人工智能 第二次作业

管仁阳 519021911058

第一题

1. 系统搜索算法不仅可以得到**最优解**,也可以得到**路径**；局部搜索算法适用于只需要得到**最优解**但是不关心**路径**的问题。
2. 局部搜索算法可以解决**状态空间较大、无限或连续**的问题，这些问题系统搜索算法无法解决。
3. 局部搜索算法适合解**单纯的优化问题**，这些问题并不适用于系统搜索算法的标准定义
4. 从**数据结构**上来说，局部搜索算法不需要保持一个搜索树，所以当前结点不需要保存**状态**和**目标函数**

第二题

(1)

$$\frac{\partial f(x,y,z)}{\partial x} = e^x(xy+y+2z)$$
$$\frac{\partial f(x,y,z)}{\partial y} = xe^x$$
$$\frac{\partial f(x,y,z)}{\partial x} = 2e^x$$

由于**梯度反方向是函数值局部下降最快的方向**，所以当$(x,y,z)=(0,1,-1)$时移动方向为$(1,0,-2)$

(2)

爬山法会遇到的问题：

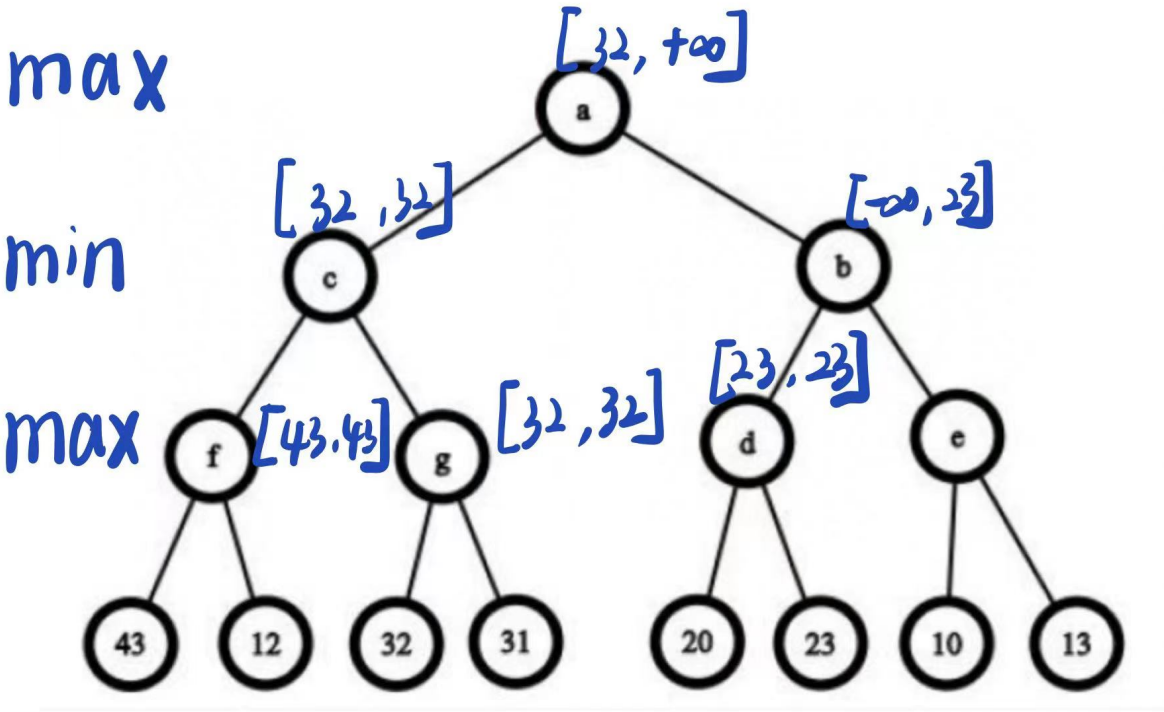
1. 局部最优解：爬山算法到了局部最优解之后就找不到下一个要更新的方向，故终止。最终困在局部最优解而得不到全局最优解。
2. “山脊”：爬山算法会被困在“山脊”位置，这使用贪心的算法很难跳出来。
3. “高原”：高原就是一段目标函数值相同的连续的状态空间。在这种情况下，如果允许移动到相邻的节点，就可以摆脱“高原“的束缚。但是如果这是个**局部最优**"高原"，又会导致死循环。我们可以通过**限制最大移动次数**来摆脱这个问题。

