VIII. SUPPLEMENTARY MATERIALS

A. Problem descriptions

We will provide a brief description of each of the IPPC problems used for the experiments.

- **Sailing Wind**: Originally proposed by Robert Vanderbei [15], the goal of Sailing Wind is to move a ship that starts at $(1, 1)$ on an $n \times n$ grid to (n, n) with minimal cost. There is no consistent use of a transition and reward function throughout the literature. There may just be two available actions (*down*, *right*) [9] or up to seven (each adjacent cell except the one facing a stochastic wind direction) [1]. The cost of each action is dependent on the current wind direction which stochastically changes its direction at each step independent of the player's actions.
- **Game of Life**: The original game of life by John Conway [5] is a cellular automaton and modified into a stochastic MDP as a test problem for the International Probabilistic Planning Competition [12] by introducing noise to the deterministic state transition, setting the current number of alive cells as the reward, and allowing the agent to choose one cell which will contain a living cell with a high probability. States are elements in $\{0, 1\}^{n \times n}$ describing whether there is an alive cell at each cell on a grid. To reduce the action space that scales quadratically which the grid length, we allow only a subset of the original actions, which is to specify one alive cell that is prevented from dying.
- **SysAdmin**: Used as a test problem for the IPPC 2011, a SysAdmin instance is a graph (describing a network topology) with $n \in \mathbb{N}$ vertices. The state space is $\{0, 1\}^n$ (describing which machines are currently operating) and the action space is $\{1, \ldots, n\}$ (describing with machine to reboot). At each step, the reward is dependent on the machines that are currently working, a reboot causes the rebooted machine to have a high chance of working in the next step. Machines can randomly fail at each step, however this probability is increased when a neighbor fails.
- **Navigation**: Navigation was a test problem for the International Probabilistic Planning Competition 2011 [12]. The goal is to move a robot on an $n \times m$ grid from $(n, 1)$ to (n, m) in the least number of steps. The robot may move to any of the four adjacent tiles, however, each tile is assigned a unique probability with which the robot is reset back to $(n, 1)$. At each step the agent incurs a constant negative reward, making the objective to reach the goal state as quickly as possible.
- **Academic Advising**: The Academic Advising domain was used for the IPPC 2014 [7]. The agent is a student whose goal is to pass certain academic classes. Formally, the state is an element in $\{H, L, F, NT\}^n$ (representing for each course whether it has not been taken, failed, or passed with a low or high grade) and the agent's action is to choose a course to take. The course outcome depends on the states of the prerequisite courses. The episode ends, when all courses are passed, and while not all mandatory courses, a subset of all courses, are passed, the agent incurs a constant penalty per step.
- **Tamarisk**: Tamarisk is yet another problem from the IPPC 2014 [7] which models the expansion of an invasive plant in a river system. The river system is modelled as a chain of reaches where each reach contains a number of slots that may be unoccupied, occupied by a native plant, or occupied by the invasive Tamarisk plant. Both plant types spread stochastically to neighboring states with a higher probability of spreading downstream. At each time step, the agent chooses an action for one reach, which are doing nothing, eradicating Tamarisk, or restoring a native plant. The action chosen at a reach is applied to all slots in that reach. Except for the do-nothing action, all actions can randomly fail. The agent has to balance the action's costs with the penalties incurred for existing Tamarisk plants.
- **Racetrack**: Racetrack was first described by Martin Gardner [6] where the goal is to move a car in as few steps as possible to a goal-position on a graph starting from a random position. Each state is a tuple in $\mathbb{Z}^2 \times \mathbb{Z}^2$ that represents the current position and velocity vector. The available actions are adding a vector from $\{-1, 0, 1\}^2$ to the velocity vector (i.e. accelerating). At each step, with some probability, the chosen acceleration vector is replaced by $(0, 0)$. The state transitions by first adding the acceleration vector to the current velocity vector and then adding the velocity vector to the position vector. If a boundary or an obstacle has been hit, the car's position is reset to one of the initial positions with zero velocity. Some implementations however, set the velocity vector to zero and place the car at an empty tile closest to the crash position that is on the straight line connecting the last position to the crash position [2].
- **EarthObservation**: EarthObservation was a test problem for the IPPC 2018 which models a satellite orbiting earth. Formally, each state is a position on a 2-dimensional grid, representing the satellite's longitudinal position and the latitude the camera is aimed at as well as weather levels for some designated cells. At each step, the weather levels stochastically change independent of the agent's actions which are to idle, to take a photo of the current position, or increment/decrement the current cells y-position (i.e. shifting the camera focus). A reward is obtained if one of the designated cells is photographed with an amount depending on the cell's current weather condition.
- **Triangle Tireworld:** Tireworld was proposed as a test problem for the IPPC 2004 [18]. In the original goal-based version, the agent is a car that traverses a graph. At each step, the car may move to an adjacent node, change its tire, or load a tire. The goal is to reach a designated goal node. At each step, the car's tire may randomly break. If the car isn't carrying a spare tire, the goal can no longer be reached. Otherwise, if available, a spare tire (at most one can be carried) must be used to replace the current tire. Some nodes contain spare tires, which when the agent visits them, can be picked up.

