دانشکده مهندسی کامپیوتر

هوش مصنوعی و سیستم‌های خبره

تمرین تشریحی سوم[1]

نام و نام خانوادگی - شماره دانشجویی . . . .

مدرس . . . . محمدطاهر پیله‌ور - سید صالح اعتمادی

طراحی و تدوین . . . . مرسده ایرانی - مهسا قادران

تاریخ انتشار . . . . ۲۸ مهر ۱۳۹۹

تاریخ تحویل گروه ۱ . . . . ۵ آبان ۱۳۹۹

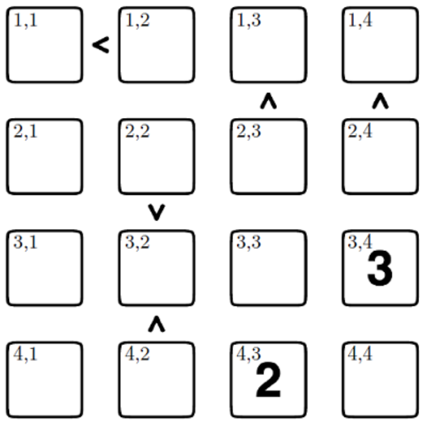
تاریخ تحویل گروه ۲ . . . . ۲۸ مهر ۱۳۹۹

[1] در طراحی این تمرین از منابع کورس CS188 دانشگاه برکلی استفاده شده است.

۱ CSP Futoshiki

Futoshiki is a Japanese logic puzzle that is very simple, but can be quite challenging. You are given an n x n grid, and must place the numbers 1,..., n in the grid such that every row and column has exactly one of each. Additionally, the assignment must satisfy the inequalities placed between some adjacent squares.

Bellow is an instance of this problem, for size n = 4. Some of the squares have known values, such that the puzzle has a unique solution. (The letters mean nothing to the puzzle, and will be used only as labels with which to refer to certain squares). Note also that inequalities apply only to the two adjacent squares, and do not directly constrain other squares in the row or column.



Let's formulate this puzzle as a CSP.
We will use 4^2 variables, one for each cell,with X_{ij} as the variable for the cell in the ith row and jth column (each cell contains its i, j label in the top left corner). The only unary constraints will be those assigning the known initial values to their respective squares (e.g. $X_{34} = 3$).

۱.۱

Complete the formulation of the CSP using only binary constraints (in addition to the unary constraints speciced above. In particular, describe the domains of the variables, and all binary constraints you think are necessary. You do not need to enumerate them all, just describe them using concise mathematical notation. You are not permitted to use n-ary constraints where n $\geqslant$ 3.

پاسخ:

Domains: $X_{ij} \in \{1, 2, 3, 4\}, \forall i, j$

Unary constraints: $X_{34} = 3$, $X_{43} = 2$

Inequality binary constraints: $X_{11} < X_{12}$, $X_{13} < X_{23}$, $X_{14} < X_{24}$, $X_{32} < X_{22}$, $X_{32} < X_{42}$

Row binary contraints: $X_{ij} \neq X_{ik}, \forall i, j, k, j \neq k$

Column binary contraints: $X_{ij} \neq X_{kj}, \forall i, j, k, i \neq k$

۲.۱

After enforcing unary constraints, consider the binary constraints involving X_{14} and X_{24}. Enforce arc consistency on just these constraints and state the resulting domains for the two variables.

پاسخ:

$X_{14} \in \{1, 2\}$, $X_{24} \in \{2, 4\}$

Note that both threes are removed from the column constraint with X_{34}

۳.۱

Suppose we enforced unary constraints and ran arc consistency on this CSP, pruning the domains of all variables as much as possible. After this, what is the maximum possible domain size for any variable? [Hint: consider the least constrained variable(s); you should not have to run every step of arc consistency.]

پاسخ:

The maximum possible domain size is 4 (ie, no values are removed from the original domain). Consider X_{21} - we will not be able to eliminate any values from its domain through arc consistency.

