Actor-Critic Policy Learning in Cooperative Planning

Josh Redding, Alborz Geramifard and Jonathan How
{jredding,agf,jhow}@mit.edu
Department of Aeronautics & Astronautics
Massachusetts Institute of Technology
77 Massachusetts Avenue, Cambridge, MA 02139

Abstract

In this paper, we introduce a method for learning and adapting cooperative control strategies in real-time stochastic domains. Our framework is an instance of the intelligent cooperative control architecture (iCCA). The agent starts by following the "safe" plan calculated by the planning module and incrementally adapting the policy to maximize rewards. Actor-critic and consensus-based bundle algorithm (CBBA) were employed as the building blocks of the iCCA framework. We demonstrate the performance of our approach by simulating limited fuel unmanned aerial vehicles aiming for stochastic targets. The integrated framework boosted the optimality of the solution by %10 compared to running each of the modules individually.

Introduction

Planning for heterogeneous teams of mobile, autonomous, health-aware agents in uncertain and dynamic environments is a challenging problem. In such a setting, the agents are simultaneously engaged and continuously interact with each other, their surroundings and with potential threats. They may encounter evasive targets and need to reason through adversarial actions with insufficient data. Or, agents may receive delayed, lossy and contaminated communications or experience sensor and actuator failures. In addition, autonomous agents must be robust to unmodeled dynamics and parametric uncertainties while remaining capable of performing their advertised range of tasks.

Add to these challenges the requirement that all sensing, computation and decision-making be done onboard the agent and the problem becomes simultaneously more realistic and difficult. The goal of embedded reasoning is to advance onboard system capabilities in solving complex tasks, such as adapting teams of autonomous agents to dynamic and uncertain environments. Embedded reasoning seeks to enable "learning from experience" for these agents and thereby improve their ability to make and keep plans. While this research does not present results obtained from truly embedded algorithms, we are, however, currently in the active process of developing many of the elements presented here at the microprocessor level.

Although much work has been done in the area of multi-agent planning in uncertain environments (Kaelbling, Littman, and Cassandra 1998; Russell and Norvig 2003; H. Brendan McMahan 2003; Murphey and Pardalos 2002), there yet remains many open questions, such as:

- *How to improve planner performance over time in the face of uncertainty and a dynamic world?*
- *How to use current knowledge and past observations to become both robust to likely failures and intelligent with respect to unforeseen future events?*

In this research, we are interested specifically in improving the performance of the system over time in an uncertain world. To study this problem, we use the intelligent cooperative control architecture (iCCA)(Redding et al. 2010) which is shown in Figure 1.

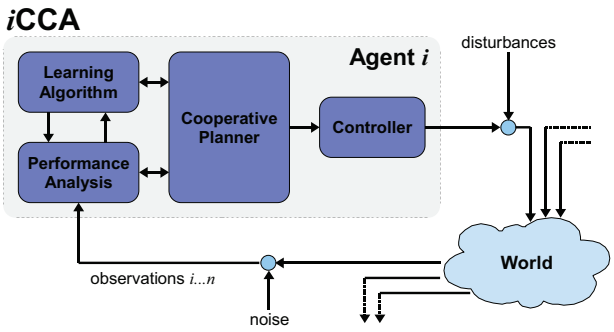

Figure 1: An **i**ntelligent **C**ooperative **C**ontrol **A**rchitecture, a framework for the integration of cooperative control algorithms and machine learning techniques.

iCCA is comprised of a cooperative planner, a learner, a metric for performance-to-date. For this research, we use the consensus-based bundle algorithm (CBBA) (Choi, Brunet, and How 2009) as the cooperative planner to solve the multi-agent task allocation problem. For the learning algorithm, we implemented an actor-critic reinforcement learner which uses information regarding performance to explore and suggest new behaviors that would likely lead to more favorable outcomes than the current behavior would produce. The performance analysis block is implemented as a "risk" analysis tool where actions suggested by the learner can be

overridden by the baseline cooperative planner if they are deemed too risky. This synergistic planner-learner relationship yields a "safe" policy in the eyes of the planner, upon which the learner can only improve. Ultimately, this relationship will help to bridge the gap to successful and intelligent execution in real-world missions.

