Automatic Goal Discovery in Subgoal Monte Carlo Tree Search

Dominik Jeurissen, Mark H.M. Winands, Chiara F. Sironi
Department of Data Science and Knowledge Engineering
Maastricht University
Maastricht, Netherlands
dominik.jeurissen@student.maastrichtuniversity.nl
{m.winands, c.sironi}@maastrichtuniversity.nl

Diego Perez-Liebana
Game AI Group
Queen Mary University of London
London, United Kingdom
diego.perez@qmul.ac.uk

***Abstract*—Monte Carlo Tree Search (MCTS) is a heuristic search algorithm that can play a wide range of games without requiring any domain-specific knowledge. However, MCTS tends to struggle in very complicated games due to an exponentially increasing branching factor. A promising solution for this problem is to focus the search only on a small fraction of states. Subgoal Monte Carlo Tree Search (S-MCTS) achieves this by using a predefined subgoal-predicate that detects promising states called subgoals. However, not only does this make S-MCTS domain-dependent, but also it is often difficult to define a good predicate. In this paper, we propose using quality diversity (QD) algorithms to detect subgoals in real-time. Furthermore, we show how integrating QD-algorithms into S-MCTS significantly improves its performance in the Physical Travelling Salesmen Problem without requiring any domain-specific knowledge.**

I. Introduction

Monte Carlo Tree Search (MCTS) [1] is a heuristic search algorithm that achieves good performance in a wide range of games without requiring any domain knowledge. Compared to algorithms like minimax, MCTS can often handle more complex games with a high branching factor. This can be attributed to MCTS investing the available computation-time to more promising trajectories instead of equally distributing it. However, as the complexity of a game grows, MCTS has to sample more trajectories to estimate an action's value accurately. This leads to the issue that MCTS tends to play nearly randomly in very complicated games.

Subgoal-MCTS (S-MCTS) [2] tries to solve this issue by focusing the value estimation on a small set of subgoal-states. Instead of optimizing the whole trajectory at once, a low-level search optimizes partial trajectories that lead to predefined subgoals. A high-level MCTS search then only chooses between found subgoals.

One problem with S-MCTS is that the developer must provide a subgoal-predicate. Not only does this make the technique domain-dependent, but it is also often quite challenging to provide a good subgoal-predicate. For example, if the distances between subgoals are too long, then the low-level search will have trouble finding any subgoals. Additionally, if some parts of the search space have no subgoals defined, S-MCTS will not consider this subspace at all. Although this might be a desirable property in some cases, it can also cause the algorithm to miss important parts of the search space.

To make S-MCTS domain-independent and improve its performance, this paper proposes using quality diversity search (QD-search) for finding suitable subgoals. In the following sections, we will review existing approaches for discovering subgoals and motivate why QD-search may be better suited for this. We then show how QD-search can be integrated into S-MCTS and finally analyze the resulting performance.

II. Background

A. Subgoal MCTS

In many environments, agents have to choose between numerous actions, making it infeasible to explore every individual action. Additionally, many actions may yield similar results, and as such, finding the optimal trajectory may not be necessary. Subgoal-MCTS (S-MCTS) [2] adapts MCTS to better work in these kinds of environments. It splits the problem into two sub-problems, a low-level search for finding short action sequences called macro-actions and a high-level MCTS-search that chooses between macro-actions.

The high-level MCTS search is mostly similar to standard MCTS. However, when traversing the tree, each edge now requires executing multiple actions instead of one, and it has to keep track of the accumulated reward while executing the macro-action. Additionally, the expansion step is replaced with the low-level search for finding suitable macro-actions.

The low-level search replaces the expansion step in traditional MCTS, and its goal is to search for macro-actions that lead to a state that fulfils a user-defined subgoal predicate. The purpose of the predicate is to identify states that are likely to lead to the final goal. Thus, whenever a node is expanded, the low-level search is executed until it finds a trajectory that leads to a new subgoal. The found trajectory is then added to the high-level search tree. If the search finds a trajectory to a previously discovered subgoal, it will replace the old trajectory with the new one if it has a higher reward. Trajectories are sampled by uniformly sampling actions until a subgoal or a predefined horizon is reached. It is important to note that the search does not have to find all subgoals at once since the expansion step only requires one macro-action at a time.

