Written Assignment 2

Deadline: February 16th, 2019

Instruction: You may discuss these problems with classmates, but please complete the write-ups individually. (This applies to BOTH undergraduates and graduate students.) Remember the collaboration guidelines set forth in class: you may meet to discuss problems with classmates, but you may not take any written notes (or electronic notes, or photos, etc.) away from the meeting. Your answers must be typewritten, except for figures or diagrams, which may be hand-drawn. Please submit your answers (pdf format only) on Canvas.

1 Q1. CSP Futoshiki [30 points]

Futoshiki is a Japanese logic puzzle that is very simple, but can be quite challenging. You are given an \(n \times n\) grid, and must place the numbers 1, 2, \ldots, \(n\) 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.

To the right 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 \(i^{th}\) row and \(j^{th}\) 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. \(X_{2,4} = 3\)).

(a) [6 points] Complete the formulation of the CSP using only binary constraints (in addition to the unary constraints specified 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 $n \geq 3$.

(b) [6 points] After enforcing unary constraints, consider the binary constraints involving $X_{3,4}$ and $X_{4,4}$. Enforce arc consistency on just these constraints and state the resulting domains for the two variables.

(c) [6 points] 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.]

(d) [6 points] 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?

(e) [6 points] By inspection of column 2, we find it is necessary that $X_{2,2} = 1$, 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.
Q2. CSPs: Properties [20 points]

(a) [4 points] 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

(b) [4 points] 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) (provide explanation)

\[0 \quad O(1) \quad O(nd^2) \quad O(n^2d^3) \quad O(d^n) \quad \infty\]

(c) [4 points] 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) (provide explanation)

\[0 \quad O(1) \quad O(nd^2) \quad O(n^2d^3) \quad O(d^n) \quad \infty\]

(d) [4 points] 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) (provide explanation)

\[0 \quad O(1) \quad O(nd^2) \quad O(n^2d^3) \quad O(d^n) \quad \infty\]

(e) [4 points] Constraint Graph Consider the following constraint graph:

In (≤)2 sentences, describe a strategy for efficiently solving a CSP with this constraint structure.
Q3. Games: Alpha-Beta Pruning [30 points]

For each of the game-trees shown below, state for which values of $x$ the dashed branch with the scissors will be pruned. If the pruning will not happen for any value of $x$ write “none”. If pruning will happen for all values of $x$ write “all”. Provide explanations of your answers.

(a) 

(b) 

(c) 

(d) 

(e)
Q4. Game Trees: Friendly Ghost [20 points]

Consider a two-player game between Pacman and a ghost in which both agents alternate moves. As usual, Pacman is a maximizer agent whose goal is to win by maximizing his own utility. Unlike the usual adversarial ghost, she is friendly and helps Pacman by maximizing his utility. Pacman is unaware of this and acts as usual (i.e. as if she is playing against him). She knows that Pacman is misinformed and acts accordingly.

(a) [4 points] In the minimax algorithm, the value of each node is determined by the game subtree hanging from that node. For this version, we instead define a value pair \((u,v)\) for each node:

- \(u\) is the value of the subtree as determined by Pacman, who acts to win while assuming that the ghost is a minimizer agent, and
- \(v\) is the value of the subtree as determined by the ghost, who acts to help Pacman win while knowing Pacman’s strategy.

For example, in the subtree below with values \((4,6)\), Pacman believes the ghost would choose the left action which has a value of 4, but in fact the ghost chooses the right action which has a value of 6, since that is better for Pacman. For the terminal states we set \(u = v = \text{Utility}(\text{State})\).

Fill in the remaining \((u,v)\) values in the modified minimax tree below, in which the ghost is the root. The ghost nodes are upside down pentagons and Pacman’s nodes are rightside up ones.

(b) [4 points] In the game tree above, put an ‘X’ on the branches that can be pruned and do not need to be explored when the ghost computes the value of the tree. Assume that the children
of a node are visited in left-to-right order and that you should not prune on equality. Explicitly write down “possible” below if no branches can be pruned, in which case any ‘X’ marks above will be ignored.

(c) [4 points] What would the value of the game tree be if instead Pacman knew that the ghost is friendly? What is the value (i.e. a single number) at the root of the game tree?

