KBAI
Fall 2015

Learning new skills by observing humans:

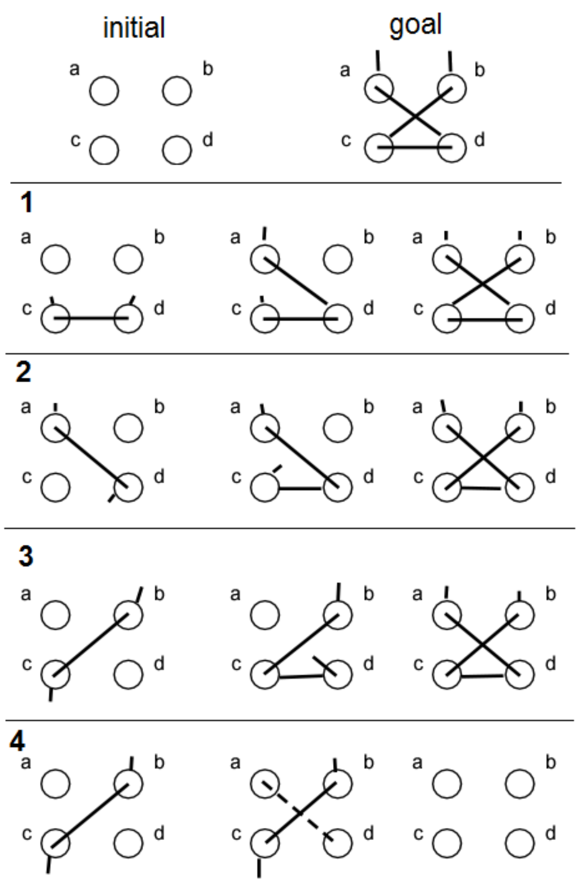
My robot has a plan to tie shoelaces

The goal is to design robots that can observe humans completing a task (tie shoelaces) and learn to complete the task themselves.

The challenges:

Most people would not consider tying shoelaces as a skill since it is so easy. We all learned it at our early ages and very likely repeat it often. However, it is a quite challenge task for robots. First, most robots never wear shoes, thus they have no chance to practice it like humans. Second, just by looking at a simple example in Figure 1, you will find out that tying shoelace does contain some complexity. As presented in Figure 1, to tie shoelace for one shoe with only four eyelets (a, b, c, d), we can start by connecting c-d, followed by connecting d-a, and c-b (Figure 1.1), or we can try connecting a-d, d-c and c-b (Figure 1.2). We can also try by connecting c-b, c-d and d-a (Figure 1.3). But if you think it can be done in a random order, then you are wrong. There are some internal constrains. For example, you cannot tie the shoelace in the order as c-b, a-d and c-d (Figure 1.4). ***Here, I present how robots learn the skill from human by planning.***

Figure 1



How robots learn a skill by planning:

In Artificial Intelligence, planning is about the decision making to achieve goals. It contains the following steps:

1, identify the initial state and goal.

2, divide the goal to a series of partial plans.

3, identify the internal constrains between partial plans.

4, achieve the goal by performing partial plans in the correct order.

KBAI
Fall 2015

1, identify the initial state and goal.

Considering this specific task (tying shoelaces), the initial state and goal can be easily identified by robots. Shoe eyelet can be considered as a class of object, which has two states (*Open*, or *Filled*). In initial state, all four eyelets are at Open states. After achieved the goal, all four eyelets are in Filled states, and their connections can be stored as list of tuples like: *style = [(a, d), (b, c), (c, b)]*. Robots can obtain the style list through visual examining human's shoes (when they find some new fashion styles) or retrieving it from their memory after humans' training. After training, robots should have a series of style lists stored in their memory.

2, divide the goal to a series of partial plans.

The goal can be easily divided into small plans by connecting eyelets in each tuple in the style list:

Connect(a, d), Connect(b, c), Connect(c, d)

3, identify the internal constrains between partial plans.

The operation, *Connect(eyelet1, eyelet2)*, connects two eyelet objects, and change their states from Open to Filled. Robots will take some time to figure out constrains in the Connect operation. By observing human's practice, robots may first notice that there is no constrains at the initial step. As shown in Figure 1.1-1.3, humans can perform *Connect(a, d), Connect(b, c) or Connect(c, d)* at the initial step. Constrains do exist at later steps. Since eyelets only have two states: *Open* or *Filled*, constrains may related to eyelets' states.

In Figure 1,

1.1: succeed.

