Program Guided Agent

ICLR 2020 (Spotlight)

```
Program

def run():
    if is_there[River]:
        mine(Wood)
        build_bridge()
        if agent[Iron]<3:
            mine(Iron)
            place(Iron, 1, 1)
        else:
            goto(4, 2)
        while env[Gold]>0:
            mine(Gold)
```

Shao-Hua Sun
Te-Lin Wu
Joseph J. Lim
Follow an Instruction to Solve a Complex Task

Recipe: cooking fried rice
Stir-fry the onions until tender, and repeat this for garlic and carrots, if you have soy sauce, add some. Pour 2/3 cups the whisked eggs into the stir-fried and scramble.
Recipe: cooking fried rice

Stir-fry the onions until tender, and repeat this for garlic and carrots, if you have soy sauce, add some. Pour \(\frac{2}{3}\) cups the whisked eggs into the stir-fried and scramble.

Ambiguities in Language

- Scoping
- Coreferences
- Entities

Bandanau et al. in ICLR 2019
Misra et al. “Mapping Instructions to Actions in 3D Environments with Visual Goal Prediction” in EMNLP 2018
Hermann et al. “Grounded Language Learning in a Simulated 3D World” in arXiv 2017
Program

Function: cooking fried rice

for item in [onions, garlic, carrots]:
    if is_there("soy sauce"):
        add("soy sauce", "pot")
    while not tender(item):
        stir_fry(item)
pour(whisked("eggs"), "pot", 0.66)
scramble("eggs")

Advantages of Programs

- Explicit scoping
- Resolved Coreferences
- Resolved Entities
Problem Formulation

Program

```python
def run():
    if is_there[River]:
        mine(Wood)
        build_bridge()
    if agent[Iron]<3:
        mine(Iron)
        place(Iron, 1, 1)
    else:
        goto(4, 2)
    while env[Gold]>0:
        mine(Gold)
```
Program

```python
def run():
    if is_there[River]:
        mine(Wood)
        build_bridge()
    if agent[Iron]<3:
        mine(Iron)
        place(Iron, 1, 1)
    else:
        goto(4, 2)
    while env[Gold]>0:
        mine(Gold)
```

State

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
0 1 3
```
Problem Formulation

def run():
    if is_there[River]:
        mine(Wood)
        build_bridge()
    if agent[Iron]<3:
        mine(Iron)
        place(Iron, 1, 1)
    else:
        goto(4, 2)
    while env[Gold]>0:
        mine(Gold)

Program

State

Execution
Problem Formulation

Program

```python
def run():
    if is_there[River]:
        mine(Wood)
        build_bridge()
        if agent[Iron]<3:
            mine(Iron)
            place(Iron, 1, 1)
        else:
            goto(4, 2)
    while env[Gold]>0:
        mine(Gold)
```

State

Execution
Problem Formulation

Program

```
def run():
    if is_there[River]:
        mine(Wood)
    build_bridge()
    if agent[Iron]<3:
        mine(Iron)
        place(Iron, 1, 1)
    else:
        goto(4, 2)
    while env[Gold]>0:
        mine(Gold)
```

State

Execution
Problem Formulation

Program

```python
def run():
    if is_there[River]:
        mine(Wood)
    build_bridge()
    if agent[Iron]<3:
        mine(Iron)
        place(Iron, 1, 1)
    else:
        goto(4, 2)
    while env[Gold]>0:
        mine(Gold)
```

State

Execution

```
x3  x1  x0
```

```
x4  x1  x0
```
Problem Formulation

Program

```
def run():
    if is_there[River]:
        mine(Wood)
        build_bridge()
        if agent[Iron]<3:
            mine(Iron)
            place(Iron, 1, 1)
        else:
            goto(4, 2)
    while env[Gold]>0:
        mine(Gold)
```

State

Execution
def run():
    if is_there[River]:
        mine[Wood]
        build_bridge()
        if agent[Iron]<3:
            mine[Iron]
            place[Iron, 1, 1]
    else:
        goto(4, 2)
        while env[Gold]>0:
            mine[Gold]

Problem Formulation

Program  State  Execution
Problem Formulation

Program

```python
def run():
    if is_there[River]:
        mine(Wood)
        build_bridge()
        if agent[Iron]<3:
            mine(Iron)
        place(Iron, 1, 1)
    else:
        goto(4, 2)
    while env[Gold]>0:
        mine(Gold)
```

State

Execution
Problem Formulation

Program

