1-1-2012

Using Monte Carlo Tree Search for Replanning in a Multistage Simultaneous Game

Daniel Beard

Philip Hingston
Edith Cowan University

Martin Masek
Edith Cowan University

10.1109/CEC.2012.6256428

This is an Author's Accepted Manuscript of: Beard, D. R., Hingston, P. F., & Masek, M. (2012). Using Monte Carlo Tree Search for Replanning in a Multistage Simultaneous Game. Proceedings of 2012 IEEE Congress on Evolutionary Computation. (pp. 1-8). Brisbane, Australia. IEEE. Available here

Using Monte Carlo Tree Search for Replanning in a Multistage Simultaneous Game

Daniel Beard, Philip Hingston and Martin Masek
School of Computer and Security Science
Edith Cowan University
Perth, Western Australia, 6050
Email: d.beard@our.ecu.edu.au;p.hingston@ecu.edu.au;m.masek@ecu.edu.au

***Abstract*—In this study, we introduce MC-TSAR, a Monte Carlo Tree Search algorithm for strategy selection in simultaneous multistage games. We evaluate the algorithm using a battle planning scenario in which replanning is possible. We show that the algorithm can be used to select a strategy that approximates a Nash equilibrium strategy, taking into account the possibility of switching strategies part way through the execution of the scenario in the light of new information on the progress of the battle.**

I. Introduction

In recent years, Monte Carlo Tree Search (MCTS) has been very successfully applied to certain kinds of adversarial planning, notably for the game of Go [1] and for General Game Playing [2], but also for other games and in other domains. A recent survey paper provides an up to date overview of MCTS [3]. Most research to date has been focussed on move selection in sequential games.

In this paper, we introduce a new MCTS-like algorithm, MC-TSAR (Monte Carlo Tree Search for Adversarial Replanning). MC-TSAR is an algorithm for planning and replanning in an adversarial contest in which the adversaries *simultaneously* select high-level strategies, and, from time to time, have the opportunity to *replan*, in the sense of changing strategy in response to unfolding events.

This new algorithm is applicable to certain types of games, such as Real Time Strategy (RTS) games, in which moves are carried out in the context of an overall high-level strategy, where competing sides choose their strategies simultaneously rather than sequentially, and where a change in strategy may be required as the game progresses and new or updated information becomes available. See, for example, [4] and [5], which both argue that a competent player of an RTS game needs to plan at multiple, hierarchically nested levels. Many real-world adversarial planning domains have similar characteristics. An obvious example is military planning (obvious because of similarities with RTS games). Others include counterterrorism and security (e.g. [6]), and computer network security [7].

The rest of this paper is structured as follows: in the next section we review recent work that uses Monte Carlo methods in domains similar to those we are interested in. We then describe our new algorithm, MC-TSAR, and explain how it differs from earlier work. We then introduce a tactical battle planning scenario on which to test the algorithm. We are able to estimate analytically the performance of MC-TSAR against less capable planners applied to this scenario. We then present actual results from a series of simulations using the scenario, to verify that MC-TSAR performs as expected. We then introduce a more complex scenario that provides the opportunity for better planning, and present results showing that MC-TSAR is able to obtain better outcomes in this scenario. In the final section, we conclude with some discussion on how MC-TSAR could be further developed and combined with other methods, expanding its range of applicability both to more complex problems and to more detailed planning.

II. Related Work

There have been several previous studies investigating the use of Monte Carlo methods for planning in RTS games - which provide one example of the kind of adversarial planning problem that we are interested in.

Chung et al. [8] introduced MCPlan, a Monte Carlo planner for RTS games, and tested it in a Capture the Flag scenario simulated using ORTS [9]. MCPlan simply generates random plans and simulates each plan against randomly generated opponent plans for some number of steps, using a heuristic evaluation function to evaluate the final states. It then chooses the plan that has the statistically best results. Plans can be at whatever level of abstraction the implementor chooses. Compared to our proposed method, MCPlan does not explicitly account for replanning, and does not simulate to the end of the game, relying instead on domain knowledge in the form of an evaluation function. It also does not consider the opponent's reasoning processes.

In [10], this idea was extended to take into account the (simultaneous) choices of the opponent, proposing to use multiple simulations to evaluate each player's strategies against those of the opponent, and then choosing a Nash equilibrium strategy (which may be a mixed strategy) based on these evaluations, and using this chosen strategy until the next decision point. The whole procedure is repeated at each decision point. This is quite similar to our proposal except that, as in [8], replanning is only done in a reactive way, whereas we take the possibility of future replanning into account when planning. Also, their system is designed for constant updating of plans in real time, whereas we envisage planning and replanning only at a few key points.

