Cleaning Efficiently:
A Case Study in Modeling Goal Reasoning and Learning

Mark Roberts MARK.ROBERTS.CTR@NRL.NAVY.MIL
NRC Postdoctoral Researcher, Naval Research Laboratory (Code 5514), Washington, DC USA

David W. Aha DAVID.AHA@NRL.NAVY.MIL
Adaptive Systems Section, Naval Research Laboratory (Code 5514), Washington, DC USA

Abstract

Goal Reasoning concerns actors that deliberate about their own goals, and learning is often applied to improve performance of such actors. A recent model of Goal Reasoning proposed by Roberts et al. lacks a simplified working example with an explanation of how the model can incorporate learning. We describe a thought experiment modeling the decision making of floor-cleaning robots, which are simple goal reasoning actors. We observe that such robots often have a less-than-optimal action policy that, though sufficient for cleaning in the limit, could be improved through learning to reduce the total cleaning time or reduce energy usage. We start with by modeling the straightforward policy for a simple robot named Vacuous. We then ponder a hypothetical robot, named KleeneStar, which optimally cleans in the fastest time possible. Finally, we describe a model for an improved robot, named Vakleene, which iteratively reduces its cleaning time through learning. While these examples are only a thought experiment, this work provides a model of a simplistic goal reasoning process that can learn.

1. Motivation

Small, floor-cleaning robots have become a popular home appliance to supplement regular floor cleaning performed by a human. These systems are best exemplified by the Roomba, introduced by iRobot in 2002. The Roomba contains a front bumper to detect a collision, infrared proximity sensors to detect nearby objects or sudden changes in height, and a radio sensor to detect beacons. It can clean a variety of floor surfaces for several hours between charges and newer versions use radio beacons to navigate multiple rooms, avoid hazards, or return to a self-charging base station.

Anyone who has interacted with these devices can observe that its task planning is rudimentary. The devices exhibit three simple behaviors as shown in Figure 1 taken from the Roomba's User Manual (iRobot 2008). The Spiral behavior allows the device to clean a large area in a spiral fashion. The Wall Following behavior traces a wall or another obstacle to the right side while cleaning close to the edge. Finally, the Room Crossing behavior randomly turns and crosses the room while cleaning. Even with random selection of the actions, the robot should eventually traverse the entire floor in the limit. While sufficient, this action selection mechanism could be improved with learning that accounts for the room and obstacles within the room.

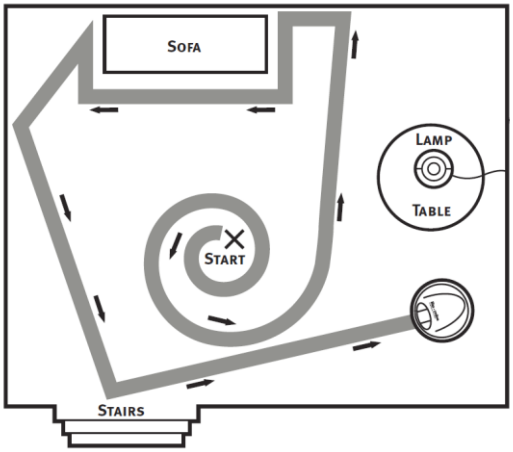

Figure 1. A typical Roomba cleaning pattern from the Roomba User Manual (iRobot 2008).

We will show in this paper how a floor-cleaning robot can be modeled as an instantiation of a Goal Reasoning process. We created a Goal Reasoning Model that captures the variety systems that perform Goal Reasoning (Roberts et al. 2014, 2015). Our model distinguishes systems by their design choices and, thus, facilitates their comparison. For example, instantiations of this model can represent iterative plan repair (e.g., Chien et al. 2000), replanning (e.g., Yoon et al. 2007), and Goal-Driven Autonomy (e.g., Klenk et al. 2013).

We have two aims in this paper. First, we model the robot vacuum as a Goal Reasoning process using the Goal Reasoning Model. Second, we describe how learning could augment this basic model to reduce cleaning time. We present a thought experiment to elucidate instantiations of Goal Reasoning, using the robot vacuum as a case study. We begin with a brief exposition of the model, and then use it to model three robot vacuum agents: (1) Vacuous is a simple robot with a straightforward, fixed action selection policy that is extremely suboptimal. (2) KleeneStar[1] is a hypothetical optimal robot vacuum that can guarantee cleaning in the minimal cleaning time but whose policy could never be realistically computed on such a limited robot platform. (3) Vaklene is a learning-enabled robot vacuum whose cleaning time asymptotically approaches KleeneStar.