更好的替代方法：

1. Stochastic hill climbing：随机选择移动到使得目标函数增大的状态，而移动到每种状态的概率随着梯度的不同而变化。这种方法通常收敛更慢，但是特定情况下效果更好
2. First-choice hill climbing：随机生成后继状态，直到生成了一个优于当前状态的状态。这种方法在后继状态很多时候效果较好
3. Random-restart hill climbing：随机选择一些初始状态并从这些状态开始执行爬山算法，直到找到目标。

第三题

(1) a最终的值为 32



(2)

会访问 **12** 个结点，访问顺序为：**a-c-f-43-12-g-32-31-b-d-20-23**

第四题

(1) 先尝试给 X_3 赋值，因为它的度最大（参与最多约束）

(2) $X_1 \neq X_3, X_3 \neq X_4$弧双向相容，$X_2$关于$X_3$弧相容

(3)

步骤	需要检查的弧 Xi → Xj	Xi 值域的变化	添加进入搜索列的弧
0	X2 → X1	$D_2 = \{3, 4\}$	X3 → X2(事实上已在队列中)
1	X1 → X2	$D_1 = \{3, 4\}$	X2 → X1, X2 → X1(事实上已在队列中)
2	X3 → X2	$D_3 = \{2, 3, 5, 6\}$	X1 → X3(事实上已在队列中), X4 → X3(事实上已在队列中)
3	X2 → X3	无变化	无
4	X4 → X3	无变化	无
5	X3 → X4	无变化	无
6	X3 → X1	无变化	无
7	X1 → X3	无变化	无
...	...	...	...

第二次作业

提交 DDL：2021 年 11 月 4 日 0 时

作业完成形式有三种：
（1）你可以手写自己的解答并拍照，再将照片整理成一份 word/pdf 文件并提交。
（2）你可以使用 word 文档进行编辑，最后提交 word/pdf 文件。
（3）你可以使用 latex 进行编辑，最后提交 pdf 文件。
如果你没有在 DDL 之前提交作业，请及时在微信群里联系助教进行补交。如果对作业有任何问题，你可以在从微信里询问助教谢瑜璋，或者发邮件到 constantjxyz@sjtu.edu.cn。

1 本次作业可能用到的知识点

本次作业可能会用到以下知识点：
（1）局部搜索算法（例如爬山法）与系统搜索算法（例如我们之前讲过的 BFS、A* 算法）的定义、特点、复杂性、适用范围。
（2）爬山法的定义、限制性，代替爬山法的其他局部搜索算法。
（3）多元函数的求偏导方法，最优化问题的最速下降法。
（4）minimax 搜索树的定义、计算，$\alpha-\beta$ 剪枝法的计算过程。
（5）CSP 问题的定义，求解过程中选择变量的顺序，弧相容性（arc consistency）的定义，使用回溯法与 AC3 算法求解 CSP 问题的过程。

2 第一题

尝试比较局部搜索算法（例如爬山法）与系统搜索算法（例如宽度优先搜索、A* 算法）。

解答：（1）扩展新节点：在扩展节点时，系统搜索算法需要考虑当前节点在整个搜索空间中产生的所有可能的新状态，并将所有可能的新状态加入查询队列。但是局部搜索算法通常从当前状态出发，移动到附近的某个节点，并将改变后的新状态加入队列。
（2）路径：系统搜索算法一般需要储存从根节点到叶子结点的每一条路径，每一条路径可以视为问题的一个可能解。而局部搜索算法一般不存储路径，只关注叶子结点最终的状态。
（3）复杂性：系统搜索算法一般时间复杂性和空间复杂度都较高。对于相同规模的问题，局部搜索算法需要扩展的节点较少、且一般不需要储存路径，故一般时间复杂性和空间复杂性都要低一些。
（4）适用范围：系统搜索算法可以用于探索问题的所有解或者证明命题是否成立，而局部搜索算法一般用于最优化问题。