In [11], the authors proposed and tested a Monte Carlo based planner that uses UCT (Upper Confidence Bound applied to Trees) [12], and assigns abstract actions to groups of agents in the context of tactical assault planning in an RTS game. They found that the tactical planner performed well in a variety of test scenarios, without making use of human domain knowledge. Their planner is designed to be used at specific decision points within the larger game. This work differs from ours in a number of respects: firstly, our approach is able to support and account for replanning; secondly, we aim to plan at strategic level or at least at a high level, rather than attempting to select specific actions; and lastly, their planner treats the game as a turn-based game and does not account for simultaneous choices by the two players. As they are when working with short times between decision points at the tactical level, this is a reasonable simplification.

A number of authors have investigated the use of UCT for simultaneous games, with Finnsson et al. [13] showing how this can be done. Sturtevant [14] showed that UCT converges to a mixed strategy equilibrium, but as shown in [15], not necessarily to a Nash equilibrium.

III. The Planning Algorithm - MC-TSAR

MC-TSAR (Monte Carlo Tree Search for Adversarial Replanning) is a new algorithm for planning and replanning in an adversarial contest in which the adversaries simultaneously select their strategies, and, from time to time, have the opportunity to change strategy in response to unfolding events.

We follow a common practice and view the contest as a game, in which the adversaries are players, and the problem is to select a player's actions so as to maximise his expected payoff at the end of the game. In board games like chess, checkers, or go, players take turns in choosing their actions (or moves), and choose a new move at each step in the game. In these kinds of games, especially those with moderate branching factors and strong evaluation functions, minimax game tree search and its many variants have been very popular and successful. A game tree is built starting from the current game state, and with a branch for each legal choice for the next move in that state. The tree is expanded to a certain depth, and the leaves are evaluated using a heuristic evaluation function. These values are then propagated up the tree to the root, using a minimax rule, and finally the player chooses a move that leads to a subtree with the most favourable evaluation for his side.

More recently, Monte Carlo-based tree search (MCTS) algorithms have been developed and applied to games with high branching factors, where there is no strong evaluation function. MCTS is similar to a minimax game tree search, except that paths in the game tree are continued to the end of the game, with statistical estimates of success replacing state evaluation. Efficient versions of MCTS, using UCT to determine which subtrees to explore most thoroughly, have been used with great success on a variety of games. The main steps in Monte Carlo Tree Search are shown in Figure 1.

A partial game tree is constucted, and is then expanded and refined as much as possible during the allowed planning time. As shown in Figure 1, in each expansion step, the tree is traversed until an unexpanded game state is reached (*selection*), a move is proposed and a new game state is added to the tree (*expansion*), *rollouts* or fast simulated games are played out starting from this state to estimate the value of this state (*simulation*), and this value is backpropagated up the tree to update the estimated values of other nodes in the tree, using minimax (*backpropagation*). These estimated state values are used to select the next move for the player.

MC-TSAR is similar to - it might be considered a variant of - Monte Carlo Tree Search (MCTS). Like MCTS, our algorithm is based on a game tree. Figure 2 shows the main structure of the part of the algorithm that expands the game tree. The key differences between our algorithm and MCTS are

1) At decision nodes, there are branches corresponding to each possible choice of a *pair* of strategies: one strategy for the player and one for the opponent;
2) At decision nodes, a *Nash equilibrium* is computed (rather than a minimax solution), and the resulting payoff values are propagated up the tree; and
3) When expanding a node of the tree, the chosen pair of strategies is used to select moves in the game for each side, until either the game ends, or the next decision point is reached. The intermediate states are not stored in the tree, just the state at the next decision point.

Thus MC-TSAR is aimed at choosing strategies, rather than moves, and the players are assumed to select Nash equilibrium strategies at each decision point. Note that this is a safe assumption for each player to make : if the opponent does not play a Nash equilibrium, he cannot get a better outcome. On the other hand, if a player has knowledge of his opponent's likely play (i.e. if he has a useful opponent model), then he may forego a better outcome by playing the Nash equilibrium, so MC-TSAR may not be appropriate. We describe the algorithm in pseudocode below. Algorithm 1 shows the overall idea - a game tree is built with the starting state at the root, and multiple subtrees for each subsequent decision point. The tree is expanded as much as allowed in the given planning time, by carrying out multiple "rollouts" of the game, and then expected payoffs are propagated backwards up the tree, recursively solving subgames at each decision point, until the game is solved at the root node, providing the optimal mixed strategy Nash equilibrium based on the results of the game rollouts.