۴.۱

Suppose we enforced unary constraints and ran arc consistency on the initial CSP in the figure above. What is the maximum possible domain size for a variable adjacent to an inequality?

پاسخ:

The maximum domain size is 3 - you must always eliminate either 1 or 4 from a variable participating in an inequality constraint.

۵.۱

By inspection of column 2, we find it is necessary that $X_{32} = 1$, despite not having found an assignment to any of the other cells in that column. Would running arc consistency find this requirement? Explain why or why not.

پاسخ:

No, arc consistency would not find this requirement.
Encorcing the X_{32} $\longrightarrow$X$_{42}$ and the X_{42} $\longrightarrow$X$_{43}$ arc leaves X_{42} with a domain of {3,4} .
Enforcing the $X_{32} < X_{22}$ constraints leaves $X_{32} \in \{1, 2, 3\}$ *and* X$_{22}$ $\in \{2, 3, 4\}$.
Enforcing that they are all different does not remove any values.
After this point, every arc in this column is consistent and X_{32} is not required to be 1.

۲ CSPs: Properties

۱.۲

When enforcing arc consistency in a CSP, the set of values which remain when the algorithm terminates does not depend on the order in which arcs are processed from the queue.

True False

پاسخ:

$True$

۲.۲

In a general CSP with n variables, each taking d possible values, what is the maximum number of times a backtracking search algorithm might have to backtrack (i.e. the number of the times it generates an assignment, partial or complete, that violates the constraints) before finding a solution or concluding that none exists? (circle one)

0 O(1) O(nd^2) $O(n^2d^3)$ $O(d^n)$ ∞

پاسخ:

$O(d^n)$

In general, the search might have to examine all possible assignments.

۳.۲

What is the maximum number of times a backtracking search algorithm might have to backtrack in a general CSP, if it is running arc consistency and applying the MRV and LCV heuristics? (circle one)

0 O(1) O(nd^2) $O(n^2d^3)$ $O(d^n)$ ∞

پاسخ:

$O(d^n)$

The MRV and LCV heuristics are often helpful to guide the search, but are not guaranteed to reduce back- tracking in the worst case. In fact, CSP solving is NP-complete, so any polynomial-time method for solving general CSPs would consititute a proof of P = NP (worth a million dollars from the Clay Mathematics Institute!).

۴.۲

What is the maximum number of times a backtracking search algorithm might have to backtrack in a tree- structured CSP, if it is running arc consistency and using an optimal variable ordering? (circle one)

0 O(1) O(nd^2) $O(n^2d^3)$ $O(d^n)$ ∞

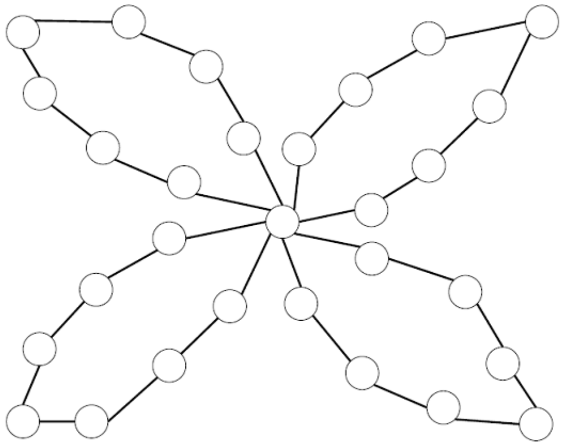
پاسخ:

0

Applying arc consistency to a tree-structured CSP guarantees that no backtracking is required, if variables are assigned starting at the root and moving down towards the leaves.

۵.۲

Constraint Graph Consider the following constraint graph:



In two sentences or less, describe a strategy for efficiently solving a CSP with this constraint structure.

