

U.S. NAVAL ACADEMY
COMPUTER SCIENCE DEPARTMENT
TECHNICAL REPORT



Robot Imitation Learning of High-Level Planning Information

Crabbe, Frederick L. Hwa, Rebecca

USNA-CS-TR-2005-03

May 2, 2005

Robot Imitation Learning of High-Level Planning Information

Frederick L. Crabbe
United States Naval Academy
Computer Science Department
572C Holloway Rd Stop 9F
Annapolis, Maryland 21402
crabbe@usna.edu

Rebecca Hwa
University of Pittsburgh
Computer Science Department
210 South Bouquet Street
Pittsburgh, PA 15213
hwa@cs.pitt.edu

May 2, 2005

Abstract

We present a system that enables a robot to learn to plan through demonstration and imitation. An imitator acquires planning operators by observing a demonstrator, segmenting the demonstrators actions into planning steps, and learning the preconditions and effects of the operators. When the imitator tries to execute its own plans, it learns to perform the operations through reinforcement learning, and corrects errors in the previously learned operator effects.

1 Introduction

Imitation learning promises to be a powerful method of training robots to behave in sophisticated ways. It enables fielded robots to develop new skills on their own without intervention. Imitation may take place in any area to which general machine learning can be applied. Currently, most robotic imitation learning systems concentrate on the mimicry of low level motor movements such as gestures[9, 8]. Imitation learning in the field of planning takes place at the planning level (e.g., sequences of operators) ignoring low level issues. Imitation learning at a high level that nonetheless takes in to account low level issues is an under-explored area, and the focus of this paper. We present a framework in which the robot learns through imitation both how to compose planning operators from low level robotic movements and how to apply those operators. Our system addresses some shortcomings of current operator learning systems, and uses a robust mechanism for learning low level behavior that does not require mimicry of individual movements.

In section 2 we present the basic control architecture that drives the robot. Section 3 presents the learning algorithms and discusses how they fit in the framework. Section 4 describes experiments testing the abilities of the learning algorithms, and finally we conclude with future work.

2 Control Architecture

The control architecture of the imitator has two layers (figure 1). The lower layer consists of a collection of reactive modules responsible for low level behavior. Individual modules are neural networks or parameterized linear or bilinear functions. The upper layer is the planning system. This layer first determines which planing conditions (predicates) are currently true. When the robot is executing a plan, the planning layer uses the conditions to determine where it is in the plan and what operator should be applied. Each planning operator is associated with a reactive module in the reactive layer. Once the operator is selected, its corresponding reactive module is activated, which then uses input to directly generate output for the robot.

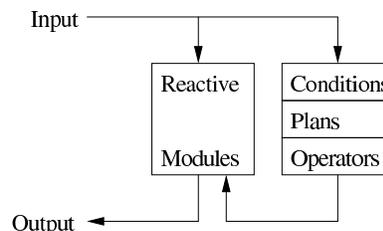


Figure 1: The control architecture.

3 Learning Method

Learning in the system takes place at both the reactive and planning layer. The system begins with a set of conditions it can recognize¹ and attempts to learn the appropriate planning operators for the domain. Learning the planning operators means learning the preconditions and effects of the operator, as well as learning the parameters in the reactive module so that it gets the desired effects. The basic process of learning in the system involves six steps:

1. The imitator observes the the demonstrator’s (low level) actions and divides the action stream into segments that correspond to steps in a plan. Each step will eventually be mapped to a planning operator.
2. For each segment, a reactive module is created in the reactive layer. This module will learn to perform the operation corresponding to that segment. The module is “primed” in order to facilitate learning (see §3.2).
3. For each segment, an operator is created in the planning layer, and initial guesses are made at the preconditions and effects of the operator.
4. The imitator stops observing the demonstrator, and is given a goal to complete. The imitator generates a plan based on its current set of operators.
5. For each reactive system selected by an operator in the plan, the robot learns to perform the operator via reinforcement learning.
6. If the plan fails, the operators are refined by adjusting the preconditions and effects of the operators, and a new plan is generated. Then the imitator continues with step 5, learning the parameters of the reactive modules for the new operators in the plan. Steps 5 and 6 are repeated until the plan succeeds.

