Towards Graphical Partially Observable Monte-Carlo Planning

Julius Pfrommer

Vision and Fusion Laboratory
Institute for Anthropomatics
Karlsruhe Institute of Technology (KIT), Germany
julius.pfrommer@kit.edu

Technical Report IES-2015-09

Abstract: Sample-based online algorithms are state of the art for solving Partially Observable Markov Decision Problems (POMDP). But also the state of the art solver POMCP still suffers from the curse of dimensionality and curse of history. In Distributed POMDP, independent agents jointly optimise their actions under some coordination mechanism where every agent has access to a subset of the observations. In this work, we introduce Graphical POMDP (GPOMDP) drawing from existing Distributed POMDP appraoches as well as graph-based formulations as found in graphical probabilistic models. Further, we propose the Graphical POMCP (GPOMCP) algorithm that combines POMCP with message passing similar to the Belief Propagation (BP) algorithm from Graphical Probabilistic Models. In preliminary tests, GPOMCP shows good performance on a common Distributed POMDP benchmark.

1 Introduction

Partially Observable Markov Decision Problems (POMDP) [KLC98] capture planning scenarios with both nondeterministic system dynamics and uncertainty about the current state. The Partially Observable Monte-Carlo Planning (POMCP) algorithm [SV10] is a current state of the art technique for solving POMDP. On some benchmarks, it led to several magnitudes of speed improvements compared to previous point-based planning methods such as SARSOP [KHL08]. A recent publication by Amato and Oliehoek [AO15] applies POMCP to Distributed POMDP in the Dec-POMDP model. In this technical report, we present preliminary findings on a novel variation of POMCP for Distributed

POMDP where value estimates are propagated in a graph structure representing a decomposition of the POMDP. The resulting algorithm combines ideas from POMCP and message-passing approaches from Graphical Probabilistic Models.

The paper is structured as follows: First, we present relevant background material in Section 2. Then, in Section 3, we present the Graphical Partially Observable Markov Decision (GPOMDP) model and in Section 4 the Graphical Partially Observable Monte-Carlo Planning (GPOMCP) algorithm. The runtime behavior of the algorithm is evaluated based on a common benchmark scenario in Section 5. The paper concludes in Section 6 with a summary and future outlook.

2 Background

2.1 Partially Observable Markov Decision Problems

Markov Decision Processes (MDP) represent scenarios for decision-making under uncertainty where the system dynamics are dependent only on its current state and chosen actions, but not on the past history that lead to the current state [Put94]. Partially Observable Markov Decision Problems (POMDP) generalize MDP to situations where the underlying state is latent and can only be observed indirectly [KLC98]. Partially Observable Monte-Carlo Decision Problems (POMDP) are defined as n-tuples

$$\langle V = S \cup A \cup R \cup O, \{\mathcal{X}_v\}, P_S^0, P_S, P_O, R, T \rangle \,.$$

The variables $v \in V$ are made up of the latent-state S, actions A, rewards R and observations O. In every period $t \in \{1, \dots, T\}$, they take on a value from a discrete domain in the state vector $\underline{x}^t \in \mathcal{X}_V = \times_{v \in V} \mathcal{X}_v$. For brevity, the components of the state vector are referred to as s^t, a^t, r^t and o^t. Alternatively, we index components with a subscript, e.g. $\underline{x}_J^t$ for some set $J \subseteq V$. The initial latent-state s^0 is drawn from the distribution P_S^0. Afterwards, at the beginning of every period t, the actions a^t are selected. The following latent-state, as well as the resulting rewards and observations are drawn according to

$$P_S(s^{t+1} \mid s^t, a^t), \quad r^t = R(s^t, a^t, s^{t+1}), \quad P_O(o^t \mid s^t, a^t, s^{t+1}) \,.$$

