The 33rd Annual Conference of the Japanese Society for Artificial Intelligence, 2019

Novelty-Based Pruningにおける木の再利用

Reusing a Search Tree in Novelty-Based Pruning

石橋 遼 *1 Ryo Ishibashi　森山 甲一 *1 Koichi Moriyama　武藤 敦子 *1 Atsuko Mutoh　松井 藤五郎 *2 Tohgoroh Matsui　犬塚 信博 *1 Nobuhiro Inuzuka

*1名古屋工業大学 大学院工学研究科 情報工学専攻
Department of Computer Science, Graduate School of Engineering, Nagoya Institute of Technology

*2中部大学 生命健康科学部 臨床工学科
Department of Clinical Engineering, College of Life and Health Sciences, Chubu University

Novelty-Based Pruning (NBP) is a Monte-Carlo Tree Search (MCTS) method that introduced a pruning mechanism based on the novelty of nodes to search more unknown nodes in limited time. When an action is chosen, MCTS methods start a new search from a root node corresponding to the new state. However, it is not appropriate for NBP because the new search clears all information on what sub-trees should be pruned and the information is created again by redundant searches. This work proposes a novel method reusing a searched tree starting with a node corresponding to the new state, which is a child of the old root node. It is expected to increase the number of unknown nodes searched in limited time. In experiments of general video game playing, the number was slightly increased; however, it was not significantly different from the normal NBP.

1. はじめに

近年、様々な分野において木探索アルゴリズムは利用されている。木探索アルゴリズムには、幅優先探索や深さ優先探索といった様々な種類があり、その中の 1 つにモンテカルロ木探索と呼ばれる手法が存在する [1]。これは、ランダムな探索を繰り返してその結果を基に評価値を推定し、木を徐々に拡張して最良となるノードを選択するという手法であり、派生手法もいくつか開発されている。その中の 1 つである Novelty-Based Pruning は探索の際に無駄を省くことで、短時間で多くのノードが探索できるアルゴリズムである [2]。

このようなアルゴリズムが利用される分野として、様々な種類があるビデオゲームを汎用的にプレイするゲームプレイング AI の研究をする General Video Game PlayingAI (GVGAI) [3] という分野が存在する。その分野においてモンテカルロ木探索は、問題特有の知識を必要とせず汎用的に利用できることから、様々なゲームプレイング AI に利用されており [4]、特に Novelty-Based Pruning は 1 人用のゲームをプレイさせる部門において優れた成績を残している [5]。

しかし、このアルゴリズムは行動を選択した後に遷移先のノードをルートノードとして再び探索する際に、以前の探索でそのルートノードの子ノードだった部分を再び探索してしまうことや、以前の探索で不要と判断した部分を探索してしまうことで無駄が生じているという問題がある。そこで本研究では、その無駄な探索を削減しより多くのノードが探索できるようにすることを目的とする。

連絡先: 連絡先: 石橋 遼, 名古屋工業大学工学研究科情報工学専攻, 愛知県名古屋市昭和区御器所町, r.ishibashi.663@nitech.jp

2. モンテカルロ木探索と Novelty-Based Pruning

2.1 モンテカルロ木探索

モンテカルロ木探索 (Monte-Calro Tree Search: MCTS) [1][6] とは最良優先探索の 1 つであり、ランダムに行動を選択してそれを基に評価値を推定し木を拡張しながら探索していく木探索アルゴリズムである。MCTS は、まず現在の状態をルートノードとして設定し、ルートノードのみで構成された木を構築する。そして、「選択」、「シミュレーション」、「拡張」、「伝搬」の 4 つの過程を繰り返し行い、評価値を推定し徐々に木を拡張しながら探索を進めていく。そして、一定時間経つか最後のノード (葉ノード) を探索したら評価値の最も高いノードに遷移するように行動を決定する。以下に 4 つの過程それぞれについて詳しく説明する。

