31st International Conference on
Automated Planning and Scheduling

August 02 – 13, 2021

WIPC 2021

Proceedings of the

Workshop on the International Planning Competition

Preface

The International Planning Competition (IPC) is held every few years in the context of ICAPS. It empirically evaluates state-of-the-art planning systems on a number of benchmark problems. The goals of the IPC are to promote planning research, highlight challenges in the planning community and provide new and interesting problems as benchmarks for future research. The IPC has an important role in the ICAPS community, being a forum to compare different algorithmic ideas and implementations, and setting standards for research and evaluation in the area. Similar to the lineage of IPC workshops organised at ICAPS 2003, 2007, 2012, 2015, and 2019 this workshop aims to review the current status of the IPC, analyse the results of the last IPC (2020), and provide a venue for discussing aspects that will be helpful for preparing forthcoming competitions.

Since we have organised the last International Planning Competition – the IPC 2020, the duty falls on us to organise the WIPC 2021.

Gregor, Daniel, and Pascal
Organizers of the WIPC,
October 2021

Organizing Committee

Gregor Behnke	University of Freiburg, Germany
Daniel Höller	Saarland University, Saarbrücken, Germany
Pascal Bercher	The Australian National University, Canberra, Australia

Program Committee

Florian Geißer	The Australian National University, Canberra, Australia
Robert Mattmüller	University of Freiburg, Germany
Marcel Steinmetz	Saarland University, Saarbrücken, Germany
Michael Thielscher	The University of New South Wales, Australia
Scott Sanner	University of Toronto, Canada
Florian Geißer	The Australian National University, Canberra, Australia
Tim Niemüller	X, The Moonshot Factory, United States of America
Sunandita Patra	University of Maryland, College Park, United States of America

Invited Talks

In addition to the paper presentations, WIPC 2020 featured the following invited talks:

- IPC 2020 Planner Presentation: HyperTensioN (Maurício Cecílio Magnaguagno)
- IPC 2020 Planner Presentation: Lilotane (Dominik Schreiber)
- IPC 2020 Planner Presentation: pyHiPOP (Alexandre Albore)
- An ELO System for Skat and other Games of Chance (Stefan Edelkamp)

Table of Contents

Proceedings of the Workshop on the International Planning Competition at ICAPS 2021

Online Policy Improvement for Probabilistic Planning: Benchmarks and Baselines

Murugeswari Issakkimuthu and Alan Fern
School of EECS
Oregon State University
Corvallis, OR 97331, USA

Abstract

The goal of Online Policy Improvement (OPI) is to use a given base policy to compute a better policy via online planning. There are OPI algorithms that come with theoretical guarantees of policy improvement under ideal conditions. However, when the ideal conditions are not met in practice, these algorithms can result in policy degradation, i.e., the new policy can perform worse than the base policy. Our goal in this paper is to move towards a better understanding of the empirical performance of OPI algorithms. We propose benchmark problems and base policies and suggest evaluation metrics for OPI. We also present baseline results on the benchmark set for two OPI algorithms, which demonstrate the baselines are a solid starting point for comparison.

Introduction

Online planning is a practical approach to solving Markov decision problems with large state spaces. An action choice is made for the current state and the selected action is executed immediately, so action decisions need to be made only for the states visited during the online planning process. Online planning aims at computing a near optimal policy. Rather, online policy improvement (OPI) is online planning with the goal of computing a policy that performs better than a given base policy.

When a base policy is available, OPI can sometimes be a safer alternative to optimal planning. For example, attempts at optimal planning under computational limits may completely fail, while OPI may provide useful results. There are different approaches to online planning. Our focus is on search-based approaches, where the Q-values of actions at the current state are estimated via lookahead search. Online planning returns an action that maximizes the Q-value estimate, while OPI can return any action with a Q-value estimate greater than that of the base policy action.

There are OPI algorithms that come with theoretical guarantees of policy improvement under ideal conditions, e.g., policy rollout (Tesauro and Galperin (1997), Bertsekas and Tsitsiklis (1996)), nested rollout (Cazenave (2009)), parallel rollout (Chang, Givan, and Chong (2004)), Limited Discrepancy Forward Search Sparse Sampling (LD-FSSS) (Issakkimuthu, Fern, and Tadepalli (2020)). However, when the ideal conditions are not met in practice, OPI algorithms can result in policy degradation, i.e., the new policy can perform worse than the base policy.

Our goal in this work is to move towards a better understanding of the empirical performance of OPI algorithms. We propose benchmark problems and base policies and suggest evaluation metrics for OPI. Our benchmark set consists of 5 domains from past International Probabilistic Planning Competitions with 10 problems of varying levels of difficulty in each domain and 2 base policies of different qualities for each problem. The benchmark set will be made available as open source. We also present baseline results on the benchmark set for two classes of OPI algorithms. In particular, these classes include algorithms that can leverage transition probabilities when available or just use the ability to sample transitions. We show that these classes are able to cover different points in the performance trade-off space, making them useful for future comparisons.

Background and Related Work

We assume basic familiarity with Markov Decision Processes (MDPs). A discrete finite-horizon MDP is a tuple $\langle S, A, P, R, H\rangle$, where S is a finite set of states, A is a finite set of actions, $P : S \times A \times S \to [0, 1]$ is a state-transition function with $P_{ss'}(a)$ denoting the probability of reaching state s' from state s on action a and $\sum_{s' \in S} P_{ss'}(a) = 1$ for all $a \in A$, $R : S \times A \to \mathcal{R}$ is a real-valued reward function defined on state-action pairs and H is an integer representing the finite horizon.

A deterministic, non-stationary policy of the MDP is a time-dependent mapping from states to actions, i.e., $\pi = \{\mu_0, \mu_1, \ldots, \mu_{H-1}\}$, where $\mu_k : S \to A$ for $k = \{0, 1, \ldots, H-1\}$. The H steps-to-go value function of the policy is V^π_H, where

$$V^\pi_k(s) = R(s, \mu_{H-k}(s)) + \sum_{s' \in S} P_{ss'}(\mu_{H-k}(s)) \cdot V^\pi_{k-1}(s')$$

for $k = \{1, 2, \ldots, H\}$ and $V^\pi_0(s) = 0$ for all $s \in S$. The H steps-to-go Q-value function with respect to π is

$$Q^\pi_H(s, a) = R(s, a) + \sum_{s' \in S} P_{ss'}(a) \cdot V^\pi_{H-1}(s')$$

for all $s \in S$ and $a \in A$. A policy π' is said to be better than a policy π if $V_H^{\pi'}(s) \geq V_H^{\pi}(s)$ for all $s \in S$. A solution of the MDP is an optimal policy π^* with value function $V_H^*(s) = \max_\pi V_H^\pi(s)$ for all $s \in S$.

Online Planning

An MDP can be solved offline via approaches like value iteration, policy iteration or linear programming (Puterman (1994)). The offline solution techniques can be computationally expensive for MDPs with large state spaces. An alternative practical approach is online planning, where plan execution is interleaved with planning. Action decisions are made only for the initial state and the states visited subsequently in the online planning process.

Our focus is on search-based online planning, where the Q-values of actions at the current state are estimated via finite-horizon lookahead search. Typically, a search tree is built with the current state at the root followed by alternating layers of action nodes and state nodes. Leaf nodes are initialized and the values of internal nodes are computed from the values of their successor nodes. An action that maximizes the Q-value estimate is returned for the current state. There are variants of search-based online planning, e.g., Sparse Sampling (SS) (Kearns, Mansour, and Ng (2002)), Forward Search Sparse Sampling (FSSS) (Walsh, Goschin, and Littman (2010)), Monte Carlo Tree Search (MCTS) (Browne et al. (2012)).

Base Policies in Online Planning. Online planning does not require a base policy, but it can benefit from one if there is one available. MCTS algorithms typically use a default base policy to initialize the values of leaf nodes using one or more rollouts of the base policy. The well-known AlphaGo and AlphaZero programs (Silver et al. (2017, 2018)) use base policies to expand their search trees. (Nguyen et al. (2014)) use a base policy as an extended action at every node of the search tree to identify better actions at the nodes. (Pinto and Fern (2017)) use a partial policy that gives a subset of actions for every state to successfully prune actions in the search tree. All these approaches can be roughly viewed as a form of OPI, even though the goal is not just to perform better than the base policy.