Open(c) & Open(d) -> Connect(c, d) == True & Filled(c) & Filled(d) [initial step]

Open(a) & Filled(d) -> Connect(a, d) == True & Filled(a)

Open(b) & Filled(c) -> Connect(b, c) == True & Filled(b)

1.2: succeed.

Open(a) & Open(d) -> Connect(a, d) == True & Filled(a) & Filled(d) [initial step]

Open(c) & Filled(d) -> Connect(c, d) == True & Filled(c)

Open(b) & Filled(c) -> Connect(b, c) == True & Filled(b)

1.3: succeed.

Open(b) & Open(c) -> Connect(c, b) == True & Filled(c) & Filled(b) [initial step]

Filled(c) & Open(d) -> Connect(c, d) == True & Filled(d)

Filled(d) & Open(a) -> Connect(d, a) == True & Filled(a)

1.4: failed.

Open(b) & Open(c) -> Connect(c, b) == True & Filled(c) & Filled(b) [initial step]

Open(a) & Open(d) -> Connect(a, d) == False

After analyzing relations between eyelets' states and Connect operation in Figure 1, it is easy to find out such constrains for Connect operation:

Connect(x, y)

If it is an initial step:

Pre-conditions: None

Post-conditions: Filled(x) & Filled(y) & Connect(x, y) == True

If it is not an initial step:

Pre-conditions: Filled(x) | Filled(y)

Post-conditions: Filled(x) & Filled(y) & Connect(x, y) == True

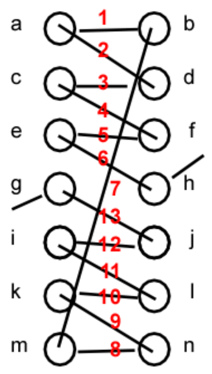
4, achieve the goal by performing partial plans in the correct order.

After identified constrains in Connect operation, robots can easily achieve the final goal. Since there is no restrictions for the initial operation, robots can randomly pick a partial plan. For the other partial plans, robots need to examine their pre-conditions, and finish these qualified partial plans first. After repeatedly examining the left partial plans, the final goal will be achieved.

Implementations:

The idea has been implemented in python (code attached, and you are welcome to try it on your own computer). I have created a class *Eyelets*, which has two attributes: name and filled. I also created two functions (*connect* and *tieShoelace*). The connect function takes two Eyelets objects and a Boolean value (to examine whether it's an initial step) as parameters, returns a Boolean value and prints out valid connections. tieShoelace function takes style (a list of tuples) as the only parameter, used a *while* loop and a *for* loop to repeatedly connect valid tuples of Eyelets objects until reaches the final goal.

Figure 2



```
connects: a + b
connects: a + d
connects: d + c
connects: c + f
connects: f + e
connects: e + h
connects: b + m
connects: m + n
connects: n + k
connects: k + l
connects: l + i
connects: i + j
connects: j + g
```

My robot can tie complicated shoelace as shown in Figure 2, which is far away from the original simple sample presented in Figure 1, showing his creativity and flexibility.

KBAI
Fall 2015

```
class Eyelets(object):
    def __init__(self, name):
        self.name = name
        self.filled = False

def connect(eyelet1, eyelet2, initial):
    if initial == True:
        eyelet1.filled = True
        eyelet2.filled = True
        print 'connects: ' + eyelet1.name + ' + ' + eyelet2.name
        return True
    elif eyelet1.filled == True:
        eyelet2.filled = True
        print 'connects: ' + eyelet1.name + ' + ' + eyelet2.name
        return True
    elif eyelet2.filled == True:
        eyelet1.filled = True
        print 'connects: ' + eyelet2.name + ' + ' + eyelet1.name
        return True
    else:
        return False

def tieShoelace(style):
    connect(style[0][0], style[0][1], True)
    style.pop(0)
    while len(style) > 0:
        temp = []
        for i in range(len(style)):
            op = connect(style[i][0], style[i][1], False)
            if op != True:
                temp.append(style[i])
        style = temp

a = Eyelets('a')
b = Eyelets('b')
c = Eyelets('c')
d = Eyelets('d')
e = Eyelets('e')
f = Eyelets('f')
g = Eyelets('g')
h = Eyelets('h')
i = Eyelets('i')
j = Eyelets('j')
k = Eyelets('k')
l = Eyelets('l')
m = Eyelets('m')
n = Eyelets('n')

style = [(a, b), (a, d), (c, d), (c, f), (e, f), (e, h), (g, j), (i, j), (i, l), (k, l), (k, n), (m, n), (m, b)]

tieShoelace(style)
```