Game of Life ILP Generator

April 14, 2020

1 Pattern synthetizator of Conway's Game of Life using Mixed-Integer Linear Programming

Combinatorial Optimization course, FEE CTU in Prague. Created by Industrial Informatics Department.

In this exercise, we take a look on famous Conway's Game of Life. However, we will not deal with a simple simulation of some patterns, but rather we translate the rules of the game into an ILP model. This will allow us to impose constraints on values of cells and use that to reason about properties of the game. For example, we can synthetizate a new patterns that have desired behavior.

Namely, in this assignment, we will focus on **finding the smallest movable object in the game**.

[1]:
```
%%capture
%matplotlib inline
import gurobipy as g
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
plt.rcParams["animation.html"] = "jshtml"
```

1.1 Patern synthetizator

Let us define our ILP. The modeling is fairly straight-forward. With the given time horizon, we will model the state of the world for each time step. The constraints ensure, how the state of the game at time $t + 1$ is updated as a function of the state at time t. The primary decision variables are x_{ijt} with the meaning if cell at (i, j) is alive at time t. The other auxilary variables are used to detect how many cells are alive in the 8-neighborhood of each cell at each time instance.

Finally, we specify the objective of our task. We want to find the smallest possible object, that appears in the top-left corner and is able to travel to the bottom-right one. Note that we do not impose any structural constraints on the object along the time steps in between.

[22]:
```
def ilp_find_smallest_movable(n, tmax):
    m = g.Model()
    x = m.addVars(n+2, n+2, tmax, vtype=g.GRB.BINARY)  # x_ijt is 1 iff cell␣
 ↪(i,j) is alive at time t
    a = m.addVars(n+2, n+2, tmax, vtype=g.GRB.BINARY)
    b = m.addVars(n+2, n+2, tmax, vtype=g.GRB.BINARY)
    c = m.addVars(n+2, n+2, tmax, vtype=g.GRB.BINARY)
    d = m.addVars(n+2, n+2, tmax, vtype=g.GRB.BINARY)

    for t in range(tmax-1):
        for i in range(1, n+1):
            for j in range(1, n+1):
                eight_neigborhood = [x[i + 1, j, t], x[i - 1, j, t],
                                     x[i, j + 1 ,t], x[i, j - 1, t],
                                     x[i + 1, j + 1, t], x[i - 1, j + 1, t],
                                     x[i + 1, j - 1, t], x[i - 1, j - 1, t]
                                    ]
                # beware: these are not ILP constraints! but can be translated␣
↪into them using big-M trick
                # for clarity and consise represetation, we will state them in␣
↪non-linear form
                m.addConstr((a[i,j,t] == 1) >> (g.quicksum(eight_neigborhood)␣
↪<= 1))
                m.addConstr((b[i,j,t] == 1) >> (g.quicksum(eight_neigborhood)␣
↪== 2))
                m.addConstr((c[i,j,t] == 1) >> (g.quicksum(eight_neigborhood)␣
↪== 3))
                m.addConstr((d[i,j,t] == 1) >> (g.quicksum(eight_neigborhood)␣
↪>= 4))
                m.addConstr(a[i,j,t] + b[i,j,t] + c[i,j,t] + d[i,j,t] == 1)

                # rules of the game
                m.addConstr((x[i,j,t] == 0) >> (x[i, j, t+1] == c[i,j,t]))
                m.addConstr((x[i,j,t] == 1) >> (x[i, j, t+1] >= b[i,j,t] +␣
↪c[i,j,t]))
                m.addConstr((x[i,j,t] == 1) >> (x[i,j,t+1] + 1 <= 1-a[i,j,t] +␣
↪1-d[i,j,t]))

    # virtual borders are always blank
    m.addConstr(x.sum(0, '*', '*') + x.sum(n+1, '*', '*') + x.sum('*', 0, '*')␣
↪+ x.sum('*', n+1, '*') == 0)

    # the syntetized configuration needs to be born in the top-left corner
    m.addConstr(g.quicksum(x[i,j,0] for i in range(6, n+1) for j in range(6,␣
↪n+1)) == 0)
    m.addConstr(g.quicksum(x[i,j,0] for i in range(6, n+1) for j in range(1,␣
↪n+1)) == 0)
    m.addConstr(g.quicksum(x[i,j,0] for i in range(1, n+1) for j in range(6,␣
↪n+1)) == 0)
    m.addConstr(g.quicksum(x[i,j,0] for i in range(1, 5) for j in range(1, 5))␣
↪>= 1)
```

```
    # but it needs to spread itself into the bottom-right corner within tmax␣
↪steps
    m.addConstr(g.quicksum(x[i,j,tmax-1] for i in range(7, n+1) for j in␣
↪range(7, n+1)) >= 1)

    # we want to find the smallest possible initial configuration
    m.setObjective(x.sum('*', '*', 0), sense=g.GRB.MINIMIZE)
    m.params.mipfocus = 1
    m.optimize()

    board = np.zeros(shape=(n, n), dtype=int)
    for i in range(1,n+1):
        for j in range(1,n+1):
            board[i-1, j-1] = int(round(x[i, j, 0].x))
    return board

# simulation loop
def update_board(frame_num, img, current_board):
    n = len(current_board)
    n = n-2
    new_board = np.zeros(shape=(n+2, n+2), dtype=int)
    for i in range(1, n+1):
        for j in range(1, n+1):
            eight_neigborhood = [current_board[i + 1, j], current_board[i - 1,␣
↪j],
                                 current_board[i, j + 1], current_board[i, j -␣
↪1],
                                 current_board[i + 1, j + 1], current_board[i -␣
↪1, j + 1],
                                 current_board[i + 1, j - 1], current_board[i -␣
↪1, j - 1]
                                ]
            if current_board[i, j] == 1:
                if sum(eight_neigborhood) <= 1:
                    new_board[i, j] = 0
                elif 3 >= sum(eight_neigborhood) >= 2:
                    new_board[i, j] = 1
                elif sum(eight_neigborhood) >= 4:
                    new_board[i, j] = 0
            elif sum(eight_neigborhood) == 3:
                    new_board[i, j] = 1
    current_board[:] = new_board[:]
    img.set_data(new_board)
    return img,
```

