

BUSS 3620.人工智能导论

优化

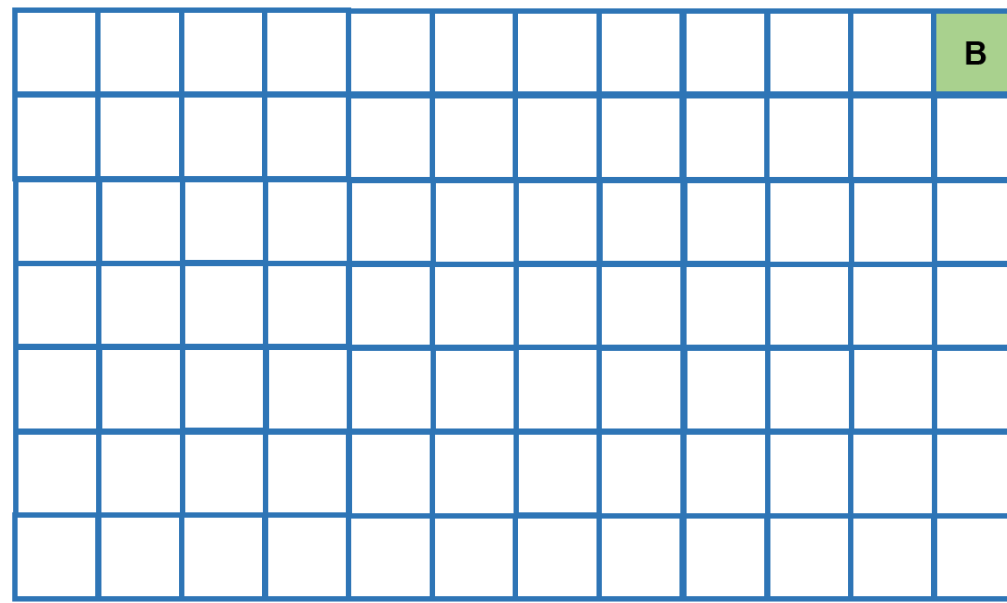
刘佳璐

安泰经济与管理学院

上海交通大学

过去几周学习了

- 搜索
 - 目标很明显
 - 找到问题的解
- 不确定性
 - 真实状态是不确定的
 - 根据观测值推测真实值
- 接下来会学习
 - 目标在哪里?
 - 找到目标(最优解)



优化

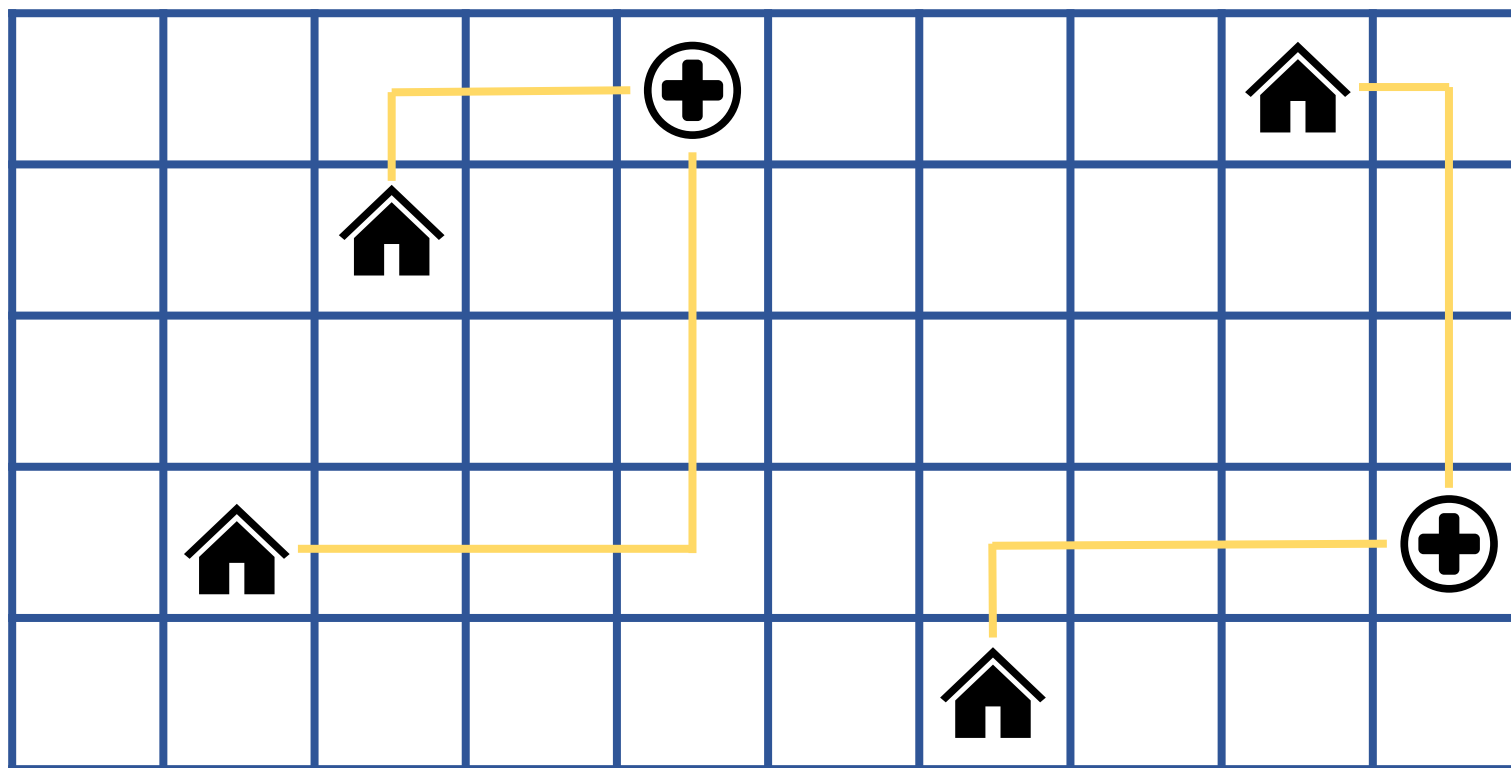
- 从一组选项(解)中选择**最佳**选项(解)



