A Motivational System That Drives the Development of Activity

Keywords: machine learning, cognitive modeling, planning, autonomous agents
Tracking Number: 706

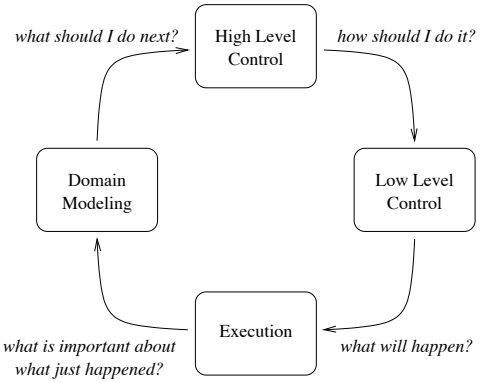

Figure 1: An interactive model of the development of activity.

1 Introduction

The interactionist philosophy of Lakoff, Johnson, and others [2] places activity at the center of conceptual development. Not only activity, but also structures such as classes, concepts, and language are all rooted deeply in the physical interaction between an agent and its environment. Explaining the development of activity, then, is central to any interactionist account of conceptual development.

A simple model of the development of activity is pictured in figure 1. This model implies that development is an cycle of incremental learning in which an agent engages in four kinds of processes: one in which an agent decides *what to do* (high-level control), one in which the agent decides *how to do it* (low-level control), one in which the agent attempts to engage in the desired behavior (execution), and one in which the agent learns from its interactions (modeling). Two kinds of knowledge can be generated in this model. From the low-level controller comes procedural knowledge of how to achieve the goals of the high-level control system (activities), and from the modeling component comes domain knowledge (classes, concepts, etc).

It should be clear that the initial decision of *what to do* has tremendous impact on the development of activity, and knowledge in general. An agent that fixates on one particular type of activity will likely learn about that activity in great depth, but little else. An agent that acts randomly will likely learn a great breadth of superficial activities, but not learn about anything in particularly great depth. A high-level control component that ignores the basic needs of the agent will almost certainly likely lead to the demise of the agent.

Numerous approaches to the modeling and low-level control processes exist in literature. We take the execution component to be the native ability of a developing agent to execute some set of primitive actions out of which it will build activities. In this paper, we defer detailed discussion of how we have implemented these three components, and instead focus on our implementation of a high-level control component that can be responsible for driving a developmental process in which an agent learns from interactions with the environment.

In the next section, we characterize the requirements of a working high-level control system are, and describe our approach to high-level control. In the section that follows, we describe two sets of experiments that demonstrate the effectiveness of our high-level control scheme. Finally, we conclude with a discussion of why our system works, and our plans to integrate this control scheme with the other components of 1 we have already implemented for the Pioneer-2 mobile robot.

2 High-Level Control

The task of a high-level control component is to produce a task specification that the low-level control system can use to produce an *activity*. In related work, it has been argued that means-ends analysis planning, such as that employed by GPS [4] and STRIPS [1], is a good candidate for a low-level control system [3]. If we adopt a planner as a low-level control system, then, the task of a high-level control component is to generate goals for the low-level control component to satisfy and the execution component to achieve.

Furthermore, a high-level control system should meet the following criteria:

- **Plausibility**: the system should produce goals that are possible for the agent to achieve.
- **Sustainability**: the system should attend to the basic needs of the agent. This includes vegetative needs like hunger, thirst, and fatigue, if they apply, as well as related needs like aversion to pain. The high-level control system must keep an agent out of trouble.
- **Learning**: the system should provide and agent with opportunities to learn about its environment.
- **Purposefulness**: the system should behave with purpose. Let us define an agent that acts *purpose-*

fully as one in which the decision to do something is always based on some criteria, and not simply random.

We have developed a simple high-level control system based on modeling motivation in intelligent agents that possesses these four desirable qualities. We address the first quality by adopting a philosophy of goal selection in development which we call *planning to act*, which we describe in the next section, and then describe how the motivational system fits within this framework and satisfies the other three criteria.

