: Treasure Hunt Problem（シミュレーション回数固定）平均収益

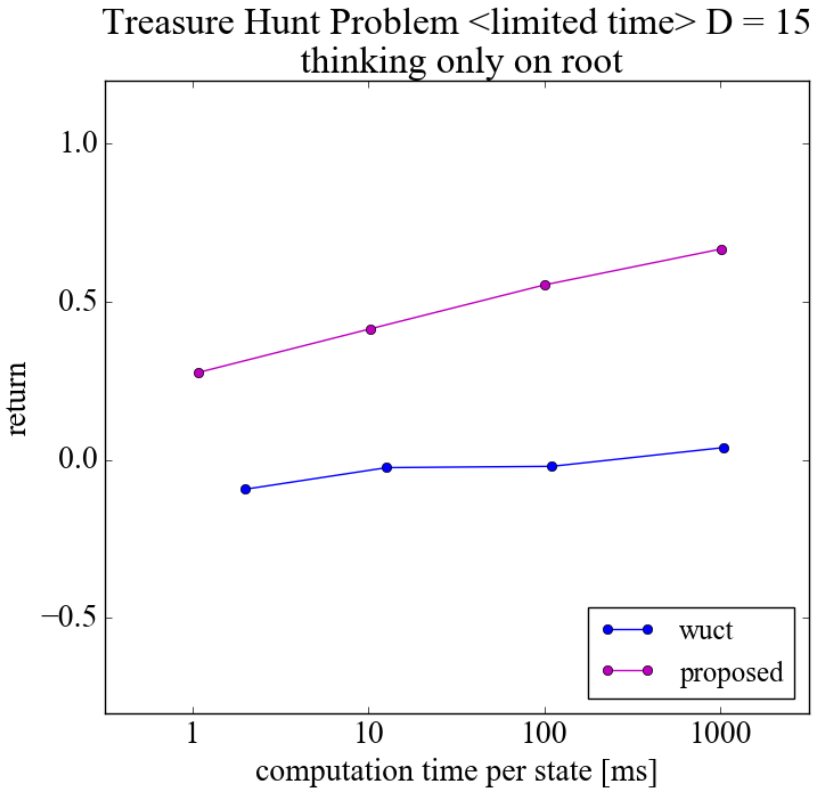

図 4.7: Treasure Hunt Problem（計算時間固定）平均収益

第5章　提案手法のカーリングへの適用

第4章にて状態空間がそれぞれ1次元，2次元の課題に提案手法を適用したが，本章ではより高次元の状態空間を扱う現実的な課題としてカーリングを扱う．

5.1　カーリング

本研究が主に対象とする，終端状態までの距離がそう長くないMDPとして扱えるより現実的な課題として，ウィンタースポーツの1つであるカーリングがある．カーリングはリンクと呼ばれる氷上にて，複数人で構成された2チームが対戦するスポーツであり,「氷上のチェス」と呼ばれるなど高度な戦略性を持つスポーツとして知られている．

カーリングの1試合はエンドと呼ばれる区切りを繰り返す形で行われる．

エンド内では，一方のチームを先攻チーム，もう一方のチームを後攻として，ストーン（以下単に石と呼ぶ）を滑らせる．カーリングにおける石を投げて止まるまでの一連はショットと呼ばれる．

1エンドでは先攻後攻が交互に1投ずつ，各チーム8投を投げ，エンド終了後にハウスと呼ばれる円内にある石のうち円の中心に最も近い石のチームが，ハウス内で相手の最も中心に近い石よりも内側にある自軍の石の個数分の得点を得る．例として，図5.1では黄色側のチームが1点を獲得する．ハウスの中に石が1個もない場合には両チーム0点となる．

上記の手順で行われる1エンドを複数回（一般には8エンドや10エンド）繰り返し，最終的な獲得得点が多い方が勝者となる．各エンドでは最後に石を投げることができる後攻が有利なため，あるエンドで1点以上得点したチームを次のエンドの先攻とする．また全エンドが終了した場合には，引き続き同じルールでの延長戦が行われ，先に得点したチームが勝者となる．

以上がカーリングの主なルールであるが，カーリングにおいて多人数性が絡んでくるルールとして,「スウィーピング」という，投げた石の軌道上をチームメンバーがブラシでこすることができるというルールがある．スウィーピングは声を掛け合いながら複数人で協力して進める必要があるので，この点を考えるとカーリングはマルチエージェント環境として捉える必要が出てくる．

5.2　デジタルカーリング

一方，カーリングの戦略を大まかに議論する上でスウィーピングのルールを取り除き，二人プレイヤの敵対型環境として単純にモデル化し計算機上のシミュレーションで再現する試みとして伊藤らの「デジタルカーリング」フレームワーク [21][22] が挙げられる．こ

