CSP Representation of Akari

Definitions:

Domain = {0,1} *0/1* for is there light bulb or no

Vertex = WhiteBox = [i, (v), (h)] *i* for the index, *v* for vertical white box group, *h* for horizontal white box group,

(v) and (h) will be defined via python code, and will only store the indices of the relevant white boxes

Other definitions:

B = empty black box

(0,1,2,3,4) = black box with the number

adjacentWhiteBox = adjacent white box list for black box with a number (unique list for every black box with a number)

W = representing white boxes

L = white box with a light bulb

X = outer space of the game board (only visible on debugging)

constraints:

exactSum(adjacentWhiteBox) = inner number of the black box

This line ensures that there are exact amount of light bulbs near the black boxes with number

maxSum(v) = 1

This line ensures there are no light bulbs illuminating each other vertically

maxSum(h) = 1

This line ensures there are no light bulbs illuminating each other horizontally

minSum(WhiteBox(v and h)) = 1

This line ensures every white box is illuminated

Discussion on Akari, comparing A* and CSP

Algorithmic perspective:

A*: For games like Akari, Sudoku, magic square, etc… there might be enormous number of states, generating all of them , defining a good heuristic, making this heuristic efficient enough, all of them will require great effort and all of these will change drastically between every genre.

CSP: Games like Akari, Sudoku, rooks. etc… defined by simple rules, which can be easily translated into constraints. We only need to define vertices and respective domains, which is quite easier compared to constructing an A* search for each problem.

Coding perspective:

Generating the states, keeping a list of all states for visited, generating a path, assigning heuristic for each state, all of them require separate functions or statements, an A* search program will be much longer than CSP solver code when compared. CSP will require respectively less coding effort. Also it is lot more easier to debug, since debugging a path to be constructed from tree with A* search can be get out of hand very quickly depending on the branching factor (need to inspect cost, heuristic, child-parent relationship, path construction, visited list, frontier for each state and each node). On the other hand, CSP nearly requires no debugging, if the constraints and the definitions are correct, it will solve itself, no need to track the solving progress.