- 例：社区医院选址

- 建造两个社区医院

- 尽量减少从每个住宅区到社区医院的距离



BUSS 3620.人工智能导论

#1. 局部搜索

刘佳璐

安泰经济与管理学院

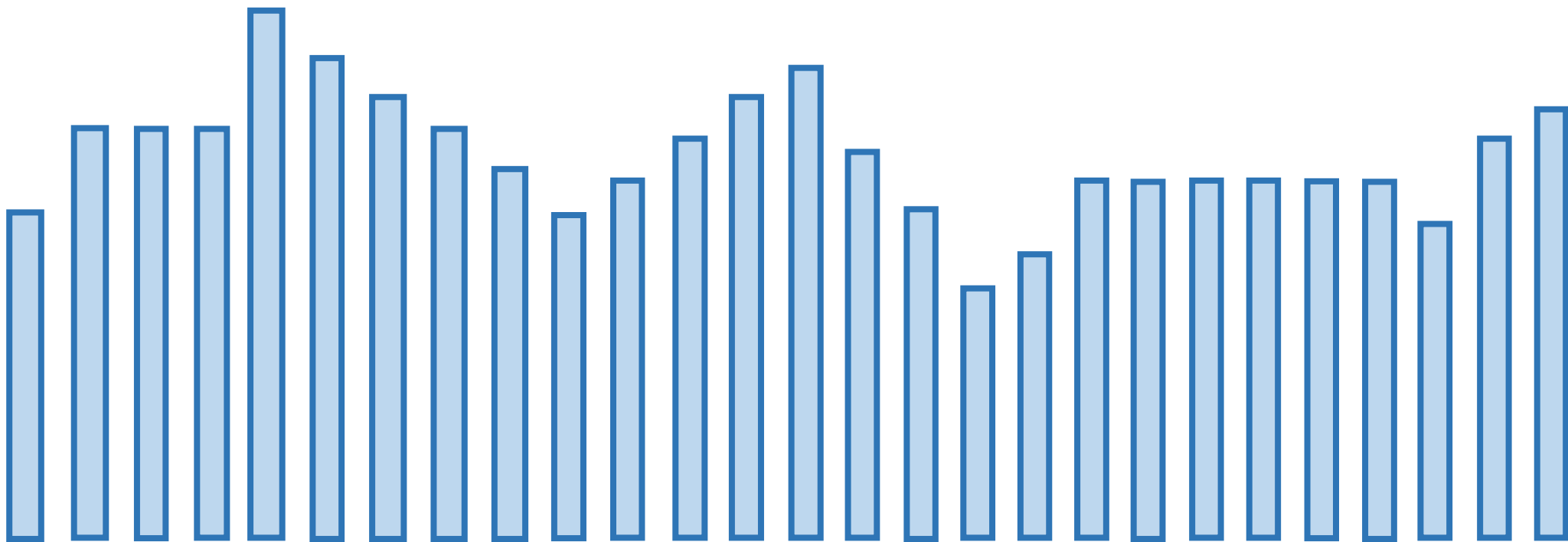
上海交通大学

优化

- 局部搜索 Local search
 - 从~~一个~~节点(解)出发，然后搜索~~相邻~~的节点(解)进行迭代的搜索算法

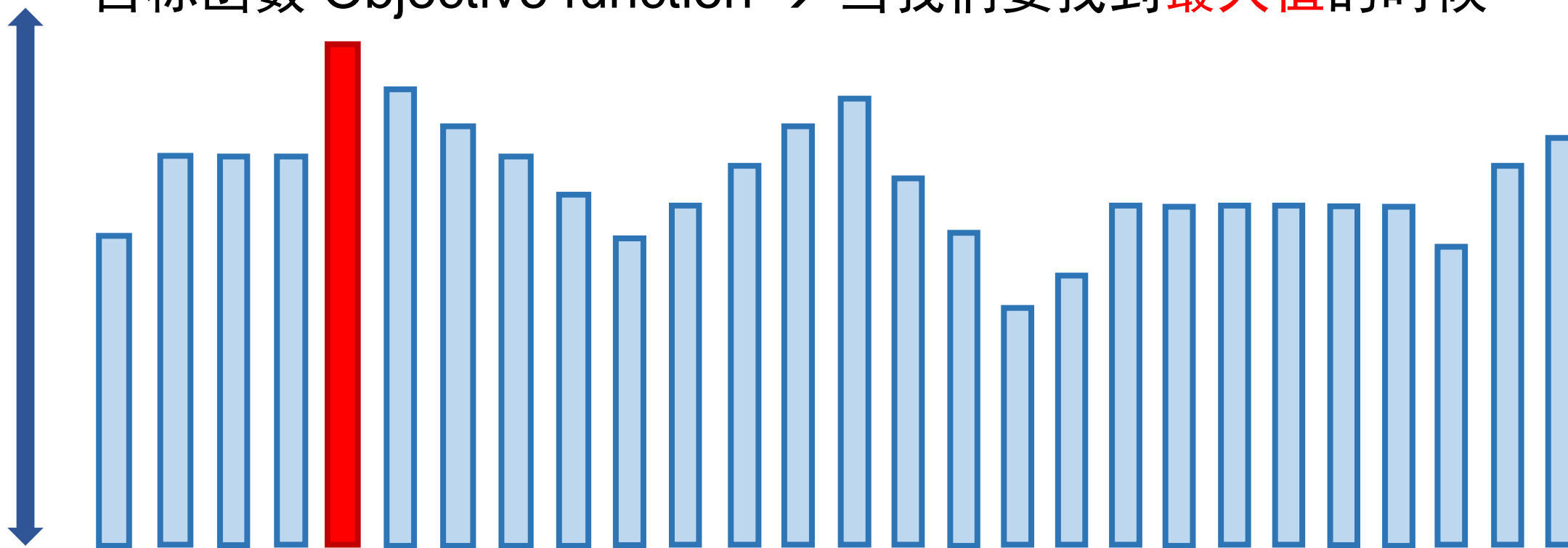
状态空间图 State-space landscape

- 每个柱子指代一个状态
 - 柱子的高度指代一个状态的某个值（例如，距离）



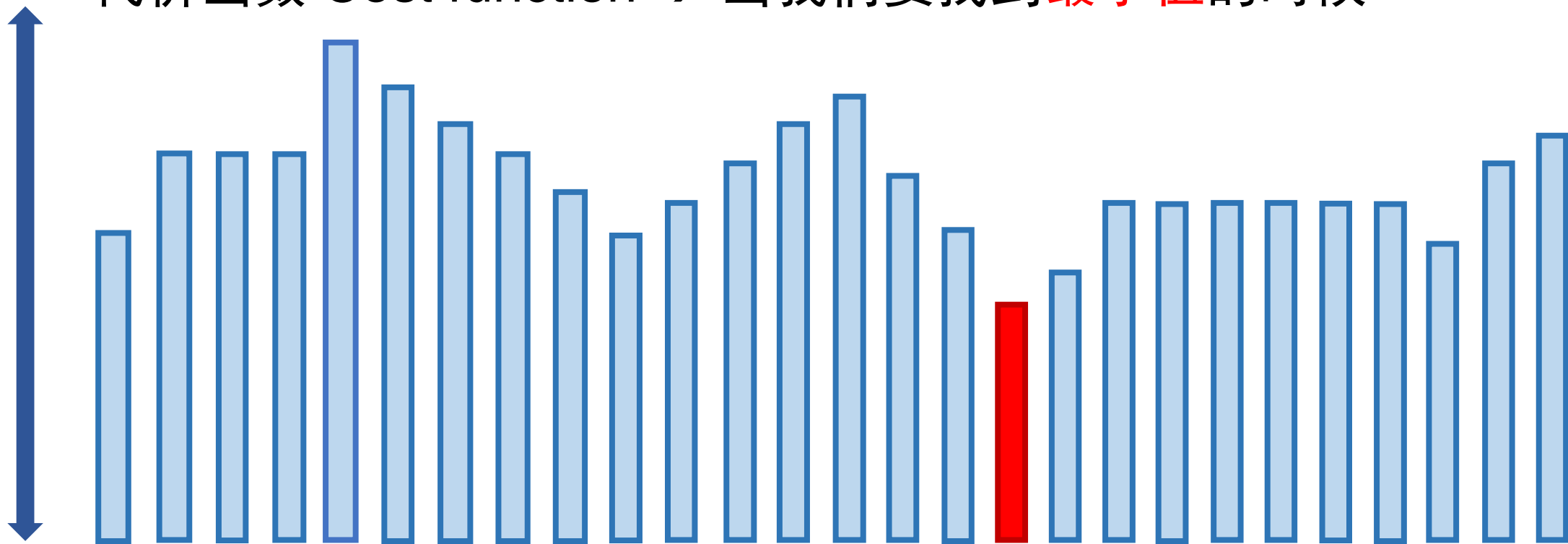
全局最大值 Global maximum

- 一个状态的值高于所有其他状态的值
- 目标函数 Objective function → 当我们要找到**最大值**的时候



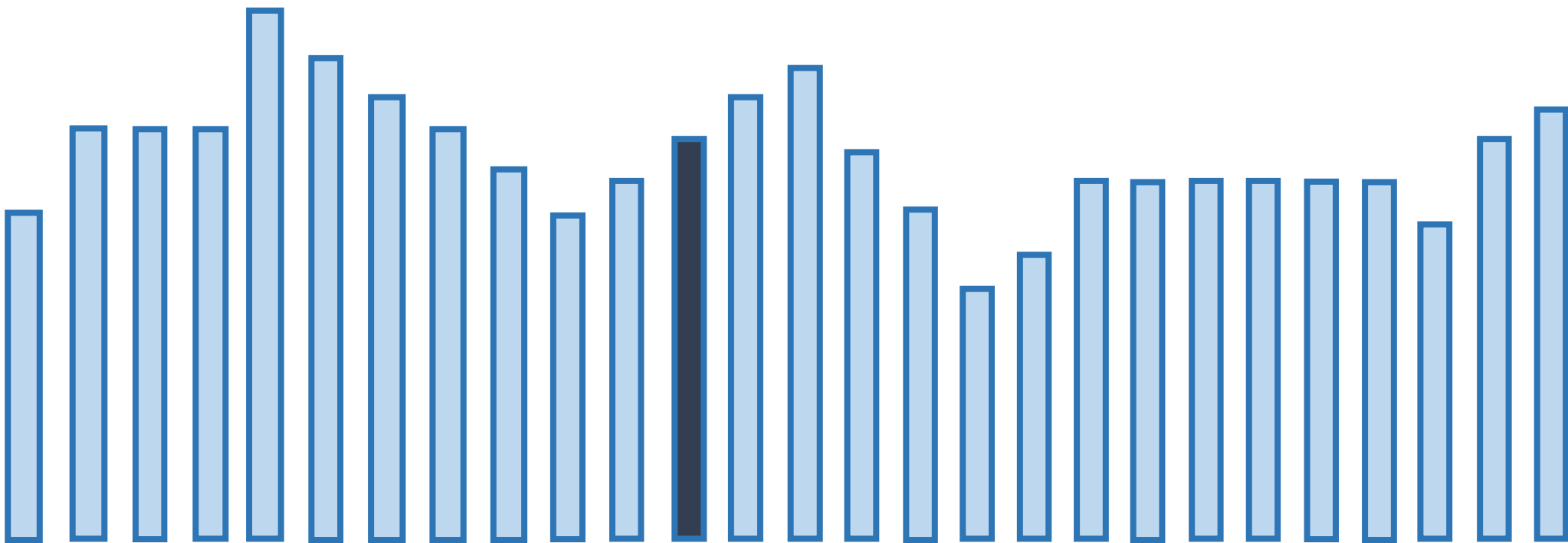
全局最小值 Global minimum

- 一个状态的值小于所有其他状态的值
- 代价函数 Cost function → 当我们要找到**最小值**的时候



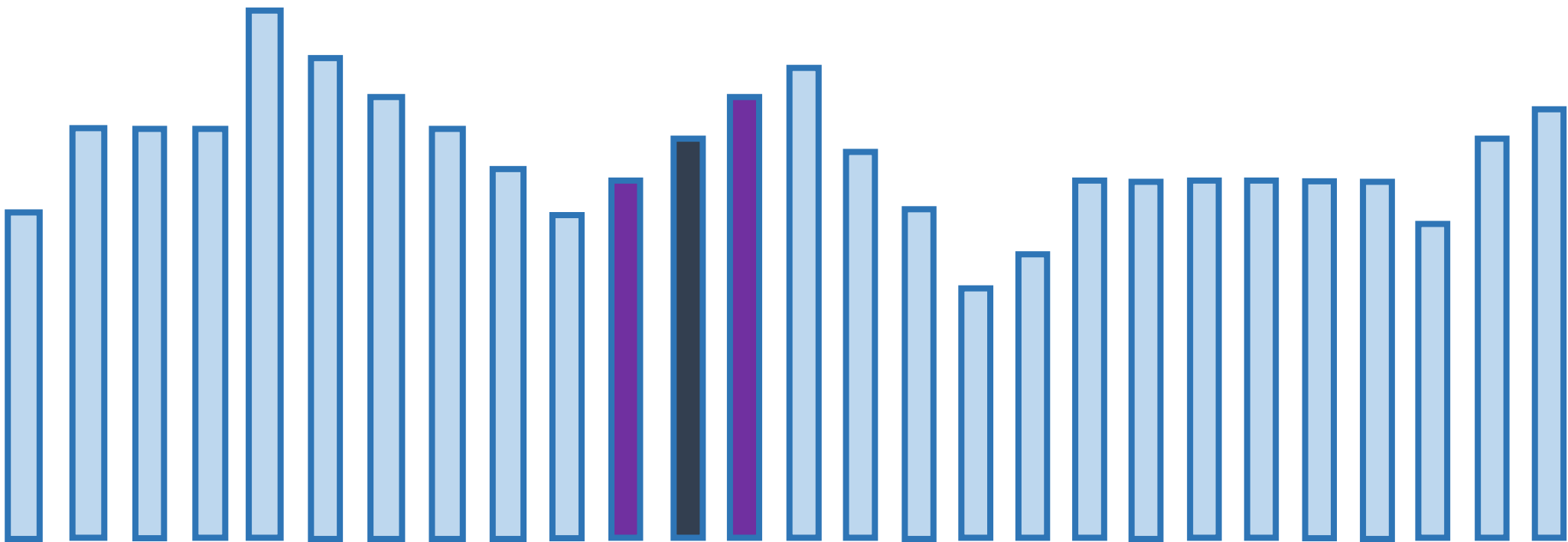
局部搜索 Local search