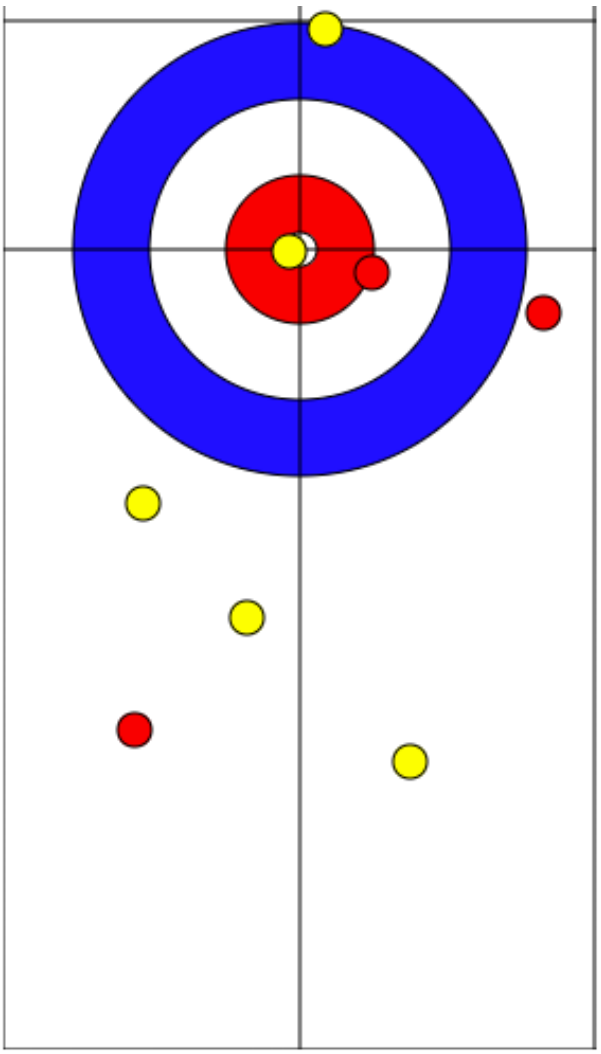

図 5.1: 黄色側チームが 1 点を獲得する石の配置の例

のフレームワーク上では，各チームを単独のプレイヤに見立てて石の初速度を行動として受け取り，氷上での石の運動を独自にモデル化し物理演算エンジンの Box2D[1] を用いて計算する．ただし，元が人間のスポーツであるためプレイのミスがある点，氷の状態が実際には均一ではない点（デジタルカーリング内では氷の状態は一様に同じ）のモデル化として，指定された初速度に対してノイズが加えられる．これにより，状態遷移に不確定性が加えられている．

これまでデジタルカーリングフレームワーク上でのプログラムの大会も複数回開催され [23]，アルゴリズムの性能検証の土台となっている．

本章では，このデジタルカーリングフレームワーク上での敵対的な環境下におけるプランニングで提案手法での性能を検証する．

[1] http://box2d.org/

5.3　カーリングの行動決定における既存研究

本研究で扱うデジタルカーリングフレームワークにおいては，加藤ら [24] や山本ら [25] が状態評価関数を用いた expectimax 探索の有用性を示している．また森ら [26] は実際のスポーツにて形式化されている各エンドの序盤戦略をデジタルカーリングに適用した．

他に，Yee ら [27] はカーリングの状態遷移の特徴「行動に少量のノイズが加えられること」を利用しある行動に対して近傍の離散的な行動の評価を確率密度関数で畳み込むことで行動価値を推定し，さらに行動候補の生成個数を段々と増やすことで連続空間から良い行動を探索するモンテカルロ木探索 kernel regression UCT (KR-UCT) を提案し，独自フレームワークにおいて有効性を実証している．KR-UCT が行動空間に対しての価値の連続性を利用している一方で本研究の提案手法は状態空間での連続性に着目しており，互いに別の側面からモンテカルロ木探索を適用した手法といえる．

一方，これまでカーリングの行動を end-to-end で直接学習させる有力なモデルは提案されていない．そのため探索的手法の性能を調べるベンチマークとして意義があると考えている．

5.4　デジタルカーリングの問題定式化

デジタルカーリングを本研究が扱っている MDP として解釈する上では，以下のような定式化が可能である．

状態 x 残りエンド数，エンド内の残り投数，先攻か後攻か，点差， 石の配置

行動（ショット） a 石を投げる初速度 v_x, v_y，石の回転方向 w

報酬 勝ち 1，負け 0

試合全体を 1 エピソードとして扱うこともできるが，状態のうちエンド間をまたいで関係するものは残りエンド数，先手後手，点差だけであり，これらは全て離散的でかつ取りうる値のパターン数は，10 エンド程度であればそれほど多くない．かつカーリングにおいて点差と勝率の関係は相手によらず一定という仮定をおけば，エンド終了後に次エンドの先手後手とその時点での得点差，残りエンド数から勝率を計算するモデルを作り，それが正しいことを前提に置いてエンド終了時点までのみを探索する方法が考えられる．

本研究ではこのエンド終了時点までの探索によって提案手法の評価を行った．本研究で用いた，残りエンド数と点差ごとの勝率は表 5.1 に示した各エンドの得点の分布に従って全てのエンドで得点されると仮定した場合の勝率とした．ただし延長戦の先手勝率は 0.25 を仮定した．この表は筆者のカーリングについての知識から作成したものであり，公開されているデジタルカーリングのプログラム同士の棋譜における得点分布にある程度近くなるように手調整している．ただし「残りエンド数や点差に関わらず 1 エンドの得点分布が一定」という仮定はカーリングにおいてはあまり正しくないと考えられるので，条件による得点分布の場合分けを行うことでより正確な勝率予測が出来る可能性がある．

さらに，エンド数が大きい場合には得点差の最大値は線形に増えていくが，現実的な範囲として想定する得点差は先手の 14 点負けから 17 点リードまでの 32 通りとした．カーリングでは得点をリードしたチームが，大量失点をしないような安全な戦略をとることが