2. Background: The Goal Reasoning Model

Deliberating about objectives – how to prioritize and attain (or maintain) them – is a ubiquitous activity of all intentional entities (i.e., actors). We apply the Goal Reasoning Model presented by Roberts et al. (2014, 2015) that models Goal Reasoning as a State Transition System consisting of goal nodes that track a goal's state and transitions defined by a Goal Lifecycle. Note that we adopt the notation provided by Roberts et al. (2015).

Let g_i be the actor's i^{th} goal of m goals ($0 \leq i \leq m$) that the actor wishes to attain (maintain). To avoid confusion with the use of the word *state* as it is typically applied in planning systems, we will use L to represent the *language* of the actor, $L = L_{external} \cup L_{internal}$. Often, $L_{external}$ will concern external state the actor is tracking (e.g., its location, its sensor values). $L_{internal}$

[1] With apologies to Stephen Kleene.

represents the predicates and state required internally to the actor (e.g., the predicates $attain(g)$ or $maintain(g)$, the state of all goals).

A goal g is tracked within the actor as part of a goal node N^g that transitions according to a Goal Lifecycle (Figure 2). Decisions consist of applying a *strategy* (arcs in Figure 2) to transition a goal node among *modes* (rounded boxes). Goal nodes in an *active* mode are those that have been formulated but not yet dropped. The **formulate** strategy determines when a new goal node is created. Vattam et al. (2013) describe goal formulation strategies. The **drop** strategy causes a goal node to be "forgotten" and can occur from any active mode; this strategy may store the node's history for future deliberation. To **select** N^g indicates intent and requires a formulated goal node. The **expand** strategy decomposes N^g into a goal network (e.g., a tree of subgoal nodes) or creates a (possibly empty) set of expansions X. Expansion is akin plan generation, but is renamed here to generalize it from specific planning approaches. The **commit** strategy chooses an expansion $x \in X$ for execution; a static strategy or domain-specific quality metrics may rank possible expansions for selection. The **dispatch** strategy slates x for execution; it may further refine x prior to execution (e.g., it may allocate resources or interleave x's execution with other expansions).

Goal nodes in executing modes (Figure 2, dashed lines) can be subject to transitions resulting from expected or unexpected state changes. The **monitor** strategy checks progress for N^g during execution. Execution updates arrive through the **evaluate** strategy. In a nominal execution, the information can be either resolved through a **continue** strategy or the **finish** strategy marks the goal node as *finished*.

During execution, the **evaluate** strategy determines how events or new information impacts goal node and the **resolve** strategies define the possible responses. If the evaluation does not impact N^g, the actor can simply **continue** the execution. However, if the event impacts the current execution then other strategies may apply. One obvious choice is to modify the world model using **adjust**, but this does not resolve the mode of N^g and further refinements are required. The **repair** strategy repairs expansion x so that it meets the new context; this is frequently called *plan repair*. If no repair is possible (or desired) then the **re-expand** strategy can reconsider a new plan in the revised situation for the same goal; this is frequently called *replanning*. The **defer** strategy postpones the goal, keeping the goal node *selected* but removing it from execution. Finally, **formulate** creates a revised goal g'; the actor may then drop the original goal g to pursue g' or it could consider pursuing both goals in parallel; this is similar to the concept of goal transformation provided by Cox and Veloso (1998).

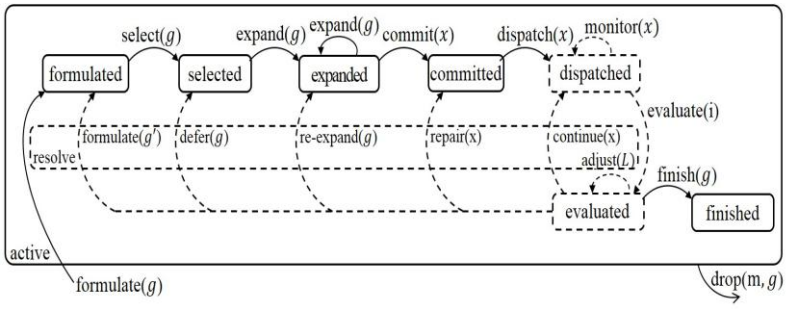

Figure 2: The goal lifecycle (Roberts et al. 2014). Strategies (arcs) denote possible decision points of an actor, while modes (rounded boxes) denote the status of a goal (set) in the goal memory.