3.1 Segmentation

In a robotic system, a plan operator (the basic unit of a plan) may consist of many low level actions. Therefore, in an imitation learning scenario, the first step is to segment the stream of input data describing the demonstrator’s actions into chunks corresponding to the basic planning units. Most of such segmentation is currently done by hand [4] or using heuristics based on properties of the motors, such as when joint velocities cross zero[7]. However, a single planning unit may take place over the course of several minutes and include thousands of zero velocity crossings (e.g., ‘**fetching the Illudium Q-36 Space Modulator**’). We want to segment the data only after the achievement of the goal. We are developing two automatic segmentation methods for our problem. The first relies on the demonstrator having the plan steps in mind and indicating the end of steps with an explicit signal. The second method relies on statistical properties of continuous and repetitive actions to induce the points of discontinuity in the stream.

3.1.1 Demonstrator Signal Segmenting

In this case, the demonstrator signals positively when a step is completed, or negatively during a step if the demonstrator is in a situation to be avoided. Demonstrator signal segmenting has the advantage that it can be 100% reliable. As long as the demonstrator has a correct model of the plan, it can correctly convey it to the imitator. It also has the advantage that negative signals emitted by the demonstrator can improve the performance of the imitator. If the demonstrator finds itself in a state that is dangerous or potentially damaging to the step, it can emit a negative signal to indicate this. Demonstrator signal segmenting has the disadvantage that it requires the cooperation of the demonstrator, which may not always be available.

¹Directly learning the descriptives from raw sensor data is an interesting and challenging problem that we want to address in the future.

3.1.2 Induced Segmentation

Without the cooperation or expertise of the demonstrator, segments may still be automatically induced from the demonstrator’s action stream, if it contains multiple executions of the planning operators that the imitator is trying to learn. Repetitions often occur naturally in plans. For example, in a construction plan, the demonstrator may execute ‘pick up building block’ and ‘place building block on top of another block’ multiple times. In this light, these primitives repeat in a manner similar to common linguistic entities in natural language. We are currently investigating several techniques from natural language processing for this purpose. As a baseline, we have implemented a Japanese word segmentation algorithm [1].² Based on the observation that certain words frequently co-occur as a sequence, Ando and Lee determined whether to add a segment break between words based on a metric called *cohesion factor*. A group of words is considered highly cohesive if they often occur together and rarely occur with other words. Thus, for each possible segmentation point, they compare the cohesion factor of a n -word window before and after the point with the cohesion factors of all the n -word windows that cross that point. If the sequence windows on either side of the point are significantly more cohesive than the sequences crossing the point, then the algorithm determines that a segment ends at that point. In our experiments, the cohesion measure was the average distance of the vectors of the window from the centroid of the window. We used a window of four samples.

3.2 Vicarious Learning and Priming

Once the input is segmented, a Q-learning [11] module is created in the reactive layer for each segment. The module is primed using a technique from Vicarious Learning (or VL – described in [5]). VL is defined as learning by an agent from another agent by observing both the actions of and rewards to the other agent. The agents signal their pleasure and displeasure with their status in the environment when events occur, and the learners perceive these signals. If the system is using demonstrator signaled segmenting, the signal from the demonstrator (positive or negative) is used as a vicarious signal. If the system is using another segmenting system, the end of a segment is considered a positive signal. When the imitator detects a vicarious signal, the eligibility trace of the active vicarious Q-learning subsystem is set with the input state of the model agent, and a reinforcement signal matching the vicarious signal is fed to the system, modifying the weight vector. The advantages to using vicarious learning for the operators are: by concentrating on the final goal of the step, the imitator can learn well from an inefficient demonstrator; by ignoring the low-level motions, imitators can learn from demonstrators with different morphologies, and negative information can be conveyed to the imitator.