できるので 10 点以上点差が離れる場合には 10 エンド以内の試合であればほぼ勝敗が決していると考えて問題なく，そのため想定した「先手の 14 点負けから 17 点リード」の範囲の端では予測勝率はリードした側にほぼ 1 の値がついているため，結果への影響は小さいと考えている．

表 5.1: デジタルカーリングの得点分布として定めた値

次エンド先後交代あり		次エンド先後交代なし	
得点	割合	得点	割合
後手 8	0.000195461	先手 8	0.00000547291
後手 7	0.000781844	先手 7	0.0000547291
後手 6	0.00312738	先手 6	0.0000547291
後手 5	0.0125095	先手 5	0.000547291
後手 4	0.0390922	先手 4	0.00547291
後手 3	0.070366	先手 3	0.0156369
後手 2	0.195461	先手 2	0.0547291
後手 1	0.390922	先手 1	0.172006
		0	0.0390922

5.5 デジタルカーリングに対する提案手法の適用詳細

5.5.1 状態木の実装

図 3.1 で示した状態木のカーリングにおける実装について，現れた状態集合に対してのみノードを生成する点，各状態集合にハッシュ値を割り当てる点は引き続き実装しており，ハッシュ値の計算は具体的には以下のように行った．
なお，位置座標は石を投げる点から見て横方向に x 軸，奥行き方向に y 軸を取っている．
1. それぞれの石の連続空間での x 座標と y 座標のそれぞれを小さなステップサイズの離散座標（各 15 ビット）に変換
2. 離散座標の各ビットの値によるゾブリストハッシュ[29]（石の色ごとに別）を加算によって計算

このとき下位の $15 - d$ ビットを無視することで分割回数 d の状態集合のハッシュ値を計算できる
3. 手数，ハウス中心からの石の並びの情報を加えたハッシュ値とする
4. 開番地法によるハッシュテーブル上のエントリに到達回数や報酬和を記録

1 で石の座標を 15 ビットで表現することで，状態木における状態空間の最大分割回数を 15 回と定めた．状態集合における各座標の範囲の幅は x 軸，y 軸ともに分割 15 回の最深部では 0.5mm 程度，状態木の根ノードでは 15.8m 程度でプレーエリア全体が収まるようにした．

また2において石の座標からハッシュ値への変換の際に石を区別せず先攻の石か後攻の石かのみで場合分けすることで，同じ色の石の位置が入れ替わった等価な状態集合をまとめあげることを可能とした．

一方，状態集合をハッシュ値によって見分けていることにより，低い確率ではあるが異なる状態集合が同じ場所に記録されてしまう可能性がある．本研究における実装では，ハッシュテーブルはエントリ数約 100 万の固定サイズかつエントリの上書きをせず，リハッシュ回数に上限を設けたため，容量オーバーやリハッシュ回数オーバーによって該当の状態集合のエントリを作成できない場合がある程度あり，厳密に状態木を再現できてはいない．

状態木の作成において手数に制限は設けず，到達した状態全てに対して，対応する状態木が作成されていない場合には状態木の作成を試みた[2]．また，状態木のノードに記録された報酬情報は手数が進んでも有効であるものが含まれるが，本論文の実験においては状態木のノードを記録したハッシュテーブルは行動決定機会ごとにクリアした．

5.5.2 シミュレーションの実装

シミュレーション中で状態木の挙動を定めるそれぞれの関数については以下の設定とした．

状態木の分割回数 d のノードの展開閾値 $th(d)$ は

$$th(d) = 1.3^d \tag{5.1}$$

状態への到達回数を合算する際の重み付けの関数 $w(d)$ は以下とした．

$$w(d) = (d+1)^4 \tag{5.2}$$

各ノードの到達回数を $w(d)$ で重み付けて算出した状態 x の仮の到達回数 $\tilde{n}(x)$ から $n(x)$ への変換は

$$f(n_{sum}) = \tilde{n}_{sum}^{0.4} \tag{5.3}$$

とし，さらにシミュレーション中の UCB1 値の計算の際の係数 $C_{UCB} = 1$ とした．

さらに，何点か提案手法のアルゴリズム 5 から変更を加え，実際にカーリングに対して適用したアルゴリズムをアルゴリズム 8 に示した．

変数 $N_{s,a}, R_{s,a}, N_s$ と関数 $w(depth), th(depth), f(frequency)$ はアルゴリズム 5 と同じものを意味している．なお，カーリングにおいて割引率 γ は常に 1 であり即時報酬はエンドが終了するエピソード末端でしか得られないことを前提としている．それによって状態遷移関数 $transition(state, action)$ は次の状態のみを返すように変更されている．

これ以外にもアルゴリズム 5 からの変更点が数点ある．

1 点目が「ノードにおける行動の結果の初期化時に事前分布として N_{new} 回分は選択したことにする」点である．

[2]ただし特別な状態として，石の半径を l としてハウスの近辺 $2l$ とその手前側の範囲に一部でも入っている石が無い場合には空の状態と認識し，状態木とは別に結果を保存した．