Goal Reasoning can be modeled as a goal State Transition System $\Sigma = (M, R, \delta_{GR})$, where M is a goal memory of goal nodes, R is a set of refinement strategies that transition goals of the system, and $\delta_{GR} : M \times R \rightarrow M'$ is a transition function restricting the allowed transitions for M.

A goal memory M stores a goal node for each of the m goals. For goal g_i, $N^{g_i} = (g_i, parent, subgoals, C, o, X, x, q)$ is a goal node where:

- g_i is the goal that is to be achieved (or maintained);
- $parent$ is the goal whose subgoals include g_i;
- $subgoals$ is a list containing any subgoals for g_i;
- C is a set of constraints on g_i. Constraints could be temporal (finish by a certain time), ordering (do x before y), maintenance (remain safe), resource (use a specific resource), or computational (only use so much CPU or memory). A partition $C = C^{provided} \cup C^{added}$ separates constraints into those provided to the actor independent of whatever invoked it (e.g., a human operator, meta-reasoning process, or coach) and those added during refinement. Top-level constraints can be pre-encoded or based on drives (e.g., (Coddington et al. 2005; Young and Hawes 2012)). Hard constraints in C must be satisfied at all times, while soft constraints should be satisfied if possible.
- o is the current goal lifecycle mode (detailed below).
- X is a set of expansions that will achieve goal g_i. The types of expansions for a goal depend on its type. For goals tracking external state, expansions might include a set of plans Π. Other goals might expand into a goal network, a task network, a set of parameters for flight control, etc. The **expand** strategy creates X.
- $x \in X$ is the currently selected expansion, performed with the **commit** strategy.
- q is a vector of one or more quality metrics. For example, these could include the *priority* of a goal, the *inertia* of a goal indicating a bias against changing its current mode because of prior commitments, the net value (e.g., cost, value, risk, reward) associated with achieving g_i using selected expansion x.

The refinement strategies R are drawn from the Goal Lifecycle (Figure 2). For convenience, we sometimes refer to the goal node N^g as simply the goal g, though it should be clear that all strategies are functions that transition some N^g. We partition the refinement strategies $R = R^{provided} \cup R^{added} \cup R^{learned}$ to distinguish between strategies that the actor was provided prior to the start of its lifetime (e.g., through design decisions), representations that were added to its model as a result of execution in an environment (e.g., a new object is sensed), and those it learned for itself (e.g., the actor adjusts its expectations for an action after experience).

The transition function δ_{GR} specifies the allowed transitions between modes because not every strategy will apply to every goal or every situation. In a domain-independent fashion, δ_{GR} is defined by the arcs in the lifecycle. However, a system or domain may modify (through composition, ablation, or additional constraints) the transitions for M.

Once Goal Reasoning is modeled as Σ, it is easy to see that it is a process through which M is iteratively refined through transitions of R as restricted by δ_{GR}. The Goal Reasoning Problem and one way of solving it, called Goal Refinement, is defined by Roberts et al. (2014, 2015b). For space reasons, we focus only on the definition of Σ for our cleaning robots in this paper.

3. Modeling the robots

As mentioned, the available actions of a Roomba include Spiraling, Wall Following, and Room Crossing. Here we outline our central assumptions and more carefully define these actions so that we can compose them into policies for the three robots Vacuous, KleeneStar, and Vakleene.

We make several assumptions to perform our analysis: static environment, perfect sensing, perfect localization, and deterministic action outcome. These assumptions are unrealistic, given that the robotic platform's sensors and actuators are inexpensive and produce noisy observations. Our assumptions are easily violated by the presence of certain furniture (e.g., short legs that cause the robot to get stuck under an edge) or dynamic objects (e.g., a pet or person). Nevertheless, making these assumptions is necessary to gain traction on our analysis. Because we can model each of these assumptions in a simulator, we can accurately measure their impact on our analysis.