2.1 Goals in Development

The problem of how to generate goals for a planner to achieve is one that is traditionally not automated. Planning goals are, in general, specified as desirable sensory or perceptual states by some external source, usually the experimenter. This method of goal selection typically ensures that the planner will be working on a tractable, if not useful problem, but leaves a lot to be desired in an account of development.

Generating goals automatically is, on the surface, a daunting task. If we take a reasonably complicated agent, such as the Pioneer-2, with some 65 real-valued sensors, the space of goals is multidimensional and unbounded. On top of that, most of the 65-dimensional, continuous space represents situations that are not possible or highly improbable, such as positive translational velocity while the stall sensors are reporting the robot is stalled. What makes a sensory state with sonar-3 reporting a value of 715.3 a better goal that one in which sonar-3 is reporting 802.1? On the surface, automatic goal generation by a developing agent is horribly underconstrained.

On the other hand, vast regions of this 65-dimensional, continuous goal space are roughly equivalent at various times in an agent's decision-making process. If the robot is considering turning in place, it can base its plans on whether or not its sonar reports something is in the way. In this case, if the sonar reports anything beyond 250 millimeters, the turn will succeed. It doesn't matter if it reports 250, 500, or 5000 meters. This part of the 65-dimensional, continuous space really only has two values that matter, "under 250" and "over 250".

There is a simple way to leverage this observation into a tractable solution to the goal-selection problem. The key to the observation that a highly dimensional space can be carved up into a few regions that matter is that intention is based on activity. The Pioneer's sonar space can be collapsed to a binary decision because the agent's goal is to turn, not to achieve an arbitrary state. Said differently, the goals of an agent, and those things that are rewarding to the agent, are *activities*, not sensory configurations or states of being. We call this paradigm *planning to act*, and this simple distinction is crucial to high-level control for the following reasons:

- It constrains goal selection to a space of goals that is bounded, and small. An agent looks at its list of things it has done, and picks the one it likes best to do again.
- It constrains goal selection to a space of goals that are physically possible. It's not choosing arbitrary states, it's selecting from *things it has already done before*.

All that remains is to define a framework for preferring one activity over another that meets the criteria of sustainability, learning, and purposefulness.

2.2 A Model of Motivation

The planning to act framework reduces the high-level control process to a matter of evaluating the activities available to an agent and choosing one according to that evaluation. The details of how one does this determine how well the criteria of sustainability, learning, and purposefulness are attended to. Perhaps the simplest way to perform this evaluation is to define a preference relation, and simply choose the activity that is preferable to all others.

Let $\mathcal{A}$ denote the set of all the activities that an agent knows about. The preference relation for agent x, $\psi_x(a_1, A_2, s_t)$, holds iff in sensory state s_t agent x prefers to attempt a_1 over a_2. The goal of the system at time t, then, is a_g such that $\forall a_{n \neq g}(\psi_x(a_g, a_n, s_t))$.

If we were interested only in the learning criterion, we could base ψ on information gain. Thus, every decision that agent x would make would be based on the amount of information to be gained about a particular action. Agent x would be constantly exploring. Likewise, if the agent had some single, all-encompassing vegetative concern, such as keeping its battery from running out, we could base ψ on the expected loss or gain of battery power. This agent would constantly be attempting to maintain its battery level. While both of these formulations of ψ satisfy the purposefulness criterion, neither satisfies all three.

Rather, an agent whose goal is to truly explore the affordances of its environment, all the while ensuring to stay alive, must attend to a variety of factors that motivate its behavior. The relation ψ should reflect the variety of these factors. For a mobile robot exploring the surface of Mars, for example, ψ may reflect three distinct factors: *fatigue*, or the need of the robot to recharge its battery when the voltage gets low, *crash avoidance*, or the need of the robot to keep from colliding with other objects at high speeds, and *curiosity*, or the need of an agent to improve its models of activity and the environment when they are not acceptably predictive. We call each of these components to behavior a *motivational factor*. Individually, each expresses a basic need of the agent, and collectively, they comprise behavior.