Algorithm 8 カーリングにおける状態木を用いたシミュレーション

A_s ノード s に含まれる状態で選択された行動の集合
R_s ノード s に含まれる状態からのシミュレーションで得た収益の合計値
N_{new} ノードの行動回数の事前値
N_{policy} 行動回数の事前値
function $simulationInCurling(x)$
if x が終端状態 **then**
  return x の終端報酬
end if
$A_x \leftarrow$ 状態 x で可能な行動の集合
$\tilde{n} \leftarrow$ 行動から実数へのマップ 各要素の値を 0 で初期化
$\tilde{r} \leftarrow$ 行動から実数へのマップ 各要素の値を 0 で初期化
$s \leftarrow$ 状態 x を含む状態木の根ノード
$_, r_{sim} \leftarrow simulationOnStateTree(x, 0, s, A_x, \tilde{n}, \tilde{r})$
return r_{sim}

function $simulationOnStateTreeInCurling(x, d, s, A_x, \tilde{n}, \tilde{r})$
$W \leftarrow w(d)$
for $a \in A_x$ **do**
  if $a \in A_s$ **then**
    $\tilde{n}[a] \leftarrow \tilde{n}[a] + W \times N_{s,a}$, $\tilde{r}[a] \leftarrow \tilde{r}[a] + W \times R_{s,a}$
  else
    $\tilde{n}[a] \leftarrow \tilde{n}[a] + W \times N_{new}$, $\tilde{r}[a] \leftarrow \tilde{r}[a] + R_s/N_s$
  end if
end for
if $N_s \geq th(d)$ かつ s の子ノードで状態 x を含むものが未作成 **then**
  状態 x を含む s の 子ノード s' を追加
  $N_{s'} \leftarrow 0$
end if
if s の子ノードで状態 x を含むものが未作成 **then**
  $\tilde{n}_{sum} \leftarrow$ summation of $\tilde{n}$
  $n_{sum} \leftarrow f(\tilde{n}_{sum})$
  $n \leftarrow A_x$ 内の行動から実数へのマップ a に対する要素は $\tilde{n}[a] \times n_{sum}/\tilde{n}_{sum}$
  $r \leftarrow A_x$ 内の行動から実数へのマップ a に対する要素は $\tilde{r}[a] \times n_{sum}/\tilde{n}_{sum}$
  $n_{sum} \leftarrow n_{sum} + N_{policy} \times \#A_x$
  $a_{sim} \leftarrow A[\mathrm{argmax}_{i \in 0..\#A-1} \left(\frac{r[i]}{n[i]} + C_{UCB} \sqrt{\frac{2 \ln n_{sum}}{n[i]}} \right)]$
  $r_{sim} \leftarrow -simulationInCurling(transition(x, a_{sim}))$
else
  $s' \leftarrow$ 状態 x を含む s の子ノード
  $a_{sim}, r_{sim} \leftarrow simulationOnStateTreeInCurling(x, d+1, s', A, \tilde{n}, \tilde{r})$
end if
if ノード s に $N_{s,a_{sim}}$, $_{s,a_{sim}}$ が記録されていない **then**
  $N_{s,a_{sim}} \leftarrow N_{new}$, $R_{s,a_{sim}} \leftarrow 0$, $N_s \leftarrow N_s + N_{new}$
end if
$N_{s,a_{sim}} \leftarrow N_{s,a_{sim}} + 1$, $R_{s,a_{sim}} \leftarrow R_{s,a_{sim}} + r_{sim}$
$N_s \leftarrow N_s + 1$, $R_s \leftarrow R_s + r_{sim}$
return a_{sim}, r_{sim}

2 点目は「候補行動についての結果がノードになかった場合には代わりに事前分布として N_{new} 回分，そのノード全体での収益の平均の報酬を得たことにしておく」点である．これは同じ状態集合であれば他の行動における価値も近いだろうという知識に基づいており，状態木のルートからリーフまでのパス中に目的の行動が記録されていなかった場合の補完的役割を果たす．

3 点目は，n_{sum} に事前分布的に $N_{policy} \times \#A_x$ を足していることである．

ただし，この 2 点目に関しては $\tilde{r}[a]$ に加算する R_s/N_s にも重み W をかけるのが自然であるし，3 点目については $n(x,a)$ と $r(x,a)$ に N_{policy} を足すことで n_{sum} との整合性がとれるが，実際にアルゴリズム 8 の手順で検証を行ったため，提案手法の検証中の別バージョンとして話を進める．

5.5.3 モンテカルロ木探索の実装

デジタルカーリングに提案手法を実装する上で，シミュレーションのアルゴリズム以外についても提案手法のベースから 3 点の変化を加えた．

1 点目は行動空間の離散化をルート状態とシミュレーション中の状態で別とするという変化である．

カーリングの行動空間は連続であり，カーリングの行動空間全体のうち良い行動は石に当てるなどのピンポイントな行動が多いため，行動空間全体に対して行動価値は極大値の周囲で急峻に上がっているだろうと考えられる．そのため行動空間全体から一様に離散的な行動を与えても大部分が価値の低い行動であろうと考えた．

そこでシミュレーション内の行動に関しては以下のようにした．

- すべての局面　→ハウス中心へのドロー（右回転，左回転）
- ハウス内に相手の石がある　→　相手の最も内側の石をヒット（右回転，左回転）

行動候補の数は石の配置により 2 または 4 である．実際のカーリングではより多様なショットが求められるが，これにガードストーンを加えた程度のデジタルカーリングのサンプルプログラム [3] であっても，筆者は苦戦する程度の強さではあったため，最低限の戦略としては有効であると考えている．