1.2 Solving the model

Now we will look for such configuration.

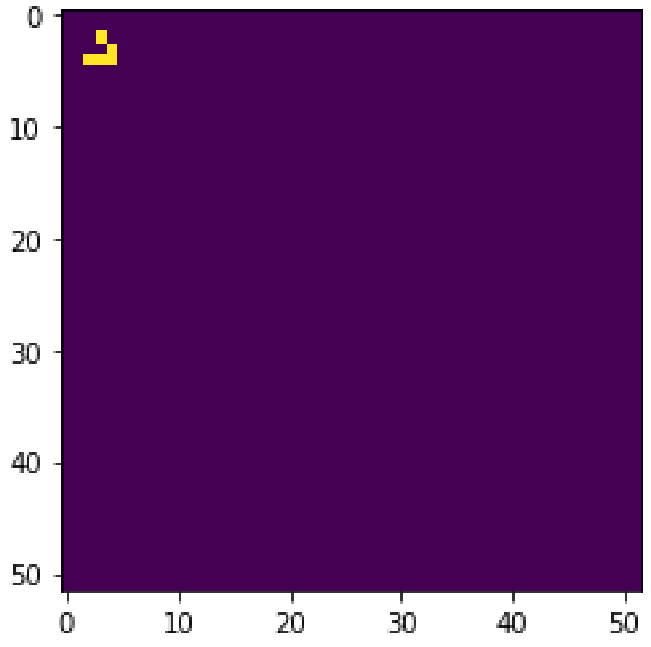
```
[ ]: config = ilp_find_smallest_movable(12, 10)
```

1.3 Displaying the results

This is our initial configuration that was found by ILP for which it claims it can "move".

```
[24]: n = 50
      board = np.zeros(shape=(n+2, n+2), dtype=int)
      board[0:len(config), 0:len(config)] = config

      fig, ax = plt.subplots()
      img = ax.imshow(board, interpolation='nearest')
      ani = animation.FuncAnimation(fig, update_board, fargs=(img, board),
                                    frames = 180,
                                    interval=100)
```



And lets test it how it moves!

So, our ILP has found, in a certain sense, the smallest movable object in the game. And it happens to be famous glider configuration! Moreover, we proved that this is indeed the smallest such object.

This example has demonstrated a capability of ILP to reason about complex systems. Altought this specific example is just a puzzle, similar approach can be applied to more practical things.

1.4 More tips for you

Try to adjust objective and constraints such that our ILP will synthetize the following patterns: * glider gun * oscilators * spaceship * change constraints such that the cells live on a torus instead of a bounded square

Or maybe you will discover new structures that are not known so far!