Algorithms 2 through to 5 give more detail in pseudocode form. Note that there are several choices to make in Algorithm 3 - which pair of strategies to select for a rollout, and then whether to follow a previous rollout to the next decision point, or to start a new subtree for this rollout. Together, the rules for making these choices determine a *default policy*. In this initial work, we use random selection to choose a strategy pair, and a simple rule to decide whether to start a new subtree

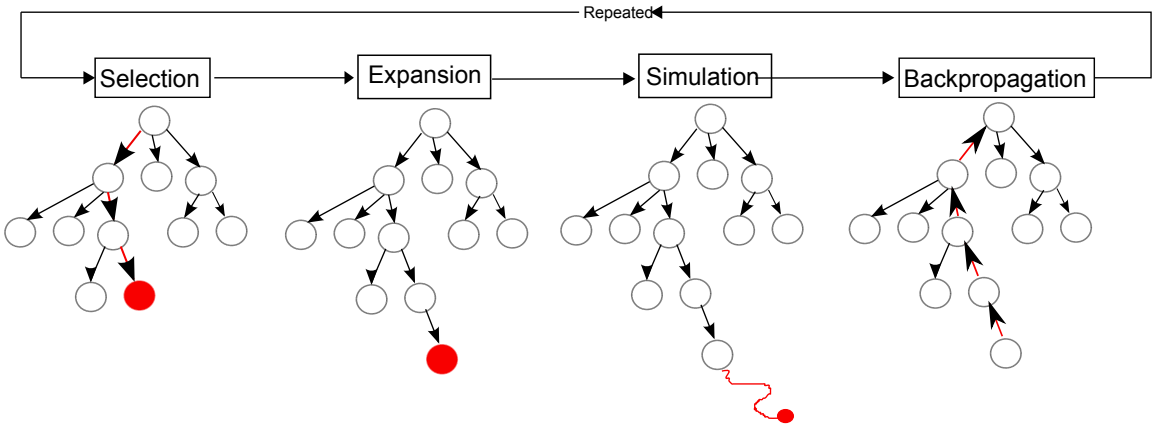

Fig. 1. Game tree expansion in MCTS. First a path from the root to a leaf is traversed. The selected leaf is then expanded by one move, adding another node to the tree. A rollout is executed starting at this node, and finally, the score at the end of the rollout is propagated back up the tree. (Based on Figure 1 of [16])

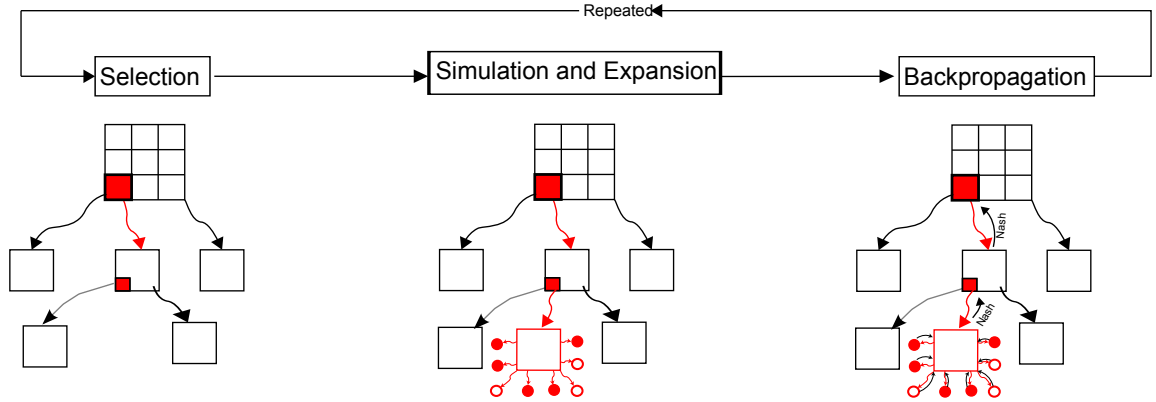

Fig. 2. Game tree expansion in MC-TSAR. Only some branches are shown. First, a path is traversed starting at the root. At each step, a pair of strategies is chosen, and a branch with that strategy pair is followed, until some pair is selected for expansion. In the figure, the path is shown in red, and the chosen pair is at depth 2. Next, the tree is expanded, adding a new subtree by executing a rollout using the selected strategies, until either the next replanning point is reached, or the simulation ends. If a replanning point was reached, a new subtree is built for each strategy pair, as in the figure. Finally, scores from final states are backpropagated up the tree, updating values by averaging scores for each strategy pair, and finding Nash equilibria at each decision point.