The remainder of this paper details the integration of learning with cooperative control and shows how the marriage of these two fields results in an intelligent, adaptable planning scheme in the context of teams of autonomous agents in uncertain environments. In the following section, we motivate and formally states the multi-agent planning problem. The specifics of the proposed architecture are then detailed and followed by a discussion of simulation results and conclusions.

Problem Statement

Having introduced the general problem, a few key research gaps and outlined the proposed solution, we now prepare to delve deeper into our approach by formally presenting the problem we aim to solve and by giving background information. We first outline a small yet difficult multi-agent scenario where traditional cooperative control tends to struggle. We then formulate the planning problem associated with this complex scenario and further discuss existing solutions. The following section then discusses the technical details of the proposed solution approach and explains how it fills the research gap of interest.

Problem Scenario

Here we outline the scenario in which we developed and tested each of the modules in the iCCA framework. Referring to Figure 2, we see a depiction of the mission scenario of interest where a team of two fuel-limited UAVs cooperate to maximize their total reward by visiting valuable target nodes in the network. The base is highlighted as node 1 (green circle), targets are shown as blue circles and agents as triangles. The total amount of fuel for each agent is highlighted by the number inside each triangle. For those targets with an associated reward it is given a positive number nearby. The constraints on when the target can be visited are given in square brackets and the probability of receiving the known reward when the target is visited is given in the white cloud nearest the node.[1] Each reward can be obtained only once and all edges take one fuel cell and one time step. We also allow UAVs to loiter on any nodes for the next time step. The fuel burn for loitering action is also one except for the UAVs sStaying in the base, where they assumed to be stationary and sustain no fuel cost. The mission horizon was set to 8 time steps.

Markov Decision Processes

As the scenario above is modeled as a multi-agent Markov Decision Process (MDP) (Howard 1960; Puterman 1994; Littman, Dean, and Kaelbling 1995), we first provide some

[1]If two agents visit a node at the same time, the probability of visiting the node would increase accordingly.

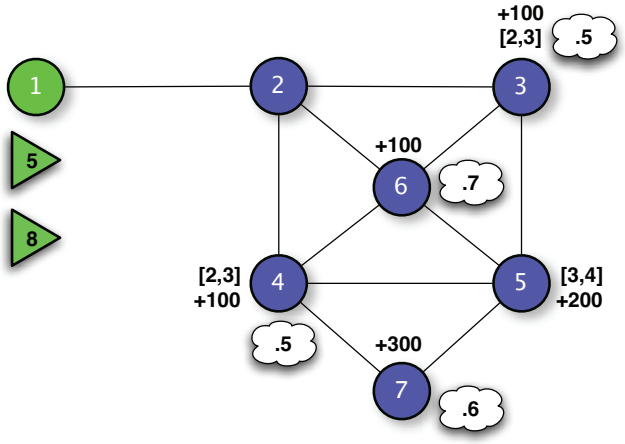

Figure 2: The mission scenario of interest: A team of two UAVs plan to maximize their total reward by cooperating to visit targets. Target nodes are shown as circles with rewards noted as positive values and the probability of receiving the reward shown in the associated cloud. Note that some target nodes have no value. Constraints on the allowable visit time of a target are shown in brackets.

