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.

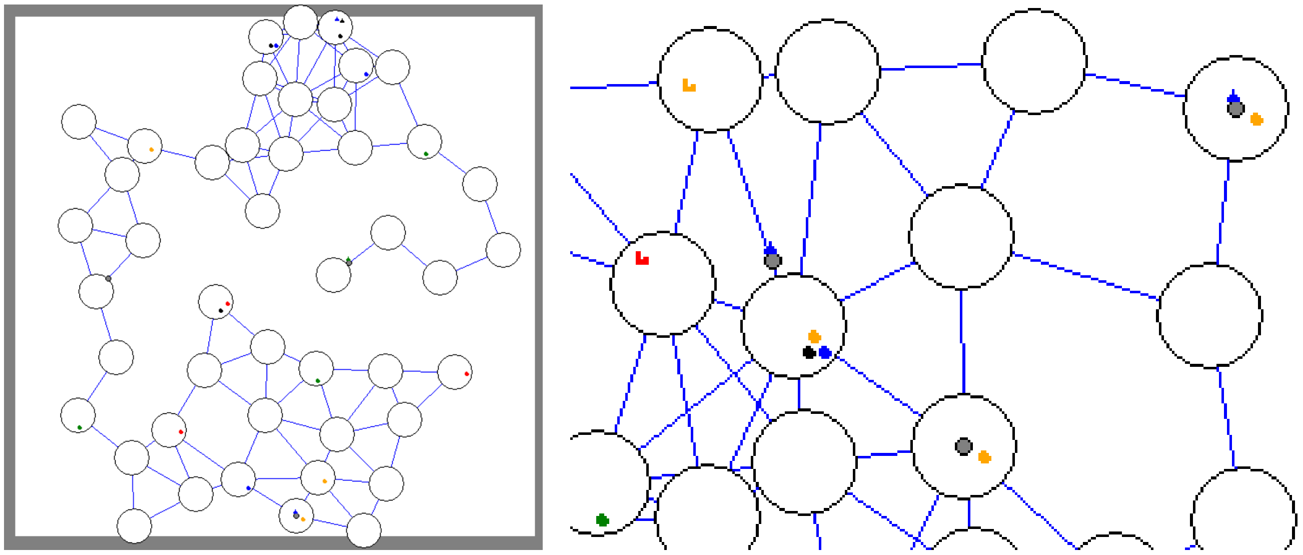

Figure 1: Work-in-progress graphics for CRAFTBOTS. The full simulation (left box) shows the nodes and network and enlarged section (right). Agents are shown as grey circles, mines as coloured dots, sites as stacked polygons, resources as coloured triangles. For example, in the enlarged view there are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