A Monte-Carlo Path Planner for Dynamic and Partially Observable Environments.

Munir Naveed, Diane Kitchin, Andrew Crampton, Lukáš Chrpa, and Peter Gregory

***Abstract*—In this paper, we present a Monte-Carlo policy rollout technique (called MOCART-CGA) for path planning in dynamic and partially observable real-time environments such as Real-time Strategy games. The emphasis is put on fast action selection motivating the use of Monte-Carlo techniques in MOCART-CGA. Exploration of the space is guided by using corridors which direct simulations in the neighbourhood of the best found moves.**

MOCART-CGA limits how many times a particular state-action pair is explored to balance exploration of the neighbourhood of the state and exploitation of promising actions. MOCART-CGA is evaluated using four standard pathfinding benchmark maps, and over 1000 instances. The empirical results show that MOCART-CGA outperforms existing techniques, in terms of search time, in dynamic and partially observable environments. Experiments have also been performed in static (and partially observable) environments where MOCART-CGA still requires less time to search than its competitors, but typically finds lower quality plans.

INTRODUCTION

Monte-Carlo Policy Rollout algorithms [1] [2] [3] approximate action values using Monte-Carlo simulations. Monte-Carlo simulations require a model that can simulate the effects of an action using a state transition function and assign a reward for the state-action pair. These algorithms run several rollouts (or iterations from the current state) to approximate the values of the action applicable at the current state and select the action that has the highest value compared with other actions applicable at that state. These algorithms have been successfully applied to solve planning problems in deterministic domains (e.g. in Go [4]) and non-deterministic domains (e.g. Solitaire [5]).

Monte-Carlo based algorithms are also suitable for online planning in dynamic and partially observable domains with large state spaces such as real-time strategy (RTS) games. The domain world of an RTS game requires online planning within tight time constraints (e.g. there is a time-limit of 1-3 milliseconds per planning search in the titles produced by the game company Bioware [6]). In an RTS game (e.g. WarCraft™and StarCraft™), the computer player requires AI planning to solve different tasks (e.g. path planning, resource management and tactical assault planning). Path planning is one of the most challenging tasks in an RTS game because it is required by all movable characters in a game. Many pathfinding approaches, for example A* [7], navigational mesh [8] and way-points [9], are not suitable due to partial observability and real-time constraints.

This paper addresses the single-agent path planning problem in a RTS gaming environment where the topology of the game world changes during the problem solving process. The environment is partially observable, i.e. the planning agent can see only a small part of the topology of the game world at any time during the problem solving process. The main challenging issues for a path planner in this kind of environment are incomplete information about the domain world, hard real-time constraints, a non-deterministic action model and dynamic changes in the domain world.

We present a Monte-Carlo policy rollout algorithm that performs a rollout under a tight time bound and applies a selective action sampling at each state of the look-ahead search. To balance the trade-off between the exploration of new actions and exploitation of the best action, we define the convergence of an action value by imposing a limit on the number of times the action value remains unchanged during simulations. If an action value exceeds the limit, then the action is assumed to be converged with respect to the limit. Converged actions are not sampled during the simulations. The exploration of new actions in a simulation is limited within a group of actions (called a corridor) which are local to the current best action. This ensures that sampling is performed in the neighbourhood of current promising solutions. The algorithm is called MOnte-CArlo Real-Time Corridor based Greedy Action sampling (MOCART-CGA).

We present empirical results that demonstrate MOCART-CGA has stronger performance than a state-of-the-art Monte-Carlo path planner [10], both in time to search for the next action and in plan quality. We also demonstrate that MOCART-CGA takes a smaller amount of time to search than current state-of-the-art path planning algorithms [11], [12]. In these cases, the plan quality of MOCART-CGA is comparable, and in certain cases higher. By searching faster, MOCART-CGA is a more responsive real-time algorithm than its competitors.

RELATED WORK

Tesauro [2] explores a policy rollout algorithm in a stochastic board game called Backgammon. The simulation model in Tesauro's work takes several iterations per move to decide an action at the current move. This approach is expensive for a real-time application. Kearns et al. [13] present a sparse sampling based approach that generates a look-ahead tree of fixed depth H but each action applicable at a state (seen during the look-ahead search) is sampled C

Diane Kitchin, Andrew Crampton, Peter Gregory and Lukáš Chrpa are in School of Computing and Engineering, University of Huddersfield, UK.

Munir Naveed is at Department of Software Engineering, Fatima Jinnah Women University, Pakistan, Contact email: munir@fjwu.edu.pk;

times in a simulation. The number of samples in a simulation is exponential in H. To avoid exploring all actions in a sparse sampling scheme, Auer et al. [14] demonstrate an adaptive action sampling approach (called Upper Confidence Bounds or UCB) that selects only one action per state in the look-ahead search in a simulation. It selects the best action as a sample at a state. To balance the exploration of new actions and exploitation of the best action, Auer et al. present upper bounds on the selection of an action as a sample at a state. However, this approach continues the exploration of new actions forever.