پاسخ:

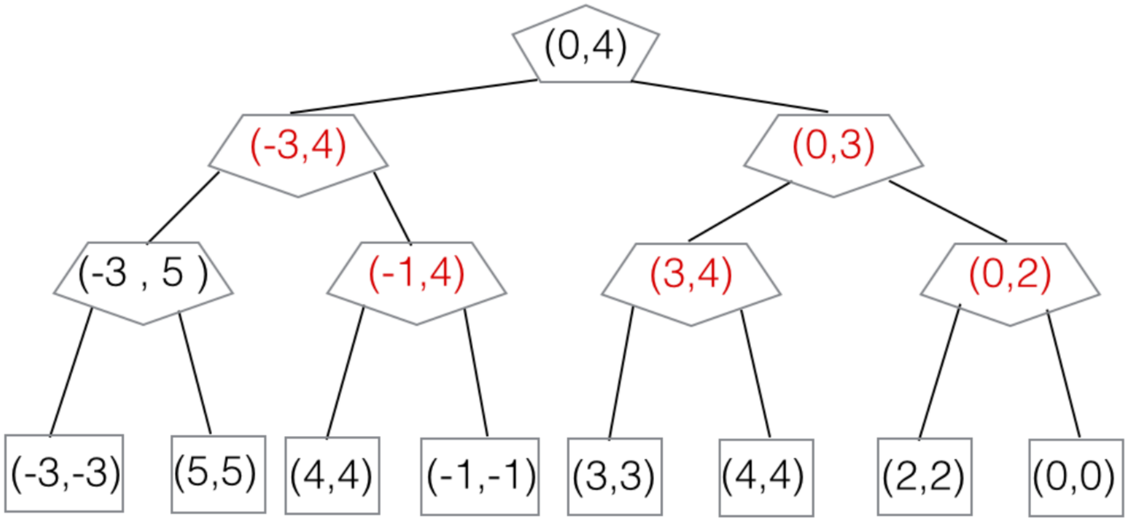
Loop over assignments to the variable in the middle of the constraint graph. Treating this node as a cutset, the graph becomes four independent tree-structured CSPs, each of which can be solved efficiently.

۳ One Wish Pacman

۱.۳

Power Search. Pacman has a special power: once in the entire game when a ghost is selecting an action, Pacman can make the ghost choose any desired action instead of the min-action which the ghost would normally take.
The ghosts know about this special power and act accordingly.

(i) Similar to the minimax algorithm, where the value of each node is determined by the game subtree hanging from that node, we define a value pair (u, v) for each node: u is the value of the subtree if the power is not used in that subtree; v is the value of the subtree if the power is used once in that subtree. For example, in the below subtree with values (-3, 5), if Pacman does not use the power, the ghost acting as a minimizer would choose -3; however, with the special power, Pacman can make the ghost choose the value more desirable to Pacman, in this case 5.
Reminder: Being allowed to use the power once during the game is different from being allowed to use the power in only one node in the game tree below. For example, if Pacman's strategy was to always use the special power on the second ghost then that would only use the power once during execution of the game, but the power would be used in four possible different nodes in the game tree.
For the terminal states we set u = v = Utility(State).
Fill in the (u, v) values in the modified minimax tree below. Pacman is the root and there are two ghosts.