3.3 Planning Operators

Once a segment is identified and the reinforcement learning system primed, the imitator guesses at the demonstrator’s goal in performing each segment. This corresponds to identifying the preconditions and effects of each segment. To do so, the imitator must start with a reasonable set of logical descriptives for the domain and the ability to identify when they apply. Whenever a segment begins, the planning system records the current state of the demonstrator. When the end of the segment is identified, the state of the demonstrator is examined again. We use a system similar to Wang[10]. The world is described with STRIPS style operators[6]; preconditions are hypothesized based on the world state at the start of the segment; and effects are hypothesized based the world state at the at the end. There are several significant differences between Wang’s idealized planning world and that of the robots we considered. Wang assumes: a) that the learner has full access to the world state and b) that the operator can always be identified. Our robots do not have this information, therefore they generally will not create operators with the appropriate level of generality. For example, in Wang’s system, if the demonstrator uses the operator `hold-with-vice` multiple times, each time in a slightly different world state, the imitator can recognize the operator by name and deduce the preconditions and effects of the operator (i.e., the common predicates between the instances). In our case, the imitator does not have a prescribed set of operators a priori, therefore it cannot identify two instances of the same operator if the observed preconditions and effects are slightly different. This makes the refinement process less reliable. Moreover, since many robots do not have a global view of the world, it is hard to distinguish the effects of an operator from conditions that become observable. Thus, the effects of an operator are not always directly inferable

²Word segmentation is often necessary for Asian languages because the text does not contain word delimiters. An English equivalent of the word segmentation problem is to determine word boundaries from an English text in which all the white spaces have been removed.

from the end state. Without full knowledge of the world state, the robot has difficulties with recognizing common operators and with correctly determining the effects of operators. We partially address this by using a relatively small radius in which the imitator recognizes conditions to be true. After the operator is created, we do not attempt to refine the operator further at this stage; instead, a new operator is generated for each segment with a different set of preconditions and effects. Some operator refinement is possible in the Plan Refinement stage (see §3.6). Furthermore, Wang uses the *difference* between the world state before and after the operator was applied to deduce effects. Because our robots do not always correctly identify the preconditions and effects of an operator, the difference approach is not applicable. For example, if a mobile robot observed a demonstrator pick up a green object and then move to a blue object, the imitator might infer a **Go-Blue** operator with preconditions **Holding-Green** and effects **Holding-Green, Nextto-Blue**. The imitator cannot leave **Holding-Green** off the preconditions list because it does not know that it is unnecessary for the operation. In Wang’s system, this precondition would be refined away through resolving multiple instances of **Go-Blue**. In our system, the operator will always have the precondition, and therefore, **Holding-Green** will also always be true after the operator’s application. The result of these differences with Wang is that the imitator ends up with many similar operators that vary only slightly in preconditions and effects.

3.4 Plan Generation

Once the imitator has a set of operators, it can begin to formulate plans to solve problems in the environment. The system is designed so that the initial plan can be generated by any standard STRIPS planner. We generate plans using a state-space regression planner with a depth first iterative deepening search.

3.5 Operator Learning

Once a plan has been formulated, the imitator can begin executing the plan, but does not know how to execute the steps. When it attempts to perform a step, the imitator action the output of the reactive system associated with that operator. Recall that the weight vector for the system was primed in the Priming step. If the imitator uses a generalization function (such as a neural network or weighted linear function) and the space is smooth, then this can draw the observer toward the model’s state, as shown in [2, 5]. Once the imitator’s input is similar to the demonstrator’s when it emitted the positive signal, the similarity in states generates another positive reinforcement to the imitator, solidifying the skill in the system.

3.6 Plan Refinement

Since the unrefined preconditions and effects associated with each operator are likely to contain some errors, the plans as executed by the imitator may fail. For example, if a plan depends upon what the imitator erroneously believes to be an effect of an operator, the plan may fail when the imitator tries to apply the operator. It is important to detect the failure, repair the operator and refine the plan. Currently, we detect the failure with a timeout, similar to [3], except that the timeout value is decreased whenever an expected effect of the operator becomes true. Steps in a plan typically end once the desired effects become true. If the imitator is performing a step and most but not all the effects become true, this is a clue that the effects that did not become true should not be part of the operator.