relevant background. The MDP framework provides a general formulation for sequential planning under uncertainty. An MDP is defined by tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of possible actions. Taking action a from state s has $\mathcal{P}^a_{ss'}$ probability of ending up in state s' and receiving reward $\mathcal{R}^a_{ss'}$. Finally $\gamma \in [0, 1]$ is the discount factor used to prioritize early rewards against future rewards.[2] A trajectory of experience is defined by sequence $s_0, a_0, r_0, s_1, a_1, r_1, \cdots$, where the agent starts at state s_0, takes action a_0, receives reward r_0, transit to state s_1, and so on. A policy π is defined as a function from $\mathcal{S} \times \mathcal{A}$ to the probability space $[0, 1]$, where $\pi(s, a)$ corresponds to the probability of taking action a from state s. The value of each state-action pair under policy π, $Q^\pi(s, a)$, is defined as the expected sum of discounted rewards when the agent takes action a from state s and follow policy π thereafter:

$$Q^\pi(s, a) = E_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_0 = s, a_0 = a, \right].$$

The optimal policy π^* maximizes the above expectation for all state-action pairs:

$$\pi^* = argmax_a Q^{\pi^*}(s, a)$$

MDP Formulation

Here, we formulate the scenario of interest into an MDP, as described above.

State Space $\mathcal{S}$: We formulated the state space as $[N_1, F_1, \ldots, N_n, F_n, V_1, \ldots, V_m, t]^T$, where N_i and F_i are

[2]γ can be set to 1 only for episodic tasks, where the length of trajectories are fixed.

integer values highlighting the location and the remaining fuel for UAV i, V_j is a single bit signaling if node j has been ever visited before, and t is the current time step. There are n UAVs and m nodes participating in the scenario.

Action Space $\mathcal{A}$: Action space is $[N_1^+, \ldots, N_n^+]$ where N_i^+ is the node to which UAV i is traveling or, where it will be at the next time interval.

Transition Function $\mathcal{P}_{ss'}^a$: Transition function is deterministic for the UAV position, fuel consumption, and time variables of the state space, while it is stochastic for the visited list of targets. The detailed derivation of the complete transition function should be trivial following Figure 2.

Reward Function $\mathcal{R}_{ss'}^a$: The reward on each time step is stochastic and calculated as the sum of rewards from visiting new desired targets minus the total burnt fuel cells on the last move (Figure 2). Notice that a UAV receives the target reward only if it lands on an unvisited rewarding node and lucky enough to obtain the reward. In that case, the corresponding visibility bit will turn on, and the agent receive the reward. We set the crash penalty to -800, which occurs if any UAVs runs out of fuel or is not at the base by the end of the mission horizon.

iCCA

In this section, we detail our instance of the intelligent cooperative control architecture (iCCA), describing the purpose and function of each element and how the framework as a whole fits together with the MDP formulated in the previous section.

iCCA Elements

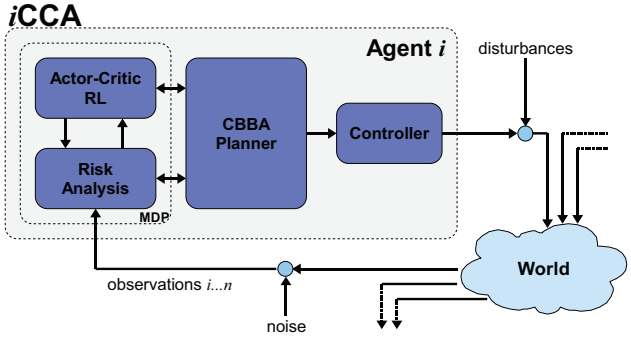

Figure 3: iCCA framework as implemented. CBBA planner and risk analysis and the actor-critic learner formulated within an MDP.

Cooperative Planner

At its fundamental level, the cooperative planner yields a solution to the multi-agent path planning, task assignment or resource allocation problem, depending on the domain. This means it seeks to fulfill the specific goals of the application in a manner that optimizes an underlying, user-defined *objective function*. Many existing cooperative control algorithms use observed performance to calculate *temporal-difference errors* which drive the objective function in the desired direction (Murphey and Pardalos 2002; Bertsekas and Tsitsiklis 1996). Regardless of how it is formulated (e.g. MILP, MDP, CBBA, etc...), the cooperative planner, or cooperative control algorithm, is the source for baseline plan generation within *iCCA*.