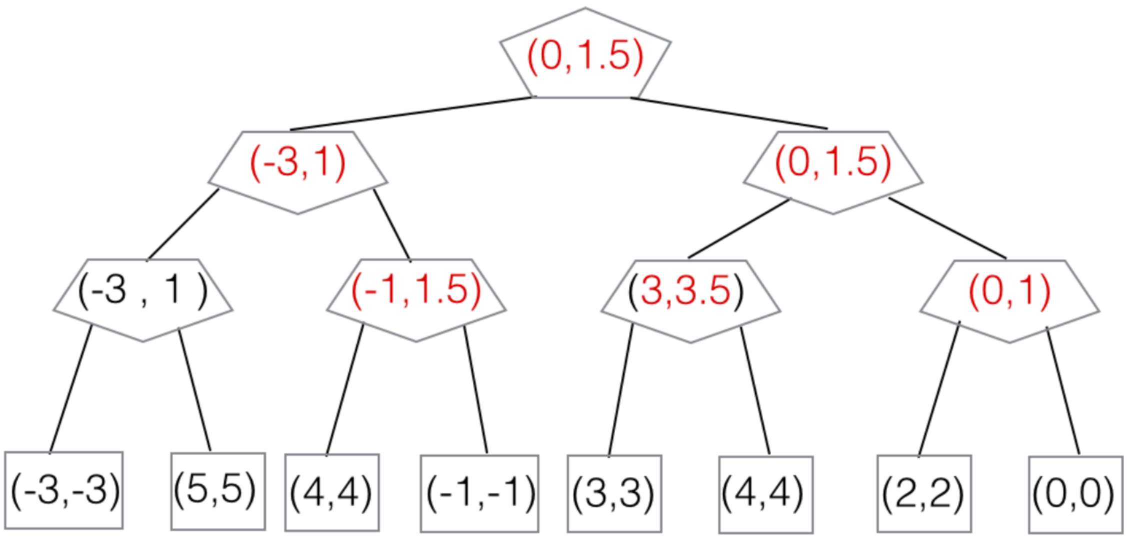


(ii) Complete the algorithm below, which is a modification of the minimax algorithm, to work in the general case: Pacman can use the power at most once in the game but Pacman and ghosts can have multiple turns in the game.

```
function VALUE(state)
    if state is leaf then
        u ← UTILITY(state)
        v ← UTILITY(state)
        return (u, v)
    end if
    if state is Max-Node then
        return MAX-VALUE(state)
    else
        return MIN-VALUE(state)
    end if
end function

function MAX-VALUE(state)
    uList ← [ ], vList ← [ ]
    for successor in SUCCESSORS(state) do
        (u′, v′) ← VALUE(successor)
        uList.append(u′)
        vList.append(v′)
    end for
    u ← max(uList)
    v ← max(vList)
    return (u, v)
end function

function MIN-VALUE(state)
    uList ← [ ], vList ← [ ]
    for successor in SUCCESSORS(state) do
        (u′, v′) ← VALUE(successor)
        uList.append(u′)
        vList.append(v′)
    end for

    u ← ____min(uList)____

    v ← ____max(max(uList), min(vList))____

    return (u, v)
end function
```

۲.۳

Weak-Power Search. Now, rather than giving Pacman control over a ghost move once in the game, the special power allows Pacman to once make a ghost act randomly. The ghosts know about Pacman's power and act accordingly.
(i) The propagated values (u, v) are defined similarly as in the preceding question: u is the value of the subtree if the power is not used in that subtree; v is the value of the subtree if the power is used once in that subtree.
Fill in the (u, v) values in the modified minimax tree below, where there are two ghosts.



(ii) Complete the algorithm below, which is a modification of the minimax algorithm, to work in the general case: Pacman can use the weak power at most once in the game but Pacman and ghosts can have multiple turns in the game.
Hint: you can make use of a min, max, and average function

function VALUE($state$)
　if $state$ is leaf **then**
　　$u \leftarrow$ UTILITY($state$)
　　$v \leftarrow$ UTILITY($state$)
　　return (u, v)
　end if
　if $state$ is Max-Node **then**
　　return MAX-VALUE($state$)
　else
　　return MIN-VALUE($state$)
　end if
end function

function MAX-VALUE($state$)
　$uList \leftarrow [\,], vList \leftarrow [\,]$
　for $successor$ in SUCCESSORS($state$) **do**
　　$(u', v') \leftarrow$ VALUE($successor$)
　　$uList$.append(u')
　　$vList$.append(v')
　end for
　$u \leftarrow \max(uList)$
　$v \leftarrow \max(vList)$
　return (u, v)
end function

function MIN-VALUE($state$)
　$uList \leftarrow [\,], vList \leftarrow [\,]$
　for $successor$ in SUCCESSORS($state$) **do**
　　$(u', v') \leftarrow$ VALUE($successor$)
　　$uList$.append(u')
　　$vList$.append(v')
　end for

　$u \leftarrow$ $\min(uList)$

　$v \leftarrow$ $\max(\text{avg}(uList), \min(vList))$

　return (u, v)
end function