In our model of motivation, we represent each motivational factor F numerically with a *drive coefficient* $\mu_F(s_t)$. The drive coefficient expresses the relative important of F in state s_t; the importance of the fatigue factor, for instance, may increase as the battery voltage decreases. Likewise, each activity a_n will have an

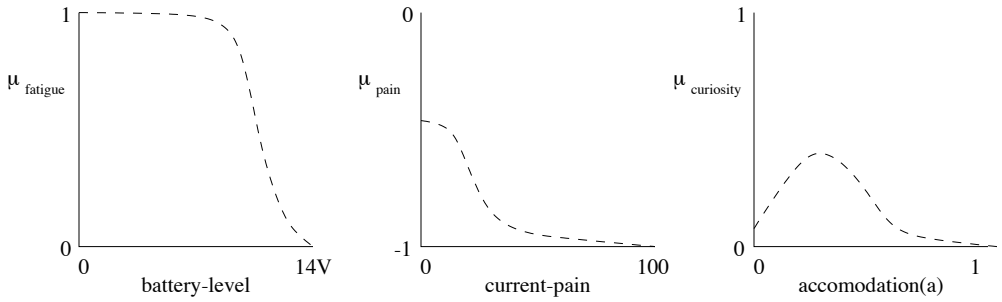

Figure 2: Coefficient functions for each of the Pioneer-2's primary motivational factors.

expected motivational payoff $E_\Delta(a_n, s_t)$ associated with it, and the product of the coefficient and payoff yields the factor value of F for a_n. The sum of these products over $F \in \mathcal{F}$ is the *desirability* of activity a_n in state s_t.

$$d(a_n, s_t) = \sum_{F \in \mathcal{F}} \mu_F(s_t) E_\Delta(a_n, s_t) \tag{1}$$

and

$$\psi(a_n, a_m, s_t) \iff (d(a_n, s_t) > d(a_m, s_t)) \tag{2}$$

It is worth noting that each factor value comprises an internal (perhaps genetic) part, $\mu_F(s_t)$, and a part contributed by the environment which must be learned, the quantity $E_\Delta(a_n, s_t)$. Figure 2 shows some possible coefficient functions for three sample motivational factors designed for the Pioneer-2 mobile robot. The coefficient function for fatigue depends on the robot's battery level, and is near zero when the battery is fully charged at 14V, indicating that fatigue plays no part in goal selection in the fully charged state. As the battery level drops, though, the coefficient rises, most dramatically in the range 10V-12V. Fatigue becomes an issue in this range, as the robot becomes unreliable when its voltage drops below around 11V.

The coefficient function for pain[1], in contrast, starts in the negative range, indicating that the possibility of pain reduces the desirability of an activity regardless of whether the robot has recently experienced pain, and when the robot has recently experienced pain, the inhibiting effect of pain avoidance only increases.

The curiosity coefficient depends on the activity under consideration, and a measure called *accommodation*. Accommodation measures the agent's familiarity with a particular activity. If the activity is new, then accommodation is low, and curiosity makes a positive contribution to that activity's desirability. As the agent exercises the activity, and builds better models of how the activity works, accommodation increases, and the contribution to desirability tapers off. The curiosity coefficient reacts to the novelty of an activity according to a bell shaped curve, in a manner consistent with infant accommodation studies [5]. Essentially, the curiosity factor asserts that all other things equal, an agent prefers to engage in activities it can learn most about.

[1]We use a simulated pain sensor for the Pioneer-2 mobile robot. Sudden contact with immovable objects produces surges of activity in the pain sensor.