The latent-state variables are not known to the choice-making entity and only the history $h^t = (\mathcal{X}_A \times \mathcal{X}_R \times \mathcal{X}_O)^{t-1}$ of previous actions, rewards and observations can be used for action selection. Histories can be concatenated with values from the current period and the empty history is written as ϵ. A policy π is a deterministic mapping from the history of previous periods to current actions. Rewards are included in the history so that we may refer to them in algorithms that improve policies based on sampled scenario rollouts. But rewards are treated as unobservable for the policies themselves. The value of a policy is the expected reward over the T periods given by Bellman's equation. The goal of solving a POMDP is to find a policy of maximum value.

$$V_\pi^t(s^t, h^t) = \mathbb{E}\left[\sum_{\rho \in \underline{x}_R^t} \rho + \underset{t<T}{\mathbb{1}} V_\pi^{t+1}(s^{t+1}, h^t a^t r^t o^t) \,\middle|\, a^t = \pi(h^t)\right]$$

$$V(\pi) = \sum_{s \in \mathcal{X}_S} P_S^0(s) V_\pi^0(s, \epsilon)$$

The so-called Q-value is defined as the expected rewards for choosing action a after an observed history h and following the policy π afterwards.

$$Q_\pi^t(h^t, a^t) = \sum_{s^t} P(s^t \mid h^t)\, \mathbb{E}\left[\sum_{r \in r^t} r + \underset{t<T}{\mathbb{1}} V_\pi^{t+1}(s^t, h^t a^t r^t o^t)\right]$$

2.2 Monte-Carlo Tree Search and POMCP

Monte-Carlo Tree Search (MCTS) [BPW$^+$12] is a recent approach to search for an optimal policy in multi-period scenarios. MCTS cyclically generates example rollouts and updates the estimation for the value of actions at a specific position in the scenario tree. If the scenario is deterministic, then the scenario tree has a branching factor according the number of possible action-choices in every period. If the scenario is stochastic, then the possible state-transitions lead to an additional branching. Algorithm 2.1 shows the generic MCTS algorithm with rollout for POMDP scenarios. The performance of MCTS largely depends on the selection of actions for exploration. The question is whether to select actions that have shown good performance in the past, or whether to analyze less-explored branches of the scenario tree that might contain some undiscovered potential. Recently, the Upper-Confidence Bound (UCB) princple originally developed for bandit-games [ACBF02] has become a popular choice. In UCB, actions are evaluated according to their past performance with an additional bias that favors actions for which less empirical evidence is available.

$$\widehat{Q}[ha] + \alpha\sqrt{\frac{\log(n[h])}{n[ha]}}$$

Algorithm 2.1 The Monte-Carl Tree Search Algorithm with rollout for POMDP

```
1: procedure MCTS
2:     INITIALIZE
3:     while enough time do
4:         $h \leftarrow$ ROLLOUT
5:         UPDATE($h$)
6:     end while
7:     return BESTACTION
8: end procedure
1: procedure ROLLOUTPOMDP
2:     $h \leftarrow \epsilon$
3:     $s \sim P_S^0$
4:     for $t \in \{1, \ldots, T\}$ do
5:         $a \leftarrow$ EXPLORATIONACTION($h$)
6:         $s' \sim P_S(s, a)$; $r \leftarrow R(s, a, s')$; $o \sim P_O(s, a, s')$
7:         $h \leftarrow haro$
8:         $s \leftarrow s'$
9:     end for
10:    return $h$
11: end procedure
```

Variables indexed with square brackets denote maps. The default value for any index is zero. The map $\widehat{Q}$ contains estimations of Q-values and is updated after every rollout. The counter n is increased by one every time a certain history occurs during the rollouts. The exploration/exploitation tradeoff can be tuned with the weighting parameter α. This leads to an exploration strategy that prunes less promising branches implicitly. Still, convergence is guarantueed for many algorithms employing UCB since every branch is visited infinitely often in the limit. UCB-based exploration has led to huge performance increases for many game-playing AIs, in particular those for games with large branching factors, such as Go [GKS$^+$12]. Partially Observable Monte-Carlo Planning (POMCP, [SV10]) is the application of UCB to solving POMDP in the MCTS framework (see Algorithm 2.2). Note that the value estimation update in POMCP is relatively simplistic. It just gives the average rewards experienced after a given history/action combination during the rollouts. This converges to the actual Q-value of optimal play since the UCB-based action selection will choose optimal actions infinitely more often in the limit. More complex update mechanisms, e.g. updating beliefs on the value of an action in a Bayesian setting, may lead to better estimations for a given set of samples. However, empirical evidence shows an advantage for algorithms with fast updates that process more rollout/update cycles for the same computational effort. Competitive implementations of POMCP process several hundred thousand rollouts per second.