- **CooperativeRecon**: This domain models a robot having to prove the existence of life on a foreign planet. The robot is modeled as moving on a 2-dimensional grid which contains a number of objects of interest and a base. If the agent is at an object of interest, it can survey the object for the existence of water and life. The probability of a positive result of the latter is dependent on whether water has been detected. If life has been detected, the agent may photograph the object of interest which is the only way to gain a reward. Each detector may break on usage making it either unusable or decreasing its chance of working. The detectors can be repaired at the base.
- **Manufacturer**: In this domain, the agent manages a manufacturing company. The agent's ultimate goal is to sell goods to customers. However, to sell a good, the agent has to first produce the good, which may require building factories and acquiring the necessary goods required for production. Additional difficulty comes from the fact that the goods' price levels vary stochastically.
- **Skills Teaching**: This domain models a student-teacher interaction, where the agent plays the role of the teacher. There is a fixed number of skills that form a directed graph of prerequisites. The student possesses one of three levels of sufficiency at each skill. The agent is rewarded for each skill being at the highest sufficiency and punished for each skill at the lowest sufficiency level. At each, step the agent may choose a skill for which to pose a question to the student or give the student a direct hint. The student can increase their sufficiency at that skill for correctly answering a question and lose sufficiency for answering wrong. The probability of getting a question right is dependent on the sufficiency of the skill's prerequisite. A hint can elevate the student to the medium sufficiency level directly but only if all prerequisites are at the highest sufficiency.
- **Traffic**: This problem models a traffic system in which the agent is tasked with controlling/advancing intersections with the goal of minimizing congestion. The traffic system is modeled as a directed graph and each vertex is either empty or occupied. Occupancy flows along the graph's edges except for some designated intersection edges where the flow is dependent on the intersection's state. The only stochasticity of this MDP arises in the form of cars spawning randomly at the designated perimeter vertices. The agent receives a reward equal to the negative number of occupied vertices that have one predecessor vertex that is also occupied.

B. $(\varepsilon_a, \varepsilon_t)$-OGA, an extension of OGA to non-exact abstractions

Preliminary experiments showed that in almost all domains, standard OGA-UCT exclusively detects only action equivalences but abstracts no states. We believe that this is primarily due to two reasons.

- Most environments have action spaces featuring at least 10 actions per state. Even when statistically speaking there is a high chance that one action finds a matching action in a potential to-be-grouped state, the probability of finding a match for each action decays exponentially in the number of actions.
- Since there is only node at depth zero, state abstractions could only ever be found at depth one of the search tree which themselves have to be bootstrapped of actions abstractions found at depth one. However, due to a high action branching and stochastic branching factor in most environments, the majority of depth one actions have not sampled all possible successors, making finding abstractions difficult.

Furthermore, in environments with a high stochastic branching factor, it is practically impossible for two actions to have sampled the exact set of successors, a necessary condition for OGA-UCT. The two above-mentioned problems cause OGA-UCT to miss out on correct abstractions.