Online Policy Improvement

Online Policy Improvement (OPI) has the goal of doing better than a given base policy, so a base policy is a required input for OPI algorithms. Once the Q-values of actions are estimated for the current state, OPI can return any action with a Q-value greater than that of the base policy action. OPI can therefore be done with as few as one off-policy action (non base-policy action) and the base policy action at the root. There are several existing OPI algorithms that come with theoretical guarantees of policy improvement. We discuss a few such algorithms below.

Policy Rollout. The policy rollout algorithm is an online implementation of a single offline policy improvement step over the base policy value function. The policy improvement step is based on the fact that actions with Q-values greater than the Q-value of the base policy action will be better than the base policy action for a given state. When the base policy is substituted with improved actions at one or more states, the resulting policy will be better than the base policy. In the online version, the Q-value of an action at the current state is estimated as an average over multiple base policy trajectories starting with that action. The estimates will get close to the actual values when the average is computed with a large number of trajectories of sufficient length. The policy rollout algorithm has been shown to be effective in different applications (Tesauro and Galperin (1997), Bertsekas and Castanon (1999)).

Nested Rollout. The nested rollout algorithm (Cazenave (2009)) is an online implementation of a sequence of iterations of the policy iteration algorithm, in contrast to the policy rollout algorithm that implements just one iteration of the policy iteration algorithm. The policy computed at each iteration of the policy iteration algorithm will be better than all the previous policies along the sequence. Hence nested rollout is guaranteed to return a better policy when the Q-values are estimated with a large number of simulated trajectories of sufficient length.

Parallel Rollout. The parallel rollout algorithm takes multiple base policies as input to compute a policy that is better than all the base policies (Chang, Givan, and Chong (2004)). It is an online version of the offline policy switching algorithm that returns for every state the action of the base policy with the highest value among all the base policies. The resulting policy is guaranteed to be better than all the base policies. Parallel rollout estimates the values of all base policies for the current state as the average over multiple trajectories of the base policy. The average must be computed over a large number of trajectories of sufficient length for parallel rollout to return a better policy.

Limited Discrepancy Forward Search Sparse Sampling (LD-FSSS). LD-FSSS is a version of FSSS (Walsh, Goschin, and Littman (2010)) with a class of choice functions called Limited Discrepancy Choice Function (LDCF) (Issakkimuthu, Fern, and Tadepalli (2020)). A choice function defines the off-policy actions (discrepancies) available at the internal nodes of the search tree. LDCF limits the number of discrepancies along each root-to-leaf path and the depth up to which discrepancies are allowed in the search tree. It also restricts the set of discrepancies for every state to be non-increasing with depth. The base policy action is expanded at all the internal nodes of the search tree. When leaf nodes are initialized to base policy values and action values are computed with all possible successors with the true state-transition probabilities, LD-FSSS is guaranteed to return a policy better than the base policy.

	Sysadmin		Game of Life		Tamarisk		Skill Teaching		Wildfire	
#	Bad	Good	Bad	Good	Bad	Good	Bad	Good	Bad	Good
1	171 ± 7	335 ± 5	76 ± 13	177 ± 10	-178 ± 21	-144 ± 20	65 ± 2	65 ± 2	-647 ± 308	-594 ± 251
2	250 ± 11	304 ± 9	82 ± 8	119 ± 9	-590 ± 29	-502 ± 25	-60 ± 0	75 ± 2	-9015 ± 387	-8998 ± 344
3	421 ± 15	554 ± 14	112 ± 7	137 ± 5	-264 ± 38	-222 ± 33	56 ± 15	81 ± 14	-1439 ± 412	-1529 ± 432
4	379 ± 9	487 ± 15	216 ± 17	327 ± 15	-804 ± 30	-681 ± 31	-64 ± 7	57 ± 17	-9050 ± 791	-8585 ± 801
5	518 ± 9	627 ± 17	236 ± 9	281 ± 8	-642 ± 44	-618 ± 37	-64 ± 19	-64 ± 19	-1010 ± 343	-754 ± 257
6	541 ± 15	575 ± 17	244 ± 6	276 ± 5	-952 ± 29	-860 ± 33	-16 ± 31	-10 ± 30	-7697 ± 663	-7374 ± 761
7	597 ± 10	721 ± 17	303 ± 18	462 ± 10	-826 ± 48	-706 ± 45	-297 ± 10	-82 ± 26	-5729 ± 475	-5447 ± 418
8	481 ± 11	583 ± 16	336 ± 14	429 ± 9	-1188 ± 29	-1108 ± 34	-498 ± 12	-201 ± 35	-9912 ± 583	-9792 ± 619
9	780 ± 12	839 ± 15	337 ± 12	421 ± 5	-868 ± 63	-758 ± 52	-166 ± 31	-175 ± 28	-4939 ± 715	-4840 ± 747
10	523 ± 11	616 ± 17	257 ± 22	473 ± 26	-1225 ± 41	-1087 ± 47	-623 ± 12	-239 ± 37	-10834 ± 711	-10584 ± 660

Table 1: Performance of the two base policies

The OPI algorithms mentioned above are all based on offline procedures that are theoretically guaranteed to return a policy better than the base policy. However, the ideal conditions on the number of sampled trajectories, lengths of trajectories, leaf initialization to base policy values, perfect state transitions of actions might not hold in practice. In that case, the online implementations can result in a policy that is worse than the given base policy.

OPI Baselines

Our baselines are variants of the policy rollout algorithm with a Q-value adjustment heuristic to deal with policy degradation to some extent. Let s_0 be the current state for which an action decision is to be made, A_{s_0} be the set of actions expanded at s_0, L be the lookahead horizon and π be the base policy. The base policy can be non-stationary. In order to keep the notation simple, we describe the baselines and heuristic with a deterministic, stationary policy $\pi = \{\mu_0, \mu_1, \ldots, \mu_{L-1}\}$, where $\mu_k = \mu$ for $k = 0, \ldots, L-1$ and $\mu : S \to A$. We use $\pi(s)$ instead of $\mu(s)$ for the base policy action at state s.

Baseline 1: MC Policy Rollout

Our first baseline is a version of the Monte Carlo (MC) policy rollout algorithm (Bertsekas and Tsitsiklis (1996), Tesauro and Galperin (1997)). The Q-values of actions at s_0 are estimated from sampled trajectories without building a search tree. The Q-value estimate of an action is the average of the values of multiple base-policy trajectories starting with that action. If N is the number of trajectories, then

$$\hat{Q}^{\pi}_{L}(s_0, a) \;=\; \frac{1}{N} \sum_{i=1}^{N} \left(R(s_0, a) \;+\; \sum_{k=1}^{L-1} R(s^i_k, \pi(s^i_k)) \right),$$

where s^i_k is the k^{th} subsequent state of trajectory i and $s^i_1 \in \{s' \in S : P_{s_0 s'}(a) > 0\}$ and $s^i_{k+1} \in \{s' \in S : P_{s^i_k s^i_{k+1}}(\pi(s^i_k)) > 0\}$ for $0 < k < L$ and $0 < i < L$. We note that $\hat{Q}^{\pi}_{L}(s_0, a)$ is an unbiased estimate of $Q^{\pi}_{L}(s_0, a)$.

Baseline 2: DAG Policy Rollout

The second baseline builds a search DAG (Directed Acyclic Graph) to make better use of samples compared to the first baseline. The DAG will have s_0 at its root followed by a sequence of state-node layers $(S_1, S_2, \ldots, S_L)$. While computing the backup values of states in the DAG, every state in layer S_i is assumed to be connected to all the states in layer S_{i+1}. This baseline is also a version of the policy rollout algorithm, so off-policy actions are allowed only at the root and only the base policy action is allowed at all other internal nodes of the DAG.

DAG Construction. We expand the base policy action $\pi(s_0)$ and one or more off-policy actions at the root node s_0 and generate b_0 successors for each action using the true state-transition probabilities. The b_0 successors generated for an action can have repeated states. We put together all the generated successors of all the actions and eliminate duplicates to form the subsequent state-node layer S_1, i.e.,

$$S_1 \;=\; \bigcup_{a \in A_{s_0}} GSucc(s_0, a, b_0),$$

where $GSucc(s_0, a, b_0)$ is the set of distinct successors of action a taken b_0 times at state s_0 such that $GSucc(s_0, a, b_0) \subseteq \{s' \in S : P_{s_0 s'}(a) > 0\}$ and $|GSucc(s_0, a, b_0)| \leq b_0$.