- 当前状态 Current state
 - 当前正在考虑的状态



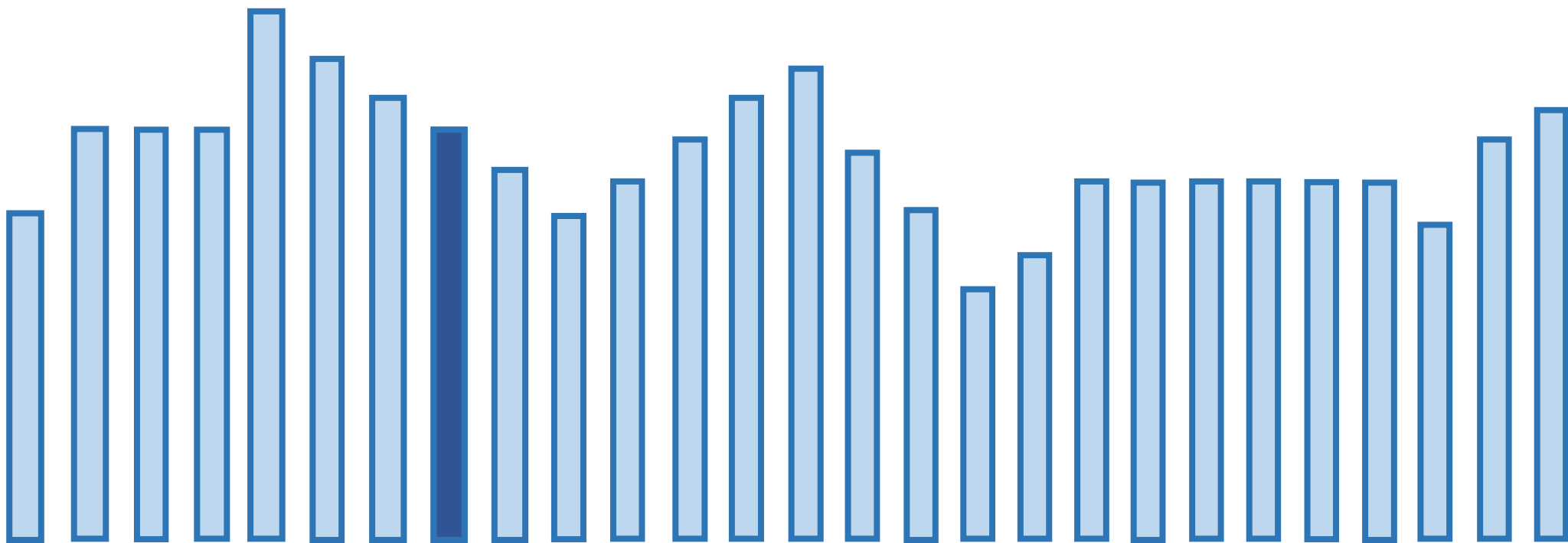
局部搜索 Local search

- 相邻状态 Neighbor state
 - 当前状态可以转换到的状态



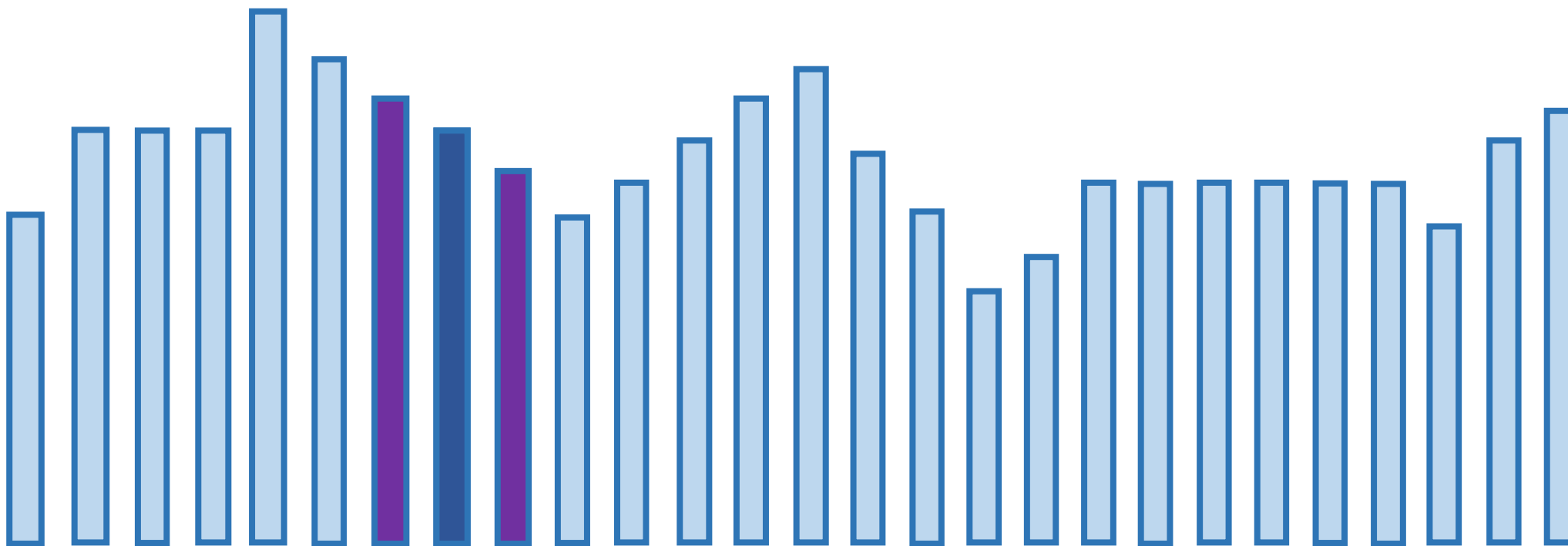
爬山算法 Hill Climbing – 找全局最大

- 从一个状态开始



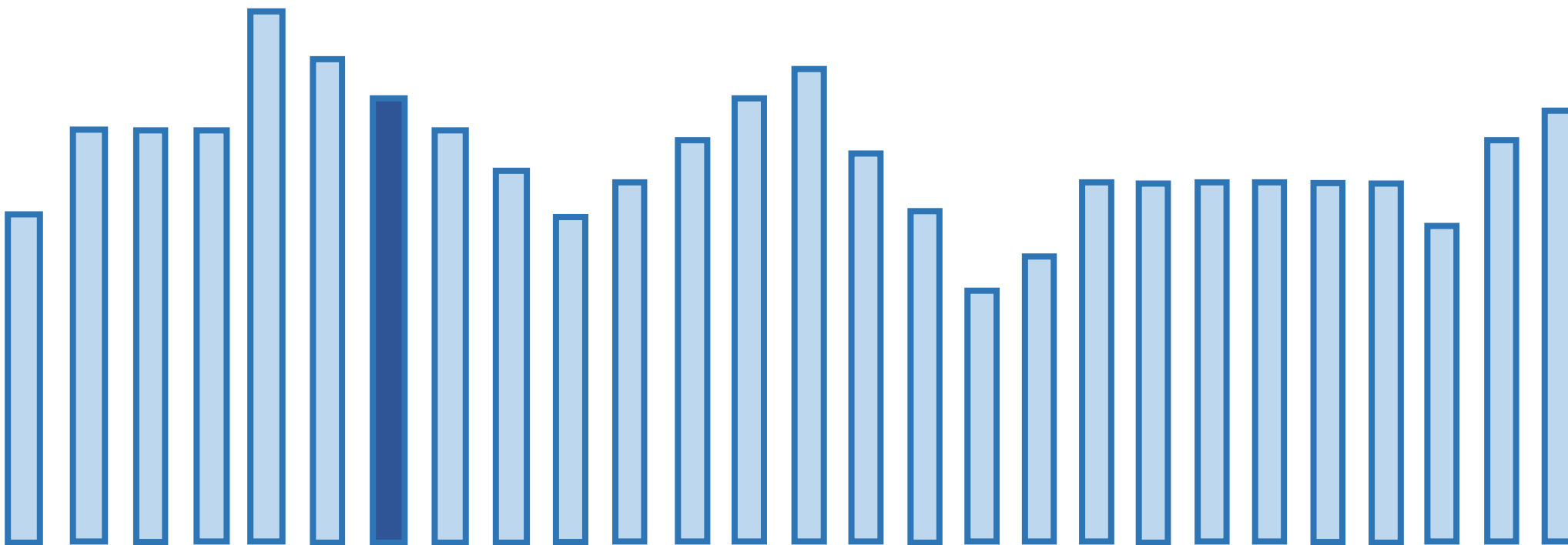
爬山算法 Hill Climbing – 找全局最大

- 比较这个状态的相邻状态



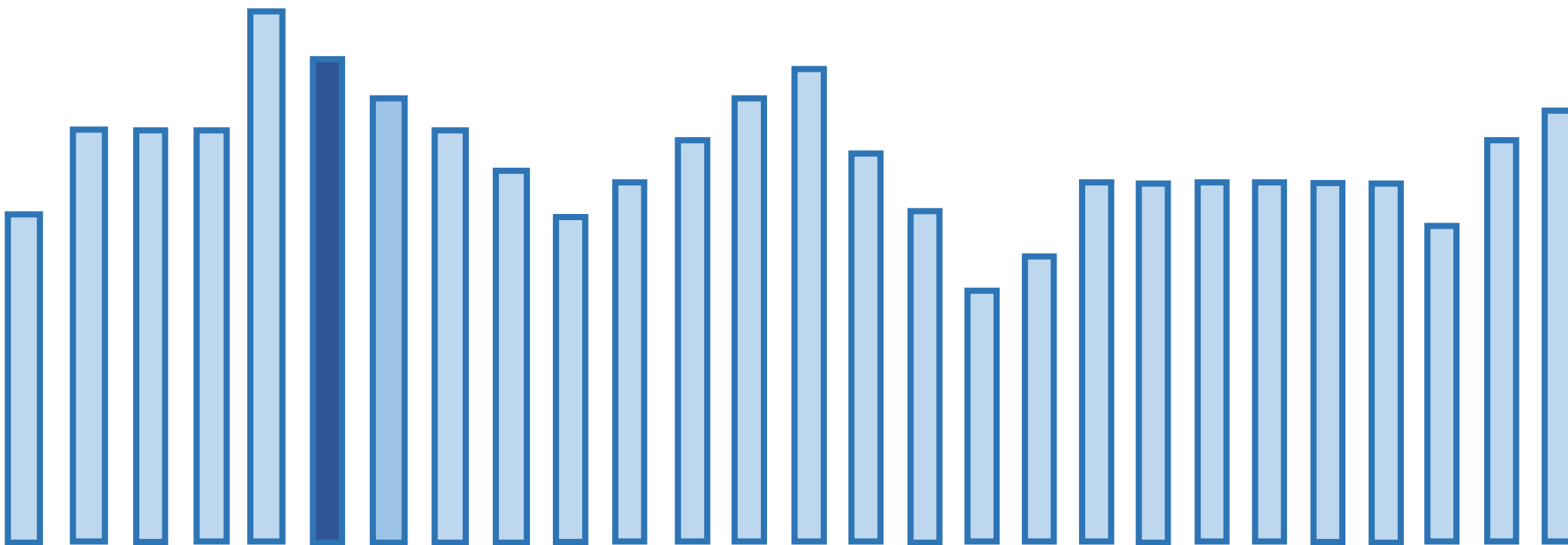
爬山算法 Hill Climbing – 找全局最大

- 选择值**更高**的相邻状态，并移动过去



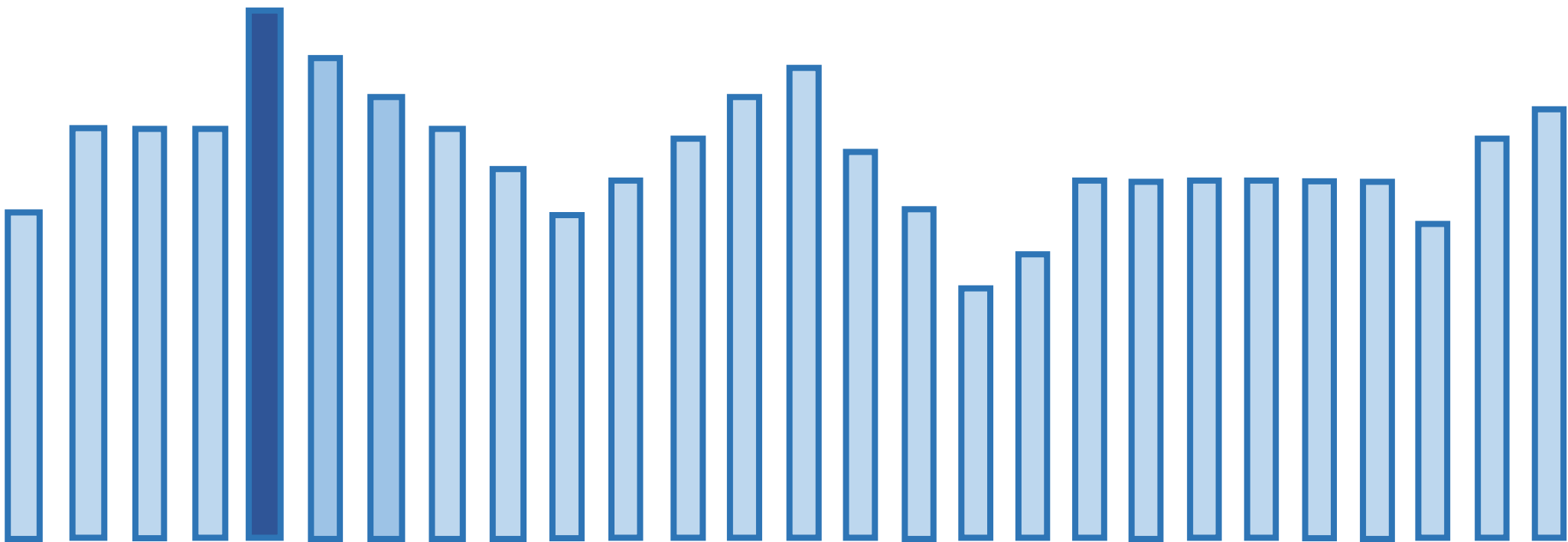
爬山算法 Hill Climbing – 找全局最大

- 选择值**更高**的相邻状态，并移动过去



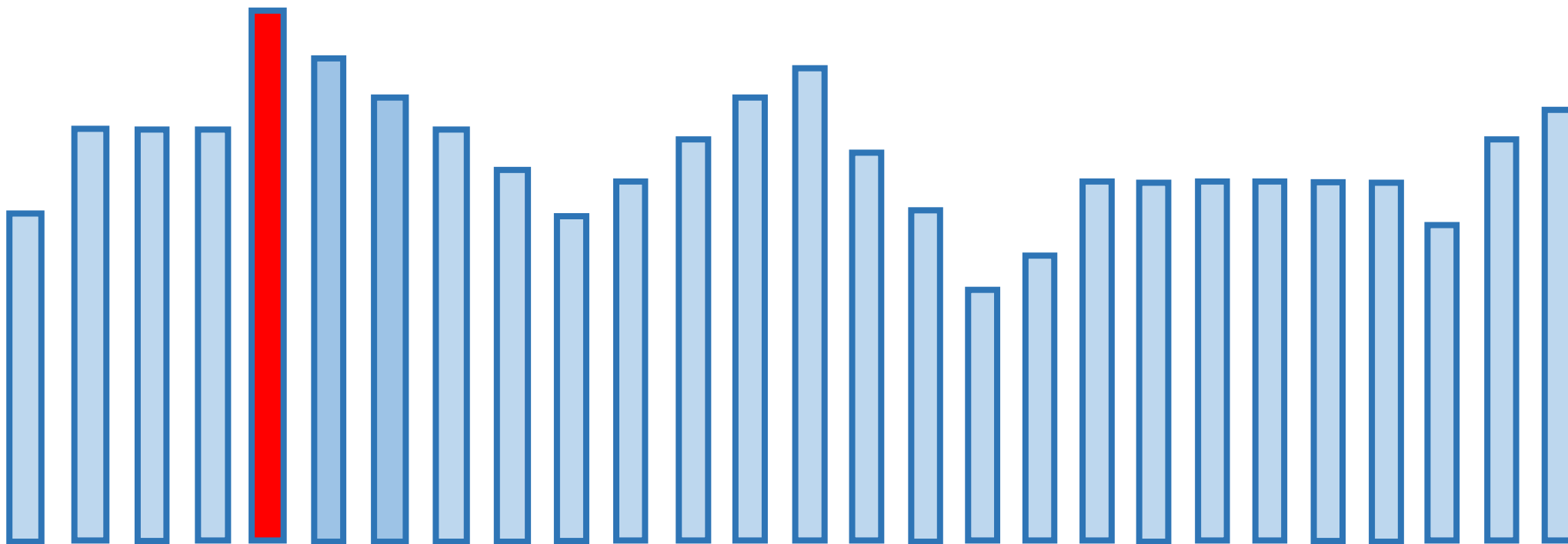
爬山算法 Hill Climbing – 找全局最大

- 选择值**更高**的相邻状态，并移动过去



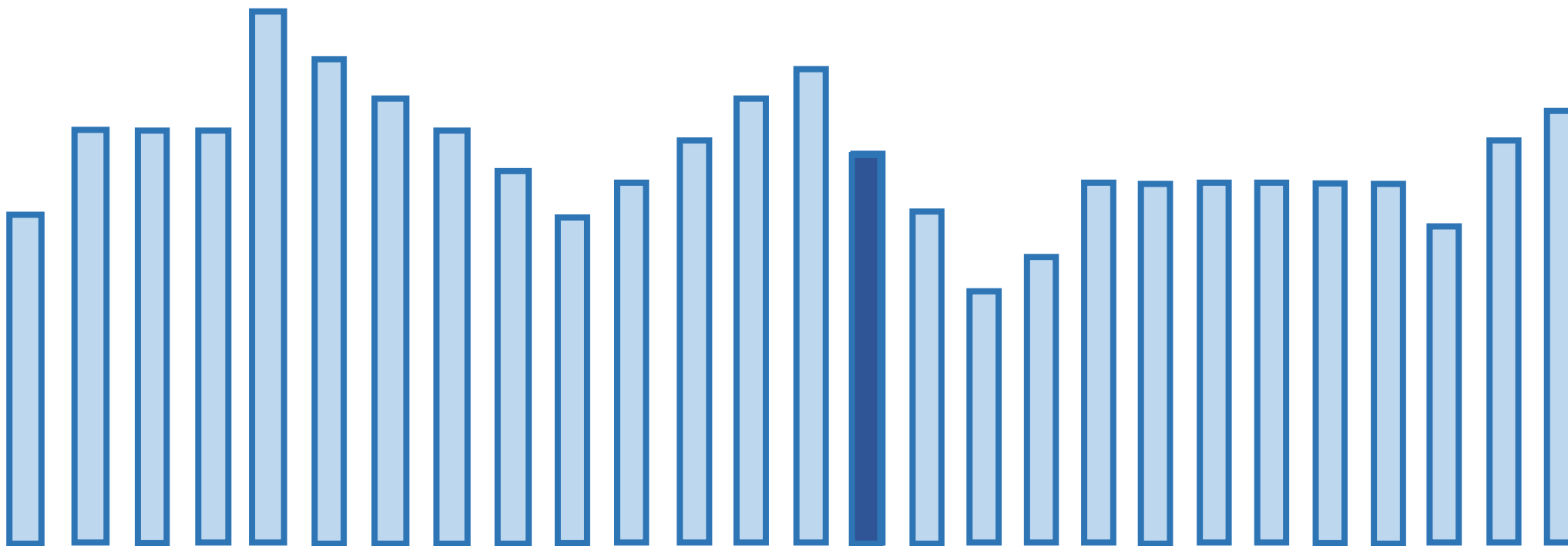