Kocsis and Szepesvári [3] present a variation of UCB, called Upper Confidence Bounds applied to Trees (UCT), that performs selective action sampling in a policy rollout fashion. The default rollout policy is random. UCT has been successful in Go [4] and Solitaire [5] games. However, UCT and UCB are only applicable in a domain if the action values are in the range [0,1]. Balla et al. [15] present a variation of UCT in an RTS game to solve the tactical assault problem. The variation of UCT uses a reward function that can optimise one parameter (time or health factor) in a simulation model.

The concept of using focussed rollouts in UCT has been explored by Laviers and Sukthankar [16] in a real-time stochastic but fully observable and static domain world, called Rush Football Simulator. In this domain world, two teams play football on a rectangular field of fixed size where the topology of the field remains the same during a game. In [16], the UCT simulations determine the best action for each key player in a subgroup of a team. The UCT rollouts for each player are kept focussed in a small region. The focussed region is determined offline using a large set of training examples. The approach also has to re-compute the focussed region of each player if the playing field is changed or if the game is played on a new field. In MOCART-CGA, the rollouts are focussed using a corridor of actions which is the same for all planning agents and does not depend on the topology of the map. Therefore, MOCART-CGA does not need to re-compute a corridor if a map is changed.

CadiaPlayer [17] is a variation of UCT that has been explored in general game playing where each game world is represented in a first-order logic language. CadiaPlayer has been successful in general game playing competitions. MOCART-CGA is similar to CadiaPlayer in the sense that the action values are maximised to synthesize a plan, however, CadiaPlayer also uses the average action values to guide the look-ahead search towards potentially useful but unexplored actions.

The MOCART-CGA planner has similarities with real-time heuristic search planning algorithms in the sense that these algorithms interleave planning and plan execution e.g Local Search Space-Learning Real-Time A* (LSS-LRTA) [11] and Real-Time D*-Lite (RTD) [12]. LSS-LRTA and RTD solve path planning problems under tight real-time constraints and interleave planning and plan execution. LSS-LRTA expands the look-ahead search of fixed depth using A* and updates the heuristic value of each state of the look-ahead search by using Dijkstra's shortest-path algorithm [18]. RTD is a combination of LSS-LRTA and D*-Lite [19]. D*-Lite is a global path planning algorithm with a backward incremental heuristic search.

aLSS-LRTA [20] is a modification of LSS-LRTA such that the greedy action selection is avoided if the planner gets stuck in a heuristic depression. aLSS-LRTA has been evaluated using the static settings of the pathfinding benchmarks. The results show that aLSS-LRTA produces solutions with better quality than LSS-LRTA.

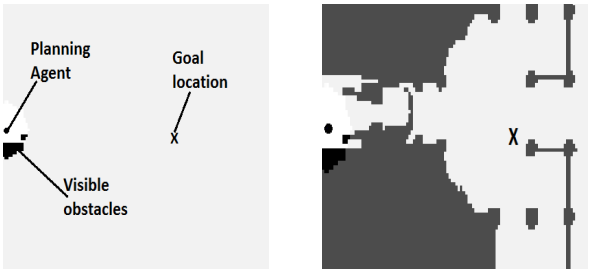
f-LRTA* [21] is a variation of Learning Real-Time A* [22] that updates the estimate of the cost of reaching the goal location along with learning the cost of moving from the start location to the current state. f-LRTA* is similar to LSS-LRTA in terms of learning the heuristic values. The results show that f-LRTA* expands fewer states than LSS-LRTA to solve path planning problems.

The MCRT planner [10] is a Monte-Carlo Planner based on Rapidly-exploring Random Trees (RRT) [23], applied to an RTS game. MCRT performs better than LSS-LRTA for solving path planning problems in a typical RTS game when collecting resources. The advantage gained by MCRT in these problems arises from the fact that as the resources are collected, the same parts of the map have to be repeatedly traversed. MCRT, therefore, is more effective than LSS-LRTA for multiple-journey path planning problems.

Problem Description

In this paper, we address the single-agent path planning problem where a planning agent is equipped with very limited information about the game world. The planning agent has a sensor that can provide local information in the current neighbourhood of the agent. This sensor information contains the current position and velocity of the planning agent, the position of the goal state(s), and the set of locations (and their status) that are within the line of the sight of the planning agent. The status of a neighbouring location can be either occupied or empty. If a location is passable, it is called empty otherwise it is occupied. The planning agent also knows about the size of the world and the set of actions it can perform. The topology of the world can change during problem solving. There are two kinds of obstacles in the domain world: static obstacles and dynamic obstacles. Static obstacles stay in one place for the whole game while dynamic obstacles are the movable characters in the game. The presence of dynamic obstacles make the action model non-deterministic. The main challenging issue for a path planner in this kind of setting is to respond within a short time to solve a global path planning problem in an initially unknown and dynamically changing game world. Figure 1 shows an example of a planning problem in a partially observable environment of Arena2 (a benchmark). Figure 1(a) shows the agent's viewpoint at the start of planning where only a small part of the map is visible to the agent. The actual topology of the game world for the same problem is shown in Figure 1(b). With such limited visibility, the

planning agent is also bound to respond within a fixed time interval (no matter how small the interval is). The main motivation of using the Monte-Carlo planning techniques in this kind of environment is to approximate an answer for a given planning problem by running simulations within a predefined time-limit.



(a) Agent's viewpoint (b) Actual topology

Fig. 1. A viewpoint of a planning agent in a partially observable environment of Arena2.