The actuator action space of the robot consists of *Vacuum*, *Turn(radians)*, and *Drive(timeInSeconds)*. Vacuum simply turns on the vacuum motor. Turn is a straightforward action that is always possible. Drive is conditioned on sensor observations. The *bumper* is true iff the robot has driven into an obstacle anywhere along its front side. The *right* obstacle sensor returns the distance to any object on the robot's right side up to 50 centimeters; similarly there are *left* and *front* obstacle sensors. These sensors are strategically angled toward the floor so that they should always return some value within 10 centimeters. If the sensor traverses an edge (e.g., the top step of a stairwell) it returns a very large value, indicating a drop that the robot should avoid. With these sensors observations, it is possible to define a simple, effective *drive* action as follows:

```
drive(timeInSeconds)
    startInSeconds ← getTimeInSeconds()
    elapsedInSeconds ← 0
    while (isSafe() && (elapsedInSeconds < timeInSeconds))
        move_forward(0.1 seconds)
        elapsedInSeconds ← getTimeInSeconds() – startInSeconds
```

where the function isSafe() returns true if (bumper == false) && (front < 10) && (left < 10) && (right < 10) && (front > 3) && (left > 3) && (right > 3).

Turn and Drive are composed into the following complex tasks:

- **Sprial(timeInSeconds)** is composed of turning on the vacuum and executing a series of alternating turn right and drive commands where the radians turned decreases and the time driven increases over time.
- **Follow(timeInSeconds)** is composed of a loop that ensures the robot is between 3 and 5 centimeters of an obstacle on the right and drives forward by half-second intervals, checking to perform corrective turns as needed.
- **Cross(turnInRadians, timeInSeconds)** alternates Turn(turnInRadians) followed by Drive(timeInSeconds). If an obstacle is detected during Drive, the robot continues to the next Turn action.
- **ReturnToBase()** reactively applies Turn and Drive actions to move the vehicle toward the Base, which is sending out a beacon signal.

3.1 Vacuous: The sufficient but sub-optimal robot vacuum

Let us now define a simple task selection policy to clean a floor for the Vacuous. This policy should eventually traverse the entire floor, though it will cover certain areas repeatedly. There may be some cases where obstacles or room shapes cause cycling (partially avoided by using a

prime number in the turn radius), in which case it would be easy to add a random component to this policy to break such cycling.

```
VacuousPolicy(cleaningTimeInSeconds)
    startInSeconds ← getTimeInSeconds()
    elapsedInSeconds ← 0
    while (isSafe() && (elapsedInSeconds < cleaningTimeInSeconds))
        time ← randomInt(30)
        if (3< right < 5) Follow(time)
        else
            task ← randomInt(1) //randomly select either 0 or 1
            if (task == 0) then Spiral(time)
            else
                turn ← −(19 × Π/180)  // Turn by prime closest to 20 degrees
                Cross(turn, time)
        elapsedInSeconds ← getTimeInSeconds() – startInSeconds
    returnToStation()
```

To model this policy as a Goal Reasoning process, we define a system architecture and its goals. Figure 3 shows the system architecture of our model. The Goal Reasoner manages goals, refines them via strategies, and sends a single instantiated task to the Executive, which applies Turn or Drive commands to the robot. The Executive receives sensor readings from the robot and abstracts them into updates for the Goal Reasoner.

We define goals for maintaining safety and signaling that the cleaning time has expired plus a goal for each task. All goals in this model have identical formulate and select strategies. When the goal memory of Vacuous initializes, it formulates and selects all goals. The goals are then triggered by specific events. The TimeExpiredGoal signals the end of the cleaning time. The expand strategy is a NoOp and this goal is immediately dispatched. It monitors the system time and calls evaluate(TimeExpired) to all other goals. The goal is then marked as finished and dropped. The MaintainSafetyGoal ensures the robot remains safe. Similar to the TimeExpiredGoal, the expand strategy is a NoOp and this goal is immediately dispatched. It monitors the sensor status and, when the bumper is activated or when left/right/front detects a drop, it calls evaluate(Unsafe) to all other dispatched goals. Since this goal is central the robot's safety, it is never marked as finished or dropped (i.e., its finish and drop strategies are undefined thus prohibiting these transitions).