Algorithm 2.2 The POMCP algorithm in the MCTS framework

```
1: procedure InitializePOMCP
2:     $n[\cdot] \leftarrow 0$; $\widehat{Q}[\cdot] \leftarrow 0$
3: end procedure
1: procedure UpdatePOMCP($h$)
2:     $\rho \leftarrow 0$
3:     for $t \in \{T, \ldots, 1\}$ do
4:         $\rho \leftarrow \rho + \sum_{r \in R} x_r^t$
5:         $n[h^t] \leftarrow n[h^t] + 1$
6:         $n[h^t a^t] \leftarrow n[h^t a^t] + 1$
7:         $\widehat{Q}[h^t a^t] \leftarrow \widehat{Q}[h^t a^t] + \frac{\rho - \widehat{Q}[h^t a^t]}{n[h^t a^t]}$
8:     end for
9: end procedure
1: procedure ExplorationActionPOMCP($h$)
2:     if $\exists a : n[ha] = 0$ then
3:         return $\sim U(\{a : n[ha] = 0\})$
4:     end if
5:     return $\arg\max_a \widehat{Q}[ha] + \alpha\sqrt{\frac{\log(n[h])}{n[ha]}}$
6: end procedure
1: procedure BestActionPOMCP
2:     return $\arg\max_{a \in \mathcal{X}_A} \widehat{Q}[a]$
3: end procedure
```

2.3 Decentralized and Multiagent POMDP

Decentralized planning in POMDP settings requires superexponential time to solve in the worst case [BGIZ02]. Thus, recent work has focused on heuristic solvers and specific classes of Distributed POMDP where some underlying structure can be exploited. Examples for such models are ND-POMDP [NVTY05], Dec-POMDP [OSWV08] and I-POMDP [GD05]. Readers are referred to [SZ08] for an in-depth discussion and equivalence results. A comparison between Dec-POMDP, ND-POMDP and our approach GPOMDP is given at the end of the following section.

3 Graphical POMDP

Graphical POMDP (GPOMDP) are defined as n-tuples comprised of a standard POMDP with an additional set of agents I.

$$\langle V = S \cup A \cup R \cup O, \{\mathcal{X}_v\}, P_S^0, P_S, P_O, R, T, I \rangle \,.$$

Agents $I \in 2^V$ are choice-making entities. Every agent $i \subseteq V \setminus S$ has access to a subset of the actions $A_i = A \cap i$, rewards $R_i = R \cap i$ and observations $O_i = O \cap i$. Agents may overlap by sharing some variables $(i, j) \in I^2, \Delta_{ij} = i \cap j$. Overlapping agents are in the set of neighbours $N(i) = \{j \in I : \Delta_{ij} \neq \varnothing\}$. In distributed settings, variable sharing can be achieved via lossless communication of action choices and observations. Every agent i has access to a reduced history $h_i^t \in (\mathcal{X}_i)^{t-1}$ in time period t. The action-choices are made by either assigning disjoint controlled actions $A_i^C \subseteq A_i$ and local policies $\pi_i^t : (\mathcal{X}_i)^{t-1} \to \mathcal{X}_{A_i^C}$ to every agent or via some additional online coordination mechanism.