We then expand the base policy action for all the states in layer S_1 and generate b successors for each state using the true state-transition probabilities. Again, the b successors generated for a state can have repeated states. We put together all the generated successors of all the states and eliminate duplicates to form the subsequent state-node layer S_2. We follow the same process to create all subsequent state-node layers $S_3, \ldots, S_L$. Formally,

$$S_{k+1} \;=\; \bigcup_{s \in S_k} GSucc(s, \pi(s), b)$$

for $k = 1 \ldots, L-1$, where $GSucc(s, \pi(s), b)$ is the set of distinct successors of the base policy action $\pi(s)$ taken b times at state s such that $GSucc(s, \pi(s), b) \subseteq \{s' \in S : P_{ss'}(\pi(s)) > 0\}$ and $|GSucc(s, \pi(s), b)| \leq b$.

Q-Value Computation. We set the values of leaf nodes to zero. We estimate the value of each state node in layers S_1 through S_{L-1} as the immediate reward of the base policy action for the state plus a weighted average of the values of all the state nodes in the following layer. Let $\hat{V}_{L-k}(s)$ denote the value estimate of state s in layer S_k in the DAG. Then $\hat{V}_0(s) = 0$ for all $s \in S_L$ and

$$\hat{V}_{L-k}(s) = R(s, \pi(s)) + \sum_{s' \in S_{k+1}} \hat{P}_{k,ss'}(\pi(s)) \cdot \hat{V}_{L-k-1}(s'),$$

where

$$\hat{P}_{k,ss'}(a) = \frac{P_{ss'}(a)}{\sum_{s' \in S_{k+1}} P_{ss'}(a)}.$$

The L steps-to-go Q-value of an action at the root node s_0 is the immediate reward of the action plus a weighted average of the values of its successors in layer S_1, i.e.,

$$\hat{Q}^{\pi}_{L}(s_0, a) \;=\; R(s_0, a) \;+\; \sum_{s' \in S_1} \hat{P}_{0,s_0 s'}(a) \cdot \hat{V}_{L-1}(s').$$

We note that $\hat{Q}^{\pi}_{L}(s_0, a)$ computed using normalized weights can be a biased estimate of $Q^{\pi}_{L}(s_0, a)$.

The Q-value Adjustment Heuristic

The purpose of the Q-value adjustment heuristic is to make it harder for off-policy actions to qualify as better actions in the current state. The OPI algorithm will then be conservative in switching to off-policy actions. We achieve this by computing an error margin for the Q-value estimate of each action at s_0. We then increase (decrease) the Q-value estimate of the base policy action (off-policy actions) by their respective error margins.

Let $\epsilon^{\pi}(s_0, a)$ denote the error margin for action a. We have 2 formulas for the error margins and hence 2 different heuristics, namely, the *C-Heuristic* and the *PC-Heuristic*.

- **C-Heuristic.** The error margin is a fraction of the absolute Q-value estimate of the action, i.e.,

  $$\epsilon^{\pi}(s_0, a) \;=\; C \cdot |\hat{Q}^{\pi}_{L}(s_0, a)|,$$

  where $C \in [0, 1]$ is a parameter.

- **PC-Heuristic.** The error margin has an additional state-action dependent factor equal to the total probability of next-states not covered while generating successors and hence not used in estimating the Q-value, i.e.,

  $$\epsilon^{\pi}(s_0, a) \;=\; D(s_0, a) \cdot C \cdot |\hat{Q}^{\pi}_{L}(s_0, a)|,$$

  where $C \in [0, 1]$ and $D(s_0, a) = 1 - \sum_{s' \in S_1} P_{s_0 s'}(a)$.

The adjusted Q-value estimate of action a at s_0 is then

$$\tilde{Q}^{\pi}_{L}(s_0, a) \;=\; \begin{cases} \hat{Q}^{\pi}_{L}(s_0, a) \;+\; \epsilon^{\pi}(s_0, a), & \text{if } a = \pi(s_0) \\ \hat{Q}^{\pi}_{L}(s_0, a) \;-\; \epsilon^{\pi}(s_0, a), & \text{if } a \neq \pi(s_0) \end{cases}$$

Both MC policy rollout and DAG policy rollout return an action $\hat{a}$ that maximizes the adjusted Q-value estimates for s_0, i.e.,

$$\hat{a} \;\in\; \arg\max_{a \in A_s} \tilde{Q}^{\pi}_{L}(s_0, a).$$

Benchmarks

Our initial OPI benchmark set consists of the following 5 domains from the past International Probabilistic Planning Competitions (IPPC): (1) Sysadmin, (2) Game of Life, (3) Tamarisk, (4) Skill Teaching and (5) Wildfire. Each domain comes with a standard set of 10 problems of varying sizes and difficulty levels. Further details on the IPPC can be found at (http://www.icaps-conference.org/index.php/Main/Competitions). Both the domains and problems are described using the Relational Dynamic Influence Diagram Language (RDDL) (Sanner (2010)).

Sysadmin. This domain is about keeping as many computers up as possible in a computer network. The state of a computer is affected by the states of computers connected to it besides an external random factor. Actions are to reboot computers to bring them up. The immediate reward of a state-action pair is the number of computers running minus the action cost. The state space is factored with binary state variables for the computers in the network. The size of the state space ranges from 2^{10} to 2^{50} for the 10 problems.

Game of Life. This is a grid based domain with cells in the grid either alive or dead. The goal is to have as many cells alive as possible. The state of a cell is affected by the states of the cells around it besides an external random factor. Actions are to set cells to bring them alive. The immediate reward of a state-action pair is the number of cells alive minus the action cost. The size of the state space ranges from 2^9 to 2^{30} for the 10 problems.

Tamarisk. This domain is about eradicating an invasive plant species called tamarisk to promote a native plant species in a given region. The region is divided into reaches each consisting of a certain number of slots. Tamarisk can spread from a reach to an adjacent (downstream) reach. Actions are to eradicate tamarisk or restore native species in reaches. The immediate reward of a state-action pair is a penalty for the number of slots invaded by and vulnerable to tamarisk plus the action cost. The state space is factored with two state variables per slot indicating the presence of tamarisk or native species. The size of the state space can range from 2^{16} to 2^{48} for the 10 problems.

Skill Teaching. This domain is about teaching a student a given set of skills via hints and questions. There are prerequisites for some of the skills. A student can attain medium

	C0			C0.1			PC0.1			C0.2			PC0.2			C0.3			PC0.3		
Bad Base Policy	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L
Sysadmin	8	1	1	2	8	0	3	7	0	0	10	0	2	8	0	0	10	0	1	9	0
Game of life	10	0	0	8	2	0	9	1	0	5	5	0	7	3	0	2	8	0	5	5	0
Tamarisk	0	2	8	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0
Skill Teaching	5	5	0	3	7	0	5	5	0	3	7	0	6	4	0	3	7	0	6	4	0
Wildfire	0	6	4	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0
	C0			**C0.1**			**PC0.1**			**C0.2**			**PC0.2**			**C0.3**			**PC0.3**		
Good Base Policy	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L
Sysadmin	0	2	8	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0
Game of life	3	7	0	2	8	0	4	6	0	0	10	0	2	8	0	0	10	0	2	8	0
Tamarisk	0	0	10	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0
Skill Teaching	0	10	0	1	9	0	0	10	0	0	10	0	1	9	0	0	10	0	0	10	0
Wildfire	0	9	1	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0

Table 2: WTL Scores for DAG Policy Rollout

	C0			C0.1			PC0.1			C0.2			PC0.2			C0.3			PC0.3		
Bad Base Policy	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L
Sysadmin	7	3	0	2	8	0	3	7	0	0	10	0	1	9	0	0	10	0	1	9	0
Game of life	8	2	0	6	4	0	8	2	0	2	8	0	6	4	0	1	9	0	6	4	0
Tamarisk	0	0	10	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0
Skill Teaching	5	5	0	6	4	0	5	5	0	3	7	0	5	5	0	3	7	0	5	5	0
Wildfire	0	3	7	0	10	0	0	9	1	0	10	0	0	10	0	0	10	0	0	10	0
	C0			**C0.1**			**PC0.1**			**C0.2**			**PC0.2**			**C0.3**			**PC0.3**		
Good Base Policy	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L
Sysadmin	0	0	10	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0
Game of life	1	6	3	0	10	0	2	8	0	0	10	0	1	9	0	0	10	0	1	9	0
Tamarisk	0	0	10	0	9	1	0	9	1	0	10	0	0	10	0	0	10	0	0	10	0
Skill Teaching	0	9	1	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	1	9	0
Wildfire	0	4	6	0	10	0	0	9	1	0	10	0	0	10	0	0	10	0	0	10	0

Table 3: WTL Scores for MC Policy Rollout

proficiency in a skill if they know all the prerequisites. A student can attain high proficiency if they have medium proficiency in that skill and answer questions correctly. Answering a question wrong can decrease the proficiency level. Actions are to give hints or ask questions in skills. The immediate reward is a bonus for high proficiency and a penalty for medium proficiency in skills. The state space is factored with six state variables per skill. The size of the state space can range from 2^{12} to 2^{48} for the 10 problems.