NOTATION

We are interested in solving path planning problems in a domain world that is partially visible, real-time and dynamic (like a typical RTS game). The planning agent knows about its current location and the goal location. Initially, the planning agent can see only a limited part of the domain world near its current position.

A state is constructed using a rectangular octile grid of size $w \times h$ (w is the width and h is the height) where each cell of the grid is either empty, blocked or unknown in a state. A cell is unknown only until the agent sees the cell (the cell is in the agent's visibility range). Then the cell can be declared blocked or empty and the agent is aware of the status of the cell even though the cell is out of the agent's visibility range. The number of unknown cells is therefore monotonically decreasing.

As the map is an octile grid, the possible actions are moving to one of the eight compass directions, (N, NE, ..., NW). The effects of actions are non-deterministic due to the dynamic changes in the domain world during problem solving.

Definition 1
The domain world has a finite set of states S and a finite set of actions A.

Definition 2
$App(s)$ is a function ($App : S \rightarrow 2^A$) that determines a set of applicable actions at s.

Definition 3
$Next(s, a)$ is a function ($Next : S \times A \rightarrow 2^S$) that defines the set of states which are possible to reach from s by taking action $a \in App(s)$.

Definition 4
$P(s, a, s')$ is a function ($P : S \times A \times S \rightarrow [0, 1]$) that gives the probability of reaching a state s' from a state s if action a is taken at s.

Definition 5
$Transition(s, a)$ is a stochastic transition function ($Transition : S \times A \rightarrow S$) that selects a next state $s' \in Next(s, a)$ for a state-action pair (s, a) using the probability distribution P. $Transition$ prioritises the selection of the most likely next state.

Definition 6
$R(s, a)$ is a reward function ($R : S \times A \rightarrow \mathbb{R}$) that assigns a value to a state action pair (s, a) in a simulation model.

MOCART-CGA uses a simulation model of the domain world. The simulation model uses the function $Transition$ to randomly select the next state s' of a state-action pair (s, a) for all $s \in S$ and $a \in App(s)$ and then uses R to assign a reward value for the pair. If P is not available in a domain, MOCART-CGA updates the probabilities using the online interaction with the environment (as shown in Figure 5). The probability of moving to an occupied state s_x from s with any action $a \in App(s)$ is always zero i.e. $P(s, a, s_x) = 0$.

$V(s)$ is the state value and $Q(s, a)$ is the estimated value for action $a \in App(s)$ at state s. These values are computed using the simulation model of MOCART-CGA. The best action $a_b \in App(s)$ at s has the highest estimated value at s. It is computed using equation (1) (adapted from [2]).

$$a_b = \arg\max_a(Q(s, a)) \quad (1)$$

CORRIDORS IN MONTE-CARLO PLANNING

The main contribution of this work is to introduce the notion of a *corridor* to Monte-Carlo based path planning. Corridors are a method for directing simulations in a way that creates diversity in the selected actions and that restricts undesirable artifacts (such as cycles) from simulations. The concept of a corridor relies on the underlying domain having a concept of 'orientation' or another measure of action relatedness: path-planning domains are therefore obvious candidates.

After an action, a, is executed within a simulation, the subsequent action must be drawn from a subset of the applicable actions. In our work, this subset is determined by the orientation of the agent after executing action a. The following action must be in the same direction, or deviating by only a small, fixed amount. However, in general, this subset of applicable actions may be computed by any function that determines relatedness of two actions.

We now define the concepts of a corridor and relatedness more formally:

Definition 7
A relatedness function is defined as a function $Rel : S \times A \times A \rightarrow \{0, 1\}$. $Rel(s, a, a')$ returns 1 if and only if a and a' are related by some definition in state s.

Given this definition of relatedness, we can now define what we mean by a corridor:

Definition 8
A corridor $A_c(a, s)$ is a function ($A_c : A \times S \rightarrow 2^A$) that determines a restricted set of actions relevant to a (in s) such that $A_c(a, s) = \{a' | a' \in App(s), Rel(s, a, a') = 1\}$.

The actions are grouped in the form of a corridor to restrict the possible action choices and therefore speed up exploration of new actions in simulations during online planning. In our work, we define relatedness as the closeness of two compass directions. Recall that all actions in our domain are selected from the eight compass directions. An action is related to the set of three actions that contain a and the actions to each side of a. For example, the related set of an action "NORTH" is the set {"NORTH-WEST", "NORTH", "NORTH-EAST"}. Note that in a different problem domain, a different relatedness function would be relevant. For example, in trajectory planning, speed limits and other factors would need to be taken into account.

Corridors in MOCART-CGA

One motivation for the use of corridors in planning is that each corridor creates a simulated path that continually extends from the current state, and this leads to a greater number of new actions being sampled. In Monte-Carlo simulations, the exploration of new actions is an important and essential part of the search because action values are estimated using locally sampled information. More diverse sampling leads to more information per simulation.

However, indiscriminate exploration of new actions leads to the sampling of many actions that are not useful to solve the current planning problem; this process is computationally expensive within the planning search process and more simulations are required to find useful information. It is therefore useful to restrict the choices of actions to explore promising areas of the search space. This is the key function of corridors. In MOCART-CGA, the exploration of new actions is limited to be within the corridor of the best action at any state seen during the look-ahead search. One side-benefit of using this type of simulation is that simulations avoid looping behaviours. This is because each simulation follows a trajectory restricted by a corridor, which removes the option to return to the same state.