爬山算法 Hill Climbing – 找全局最大

- 直到这个状态周围的相邻状态的值都更小



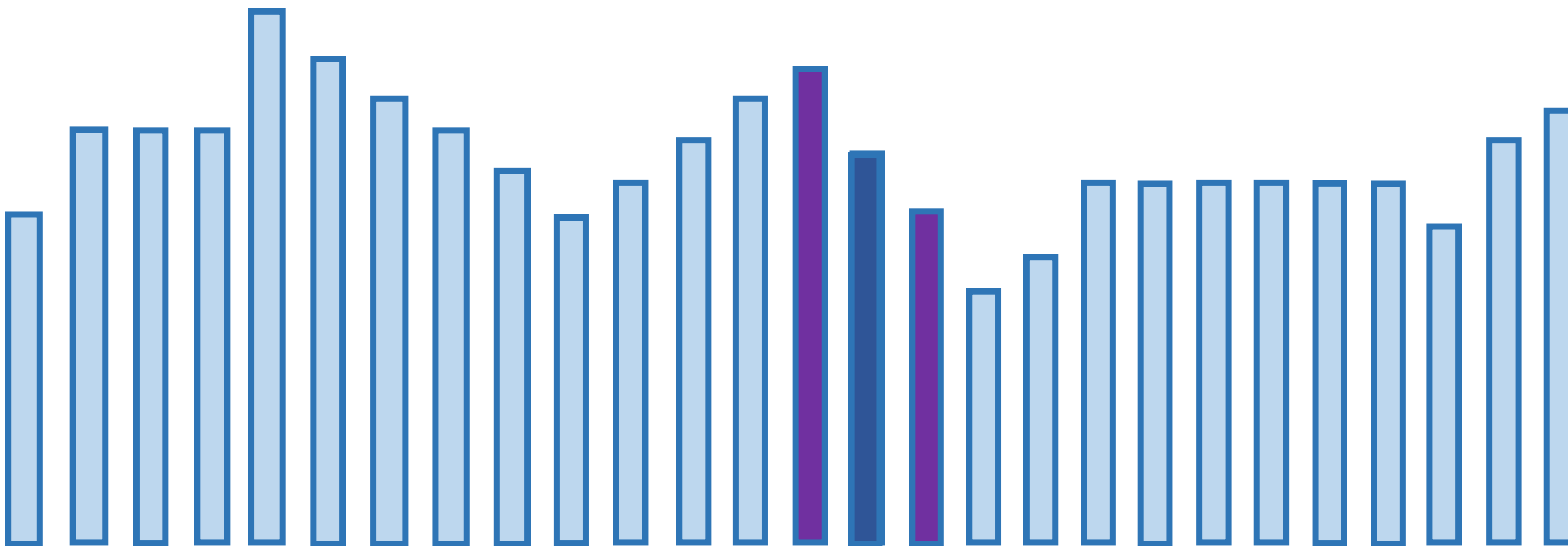
爬山算法 Hill Climbing – 找全局最小

- 从一个状态开始



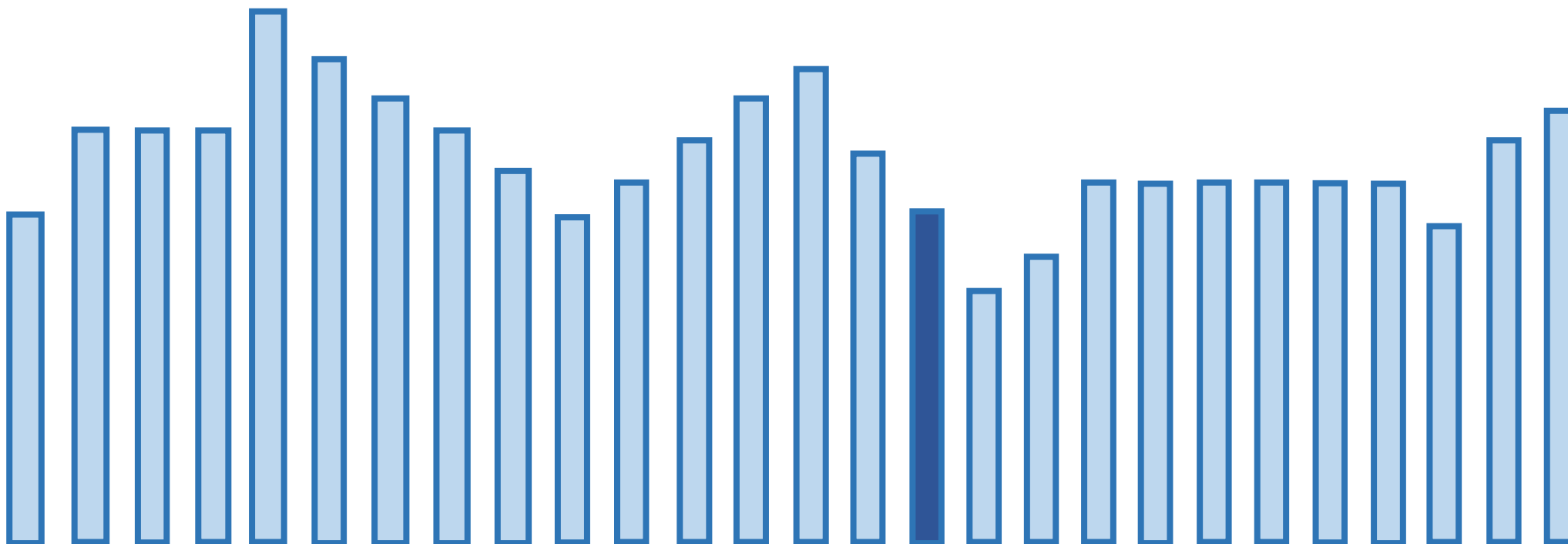
爬山算法 Hill Climbing – 找全局最小

- 比较这个状态的相邻状态



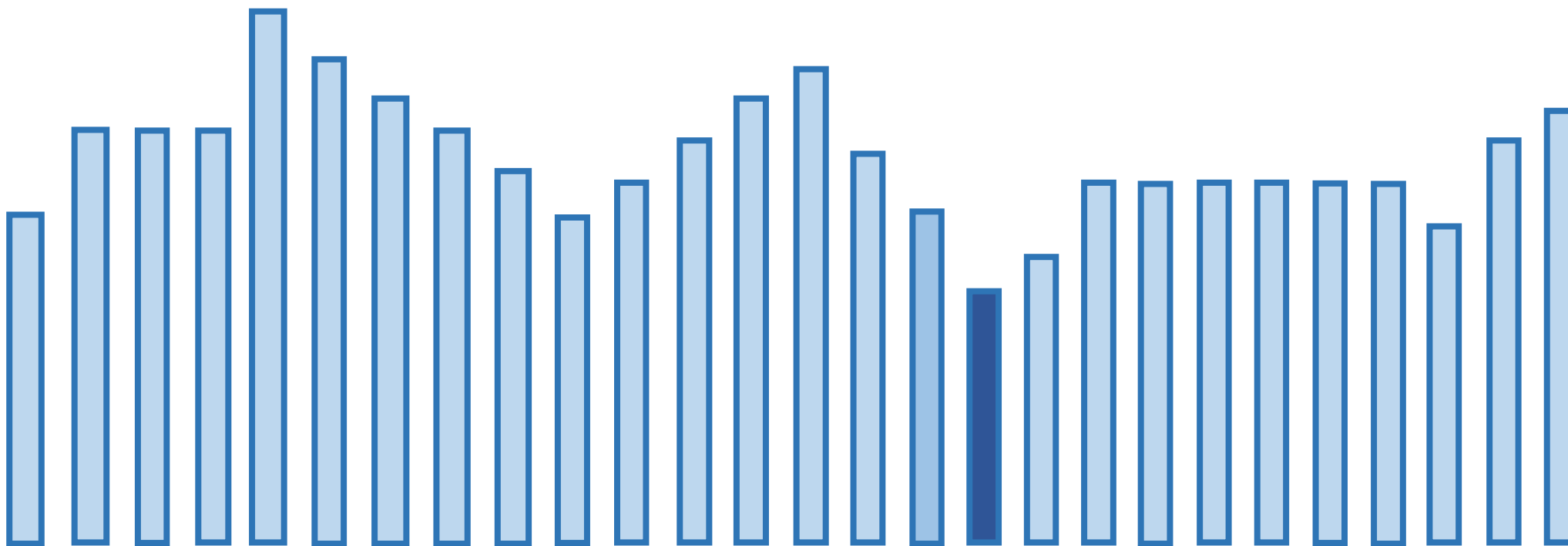
爬山算法 Hill Climbing – 找全局最小

- 选择值**更小**的相邻状态，并移动过去



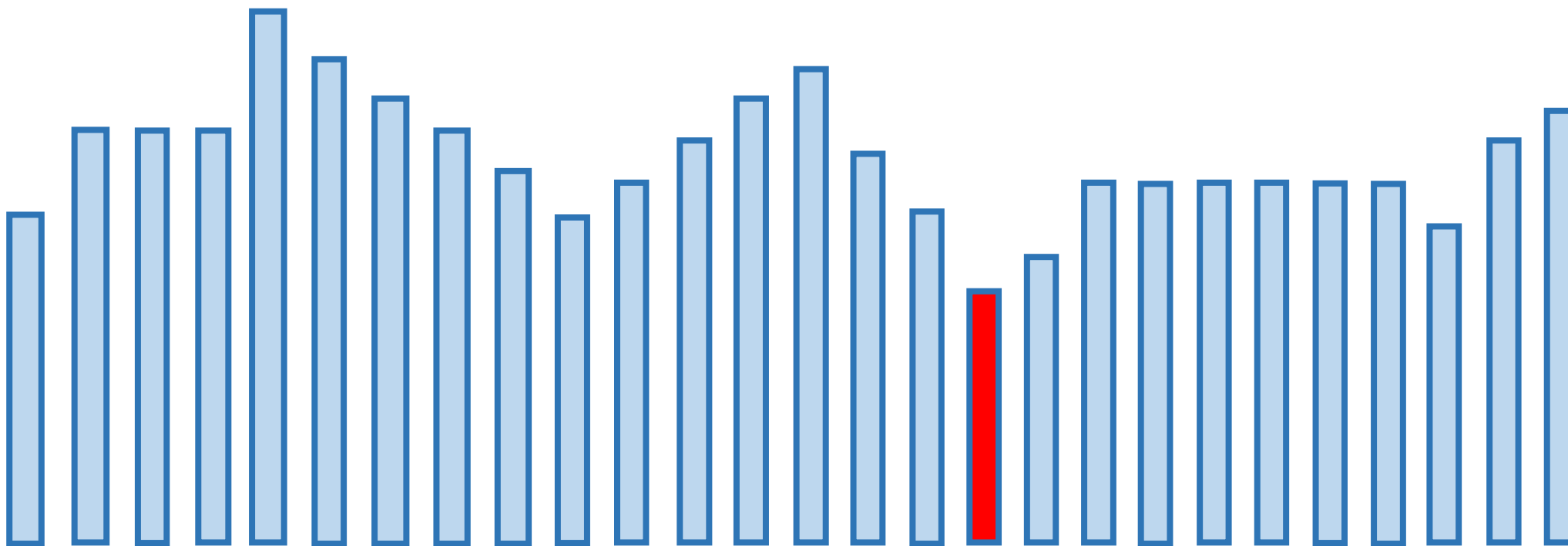
爬山算法 Hill Climbing – 找全局最小

- 选择值**更小**的相邻状态，并移动过去



爬山算法 Hill Climbing – 找全局最小

- 直到这个状态周围的相邻状态的值都**更大**



爬山算法 Hill Climbing – 找全局最大(小)

function Hill-Climb(*problem*):

current = *problem* 的初始状态

重复:

neighbor = *current* 的相邻状态中值最高(小)的状态

if *neighbor* 都比 *current* 更小(大):