Wildfire. This is about controlling the spread of fire in a region modeled as a grid. A few cells are marked as targets that need to be protected from fire. A grid cell with fuel in it is more likely to burn if many of its neighbors are burning and a burning cell continues to burn until the fire is extinguished. Actions are to cut out fuel from cells or put out fire in cells. The immediate reward for a state-action pair is a penalty for burned out or burning cells plus an action cost. The penalty is high for the target cells. The state space is factored with two state variables per cell indicating the presence of fuel and fire. The size of the state space can range from 2^{18} to 2^{72} for the 10 problems.

Base Policies

We have two base policies for each problem making a total of 100 base policies for the 50 problems. For each problem, one of the base policies is of high quality, while the other is of relatively poor quality. Table 1 shows the performance of the two base policies. The numbers denote the average finite horizon sum of rewards for 100 evaluation runs with 95% confidence intervals. For the domains tamarisk and wildfire, the two policies are of roughly the same quality.

All our base policies are Neural Networks (NNs) taking a factored state as input and returning a probability distribution over actions as output. The action with the highest probability is taken as the base policy action for the state. We have used two distinct NN architectures for each domain - a linear architecture and one with 3 hidden layers and sparse connections defined by state-variable transitions. The linear architecture performs best for sysadmin and skill teaching, while the non-linear architecture performs best for game of life, tamarisk and wildfire. We have adopted the architecture, dataset and training method for the NN base policies from (Issakkimuthu, Fern, and Tadepalli (2018)).

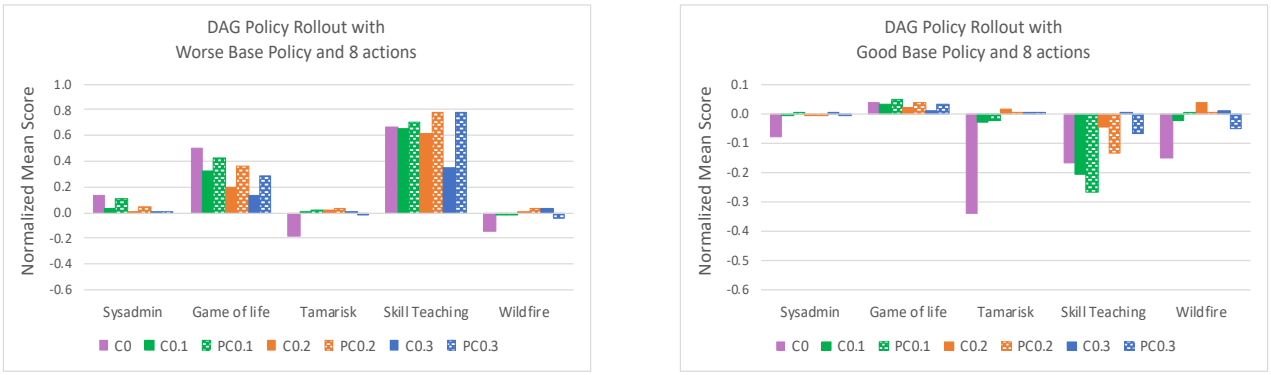

Figure 1: Normalized Mean Scores for DAG Policy Rollout

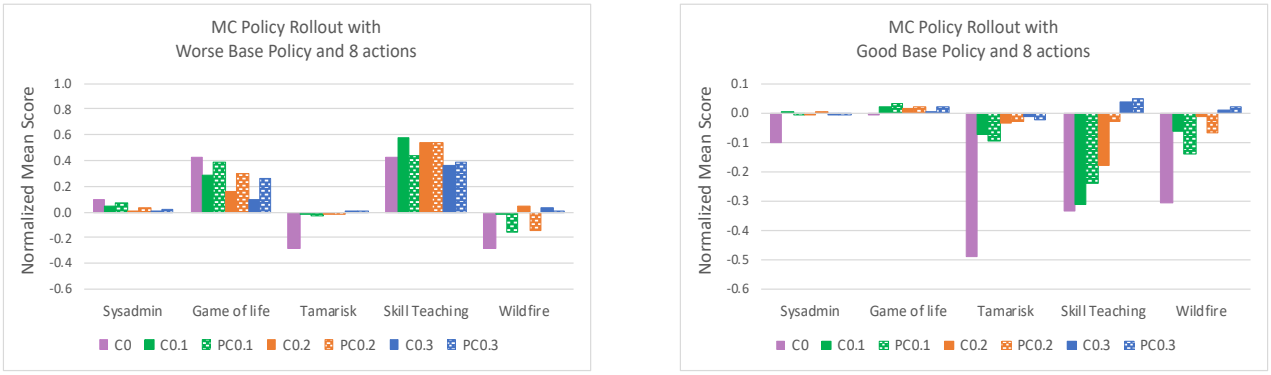

Figure 2: Normalized Mean Scores for MC Policy Rollout

Evaluation Metrics

We estimate the expected finite horizon sum of rewards of the base policy and the OPI policy by taking the average over 100 evaluation runs. In order to assess OPI policies for performance degradation, we propose the following 3 evaluation metrics.

1. **Win-Tie-Loss (WTL) Count.** For each domain we compute a triple of integers that add up to 10 (the number of problems in the domain). The components W, T and L stand for the number of wins, ties and losses achieved by OPI against the base policies. The win, tie or loss outcome for a problem is defined in terms of the standard 95% Confidence Intervals (CIs) around the mean performance of the base policy and the OPI policy. If the CI of OPI is totally above the CI of the base policy then the outcome is a win for OPI. If the CIs overlap then the outcome is a tie. Otherwise it is a loss. Wins are hard to achieve because the condition for wins is very strict. The WTL count is a simple and natural measure of the performance of OPI algorithms. OPI algorithms with losses can be considered unreliable for the domain. OPI algorithms with wider CIs will have many ties and no losses, which might look fine under the WTL criterion. This drawback can be addressed by using the following metric along with WTL counts.

2. **Normalized Mean Score (NMS).** For each domain we compute a single real number that indicates OPI performance. The NM score of a domain is the average of the NM scores of the 10 problems. The NM score for problem k is defined in terms of the mean performance of the base policy $\nu(k)$ and the mean performance of the OPI policy $\nu'(k)$ for that problem, i.e.,

$$NMS(k) = \frac{\nu'(k) - \nu(k)}{|\nu(k)|}.$$

A positive normalized mean score indicates better average improvement, a negative score indicates worse average degradation and a zero score indicates equivalent average performance of OPI for the domain.