2.3 How Curiosity Drives Learning

The two jobs of the motivational system are to keep the agent out of trouble and to provide learning opportunities for the domain modeling component. The self-preservation motivational factors like pain avoidance and fatigue handle the first job by influencing the preference relation when it is important, and allowing other factors to control ψ when attending to basic needs is unimportant. This is the aspect of activity that is sometimes referred to as *exploitation*, or using what you do know to your benefit.

Exploration, on the other hand, is activity performed to stimulate learning. Researchers in reinforcement learning realized early on that managing the exploration versus exploitation tradeoff was critically important. A popular approach to this tradeoff was to mix random activity with greedy activity, and slowly wean the agent off of exploration altogether under the assumption that learning would converge on optimal behavior, and exploration would no longer be necessary.

This approach to exploration is unfortunate for the following reasons: First, acting randomly does not guarantee that an agent will be led towards opportunities to learn, and second, exploration and learning are treated in isolation. There is no correlation between needing to learn and generating examples. There is simply a probability with which the agent will do something other than what its policy is telling it to do.

We posit that exploration can be, and should be, the result of exploitation done with models that need work, and that our motivational system, coupled with a means-ends analysis planner and a mechanism for learning operators, is an example of this principle in action.

A Simple Learning Domain

This is all best illustrated with an example. Consider a robot working in a parts preparation factory. The robot has a gripper and a paint gun, and a conveyor belt running in front of it, on which blocks pass by. The robot

may attempt to pick up what is in front of it, open its gripper to drop what it is holding, fire its paint gun at its gripper, rest, or activate a refill switch to refill its paint gun.

The robot may sense whether it is holding something, whether that thing has been painted, its current fatigue, and its current paint level. The *outcome* of each of its five actions is based on the four observable sensors; if the robot sprays its paint gun, it may paint a block, it may repaint a block, it may paint its own gripper, or it may be out of paint and shoot nothing at all, for example. The robot gets feedback as to which outcome actually unfolds.

This particular robot acts under the influence of 3 motivational factors, curiosity, fatigue, and *reward*, an exogenous signal which signals the robot when it has done something good (like paint and release a block) or bad (like attempt a refill when its paint gun is already full). Excessive fatigue may cause some actions to fail unexpectedly.

The high-level control system selects *outcomes* as goals based on their desirability, which is a composite rating based on novelty and expected reward. The low-level control system generates a plan which will end in the desired outcome. The execution module executes the action, and the modeling component records the feedback and updates its models of each outcome's preconditions and postconditions.

So, exactly how does exploitation equal exploration? An example occurs almost immediately for our factory robot. Suppose the agent, right after being turned on, executes the GRASP command. It picks up a block, and receives feedback that it has experienced a new outcome, which for this discussion, we will call `pick-up`. Immediately, this outcome will be the most highly rated because of its novelty, and the high-level control system will suggest that the robot try and reproduce it. Due to its poor (nonexistent) model of how `pick-up` works, the low-level control planner will suggest that activating the GRASP controller will work again. But, since the robot is already holding something, the GRASP controller experiences a different outcome, namely the `pu-fail` outcome, which comes about when the robot tries to grasp something when its gripper is full.

In the next section, we will see that examples of this type abound in this simple simulator, and that by about 60 invocations of the high-level control system, the robot has experienced all 19 of the outcomes of the 5 primitive actions. The governing principle is that planning to reproduce outcomes with flawed models produces flawed plans, which in turn produce unexpected results, which are opportunities to revise the flawed models.

3 Experiments

We have run two sets of experiments to show the behavior of an agent guided by our motivational system. The first is the factory robot described in section 2.3, and the second, on a general set of randomly generated Markov decision processes.

In both sets of experiments, we implemented a simple planner and learning schedule. The planner is a simple generate-and-test planner that uses the simulator to evaluate whether or not it thinks a plan will succeed. It is purposely allowed to generate illegal plans if it has not experienced outcomes that would allow it to verify that the plans were indeed illegal. An example of this allowable illegal plan would be the GRASP plan described in section 2.3. The agent does know about the `pick-up` outcome, and thus it can use it in plans, but since it has not experienced `pu-fail`, it cannot verify that the plan would fail.