 return *current*

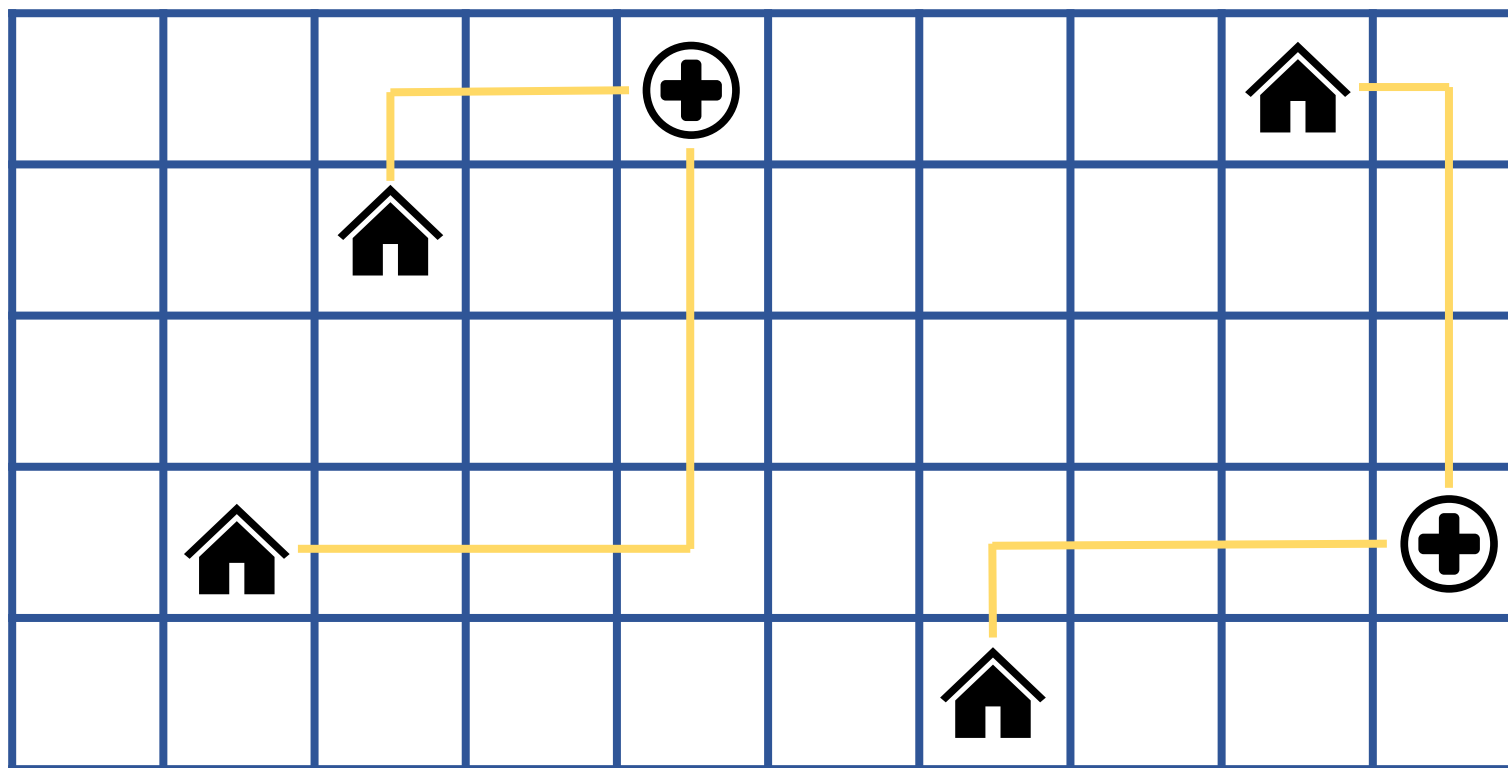
current = *neighbor*

爬山算法 Hill Climbing 示例

- 建造两个社区医院

- 尽量减少从每个住宅区到社区医院的距离

Cost: 17

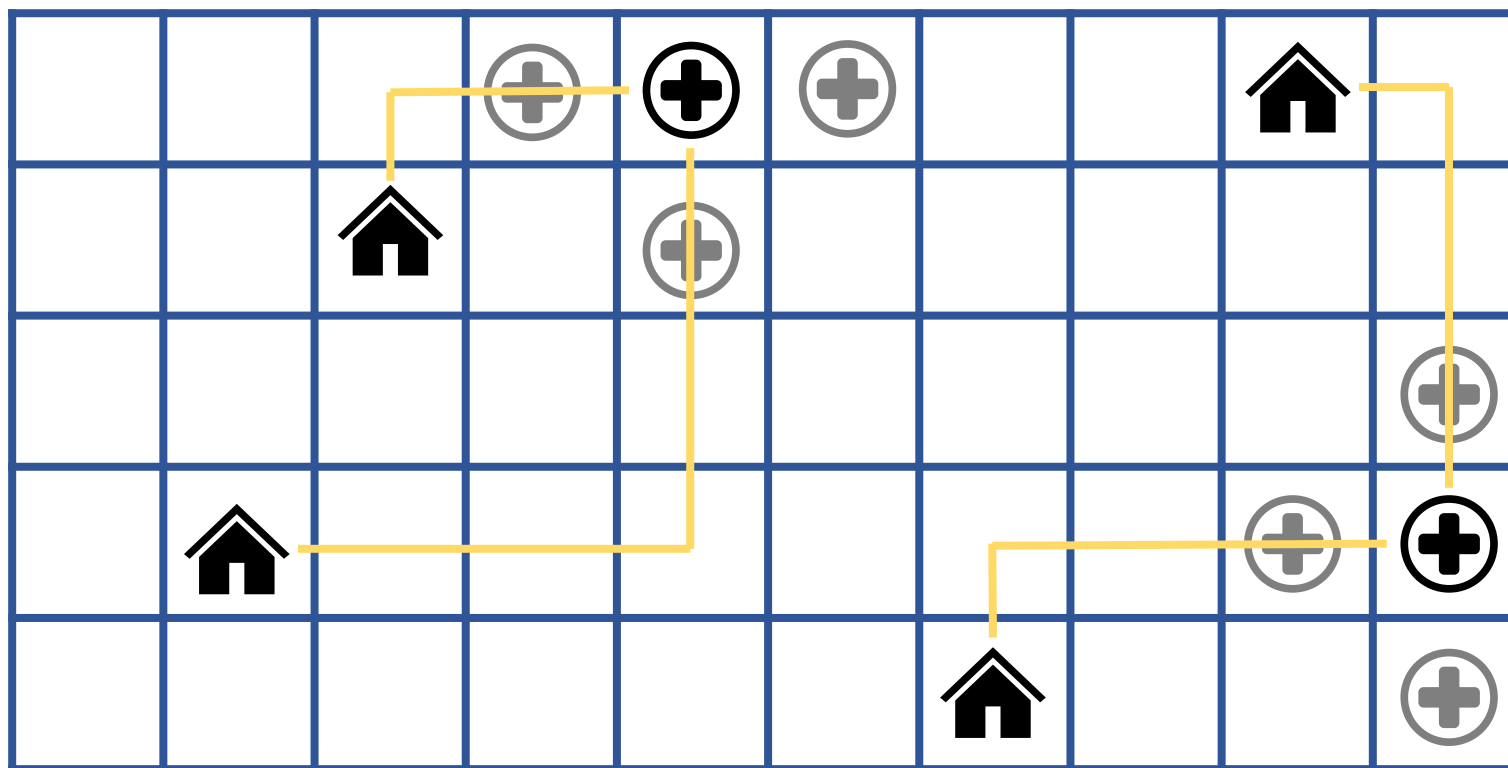


爬山算法 Hill Climbing 示例

- 建造两个社区医院

- 尽量减少从每个住宅区到社区医院的距离

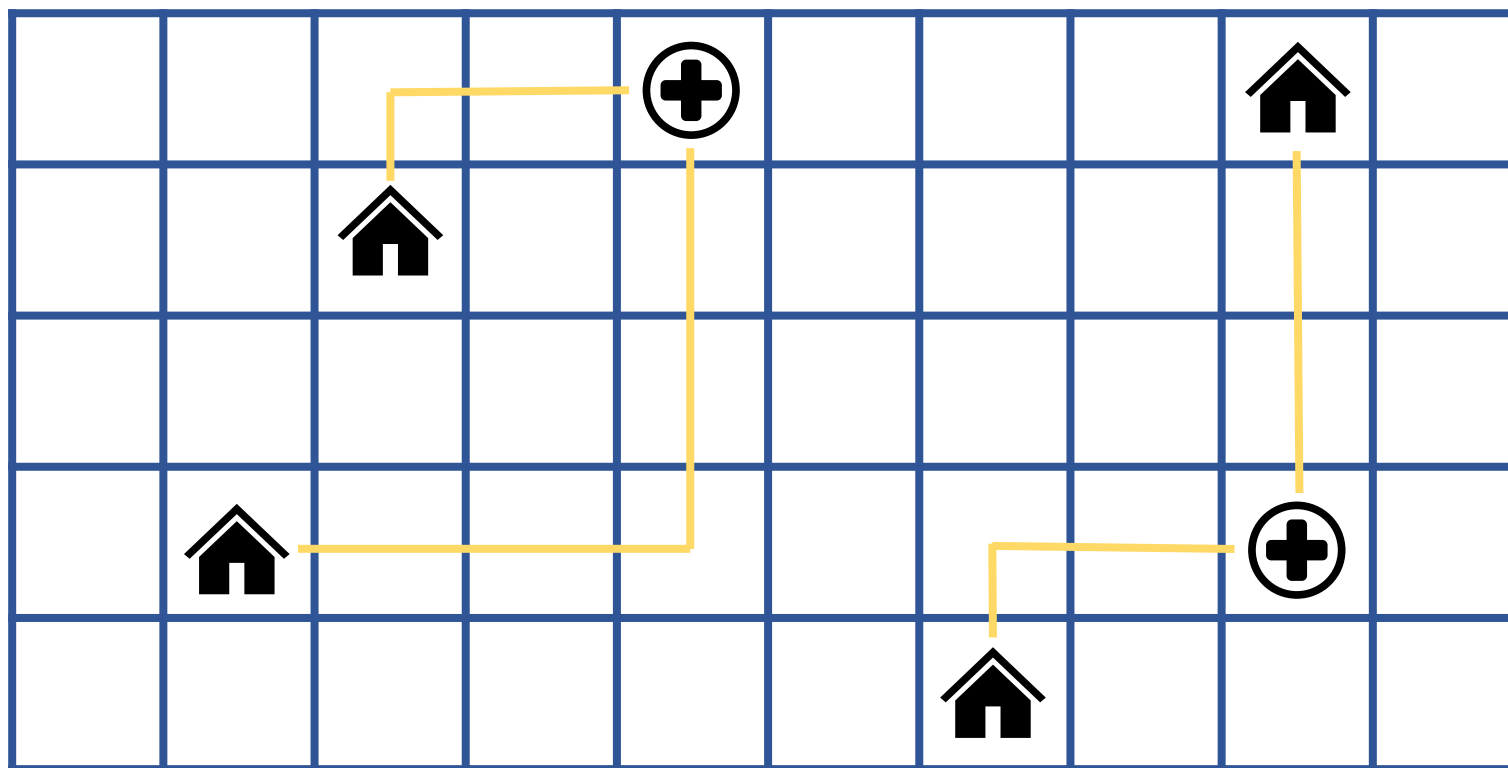
Cost: 17



爬山算法 Hill Climbing 示例

- 建造两个社区医院
 - 尽量减少从每个住宅区到社区医院的距离

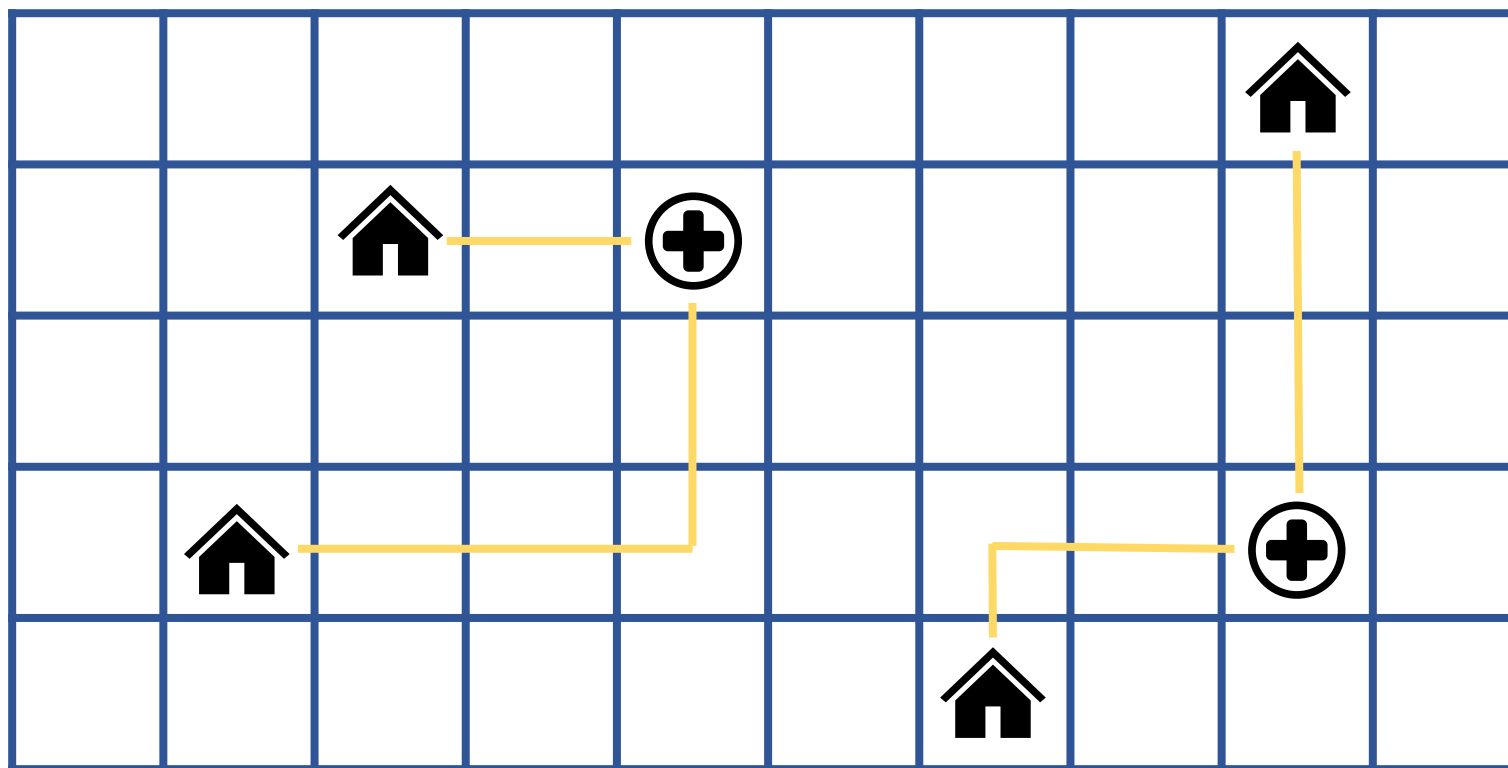
Cost: 15



爬山算法 Hill Climbing 示例

- 建造两个社区医院
 - 尽量减少从每个住宅区到社区医院的距离

Cost: 13

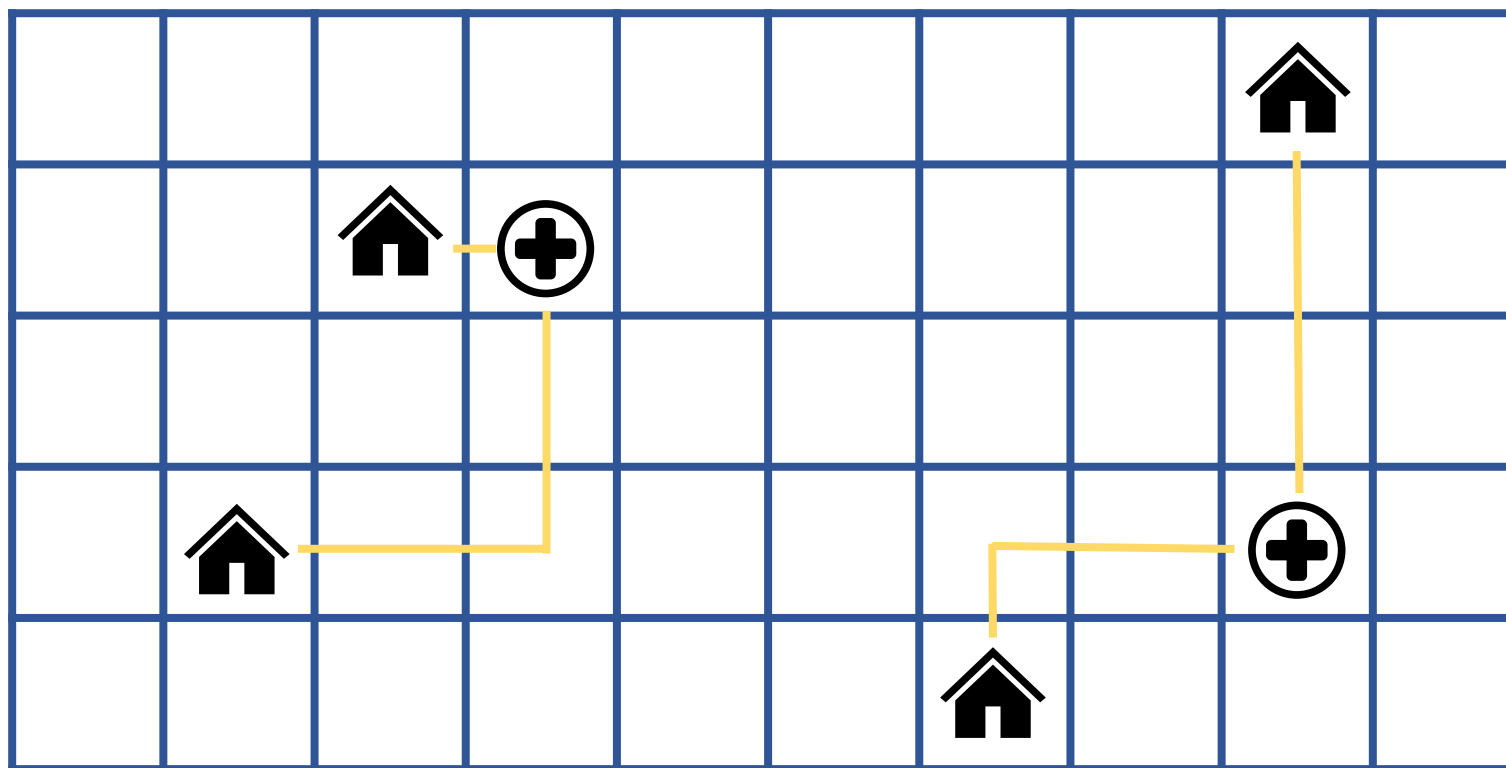


爬山算法 Hill Climbing 示例

- 建造两个社区医院
 - 尽量减少从每个住宅区到社区医院的距离

Cost: 11

这是最好的位置吗？

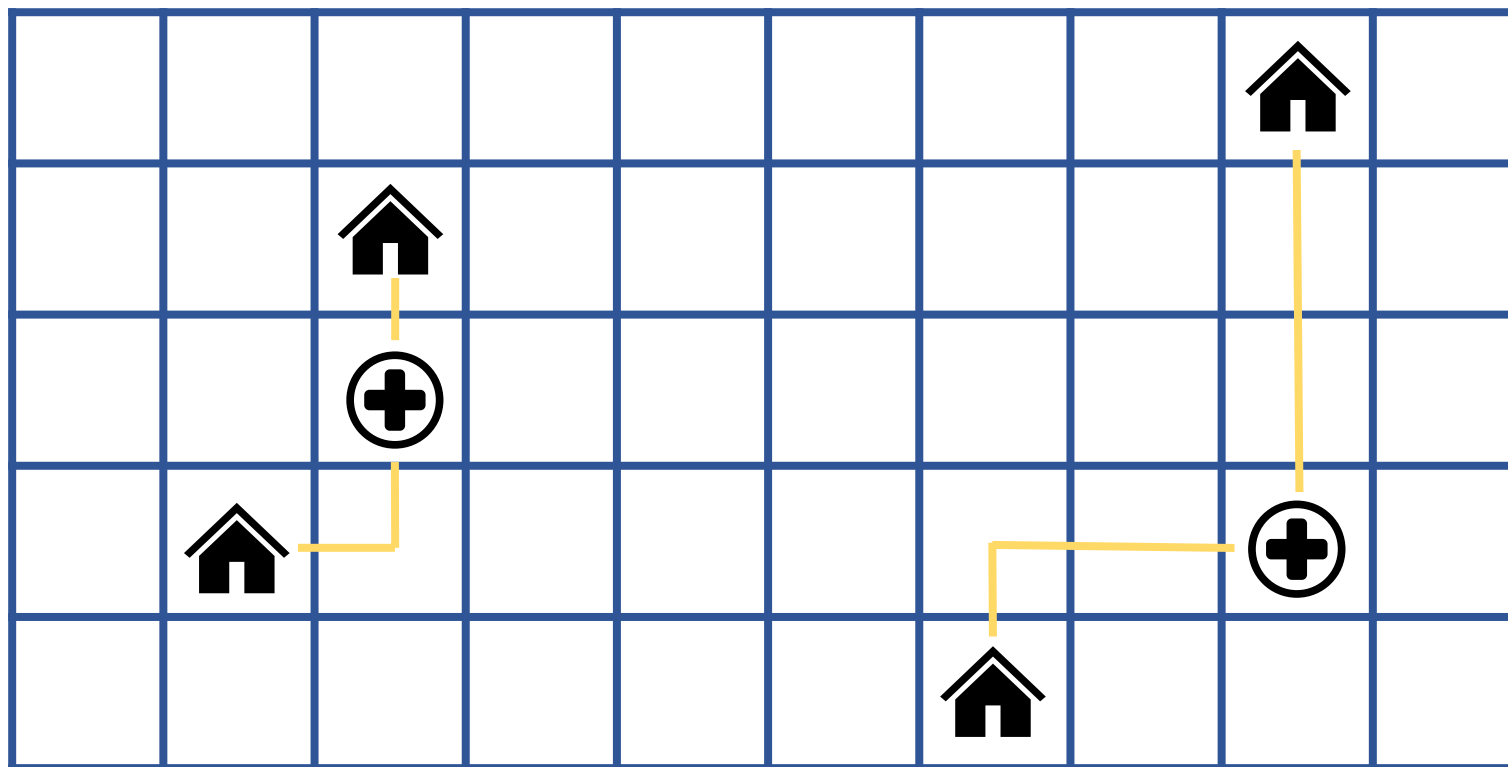


爬山算法 Hill Climbing 示例

- 建造两个社区医院
 - 尽量减少从每个住宅区到社区医院的距离

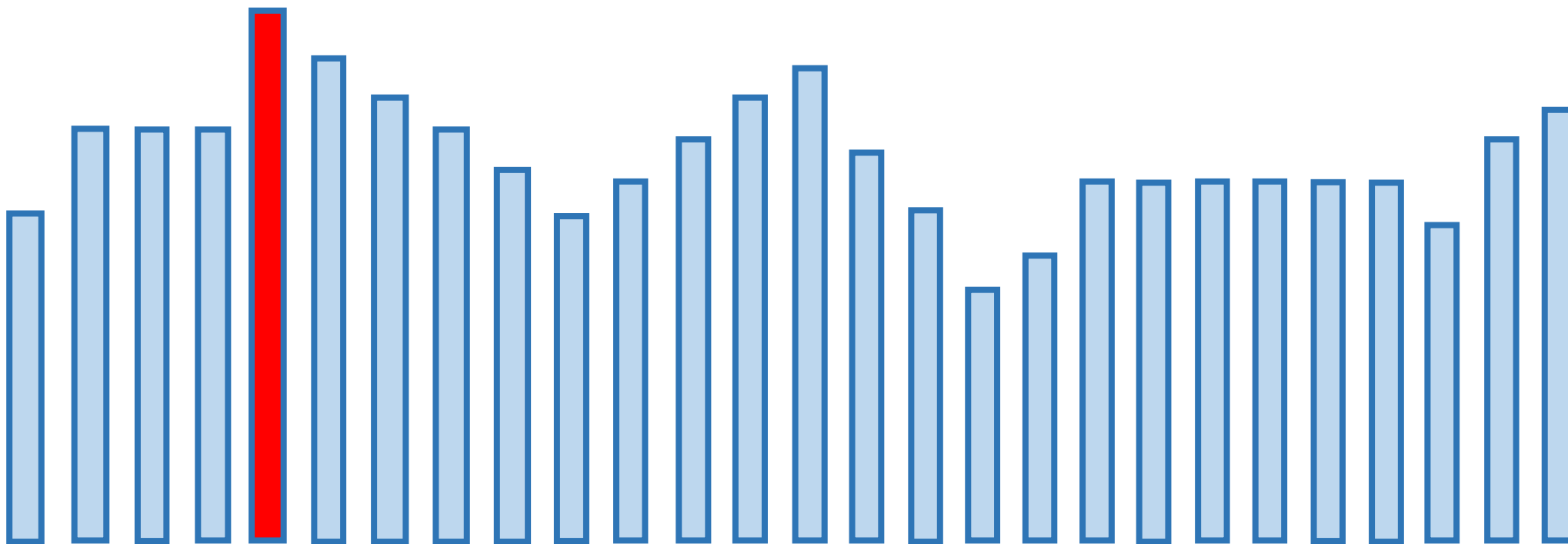
Cost: 9

这个位置更好



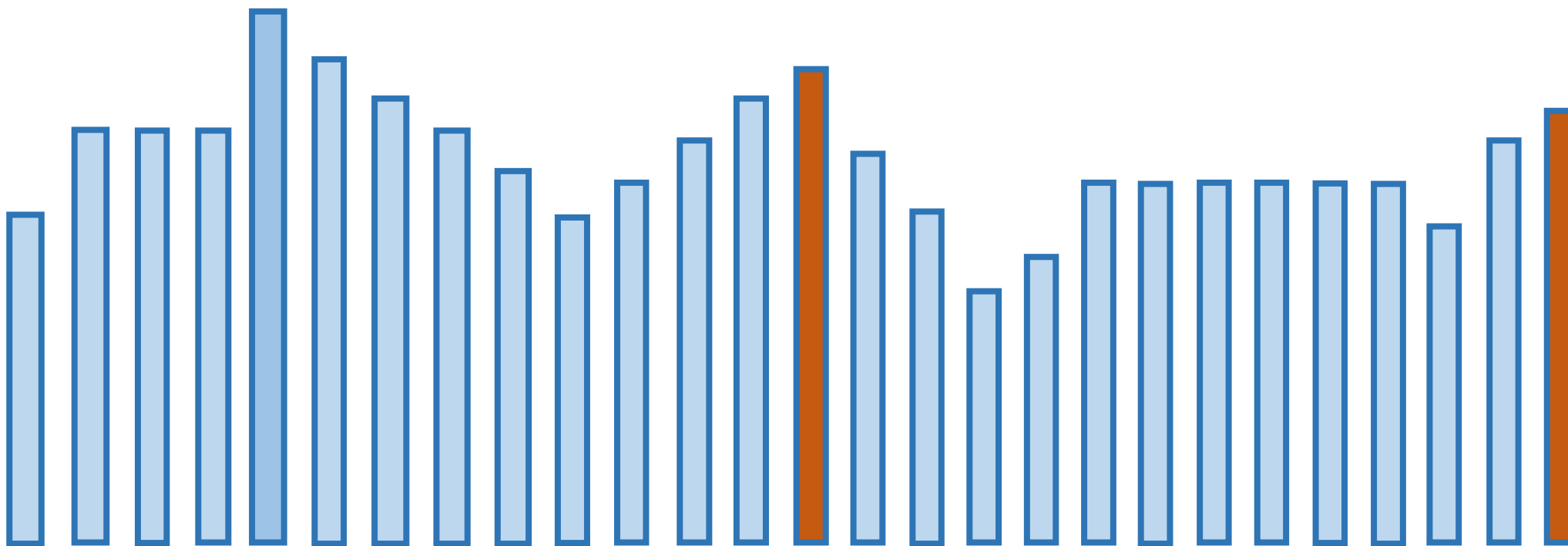
爬山算法的潜在问题

- 想要找到全局最大值



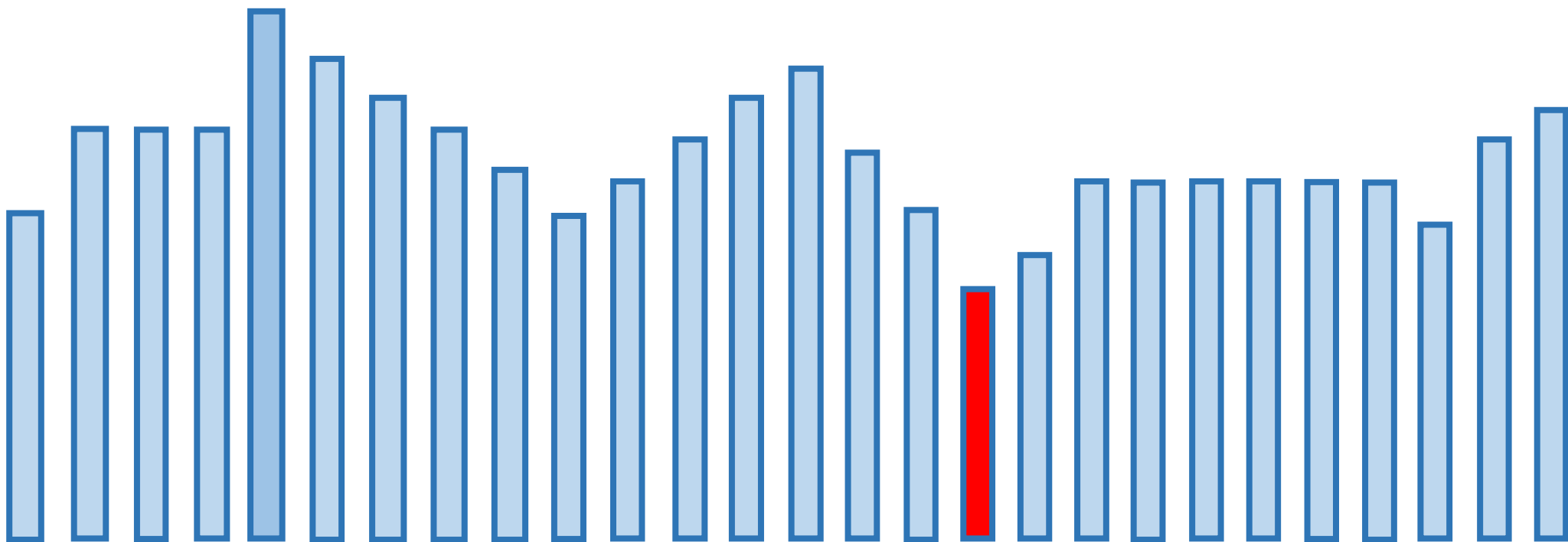
爬山算法的潜在问题

- 想要找到全局最大值
 - 可能找到是局部最大值（状态的值高于其相邻状态）



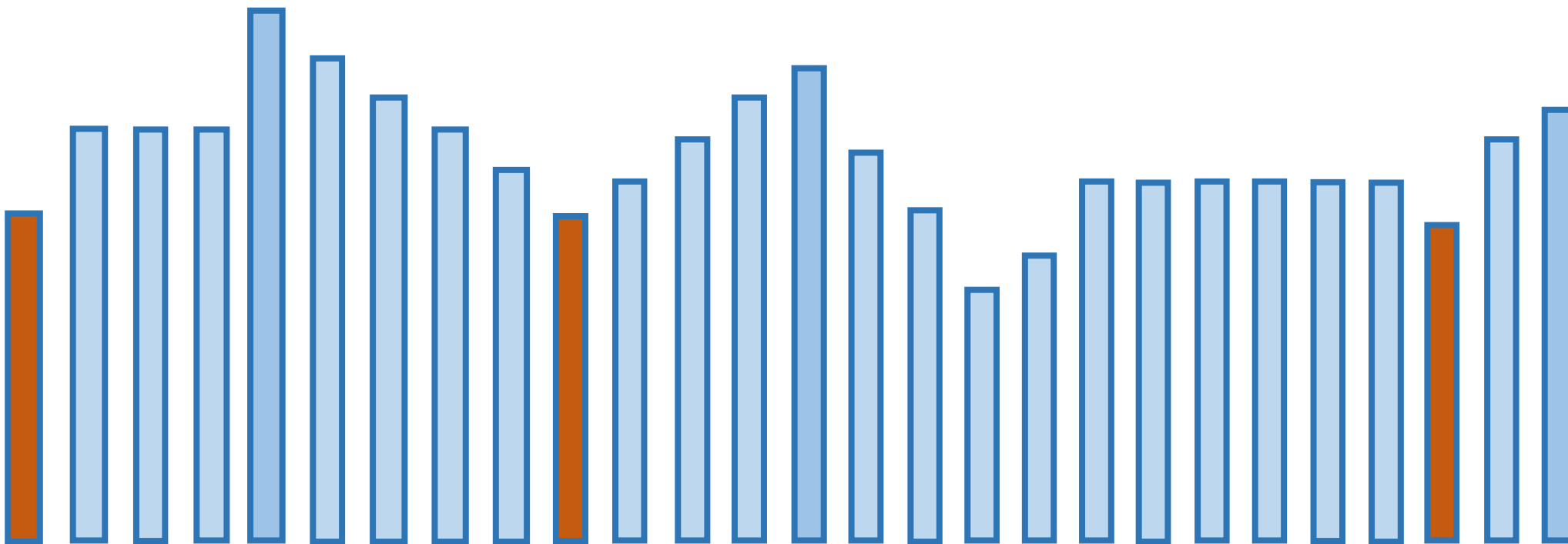
爬山算法的潜在问题

- 想要找到全局最小值



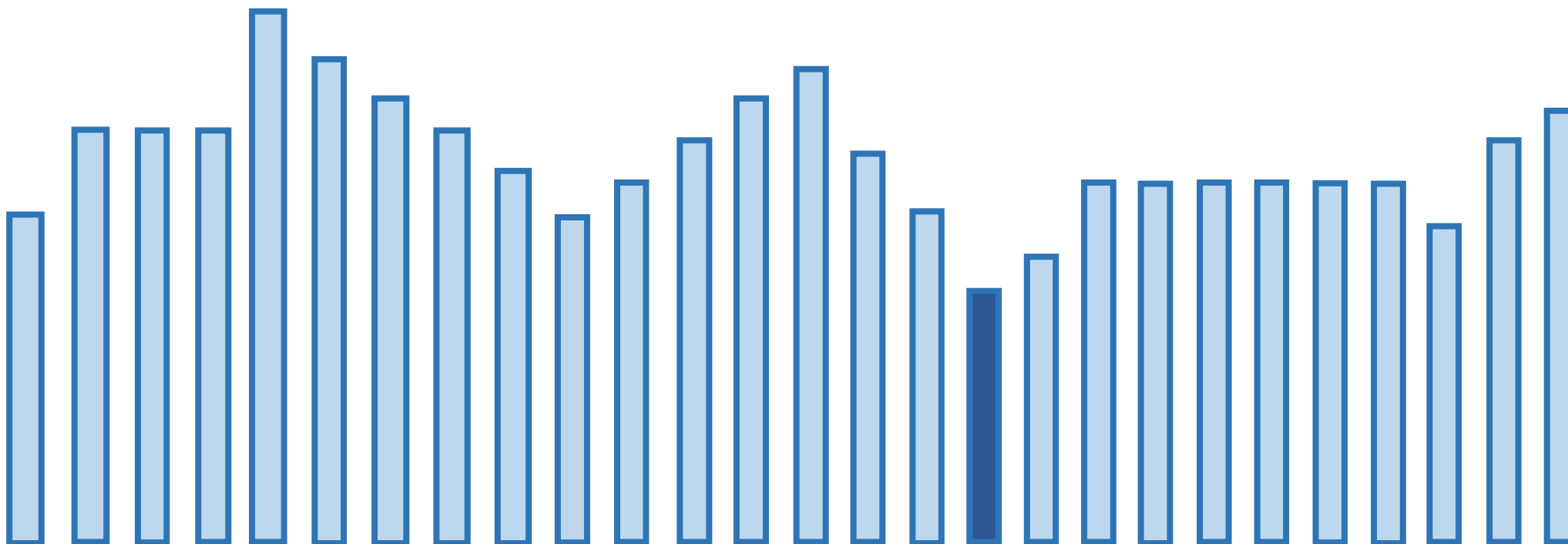
爬山算法的潜在问题

- 想要找到全局最小值
 - 可能找到是局部最小值（状态的值小于其相邻状态）



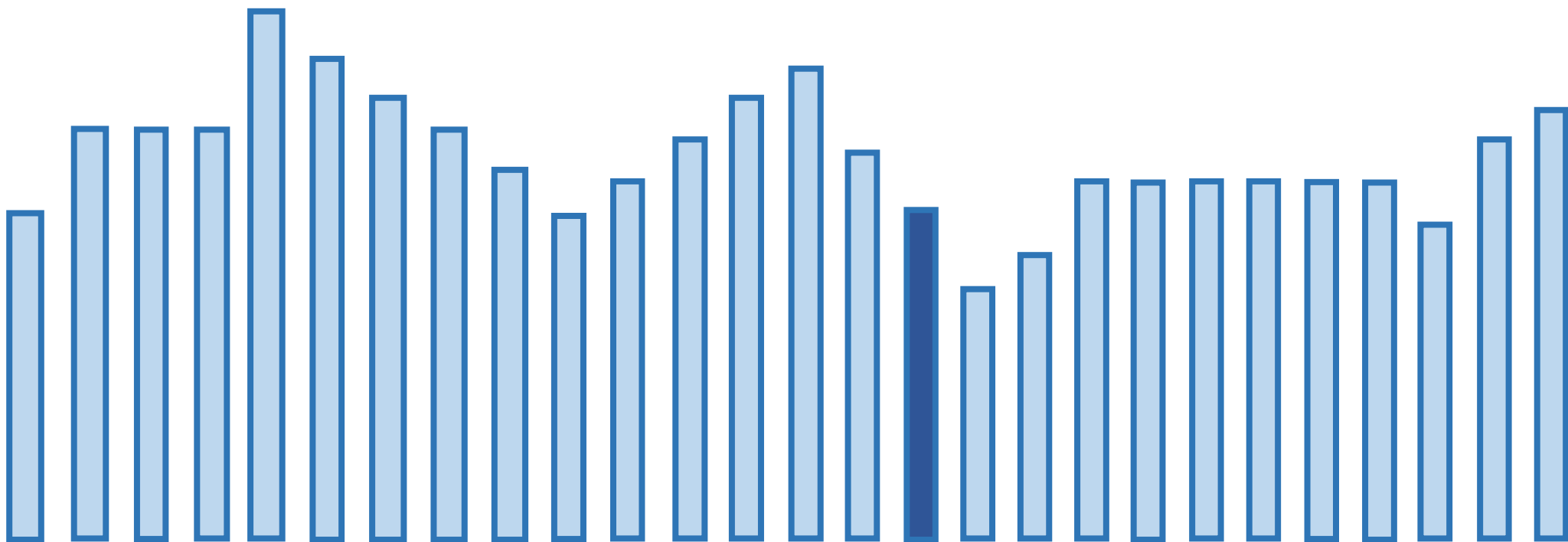
爬山算法的潜在问题

- 可能并不总是找到最优解



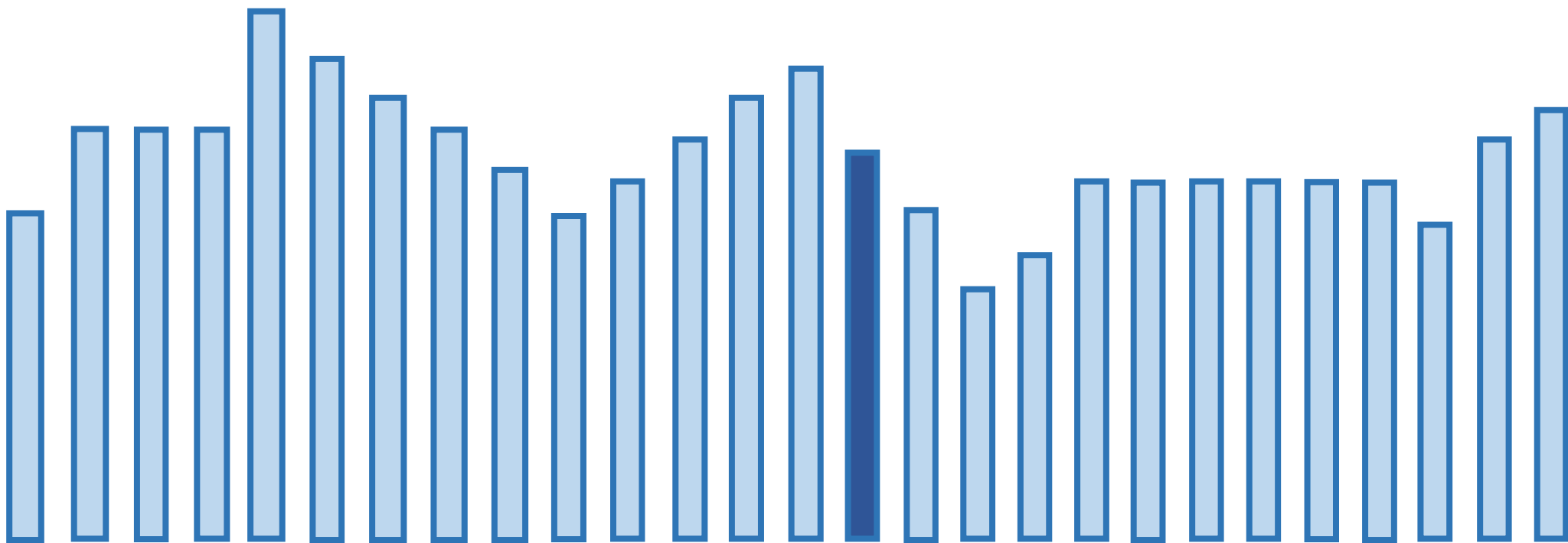
爬山算法的潜在问题

- 可能并不总是找到最优解



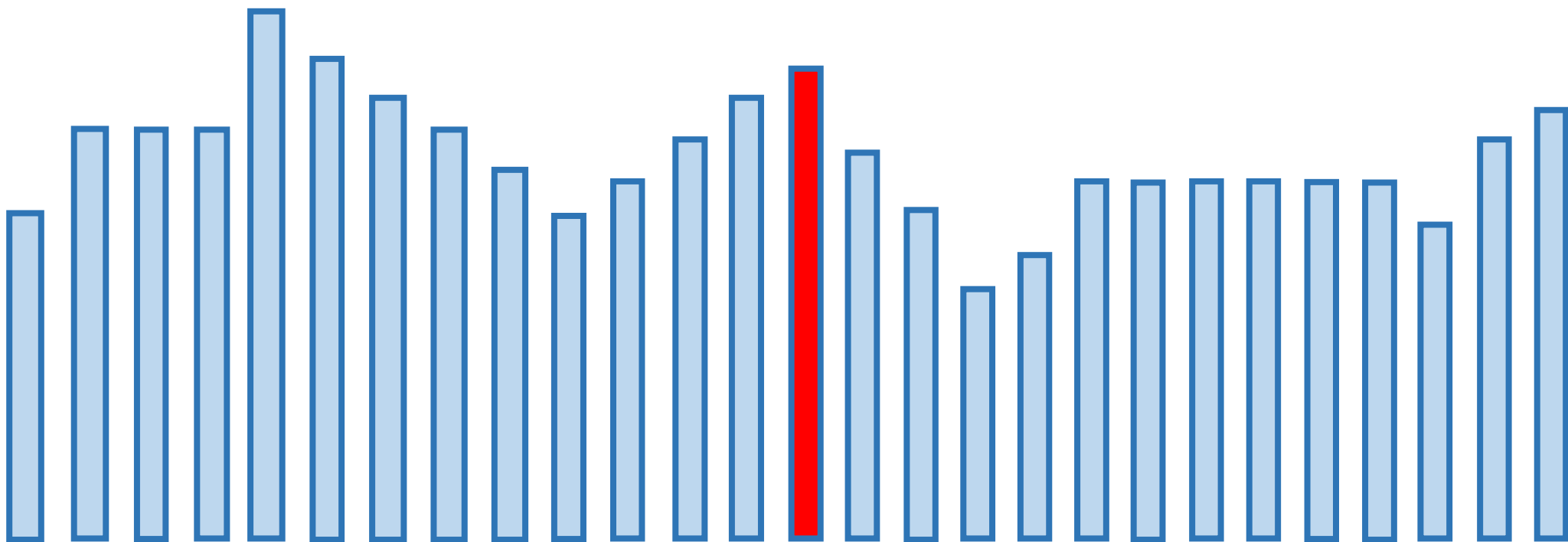
爬山算法的潜在问题

- 可能并不总是找到最优解



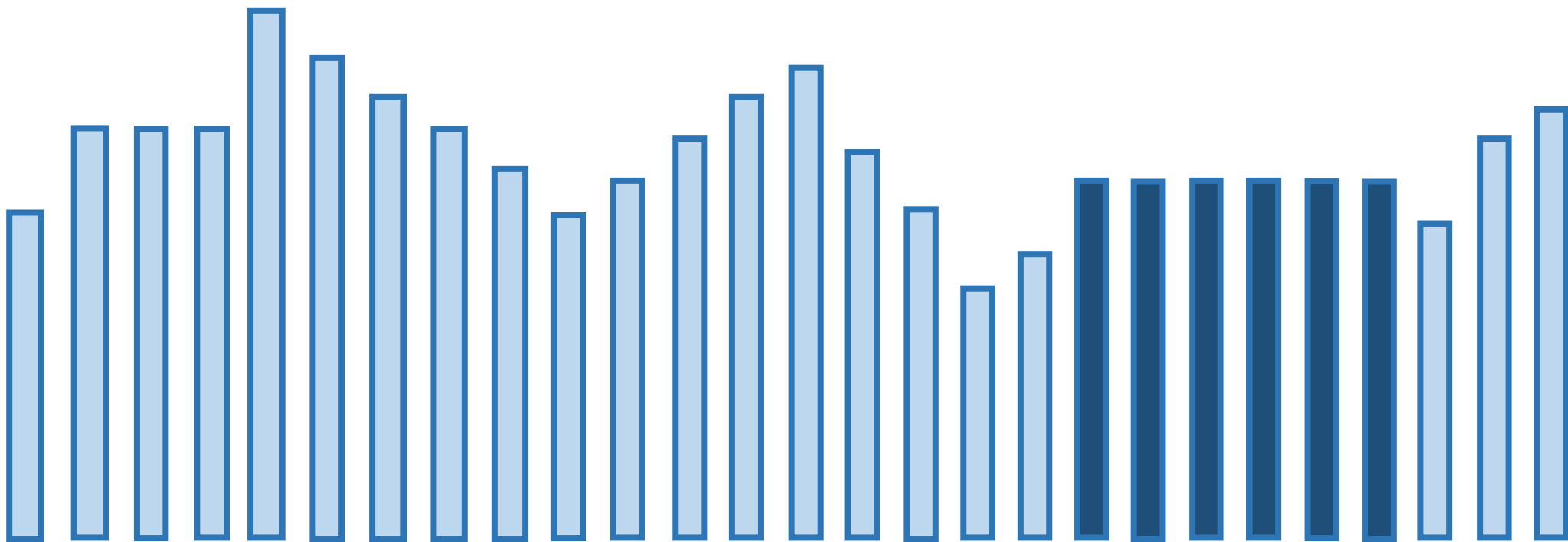
爬山算法的潜在问题

- 可能并不总是找到最优解



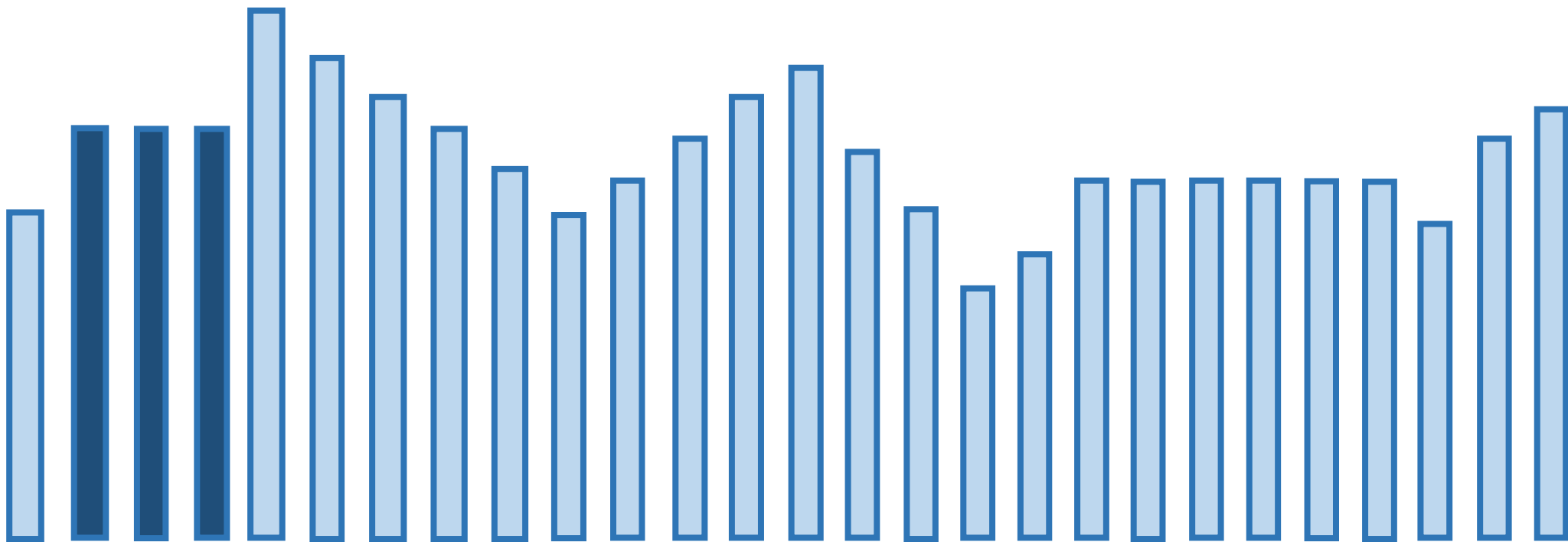
爬山算法的潜在问题

- 平坦局部最大值/最小值 Flat local maximum/minimum
 - 多个等值状态是相邻的，从而形成一个高原，其邻居状态的值都较差