3. **Normalized Percentile Score (NPS).** In addition to NMS, another measure like the average of the bottom $\alpha\%$ of the 100 evaluation runs might be useful to ensure that OPI does not crash badly when the base policy does not. The NP score of a domain is the average of the NP scores of the 10 problems. The NP score for problem k is defined in terms of the average of the bottom $\alpha\%$ of evaluation runs of the base policy $\zeta_\alpha(k)$ and the average of the bottom $\alpha\%$ of evaluation runs of the OPI policy $\zeta'_\alpha(k)$ for

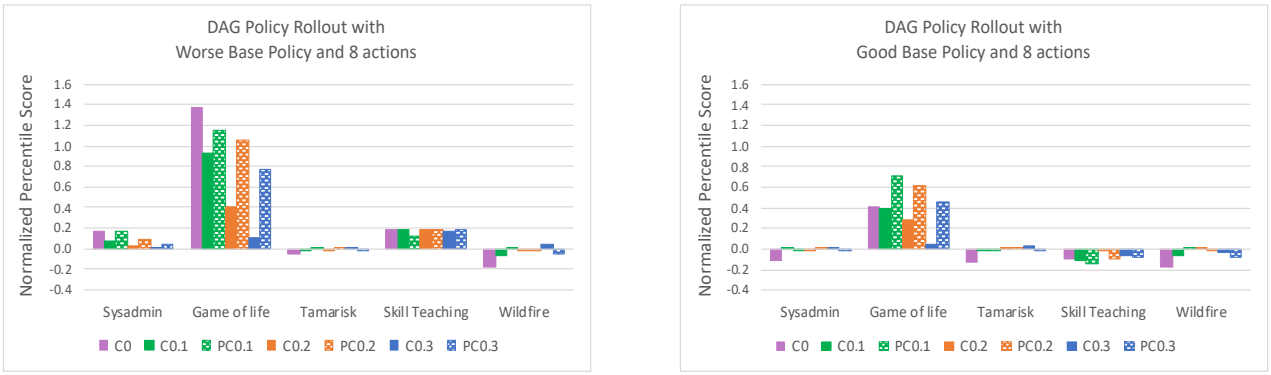

Figure 3: Normalized Percentile Scores for DAG Policy Rollout

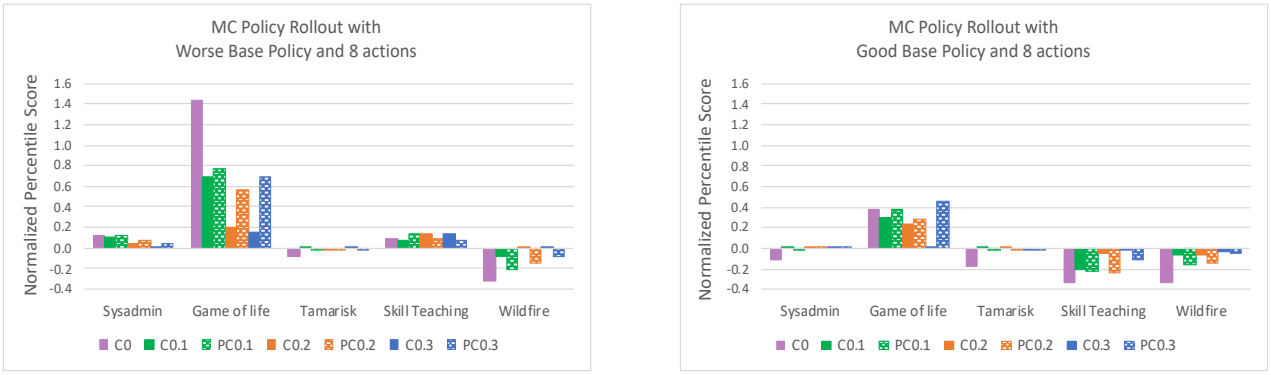

Figure 4: Normalized Percentile Scores for MC Policy Rollout

that problem, i.e.,

$$NPS_\alpha(k) = \frac{\zeta'_\alpha(k) - \zeta_\alpha(k)}{|\zeta_\alpha(k)|}.$$

Positive, negative and zero NP scores indicate improved, worse and equivalent low end average of OPI. NPS is analogous to the notion of Conditional Value at Risk (CVaR) in statistics.

Experiments

Here we provide an example of our evaluation procedure for the DAG and MC OPI baselines. The primary goal is to demonstrate the evaluation methodology and show that the baselines are strong for both the cases when transition probability information is not available (MC with C-Heuristic) and when the transition probability is available (DAG and MC with PC-Heuristic).

Setup. The lookahead depth is 4 for both DAG and MC policy rollout. Leaf nodes are set to zero. The number of root actions is 8 or the number of actions applicable whichever is less. These are the top 8 actions according to the base policy probabilities and therefore include the base policy action. The number of successors generated for a state-action pair in the case of DAG policy rollout is 3, i.e., $b_0 = b = 3$. In order to compare DAG and MC policy rollout results, we first run DAG policy rollout for the current state and then run MC policy rollout for the same amount of time. The number of rollout trajectories in MC policy rollout can therefore vary from state to state.

The parameter C of the Q-value adjustment heuristic takes values from the set $\{0,\ 0.1,\ 0.2,\ 0.3\}$. For MC policy rollout with the PC-Heuristic, we record the successors of the root state s_0 for every action to compute $D(s_0, a)$ using the true state-transition probabilities. In our experiments, the planning horizon is 40 and actions are limited to those with at most one action bit set. Our experiments were run on a HPC cluster with the RDDLSim library (https://github.com/ssanner/rddlsim) for evaluation.

Notation. In all the tables and charts, the labels Cx and PCx have been used to denote OPI policies with different Q-value adjustment heuristics. Cx stands for the C-Heuristic with parameter C set to value x. PCx stands for the PC-Heuristic with parameter C set to value x. C0 corresponds to the OPI policy without Q-value adjustment as it is equivalent to the first heuristic with parameter C set to 0.

WTL Results. Tables 2 and 3 show the WTL scores for DAG and MC policy rollout for the two base policies. The first major observation is that both DAG and MC rollout without Q-value adjustment (C0) have many losses across the domains. This is particularly the case for the higher-quality base policy, which is frequently degraded.

Next, we consider the influence of C on performance. We see, as expected, that as C increases the number of losses decreases and the wins tend to increase. In particular, for $C > 0$ there are only a very small number of losses and no losses for $C \geq 0.2$. At the same time we see that in 3 of the domains there are a significant number of wins for the lower-performing base policy even for the largest value of $C = 0.3$. There are zero wins for the high-performing base policy, which is due to the strict conditions under which we judge a win (non-overlapping confidence intervals). We will see later, however, that even in these cases there is improvement when considering the expected values.

Next, comparing the results for Cx versus PCx, we see that including the probability of states "not covered" typically results in more wins without a substantial increase (or any increase) in losses. This, shows that the PCx heuristic is able to effectively modulate the Q-value adjustment on a per state-action basis to improve OPI performance. This shows that when transition probabilities or good estimates are available, PCx can be an effective approach to maintaining safety while improving performance.

Finally, we see that the DAG policy rollout generally performs better that the MC approach in terms of total number of wins while not increasing the number of losses. This shows that there can be value in reuse of MC samples in a DAG structure and also exploiting the transition model within the DAG model compared to a pure MC approach. The design space of DAG structures used for value estimation is large and it is reasonable to expect that optimization of the structure could result in further improvement.

Overall, the results indicate that the DAG and MC approaches offer a strong parameterized space of baselines for safe OPI that span different WTL trade-offs for comparison to other approaches. In particular, MC offers a baseline that requires no information about transition probabilities, while DAG is able to exploit transition information when available. The results also indicate that with respect to number of wins, there is significant room to improve over the baselines for the higher-quality base policy.

Normalized Mean Scores. The bar charts in Figures 1 and 2 show the NMS for DAG and MC policy rollout for both the base policies. We first see that for both DAG and MC policy rollout the scores are positive in sysadmin, game of life and skill teaching for the bad base policy and somewhat positive in game of life for the good base policy. Rather, the scores are negative or close to zero everywhere else. The scores are particularly negative for the case of no Q-value adjustment (C0). Increasing the value of C largely mitigates the degradation and including the "missing probability" into the adjustment tends to be better Finally, again, DAG policy rollout performs slightly better than MC policy rollout, showing the baselines are able to leverage the availability of transition probabilities for better performance.

Normalized Percentile Scores. The bar charts in Figures 3 and 4 show the NPS for the bottom 5% evaluation runs for DAG and MC policy rollout for both the base policies. These results are qualitatively similar to those for NMS. However, we do see that the NPS values tend to be higher than the corresponding NMS values. This indicates that the baseline OPI methods are generating more improvement or less degradation for the lower end of the performance profile (NPS) compared to the mean performance. In other words, these OPI baselines are suggesting they are particularly effective at improving worst case performance.