「選択」は、子ノードを追加して木を拡張するノードを選択する過程である。先ずルートノードから開始し、拡張の余地があるノード (そのノードにおいてとり得る行動の数より探索した子ノードの数が少ないノード) に到達するまでノードを選んでいく。ノードを選ぶ上では、推定評価値が高いノードを優先する一方で、探索の上で訪れた回数が少ないノードも選ばれなければならず、これら 2 つの要素のバランスが重要になる。これらのバランスが取れた選び方はいくつか提案されているが、その中で最もよく使用されているのが UCB1 アルゴリズム [1] である。これは、現在のノードを p、p の子ノードの集合を S として、以下の式の値が最大となるような子ノード $s \in S$ を選ぶという方法である。

$$v_s + C \times \sqrt{\frac{\ln n_p}{n_s}} \tag{1}$$

上記の式では、v_sは正規化した s の推定評価値 ($v_s \in [0, 1]$) 、n_p, n_sはそれぞれ p と s を訪れた回数、C は 推定評価値と訪れた回数のバランスを調整するための定数を表している。

「シミュレーション」では追加する予定のノードから仮想のノードを遷移していき、そのノードの推定評価値を計算する。

「選択」の過程で選択されたノードの子ノードの中で、未探索のものを 1 つ選んでそのノードから仮想葉ノード (これ以上拡張できないノード) か、指定した深さの仮想ノードに到達するまでランダムに行動を選択することで遷移していく。評価値の推定には s を追加する予定のノード、t を遷移を繰り返した際の末端の仮想ノード とし、以下の式を適応させる [6]。

$$v_s = \begin{cases} I + v_t & \text{if T is a winning node.} \\ -I + v_t & \text{if T is a losing node.} \\ v_t & \text{if T is a non-terminal node.} \end{cases}$$

上記の式に於いて v_sは s の推定評価値、v_tは t における価値、I は極端に大きい値 (v_tの最大値を v_{tmax}とすると $I \gg v_{tmax}$) を表し、t が望ましい仮想葉ノード (*winning node*) だった場合は v_tに I を加えた値、望ましくない仮想葉ノード (*losing node*) だった場合は v_tから I を引いた値、t が 仮想葉ノードでなかった (*non-terminal node*) 場合は v_tの値が s の推定評価値となる。

「拡張」ではノードを追加して木を拡張する。「シミュレーション」で推定評価値を求めたノードの情報を親ノードに追加し、その追加するノードに親ノードの情報を追加する。これによりノードの追加が行われ、木が拡張される。

「伝搬」では追加したノードの推定評価値をルートノードへ伝搬させる。「シミュレーション」で求めた追加したノードから推定評価値の和を取りながら逆方向に伝搬させる。そして、ルートノードの子ノードにその値を反映させる。このときの 子ノード c の新しい推定評価値 v_c^*は c の古い評価値を v_c、c から追加したノードまでの経路上のノードの集合を N、$n \in N$ の推定評価値を v_n とすると以下の式で表される。

$$v_c^* = v_c + \sum_{n \in N} v_n \tag{2}$$

以上の 4 つの過程を一定の時間内で繰り返し (図 1 参照) 、最終的なルートノードの子ノードの評価値を元に行動を選択する。そして、選択した行動によって遷移した先のノードを新たにルートノードに設定して再び探索する。

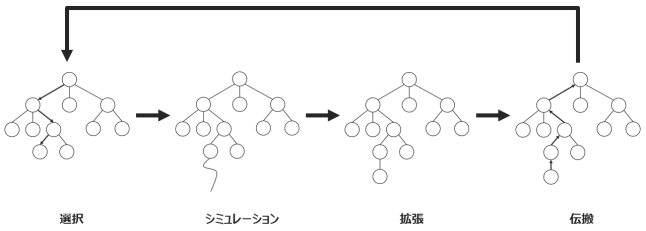

図 1: モンテカルロ木探索

2.2 Novelty-Based Pruning

Novelty-Based Pruning (NBP) [2][6] は MCTS から派生した探索アルゴリズムの 1 つで、先に探索した状態と似た状態が出た場合、それ以降は探索せずに新しい状態を優先して探索するというものである。このアルゴリズムは、使用する上で近傍ノードと新規度 (とそれに伴って新規性) を定義して、ノードの情報として蓄えさせる必要がある。