978-1-6654-3886-5/21/$31.00 ©2021 IEEE

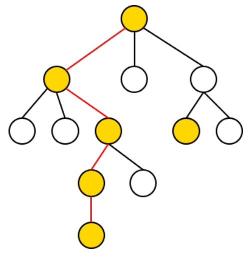
Fig. 1. Example of how optimal subgoals (Marked golden) can enable S-MCTS to find optimal trajectories (Marked red).

Once the search cannot find any new subgoals anymore, the corresponding node in the high-level tree is treated as fully expanded, and the search will not be executed anymore for that node.

The main idea behind this approach is that the low-level search can greedily optimize local trajectories, while the high-level search uses the greedy results to optimize globally. Furthermore, by splitting the search into multiple smaller ones, the branching factor is reduced significantly.

B. *Quality Diversity Search*

Many optimization algorithms focus on maximizing the performance of a behaviour. Instead, Quality Diversity (QD) [3] algorithms try to find as many diverse behaviours as possible while also seeking the behavioural niche's best individual.

QD-algorithms make use of a concept called behaviour characterization. The behaviour characterization is usually a vector representing the sequence of actions taken by an individual, but it can also include additional information. For example, if the goal is to navigate a maze, the behaviour-vector could be the agent's final position. Another vital part of QD-algorithms is an archive of previously discovered behaviours. With the archive, the novelty of a new behaviour can be computed by summing the distances to the nearest neighbours in the archive.

QD-algorithms seek to explore the entire behaviour space, which can help avoid deceptive fitness functions used in traditional optimization algorithms. Although this can be achieved using novelty alone [4], QD-algorithms also enforce a fitness constraint to find novel and high performing solutions.

III. Subgoal Discovery in Subgoal-MCTS

A. *Optimal subgoals*

The optimal subgoal is a state that lies on an optimal trajectory, meaning there is no trajectory with a higher return. In order to find optimal trajectories with S-MCTS, at least one optimal subgoal has to be found, given that the search starts from an optimal subgoal. Fig. 1 shows how this property enables S-MCTS to find optimal trajectories.

B. *Any subgoal is sufficient*

Existing approaches for subgoal discovery implicitly search for optimal subgoals. For example, in one approach, bottlenecks are used as subgoals [5]. Bottlenecks are states, which are frequently visited on highly rewarding trajectories, such as a door in a gridworld. Since bottlenecks have to be visited to reach the goal, they are automatically an optimal subgoal. Another approach is to find states with high empowerment [6]. Empowerment measures how much influence actions have on the future states that the agent can reach. High empowerment ensures that the agent has many options available, making it more likely to find an optimal trajectory.

Although these approaches are viable techniques for discovering subgoals, they either require much sampling or a learning algorithm. Since the goal is to use the discovered subgoals in S-MCTS, we cannot afford to spend much time detecting a single subgoal. However, it may not be necessary to search for optimal subgoals explicitly. Subgoals in S-MCTS are effectively a way to prune a bunch of trajectories. For example, given a room with two doors, we may define one subgoal for each door to sufficiently explore all rooms. By doing this, we prune away all trajectories except two. However, if only one trajectory is used, the high-level search loses the ability to explore the entire state-space. What that means is that we only have to find a set of trajectories that enable the high-level search to sufficiently explore the state-space. It does not even matter if the trajectory leads immediately to the door, as long as we eventually move through it.

The question that remains is how we can select subgoals in order to explore the state-space. Fortunately, quality diversity algorithms already solve this problem. Although QD-algorithms focus on finding diverse behaviours, we can always define the behaviour as the state reached by a trajectory. We can then use any QD-algorithm to find a set of trajectories that lead to novel states with high fitness. The resulting trajectories can then be used as subgoals in the S-MCTS search.

C. *Integrating Quality Diversity into S-MCTS*

In general, the S-MCTS algorithm stays the same, except that we replace the low-level predicate-based search with a QD-search. Furthermore, we assume that S-MCTS can access a behaviour-characterization function that maps a trajectory onto a latent space representing the behaviour.