The exploration scheme in MOCART-CGA also gives importance to actions that are less explored by increasing their chance of selection. The corridors are kept overlapping i.e. two corridors (each one by a different action) can have one or more common actions, so the exploration of the new actions can move from one corridor to another in two consecutive simulations.

MOCART-CGA

MOCART-CGA uses a Monte-Carlo strategy for selecting the best action in a current state s. For a limited number of times ($limit$) it performs a greedy look-ahead search from the current state s to a fixed depth (d). The search is guided by using corridors restricting sets of applicable actions in order to keep a direction of movement to avoid inefficient zig-zag moves. For each state-action pair visited during the look-ahead search, MOCART-CGA computes reward values using R function (see Figure 4). If the depth d is reached then the state in this depth is evaluated using an admissible heuristic based on actual distance to the goal. The estimated value of a state-action pair $s \times a$, i.e. $Q(s, a)$, is computed as a sum of all reward values of state-action pairs explored during the search including the reward for the state in the depth d. $Q(s, a)$ stands for the expected long term reward of the action a sampled in the current state s. If the newly computed value of $Q(s, a)$ is greater than the current value of $Q(s, a)$ then the value is updated otherwise it remains unchanged. If $Q(s, a)$ consecutively remains unchanged for n_{limit} times, then we say that an action a is *converged* in a state s. Converged actions are no longer explored by MOCART-CGA which means that the estimated value for such state-action pairs cannot be modified at this stage. We say that a state s is *converged* if all the applicable actions in s ($App(s)$) are converged. If s is converged, then MOCART-CGA selects the best action a at s (i.e., where $Q(s, a)$ is the highest).

Overview

The look-ahead expansion of a current state in MOCART-CGA is based upon a Monte-Carlo tree search paradigm [24]. The number of expansions of the current state is limited by a predefined constant. MOCART-CGA selects an action in each state visited during the look-ahead search until the search reaches a predefined depth.

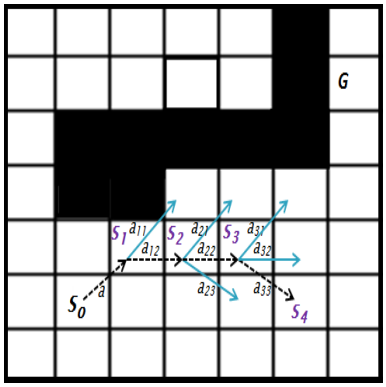


Fig. 2. An example of an iteration of MOCART-CGA.

An example of an iteration of MOCART-CGA is shown in Figure 2 on a grid of size 7×7 cells. A blank cell represents a walkable position and the black cells represent obstacles. The example shows a look-ahead search of depth four. s_o represents the initial state and G is the goal state of the planning problem. MOCART-CGA randomly chooses an action a in s_o. The result of applying a in s, a state s_1, is estimated using the transition function i.e. $Transition(s_o, a)$. A corridor of actions is created in s_1 which includes the actions a_{11} and a_{12} such that $a_{11}, a_{12} \in A_c(a, s_1)$. MOCART-CGA randomly selects an action (say a_{12}) from the corridor and moves to a state s_2 by using $Transition(s_1, a_{12})$. MOCART-CGA randomly selects an action (say a_{22}) in the state s_2 from a corridor $A_c(a_{12}, s_2) = \{a_{21}, a_{22}, a_{23}\}$. The same process continues until the look-ahead search reaches a predefined depth (i.e. four) and the 'leaf' state s_4 is evaluated using the reciprocal of the shortest distance estimate between

s_4 and G. The cumulative reward is calculated as a sum of the rewards of all state-action pairs seen during the look-ahead search and the evaluation of the 'leaf' state (s_4). If the cumulative reward is greater than $Q(s_o, a)$, then $Q(s_o, a)$ is updated otherwise it remains unchanged. If $Q(s_o, a)$ remains unchanged for a given number of consecutive iterations then a is not selected in s_o in future expansions. If s_o is visited for the first time, then each applicable action in s_o is explored at least once.

Algorithmic Details

The algorithm MOCART-CGA is depicted in Figure 3. If the current state s_c is converged then MOCART-CGA returns the best action applicable in s_c, i.e with the highest action value (Line 2). Otherwise MOCART-CGA performs a $limit$ number of rollouts to estimate the action values in s_c. In each rollout, MOCART-CGA randomly chooses an action $a \in App(s_c)$ which has not yet been sampled or which has been sampled the least number of times (Line 6). The next state s_n (a result of applying a in s_c) is estimated by using the stochastic transition function $Transition(s_c, a)$ (Line 7). The immediate reward $R(s_c, a)$ of the state-action pair (s_c, a) is computed and stored in r_n (Line 8). s_n is expanded to a depth $depth - 1$ in the following way. Choosing an action in s_n (Line 10) is done in a similar way as before (choosing a in s_c). If the chosen action a' has already been sampled in s_n in the previous searches, then it randomly selects an action from the corridor of a (i.e., $A_c(s_n, a)$). Selection is done in the same way as before (e.g. Line 6). The immediate reward of the state-action pair (s_n, a) is computed and added to r_n (Line 13). The next state s_{next} is determined by again using the $Transition$ function (Line 14 and then $s_n := s_{next}$ (Line 15). The expansion phase lasts for $depth - 1$ iterations (the FOR loop). At the depth $depth$ the leaf node s_n is evaluated using the distance estimation to the goal $dist(s_n, g)$. The long term reward r_n is then increased by $1/dist(s_n, g)$ (Line 17). If r_n is greater than $Q(s_c, a)$, then $Q(s_c, a)$ is updated, i.e., $Q(s_c, a) = r_n$ (Lines 19-20). Then MOCART-CGA performs another rollout until the number of performed rollouts reaches $limit$ (s_c is the start node for each rollout). At the end, MOCART-CGA selects the best action in s_c (Line 23) and returns it for execution by the planner (Line 24).