- we start a new one 10% of the time.

Input: state (the current game state)
Output: A mixed strategy to play

```
begin
    t ⇐ buildInitialGameTree(state);
    while planningTimeRemaining > 0 do
        growGameTree(t);
    end
    return solveGameTree(t);
end
```

Algorithm 1: select(state): select a mixed strategy to play from the current state until the next decision point

IV. The Test Scenario

To test MC-TSAR, we designed a simple battle scenario (see Fig 3). This scenario is simple enough that we can *approximate* it with an even simpler theoretical model, for which we can "solve" the strategy selection problem (see subsection IV-A). This allows us to check whether MS-TSAR is able to determine "correct" strategy choices. The solution methods for the simplified scenario mirror those of MC-TSAR, except that in MC-TSAR, Monte Carlo methods are used to estimate expected outcomes, whereas the simplified model can be solved analytically. We emphasise that, in general, any reasonably complex scenario cannot be solved using the analytical method.

In this battle scenario, there are two teams, which we call Red and Blue. The Blue team's goal is to defend three valuable sites, X, Y and Z. Red's goal is to capture one of these sites. The three sites are assigned relative values of 2:3:5, which might represent lesser or greater strategic value in a larger conflict, for example.

At the start of the scenario, each team has a squad of agents

Input: state (the current game state)
Output: A game tree

```
1  begin
2      t ⇐ a new game tree for this state with no branches;
3      if state is not final then
4          strategyPairs ⇐
           all pairs of candidate strategies;
5          foreach pair ∈ strategyPairs do
6              play the game starting at state, using the
               strategies in pair, until the next decision
               point is reached or the game ends;
7              s' ⇐ the new game state;
8              t' ⇐ buildGameTree(s');
9              add t' as a branch of t;
10         end
11     end
12     return t;
13 end
```

Algorithm 2: buildGameTree(state): build an initial game tree starting from the given state

Input: tree (a game tree)

```
1  begin
2      state ⇐ the state at the root of tree;
3      if state is not final then
4          pair ⇐ select a pair of candidate strategies;
5          if start a new subtree then
6              play the game starting at state, using the
               strategies in pair, until the next decision
               point is reached or the game ends;
7              s' ⇐ the new game state;
8              t' ⇐ buildGameTree(s');
9              add t' as a branch of tree;
10         end
11         else
12             t ⇐ select a subtree;
13             growGameTree(t);
14         end
15     end
16 end
```

Algorithm 3: growGameTree(tree): expand a game tree by executing another game rollout. There are several choices to make here - which strategies to select, and whether to start a new subtree.

Input: tree (a game tree)
Output: A pair of mixed strategies

```
1  begin
2      strategyPairs ⇐ all pairs of candidate strategies;
3      foreach pair ∈ strategyPairs do
4          calculate score(tree, pair);
5      end
6      construct a payoff matrix using these scores;
7      return a mixed strategy Nash equilibrium derived
       from this payoff matrix
8  end
```

Algorithm 4: solveGameTree(tree): find a pair of mixed strategies forming a Nash equilibrium for playing the game from this point

Input: tree (a game tree), pair (a pair of strategies)
Output: An expected score

```
1  begin
2      total ⇐ 0;
3      foreach branch b of this tree for this strategy pair do
4          state ⇐ the state at the root of b;
5          if state is final then
6              total ⇐ total + payoff;
7          end
8          else
9              pair' ⇐ solveGameTree(b);
10             total ⇐ total + score(b, pair');
11         end
12     end
13     return total / number of branches;
14 end
```

Algorithm 5: score(tree, pair): calculate an expected score when this pair of strategies is used to play the game from this point

(squads are evenly matched) located at its respective team base (R for Red or B for Blue). Each team can order its squad to march from its base to one of the target sites, X, Y or Z. For the Blue team, there are four possible routes from the Blue base, via either of two intermediate locations S or T, to one of the target sites: B-S-X, B-S-Y, B-T-Y, B-T-Z. Similarly, there are four possible routes from the Red base via either of two locations U or V: R-U-X, R-U-Y, R-V-Y or R-V-Z.