For every leaf node in the high-level search tree, we store data for a low-level QD-search. The goal of the QD-search is to find a set of trajectories that lead to high rewards and novel states. Note that the novelty is measured locally, meaning subgoals from the high-level tree are not used in the low-level QD-search.

In comparison to the low-level search in S-MCTS, the QD-search does not return one subgoal per high-level iteration. Instead, we assume that the QD-search is an anytime algorithm, and whenever we visit a leaf node, the QD-search is advanced by one step. Once the QD-search search has been advanced sufficiently often, the search is considered complete, and the best trajectories will be added as subgoals to the leaf node. However, in S-MCTS the low-level search always returns a found subgoal for usage in the simulation and backpropagation step of the high-level search. Since we no longer return subgoals until the QD-Search search is completed, we instead assume that the QD-Search search samples precisely one trajectory per step. The state reached by the sampled trajectory is then used in the high-level search. Note that this may cause problems with the value estimation in the high-level search since most of the trajectories sampled by the QD-Search will not be used as subgoals.

D. Low-Level QD-Search

Because we use MCTS for the high-level search, we decided to use it also for the low-level search. To ensure that the low-level MCTS search can find novel trajectories, we use an adapted reward function. Concretely we use as a reward the latent distance to the subgoal node that contains the low-level search. Given a distance-metric $d(s_1, s_2) \to \mathbb{R}_0^+$, and the behaviour vector of the subgoal node s_g we define the reward at each timestep as follows:

$$r(s_i, s_{i+1}) = d(s_{i+1}, s_g) - d(s_i, s_g) \tag{1}$$

The reward function is positive when the distance to the high-level subgoal increases and becomes negative if we get closer. Note that the function does not consider how long it takes to reach a certain distance, resulting in idling where the agent moves back and forth between two positions. That is why we use a discounted sum to accumulate all rewards because a discount factor below 1 will make idling result in a lower score overall. This procedure results in an MCTS algorithm that focuses on finding trajectories that quickly move away from the high-level subgoal.

As described in Section III-C, once the low-level search was advanced a set amount of time, it has to return a set of subgoals. Algorithm 1 contains the pseudocode for the subgoal selection. It works by selecting a set of subgoal candidates from the low-level MCTS tree, which are all trajectories with a specific length. The length is kept the same to make the comparison easy. Subgoals are then iteratively added to an archive by evaluating candidates using a relative novelty and reward score. We use the same scores as used in NOVELTY SEARCH WITH LOCAL COMPETITION [7]. The novelty score is the sum of distances to the k nearest subgoals, and the reward score is the number of k nearest subgoals with a lower reward than the candidate. Both scores are normalized between 0 and 1 and then combined to a final score using a weighted sum with $\alpha = 0.5$. We add subgoals until we selected a percentage of all available candidates. For this paper, we select 2% of the candidates.

Algorithm 1 Subgoal selection

```
Require:
    Root of the low-level MCTS search tree $r$
    Length of a macro action $L$
    Number of neighbors $k$ for novelty and reward score
    Percentage $p$ of trajectories to select
    Weighting for rewards $\alpha$
 1: procedure SELECTSUBGOALS
 2:     $cand \leftarrow SelectNodes(r, L)$                        ▷ Subgoal candidates
 3:     $c_{max} \leftarrow c \in cand$ with highest reward
 4:     $subgoals \leftarrow \{c_{max}\}$                          ▷ Start with best candidate
 5:     $sCount \leftarrow p \cdot |cand|$                         ▷ Subgoal Count
 6:     while $|subgoals| < sCount$ do
 7:         for $c_i$ in $cand$ do                                  ▷ Update scores
 8:             $neighbors \leftarrow kClosest(subgoals, c_i, k)$
 9:             $rScore \leftarrow rewardScore(neighbors, c_i)$
10:             $nScore \leftarrow noveltyScore(neighbors, c_i)$
11:             $assignScore(c_i, \alpha \cdot rScore + (1 - \alpha) \cdot nScore)$
12:         $c_{best} \leftarrow c \in cand$ with highest score
13:         $subgoals \leftarrow subgoals \cup \{c_{best}\}$
14:     return $subgoals$
```

IV. EXPERIMENTS AND RESULTS

A. Physical Travelling Salesman Problem

To evaluate the performance of the proposed algorithm, we used the framework for the Physical Travelling Salesman Problem (PTSP) [8] competition[1]. In the competition, an agent has to steer a ship to reach ten waypoints scattered around a map. An agent has 1000 steps to reach a waypoint. After reaching a waypoint, the time limit is reset back to 1000. To move, the agent can control the ship's thruster and rotate the ship left and right. Although this results in only six actions total, it takes many steps to move between waypoints. In fact, the search depth required for this environment is so high that we had to lower it by repeating a chosen action fifteen times before moving to the next one.