Summary

In this work, we have drawn attention to the important practical issue of policy degradation in OPI. We have proposed benchmarks and evaluation metrics for OPI with the goal of improving our understanding of the empirical performance of OPI algorithms. We have also presented OPI baselines with a heuristic to deal with policy degradation. The baselines form a class of methods that can use transition probabilities if available or only utilize samples. The parameterized baselines are demonstrated to span the trade-off space of OPI performance, making them useful points for future comparison. Further, the baselines demonstrate benefit from using transition probability information, making them useful for comparing to evaluation settings with and without that information. The DAG-based baseline makes better use of samples by constructing a search DAG instead of discarding sampled trajectories. It will be interesting to consider future extensions that continue to search off-policy actions within the DAG rather than only at the root. Overall, we hope that this work helps set the stage for more work on safe OPI algorithms backed by solid evaluations and comparisons to strong baselines.

References

Bertsekas, D. P.; and Castanon, D. A. 1999. Rollout Algorithms for Stochastic Scheduling Problems. *Journal of Heuristics* 5(1): 89–108.

Bertsekas, D. P.; and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in games* 4(1): 1–43.

Cazenave, T. 2009. Nested Monte-Carlo Search. In *Twenty-First International Joint Conference on Artificial Intelligence*.

Chang, H. S.; Givan, R.; and Chong, E. K. P. 2004. Parallel Rollout for Online Solution of Partially Observable Markov Decision Processes. *Discrete Event Dynamic Systems* 14(3): 309–341. ISSN 0924-6703. doi:10.1023/B:DISC.0000028199.78776.c4. URL https://doi.org/10.1023/B:DISC.0000028199.78776.c4.

Issakkimuthu, M.; Fern, A.; and Tadepalli, P. 2018. Training Deep Reactive Policies for Probabilistic Planning Problems. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*.

Issakkimuthu, M.; Fern, A.; and Tadepalli, P. 2020. The Choice Function Framework for Online Policy Improvement. AAAI Press, Proceedings of the Thirty Fourth AAAI Conference on Artificial Intelligence (AAAI-20).

Kearns, M.; Mansour, Y.; and Ng, A. Y. 2002. A Sparse Sampling Algorithm for near-optimal Planning in Large Markov Decision Processes. *Machine Learning* 49(2-3): 193–208.

Nguyen, T.-H. D.; Silander, T.; Lee, W.-S.; and Leong, T.-Y. 2014. Bootstrapping Simulation-based Algorithms with a Suboptimal Policy. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.

Pinto, J.; and Fern, A. 2017. Learning Partial Policies to Speedup MDP Tree Search via Reduction to IID Learning. *The Journal of Machine Learning Research* 18(1): 2179–2213.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1st edition. ISBN 0471619779.

Sanner, S. 2010. Relational Dynamic Influence Diagram Language (RDDL) : Language Description. URL http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; and Hassabis, D. 2018. A General Reinforcement Learning Algorithm that masters Chess, Shogi, and Go through self-play. *Science* 362(6419): 1140–1144. ISSN 0036-8075. doi:10.1126/science.aar6404. URL https://science.sciencemag.org/content/362/6419/1140.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the Game of Go without Human Knowledge. *Nature* 550(7676): 354.

Tesauro, G.; and Galperin, G. R. 1997. Online Policy Improvement using Monte-Carlo Search. Advances in Neural Information Processing Systems (NIPS-1997).

Walsh, T. J.; Goschin, S.; and Littman, M. L. 2010. Integrating Sample-based Planning and Model-based Reinforcement Learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, 612–617. AAAI Press. URL http://dl.acm.org/citation.cfm?id=2898607.2898706.

Designing an Adaptable Benchmark and Competition Simulation for Integrated Planning and Execution

Liudvikas Nemiro[1], **Gerard Canal**[2], **Oscar Lima**[3], **Michael Cashmore**[1], **and Mark Roberts**[4]

[1] University of Strathclyde, Glasgow, Scotland, UK
[2] Department of Informatics, King's College London, UK
[3] DFKI Niedersachsen Lab, Osnabrück, Germany
[4] The U.S. Naval Research Laboratory

Abstract

Effectively using a planning system as the executive of an agent acting in real time poses a variety of challenges in integrating planning and execution. Many integrated systems have been developed with a focus on particular challenges, and it has been typically difficult to test, benchmark, and compare these systems. To do so requires a benchmark that has transparent and well-defined rules, and can be adapted to exhibit the problem characteristics of interest. In this paper, we propose a new benchmark simulation for integrated planning and execution, designed to be accessible and adaptive. We describe the simple core scenario of the simulation and how it can be configured to present more challenging scenarios. We describe our plans for the development of the simulation as a competition, benchmarking, and teaching tool, and encourage the community to contribute to its design.

1 Introduction

Planning systems are a natural approach to the deliberative control of autonomous systems. Different integrations of planning systems into the executives of autonomous systems results in a diverse set of systems (Muscettola et al. 2002; McGann et al. 2008; Niemueller, Hofmann, and Lakemeyer 2019). The International Planning Competition (IPC) tackles the comparison of planning systems through its multiple tracks. However, comparing and benchmarking planning executives poses an interesting challenge. Planning system and execution capabilities are often developed together, making it hard to decouple the reasoning from the execution environment.

In this paper, we introduce the initial design of the CRAFTBOTS simulation for benchmarking and comparing planning and execution systems. CRAFTBOTS simulates a logistics scenario for one or more actors. The simulation consists of a minimal core scenario that can be configured with a variety of additional modules to introduce more sophisticated problem characteristics.

Our goal is to make the simulation available for research, competition, and teaching. Thus, our main considerations in the design and implementation of CRAFTBOTS is to be

1. *light-weight and portable*. The simulation is written entirely in Python3 with no external dependencies. This will facilitate ease of use for teaching, lower the barrier of entry for planning or executive systems, and provide a starting point for additional enhancements.
2. *accessible* - the API is as simple as possible, exposing a set of Python3 methods. Interfaces to adapt to other platforms are planned, such as a ROS actionlib server (Quigley et al. 2009), OpenAI Gym environment (Brockman et al. 2016), and High-Level Robot API[1]. This should allow existing systems for planning and execution, such as the CLIPS agent (Niemueller, Hofmann, and Lakemeyer 2019) and ROSPlan (Cashmore et al. 2015) to be directly applied to this scenario.
3. *adaptable to different execution requirements*, so that users can configure the simulation to test approaches for handling non-determinism, temporal constraints, oversubscription, or other combinations of problem characteristics.

The simulation is still under active development, but the work in progress is available as open-source software[2] with open issues and discussion pages.

One motivation of CRAFTBOTS is to provide a foundation for a planning and execution competition. The idea of the competition is not to organise an event with one or more tracks, but instead to host an online competition open to submissions all year round, similar to the Sparkle challenge[3]. The competition leader-board can be filtered by problem characteristics, based on the simulation configuration. Results and insights from the competition can then be presented each year at ICAPS. As the timestamped events of a completed simulation can be efficiently saved, it would be possible to present and host complete replays as well as scores.

The software also has great potential as a teaching resource, enabling students to participate in the open competition. In addition to documentation of the code, we aim to produce a series of tutorials and exercises on intelligent control, supported by the simulation. This would allow students to submit their coursework to the competition to broaden participation, but also allows students to directly compare their submission against the state-of-the-art.

The aim of this paper is to foster discussion at this early stage of the project. We are looking to gather insight into

[1] https://github.com/DFKI-NI/high_level_robot_api
[2] https://github.com/strathclyde-artificial-intelligence/craft-bots
[3] https://ada.liacs.nl/events/sparkle-planning-19/

what challenges and problem properties should be embodied by the simulation, to foster engagement in future competitions, and to guide the development of the simulation and teaching materials into a product that is ultimately useful to the community.

1.1 Other simulators

The CRAFTBOTS scenario takes inspiration from the Robocup Logistics League (RCLL). RCLL focuses on in-factory logistics applications for teams of mobile robots[4]. A simulation of the RCLL scenario has been used for competitions with integrated planning and scheduling (Niemueller, Lakemeyer, and Ferrein 2015; Niemueller et al. 2016) beginning in 2017[5]. The simulation is built using Gazebo, a LUA-based behaviour engine, and the Fawkes Robot Software Framework adapted from the publicly released software stack of the Carologistics RoboCup team (Niemueller, Reuter, and Ferrein 2015). The simulation is also open source and has included a built-in executive based on ENTERPRISE: PIKE (Levine and Williams 2014), and ROS interface based on ROSPlan (Cashmore et al. 2015). The simulation poses a realistic challenge in the intelligent control of a mobile robot team, in planning and scheduling, plan execution, and interfacing with the robot's sensors and behaviours. In contrast, CRAFTBOTS focuses on providing challenges in the planning and execution, while simplifying the interface and underlying architecture required to run the simulation. This presents a more accessible alternative to RCLL that can be used to develop, test, and benchmark systems, while the underlying challenges are similar enough that those systems could be subsequently ported to control a RCLL robot team.