Relation to Dec-POMDP In Decentralized POMDP (Dec-POMDP) [BGIZ02], all action and observation variables are assigned to exactly one agent, so that $A = \times_i A_i,\ O = \times_i O_i$. Agents have access to their own actions and observations only. So all Dec-POMDP are GPOMDP, but GPOMDP are Dec-POMDP if and only if all agents are disjoint, i.e. $\forall (i,j) \in I^2, i \neq j \Rightarrow i \cap j = \varnothing$. However, it is possible to construct additional actions and observations in Dec-POMDP in a way that mimicks communication channels between agents [SZ08]. In consequence, for any given GPOMDP, a Dec-POMDP can be constructed that recovers sharing of actions via communication channels. Shared observations can be achieved by duplication of observation variables $P(x_{o'} \mid x_o) = \delta_{\{x_{o'} = x_o\}}$. This structure is however lost in the general Dec-POMDP and has to be rediscovered by the solvers.

Relation to ND-POMDP Networked Distributed POMDP (ND-POMDP) [NVTY05]) factorize the latent-state so that the variables are either controlled by exactly one agent or are unobservable $S = \times_i S_i \times S_u$. Action and observation variables are each assigned to a unique agent, so that $A = \times_i A_i,\ O = \times_i O_i$.

The agents are transition and observation independent and coupled only by the reward function.

$$P(s^{t+1} \mid s^t, a^t) = P(s_u^{t+1} \mid s_u^t) \prod_{i \in I} P(s_i^{t+1} \mid s_i^t, a_i^t)$$
$$P(o^t \mid s^t, a^t) = \prod_{i \in I} P(o_i^t \mid s_u^t, s_i^t, a_i^t)$$
$$R(s, a) = \sum_{c \in \mathcal{C}} R_c(u, s_c, a_c)$$

The reward function R in ND-POMDP is made up of components that depend on a cluster of agents $\mathcal{C} \in 2^I$. With a slight abuse of notation, the latent-state of a cluster is $s_c \in \times_{v \in \cup_c i} \mathcal{X}_v$ and similarle for actions. The resulting locality of interaction between agents is characterized by a graph $G_{\mathcal{C}} = (I, C := \{(i,j) \in I^2 : \exists c \in \mathcal{C}, i \in c, j \in c\})$. Most papers assuming the ND-POMDP model further allow agents to communicate observations with some or all the other agents. According to this definition, all ND-POMDP are GPOMDP. The reverse is not necessarily true, since GPOMDP do not impose transition and observation independence between agents.

4 Graphical POMCP

We now introduce the Graphical Partially Observable Monte-Carlo Planning (GPOMCP) algorithm for solving GPOMDP online in a distributed fashion. It draws from two main sources of inspiration: The POMCP algorithm [SV10], one of the state of the art online POMDP solvers, and message-passing approaches based on the Generalized Distributive Law (GDL) [AM00, KFL01]. Variants of the latter are known for example the Belief Propagation (BP) algorithm [Pea88] in Graphical Probabilistic Models and the Max-Plus algorithm used for Distributed Constraint Optimization (DCOP) [PF05].

Assume a GPOMDP where the agents form a hypergraph tree $H = (I, E)$ with agents as vertices and neighborhood relations as edges $E = \{(i,j) \in I^2 : j \in N(i)\}$. This assumption is motivated by BP, where convergence is also only guaranteed on trees. Still, BP on loopy graphs often achieves good results in practice [YFW$^+$00]. The theoretical insight and empirical evidence to draw the analogy for GPOMCP is however not in the scope of this contribution. Since the visible actions A_i overlap, agents can select contradicting sets of actions. In Algorithm 4.1 this is resolved by giving precedence to agents who make their choice later. That is necessary, since the UCB-based action selection may otherwise lead to situations where two agents have each a strong preference for different but overlapping set of actions. Selecting a mixture of the two action sets can lead to a blocking situations where the same actions are selected indefinitely in a row as both agents were not able to see the action for which they have a strong preference. By giving precedence to later agents, at least one agent gets to see the results of his preferred action set and will eventually move on to a different choice.

