

New Coverage Algorithm

Charles Zhang

July 1 2020

```
options(digits = 6)
```

Environment:

```
"""
```

```
1 — 2 — 3 — 4 — 5 — 6 — 7 — 8
9 — 10 — 11 — 12 — 13 — 14 — 15 — 16
17 — 18 — 19 — 20 — 21 — 22 — 23 — 24
25 — 26 — 27 — 28 — 29 — 30 — 31 — 32
33 — 34 — 35 — 36 — 37 — 38 — 39 — 40
41 — 42 — 43 — 44 — 45 — 46 — 47 — 48
49 — 50 — 51 — 52 — 53 — 54 — 55 — 56
"""
```

Global variable

```
ROWS = 7
COLS = 8
END = 2
START = 1
```

```
V = ROWS * COLS
S = 1:V
R = matrix(-1, V, V)
# four corners
R[1, c(2, 1+COLS)] = 0
R[COLS*(ROWS-1)+1, c(COLS*(ROWS-1)+1-COLS, COLS*(ROWS-1)+2)] = 0
R[COLS, c(COLS-1, 2*COLS)] = 0
R[V, c(V-1, V-COLS)] = 0
# four boundary edges
for(i in 2:(COLS - 1)) {
  R[i, c(i-1, i+1, i+COLS)] = 0      # up edge
  R[V-i+1, c(V-i+2, V-i, V-i+1-COLS)] = 0  # bottom edge
}
for(i in 1:(ROWS-2)) {
  R[i*COLS+1, c(i*COLS+1-COLS, i*COLS+2, i*COLS+1+COLS)] = 0  # left edge
  R[(i+1)*COLS, c(i*COLS, (i+2)*COLS, (i+1)*COLS-1)] = 0  # right edge
}
# inside vertices
for (i in 0:(COLS-3)) {
  for (j in 1:(ROWS-2)) {
    R[j*COLS+2+i, c(j*COLS+1+i, j*COLS+3+i, j*COLS+10+i, j*COLS-6+i)] = 0
  }
}
```

```

# give reward
R[2, c(1, 10, 3)] = 100
Q = matrix(0, V, V)
alpha = 0.6

rounds = 500
r = 1
get_actions <- function(s) {
  a = c()
  for (i in 1:V) {
    if(R[s,i] != -1) a = c(a, i)
  }
  return(a)
}

```

Core algorithm based on Q learning

```

while (r <= rounds) {
  s = sample(S, 1)
  while (TRUE) {
    action_space = get_actions(s)
    action <- sample(action_space, 1)
    s_next <- action
    actions_next = get_actions(s_next)
    qs = c()
    for (i in actions_next) qs = c(Q[s_next,i], qs)
    Q[s,action] <- R[s,action] + alpha * max(qs)
    s = s_next
    if (s == END) break
  }
  r <- r+1
}

```

Find Path based on Q table

```

path = c()
state = START
Q[Q == 0] <- 1000
while (length(path) < V)
{
  pre_state = state
  path = c(path, state)
  state = match((min(Q[state,])), Q[state,])
  Q[pre_state, ] = 1000
  Q[, pre_state] = 1000
}
path

```

```

## [1] 1 9 17 25 33 41 49 50 51 52 53 54 55 56 48 40 32 24 16 8 7 15 23
## [24] 31 39 47 46 38 30 22 14 6 5 13 21 29 37 45 44 36 28 20 12 4 3 11
## [47] 19 27 35 43 42 34 26 18 10 2

```