The RoboCup Rescue Agent Simulator, or ROBORESCUE, models a situation immediately following a natural disaster (Sheh, Schwertfeger, and Visser 2016). It is the basis of the RoboCup Rescue Simulation League (Akin et al. 2012), an international competition for collaborative AI agents since 2000. There are one physical and two simulated competitions held annually[6]. The Virtual Robot League is a detailed, high fidelity physics simulator within the confines of one city block and faces many of the same kinds of challenges we mentioned for RCLL. The Simulation League focuses on environments the size of a few city blocks. The simulator for this is written in Java and has a long history of code based on past competitors; in fact the simulator has some very sophisticated simulation capabilities and includes some baseline code to facilitate programming new agents. In our studies (Roberts et al. 2021), we were able to demonstrate how to connect a cognitive system that performed a centralized dispatcher function. This was a challenging task that required understanding a complex suite of interacting software components. While we enjoyed the flexibility and richness of the simulation environment provided by the server, we found that it took considerable time to start programming agents because so much time was invested in understanding the underlying architecture relative to the time invested in the aspects of planning and execution that drew us to the simulator.

A middle ground between a realistic 3D environment and something that is accessible is found in the CrazySwarm (Preiss et al. 2017). This is a lightweight Python control environment for teams of physical or virtual micro-quadrotor systems. Each vehicle can accept commands to takeoff, land, and move to specific locations. The "simulator" in this environment is a simple matplotlib viewer that can augment physical vehicles or simulate a virtual-only environment, allowing client code to easily switch between controlling physical or virtual robots. In our studies we found this environment easy to use and were able to quickly mock up scenarios, in Python, by calling a PDDL planner and linking it to a goal reasoning system (Roberts et al. 2021), though we had to write a small amount of executive and interfacing code. The downside of the environment lies in its simplicity; programming sophisticated behaviors, or adding new ones, requires writing controller code for the specific quadrotor platform. Adding new vehicles, or new behaviors, would require considerable programming, which often does not align well with running suites of experiments for testing how the planning and executive impact performance.

The microRTS game (Ontañón 2013) is a simple grid-world environment that has the kind of ease of use and flexibility for scenario generation we anticipate for CRAFTBOTS. Inspired by the Starcraft game, a Real-time Strategy (RTS) game, opponents must gather resources, construct buildings and forces, and protect or invade other territories. The microRTS has been used to demonstrate integrated execution of Hierarchical plans (Kantharaju, Ontañón, and Geib 2018) and has been featured in several competitions[7]. CRAFTBOTS has similar resource constraints but features more of a long-term cooperative situation than the competitive style of RTS games. Also, CRAFTBOTS will feature mechanisms to increase the difficulty of scenarios in several ways (e.g., online goal arrival or deadlines).

A suite of simulators have been used in learning contexts, the most notable and recent of which is the gym environment (Brockman et al. 2016). Gym is a standard interface for interacting with simulation environments that allows rapid learning. Although there are many environments we could discuss, we focus on a few that are the most appropriate for integrated planning and acting. The Malmo simulator (Johnson et al. 2016) provides a python interface to the 3D sandbox game of Minecraft and has been used in several competitions, the most recent of was MineRL[8], which provided a Gym interface for Malmo and challenged competitors to learn from recorded human players. Another Gym environment that shares several properties with our ideal system mentioned in the introduction is Gym-Minigrid (Chevalier-Boisvert, Willems, and Pal 2018), which is is a gridworld environment where an agent takes discrete cardinal actions to move between rooms to collect items or visit target cells. Finally, PDDLGym[9] is a suite that converts STRIPS PDDL files to gym environments to facilitate using a simulator for executing plans. The Gym

[4] https://ll.robocup.org/

[5] http://www.robocup-logistics.org/sim-comp

[6] http://wiki.robocup.org/Rescue_Simulation_League

[7] https://sites.google.com/site/micrortsaicompetition/home

[8] https://minerl.io/

[9] https://github.com/tomsilver/pddlgym

environment is designed to advance the study of Reinforcement Learning and is thus focused on episodic interaction between an agent and its environment. Integrating planning approach into this framework is possible, but it can require considerable effort to craft a set of scenarios to study one aspect of integrated planning and execution. Further, one has to implement (or learn) controllers for each of the actions that one would want agents to perform in a gym environment. In contrast, CRAFTBOTS will provide a set of standard controllers for actions and will also provide a suite of benchmark problems for researchers to test against.

2 Simulation Description

In this section we describe the core scenario of the simulation. Then we describe the additional modules that can be configured and characterise the problem properties that they introduce.

2.1 Core Scenario Description

The proposed scenario consists of agents that can move around the environment to collect resources of different types. The environment is represented as a network of nodes, and is illustrated in Figure 1. Agents in the world move around this network to collect resources (coloured triangles) from mines (coloured circles) and build structures (coloured, stacked polygons) at specified locations. We provide more details about these components and their interactions below.

Agents build structures to complete task goals. A task goal specifies the set of resources required to build a structure, and the node at which it should be built. Once a structure is built at that node with the required materials, then the agent increases its score an amount proportional to the number of resources. The agent should try to maximise its score over a finite horizon.

Resources can be gathered at nodes which contain a mine of that resource type. Each agent can carry only a limited amount of resources at one time. Task goals are to use these resources to build structures in specified locations. A task goal specifies the location and required resources for a structure. Achieving these goals scores points for the team.

A scenario is generated from a configuration file and random seed, and the simulator responds to commands through the Python3 API. The command interface can be run is threaded (the simulation will continue running and process commands as they arrive) meaning that planning and other reasoning must be made in real-time.

The core scenario contains a set of deterministic actions with fixed duration, described below. Additional modules introduce non-deterministic action durations and outcomes, properties unique to each resource type, and structures that can be optionally built to provide beneficial effects.

Actions Agents are each able to perform actions one at a time. If the simulation is configured to contain more than one agent, agents can perform actions in parallel. Unless otherwise specified, the actions have a very short non-zero duration.

- MOVE: the agent moves between two nodes of the graph provided that the nodes are connected. The action has a duration proportional to the length of the connection.
- DIG: When the agent is at a node that contains a mine, the agent produces one resource of the mine's resource type. The resource appears on the ground at that node. The action has a constant duration.
- PICK-UP: The agent collects a resource on the ground in the same node and adds it to the agent's inventory.
- DROP: The agent removes one resource from its inventory and adds it to the ground at the current node.
- CREATE-SITE: Creates a new construction site, corresponding to a specific set of required materials.
- DEPOSIT: The agent removes one resource from its inventory and adds it to a site at the current node. Resources cannot be recovered once deposited into a site.
- CONSTRUCT: Progresses the completion of a site at the current node. The completion is bounded by the fraction of required resources that have been deposited. Once complete, the site will transform into a completed building. The action increases the completion at a fixed rate and has a variable duration as it can be preempted at any time.

2.2 Additional Modules

In this section we describe the modules that can be enabled to increase the scenario difficulty or introduce specific problem characteristics, and our motivation for each.

Resource Properties There are five types of resource (and corresponding mines), currently identified by a color code. Each resource type has a property that can be enabled.

- BLACK: takes up the entire inventory of the agent. Thus, any agent carrying one Black resource is unable to carry any other resources at the same time.
- BLUE: The DIG action for blue resource takes 12x longer than for other resources.
- RED: can only be mined within known time intervals, defined in the configuration file. The mining action must start and finish within the interval.
- ORANGE: requires two or more agents to cooperate at the same node, performing the DIG action together. Only one orange resource is produced.
- GREEN: decays over time. It will vanish from the node, site, or agent's inventory a fixed time after it is produced.

The black resource prevents enterprising agents from stockpiling all resource types in their own inventory. The blue, red, and green resources combine with the deadlines module described in the next section to introduce temporal constraints to the problem. The green resource also adds an exogenous process to the problem that is surprisingly tricky to model in temporal PDDL (Fox and Long 2003). The orange resource introduces required coordination between two agents.