The routes are designed so that choosing the first part of a strategy (an intermediate location) restricts the choices available for the second part of the strategy (the final destination). Thus, each side gains valuable knowledge about the possible options for the enemy side during the execution of the scenario - knowledge that is not available before the action begins. Once this knowledge is available, a subsequent change of strategy could be advantageous.

When the scenario is executed, the two opposing squads follow their chosen routes to their chosen target sites. Each agent's movement is determined by its current position, the position of the next waypoint, and the locations of nearby teammates and enemies, using simple flocking rules. When an agent comes within close range of enemy agents, combat ensues and continues until either the agent is killed or all nearby enemies are killed. If the two squads of agents arrive at different destinations, Red is considered to have captured the site that the Red squad arrived at. If they arrive at the same destination, the two squads engage in battle until one squad is eliminated. If (any agent of) the Red squad survives, then

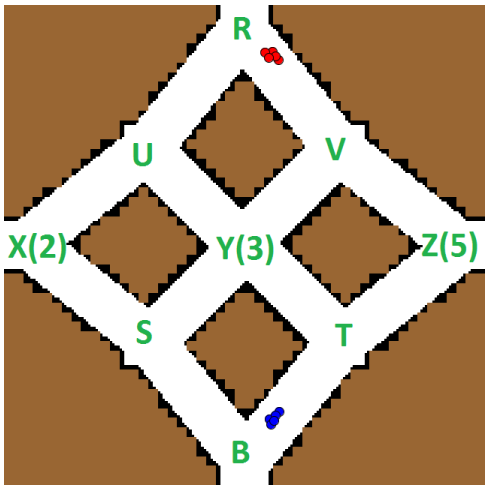

Fig. 3. Test Scenario - The white shape represents passable terrain: brown and black are impassable. At the bottom, near B, a Blue squad of 5 agents is on its way to T, and from there will be able to proceed to either Y or Z. Likewise, near the top, a Red squad is on its way to V.

Red is considered to have captured that site.

In the experiments to follow, we examine two variations on this scenario. In one variant, both teams must (simultaneously) select their strategy at the beginning of the scenario and then stick to it. In the other variation, the teams have an opportunity to change strategy when both squads have reached one of the intermediate locations. In other words, in the second variation, replanning is possible. This will allow us to investigate the ability of our algorithm to exploit the opportunity of replanning once some information on the enemy strategy is known. For example, if the Blue squad moves to intermediate location T, then Red knows that Blue can only defend either Y or Z: X cannot be defended. Red then has the option to change his intended target based on this new knowledge.

We also consider another set of variations in which the two squads can suffer some (stochastic) losses during the course of their travel. Specifically, with some probability, up to half the agents in the squad are lost to an IED (Improvised Explosive Device) when a squad arrives at one of the intermediate locations. These variations are included to allow us to evaluate how well our algorithm is able to replan in response to random events. For example, if one team suffers less losses than the other, then that team's chances of capturing a contested site are improved, changing the risk/reward equation.

A. Theoretical solutions

As mentioned above, the scenario, at least the version without IEDs, can be approximated by a simplified, more abstract one, which can be solved using game theoretic methods. We can then predict approximate expected outcomes for conflicts between teams that use various planning methods. Here we describe the approximation and tabulate theoretical solutions and expected outcomes. The calculation method mirrors the calculations used by MC-TSAR, except that in these theoretical calculations, exact payoffs are known, whereas MC-TSAR uses approximations derived from executing multiple rollouts.

In this simplified scenario, individual agents and their detailed movements are not modelled. Instead, each team chooses its route, and then both squads are moved in one step to their chosen intermediate waypoints. If replanning is allowed, then at this point, the teams simultaneouly choose the final destination for their squad – otherwise the final destination will be as chosen at the start. Both squads are then moved in one step to their final destinations. If the final destinations of the two squads are different, then Red is deemed to have captured its destination site, as in the full scenario. If the two squads have the same final destination, then Red is awarded half the value of that site, simulating equal chances of victory for each team.