Once a failure is detected, the system refines the plan, and then possibly repairs the current operator. The plan is repaired by backtracking to the deepest choice point in the search tree. We do not assume that the effects of the operator are wrong, because the failure may be due to a precondition for the operator that the imitator was unable to see when observing the demonstrator. Instead, the planner tries a new plan before the current operator to find the precondition that was necessary for the operator.

If there are no backtrack points below the current operator, the effects of the operator are assumed to be in error. The operator is repaired by removing the effect that failed to become true. A copy of the original operator is placed in a deprecated set as a last resort when no other plan can be found.

4 Experiments

We tested the system in a Khepera robot simulator with construction tasks. In order to smooth noise and account for occluded objects, input to the system was from a virtual sensor that read from a map

built by the robots. In this section we will present experiments which test in isolation the segmentation and the learning of individual operators based on the priming. We will also test the ability of the entire system to learn to imitate a plan and to generate new correct plans based on the learned operators.

4.1 Segmentation

Because the demonstrator signal segmenting system is always accurate, we test only the induced segmenter. We used our version of Ando and Lee (see §3.1.2) on three data sets. The first was gathered from the Khepera simulator of a robot performing a construction task. The input was a four dimensional vector sampled every tenth of a second. The first two values were the left and right wheel speeds. The third and fourth values were the arm and finger speeds. The second data set was gathered from a collection of Robix Arm programs written to perform a variety of simple manipulation tasks. The input was a twelve dimensional vector sampled every tenth of a second. The values were the rotational velocities of the individual servomotors. The third set was gathered from a 38 foot sailing vessel on a twenty mile race. The sailboat was human controlled, and the actions of the crew were sampled every thirty seconds. The individual values indicated course changes and adjustment to the sails. These data were hand labeled to indicate the end of segments. Figure 2 shows the results of running each algorithm on each dataset. We report both the precision and recall of the predicted segments for each algorithm, for both an exact match to the labeled segments, as well as a match within two samples either side. We see that with the Robix arm, both precision and recall were very high. This was because the velocities of the motors tended to be constant within a particular step. For the other two datasets, recall was still high but precision was very low. The segmenter was too sensitive to small similarities in the data and was overly aggressive in proposing segments.

Data		Precision	Recall
Arm	exact	96.8	83.6
	+/- 2	100	86.3
Khepera	exact	1.0	95.2
	(+/- 2)	1.0	100
Sailboat	exact	4.0	23.3
	(+/- 2)	13.8	80.0

Figure 2: Results of the segmentation experiments. The values were the rotational velocities of the individual servomotors. The third set was gathered from a 38 foot sailing vessel on a twenty mile race. The sailboat was human controlled, and the actions of the crew were sampled every thirty seconds. The individual values indicated course changes and adjustment to the sails. These data were hand labeled to indicate the end of segments. Figure 2 shows the results of running each algorithm on each dataset. We report both the precision and recall of the predicted segments for each algorithm, for both an exact match to the labeled segments, as well as a match within two samples either side. We see that with the Robix arm, both precision and recall were very high. This was because the velocities of the motors tended to be constant within a particular step. For the other two datasets, recall was still high but precision was very low. The segmenter was too sensitive to small similarities in the data and was overly aggressive in proposing segments.

4.2 Operator Learning

Once the data is segmented, the imitator can use the state at the end of the segment to prime the Q-learning mechanism. Because the vicarious segmenting system is currently more reliable than the induced segmenter, we used it in our experiments testing the learning of the individual operators. We measured how viewing a vicarious event can directly affect the speed of learning to perform a task. We also measured how effective a negative signal from the demonstrator is in keeping the imitator out of a state. In the experiments, Watkins Q-learning with replacing traces was used[11]. The Q-learning constants were set at: $\alpha = 0.15$, $\lambda = 0.2$, and $\gamma = 0.8$. The input was a six dimensional vector where the values of each dimension indicated the presence of a particular colored object in a particular direction. The individual dimensions were: green object to the left; green object front; green object right; red object left; red front; and red right.