3 第二题

我们希望使用爬山法解决一些最优化问题。
（1）假设我们需要找 $f(x,y,z)=e^x(xy+2z)$ 的最小值，且当前状态下我们有 $(x,y,z)=(0,1,-1)$，那么我们需要将当前状态向怎样的方向进行移动、能够在理论上最快靠近极值？（仅说明移动方向即可，方向用三维元组表示即可，计算过程可以参考多元函数的偏导数求解、最速下降法）。

（2）使用爬山法搜索可能会遇到哪些问题？我们可以使用哪些更好的方法来代替？

解答：（1）在爬山法中，我们需要将当前状态向其“最好”的一个邻近状态移动。从数学上的角度来看，梯度的方向是函数增长速度最快的方向，那么梯度的反方向就是函数减少最快的方向。而在该问题中，由于我们需要求的是最小值，因此我们需要求解当前函数 $f(x,y,z)$ 的梯度，并且沿着其梯度的反向进行移动，这样理论上可以最快到达极小值点。（如果反过来，要求找到一个函数的最大值点，我们可以采用梯度上升法，将物体向其梯度方向移动）。

$$\frac{\partial f}{\partial x}=\frac{\partial f}{\partial e^x}\cdot\frac{\partial e^x}{\partial x}+\frac{\partial f}{\partial(xy+2z)}\cdot\frac{\partial(xy+2z)}{\partial x}=(xy+2z)\cdot e^x+e^x\cdot y$$

$$\frac{\partial f}{\partial y}=e^x\cdot x$$

$$\frac{\partial f}{\partial z}=2e^x$$

代入 $(x,y,z)=(0,1,-1)$ 可以得到当前状态的梯度为 $(-1,0,2)$。故根据最速下降的原理，应当取的方向是梯度的反向，即 $(1,0,-2)$。

（2）爬山法是否能够找到最优解，非常依赖于给定的初始状态。除此以外，爬山法在搜索过程中可能会遇到局部极值、山肩、高原等问题，从而影响其找到最优解。

改进方法：可以使用随机重启爬山法（见课件上 random starts 部分），它通过多次重启生成随机的初始状态来保证算法的完备性；可以使用模拟退火方法（见可见上 simulated annealing 部分）：允许以一定的概率产生一些随机的后继节点（而不是总是挑选最优的后继节点）。

4 第三题

我们的 minimax 搜索树如图1所示。

（1）假如我们的 **a** 节点是 **max** 节点，请问最后 **a** 节点会得到怎样的值？

（2）假如我们使用 $\alpha-\beta$ 剪枝法进行 minimax 树的搜索，搜索过程中会从左至右访问相关节点，且 **a** 节点是 **max** 节点。算法运行过程中会访问多少个节点（包括字母标号的节点与数字标号的叶节点、忽略重复访问）？同时，请写下各节点的访问顺序（例如顺序：**”a - c - f - 43”**）。

解答：（1）**a** 层是 **max** 节点，**c**、**b** 是 min 节点，下一层的 **f**、**g**、**d**、**e** 是 max 节点。通过计算可知，**f** 为 43，**g** 为 32，由此可得 **c** 为 32。而在另一侧，**d** 为 23，**e** 为 13，**b** 为 13。最后比较 **c** 与 **b** 可知 **a** 为 32。

（2）使用 $\alpha-\beta$ 剪枝法进行 minmax 树的搜索，过程中我们需要维护 α（当前搜索路径上目前 max 得到的最好结果）与 β（当前搜索路径上 min 的最好结果），并且在适当的时候剪枝。在运行到 min 节点 **n** 时，会检查当前节点下子树返回的值与当前节点的 α 值，如果子树返回的值小于 α 值，说明该 min 节点遍历所有子树后得到的值一定小于 α 值，可以断定在第 **n-1** 步 max 做决定的时候不会让 min 有这种机会。因为在 **n-1** 步，上层的 max 就知道存在着一个策略 D，使得 value 值为 ，max 可以忽略 **n** 所在的子树。同理，在 max 节点，我们会检查子树返回的值与当前节点的 β 值，如果大于 β 即剪去该节点与其子树。

