Imitation & Reinforcement Learning

in Virtually Embodied Agents Using Program Evolution

Nil Geisweiller

Novamente LLC

Xiamen University
AGI Summer School 2009

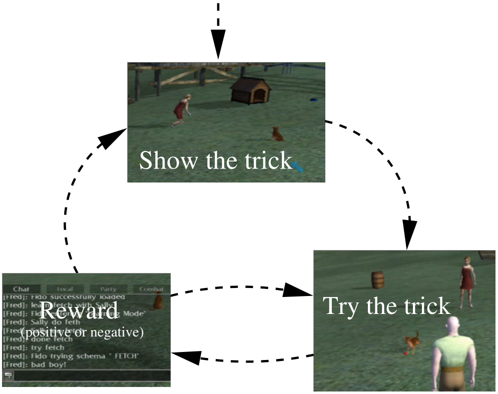
Outline

Introduction

Imitation & Reinforcement Learning

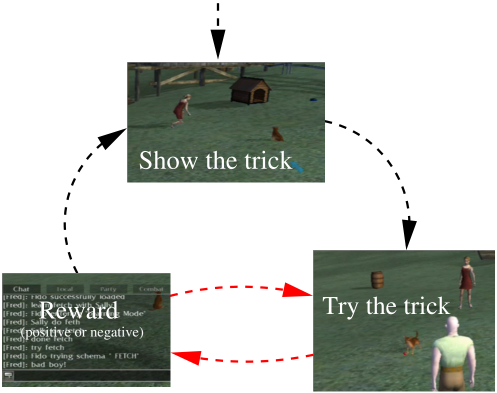
A way to communicate procedural knowledge without programming or sophisticated NLP

Imitation Reinforcement Loop in Embodiment



1. How to find rapidely a trick that fits?

Imitation Reinforcement Loop in Embodiment



1. How to find rapidely a trick that fits?
2. How to take reward into account to converge faster?

Outline

Outline

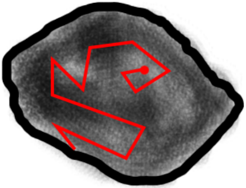
1. Introduction
2. Searching the Space of Trick Candidates
   - Overview
   - Accelerating Search
3. Taking Reward into Account
4. Conclusion

Searching the Space of Trick Candidates: Recall

1. Operational Agent Controller (OPC) provides the episodic memory of the learning session to HillClimbing (or MOSES)

Searching the Space of Trick Candidates: Recall

1. Operational Agent Controller (OPC) provides the episodic memory of the learning session to HillClimbing (or MOSES)
2. Which searches the program space to find one that fits (mimics avatar's behavior)



The pet replaies mentally the scene, but substitues the avatar to imitate by itself

$\Rightarrow$

Fitness Function

Measure how the program candidate's behavior fits the avatar's (compare their sequence of actions)

Operators involved to build program candidates

- `sequential_and`
- `action_boolean_if`
- `action_action_if`
- `action_while`
- `boolean_while`
- `action_not`
- `logical_not`
- `random_object`
- `nearest_object`

+

- Potential perceptions, `near(obj_1 obj_2)`, `is_moving(obj_3)`, etc.
- Potential actions, `grab(obj_1)`, `goto_obj(avatar_2)`, etc.

Operators involved to build program candidates

- `sequential_and`
- `action_boolean_if`
- `action_action_if`
- `action_while`
- `boolean_while`
- `action_not`
- `logical_not`
- `random_object`
- `nearest_object`

+

- Potential perceptions, `near(obj_1 obj_2)`, `is_moving(obj_3)`, etc.
- Potential actions, `grab(obj_1)`, `goto_obj(avatar_2)`, etc.

Example of Tricks in Combo

1. Fetch a random object

```
and_seq(goto_obj(random_object)
        grab(nearest_object)
        goto_obj(owner)
        drop)
```

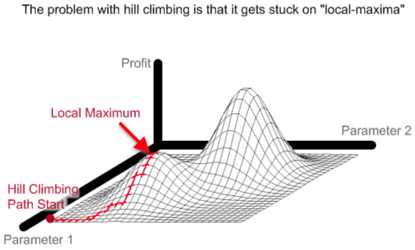
2. kicks 3 times, from the left leg if stick is near ball and from the right leg otherwise

```
action_if(near(stick ball)
          and_seq(kickLeft kickLeft kickLeft)
          and_seq(kickRight kickRight kickRight))
```

3. Dance on cue until the owner says “stop dancing”

```
boolean_while(not(has_said(owner "stop dancing"))
              action_if(is_last_action(owner "kickL")
                        tap_dance
                        lean_rock_dance))
```

HillClimbing Search Algo



HillClimbing + restart on the best non yet restarted candidate

Outline

Accelerating HillClimbing Search

1. Reduction in normal form to avoid over representation

Accelerating HillClimbing Search

1. Reduction in normal form to avoid over representation
2. Filtering Perceptions, entropy threshold

Accelerating HillClimbing Search

1. Reduction in normal form to avoid over representation
2. Filtering Perceptions, entropy threshold
3. Building-blocks of action sequences

Accelerating HillClimbing Search

1. Reduction in normal form to avoid over representation
2. Filtering Perceptions, entropy threshold
3. Building-blocks of action sequences
4. Setting carefully Occam's razor function