Many strong solutions have already been proposed for this environment, some of which can reliably reach all waypoints [9]. In general, these approaches use a domain-independent algorithm that handles the ship's steering, but the order of waypoints is precalculated and integrated into the rewards. These solutions also repeat an action to lower the search depth.

B. Steering

For this paper, we evaluate the agents without providing them with any domain knowledge. The used reward function returns 10 when a new waypoint is reached and otherwise -1. The algorithms used are the vanilla MCTS algorithm, S-MCTS, and the proposed QD-S-MCTS algorithm. Note that all algorithms have a budget of 50000 forward model calls, and they all reuse the search tree. All hyperparameters were handpicked.

S-MCTS is given access to a subgoal predicate that evenly distributes a grid of subgoal points over the map. The predicate detects a subgoal whenever a ship is in a certain radius to a subgoal point. The velocity of the ship is ignored when detecting a subgoal. The behaviour vector used for QD-S-MCTS contains only the ship's position. All algorithms are evaluated ten times on each of the ten different maps from the PTSP competition, ranging from an open area with no obstacles to a maze. The starting position for each map is always the same. However, ties in the selection phase are resolved randomly and will behave differently for each run.

Fig. 2 and Fig. 3 contain the results of the experiments. The results indicate that the QD-S-MCTS algorithm outperforms all the other tested algorithms. Overall QD-S-MCTS performed

[1]The source code, as well as all experiments, can be found at: https://github.com/CommanderCero/AutoSubgoalMCTS_PTSP

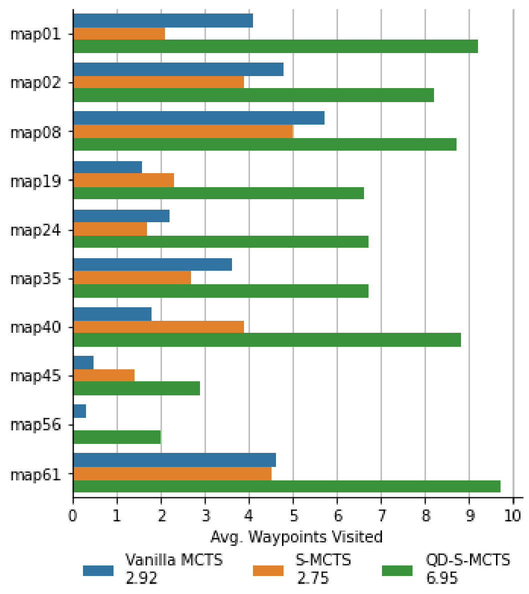

Fig. 2. The average number of waypoints an algorithm found on each map. The legend shows the average across all maps.

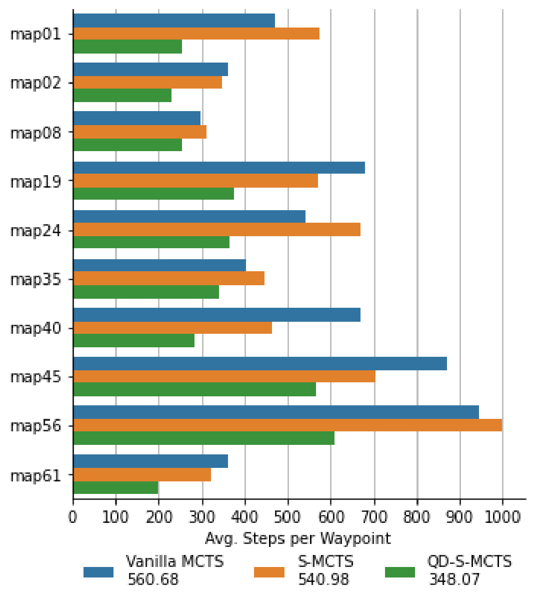

Fig. 3. The average number of steps an algoritm needed to reach a waypoint. The legend shows the average across all maps.