(d) [4 points] Complete the following modified minimax algorithm, to work in the original setting where the ghost is friendly unbeknownst to Pacman. (No pruning in this subquestion)

```plaintext
function Value(state)
    if state is leaf then
        \( (u, v) \leftarrow (\text{Utility}(state), \text{Utility}(state)) \)
        return \( (u, v) \)
    end if
    if state is Ghost-Node then
        return Ghost-Value(state)
    else
        return Pacman-Value(state)
    end if
end function
```

```plaintext
function Ghost-Value(state)
    \( (u, v) \leftarrow (+\infty, -\infty) \)
    for successor in Successors(state) do
        \( (u', v') \leftarrow Value(successor) \)
        \( (u, v) \leftarrow (\bar{u}, \bar{v}) \)
    end for
    return \( (u, v) \)
end function
```

```plaintext
function Pacman-Value(state)
    \( (u, v) \leftarrow (-\infty, +\infty) \)
    for successor in Successors(state) do
        \( (u', v') \leftarrow Value(successor) \)
        \( (u, v) \leftarrow (\bar{u}, \bar{v}) \)
    end for
    return \( (u, v) \)
end function
```

Complete the pseudocode by choosing the option that fills in each blank above. The code blocks \( A_1 - A_8 \) update \( \bar{u} \) (options for (i) and (iii)) and \( B_1 - B_8 \) update \( \bar{v} \) (options for (ii) and (iv)). If any of the code blocks are not needed, choose ‘None’.

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
</tr>
</thead>
<tbody>
<tr>
<td>if ( u' &lt; u ) then ( \bar{u} \leftarrow u' ) end if</td>
<td>if ( u' &lt; v ) then ( \bar{u} \leftarrow u' ) end if</td>
<td>if ( v' &lt; u ) then ( \bar{u} \leftarrow u' ) end if</td>
<td>if ( v' &lt; v ) then ( \bar{u} \leftarrow u' ) end if</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A5</th>
<th>A6</th>
<th>A7</th>
<th>A8</th>
</tr>
</thead>
<tbody>
<tr>
<td>if ( u' &gt; u ) then ( \bar{u} \leftarrow u' ) end if</td>
<td>if ( u' &gt; v ) then ( \bar{u} \leftarrow u' ) end if</td>
<td>if ( v' &gt; u ) then ( \bar{u} \leftarrow u' ) end if</td>
<td>if ( v' &gt; v ) then ( \bar{u} \leftarrow u' ) end if</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
</tr>
</thead>
<tbody>
<tr>
<td>if ( u' &lt; u ) then ( \bar{v} \leftarrow v' ) end if</td>
<td>if ( u' &lt; v ) then ( \bar{v} \leftarrow v' ) end if</td>
<td>if ( v' &lt; u ) then ( \bar{v} \leftarrow v' ) end if</td>
<td>if ( v' &lt; v ) then ( \bar{v} \leftarrow v' ) end if</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B5</th>
<th>B6</th>
<th>B7</th>
<th>B8</th>
</tr>
</thead>
<tbody>
<tr>
<td>if ( u' &gt; u ) then ( \bar{v} \leftarrow v' ) end if</td>
<td>if ( u' &gt; v ) then ( \bar{v} \leftarrow v' ) end if</td>
<td>if ( v' &gt; u ) then ( \bar{v} \leftarrow v' ) end if</td>
<td>if ( v' &gt; v ) then ( \bar{v} \leftarrow v' ) end if</td>
</tr>
</tbody>
</table>

6
(e) [4 points] Complete the algorithm below, which is a modification of the alpha-beta pruning algorithm, to work in the original setting where the ghost is friendly unbeknownst to Pacman. We want to compute Value(Root Node, $\alpha = -\infty, \beta = +\infty$. You should not prune on equality. Hint: you might not need to use $\alpha$ or $\beta$, or none of them (e.g. no pruning is possible).

```
function VALUE(state, $\alpha$, $\beta$
    if state is leaf then
        $(u, v) \leftarrow \text{UTILITY}(state), \text{UTILITY}(state))$
        return $(u, v)$
    end if
    if state is Ghost-Node then
        return GHOST-VALUE(state, $\alpha$, $\beta$
    else
        return PACMAN-VALUE(state, $\alpha$, $\beta$
    end if
end function
```

```
function GHOST-VALUE(state, $\alpha$, $\beta$
    $(u, v) \leftarrow (+\infty, -\infty)$
    for successor in SUCCESSORS(state) do
        $(u', v') \leftarrow \text{VALUE}(successor, $\alpha$, $\beta$)
        $(u, v) \leftarrow (u, v)$
    end for
    return $(u, v)$
end function
```

```
function PACMAN-VALUE(state, $\alpha$, $\beta$
    $(u, v) \leftarrow (-\infty, +\infty)$
    for successor in SUCCESSORS(state) do
        $(u', v') \leftarrow \text{VALUE}(successor, $\alpha$, $\beta$)
        $(u, v) \leftarrow (u, v)$
    end for
    return $(u, v)$
end function
```