Figure 3 shows the learning curve of robots learning to perform the task of moving to a green object in a four foot by four foot environment with five differently colored objects. Touching a green object generates positive reinforcement, and touching a red object generates negative. All data points are the average of ten runs. The baseline learns the task purely through

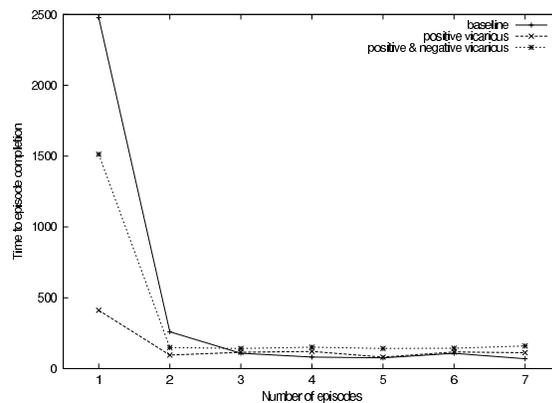


Figure 3: Learning curves of Vicarious Learning.

	No negative signal	with negative signal
Avg. distance	33.83	84.30
Number touches	3.2	1.6

Figure 4: Results of vicarious learning with negative signals. Distance is measured in cm.

reinforcement learning. In the other two curves, the robots have been primed with vicarious learning. The bottom curve shows that an agent primed with just a positive vicarious signal is excellent at the task on the first episode, and has converged by the second. The middle curve shows results for robots primed with both a positive vicarious signal (touching green objects) and negative vicarious signal (touching red objects). The curve shows that in this situation, the robot acquires the skill more slowly than one primed with just a positive signal, but still much faster than the baseline agent.

Figure 4 describes how well the negative vicarious signal keeps the imitator away from the red objects. In comparing robots that have just a positive vicarious signal with those that have had both positive and negative, it shows over the course of seven episodes, the average distance the two kinds of agents keep from the negative (red) object, and the number of times each type came into contact with the red object. Although the negative signal slows the learning speed, it does enable the agent to better avoid the negative state.

4.3 Multi-step Sequence

The next experiment tests the ability of the system to put multiple planning steps together in a sequence. In the experiment the demonstrator performed a multi-step plan, and the imitator learned to perform the same steps in that order. Because the steps need only be memorized by the imitator, rather than generated by the planner, it is unnecessary to refine the plans or improve the operators. The construction task in this experiment is wall building: the agents must go to a piece of building material, pick it up, bring it to the wall, move to the correct place at the end of the wall, and drop the material.

The demonstrator and imitator were placed in the environment with the two marker objects and a collection of randomly placed building objects. The imitator follows and observes the demonstrator performing the wall building through one cycle, from finding a building object to placing it on the wall. After one cycle, the demonstrator was removed and the imitator left to learn on its own. As it explored the environment, it learned to perform the steps of the plan sequence, and then repeated the sequence to build the wall.

Figure 5 shows the walls built by the demonstrator and imitator. The left wall was built by the demonstrator using a hand coded plan and reactive modules. The right wall was built entirely by the learner. Each case was run 10 times and the length of time taken to build the wall was recorded. The average time for the demonstrator was 1015.2 seconds, and the average time for the imitator was 1596.9 seconds.

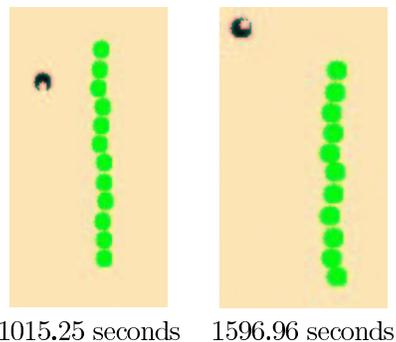


Figure 5: Completed walls built by the teacher alone, and the learner alone.