1) Solving the game: Using these rules, we can derive the payoff matrix for the case when no replanning is allowed, shown in Table I. Nash equilibrium strategies for Red and Blue can then be found by deriving a linear programming problem from this payoff matrix. The solution is for Red to choose R-U-Y with probability $5/8$ and R-V-Z with probability $3/8$, and Blue to choose B-S-Y with probability $1/8$ and B-T-Z with probability $7/8$, for an expected payoff of 2.8125.

TABLE I
PAYOFFS FOR THE SIMPLIFIED SCENARIO WITHOUT REPLANNING

		Blue			
		B-S-X	B-S-Y	B-T-Y	B-T-Z
Red	R-U-X	1	2	2	2
	R-U-Y	3	1.5	1.5	3
	R-V-Y	3	1.5	1.5	3
	R-V-Z	5	5	5	2.5

We can also derive solutions for the case where replanning is allowed. First, we consider the subproblem of choosing a new strategy after both squads are at their intermediate locations. For example, suppose that Blue initially chose B-S-X or B-S-Y i.e. one of the routes with S as the intermediate location, and Red chose an intermediate location of U. We can then derive a payoff matrix for the possible final destinations of the two players as in Table II (a). The solution for this subproblem is for Red to choose X with probability $3/5$ and Y with probability $2/5$, and for Blue to choose X with probability $1/5$ and Y with probability $4/5$. The expected payoff is 1.8.

By solving each subproblem in a similar manner, we can derive a payoff matrix for each possible choice of intermediate location for the two teams, as in Table II, and use the payoffs to derive an overall payoff matrix as in Table III. The solution for the whole scenario including replanning, as derived from this payoff matrix, is for Red to always move first to V, and Blue to always move first to T. After the two teams have chosen and moved to V and T respectively, Red next goes to Y with probability $5/8$ and Z with probability $3/8$, while Blue moves to Y with probability $1/8$ and Z with probability $7/8$. The expected overall payoff is 2.8125, the same as for the no-replanning case, with the same distribution of final destinations, but via different routes.

		Blue	
		S-X	S-Y
Red	U-X	1	2
	U-Y	3	1.5

(a) Red has moved to U and Blue has moved to S

		Blue	
		T-Y	T-Z
Red	U-X	2	2
	U-Y	1.5	3

(b) Red has moved to U and Blue has moved to T

		Blue	
		S-X	S-Y
Red	V-Y	3	1.5
	V-Z	5	5

(c) Red has moved to V and Blue has moved to S

		Blue	
		T-Y	T-Z
Red	V-Y	1.5	3
	V-Z	5	2.5

(d) Red has moved to V and Blue has moved to T

TABLE II
PAYOFF MATRICES (SIMPLIFIED SCENARIO) FOR FINAL MOVES, AFTER RED AND BLUE HAVE MOVED TO THEIR INTERMEDIATE LOCATIONS.

TABLE III
PAYOFFS (SIMPLIFIED SCENARIO) FOR INITIAL MOVES, ASSUMING BOTH PLAYERS MAKE THEIR FINAL MOVES USING THE NASH SOLUTIONS FOR THE RESULTING SUBPROBLEMS

		Blue	
		B-S	B-T
Red	R-U	1.8	2
	R-V	5	2.8125

2) Predicting performance: We can extend this analysis to calculate expected outcomes of contests between various players that use different planning methods. First, we consider the case in which the scenario is executed *without replanning* - the two players must select their respective strategies at the start and then stick with their choices throughout execution of the scenario. The calculated outcomes are shown in Table IV.

TABLE IV
CALCULATED EXPECTED PAYOFFS FOR RED, WHEN NO REPLANNING IS ALLOWED (SIMPLIFIED SCENARIO)

		Blue	
		random	planner
Red	random	2.65625	2.609375
	planner	3.046875	2.8125

The *random* player simply randomly chooses any strategy with equal probability, while the *planner* players choose a strategy based on the Nash solution derived from Table I. It can be seen that in all cases, planning is beneficial (the Red player gets a higher payoff when planning than when playing randomly, and Blue is able to force a lower payoff for Red by planning).

Second, we consider the case where replanning is allowed – that is, the players are permitted to change their plan part way through the scenario. A new kind of player, the *replanner* actually takes into account that replanning will be allowed, when choosing their initial strategies (using the values in Table III). The calculated outcomes are shown in Table V. Once again, more planning is seen to be beneficial in nearly all cases.