爬山算法的潜在问题

- 肩部 Shoulder
 - 多个等值状态是相邻的，从而形成一个高原，其邻居状态的值有好有坏

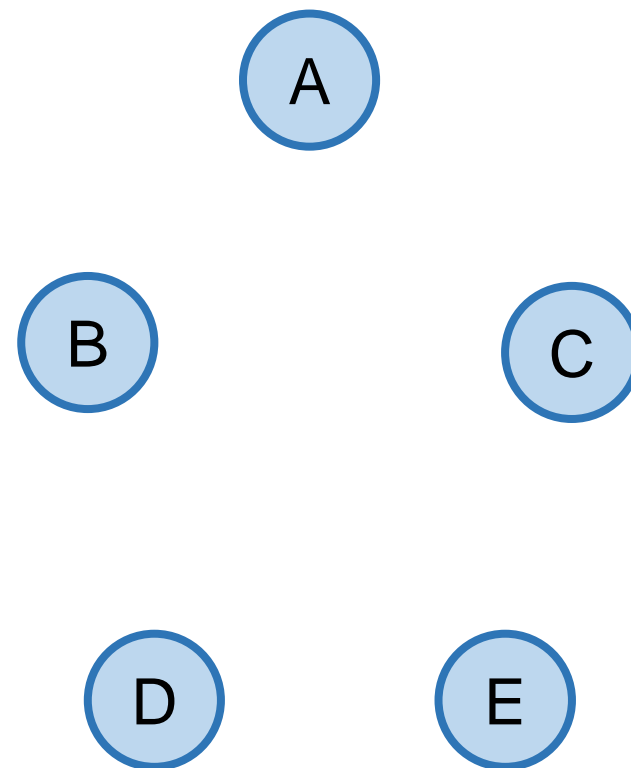


爬山算法的变种

- 最抖爬山算法 Steepest-ascent hill climbing
 - 选择**值最高的**相邻状态
- 随机爬山算法 Stochastic hill climbing
 - 从值较高的相邻状态中**随机**选择一个状态
- 首选爬山算法 First-choice hill climbing
 - 选择**第一个**值较高的相邻状态
- 随机重启爬山算法 Random-restart hill climbing
 - 执行爬山算法**多次**

练习 #1

- Project#1 中多个包裹送达问题
 - 可看做一个可以用局部搜索解决的问题
 - 假设知道每两点之间的最短路径
- ① 每一个状态指什么？
 - ② 每一个状态的代价函数是什么？
 - ③ 相邻状态是什么？



练习 #2

- 将数独问题转换成优化问题
 - 使1-9每个数字在每一行、每一列和每一宫中都只出现一次
- ① 每一个状态指什么？
 - ② 每一个状态的代价函数是什么？
 - ③ 相邻状态是什么？

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

有问题吗？

- 请随时举手提问。



BUSS 3620.人工智能导论

#2.1 代码示例：社区医院选址

刘佳璐

安泰经济与管理学院

上海交通大学

代码框架

迷宫

- 将问题抽象
 - 如何让计算机明白迷宫？
- 按照解搜索问题的步骤解迷宫
- 输出成果

社区医院选址

- 将问题抽象
 - 如何让计算机明白选址问题？
- 按照解优化问题的步骤制作AI
- 输出成果

如何让计算机明白选址问题

- 有一份地图，上面有一些居民区
 - 模拟这部分数据，测试算法效果
- 每个地点用 (i, j) 来表示
- 两种设施：hospital, houses

按照解优化问题的步骤制作AI

function Hill-Climb(*problem*):

current = *problem* 的初始状态 有哪些可选的状态(*available_spaces*)

重复:

找到邻居(*get_neighbors*), 找到状态的值(*get_costs*)

neighbor = *current* 的相邻状态中值最 (小) 的状态

if *neighbor* 都比 *current* 更 (大):

 return *current*

current = *neighbor*

使用爬山算法的一些准备

- 初始状态：从地图上可以被放置hospital的地方选择
- 练习 #3:
- 构造函数available_spaces():
 - 输入：无
 - 功能：找到整个地图上可以被放置hospital的坐标
 - 无house, 无hospital
 - 输出：坐标集合

使用爬山算法的一些准备

- 相邻状态
- 练习 #4:
- 构造函数 `get_neighbors()`:
 - 输入：横坐标 `row`, 纵坐标 `col`
 - 功能：找到这个坐标的相邻坐标
 - 无 `house`, 无 `hospital`
 - 输出：坐标集合

使用爬山算法的一些准备