在搜索过程中，刚开始的过程与常规的 minmax 树相同，我们用类似于 DFS 的算法对搜索树进行访问。我们会依次访问 **a - c - f - 43 - 12 - g - 32 - 31**。此时我们可知 **e** 节点的值为 32，由此在 **a** 节点处，其维护的 α 值为 32。之后我们会再访问 **b**、**d**、**20**、**23** 节点，此时我们可以推定 **d** 节点的值为 23。那么当我们重新返回 **b** 节点时，可知 **b** 节点的取值必须小于等于 23，也就是小于该节点得到的 α 值。那么此时 **b** 节点所在的整个分枝会被剪掉，算法直接返回 **b** 的上一层，也就是 **a** 节点。

算法总共会访问 12 个节点，顺序为 **a - c - f - 43 - 12 - g - 32 - 31 - b - d - 20 - 23**。

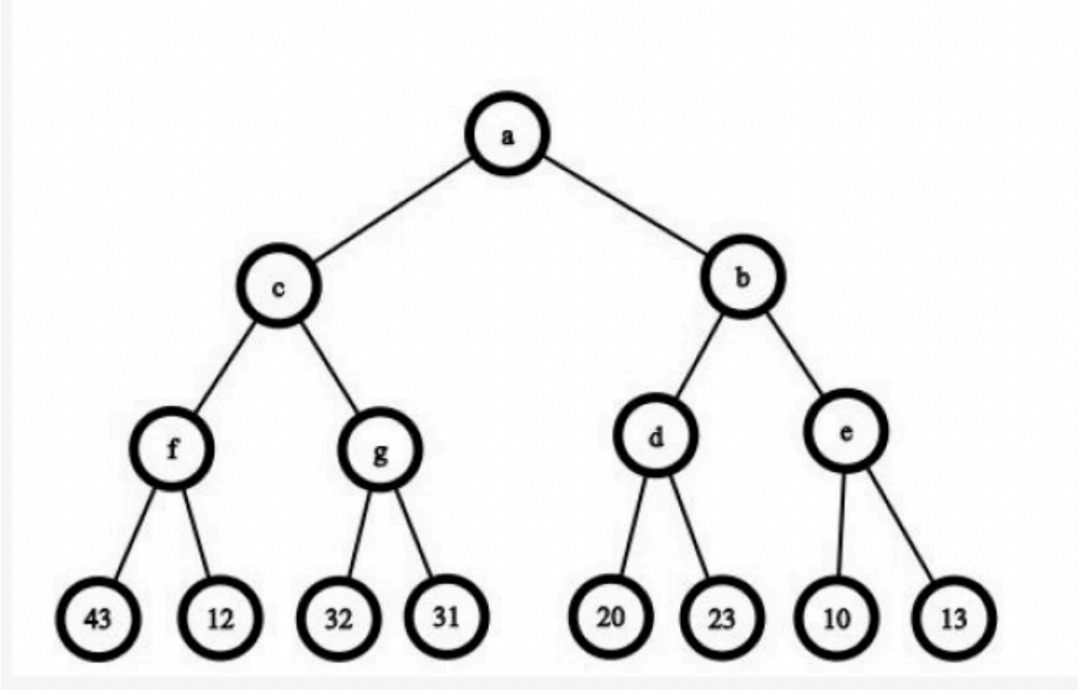


图 1: 第三题的对抗搜索树

5 第四题

考虑一个这样的 CSP 问题：我们需要给变量X_1, X_2, X_3, X_4 赋值，需要满足以下约束：$(a)X_1 \geq X_2, (b)X_2 > X_3$ *or* $X_3 - X_2 = 2, (c)X_3 \neq X_4, (d)X_1 \neq X_3$。

（1）根据 CSP 问题赋值求解的 Most constraining variable 规则，我们应该最先尝试给哪个

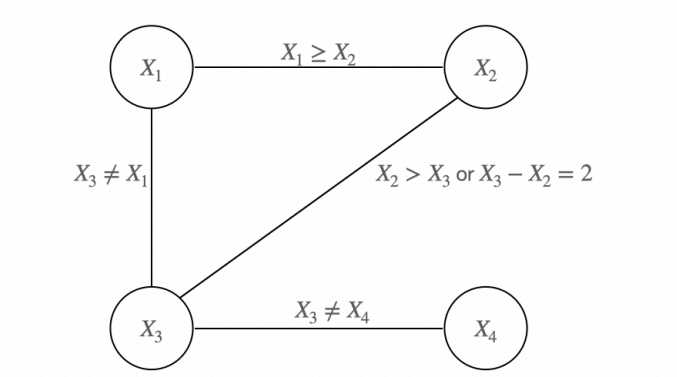


图 2: 第四题的 csp 问题

变量赋值?