一方で，ルートノードにおける候補ショットの生成は，より広い行動空間から選ぶため加藤ら [24] の手法を参考にして目標到達座標を離散化しすることで行った．加藤らの手法においては，主にドロー系のショットとしてハウス内とその前方に等間隔で目標到達座標の点を生成する．さらにテイクアウト系のウェイトの大きいショットとして，遠方に一列の目標到達座標の点を生成する．本研究では加藤らの離散化から範囲を一部変更し，また大会の外乱ルールの変更に合わせて離散化の幅を変更した．石の半径 l に対して，ドロー系のショットの座標を幅 $(\frac{l}{2}, 2l)$ で 63×31 個の点を配置し，ヒット系のショットは幅 $(\frac{l}{4}, \infty)$ で 189×1 個の点を配置した．
結果として目標到達座標は 2142 個，ショットに右回転と左回転があるためショットは計 4284 個生成された．

文献 [24] から目標到達座標の離散化の図を引用し図 5.2 に掲載した．

[3] URL

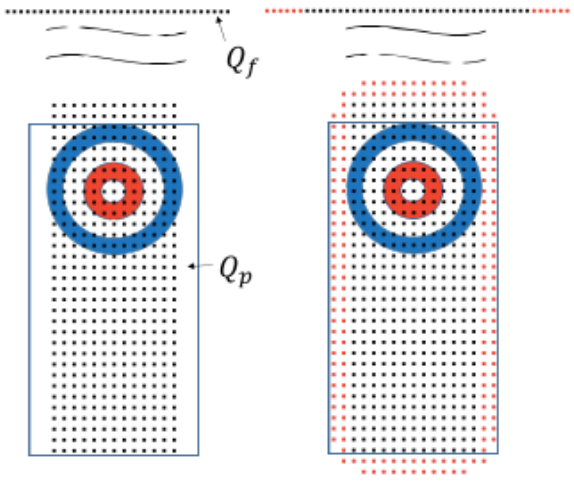

図 5.2: カーリングのショットの目標到達座標の離散化 加藤ら [24] より引用

図 5.2 において点群 Q_p がドロー系のショット，Q_f がヒット系のショットを表す．本研究においては，図中の黒い点で示された最終的に選択の候補となる目標到達点の数が 1440 個，赤い点で表されたショットの価値の期待値算出の際に用いられる点も含めると 2142 個となった．なお，文献 [24] においては価値計算を行うための近傍点を円形にとっていることが図 5.2 から読み取れるが，本研究においては計算の簡略化のため価値計算に用いる近傍の点は長方形の範囲内としている．

第 3 章で述べた手法からの変更の 2 点目は，ルートノードでの行動価値の計算とシミュレーションの割り振りを変更した点である．

ルートノードでの行動の生成において到達座標に対して格子状にショットを配置した理由として，デジタルカーリングにおいてはショットに加えられるノイズが到達座標に対して正規分布でずれるように与えられることがある．
この点を利用すると，ある行動の「ノイズを考慮した行動価値」は，離散化された近傍の行動の「ノイズが無い場合の」行動価値と「ノイズの確率密度」から算出することが可能になる．これは既存研究 [24][25] で使われている手法であり，デジタルカーリングにおいて比較的少ない評価回数で信頼性の高い行動価値の算出を行えることが示唆されている [25]．本研究でもこの手法を利用し，ルート状態においてのみノイズを加えずにシミュレーションを行った．

ただし，ノイズを考慮した行動価値の計算においては周辺の行動が全て評価されている必要がある．そのため格子状に配置した行動のうち端の方に配置された行動の価値を計算することはできず，結果として最終的にノイズを考慮した行動価値を算出し最適行動として出力できる候補はこの周囲の行動を除いた行動集合となる．目標到達座標 2142 個に対して，周囲を除いた最終的な候補の点は 1440 個，行動候補の数としては右回転と左回転の 2880 個とした．

最終的にノイズを含んだ行動価値を計算するため，ノイズが無い場合に価値が低そうな行動であってもノイズを考慮して最適行動になりそうな点の付近であれば正確な評価を行う必要があるので，ルート状態でのシミュレーションの割り振りに UCB 等のバンディット手法をそのまま適用はできない．それでも工夫の余地はあるが，本研究においては単純

に全ての候補行動をランダムに並べ替えて，先頭から順にシミュレーションを割り振ることを 1 周として，複数周行うことで各行動の評価の精度を向上させる実装とした．

本研究でデジタルカーリングに対して適用したアルゴリズムの全体はアルゴリズム 9 のようになった．$transitionWithoutNoise(state, action)$ 関数がノイズを加えずに状態遷移を行う関数である．