The remaining goals are named according to the task they manage: SprialGoal performs the Sprial(timeInSeconds) task, FollowGoal performs Follow(timeInSeconds), CrossTaskGoal performs Cross(turnInRadians, timeInSeconds), and ReturnToBaseGoal performs ReturnToBase(). Each of these goals is selected automatically (as are all goals in this domain) and will then cycle through the expanded, committed, dispatched, and evaluated modes during the execution of a cleaning cycle. Thus, the strategies of the task-oriented goals form the core functionality of this model for controlling the robot. For each goal managing a task, the expand strategy creates a single policy that is eventually dispatched to perform its respective task with the appropriate random time (or turn) variables. Since only one task can be sent to the Executive at

Figure 3: The system architecture of the floor-cleaning robot.

one time, the goals must coordinate via a global semaphore, ExecutiveLock. Therefore, the commit strategies of the goals check this lock before they commit to performing the task and release the lock when applying a resolution strategy that results in moving back to selected (via the defer strategy). This provides an opportunity for other goals to apply commit if they are able to do so. Otherwise, a task goal stays in dispatched until it receives information, via the integrate strategy, that it should pause. The TimeExpiredGoal or MaintainSafetyGoal can signal TimeExpired or Unsafe and a goal transition to committed if one of these is active. At this point, any dispatched task goal should apply an evaluate strategy followed by a resolve strategy.

Once formulating the goals, the Goal Reasoner simply iterates through the goals in the system and attempts to apply the next available strategy. If a goal can proceed, it does. Otherwise it remains in its current mode.

3.2 KleeneStar: The optimal robot vacuum

It is straightforward to show that an optimal policy exists for the robot vacuum given the simplifying assumptions we made above. We can reduce this problem to the Travelling Salesperson Problem (TSP). First, we discretize the floor into cells small enough such that a "visit" by the robot equates to having cleaned that cell. We then label each cell and create an adjacency map of the entire map. We solve the problem with any algorithm suitable for the TSP.

Unfortunately, TSP is an NP-Hard problem, and we face some obstacles implementing it on the limited computing platform of the robot's microcontroller. This leads us to our approximation using machine learning techniques, as outlined next.

3.3 Vakleene: The sub-optimal robot vacuum that learns

We claim that applying learning could improve the cleaning time of Vacuous. The parameters for learning are straightforward in this model, and consist of learning how often to apply each task and how long to perform each task. Let $cleanTime$ be the maximum time for the robot to clean. For task i, let p_i denote the probability of performing the task, let $t_i = (\text{minTime}_\text{i}, \text{maxTime}_\text{i})$ denote the minimum and maximum time to perform the task where $0 \leq \text{minTime}_\text{i} \leq \text{maxTime}_\text{i} < \text{cleanTime})$, and let $r_i = (\text{minRadian}_\text{i}, \text{maxRadian}_\text{i})$ denote the minimum and maximum radians the robot can turn for the task where $-\pi \leq \text{minRadian}_\text{i} \leq \text{maxRadian}_\text{i} \leq \pi$. Then an instantiation of the system is determined by the parameters $F = (cleanTime, reward, p, t, r)$, where $reward$ is a count of the number of cells visited, p, t, r are the task probabilities, time and radian values and $p_{Spiral} + p_{Follow} + p_{Cross} = 1$.

It should be clear that the Vacuous is one instantiation of F. The parameters chosen for Vacuous may have given the best performance across a variety of rooms and obstacles. We hypothesize that a specific room (and room obstacles) lead to a structure that favors particular instantiations of F. If true, then Vacuous could be improved by learning F for the room that it is currently cleaning, leading to Vakleene, a cleaning robot that can learn.

Vakleene could apply learning (at least) two ways. First, it could perform online modification of F based on how much cumulative reward each task earns over time. For example, Vacuous could proportionally increase the time t_i for task i if, during the past k tasks run, it earned more reward; it might even adjust F as a function of the discounted reward over the past k tasks run (cf. Kaebling, Littman, & Moore 1996). Similarly, Vakleene could modify r_i for a task i that is getting insufficient reward. To help it avoid local minima in the parameter space, Vakleene could use a stochastic component (e.g., simulated annealing) to broaden its parameter search. Online

learning like this can be modeled as part of the defer strategy, which checks the last k tasks run and modifies F appropriately.