best on maps with wide-open areas. In these cases, the subgoals are spread evenly around the map, allowing the search to find waypoints far away from the ship's current position. However, QD-S-MCTS struggled especially with maps with narrow corridors, like Map45 containing a cave system and Map56 which is a maze. What often happens in these maps is that all found subgoals are clumped together in a single area. This may happen because navigating the map is difficult, and it is easier to just move the ship back and forth. This problem may be solved by integrating the previously visited subgoals into the novelty computation. Another problem that often occurs is that when no new waypoints can be found anymore, and QD-S-MCTS must rely on luck to move in the correct direction. While this is unavoidable initially, integrating some form of novelty reward into the high-level search may also help with this problem.

At last, the main reason why S-MCTS failed to outperform Vanilla MCTS is that it treats all states beyond a specific horizon as subgoals, resulting in a huge branching factor. Concretely, whenever S-MCTS searches for new subgoals, it will stop the search after a set horizon. However, it still treats horizon-states as subgoals, resulting in an algorithm that treats nearly all states as subgoals. Essentially, the horizon-states make it difficult for S-MCTS to leverage the pruning that the subgoals provide.

V. Conclusion and Upcoming Work

In this paper, we proposed a new domain-independent search algorithm called QD–S-MCTS, which replaces the predicate-based subgoal search from Subgoal-MCTS with a quality diversity search. Results show that the proposed algorithm achieves high performance in an environment where the most optimal trajectory is often very hard to find.

As of now, we have used a modified MCTS algorithm for the quality diversity search. As future research, we will investigate the impact of other quality diversity algorithms on the performance of QD-S-MCTS. Additionally, we only tested how QD-S-MCTS performs without any domain knowledge. As a next step, it is also interesting to see if the shown results still hold if we introduce domain knowledge.

References

[1] L. Kocsis and C. Szepesvári, “Bandit based monte-carlo planning,” in *Machine Learning: ECML 2006*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 282–293.

[2] T. Gabor, J. Peter, T. Phan, C. Meyer, and C. Linnhoff-Popien, “Subgoal-based temporal abstraction in monte-carlo tree search,” in *Proc. of the Twenty-Eighth Int. Joint Conf. on AI, IJCAI-19*, 7 2019, pp. 5562–5568.

[3] J. K. Pugh, L. B. Soros, and K. O. Stanley, “Quality diversity: A new frontier for evolutionary computation,” *Frontiers in Robot. and AI*, vol. 3, p. 40, 2016.

[4] J. Lehman and K. Stanley, “Exploiting open-endedness to solve problems through the search for novelty,” *Artificial Life - ALIFE*, 01 2008.

[5] A. McGovern and A. G. Barto, “Automatic discovery of subgoals in reinforcement learning using diverse density,” in *Proc. of the Eighteenth Int. Conf. on Machine Learning*, ser. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, p. 361–368.

[6] K. Gregor, D. J. Rezende, and D. Wierstra, “Variational intrinsic control,” *CoRR*, vol. abs/1611.07507, 2016. [Online]. Available: http://arxiv.org/abs/1611.07507

[7] J. Lehman and K. Stanley, “Evolving a diversity of creatures through novelty search and local competition,” in *Genetic and Evol. Comput. Conf., GECCO'11*, 01 2011, pp. 211–218.

[8] D. Perez, P. Rohlfshagen, and S. M. Lucas, “The physical travelling salesman problem: WCCI 2012 competition,” in *2012 IEEE Congr. on Evolutionary Comput.*, 2012, pp. 1–8.

[9] D. Perez, E. J. Powley, D. Whitehouse, P. Rohlfshagen, S. Samothrakis, P. I. Cowling, and S. M. Lucas, “Solving the physical traveling salesman problem: Tree search and macro actions,” *IEEE Trans. on Comput. Intell. and AI in Games*, vol. 6, no. 1, pp. 31–45, 2014.