(2)假如我们规定变量变量X_1, X_2, X_3, X_4 的值域分别为 $D_1 = \{1, 2, 3, 4\}, D_2 = \{3, 4, 5, 8, 9\}, D_3 = \{2, 3, 5, 6, 7, 9\}, D_4 = \{3, 5, 7, 8, 9\}$。请问变量$X_1, X_2, X_3, X_4$ 的哪些弧满足弧相容性（arc consistency）?

（3）我们对该 CSP 问题在当前状态下运行 AC3 算法，请完成下方表格的步骤 1-7。

初始的搜索列：$\{X_2 \to X_1, X_1 \to X_2, X_3 \to X_2, X_2 \to X_3, X_4 \to X_3, X_3 \to X_4, X_3 \to X_1, X_1 \to X_3\}$。

步骤	需要检查的弧 $X_i \to X_j$	X_i 值域的变化	添加进入搜索列的弧
0	$X_2 \to X_1$	D_2变为$\{3,4\}$	无
1	$X_1 \to X_2$		
2	$X_3 \to X_2$		
3	$X_2 \to X_3$		
4	$X_4 \to X_3$		
5	$X_3 \to X_4$		
6	$X_3 \to X_1$		
7	$X_1 \to X_3$		
...	...	...	...

解答：（1）根据 Most constraining variable 规则，我们应该选择与其他变量约束最多的变量进行第一次赋值。而在我们的题目中，变量X_3 与X_2, X_1, X_4 都有约束关系，因此我们应选择先给变量X_3 赋值。

（2）弧相容：如果一条弧 $X \to Y$ 满足 X 值域中的每一个可能的取值，Y 中都能有相应的合法取值满足约束，则称该弧满足弧相容性，注意弧相容的定义时单向的。

根据定义，我们本题中满足相容性的弧有 $X_2 \to X_3, X_1 \to X_3, X_3 \to X_1, X_3 \to X_4, X_4 \to X_3$。当 X_1 取 1 或者 2 时，X_2 中没有符合约束的取值；当 X_2 取 5 或者 8 或者 9 时，X_1 中没有符合约束的取值；当 X_3 取 9 时，X_2 没有符合约束的取值。故 $X_1 \to X_2, X_2 \to X_1, X_3 \to X_2$ 不满足弧相容性。

有部分同学还写了诸如 $X_1 \to X_4$ 之类的弧，但是我们从该 csp 问题的约束图中可知，两变量之间没有直接的约束关系，我们一般不将其视为弧。

（3）

需要检查的弧 $X_i \to X_j$	X_i 值域的变化	添加进入搜索列的
$X_2 \to X_1$	D_2变为$\{3,4\}$	-
$X_1 \to X_2$	D_1变为$\{3,4\}$	$X_2 \to X_1$
$X_3 \to X_2$	D_3变为$\{2,3,5,6\}$	-
$X_2 \to X_3$	-	-
$X_4 \to X_3$	-	-
$X_3 \to X_4$	-	-
$X_3 \to X_1$	-	-
$X_1 \to X_3$	-	-
...	...	...

注意几个要点：（a）每个约束条件都需要形成两条弧，我们要分别检查其相容性（b）对于一条弧 $X_i \to X_j$ ，我们需要满足的条件是对于 X_i 值域中的**任意**值，X_j 值域中都**存在**符合约束的值（c）在完成检查弧 $X_i \to X_j$ 后，如果 X_i 值域有变化，需要添加指向 X_i 的弧进入搜索队列，例如 $X_k \to X_i$ 将被加入搜寻队列，但是 $X_i \to X_k$ 无需加入。并且如果需要新添加的弧已经在搜索队列中了，就无须再次添加（d）在检查 $X_1 \to X_2$ 的时候，由于 D_1 发生了变化，所以要将以 X_1 为终点的弧加入搜索列中，由于 $x_1 \to x_2$ 已经被检查过了，故已经从搜索列中被去除，需要重新添加进入搜索列。