Plots of desirability and outcome counts for the various activities of the factory robot domain are shown in figure 4. The results plotted are for a single run of the experiment, but results are similar across trials and different starting configurations. By 60 steps into the simulation, all 19 outcomes have been experienced by the simulated agent. By about 100 steps into the simulation, the effects of novelty for 16 of the 19 outcomes have dropped to levels sufficient to make them undesirable except as steps in a plan to achieve some other outcome. Shortly after, the novelty of two of the remaining three, `repaint` and `drop-incomplete-block` wears off, and `drop-complete-block` is clearly the most desirable outcome. The factory robot settles into an optimal strategy of executing plans that include alternately finishing blocks, resting, and refilling its paint gun. This is evidenced in the plot of outcome counts, in which only these three outcomes, and the required plan steps for achieving them continue to be executed.

Rather than hand-build increasingly sophisticated domains such as the block painting domain to show generality and scalability, we decided to apply our motivational system to randomly generated Markov decision processes. The algorithm we used to generate MDPs is as follows:

1. Generate a state variable with a user-specified number of states.
2. Generate a user-specified number of generic actions.
3. Break the states up into subcycles of random size.
4. Generate transitions within the subcycles.
5. Generate transitions between the subcycles such that there are no dead-ends or unreachable states in the process.
6. Generate an outcome for each transition and assign it randomly to an action.
7. Generate default outcomes that do not change state for each action which does not have an outcome for each state so that every action has an outcome in every state.
8. Assign "reward" values for each outcome. In our experiments, there is a 10% chance of an outcome having a negative reward in the range $[-0.5 \ldots 0]$, and a 10% chance of an outcome having a positive reward in the range $[0 \ldots 0.5]$.

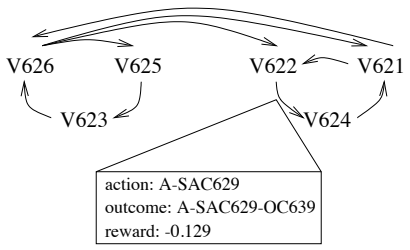

Figure 3: A sample, randomly generated MDP.

A small, randomly-generated MDP is shown in figure 3. Default outcomes are omitted for clarity, and a sample outcome label assigned to one of the transitions is shown, along with its reward value. In the final MDP, each transition is assigned such an outcome label. The only essential characteristic of these randomly generated MDPs is that each state must be reachable from any other state by a legal sequence of actions.

Figure 5 shows plots of desirability versus time and outcome counts versus time for a 5 action, 10 state, 22 outcome MDP. Outcome labels, which are randomly generated tokens like `A-SAC346-OC373`, have been omitted from the graph for clarity. As in the block painting domain, the 22 outcomes have all been discovered by step 100, and the most desirable outcome has distinguished itself by 200 steps into the simulation. Beyond that point, only six outcomes continue to be executed as plan components to exercising the outcome with the highest E_{Δ}^{reward}.

Figure 6 shows desirability and outcome counts versus time for a larger MDP with 10 actions, 20 states, and 44 outcomes. In this problem, our simple planner was bogged down with a very large search space of 44 operators, and due to time constraints, we cut the trials short after 210 steps. Still, the system managed to find 39 of the 44 possible outcomes after only 140 steps, and had become focused on a sequence of two rewarding outcomes shortly thereafter. Interestingly, as the graph of outcome counts shows, one of the highly rewarding outcomes had been discovered almost instantly. Over the course of exploring novel outcomes for the next 140 steps, the agent came across a new rewarding outcome. Soon after, the agent entered into a policy of alternatively planning for the original outcome and the new outcome, the two of which form a cycle, one leading into the other.