4.4 Full Planning System

We create a planning test scenario to verify that the imitator can generate plans with acquired operators, and simultaneously learn to perform the operators, detect plan failures, and repair the plan. In the scenario, the

Condition	Description
Near-X	Robot within fifteen cm of an object colored X.
Nextto-X	Robot close enough to an X-colored object to pick it up.
Facing-X	Robot facing an X-colored object.
Near-XY-Collection	Robot near a group of adjacent objects colored X & Y.
Clear-X	X-colored object free to be picked up.

Figure 6: The conditions detectable by the imitator.

imitator has a goal to find a green object and move it next to a blue object. There are two demonstrators. The first demonstrates the basic process of moving the green object to the blue. The second demonstrator shows that when the green object is blocked by a red object, moving the red object will

Demonstrator's Operator name	Imitator's Operator name	Preconditions	Effects
Demonstrator One			
Go-Green	Op0	<none>	Nextto-Green Facing-Green Clear-Green
Pickup-Green	Op1	Nextto-Green Facing-Green Clear-Green	Holding-Green
Go-Blue	Op2	Holding-Green	Holding-Green Near-Blue Clear-Blue
Drop-Green	Op3	Holding-Green Near-Blue Clear-Blue	Nextto-Green Facing-Green Clear-Green Clear-Blue Near-GreenBlue-Collection
Demonstrator Two			
Go-Red	Op4	<none>	Near-Green Nextto-Red Facing-Red Clear-Red Near-GreenRed-Collection
Pickup-Red	Op5	Near-Green Nextto-Red Facing-Red Clear-Red Near-GreenRed-Collection	Near-Green Holding-Red Clear-Green
Pull-Away	Op6	Near-Green Holding-Red Clear-Green	Holding-Red
Drop-Red	Op7	Holding-Red	Nextto-Red Facing-Red Clear-Red
Go-Green	Op8	Nextto-Red Facing-Red	Nextto-Green Facing-Green Clear-Green

Figure 7: The two demonstrators' plans. The first column is the name of the operator as used by the demonstrator. The second column is the name the imitator gives to the operator. The last two columns are the preconditions and effects of the operators.

unblock the green. When the imitator is given the goal to move the green object next to the blue, the green object is blocked by the red. The imitator needs to create a plan built from operators learned from both demonstrators to solve the task.

The initial set of conditions are described in figure 6. The imitator was given the ability to recognize all of the conditions at the start of the experiment. The plans that the demonstrators followed are shown in figure 7, with the preconditions and effects inferred by the imitator. When the imitator is tested, it initially generates the plan shown in figure 8, left. This plan is identical to the plan performed by the first demonstrator. The imitator learns in the reactive module to approach the green object, and eventually arrives, but of the four expected effects, only three are true, **Near-Green**, **Nextto-Green**, and **Facing-Green**. The imitator generates a new plan (figure 8, middle) and deletes **Clear-Green** from the operator **op0**. In the new plan, the imitator tries to apply **op8** which requires that the imitator start next to a red object, so the first step is **op4**. Once the imitator successfully applies **op4**, it tries to reapply **op8**. This again fails because the object is not clear. The imitator finally generates the plan shown in figure 8, right. This plan is sufficient to complete the task. In 10 runs of the imitator in the planning phase, the plan was successfully completed in 3496.3 seconds.

5 Conclusion and Future Work

We have described a system that learns high level information about a robotic planning domain using both high and low level information. It addresses several issues not accounted for in other operator learning systems. The use of vicarious learning to learn motor skills enables the robots to gain low level skills by imitation without direct mimicry of motion. There are many areas that we are interested in pursuing in the future. We will: complete the design and testing of the induced segmenter; create mechanisms for detecting conditions from the raw sensor data and automatically discovering new conditions for use in the planning operators; develop mechanisms for merging multiple operators into one; develop a specialized planner to take advantage the special properties of the operators; explore the planning step failure detection; and develop a large test suite of planning problems across multiple robot platforms.