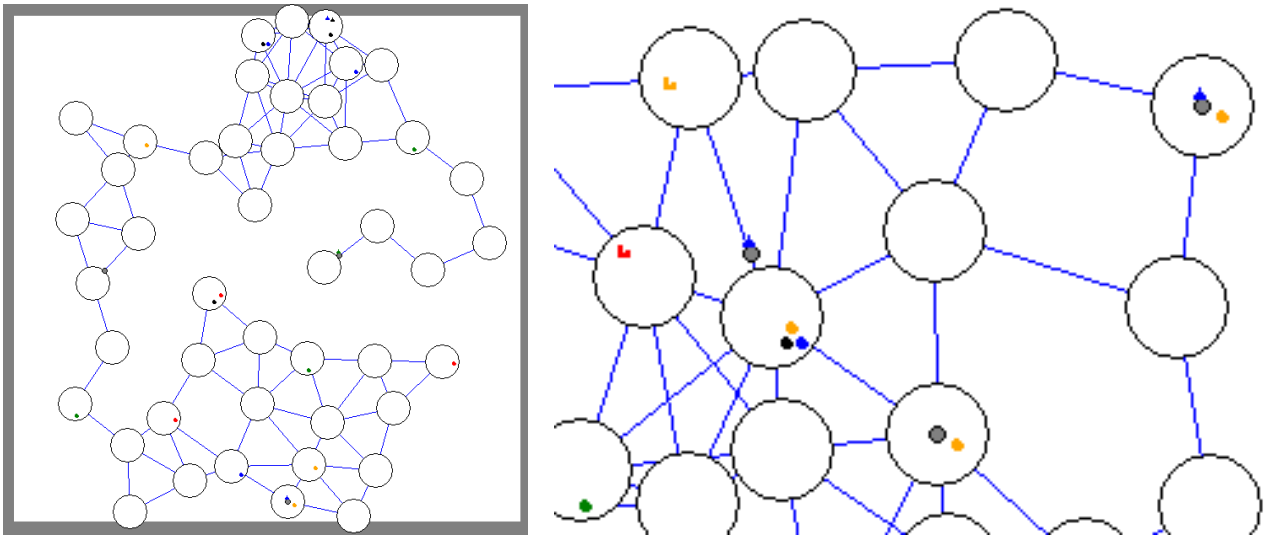

Figure 1: Work-in-progress graphics for CRAFTBOTS. The full simulation (left box) shows the nodes and network and enlarged section (right). Agents are shown as grey circles, mines as coloured dots, sites as stacked polygons, resources as coloured triangles. For example, in the enlarged view there are two agents carrying blue resources while there are yellow and red sites in the upper left.

Dynamic Goals, Deadlines, Oversubscription These three modules follow the examples set by the Robocup Logistics scenario.

- The simulation can be configured to produce dynamic task goals according to a randomised schedule.
- Each task goal can be associated with a deadline, by which time the building must be completed or it will not score points.
- Finally, the number of goals and tightness of deadlines stated in the initial state of the simulation, or produced according to the randomised schedule, will mean only a subset of the goals will be possible to complete.

Temporal Uncertainty Actions can be configured to have uncertain duration by specifying the mean and standard deviation duration for each action type.

Non-deterministic Actions Actions can be given configured to have a probability of failure. The effects of a failed action depend upon the action type. Digging actions end immediately without producing a resource; pick, drop, deposit, and start site actions will simply fail to produce their effects; failed movements result in the agent reversing direction towards the origin node and optionally disable the connection for a period of time; failed construct actions end immediately and halve the current progress of the building.

Building Properties In addition to the buildings required as the task goals, the agents are able to build additional buildings that do not score points, but provide a passive benefit. There are currently four different buildings available for agents to construct.

- BATTERY: increases the movement speed of the agents. Each constructed battery decreases the duration of movement times by 10 percentage points (pp), to a maximum of $50pp$.
- MANAGEMENT: increases the construction speed of the agents. Each constructed management increases the progress rate of the construction action by $10pp$, to a maximum of $100pp$.
- TOOLS: decrease the time required to mine resources. Each constructed tools decreases the duration of movement times by $10pp$, to a maximum of $50pp$.
- MILLS: increase the inventory capacity of all agents. This does not affect the property of the black resource.

These buildings are intended to present an interesting choice between focusing immediate efforts on collecting points, or investing into improving the situation in order to more easily gather points in the future. This choice requires reasoning about the horizon of the scenario.

3 Conclusion

We believe the core scenario presents a fairly simple problem to be planned for and enacted, and that by enabling all of the additional modules described above, the scenario represents a very challenging domain for both planning and execution. We also intend to include additional modules that introduce partial observability and limited communication between agents. We are bringing this to the Workshop on the IPC because we believe this workshop to be the best venue to gather feedback about which features and priorities are of the greatest interest for the planning community.

Acknowledgements

MR thanks ONR and NRL for funding portions of this work. GC has been supported by the EPSRC project THuMP (EP/R033722/1).

References

Akin, H. L.; Ito, N.; Jacoff, A.; Kleiner, A.; Pellenz, J.; and Visser, A. 2012. Robocup rescue robot and simulation leagues. *AI magazine* 34(1): 78.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. URL arXiv:1606.01540.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera Viñas, A.; Palomeras Rovira, N.; Hurtós Vilarnau, N.; and Carreras Pérez, M. 2015. Rosplan: Planning in the robot operating system. In *Proc. of ICAPS*, 333–341. AAAI Press.

Chevalier-Boisvert, M.; Willems, L.; and Pal, S. 2018. Minimalistic Gridworld Environment for OpenAI Gym. https://github.com/maximecb/gym-minigrid.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research* 20: 61–124.

Johnson, M.; Hofmann, K.; Hutton, T.; and Bignell, D. 2016. The Malmo platform for artificial intelligence experimentation. In *Proc. of IJCAI*, 4246–4247.

Kantharaju, P.; Ontañón, S.; and Geib, C. W. 2018. μCCG, a CCG-based Game-Playing Agent for μRTS. In *Proc. of Comp. Intell. and Games*, 1–8.

Levine, S.; and Williams, B. 2014. Concurrent Plan Recognition and Execution for Human-Robot Teams. In *Proc. of ICAPS*, 490–498.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2008. A deliberative architecture for AUV control. In *2008 IEEE International Conference on Robotics and Automation*, 1049–1054. doi:10.1109/ROBOT.2008.4543343.

Muscettola, N.; Dorais, G. A.; Fry, C.; Levinson, R.; Plaunt, C.; and Clancy, D. 2002. IDEA: Planning at the core of autonomous reactive agents. In *Workshop on On-line Planning and Scheduling at the Sixth International Conference on AI Planning and Scheduling*.

Niemueller, T.; Hofmann, T.; and Lakemeyer, G. 2019. Goal Reasoning in the CLIPS Executive for Integrated Planning and Execution. In *Proc. ICAPS*, 754–763.

Niemueller, T.; Karpas, E.; Vaquero, T.; and Timmons, E. 2016. Planning Competition for Logistics Robots in Simulation. In *ICAPS Workshop on Planning and Robotics (PlanRob)*.

Niemueller, T.; Lakemeyer, G.; and Ferrein, A. 2015. The RoboCup Logistics League as a Benchmark for Planning in Robotics. In *ICAPS Workshop on Planning and Robotics (PlanRob)*. Jerusalem, Israel.

Niemueller, T.; Reuter, S.; and Ferrein, A. 2015. Fawkes for the RoboCup logistics league. In *Robot Soccer World Cup*, 365–373. Springer.

Ontañón, S. 2013. The Combinatorial Multi-Armed Bandit Problem and Its Application to Real-Time Strategy Games. In *Proc. AIIDE*, 58–64.

Preiss, J. A.; Honig, W.; Sukhatme, G. S.; and Ayanian, N. 2017. Crazyswarm: A Large Nano-Quadcopter Swarm. In *Proc. ICRA*, 3299–3304.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3.

Roberts, M.; Hiatt, L. M.; Shetty, V.; Brumback, B.; Enochs, B.; and Jampathom, P. 2021. Goal Lifecycle Networks For Robotics. In *Proc. of FLAIRS*.

Sheh, R.; Schwertfeger, S.; and Visser, A. 2016. 16 Years of RoboCup Rescue. *KIJ* 30(3): 267–277.