The R function (Figure 4, adapted from [10]) computes the immediate reward of a state-action pair. At first, R estimates the result of applying an action a in a state s_n by using the stochastic function $Transition(s_n, a)$ (Line 1) ($s_{next} = Transition(s_n, a)$). The immediate reward is computed as a fraction where the numerator consists of a number of states possibly reachable from s_n by applying a ($P(s_n, a, s_t) > 0$) and the denominator is computed as a product of a scaling factor W_d and an estimated distance from s_n to the goal $dist(s_n, g)$. The scaling factor W_d normalizes the relationship between collision-free and 'real' paths to the goal.

Function $MOCART - CGA(s_c, g)$
Read access $depth, limit, n_{limit}$;
1. IF s_c is converged
2. $a := \arg\max_{\mathrm{a \in App(s_c)}} \mathrm{Q(s_c, a)}$;
3. RETURN a;
4. ELSE
5. REPEAT
6. $a := RandomChoice(\{a \mid a \in App(s_c), \forall a' \in App(s_c) : n(s_c, a) \leq n(s_c, a')\})$;
7. $s_n := Transition(s_c, a)$;
8. $r_n := R(s_c, a)$;
9. FOR: $i = 1$ to $depth - 1$
10. $a' := RandomChoice(\{a \mid a \in App(s_n), \forall a' \in App(s_n) : n(s_n, a) \leq n(s_n, a')\})$;
11. IF $n(s_n, a') > 0$ THEN
12. $a' := RandomChoice(\{a_c \mid a_c \in A_c(a, s_n), \forall a'_c \in A_c(a, s_n) : n(s_n, a_c) \leq n(s_n, a'_c)\})$;
13. $r_n := r_n + R(s_n, a')$;
14. $s_{next} := Transition(s_n, a')$;
15. $s_n := s_{next}$;
16. END FOR
17. $r_n := r_n + 1/dist(s_n, g)$;
18. $n(s_c, a) := n(s_c, a) + 1$;
19. IF $Q(s_c, a) < r_n$ THEN
20. $Q(s_c, a) := r_n$;
21. $limit - -$;
22. UNTIL $(limit > 0)$;
23. $a := \arg\max_{\mathrm{a \in App(s_c)}} \mathrm{Q(s_c, a)}$;
24. RETURN a
End $MOCART - CGA$

Fig. 3. MOCART-CGA abstract algorithm

Function $R(s_n, a)$
1. $s_{next} := Transition(s_n, a)$;
2. $rw := \frac{\|\{s_t : P(s_n, a, s_t) > 0 \, \forall s_t \in Next(s_n, a)\}\|}{W_d * dist(s_{next}, g)}$;
3. RETURN rw
End R

Fig. 4. Reward Function

PATH PLANNER

MOCART-CGA is embedded in a real-time planner. The real-time planner interleaves planning and plan execution. At the beginning, the planning agent is placed in the given initial state. In each planning episode, MOCART-CGA looks for the best action applicable in the current state of the planning agent which is then executed (applied). After execution, the planning agent moves to a new state. At the new state, MOCART-CGA returns the best action which is executed and the agent moves to another state. This process continues until the planning agent reaches the goal state. A high level design of the planner is given in Figure 5.
The detailed description of the algorithm follows. At the beginning (Line 1) the planner initializes parameters (e.g. $limit, depth$) and puts the initial state to s. After that the

Procedure $Planner$
Read(s_o, g);
1. initialize parameters and $s := s_o$;
2. REPEAT
3. $a := MOCART-CGA(s, g)$;
4. $s' := Execute(a, s)$;
5. $n_e(s, a) = n_e(s, a)$+1;
6. $n_e(s, a, s') = n_e(s, a, s')$+1;
7. $P(s, a, s')$=$n_e(s, a, s')/n_e(s, a)$;
8. IF $n_e \geq a_{limit}$ THEN $Q(s, a) := 0$;
9. $s := s'$;
10. UNTIL $s.pos = g$;
End $Planner$

Fig. 5. A high level design of the MOCART-CGA planner

planner iteratively performs the following until the planning agent reaches the goal state g.