- 成本函数
- 练习 #5:
- 构建函数 `get_cost()`:
 - 输入：含有hospital的坐标集合
 - 功能：计算每一个houses到最近的hospital的距离之和
 - 如果hospital的坐标集合为空，返回""
 - 输出：数字

使用爬山算法找到最佳选址地点

function Hill-Climb(*problem*):

current = *problem* 的初始状态 在available_spaces中随机放置2个医院

重复: (找到解决方案或达到提前设定好的最大尝试次数时停止)

neighbor = *current* 的相邻状态中值最 (小) 的状态

if *neighbor* 都比 *current* 更 (大):

 return *current*

current = *neighbor*

使用爬山算法找到最佳选址地点

```
def hill_climb(self, maximum=None, image_prefix=None, log=False):
    """Performs hill-climbing to find a solution."""
    count = 0

    # Start by initializing testCenters randomly
    self.testCenters = set()
    for i in range(self.num_testCenters):
        self.testCenters.add(random.choice(list(self.available_spaces())))
    if log:
        print("Initial state: cost", self.get_cost(self.testCenters))
    if image_prefix:
        self.output_image(f"{image_prefix}{str(count+1).zfill(3)}.png")

    # Continue until we reach maximum number of iterations
    while maximum is None or count < maximum:
        count += 1
        best_neighbors = []
        best_neighbor_cost = None

        # Consider all testCenters to move
        for testCenter in self.testCenters:

            # Consider all neighbors for that testCenter
            for replacement in self.get_neighbors(*testCenter):

                # Generate a neighboring set of testCenters
                neighbor = self.testCenters.copy()
                neighbor.remove(testCenter)
                neighbor.add(replacement)