V. EXPERIMENT 1 - ADJUSTING TO ENEMY CHOICES

In this first experiment, we tested the performance of the three players (random, planner and replanner) on the test

TABLE V
CALCULATED EXPECTED PAYOFFS FOR RED, WHEN REPLANNING IS ALLOWED (SIMPLIFIED SCENARIO)

		Blue		
		random	planner	replanner
Red	random	2.65625	2.311719	2.28125
	planner	2.65234	2.391602	2.30469
	replanner	3.90625	3.085938	2.81250

scenario, as implemented in the multi-agent simulation toolkit, MASON [17], without the complication of IEDs. The planner and replanner players do not have exact information on payoffs available. Instead, at each decision point, they use MC-TSAR to build and grow a simulation tree, and to select a strategy. In these experiments, each squad has five agents, and 100 rollouts were used for each decision. For each pair of players, 100 games were played and the mean red payoffs were calculated, along with the standard errors of the means. Scenarios with and without replanning were run.

We expected the results to be similar to those in Table IV and Table V. However, because the planners are using payoffs estimated using a limited number of rollouts, their calculated Nash equilibrium probabilities will not be exactly the same as the theoretical ones used in the calculations above. This is explained further below. The corresponding experimental results are given in Tables VI and VII.

TABLE VI
EXPERIMENTALLY OBTAINED PAYOFFS FOR RED, WHEN NO REPLANNING IS ALLOWED (NO IEDS)

		Blue	
		random	planner
Red	random	2.620 ± 0.170	2.680 ± 0.125
	planner	2.960 ± 0.198	2.790 ± 0.182

TABLE VII
EXPERIMENTALLY OBTAINED PAYOFFS FOR RED, WHEN REPLANNING IS ALLOWED (NO IEDS)

		Blue		
		random	planner	replanner
R	random	2.620 ± 0.170	2.300 ± 0.164	2.130 ± 0.159
	planner	3.460 ± 0.175	2.770 ± 0.165	2.530 ± 0.142
	replanner	3.900 ± 0.162	2.990 ± 0.171	2.790 ± 0.185

Note that most of the results match the theoretical ones within standard error, but red payoffs in the middle row in the scenario with replanning are higher than expected. However, the trends are as expected - in nearly every case, deeper planning is beneficial. The reason that the middle row has higher than expected payoffs is that the theoretical payoff matrix in Table I has some symmetries - for example, the payoffs for the Red strategies R-U-Y and R-V-Y are the same. Therefore there are many equivalent Nash equilibria with different probabilities for these two strategies. The solver we used happened to choose one in which R-V-Y has zero probability. With noisy estimates of the payoff, however, this symmetry is broken, and R-U-Y and R-V-Y are equally likely

to be chosen. This leads to higher payoffs for Red, because at the replanning point, he can switch from a final destination of Y to a final destination of Z. Note that the planner does not anticipate this when making his initial plan because he does not take replanning into account - he just caught a lucky break.

VI. Experiment 2- adjusting to random events

In this second experiment, we introduce the complication of IEDs into the MASON simulation, as described in Section IV. This is much more complex to analyse theoretically, and we haven't attempted to do so. A "real-world" scenario would likely be even more complex and much too complicated for a theoretical analysis. However, a lot of complexity can easily be included in a simulation model, and we expect that MC-TSAR will make strong strategy choices so long as the model reflects the important features of the scenario. Experimental results using the MASON simulation including IEDs are given in Tables VIII and IX.

TABLE VIII
Experimentally obtained payoffs for Red, when no replanning is allowed (with IEDs)

		Blue	
		random	planner
Red	random	2.580 ± 0.160	2.630 ± 0.159
	planner	3.370 ± 0.187	2.880 ± 0.184

TABLE IX
Experimentally obtained payoffs for Red, when replanning is allowed (with IEDs)

		Blue		
		random	planner	replanner
R	random	2.580 ± 0.160	2.490 ± 0.150	2.130 ± 0.159
	planner	3.470 ± 0.166	2.730 ± 0.176	2.590 ± 0.163
	replanner	3.240 ± 0.203	3.040 ± 0.160	2.770 ± 0.165

These results show the expected pattern, with more planning giving better outcomes. MC-TSAR is able to competently handle uncertainty due to random events as well as that due to lack of prior knowledge of the opponent's strategy.

VII. Conclusion and Future Work

In this paper, we have introduced a new Monte Carlo Tree Search algorithm for multistage simultaneous games. The algorithm uses Monte Carlo simulation to build a game tree, which can be solved recursively to select a strategy that approximates a Nash equilibrium strategy. This strategy takes into account replanning by both sides. We have tested the new algorithm using an agent-based simulation of a battle planning scenario.