In this research, we implemented a decentralized auction protocol called the consensus-based bundle algorithm (CBBA) as the cooperative planner. The following section details this approach.

Consensus-Based Bundle Algorithm CBBA is a decentralized auction protocol that produces conflict-free assignments that are relatively robust to disparate situational awareness over the network.

CBBA consists of iterations between two phases: In the first phase, each vehicle generates a single ordered bundle of tasks by sequentially selecting the task giving the largest marginal score. The second phase resolves inconsistent or conflicting assignments through local communication between neighboring agents. In the local communication round, some agent i sends out to its neighboring agents two vectors of length N_t: the winning agents vector $\mathbf{z}_i \in \mathcal{I}_t^N$ and the winning bids vector $\mathbf{y}_i \in \mathbb{R}_+^{N_t}$. The j-th entries of the $\mathbf{z}_i$ and $\mathbf{y}_i$ indicate who agent i thinks is the best agent to take task j, and what is the score that agent gets from task j, respectively. The essence of CBBA is to enforce every agent to agree upon these two vectors, leading to agreement on some conflict-free assignment regardless of inconsistencies in situational awareness over the team.

There are several core features of CBBA identified in (Choi, Brunet, and How 2009). First, CBBA is a decentralized decision architecture. For a large team of autonomous agents, it would be too restrictive to assume the presence of a central planner (or server) with which every agent communicates. Instead, it is more natural for each agent to share information via local communication with its neighbors. Second, CBBA is a polynomial-time algorithm. The worst-case complexity of the bundle construction is $\mathcal{O}(N_t L_t)$ and CBBA converges within $\max\{N_t, L_t N_a\} D$ iterations, where N_t denotes the number of tasks, L_t the maximum number of tasks an agent can win, N_a the number of agents and D is the network diameter, which is always less than N_a. Thus, the CBBA methodology scales well with the size of the network and/or the number of tasks (or equivalently, the length of the planning horizon). Third, various design objectives, agent models, and constraints can be incorporated by defining appropriate scoring functions. If the resulting scoring scheme satisfies a certain property called *diminishing marginal gain* (DMG), a provably good feasible solution is guaranteed.

While the scoring function primarily used in (Choi, Brunet, and How 2009) was a time-discounted reward, a more recent version of the algorithm is due to Ponda (Ponda et al. 2010) and handles the following extensions while pre-

serving convergence properties:

- Tasks that have finite time windows of validity
- Heterogeneity in the agent capabilities
- Vehicle fuel cost

This research uses this extended version of CBBA as the cooperative planner.

Risk/Performance Analysis

One of the main reasons for cooperation in a cooperative control mission is to minimize some cost, or objective function. Very often this objective involves time, risk, fuel, or other similar physically-meaningful quantities. The purpose of the performance analysis module is to accumulate observations, glean useful information buried in the noise, categorize it and use it to improve subsequent plans. In other words, the performance analysis element of iCCA attempts to improve agent behavior by diligently studying its own experiences (Russell and Norvig 2003) and compiling relevant signals to drive the learner and/or the planner.

The use of such feedback within a planner is of course not new. In fact, there are very few cooperative planners which do not employ some form of measured feedback. In this research, we implemented this module as a risk analysis element where candidate actions are evaluated for risk level. Actions deemed too risky are replaced with another of lower risk. The details of this feature are given later on.

Learning Algorithm

Although learning has many forms, iCCA provides a minimally restrictive framework where the contributions of the learner to fall into either of two categories:

1. Assist the cooperative planner by adapting to parametric uncertainty of internal models
2. Suggesting candidate actions to the cooperative planner that the learner sees as beneficial