Complete the pseudocode by choosing the option that fills in each blank above. The code blocks $C_1 - C_8$ prune the search (options for (i) and (iii)) and the code blocks $D_1 - D_8$ update $\alpha$ and $\beta$ (options for (ii) and (iv)). If any of the code blocks are not needed, choose ‘None’.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>if $u &lt; \alpha$ then return $(u, v)$ end if</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_2$</td>
<td>if $v &lt; \alpha$ then return $(u, v)$ end if</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_3$</td>
<td>if $u &lt; \beta$ then return $(u, v)$ end if</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_4$</td>
<td>if $v &lt; \beta$ then return $(u, v)$ end if</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_5$</td>
<td>if $u \geq \alpha$ then return $(u, v)$ end if</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_6$</td>
<td>if $v \geq \alpha$ then return $(u, v)$ end if</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_7$</td>
<td>if $u \geq \beta$ then return $(u, v)$ end if</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_8$</td>
<td>if $v \geq \beta$ then return $(u, v)$ end if</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_1$</td>
<td>$\alpha \leftarrow \min(\alpha, u)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td>$\alpha \leftarrow \min(\alpha, v)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_3$</td>
<td>$\beta \leftarrow \min(\beta, u)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_4$</td>
<td>$\beta \leftarrow \min(\beta, v)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_5$</td>
<td>$\alpha \leftarrow \max(\alpha, u)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_6$</td>
<td>$\alpha \leftarrow \max(\alpha, v)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_7$</td>
<td>$\beta \leftarrow \max(\beta, u)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_8$</td>
<td>$\beta \leftarrow \max(\beta, v)$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Q5. MedianMiniMax (Grads only) [20 points]

You’re living in utopia! Despite living in utopia, you still believe that you need to maximize your utility in life, other people want to minimize your utility, and the world is a zero-sum game. But because you live in utopia, a benevolent social planner occasionally steps in and chooses an option that is a compromise. Essentially, the social planner (represented as the pentagon) is a median node that chooses the successor with median utility. Your struggle with your fellow citizens can be modelled as follows:

There are some nodes that we are sometimes able to prune. In each part, mark all of the terminal nodes such that there exists a possible situation for which the node can be pruned. In other words, you must consider all possible pruning situations. Assume that evaluation order is left to right and all Vi’s are distinct. Provide explanations of your choices.

Note that as long as there exists ANY pruning situation (does not have to be the same situation for every node), you should mark the node as prunable. Also, alpha-beta pruning does not apply here, simply prune a sub-tree when you can reason that its value will not affect your final utility.

<table>
<thead>
<tr>
<th>Part a.</th>
<th>(1) $V_1$</th>
<th>(2) $V_2$</th>
<th>(3) $V_3$</th>
<th>(4) $V_4$</th>
<th>(5) None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part b.</td>
<td>(1) $V_5$</td>
<td>(2) $V_6$</td>
<td>(3) $V_7$</td>
<td>(4) $V_8$</td>
<td>(5) None</td>
</tr>
<tr>
<td>Part c.</td>
<td>(1) $V_9$</td>
<td>(2) $V_{10}$</td>
<td>(3) $V_{11}$</td>
<td>(4) $V_{12}$</td>
<td>(5) None</td>
</tr>
<tr>
<td>Part d.</td>
<td>(1) $V_{13}$</td>
<td>(2) $V_{14}$</td>
<td>(3) $V_{15}$</td>
<td>(4) $V_{16}$</td>
<td>(5) None</td>
</tr>
</tbody>
</table>