The planner calls MOCART-CGA (Line 3) for the state s to select an action $a \in App(s)$ to move the planning agent towards g. The action a is executed in s and the agent moves to a new state s' (Line 4). After that, we update the counters determining how many times a was executed in s and how many times it results in s' (Lines 5-6). The counters are used for computing the probability of reaching s' by executing a in s (Line 7). If the number of executions of a in s reaches a pre-defined limit a_{limit}, then $Q(s, a)$ is set to zero to prevent further execution of a in s (Line 8). The reason for such a limitation is to enable the agent to escape a cycle in which it might be trapped (i.e., if executing a sequence of actions in some state s results in reaching the same state s).

Empirical Analysis

MOCART-CGA is empirically evaluated using four benchmark maps from a commercial game called Dragon Age: Origins [1]. The details of these benchmarks are given in Table I. In the dynamic world of these benchmarks, dynamic obstacles appear on the empty spaces of the map randomly, simulating other moving agents, for example. The immediate visible range is restricted to ten cells. The immediate visibility represents the line of sight of a moving character. The closest rival techniques for comparison with MOCART-CGA are MCRT, LSS-LRTA and RTD, and we provide comparisons with these techniques. The maximum look-ahead depth is kept at 15 for all planners. We empirically optimize the parameters for each of the planners, ensuring that each planner runs in the best configuration. The dynamics of the environment used for benchmarking are specified as follows. Each time the agent executes an action and moves to some other state, 10% of the cells randomly change their status from blocked to empty or vice versa. The experiments are performed on a machine with Intel(R) Core (TM) 2 Quad processors each of speed 2.6 GHz and 8 GB RAM. The experiments are run in Windows 7 Professional edition.

Performance Metrics

The performance of a planner is measured using two parameters: sub-optimality of the solution and time per search. Time per search means the time taken by an algorithm to select one action in one planning episode. Time per search is represented by Ts in rest of the paper.

Sub-optimality is measured using a ratio of l_p to l_o (i.e. $\frac{l_p}{l_o}$) where l_p is the length of the solution by the planner and l_o is the length of the optimal path given in the benchmark. Sub-optimality is represented by Sub. Time per search is the key performance indicator as it is important for a real-time system to respond within a pre-defined time-limit. Smaller values of these parameters represent better performance of the planner.

The performance of MOCART-CGA is compared against MCRT, LSS-LRTA and RTD for these metrics.

Map	Size	No of Planning problems
Arena2	281 × 209	300
Orz103d	463 × 4456	300
Orz702d	718 × 939	450
Orz900d	1491 × 656	300

TABLE I
THE SELECTED BENCHMARK MAPS.

Dynamic Environments

The average sub-optimality of the planners is shown in Figure 6. Average sub-optimality is calculated using the mean of ten runs. The results show that MOCART-CGA performs significantly better than MCRT planner with respect to sub-optimality. MCRT has higher sub-optimality in Arena2 than in other benchmarks. The poor performance of MCRT planner in the dynamic benchmarks is mainly due to the unfocussed rollouts. MOCART-CGA (due to the focussed rollouts) produces solution quality comparable to the state-of-the-art real-time path planners i.e. LSS-LRTA and RTD.

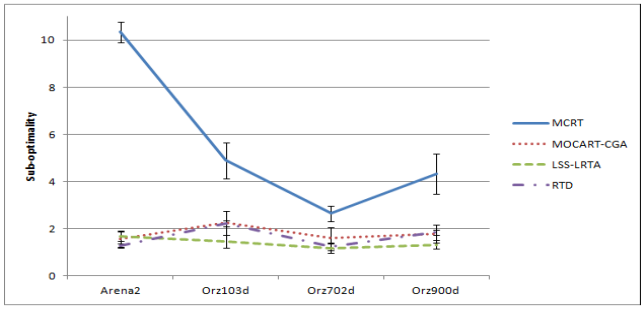


Fig. 6. Average sub-optimality (with Standard Error) of the planners on four benchmarks.

A comparison of the planners, with respect to time per search (Ts), is shown in Figure 7. The MOCART-CGA planner performs significantly better than its rivals with respect to time per search. RTD is the most expensive in terms of time per search which is due to the global backward search. RTD requires higher time per search than the other planners in all domains but produces solutions of the highest quality in the Arena2 benchmark instance (although not on other maps).

[1]http://movingai.com/benchmarks

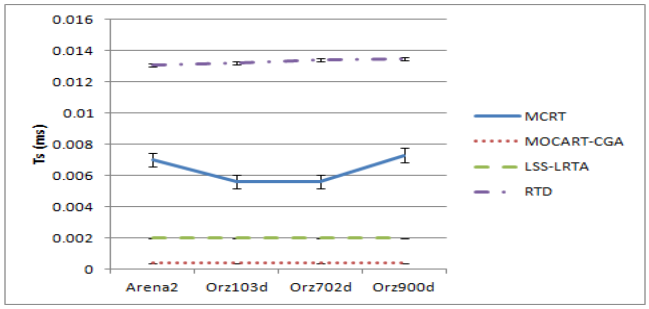


Fig. 7. Average Time per search (with standard error) of the planners on four benchmarks.