Algorithm 4.1 Action selection in GPOMCP

```
 1: procedure ExplorationActionGPOMCP($h$)
 2:     $\underline{x}_A \in \mathcal{X}_A$
 3:     for $i \in I$ do
 4:         $\underline{y}_{A_i} \leftarrow$ AgentExplorationActionGPOMCP($i, h_i$)
 5:         for $a \in A_i$ do
 6:             $x_a \leftarrow y_a$
 7:         end for
 8:     end for
 9:     return $\underline{x}_A$
10: end procedure
 1: procedure AgentExplorationActionGPOMCP($i, h_i$)
 2:     if $\exists a_i : n[h_i a_i] = 0$ then
 3:         return $\sim U(\{a_i : n[h_i a_i] = 0\})$
 4:     end if
 5:     return $\arg\max_{a_i} \widehat{Q}_i[h_i a_i] + \alpha\sqrt{\frac{\log(n[h_i])}{n[h_i a_i]}}$
 6: end procedure
 1: procedure BestActionGPOMCP($h$)
 2:     $\underline{x}_A \in \mathcal{X}_A$
 3:     for $i \in I$ do
 4:         $\underline{y}_{A_i} \leftarrow \arg\max_{a_i} \widehat{Q}_i[h_i a_i]$
 5:         for $a \in A_i$ do
 6:             $x_a \leftarrow y_a$
 7:         end for
 8:     end for
 9:     return $\underline{x}_A$
10: end procedure
```

The update procedure of the POMCP algorithm consists of a message passing and an estimation update phase. The message passing phase uses a forward/backward schedule. Recall that the agents form a tree hyper-graph. The schedule $\mathcal{S}(I)$ contains an ordered list of sender/receiver relations where the agents i wait until they have received messages from their $|N(i)| - 1$ children in the graph. Then they send out a messages to their parent. Once they have received a message from their parent, messages are sent out to all children. This is called the forward/backward schedule since the message exchange starts at the leaf of the tree, propagates through the graph and finally returns to the leafs. Note that the schedule can be efficiently implemented to run in parallel on distributed agents. But we omit discussing this possibility in this text.

Algorithm 4.2 Initialization and estimation update in GPOMCP

```
1: procedure InitializeGPOMCP($I$)
2:     $n[\cdot] \leftarrow 0$
3:     $\forall i \in I, \widehat{R}_i[\cdot] \leftarrow 0,\ \widehat{Q}_i[\cdot] \leftarrow 0$
4:     $\forall i \in I, \forall j \in N(i),\ n_{ij}[\cdot] \leftarrow 0,\ m_{i\to j}[\cdot] \leftarrow 0$
5: end procedure
1: procedure UpdateGPOMCP($h$)
2:     $\forall i \in I,\ \rho_i \leftarrow 0$
3:     $\forall i \in I, \forall j \in N(i),\ \rho_{ij} \leftarrow 0$
4:     for $t \in \{T, \dots, 1\}$ do
5:         for $(i,j) \in \mathcal{S}(I)$ do
6:             $\rho_{ij} \leftarrow \rho_{ij} + \sum_{r \in (R \cap O_i)\setminus \Delta_{ij}} x_r^t$
7:             $n_{ij}[h_{ij}^t a_{ij}^t] \leftarrow n_{ij}[h_{ij}^t a_{ij}^t] + 1$
8:             $m_{i\to j}[h_{ij}^t a_{ij}^t] \leftarrow m_{i\to j}[h_{ij}^t a_{ij}^t] + \frac{\left(\rho_{ij} + \sum_{l \in N(i)\setminus j} m_{l\to i}[h_{il}^t a_{il}^t]\right) - m_{i\to j}[h_{ij}^t a_{ij}^t]}{n_{ij}[h_{ij}^t a_{ij}^t]}$
9:         end for
10:        for $i \in I$ do
11:            $\rho_i \leftarrow \rho_i + \sum_{r \in R \cap O_i} x_r^t$
12:            $n[h_i^t] \leftarrow n[h_i^t] + 1$
13:            $n[h_i^t a_i^t] \leftarrow n[h_i^t, a_i^t] + 1$
14:            $\widehat{R}_i[h_i^t a_i^t] \leftarrow \widehat{R}_i[h_i^t a_i^t] + \frac{\rho_i - \widehat{R}_i[h_i^t a_i^t]}{n[h_i^t a_i^t]}$
15:            $\widehat{Q}_i[h_i^t a_i^t] \leftarrow \widehat{R}_i[h_i^t a_i^t] + \sum_{j \in N(i)} m_{j\to i}[h_{ij}^t a_{ij}^t]$
16:        end for
17:    end for
18: end procedure
```