近傍ノードとは対象となるノードの新規度を計る為の比較対象となるノードの事であり、以下の 4 つのノードを指す。

1. 探索済みの兄弟ノード
2. 親ノード
3. 親ノードの兄弟ノード
4. 親ノードの近傍ノード

例として、図 2 ではノード S_iの近傍ノードを灰色のノードで表す。そしてその後にノード S_{i+1}を対象とする場合は、灰色のノードにノード S_iを加えたものが 近傍ノードとなる。

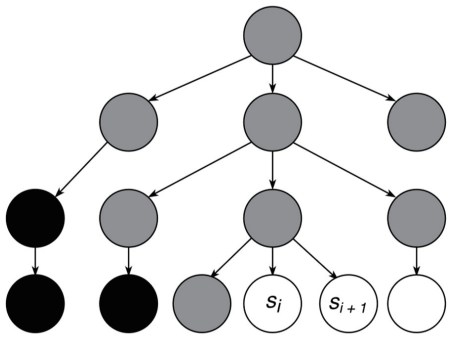

図 2: 近傍ノード

新規度とは、そのノードがどの程度新しいかを数値化したもので値が小さいほど新しい (新規性がある) ものであると判断される。新規度を計る方法は、先ず新規度を計るノードとそのノードの近傍ノードを 1 つずつ比較していく。それらのノードの状態を表す値を比較して、その値が一致するノードの数をカウントしていく。新規度は、近傍ノード中に存在する同じ値を持つノードの数+1 で計られる。例えば、図 2 の ノード S_i の状態を表現する値と同じノードが近傍ノード中に無かった場合、ノード S_i の新規度は 0+1 で 1 となる。この新規度が定めた閾値以下かそれより大きいかでノードの新規性を判断する。新規性はブール代数で表現され、新規性があると判断された場合は true, ないと判断された場合は false となる。

以上を踏まえて NBP の説明をする。新規度の計算及びノードの新規性の判断は MCTS の「選択」の過程にて実行される。「選択」の過程で拡張の余地があるノードが選択されたとき、「シミュレーション」に移行する前に、そのノードの全ての子ノードについて新規度を計る。それらの子ノードの中で、新規度が定めた閾値より大きいノードは新規性がないと判断 (値を false にする) し、「拡張」の過程で木に追加せず、「選択」の過程でも未探索のノードとして扱わない。そして新規度が定めた閾値以下の子ノードは新規性があると判断 (値を true にする) し、MCTS におけるその後の過程で探索の対象となる。後は基の MCTS と同様に新規性のある子ノードに対してシミュレートを行い、そのノードを追加して木を拡張させて、評価値をルートノードまで伝搬させる。そして一定時間繰り返した後に、求めた評価値を元に行動を選択する。そして、選択した行動によって遷移した先のノードを新たにルートノードに設定して再び探索する。なお、全ての子ノードで新規性がないと判

断された場合は、それらのノードの親ノード (「選択」の過程で選択されたノード) も新規性がないと判断し、新規性の値を false に変更する。

3. 提案手法

本研究では行動を決定して遷移する際に、遷移した先の状態をルートノードとして新しく設定する (図 3 左側) のではなく、その行動によって遷移する先のノードをそのままルートノードとして設定する (図 3 右側) 。

従来手法では、探索を開始する際に今の状態をノードで表現し、そのノードをルートノードに設定してルートノードのみから構成される木を構築する。そして、木を拡張しながら探索を進める。

今回提案する手法は、以下のとおりにルートノードを設定する際にその前の探索で決定した行動によってルートノードから遷移する先の子ノードを取り出し、そのノードが持っている情報を維持したままルートノードに設定する。

1. 再利用するための子ノードを見つける
2. 見つかった場合はその子ノードをルートノードに設定する
   (見つからなかった場合は新しくルートノードを設定する)
3. NBP を実行する