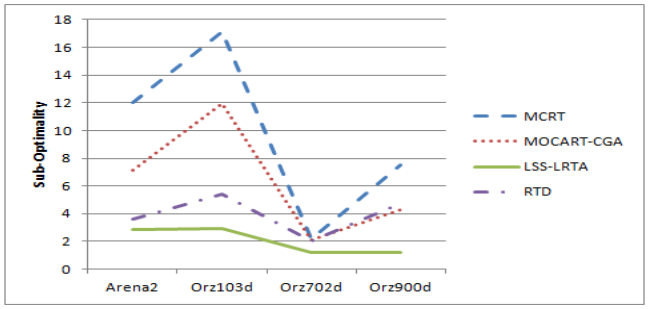


Fig. 8. Sub-optimality of the planners in the static and partially observable world of the benchmarks.

Sampling in the MCRT algorithm is more expensive than in MOCART-CGA. The difference highlights the importance of the use of corridors in action sampling to reduce the searching efforts in a planning episode in a Monte-Carlo planning technique. RTD is a promising approach that can solve path planning problems in a dynamic environment. This path planning algorithm can update (increase or decrease) the heuristic value of a cell if the grid cell is passable while LSS-LRTA cannot decrease the heuristic value of a cell if the cell is blocked due to a dynamic change. The Monte-Carlo planners decrease an action's value (e.g. MOCART-CGA does this in line 13 of Figure 3) if it leads to a blocked location and re-estimate the action value if that location is again passable due to a dynamic change.

Static Environments

We now compare performance on the same benchmark maps with no dynamic obstacles. Primarily, we include these results for completeness and to enable the reader to understand how MOCART-CGA performs in this domain. Our primary interest, as stated, is in the dynamic environments exemplified in RTS games.

The sub-optimality of the planners in the static environments of the benchmark maps is shown in Figure 8. Every planner solves all of the instances from the Orz702d map but for the other maps (i.e. Arena2, Orz103d and Orz900d) the planners could not solve all of the instances within the global time limit. For example, in the static world of Arena2, only RTD planner solves all 300 problems. And in Orz900d, not a single planner could solve all planning problems. For a fair comparison, we use the instances that all planners solve. In Arena2, there are 265 problems that are solved by all planners within the time limit. In the Orz103d and Orz900d maps, the number of problems commonly solved are 295 and 286 respectively.

The results show that LSS-LRTA performs better than all the other planners in the static benchmarks. MOCART-CGA remains better than MCRT. MOCART-CGA performs more poorly than RTD and LSS-LRTA in the Arena2 and Orz103d maps. In comparison to the dynamic benchmarks, the quality of the MOCART-CGA results degrades significantly.

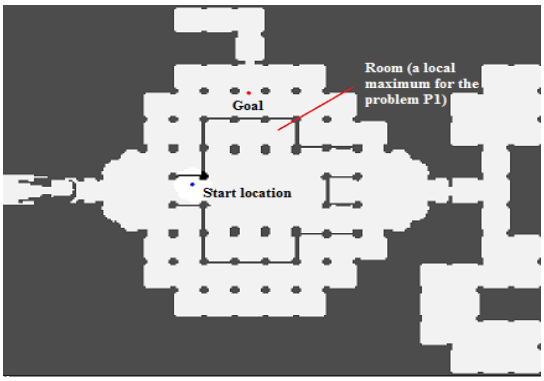


Fig. 9. A problem on the Arena2 map with a local maximum

Discussion of Results

MOCART-CGA outperforms its competitors in terms of time per search in all benchmark domains and in both dynamic and static partially observable environments. This is because MOCART-CGA has more focussed rollouts due to the use of corridors. Time per search is the key performance indicator as it is important for a real-time system such as an RTS game to select the next action within a very short time.

In terms of MOCART-CGA's average solution quality, it is much better than MCRT and comparable with LSS-LRTA and RTD. However, in some benchmarks, the quality of the plans produced by MOCART-CGA is lower than that of the other planners. The cause of this reduced quality is the slow recovery by MOCART-CGA from local maxima. Figure 9 shows a pathological problem where MOCART-CGA performs worse than RTD. The focussed rollouts of MOCART-CGA guide the planner towards the local maxima at the start of planning. To escape this local maxima, MOCART-CGA travels to every state within the local maxima at least once. Because MOCART-CGA planner imposes a bound on the number of times an action is selected to execute at a state, it does escape eventually. The searching effort required by MOCART-CGA to escape local maxima is proportional to the size of the local maxima. In the results for the static Orz103d map, the higher sub-optimality is evidence of this behaviour. The use of corridors does enable MOCART-CGA to learn more quickly than MCRT to escape from local maxima, but this is an area for future work.

In the static Orz702d map, all planners solve 450 problems and the sub-optimality of the planners is smaller as compared to their performance in other benchmark maps. In the static Orz900d benchmark instances, MOCART-CGA performs slightly better than RTD. In this map, most of the area between the start location and the goal location is occupied by static obstacles. Backward search in such cases could not guide RTD to escape the local maxima. Time per search in static environments is the same as it is in dynamic environments for all planners because they all impose a limit on the amount of planning per episode, thus MOCART-CGA has the fastest action selection mechanism in these environments also.

Future Work

The MCRT planner is a promising planner for path planning problems over multiple journeys and has been shown to perform better than LSS-LRTA in some previous studies [10]. Our results show that MOCART-CGA performs significantly better than the MCRT planner in the single journey path planning problems. These results also indicate that MOCART-CGA will improve in performance (compared to LSS-LRTA and MCRT) in path planning problems with multiple journeys.