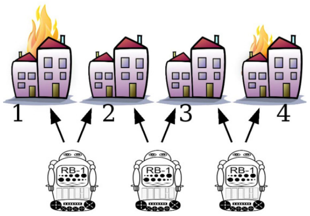

Figure 5.1: Illustration of the Firefighting benchmark problem taken from [OSWV08]

Similar to POMCP, the reward from the later periods is accumulated in a scalar ρ_i. However, ρ_i takes only the rewards into account that are visible to the agent i. In addition, we accumulate a set of summed rewards ρ_{ij} for every neighbour with the rewards that were visible to i, but not to the neigbhour $j \in N(i)$. The message from agent i to agent j then becomes an estimate for the amount of rewards the subtree behind the edge (i, j) is expected to receive during the current period and afterwards, conditioned on the joint history $h^t_{ij} \in (\mathcal{X}_{\Delta_{ij}})^t$ and action $a_{ij} \in \mathcal{X}_{A \cap \Delta_{ij}}$. The messages and the locally visible rewards are then used to update the local estimates for the true Q-Value.

5 Benchmark Results

We compare our results on the Firefighting domain introduced in [OSWV08]. It models a team of n firefighters that have to extinguish fires in a row of $n_H = n+1$ houses. In this work, we will take $n = 8$ and $T = 3$. Each house H has a fire level $l_H \in \{0, \ldots, 2\}$. The latent-state is an assignment of fire levels to every house. See Figure 5.1 for an illustration of the scenario.

At every time step, each firefighter f can choose to fight fires at house f or $f+1$. If a house H is burning ($l_H > 0$) and no firefighting agent is present, its fire level will increase by one point with probability 0.8 if any of its neighboring houses are burning, and with probability 0.4 if none of its neighbors are on fire. A house that is not burning can only catch fire with probability 0.8 if one of its neighbors is on fire. When two firefighters are at the same house, they will extinguish any present fire completely, setting the house's fire level to zero. A single agent present at a house will lower the fire level by one point with probability 1 if no neighbors are burning, and with probability 0.6 otherwise. Each firefighter can only observe whether there is a fire or not at its location. Fire is observed with probability 0.2 if $l_H = 0$, with probability 0.5 if $l_H = 1$, and with probability 0.8 otherwise. Rewards are $2 - l_H$ for each house according to the burn level reached in that period. Initially, the fire level l_H of each house is drawn from a uniform distribution. Each firefighter is represented by an agent that perceives the actions, rewards and observations of himself and of his neighbours $f - 1$ and $f + 1$ if they exist.

Figure 5.2 compares the performance of three approaches in the firefighting benchmark scenario. In all cases, we estimate Q-values with the given number of rollouts. Then, the resulting performance is evaluated as the mean reward of 500 rollouts where we choose the action with maximum estimated Q-value in every period. The first appraoch employs standard POMCP where all actions, rewards and observations are visible to a central learning an planning entity. The second approach is the FT-FV-MPOMDP algorithm from [AO15] where the first application of MCTS to Dec-POMDP in was given. The factored tree (FT) version of their algorithm is equal to GPOMCP when omitting the message passing step, therefore having each agent build up a separate evaluation. The last approach compared in the benchmark is our GPOMCP. All algorithms ran with an exploration weight factor of $\alpha = 10$.

The benchmark shows the results for running the rollout and update step for the given number of repetitions and then using the resulting Q-value estimates to guide the action selection during the T periods. Comparing the algorihtms in a pure online-planning scenario is planned for the near future.It can be seen that GPOMCP outperforms the other algorithms in the firefighting scenario. Both GPOMCP and FV-POMCP converge to a stable solution within 50,000 rollouts.