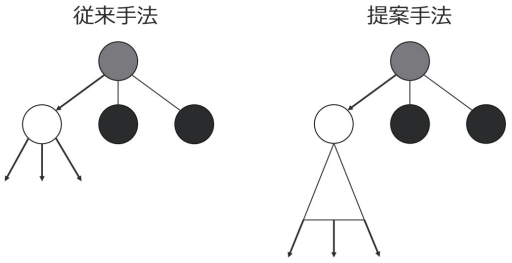

図 3: 従来手法と提案手法の比較
従来手法では遷移先のノードから探索を開始しているが提案手法では子木を再利用し子木の葉ノードから探索を開始している。

この手法では、状態の変化が前の状態に依って変化する (マルコフ性がある) 場合は遷移先のノードの子ノードが成す木 (子木) の持つ情報を失うことなく引き継ぐことができるため、子木を再利用することができるという利点がある。これにより、子木の葉から探索することができ、探索できなかった部分をより多く探索することが可能になる。

4. 実験

4.1 実験方法

本研究では、実際に NBP が使用されているゲームプレイング AI “MaastCTS2”[5] と、それに提案手法の処理を加えて改良したゲームプレイング AI “MaastCTS2_R” で、実際に 10 個の 1-Player ゲーム (1 人用のゲーム) をそれぞれ 10 回ずつプレイし、探索したノード数とゲームのスコアを比較する。

4.2 プレイするゲーム

プレイするゲームは GVGAI の競技会のページで配布されているフレームワークを利用する [3]。これは様々な種類の 2D ゲームをプログラムで扱えるように言語に変換したもので、AI の性能を評価するためのベンチマークとなる。各ゲームには自キャラクター (自分が操作するキャラクター) が存在し、エージェントはこのキャラクターを操作してゲームごとに異なる勝利条件を目指す。本研究では公開されている以下の 10 種のゲームを使用した： aliens, digdug, jaws, labyrinth, missilecommand, pacman, racebet, sheriff, survivezombies, waitforbreakfast。

ゲームを起動すると、エージェントは探索を開始するために 1 秒間の処理時間が与えられ、以降は 40 ミリ秒間隔 (この間隔を tick と呼ぶ) の探索する時間が与えられる。エージェントは各 tick の間に探索をしてどのように操作するかを決定し、その後実行してキャラクターを操作する。これをゲームが終了するまで繰り返しながらゲームを進めていく [4]。

このフレームワークが内包しているゲームは、そのゲーム自体の状況 (経過時間、現在のスコアなど) 、ゲーム内における各要素の状態 (自キャラクターの位置、体力、所持アイテムなど) 、現在の状態からとり得る操作などの情報が全て観測可能なものになっており、エージェントはこれらの情報を値のとして木探索におけるノードに蓄えることで、ゲームの各状態をノードとして扱うことが可能になっている [7]。例えば現在自キャラクターがいる座標を x, y の 2 次元の値で表現し、あるアイテムを所持しているということを 1 で表現すると、ノードには ((x, y) , 1) という情報が蓄えられる。

4.3 MaastCTS2

今回の実験で用いるゲームプレイング AI は 2016 年に Denis Soemers 氏が開発した “MaastCTS2” と呼ばれるものである。これはゲームにおける行動決定を NBP で行っている AI であり、2016 年度の GVGAI の競技会に参加しており、1-Player ゲームの部門において優秀な成績を残している [5]。

この AI では NBP で新規度を計る際に、ノードが持つ状態を表現する値 (現在いる座標、所持アイテムの有無など) の情報をまとめて 1 つの要素と見なし、それら全てが一致する場合のみ同一の状態を表現しているノードと見なす。そして新規性の判断基準は、新規度が 1 である、すなわち近傍ノード中に同一の状態を表しているノードがなかった場合のみ新規性があると判断する。

4.4 MaastCTS2_R

上述した MaastCTS2 に提案手法の処理を加えた AI が “MaastCTS2_R” である。この AI は MaastCTS2 においてルートノードを設定する処理の部分を、ノードが情報を維持したままルートノードに設定されるように改良した AI である。それ以外の部分に関しては変更はしておらず、基の MaastCTS2 と同じように処理をする。

4.5 実験結果と考察

表 1 にそれぞれの手法で実験した結果を示す。スコアと探索ノード数はそれぞれのゲームで 10 回プレイさせた際の平均値で、どちらも大きい方がより良い結果であることを示している。実験の結果から、全てのゲームにおいて提案手法の処理を加えた MaastCTS2_R の方が基となった MaastCTS と比較してより多くのノードを探索した。また、それと同時にゲームそのもののスコアも labyrinth を除いて MaastCTS2_R がよりよい結果を出している。