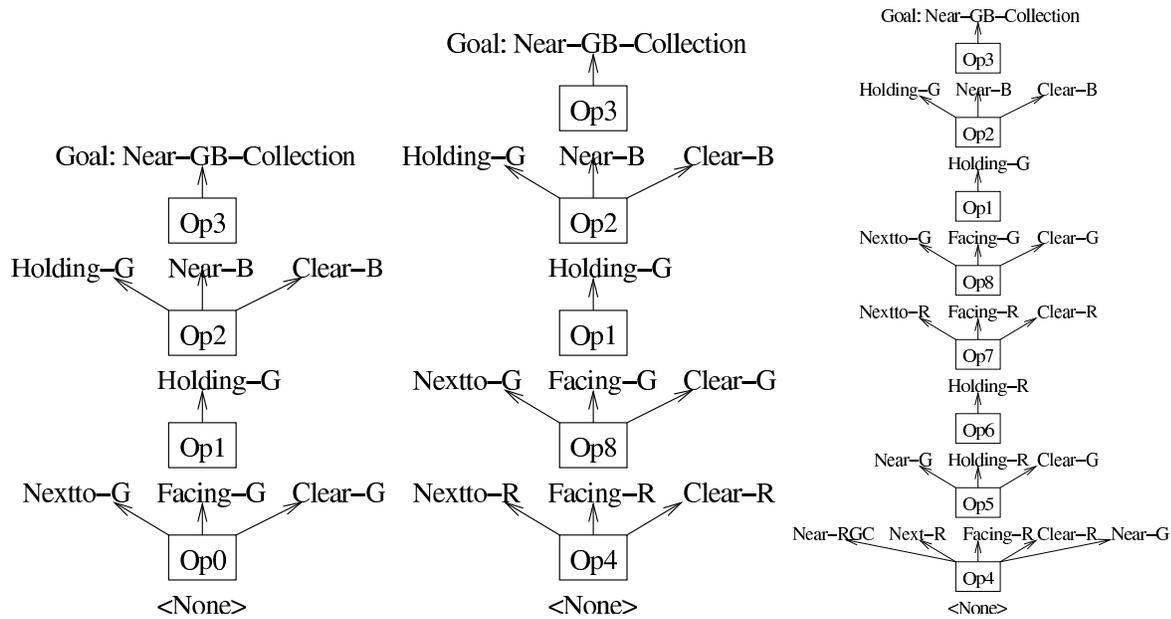


Figure 8: The imitator’s three plans. Boxes are the operators and arrows point to the effects of operators that satisfy the preconditions of the next operator. Colors are abbreviated to their first letter.

References

- [1] R. K. Ando and L. Lee. Mostly-unsupervised statistical segmentation of japanese: Applications to kanji. In *First Conference of the NAACL*, 2000.
- [2] C. Atkeson and S. Schaal. Robot learning from demonstration. In *Machine Learning: Proceedings of the Fourteenth International Conference (ICML '97)*, pages 12–20, 1997.
- [3] S. Benson. Inductive learning of reactive action models. In *International Conference on Machine Learning (ICML)*, pages 47–54, 1995.
- [4] D. Bentivegna and C. Atkeson. Learning how to behave from observing others. In *SAB’02-Workshop on Motor Control in Humans and Robots: on the interplay of real brains and artificial devices*, 2002.
- [5] F. L. Crabbe and M. G. Dyer. Goal directed adaptive behavior in second-order neural networks: The maxson family of architectures. *Adaptive Behavior*, 8(2), 2001.
- [6] R. E. Fikes and N. J. Nilsson. Strips: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.
- [7] A. Fod, M. Matarić, and O. Jenkins. Automated derivation of primitives for movement classification. *Autonomous Robots*, 12(1):39–54, 2002.
- [8] T Inamura, I. Toshima, and Y. Nakamura. Acquisition and embodiment of motion elements in close mimesis loop. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1539–1544, 2002.
- [9] M. J. Matarić. Learning to behave socially. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, 1994.
- [10] X. Wang. Learning by observation and practice: An incremental approach for planning operator acquisition. In *International Conference on Machine Learning*, pages 549–557, 1995.
- [11] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989.