Filtering Perception, Entropy Threshold

$$c < -\sum_{i=1,2} p_i \times log_2(p_i)$$

Example, entropy of `near(chair1 chair2)` is null

Building-blocks of action sequences

Example with fetch:

1. ```
   and_seq(goto_obj(random_object)
           grab(nearest_object)
           goto_obj(owner)
           drop)
   ```
2. ```
   and_seq(goto_obj(nearest_object)
           grab(nearest_object)
           goto_obj(owner)
           drop)
   ```
3. ```
   and_seq(goto_obj(ball)
           grab(ball)
           goto_obj(owner)
           drop)
   ```
4. ...

It may be faster to start from the sequence itself rather than an empty program.
   ```

Setting carefully Occam's razor function

- Problem, when the sequence is too long it easily generates over-complicated candidates

Setting carefully Occam's razor function

- Problem, when the sequence is too long it easily generates over-complicated candidates
- Solution, strong bias toward simple candidates first even if they fit less.

Setting carefully Occam's razor function

- Problem, when the sequence is too long it easily generates over-complicated candidates
- Solution, strong bias toward simple candidates first even if they fit less.
- Automatically tuning $sizePenalty_{a,b}$ based on the past learning experiences

$$sizePenalty_{a,b}(p) = exp(-a \times log(b \times |A| + exp(1)) \times |p|)$$

Some benchmarks

Reduct	ActSeq	Entropy	Occam	Setting
On	Off	0.1	0.03	$conf_1$
Off	Off	0.1	0.03	$conf_3$
On	Off	0	0.03	$conf_4$
On	Off	0.1	0.3	$conf_7$
On	On	0.1	0.03	$conf_9$
On	On	0.1	0.025	$conf_{10}$

Table: Settings for each learning experiment

Setting	Eval	Time
$conf_1$	653	5s18
$conf_3$	1073	8s42
$conf_4$	28287	4mn7s
$conf_7$	3121	23s42
$conf_9$	89	410ms
$conf_{10}$	33	161ms

Table: fetch_ball

Setting	Eval	Time
$conf_1$	2783	21s47
$conf_3$	15069	2mn15s
$conf_4$	∞	∞
$conf_7$	$>$200K	$>$1h
$conf_9$	107	146ms
$conf_{10}$	101	164ms

Table: triple_kick

Setting	Eval	Time
$conf_1$	113	4s
$conf_3$	150	6s20ms
$conf_4$	$>$60K	$>$1h
$conf_7$	113	4s
$conf_9$	138	4s191ms
$conf_{10}$	219K	56mn3s

Table: double_dance

Example of Tricks in Combo

1. fetch_ball

```
and_seq(goto_obj(ball)
        grab(ball)
        goto_obj(owner)
        drop)
```

2. triple_kick

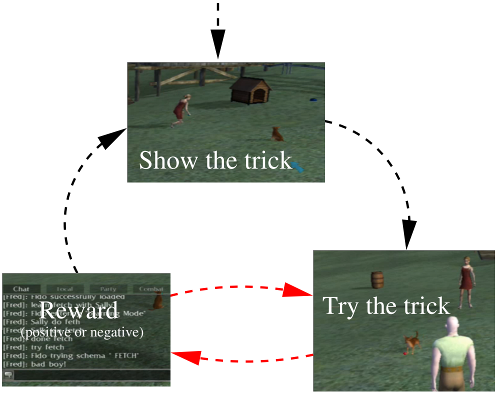
```
action_if(near(stick ball)
          and_seq(kickLeft kickLeft kickLeft)
          and_seq(kickRight kickRight kickRight))
```

3. double_dance

```
boolean_while(not(has_said(owner "stop dancing"))
              action_if(is_last_action(owner "kickL")
                        tap_dance
                        lean_rock_dance))
```

Outline

Taking Reward into Account to Converge Faster (Not implemented yet)



Taking Reward into Account to Converge Faster (Not implemented yet)

Main idea

Use the pet's trial as new exemplar weighted by owner's reward

Fitness $\longleftarrow$

- new episodes taken into account
- candidate to be compared to be similar to a good trials or dissimilar to a bad trials.

Outline

Conclusion

Pretty fast on simple tricks but...

What remains to be done:

- Improve Mental image of the scene to be more accurate (action consequence, collisions)

Conclusion

Pretty fast on simple tricks but...

What remains to be done:

- Improve Mental image of the scene to be more accurate (action consequence, collisions)
- Take into account that pet is not human (co-evolution)

Conclusion

Pretty fast on simple tricks but...

What remains to be done:

- Improve Mental image of the scene to be more accurate (action consequence, collisions)
- Take into account that pet is not human (co-evolution)
- Implement owner reward feedback for faster convergence

Conclusion

Pretty fast on simple tricks but...

What remains to be done:

- Improve Mental image of the scene to be more accurate (action consequence, collisions)
- Take into account that pet is not human (co-evolution)
- Implement owner reward feedback for faster convergence
- Improving Filters by using Attention Allocation

Conclusion

Pretty fast on simple tricks but...

What remains to be done:

- Improve Mental image of the scene to be more accurate (action consequence, collisions)
- Take into account that pet is not human (co-evolution)
- Implement owner reward feedback for faster convergence
- Improving Filters by using Attention Allocation
- Extend SizePenalty Bias to all parameters of the search algo (distribution priors, etc), Transfer Learning