New Coverage Algorithm

Charles Zhang

July 1 2020

```
options(digits = 6)
library(tictoc)
```

Environment Example(7x8):

"""

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 = 20
COLS = 20
START = 1
Reward = 100
alpha = 0.6    # discont factor
```

> Create Reward Matrix(VxV): 0 if connected, -1 if not connected. In reward matrix, [i,j] means go from i to j.

```
V = ROWS * COLS
S = 1:V
R = matrix(-1, V, V)
# four cornors
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+2+COLS+i, j*COLS+2-COLS+i)] = 0
  }
}
# give reward
if (COLS %% 2 == 0) {
  END = 2  # aviod repetition from the start point 1
  R[c(1, COLS+2, 3), 2] = Reward
# else if (ROWS %% 2 == 0) {
#   END = COLS+1
#   R[END, c(1, END+1, END+COLS)] = Reward     # still working on this
# }
} else {
  END = 2
  R[c(1, COLS+2, 3), 2] = Reward
  print("Not Hamiltomian Cycle so No Coverage Path!")
}
# initialized VxV Q table
Q = matrix(0, V, V)
```

find valid action space for the state s(<=> find connected vectex for the current state vertex)

```
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 SARSA on-policy reinforcement learning

```
tic()
rounds = 1000
r = 1
while (r <= rounds) {
  s = sample(S, 1)   # random state
  while (TRUE) {
    action_space = get_actions(s)   # action space for S
    action <- sample(action_space, 1)   # random action to the next state
    s_next <- action    # next state S'
    actions_next = get_actions(s_next)   # action space for S'
    qs = c()
    for (i in actions_next) qs = c(Q[s_next,i], qs)   # list of all Q(S',a')
    Q[s,action] <- R[s,action] + alpha * max(qs)   # update by simple bellman equation
    s = s_next    # update S
    if (s == END) break   # reach the final state
  }
  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)   # append the state
  state = match((min(Q[state,])), Q[state,])   # argmin Q(S, )
  Q[pre_state, ] = 1000     # clear the column and row of the appended state by giving a large number(c
  Q[, pre_state] = 1000
}
```

Running Time

```
toc()
```

```
## 242.072 sec elapsed
```

Coverage Path

```
path
```

```
##   [1]   1  21  41  61  81 101 121 141 161 181 201 221 241 261 281 301 321
##  [18] 341 361 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395
##  [35] 396 397 398 399 400 380 360 340 320 300 280 260 240 220 200 180 160
##  [52] 140 120 100  80  60  40  20  19  39  59  79  99 119 139 159 179 199
##  [69] 219 239 259 279 299 319 339 359 379 378 358 338 318 298 278 258 238
##  [86] 218 198 178 158 138 118  98  78  58  38  18  17  37  57  77  97 117
## [103] 137 157 177 197 217 237 257 277 297 317 337 357 377 376 356 336 316
## [120] 296 276 256 236 216 196 176 156 136 116  96  76  56  36  16  15  35
## [137]  55  75  95 115 135 155 175 195 215 235 255 275 295 315 335 355 375
## [154] 374 354 334 314 294 274 254 234 214 194 174 154 134 114  94  74  54
## [171]  34  14  13  33  53  73  93 113 133 153 173 193 213 233 253 273 293
## [188] 313 333 353 373 372 352 332 312 292 272 252 232 212 192 172 152 132
## [205] 112  92  72  52  32  12  11  31  51  71  91 111 131 151 171 191 211
## [222] 231 251 271 291 311 331 351 371 370 350 330 310 290 270 250 230 210
## [239] 190 170 150 130 110  90  70  50  30  10   9  29  49  69  89 109 129
## [256] 149 169 189 209 229 249 269 289 309 329 349 369 368 348 328 308 288
## [273] 268 248 228 208 188 168 148 128 108  88  68  48  28   8   7  27  47
## [290]  67  87 107 127 147 167 187 207 227 247 267 287 307 327 347 367 366
## [307] 346 326 306 286 266 246 226 206 186 166 146 126 106  86  66  46  26
## [324]   6   5  25  45  65  85 105 125 145 165 185 205 225 245 265 285 305
## [341] 325 345 365 364 344 324 304 284 264 244 224 204 184 164 144 124 104
## [358]  84  64  44  24   4   3  23  43  63  83 103 123 143 163 183 203 223
## [375] 243 263 283 303 323 343 363 362 342 322 302 282 262 242 222 202 182
## [392] 162 142 122 102  82  62  42  22   2
```