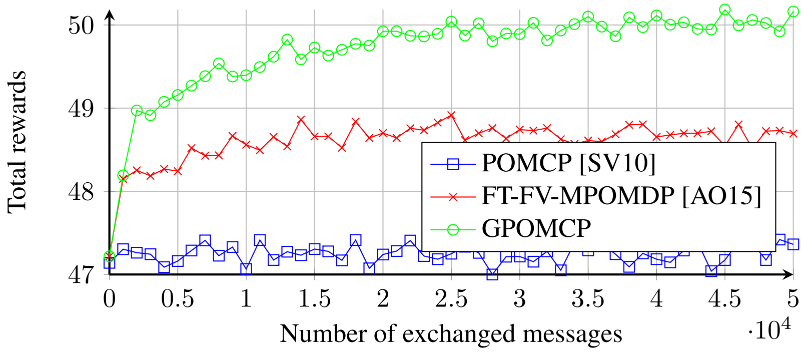

Figure 5.2: Benchmark results of the firefighting scenario.

However, GPOMCP achieves a substantially better solution. This is due to the coordination where agents also consider expected rewards even in parts of the POMDP that are not visible to them. The convergence of POMCP is even slower than what the graph indicates since POMCP has to iterate over all possible joint actions in every period. Since the number of joint actions grows exponentially in the number of action variables, the rollouts are computationally more expensive for POMCP.

6 Summary

In this contribution, we presented the Graphical Partially Observable Markov Decision Problem (GPOMDP) to capture the structure of Distributed POMDP where agents may overlap in their observable variables. Whe further introduced an algorithm for solving GPOMDP based on the well-known POMCP algorithm [SV10] and the message passing approach known from Graphical Probabilistic Models. First empirical evidence hints at favorable convergence properties for factorizable POMDP. Future work will apply GPOMCP to further benchmark problems from the literature and characterise its theoretical properties. Furthermore, we intend to apply GPOMCP to problems with continuous state and action spaces according to the principles developed in [BDMB13].

Bibliography

[ACBF02] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[AM00] Srinivas M Aji and Robert J McEliece. The generalized distributive law. *Information Theory, IEEE Transactions on*, 46(2):325–343, 2000.

[AO15] Christopher Amato and Frans A Oliehoek. Scalable Planning and Learning for Multiagent POMDPs. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[BDMB13] Lucian Busoniu, Andrew Daniels, Rémi Munos, and Robert Babuska. Optimistic planning for continuous-action deterministic systems. In *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2013 IEEE Symposium on*, pages 69–76. IEEE, 2013.

[BGIZ02] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4):819–840, 2002.

[BPW^{+}12] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon

Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[GD05] Piotr J Gmytrasiewicz and Prashant Doshi. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, pages 49–79, 2005.

[GKS+12] Sylvain Gelly, Levente Kocsis, Marc Schoenauer, Michele Sebag, David Silver, Csaba Szepesvári, and Olivier Teytaud. The grand challenge of computer go: Monte carlo tree search and extensions. *Communications of the ACM*, 55(3):106–113, 2012.

[KFL01] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2):498–519, 2001.

[KHL08] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*, volume 2008. Zurich, Switzerland, 2008.

[KLC98] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

[NVTY05] Ranjit Nair, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. In *AAAI*, volume 5, pages 133–139, 2005.

[OSWV08] Frans A Oliehoek, Matthijs TJ Spaan, Shimon Whiteson, and Nikos Vlassis. Exploiting locality of interaction in factored dec-pomdps. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 517–524. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[Pea88] Judea Pearl. Probabilistic reasoning in intelligent systems: Networks of plausible inference. 1988.

[PF05] Adrian Petcu and Boi Faltings. DPOP: A Scalable Method for Multiagent Constraint Optimization. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 266–271, 2005.

[Put94] Martin L Puterman. Markov decision processes: Discrete stochastic dynamic programming. 1994.

[SV10] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

[SZ08] Sven Seuken and Shlomo Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, 2008.

[YFW+00] Jonathan S Yedidia, William T Freeman, Yair Weiss, et al. Generalized belief propagation. In *NIPS*, volume 13, pages 689–695, 2000.