There are a number of problems with this approach. Learning is performed during cleaning, when computational resources are probably better spent maintaining the robot's safety. This process is an iterated search over a large parameter space. Given the relative infrequency of interaction with its environment (i.e., daily cleaning), converging to a maximal (or even reasonable) policy may likely take many months and Vakleene may traverse many poor configurations for F. Finally, our solution is brittle from a software engineering perspective because changes to the learning procedure require modifying every task goal.

Another approach may be to introduce an offline learning goal, which adjusts the parameters while the vehicle is charging. Consider a new goal, **ApproximateKleeneStarGoal**, which could be dispatched during charging to consider the rewards earned during the past k runs, consider any global constraints for good configurations of F, consider how often obstacles or drops were encountered, etc. If the robot were provided more accurate localization in a later version, this goal could also consider this information. Software updates to the system then consist of patching one or more strategies of ApproximateKleeneStarGoal. As an advantage, this goal could be provided (or learn) a table of common room types to speed up its learning.

There are many other ways that learning could be modeled in the Goal Reasoning Model, but these two examples suffice to demonstrate our point.

4. Related Work on Machine Learning and Goal Reasoning

Several researchers have investigated methods for applying machine learning techniques to improve the performance Goal Reasoning agents. For example, Goal Reasoning has been used in the context of learning to compose web services (Burstein et al. 2008). The most common focus is applying learning for goal formulation or goal priorities (Weber et al. 2010, 2012; Powell et al. 2011; Young and Hawes 2012; Maynord et al. 2013; Silva et al. 2013). Other researchers have studied learning in the context of explanation generation (Molineaux and Aha 2014, 2015) or learning a combination of state expectation and goal formulation knowledge (Jaidee et al. 2013).

The Goal Lifecycle bears close resemblance to that of Harland et al. (2014) and earlier work (Thangarajah et al. 2010). They present a goal lifecycle for BDI agents, provide operational semantics for their lifecycle, and demonstrate the lifecycle for an agent that controls a simulated Mars rover. In future work we plan to characterize how this lifecycle relates to the one we presented in (Roberts et al. 2014). Winikoff et al. (2010) have linked Linear Temporal Logic to the expression of goals.

5. Closing Remarks

We have applied our Goal Reasoning Model to a floor-cleaning robot and described how learning could be incorporated into the model. Although this model is presented as a thought experiment, we plan to implement the model to analyze a comparison of the approaches we outlined for a variety of simulated room types. Future work will also include characterizing when and why particular instantiations of Vakleene, the learning robot, outperforms Vacuous, the simple robot. We also plan to formally characterize how close Vakleene's learned policies can approximate KleeneStar's optimal policies.

Acknowledgements

Thanks to OSD ASD (R&E) for sponsoring this research. The views and opinions in this paper are those of the authors and should not be interpreted as representing the views or policies, expressed or implied, of NRL or OSD.

References

Burstein, Mark H., Robert Laddaga, David D. McDonald, Michael T. Cox, Brett Benyo, Paul Robertson, Talib S. Hussain, Marshall Brinn, and Drew V. McDermott. "POIROT-Integrated Learning of Web Service Procedures." In *AAAI*, 1274–79, 2008.

Chien S., Knight R., Stechert A., Sherwood R., and Rabideau, G. (2000) Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling. Proc. of the Conf. on Auto. Plan. and Sched.(pp. 300-307). Menlo Park, CA: AAAI.

Cox, M. T., & Veloso, M. (1998). Goal Transformations in Continuous Planning. In M. desJardins (Ed.), Proc. of the Fall Symposium on Distributed Continual Planning (pp. 23-30).Menlo Park, CA: AAAI Press.

Harland, J., Morley, D., Thangarajah, J., & Yorke-Smith, N. (2014). An operational semantics for the goal life-cycle in BDI agents. Auton. Agents and Multi-Agent Systems, 28(4), 682–719.

iRobot. 2008. Roomba Manual.

Jaidee, U., Munoz-Avila, H., & Aha, D.W. (2011). Integrated learning for goal-driven autonomy. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*. Barcelona, Spain.