As our focus for this research is on the latter category, we chose to formulate the learner as an actor-critic type and set it under a Markov Decision Process framework, as described previously. In actor-critic learning, the actor handles the policy, where in our experiments actions are selected based on Gibbs softmax method:

$$\pi(s,a) = \frac{e^{P(s,a)/\tau}}{\sum_b e^{P(s,b)/\tau}},$$

in which $P(s,a)$ is the preference of taking action a in state s, and $\tau \in (0,\infty]$ is the temperature parameter acting as nob shifting from greedy towards random action selection. Since we use a tabular representation the actor update amounts to:

$$P(s,a) \leftarrow P(s,a) + \alpha Q(s,a)$$

following the incremental natural actor-critic framework (Bhatnagar et al. 2007). As for the critic, we employed the Sarsa algorithm [*See* (Sutton and Barto 1998)] to update the associated value function estimate. We initialized the actor's policy by boosting the initial preference of the actions generated by CBBA. As actions are pulled from the policy for implementation, they are evaluated for risk level and can be overridden by the baseline CBBA if the action leads UAVs into a configuration where crashing scenario is inevitable. As the agents implement the policy, the critic receives rewards for the actor's actions.

As a reinforcement learning algorithm, iCCA introduces the key concept of *bounded exploration* such that the learner can explore the parts of the world that may lead to better system performance while ensuring that the agent remain safely within its operational envelope and away from states that are known to be undesirable, such as running out of fuel. In order to facilitate this bound, the risk analysis module inspects all suggested actions of the actor, and replaces the risky ones with the baseline CBBA policy. This process guides the learning away from catastrophic errors. In essence, the baseline cooperative control solution provides a form of "prior" over the learner's policy space while acting as a backup policy in the case of an emergency.

However, a canonical failure of learning algorithms in general, is that negative information is extremely useful in terms of the value of information it provides. We therefore introduce the notion of a "virtual reward". In this research, the virtual reward is a large negative value delivered by the risk analysis module to the learner for risky actions suggested by the latter and pruned by the former. When this virtual reward is delivered, the learner associates it with the previously suggested action, therefore dissuading the learner from suggesting that action again, reducing the number of "emergency calls" in the future.

Flight Results

In this section, we present and discuss simulation results for the given scenario and problem formulation. First we solved the problem using backward dynamic programing in order to use this solution as the baseline for the optimality.[3] We ran the CBBA on the converted stochastic problem into the expected deterministic problem, and ran it for 10,000 episodes. Finally we empirically searched for the best learning rates for the Actor-Critic and iCCA where the earning rate calculated by:

$$\alpha_t = \alpha_0 \frac{N_0 + 1}{N_0 + \text{Episode\#}^{1.1}}.$$

The best α_0 and N_0 have been selected through experimental search of the sets of $\alpha_0 \in \{0.01, 0.1, 1\}$ and $N_0 \in \{100, 100, 10^6\}$ for each method. For all experiments, we set the preference of the advised CBBA state-action pairs to 100. τ was set to 1 for the actor. Figure 4 depicts the performance of iCCA and Actor-Critic averaged over 60 runs. The Y-axis shows the cumulative reward, while the X-axis represents the number of interactions. Each point on the graph is the result of running the greedy policy with respect to the existing preferences of the actor. For iCCA, risky moves again were replaced by the CBBA baseline solution. Error bars represent the standard error with $\%90$ confidence interval. In order to show the relative performance of these methods with offline techniques, the optimal and CBBA solutions are

[3] This computation took about a day and can not be easily scaled for larger sizes of the problem.

highlighted as lines. It is clear that the actor-critic performs much better when wrapped into the iCCA framework and performs better than CBBA alone. The reason is that CBBA provides a good starting point for the actor-critic to explore the state space, while the risk analyzer filters risky actions of the actor which leads into catastrophic scenarios. Figure 5 shows the optimality of iCCA and Actor-Critic after 10^5 steps of interaction with the domain and the averaged optimality of CBBA through $10,000$ trials. Notice how the integrated algorithm could on average boost the best individual optimality performance of both individual components by $\%10$.