Algorithm 9 カーリングに対しての状態木を用いたモンテカルロ木探索

$allRootActions$(state) ルート状態でのシミュレーションを行う行動候補集合を返す関数
$candidateRootActions$(state) ルート状態で最終的に選ばれる行動候補集合を返す関数
C_{limit} シミュレーション回数
$c \leftarrow 0$
$A_{all} \leftarrow allRootActions(x)$
$N_{root} \leftarrow$ 要素数 $\#A_{all}$ の配列 値 0 で初期化
$R_{root} \leftarrow$ 要素数 $\#A_{all}$ の配列 値 0 で初期化
while $c < C_{limit}$ **do**
  $seq \leftarrow 0$ から $\#A_{all} - 1$ をランダムに並べ替えた配列
  for $i \leftarrow 0$ to $\#A_{all} - 1$ **do**
    $x \leftarrow transitionWithoutNoise(x, A_{all}[seq[i]])$
    $N_{root}[seq[i]] \leftarrow N_{root}[seq[i]] + 1$
    $R_{root}[seq[i]] \leftarrow R_{root}[seq[i]] + simulationInCurling(x)$
    $c \leftarrow c + 1$
  end for
end while
$A_{cand} \leftarrow candidateRootActions(x)$
for $a \in A_{cand}$ **do**
  ノイズを考慮した行動価値を，周囲の行動の価値から計算
end for
最高評価の行動を返す

第 3 章で述べた手法からの変更の 3 点目は，シミュレーションの末端報酬の正規化である．

前節で仮定した残りエンド，点差と予測勝率を仮定したが，この値を末端報酬として用いる場合,「互角か，どちらかが優勢か」「残りエンド数が多いが少ないか」によって分散が大きく変化するため探索と活用のバランスが崩れるという懸念がある．そのためエンドごとに得点の分布が 5.1 に従った際に予測勝率が平均 0，分散 1 となるように毎エンド正規化して末端報酬として用いた．UCB1[11] のボーナス項の係数を報酬の標準偏差に置き換えることの理論的な正当性については [28] において主張されており（ただしバイアス項の式は少し異なるものになっている），報酬を分散一定に正規化することはこの逆の操作と考えられる．

5.5.4　その他の実装上の工夫

カーリング課題が 4 章で扱った課題より実践的に困難な点として，カーリングは物理的な系で状態遷移が起こるためプランニングの際に高コストな物理演算を伴うという点がある．デジタルカーリングフレームワークにおいて提供されている物理演算部は汎用物理エンジンの Box2D をベースとしており，標準設定ではエンドの第 1 投において秒間数百回程度しかシミュレーションを行うことができないことを確認している．そこで本研究では筆者が独自にカーリングに特化した物理演算関数を実装して探索中にのみ使用している（対戦サーバー側での演算は公式の関数そのままとしている）．

自作の物理演算関数の返す結果は公式のフレークワークに付属の物理演算関数と異なる結果を返すことが多いが，過去のデジタルカーリングのいずれの大会においても優勝プログラムは独自に高速化した物理演算関数を用いていることがわかっており，サーバーの物理演算結果との違いは結果において重要ではないと考えられる．

5.6　対戦実験計画

ここまでカーリングに特化した実装上の工夫を何点か述べたが，あくまで本章では本研究の提案手法である連続状態空間内でのプランニングの有効性について検証することが目的である．

そのため，実験の比較対象としてはシミュレーション内の処理のみを変化させたプログラム，具体的には，シミュレーション内を生成された行動の中からランダムに選ぶ純粋モンテカルロ法を検証の相手プログラムとした．

実験においては，行動決定機会 1 回あたりのシミュレーション回数は，後攻のラストストーンのみ各ショット 1 回ずつの 4284 回，それ以外では全て各ショット 47 回の 201348 回で固定した．

対戦は先後を入れ替えて各 400 試合の計 800 試合を各対戦それぞれで行った．対戦は GAT 杯ルール [4] で 2 エンド戦と 10 エンド戦を行った．カーリングは各エンドの先攻と後攻の有利不利の差が大きいので，延長戦に入る際に先攻か後攻かは結果に大きく影響を及ぼす．カーリングの大きな試合では延長戦ありで試合が行われるため，本研究での検証プログラムも延長戦を想定し，最終エンド終了時に同点の場合の報酬を延長戦の先後で変化させているが実験に使用した対戦サーバープログラムの棋譜出力の都合により，全エンド終了後に同点の場合は試合を打ち切って結果の解析の際に考慮することとした．

実験には複数の計算機を利用した．シミュレーション回数固定としているため，使用した計算機による性能の違いは無いと仮定して順次実験を割り振った．

5.7　実験結果

結果の集計にあたって，過去のデジタルカーリングプログラムの大会の上位プログラム同士対戦の棋譜から最終エンド開始時点で同点の場合の試合結果を抽出し，そこから棋譜中のプログラム群における延長戦の先手の期待勝率を求めたところ 0.219 程度であった．

[4] デジタルカーリング公式サイト http://minerva.cs.uec.ac.jp/curling/wiki.cgi の「GAT 杯 (2016)」のページを参照.

そのため対戦結果から勝率への換算を行う際には仮に延長戦があった場合に先手で延長戦に入る場合は 0.219 勝，後手で延長戦に入る場合は 0.781 勝として計算に加えた．

上記の換算によって計算された勝率が $\frac{1}{2}$ と異なるかをカイ二乗検定で検定した．

表 5.2: 提案手法 の 純粋モンテカルロ法 に対する勝敗 (2 エンド)

	勝	後攻で延長戦	先攻で延長戦	負	換算勝率	平均
第 1 エンド先攻	128	34	67	171	0.423	0.617
第 1 エンド後攻	289	43	8	60	0.811	$(p = 4 \times 10^{-11})$

表 5.3: 提案手法 の 純粋モンテカルロ法 に対する勝敗 (10 エンド)