```python
def run():
    if is_there[River]:
        mine(Wood)
        build_bridge()
        if agent[Iron]<3:
            mine(Iron)
    else:
        goto(4, 2)
        while env[Gold]>0:
            mine(Gold)
```

State

Execution
Problem Formulation

Program

def run():
    if is_there[River]:
        mine(Wood)
        build_bridge()
    if agent[Iron]<3:
        mine(Iron)
        place(Iron, 1, 1)
    else:
        goto(4, 2)
while env[Gold]>0:
    mine(Gold)

State

Execution
def run():
    if is_there[River]:
        mine(Wood)
        build_bridge()
    if agent[Iron]<3:
        mine(Iron)
        place(Iron, 1, 1)
    else:
        goto(4, 2)
    while env[Gold]>0:
        mine(Gold)

Program

State

Execution
Problem Formulation

Program

```python
def run():
    if is_there[River]:
        mine(Wood)
        build_bridge()
    if agent[Iron]<3:
        mine(Iron)
        place(Iron, 1, 1)
    else:
        goto(4, 2)
    while env[Gold]>0:
        mine(Gold)
```

State

Execution
If a river is in the environment, mine a wood and then use it to build a bridge. And then if agent has less than three iron, place an iron at (2,3). Otherwise if no river, goto location (4,2). Finally, whenever there’s still gold in the environment, mine a gold.

While agent has no more than 11 wood, place wood at (2,4) and iron at (1,1), then place iron at (8,5) and mine gold twice, then mine gold. After the preceding procedure, sell gold and sell iron 4 times.
def run():
    while env[Gold] > 0:
        mine(Gold)
    if is_there[River]:
        build_bridge()
    place(Wood, 2, 3)

If an agent has more than 1 iron, place an iron on (2,3), and then if there are less than 3 gold in the environment, mine gold; otherwise, goto (4,2). While gold in the environment is larger than 2, keep mining gold.
```python
def run():
    while env[Gold] > 0:
        mine(Gold)
        if is_there[River]:
            build_bridge()
            place(Wood, 2, 3)
```

**Program Guided Agent**

**State**

**Program**

**Module**

**Module Output**

**Environment**

**Action**

**Policy**

**Goal**

**Program Interpreter**

**Perception Module**

**Query**

**Response**

**Module**

**Module Output**

**Environment**
Program Interpreter

• Comprehend a given program to 3 categories:
  • **Subtasks (actions):** what agent should perform
  • **Perception:** information from the environment
  • **Control flow:** decide to call different subtasks according to perceived information

```python
def run():
    while env[Gold] > 0:
        mine(Gold)
    if is_there[River]:
        build_bridge()
        place(Wood, 2, 3)
```
Perception Module

- Extract environmental information for choosing a path in a program

**Input**
- **Query**: a symbolically represented query (*e.g.* `is_there[River]`)
- **State** $s$: environment map and agent inventory status

**Output**
- Predicted **answer** to the query (*e.g.* True/False)

---

```python
def run():
    while env[Gold] > 0:
        mine(Gold)
    if is_there[River]:
        build_bridge()
        place(Wood, 2, 3)
```

---

![Diagram](image-url)
Policy

- Take low-level actions on the environment for fulfilling a subtask
- **Input**
  - Symbolically represented subtask (goal) \( g \)
  - State \( s \)
- **Output**
  - Predicted action distribution

```python
def run():
    while env[Gold] > 0:
        mine(Gold)
    if is_there[River]:
        build_bridge()
        place(Wood, 2, 3)
```
Result

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Natural language descriptions</th>
<th>Programs</th>
<th>Generalization gap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Instruction Method</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Seq-LSTM</td>
<td>Transformer</td>
<td>Seq-LSTM</td>
</tr>
<tr>
<td>test</td>
<td>54.9 ± 1.8%</td>
<td>52.5 ± 2.6%</td>
<td>56.7 ± 1.9%</td>
</tr>
<tr>
<td>test-complex</td>
<td>32.4 ± 4.9%</td>
<td>38.2 ± 1.2%</td>
<td>38.8 ± 1.2%</td>
</tr>
<tr>
<td>Generalization gap</td>
<td>40.9%</td>
<td>27.2%</td>
<td>31.6%</td>
</tr>
</tbody>
</table>

(a) Instruction Length

(b) Instruction Complexity
Conclusion

- Specific tasks using **programs**

```
def Task():
    if is_there[River]:
        mine(Wood)
        build_bridge()
    if agent[Iron] < 3:
        mine(Iron)
        place(Iron, 2, 3)
    else:
        goto(4, 2)
    while env[Gold] > 0:
        mine(Gold)
```

- Leverage the structure of programs with a modular framework
Program Guided Agent

ICLR 2020 (Spotlight)

Thank You for Your Attention

Shao-Hua Sun
Te-Lin Wu
Joseph J. Lim