In the future, we plan to further develop MC-TSAR and apply it to more complex and realistic problems. While we believe the approach is very promising, there remain many challenges and opportunities in terms of scaling and efficiency.

During the growing phase of the algorithm, several choices have to be made to decide which subtrees to explore and expand. UCT has been used very successfully for this purpose in MCTS, but it is not clear whether or how a similar method could be applied for MC-TSAR, as it is not clear how to best to ensure sufficiently accurate estimates of Nash equilibrium solutions for each subgame.

Intuitively, more rollouts should lead to more accurate estimates of payoffs, but there are at least two unknowns in this respect: how does the strength of the decision making scale with the number of rollouts performed? and how can the game tree be kept to a feasible size as the number of rollouts is increased? We intend to investigate the idea of clustering of game states, in order to combine and collapse subtrees for similar states. We hope that this will limit the size of the game tree without greatly affecting accuracy. It will also have the advantage that, as the game progresses, a new game tree can be built using an existing subtree as a starting point (as is possible with MCTS), rather than building a new game tree from scratch at each decision point.

Another unanswered question for future research: the algorithm allows a player to select from a small set of possible strategies, but how can we determine a suitable set of strategies to select from for a complex scenario? In our test scenario, the possible actions for each side can conveniently be described at a high level in terms of the routes taken by their agents, but in a real scenario things may not be so simple, and the strategy search space may be large. One possible approach that we intend to test is to use a coevolutionary algorithm to find a small set of strong candidate strategies for each side, and then to apply MC-TSAR to select from among those strategies.

References

[1] T. Cazenave and B. Helmstetter, "Combining tactical search and Monte-Carlo in the game of Go," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2005, pp. 171–175.

[2] Y.Björnsson and H. Filmar, "Cadiaplayer: A simulation-based general game player," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 1, pp. 4–15, March 2009.

[3] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2012.

[4] J. McCoy and M. Mateas, "An integrated agent for playing real-time strategy games," in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, 2008, pp. 1313–1318.

[5] B. Weber and M. Mateas, "Building human-level AI for real-time strategy games," in *AAAI Fall Symposium Series, Advances in Cognitive Systems*, 2011.

[6] D. Korzhyk, Z. Yin, C. Kiekintveld, V. Conitzer, and M. Tambe, "Stackelberg vs. Nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness," *Journal of Artificial Intelligence Research*, vol. 41, pp. 5297–327, 2011.

[7] S. Roy, C. Ellis, S. Shiva, D. Dasgupta, V. Shandilya, and Q. Wu, "A survey of game theory as applied to network security," in *Hawaii International Conference on System Sciences*, 2010.

[8] M. B. M. Chung and J. Schaeffer, "Monte Carlo planning in RTS games," in *Proc. IEEE Symposium on Computational Intelligence and Games*, 2005, pp. 117–124.

[9] ORTS - Open Real-Time Strategy. [Online]. Available: http://www.cs.ualberta.ca/mburo/orts

[10] F. Sailer, M. Buro, and M. Lanctot, "Adversarial planning through strategy simulation," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, April 2007, pp. 80–87.

[11] R. K. Balla and A. Fern, "UCT for tactical assault planning in real-time strategy games," in *Proc. Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 40–45.

[12] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Machine Learning: ECML 2006*, ser. Lecture Notes in Computer Science, J. Frnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Springer Berlin / Heidelberg, 2006, vol. 4212, pp. 282–293. [Online]. Available: http://dx.doi.org/10.1007/11871842_29

[13] H. Finnsson and Y. Björnsson, "Simulation-based approach to general game playing," in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, 2008, pp. 259–264.

[14] N. Sturtevant, "An analysis of UCT in multi-player games," in *In Computers and Games*, 2008.

[15] M. Shafiei, N. Sturtevant, and J. Schaeffer, "Comparing UCT versus CFR in simultaneous games," in *Proceedings of the IJCAI-09 Workshop on General Game Playing (GIGA'09)*, 2009.

[16] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-Carlo Tree Search: A new framework for game AI," in *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, M. Mateas and C. Darken, Eds. AAAI Press, Menlo Park, CA, USA, 2008, pp. 216–217.

[17] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "MASON: A Multi-Agent Simulation environment," *Simulation: Transactions of the Society for Modeling and Simulation International*, vol. 82, no. 7, pp. 517–527, 2005.