In this work, it has been shown that a Monte Carlo approach is effective in solving path planning problems in dynamic environments. Currently our work is restricted to single-agent path planning. We believe that a similar approach will be effective in a multi-agent situation, especially given the fact that the time to search is very small for MOCART-CGA. All of the state-of-the-art approaches to path planning in partially observable domains suffer from becoming trapped in local maxima. We plan to experiment with different reward functions, and also with ways of dynamically updating the heuristic values of the states during search, in an effort to overcome these limitations.

CONCLUSION

In this paper we have presented the MOCART-CGA planner that performs focussed rollouts in a Monte-Carlo simulation for path-finding in a partially observable dynamic environment. The planner improves on previous Monte Carlo approaches by using corridors and a notion of convergence in order to effectively trade off exploitation of the best actions and exploration of new ones. The planner is evaluated using four benchmark maps and over 1000 instances. MOCART-CGA is compared against three state-of-the-art path planning algorithms: MCRT, LSS-LRTA and RTD.

Our results demonstrate that MOCART-CGA performs better in all metrics and on both static and dynamic domains than the previous best Monte Carlo based path planning algorithm, MCRT. We have demonstrated that it produces similar quality plans in dynamic domains to RTD and LSS-LRTA. MOCART-CGA uniformly has the lowest time to search of all the algorithms in static and dynamic environments, leading to a more responsive real-time algorithm.

REFERENCES

[1] D. P. Bertsekas, J. N. Tsitsiklis, and C. Wu, "Rollout algorithms for combinatorial optimization," *Journal of Heuristics*, vol. 3, pp. 245–262, 1997.

[2] G. Tesauro and G. R. Galperin, "On-line policy improvement using monte-carlo search." in *NIPS'96*, 1996, pp. 1068–1074.

[3] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Proceedings of the 17th European Conference on Machine Learning*, 2006, pp. 282–293.

[4] C. Lee, M. Wang, G. Chaslot, J. Hoock, A. Rimmel, O. Teytaud, S. Tsai, S. Hsu, and T. Hong, "The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments," *IEEE Transaction on Computational Intelligence and AI in Games*, vol. 1, pp. 73–89, 2009.

[5] R. Bjarnason, A. Fern, and P. Tadepalli, "Lower Bounding Klondike Solitaire with Monte-Carlo Planning," in *Proceedings of the 19th International Conference on Automated Planning & Scheduling*, 2009.

[6] V. Bulitko, N. Sturtevant, J. Lu, and T. Yau, "Graph abstraction in real-time heuristic search," *Artificial Intelligence*, vol. 30, pp. 51–100, 2007.

[7] N. J. Nilsson, *Principles of Artificial Intelligence*. Springer-Verlag, 1982.

[8] D. Hamm, "Navigational Mesh Generation: An empirical approach," in *AI Game Programming Wisdom 4*, S. Rabin, Ed. Charles River Media, 2008, pp. 113–114.

[9] S. Rabin, "A* speed optimizations," in *Game Programming Gems*, M. Deloura, Ed. Charles River Media, 2000.

[10] M. Naveed, A. Crampton, D. Kitchin, and T. McCluskey, "Real-Time Path Planning using a Simulation-Based Markovian Decision Process," in *AI-2011: 31st SGAI International Conference on Artificial Intelligence*, 2011.

[11] S. Koenig and X. Sun, "Comparing Real-Time and Incremental Heuristic Search for Real-Time Situated Agents," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 18, no. 3, pp. 313–341, 2009.

[12] D. Bond, N. Widger, W. Ruml, and X. Sun, "Real-Time Search in Dynamic Worlds," in *Proceedings of the Third Annual Symposium on Combinatorial Search* , 2010.

[13] M. Kearns, Y. Mansour, and A. Y. Ng, "A sparse sampling algorithm for near-optimal planning in large markov decision processes," in *Proceedings of the 16th International Joint Conference on Artificial intelligence - Volume 2*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 1324–1331.

[14] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, pp. 235–256, May 2002.

[15] R. Balla and A. Fern, "UCT for Tactical Assault Planning in Real-Time Strategy Games," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 2009, pp. 40–45.

[16] K. Laviers and G. Sukthankar, "A real-time opponent modeling system for rush football," in *Proceedings of International Joint Conference on Arificial Intelligence*, 2011, pp. 2476–2481.

[17] Y. Björnsson and H. Finnsson, "Cadiaplayer: A simulation-based general game player," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 1, no. 1, pp. 4–15, 2009.

[18] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[19] S. Koenig and M. Likhachev, "D* Lite," in *AAAI/IAAI*. AAAI Press, 2002.

[20] C. Hernandez and J. A. Baier, "Real-time heuristic search with depression avoidance," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, Barcelona, Spain, July 2011.

[21] N. Sturtevant and V. Bulitko, "Learning where you are going and from whence you came: h- and g- cost learning in real-timee heuristic search," in *Proceedings of the Twenty-second International Conference on Artificial Intelligence*, 2011, pp. 365–370.

[22] R. E. Korf, "Real-Time Heuristic Search," *Artificial Intelligence*, vol. 42, pp. 189–211, 1990.

[23] S. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[24] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, March 2012.