Jaidee, U., Munoz-Avila, H., & Aha, D.W. (2013). Case-based goal-driven coordination of multiple learning agents. *Proceedings of the Twenty-First International Conference on Case-Based Reasoning* (pp. 164-178). Saratoga Springs, NY: Springer.

Kaelbling, L.P., Littman, M.L., & Moore, A.P. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 237-285.

Klenk, M., Molineaux, M., & Aha, D.W. (2013). Goal-driven autonomy for responding to unexpected events in strategy simulations. Comp. Intell., 29(2), 187-206.

Maynord, Michael, Michael T. Cox, Matt Paisner, and Don Perlis. "Data-Driven Goal Generation for Integrated Cognitive Systems." In *2013 AAAI Fall Symposium Series*, 2013.

Molineaux, M., & Aha, D.W. (2014). Learning unknown event models. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. Quebec City (Quebec), Canada: AAAI Press.

Molineaux, M., & Aha, D.W. (2015). Continuous explanation generation in a multi-agent domain. To appear in *Proceedings of the Third Conference on Advances in Cognitive Systems*. Atlanta, GA: Cognitive Systems Foundation.

Powell, J., Molineaux, M., & Aha, D.W. (2011). Active and interactive learning of goal selection knowledge. In *Proceedings of the Twenty-Fourth Florida Artificial Intelligence Research Society Conference*. West Palm Beach, FL: AAAI Press.

Roberts, M., Apker, T., Johnson, B., Auslander, B., Wellman, B. & Aha, D.W. (2015a). Coordinating Robots for Disaster Relief. Proc. of the Conf. of the Florida AI Research Society (to appear) Hollywood, FL: AAAI.

Roberts, M., Vattam, S., Alford, R., Auslander, B., Karneeb, J., Molineaux, M., Apker, T., Wilson, M., McMahon, J., & Aha, D.W. (2014). Iterative goal refinement for robotics. In Working Notes of the Planning and Robotics Workshop at ICAPS. Portsmouth, NH: AAAI.

Roberts, M., Vattam, S., Alford, R., Auslander, B., Apker, T., Johnson, & Aha, D.W. (2015b). Goal Reasoning to Coordinate Robotic Teams for Disaster Relief. In Working Notes of the Planning and Robotics Workshop at ICAPS. Jerusalem, Israel. AAAI.

Silva, Michael, Silas McCroskey, Jonathan Rubin, Michael Youngblood, & Ashwin Ram. "Learning from Demonstration to Be a Good Team Member in a Role Playing Game." In *FLAIRS Conference*, 2013.

Thangarajah, J., Harland, J., Morley, D., & Yorke-Smith, N. (2011). Operational behaviour for executing, suspending, and aborting goals in BDI agent systems. In Declarative Agent Lang. and Technologies VIII (pp. 1–21). Toronto, Canada: Springer.

Vattam, S., Klenk, M., Molineaux, M., & Aha, D. W. (2013). Breadth of approaches to goal reasoning: A research survey. In D.W. Aha, M.T. Cox, & H. Muñoz-Avila (Eds.) Goal Reasoning: Papers from the ACS Workshop (Tech. Report CS-TR-5029). College Park, MD: Univ. of Maryland, Dept. of Comp. Science.

Weber, Ben George, Michael Mateas, & Arnav Jhala. "Applying Goal-Driven Autonomy to StarCraft." In *AIIDE*, 2010.

Weber, Ben George, Michael Mateas, and Arnav Jhala. "Learning from Demonstration for Goal-Driven Autonomy." In Proc. *AAAI*, 2012.

Winikoff, M., Dastani, M., & van Riemsdijk, M. B. (2010). A unified interaction-aware goal framework. In Proc. of ECAI (pp. 1033–1034). Lisbon, Portugal: IOS Press.

Yoon, S.W., Fern, A., & Givan, R. (2007). FF-Replan: A baseline for probabilistic planning. Proc. of the 17th Int'l Conf. on Auto. Plan. and Sched. (pp. 352-359). Providence, RI: AAAI Press.

Young, Jay & Nick Hawes. "Evolutionary Learning of Goal Priorities in a Real-Time Strategy Game." In *AIIDE*, 87–92, 2012.