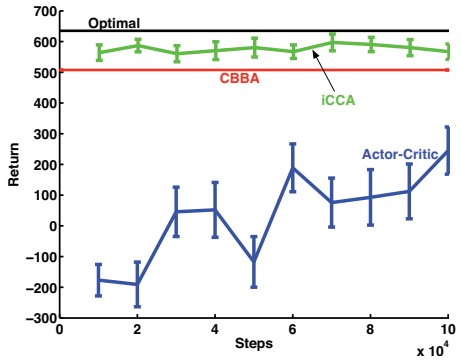

Figure 4: A comparison of the collective rewards received when strictly following plans generated by CBBA alone, actor-critic reinforcement learning outside of the iCCA environment, i.e. without initialization and guidance from CBBA, and the result when these are coupled via the iCCA framework are all compared against the optimal performance as calculated via dynamic programming

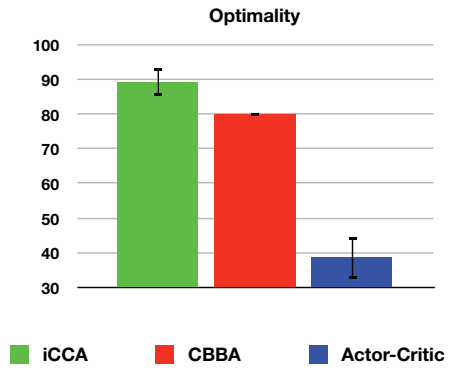

Figure 5: A comparison of optimality for CBBA alone, actor-critic reinforcement learning outside of the iCCA environment, i.e. without initialization and guidance from CBBA, and the result when these are coupled via the iCCA framework. The optimal result was found via dynamic programming.

Conclusions

In conclusion, we introduced a method for learning and adapting cooperative control strategies in real-time stochastic domains. Our framework of choice was an instance of the intelligent cooperative control architecture (iCCA) presented in (Redding et al. 2010). A "safe" plan was generated by the Consensus-Based Bundle Algorithm (Choi, Brunet, and How 2009), which initialized a policy which was then incrementally adapted by a natural actor-critic learning algorithm to increase planner performance over time. We successfully demonstrated the performance of our approach by simulating limited-fuel UAVs aiming for stochastic targets in an uncertain world.

Acknowledgments

This research was generously supported by Boeing Research & Technology, Seattle, WA and by AFOSR grant FA9550-08-1-0086.

References

Bertsekas, D., and Tsitsiklis, J. 1996. *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific.

Bhatnagar, S.; Sutton, R. S.; Ghavamzadeh, M.; and Lee, M. 2007. Incremental natural actor-critic algorithms. In Platt, J. C.; Koller, D.; Singer, Y.; and Roweis, S. T., eds., *NIPS*, 105–112. MIT Press.

Choi, H.-L.; Brunet, L.; and How, J. P. 2009. Consensus-based decentralized auctions for robust task allocation. *IEEE Trans. on Robotics* 25 (4):912 – 926.

H. Brendan McMahan, Geoffrey J. Gordon, A. B. 2003. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the Twentieth International Conference on Machine Learning*.

Howard, R. A. 1960. Dynamic programming and markov processes.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101:99–134.

Littman, M. L.; Dean, T. L.; and Kaelbling, L. P. 1995. On the complexity of solving markov decision problems. In *In Proc. of the Eleventh International Conference on Uncertainty in Artificial Intelligence*, 394–402.

Murphey, R., and Pardalos, P. 2002. *Cooperative control and optimization*. Kluwer Academic Pub.

Ponda, S.; Redding, J.; Choi, H.-L.; Bethke, B.; How, J. P.; Vavrina, M.; and Vian, J. L. 2010. Decentralized planning for complex missions with dynamic communication constraints. In *submitted to American Control Conference*.

Puterman, M. L. 1994. Markov decision processes.

Redding, J.; Undurti, A.; Choi, H.; and How, J. 2010. An Intelligent Cooperative Control Architecture. In *American Control Conference, to appear*.

Russell, S., and Norvig, P. 2003. Artificial Intelligence, A Modern Approach.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press.