In the final version of this paper, we plan to run more replications to verify the generality of the algorithm, and expand to even larger problems with more states and outcomes. The current performance limitations of the system are due to the simplicity of our planner, which resorts in large part to brute force search in the planning space.

4 Conclusions

We have presented a system that makes decisions of *what to do* within a system that develops activities. Based around means-ends analysis planning, this high-level control component is simply an algorithm for goal selection. The goals it generates meet the following four criteria: the goals are achievable in the world, the goals serve to promote the self-preservation of the agent, the goals provide opportunities for the agent to learn about the possibilities of its world, and the goals produce *purposeful* behavior, where purposeful is taken to mean *done for a reason*. The first criterion is satisfied by adopting the philosophy of *planning to act*, or the idea that the goals of a system are to engage in activity, not achieve states.

The remaining three criteria are satisfied by our motivational system. This system is models the various drives that comprise motivation, including vegetative drives like hunger and fatigue, a curiosity drive that promotes exploration, and exogenous reward drives supplied by the environment. Each drive asserts influence on the agent's preference relation ψ such that it is attended to when it becomes important.

We demonstrated in two domains, a simulated block painting robot, and a class of randomly generated MDPs, that this motivational system, coupled with a planner and a modeling component that builds planning operators around observed outcomes, can manage the exploration versus exploitation tradeoff. This is done using a single, composite preference relation that takes into account the novelty of the activities the agent is considering engaging in.

In future work, we intend to transition this motivational system to our primary research platform, the Pioneer-2 mobile robot. Already implemented for this platform are a simple planner and a sytem for learning planning operators. The large array of real-valued sensors and continuous action space of the Pioneer-2 underlines the necessity of simple principles, such as planning to act, and our simple motivational system, that can reduce the dimensionality of the developmental task.

References

[1] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(2):189–208, 1971.

[2] George Lakoff and Mark Johnson. *Metaphors We Live By*. University of Chicago Press, 1980.

[3] Paul Cohen Matthew Schmill, Tim Oates. Learning planning operators in real-world, partially observable environments. In *Proceedings Fifth International Conference on Artificial Planning and Scheduling*, pages 246–253. AAAI Press, 2000.

[4] Allen Newell and H.A. Simon. GPS: A program that simulates human thought. In E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–293. McGraw-Hill, New York, 1963.

[5] H.A. Ruff and M.K. Rothbart. *Attention in early development: Themes and variations*. Oxford University Press, New York, 1996.

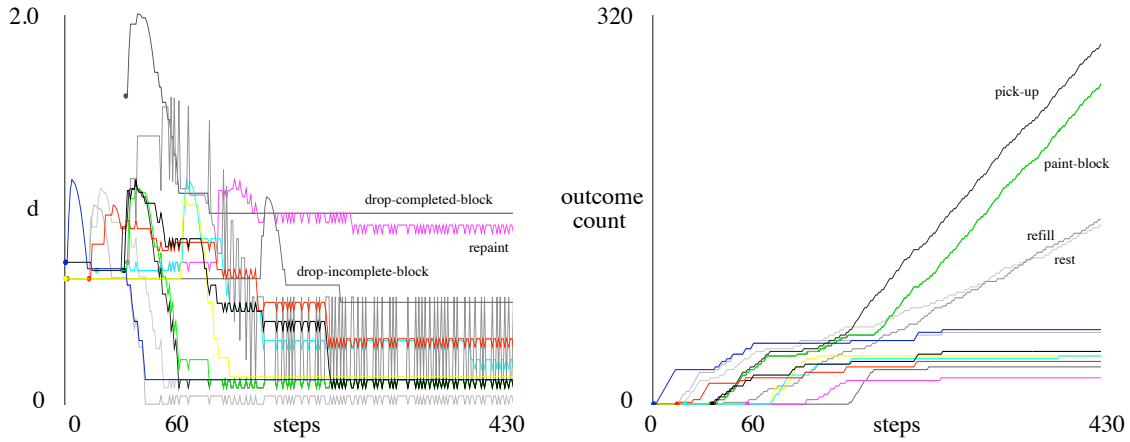

Figure 4: On the left, a plot of desirability levels versus time for the block paining robot domain. On the right, a plot of outcome counts versus time.