	勝	後攻で延長戦	先攻で延長戦	負	換算勝率	平均
第 1 エンド先攻	271	8	9	121	0.698	0.803
第 1 エンド後攻	356	8	3	33	0.907	$(p = 1 \times 10^{-65})$

2 エンド戦，10 エンドのいずれにおいても，提案手法を適用した側が純粋モンテカルロに勝ち越した．

第6章　考察

本研究では既存研究の WUCT や DPW の特徴を鑑み，改善が望まれる点について状態空間を階層的に分割した状態木の構造を利用することによるモンテカルロ木探索ベースのプランニングアルゴリズムを提案し，低次元の課題 2 つとカーリングに対して適用した．

まず実験結果についての議論を行う．第 4 章における実験にて Trap Problem では WUCT の性能に及ばなかったが，Treasure Hunt Problem においては既存手法を上回る結果となった．

この点に関して，まず本稿における実験では課題ごとにパラメータチューニングを行なわずに，大部分の手法に関しては Trap Problem と Treasure Hunt Problem に対して同じパラメータを用いているため，各課題においての結果から，どのアルゴリズムがどのような課題に適しているかを断定するほど強い主張はできないと考えている．

Trap Problem においては WUCT が良い性能をあげた．一方 Treasure Hunt Problem においては提案手法の方がシミュレーション回数単位でもよい結果であったが，本来 WUCT は木構造による近似というパラダイムに限定されない分提案手法よりも高い表現力を持っているはずである．そのため Treasure Hunt Problem においてもカーネルや他のパラメータの設定次第では違った結果が得られる可能性があると考えている．

また DPW は Trap Problem の短時間以外では他手法より低い性能という結果であったが，DPW に，シミュレーションの後の方で選んだ行動も先の方で選んだ行動とみなして結果に加算する RAVE の手法を適用した DPW-RAVE[30] も提案されている．こちらは本研究の提案とは違った形で多くの課題に共通する性質の活用を試みており，実際に $D = 5$ の Treasure Hunt Problem においてはナイーブな DPW より高い性能をあげている [30]．今回は時間の関係で比較対象には含めなかったが有力であると考えられる．DPW-RAVE は一度のシミュレーションの結果の Backpropagation にシミュレーションの経路長の 2 乗のオーダーの計算時間がかかると考えられ，計算時間ベースでの性能評価は既存研究には見られず，今後検証するべきであろう．

以上の点も考慮に入れた上で，本研究が目指した「近い状態の行動価値の利用による効率向上」「状態空間の木構造化による近似的計算による計算時間の短縮」という性質の提案手法を評価すると，既存手法に比べて適用できる課題の幅を広げたという成果は十分にあると考えている．今回の実験の課題設定において比較的短時間やシミュレーション回数の少ない状況，特に初期状態でのプランニングだけの場合でも比較的良い性能が残せたということで，複数のパラメータ設定での探索を同時に動かすなど様々な応用が広がるアルゴリズムなのではないかと期待している．

一方カーリングへの適用においては，提案手法における状態空間の分割方法では最大 30 次元の空間で状態木の分岐数が膨れ上がるために性能低下が起こる可能性があるのではないかと自然には考えられるが，純粋モンテカルロに対して勝ち越すことができ，最大 30 次元程度の状態空間においても機能しうる手法であることが示唆された．理想的には

木構造は2分木や3分木といった少ない分岐数にすることで深い階層化が可能になり良い性能が出るのではないかと期待されるが，状態空間の分割において課題に対する特殊な知識を用いない最も単純な手法においてもある程度の成果はあるということで，今後より状態空間の分割手法を改善すれば，さらに高次元の課題においても成果をあげられる可能性がある．

それ以外に今後さらに検証を進めていくべき点として，今回の実験結果からあげるべき点に「思考量が増えることが単調にプランニングの質の向上につながっているのか」という問題がある．実際に Treasure Hunt Problem の $D = 15$ 条件において，記憶ありの提案手法はシミュレーション回数や思考時間が増えるに従って収益が落ちているという結果となっている．それに対して記憶なしの場合にはそういった影響は見られなかったが，これがデータの量による影響なのか，不要な記憶が増えることによる影響なのかは検証の余地がある．過去の記憶は役に立つものだけ残して一定数を保つことが理想的であるので，そのためにも詳しい解析が必要である．

思考時間が増えることによる性能低下の面も含め，元々近似的に計算することで高速化ことを目的としているアルゴリズムであるので，どの程度幅広い課題に対して最適解を導ける保証ができうるのかは難しい問題である．一方で実践として近似による悪影響が小さく納まるようにするためにも，アルゴリズムの細部の詰めきれていない部分について向上していく余地はあると考えている．

最後に，本研究では課題のサイズなどは予めハードコーディングしたが，さらに多目的でのプランニングを行う上ではこういったパラメータについても事前分布を保持しておき，柔軟に対応できることが望ましい．またシミュレーション中の方策は本稿では行動空間全体からの完全ランダムや，筆者が意図的に選んだ中からの完全ランダムのみを扱ったが，この点に関しても同じく方策関数の事前分布をドメインに依存しない形で持つことができれば，さらに効率良くプランニングを行えることが期待される．

本研究はそのための第一歩ではあるが，小さくない一歩であると胸を張って筆を置きたい．

謝辞

本研究に際して様々なご指導を頂きました田中哲朗准教授に深謝致します.