Anand et al. [1] partly addressed this problem by introducing pruned OGA-UCT which ignores all successors of state-action pairs whose transition probability p is less than $\alpha \cdot p_{max}$ where p_{max} is the highest transition probability of all sampled successors of the state-action pair in question and $\alpha \in [0, 1]$ is some fixed parameter. For this paper however, we investigated the abstraction dropping schemes using $(\varepsilon_a, \varepsilon_t)$-OGA which we will describe now that uses the $(\varepsilon_a, \varepsilon_t)$-ASAP framework. This has three reasons. Firstly, pruned OGA does not allow for errors in the immediate reward. Secondly, pruned OGA is less flexible, as even an alpha value $\alpha = 1$ does not guarantee that all states and state-action pairs are grouped. Using the parameters $\varepsilon_a = \infty, \varepsilon_t = 2$, however, guarantees that the coarsest abstraction that simply groups everything, is found. Thirdly, the α value is less interpretable. Even using the value $\alpha = 1$ could result in pruned OGA-UCT being equivalent to OGA-UCT in some environments, e.g. those where all transition probabilities are equal. In contrast, the value $\varepsilon_t = 0$ means that transition probabilities have to perfectly match and the value $\varepsilon_t = \infty$ implies that the transitions are fully ignored when building the abstraction.

To incorporate the $(\varepsilon_a, \varepsilon_t)$-ASAP framework, we generalized the state-action pair update method as follows as inspired by Jiang et al. [9] (i.e. the following extension is equivalent to standard OGA-UCT for $\varepsilon_a = \varepsilon_t = 0$). We call the now-to-be-proposed method $(\varepsilon_a, \varepsilon_t)$-OGA. In the following, we say that two state-action pairs are **similar** if their transition error F is less than or equal to ε_t and their reward distance is less than or equal to ε_a. We also refer to the maximum of the reward distance and the transition distance as the **distance** between two state-action pairs. The following modification to the state-action pair update method is a heuristic that balances the stability and size of the abstract nodes

Firstly, each abstract Q node keeps track of its representative which is one of its original Q nodes. Furthermore, it is assigned a unique and constant ID at its creation. At its creation, an abstract node is assigned an ID equal to the total number of abstract nodes that have been created so far. The update now differentiates between two cases:

- **The state-action pair is its abstract node's representative**: If there is another abstract node to which the current state-action pair is similar and that abstract node is bigger or it is of the same size but has a bigger ID, then we transfer the current action pair to this abstract node. We randomly pick one of the remaining original nodes of the old abstract node as its new representative.
- **State-action pair is not a representative or it's the state-action pair's first update**: We check if the state-action pair is no longer similar to its abstract node's representative or if it is the first update call for that node. In that case, we find the abstract node with a similar representative and with minimal distance to the current Q node. If such an abstract node exists, we transfer the current state-action pair to this node.

C. MCTS and OGA-UCT parameters

TABLE VI: A list of the optimal exploration constants for MCTS for both the high variance and low variance setting that we described in Section VI-E. For the high variance setting we tested $\lambda_{highvar} \in \{0.5, 1, 2, 4, 8, 16, 24, 32\}$ and for the low variance we tested $\lambda_{lowvar} \in \{0.125, 0.25, 0.5, 1, 2, 4, 8, 16, 32, 64, 128, 256\}$. By l_{lowvar}, we denote the rollout lengths that we used in the low variance setting, and ε_a lists the immediate reward thresholds that we tested in our experiments. A rollout length of ∞ means rollout until the end of the episode.

Domain	$\lambda_{highvar}$	λ_{lowvar}	l_{lowvar}	ε_a
SysAdmin	2	4	5	$\{0, 1, 2\}$
Game of Life	1	2	5	$\{0, 1, 2\}$
Academic Advising	4	8	20	$\{0\}$
Tamarisk	1	1	10	$\{0, 0.5, 1\}$
Cooperative Recon	2	2	∞	$\{0, 0.5, 1\}$
Earth Observation	4	64	20	$\{0, 1, 2\}$
Manufacturer	2	2	5	$\{0, 10, 20\}$
Navigation	8	1	20	$\{0\}$
Racetrack	4	0.25	20	$\{0\}$
Sailing Wind	2	0.5	10	$\{0, 1, 2\}$
Skills Teaching	4	0.5	5	$\{0, 2, 3\}$
Traffic	1	1	5	$\{0, 1, 2\}$
Triangle Tireworld	2	4	20	$\{0\}$