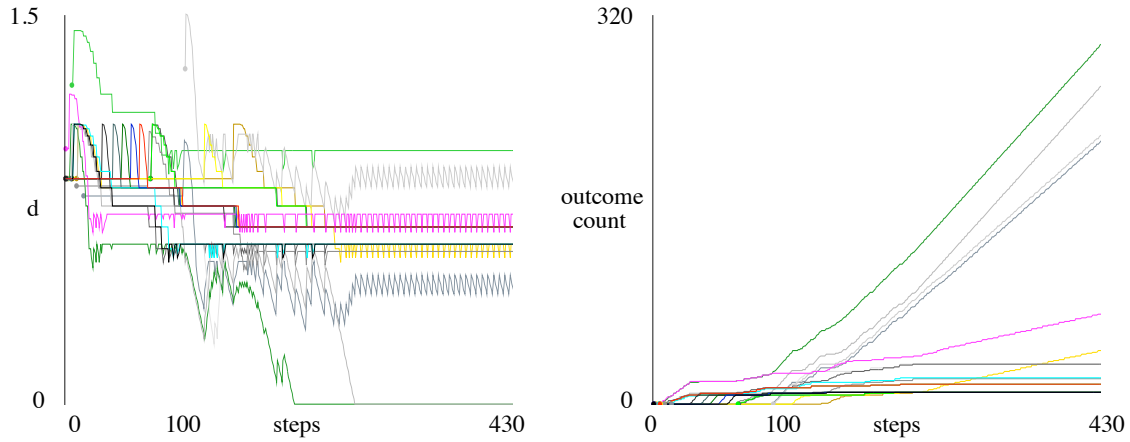

Figure 5: On the left, a plot of desirability levels versus time for a randomly generated Markov process domain with 5 actions, 22 outcomes, and 10 states. On the right, a plot of outcome counts versus time.

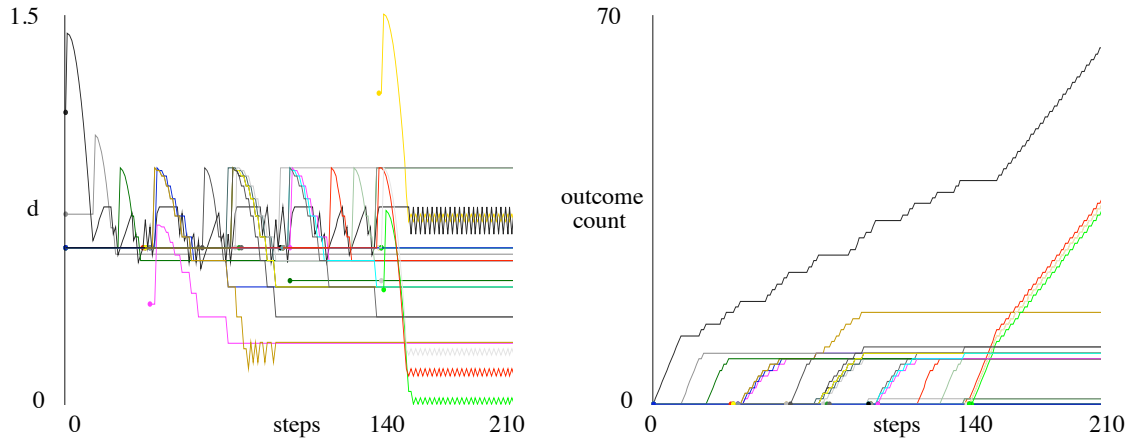

Figure 6: On the left, a plot of desirability levels versus time for a randomly generated Markov process domain with 10 actions, 44 outcomes, and 20 states. On the right, a plot of outcome counts versus time.