関連図書

[1] Poole, D., Mackworth, A.: Artificial Intelligence, http://artint.info/ (2017. 6. 29 閲覧)

[2] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D.: Mastering the game of Go with deep neural networks and tree search, Nature, Vol. 529, pp. 484 - 503 (2016)

[3] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, Martin., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning, Nature, Vol. 518, 529 - 533 (2015)

[4] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In International Conference on Machine Learning, pp. 1928 - 1937 (2016)

[5] Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., de Freitas, N.: Sample efficient actor critic with exprtience replay. ICLR 2017 (2017)

[6] Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., Degris, T.: The predictron: End-to-end learning and planning, https://arxiv.org/abs/1612.08810 (2017)

[7] Genesereth, M., Thielscher, M.: General game playing. Synthesis Lectures on Artificial Intelligence and Machine Learning, 8(2), pp. 1 - 229 (2014)

[8] Graf, T., Platzer, M.: Adaptive Playouts in Monte Carlo Tree Search with Policy Gradient Reinforcement Learning, Advances in Computer Games. Lecture Notes in Computer Science, vol 9525. Springer, Cham (2015)

[9] Robbins, H.: Some aspects of the sequential design of experiments, Bull. Amer. Math. Soc. 58, No. 5, pp. 527 - 535 (1952)

[10] Thompson, W. R.: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples, Biometrika, 25 (3 - 4) pp. 285 - 294 (1933)

[11] Auer, P., Cesa-Bianchi N., Fischer, P.: Finite-time Analysis of the Multiarmed Bandit Problem. Machine Learning, Vol. 47, pp. 235 - 256 (2002)

[12] Lai, T. L., and Robbins, H. Asymptotically efficient adaptive allocation rules. Advances in Applied Mathematics, Vol. 6, pp. 4 - 22 (1985)

[13] Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A Survey of Monte Carlo Tree Search Methods. IEEE Transactions on Computational Intelligence and AI in Games, Vol. 4, No. 1, pp. 1 - 49 (2012)

[14] Kocsis, L., Szepesvari, C.: Bandit based Monte-Carlo Planning. European conference on machine learning (ECML2006), pp. 282 - 293 (2006)

[15] Srinivas, N., Krause, A., Kakade, S. M., Seeger, M.: Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design, Proceedings of the 27th International Conference on Machine Learning (ICML 2010), pp. 1015-1022 (2010)

[16] Krause, A., Ong, C. S.: Contextual Gaussian Process Bandit Optimization, Advances in Neural Information Processing Systems, pp. 2447-2455 (2011)

[17] Weinstein, A.: Local Planning for Continuous Markov Decision Processes. Ph. D. thesis (2014)

[18] Couetoux, A., Hoock, J., Sokolovska, N., Teytaud, O., Bonnard, N.: Continuous upper confidence trees. International Conference on Learning and Intelligent Optimization, pp. 433 - 445 (2011)

[19] Couetoux, A.: Monte Carlo Tree Search for Continuous and Stochastic Sequential Decision Making Problems. doctoral thesis, Paris university (2013)

[20] Chaslot. G., Fiter, C., Hoock, J.P., Rimmel A., Teytaud, O.: Adding expert knowledge and exploration in Monte-Carlo Tree Search, Advances in Computer Games, LNCS, Vol. 6048, pp. 1 - 13 (2009)

[21] 北清勇磨, 岡田雷太, 伊藤毅志: デジタルカーリングサーバーの提案と紹介, 情報処理学会研究報告, 2014-GI-31, No. 2, pp. 1 - 5 (2014)

[22] Ito, T., Kitasei, Y.: Proposal and Implementation of "Digital Curling", 2015 IEEE Conference on Computational Intelligence and Games, pp. 469 - 473 (2015)

[23] 伊藤毅志, 森健太郎: デジタルカーリング大会報告 2015 年度, 情報処理学会研究報告, 2016-GI-36, No. 2, pp. 1 - 6 (2016)

[24] 加藤修, 飯塚博幸, 山本雅人: 不確定性を含むデジタルカーリングにおけるゲーム木探索, 情報処理学会論文誌, Vol. 57, No. 11, pp. 2354 - 2364 (2016)

[25] Yamamoto, M., Kato, S., Iizuka, H.: Digital Curling Strategy on Game Tree Search, 2015 IEEE Conference on Computational Intelligence and Games, pp. 474 - 480 (2015)

[26] 森健太郎, 伊藤毅志: デジタルカーリング AI における戦略書に基づく序盤定石の構築. ゲームプログラミングワークショップ 2016 論文集 pp. 172 - 179 (2016)

[27] Yee, T., Lisy, V., Bowling, M.: Monte Carlo Tree Search in Continuous Action Spaces with Execution Uncertainty. International Joint Conference on Artificial Intelligence (2016)

[28] Audibert, J. Y., Munos, R., and Szepesvari, C.: Exploration-exploitation trade-off using variance estimates in multi-armed bandits, Theoretical Computer Science, 410(19) pp. 1876 - 1902 (2009)

[29] Zobrist, A,: A New Hashing Method with Applications for Game Playing. ICCA journal, Vol. 13, No. 2, pp. 69 - 73 (1970)

[30] Couetoux, A., Milone, M., Brendel, M., Doghmen, H., Sebag, M., Teytaud, O.: Continuous rapid action value estimates. The 3rd Asian Conference on Machine Learning, Vol. 20 , pp. 19 - 31 (2011)