表 1: 実験結果

	MaastCTS2		MaastCTS2_R	
	スコア	探索ノード数	スコア	探索ノード数
aliens	28.8	302.2	36.4	336.3
digdug	2.9	71.7	5.9	75.4
jaws	1.4	77.2	2.9	78.2
labyrinth	0	331.2	0	382.5
missilecommand	-1.5	82.7	0.8	86.3
pacman	9.6	40.7	12.5	43.3
racebet	0.1	35.5	0.4	36.3
sheriff	-0.1	107.1	1.4	111.2
survivezombies	2044.2	187.5	2694.6	201.1
waitforbreakfast	0.2	27.1	0.4	35.0

しかし、表 2 が示すように探索ノード数については有意差がほとんど見られなかった。一方でスコアについては多少の有意差が見られたが、探索ノード数の差が有意でないことから、提案手法がスコアの上昇に寄与しているとは言い切れない。これは、tick という微小時間単位の中では再利用するノードの数が少なくなってしまうために、結果として両者の間に大きな差が出なかったと考えられる。

表 2: 各ゲームにおけるスコアと探索ノード数について、従来手法と提案手法による t 検定の結果（p 値）

	スコア	探索ノード数
aliens	0.2547	0.3194
digdug	0.0038	0.4736
jaws	0.0429	0.8269
labyrinth	-*	0.2811
missilecommand	0.0009	0.5173
pacman	0.1965	0.3031
racebet	0.1934	0.0457
sheriff	0.0811	0.6815
survivezombies	0.3595	0.4346
waitforbreakfast	0.4433	0.4263

(* : labyrinth のスコアは両手法で同じ結果だったため、p 値の算出はできなかった)

5. おわりに

本研究では、NBP における木探索部分で生じる無駄な探索を削減し、より多くのノードを探索できるようにすることを目的として、それを実現するために木を再利用するという手法を提案した。そして、その手法によって探索ノード数にどれほどの差が出るのかを観察するために、実際のゲームプレイング AI と比較を行った。また、フレームワークを利用することで実際にゲームをプレイさせ、ゲームの結果にどれほど影響するのかの確認も行った。

結果として探索ノード数はわずかに上昇したが、差は有意ではなかった。微小な時間単位の中では再利用するノードの数はそれほど多くはなく、再利用しても探索するノード数に大きな差は生じなかったと考えられる。また、スコアについても提案手法がスコアの上昇に寄与しているとは言い切れない。

今後の課題として、微小な探索時間の中でも有意差が生じるような手法を開発することが課題となる。また、探索時間を延長した場合に有意差は生じるのかを観察することも課題の 1 つである。

参考文献

[1] M. H. M. Winands, “Monte-Carlo Tree Search”, In *Encyclopedia of Compute Graphics and Games,* Springer, 2015.

[2] T. Geffner and H. Geffner, “Width-based Planning for General Video-Game Playing”, *Workshop of General Intelligence in Game-Playing Agents (GIGA),* 2015.

[3] The General Video Game AI Competition
`http://www.gvgai.net/`
(2019 年 1 月 23 日参照).

[4] D. Perez et al., “Open Loop Search for General Video Game Playing”, *Proc. 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO),* pp. 337-344.

[5] D. Soemers, “The General Video Game Agent MaastCTS2”,
`https://dennissoemers.github.io/jekyll/update/2016/09/29/the-general-video-game-agent-maastcts2.html`
(2019 年 1 月 23 日参照).

[6] D. J. N. J. Soemers et al., “Enhancements for Real-Time Monte-Carlo Tree Search in General Video Game Playing”, *Proc. 2016 IEEE Conference on Computational Intelligence and Games (CIG),* 2016.

[7] D. Ashlock et al., “General Video Game Playing Escapes the No Free Lunch Theorem” *Proc. 2017 IEEE Conference on Computational Intelligence and Games (CIG),* 2017