                # Check if neighbor is best so far
                cost = self.get_cost(neighbor)
                if best_neighbor_cost is None or cost < best_neighbor_cost:
                    best_neighbor_cost = cost
                    best_neighbors = [neighbor]
                elif best_neighbor_cost == cost:
                    best_neighbors.append(neighbor)

        # None of the neighbors are better than the current state
        if best_neighbor_cost >= self.get_cost(self.testCenters):
            return self.testCenters

        # Move to a highest-valued neighbor
        else:
            if log:
                print(f"Found better neighbor: cost {best_neighbor_cost}")
            self.testCenters = random.choice(best_neighbors)

    # Generate image
    if image_prefix:
        self.output_image(f"{image_prefix}{str(count+1).zfill(3)}.png")
```

使用随机重启爬山算法找到最佳选址地点

- 练习 #6:
- 构建函数random_restart():
 - 输入：重启的次数，输出图片的文件名前缀，是否要输出log
 - 功能：重复执行爬山算法，从所有结果中找到最佳的选址位置
 - 暂时不用写log为True以及image_prefix为None的情况
 - 输出：坐标集合

有问题吗？

- 请随时举手提问。



BUSS 3620.人工智能导论

#3. 模拟退火

刘佳璐

安泰经济与管理学院

上海交通大学

优化

- 爬山算法(变种)永远只选择更好的相邻状态
- 但如果我们只选择更好的相邻状态，我们可能会遇到下面类似的问题：

- 一旦算法找到局部最优，它就会停止运行

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		10	9	8	7	6	5	4		2
	13		11						5		3
	14	13	12		10	9	8	7	6		4
			13		11						5
A	16	15	14		12	11	10	9	8	7	6

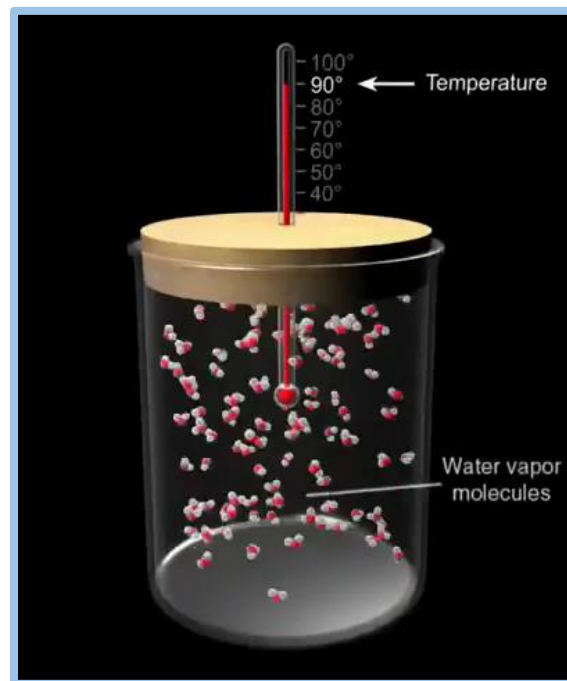
模拟退火 Simulated Annealing

- 退火 Annealing
 - 加热金属并使其缓慢冷却的过程



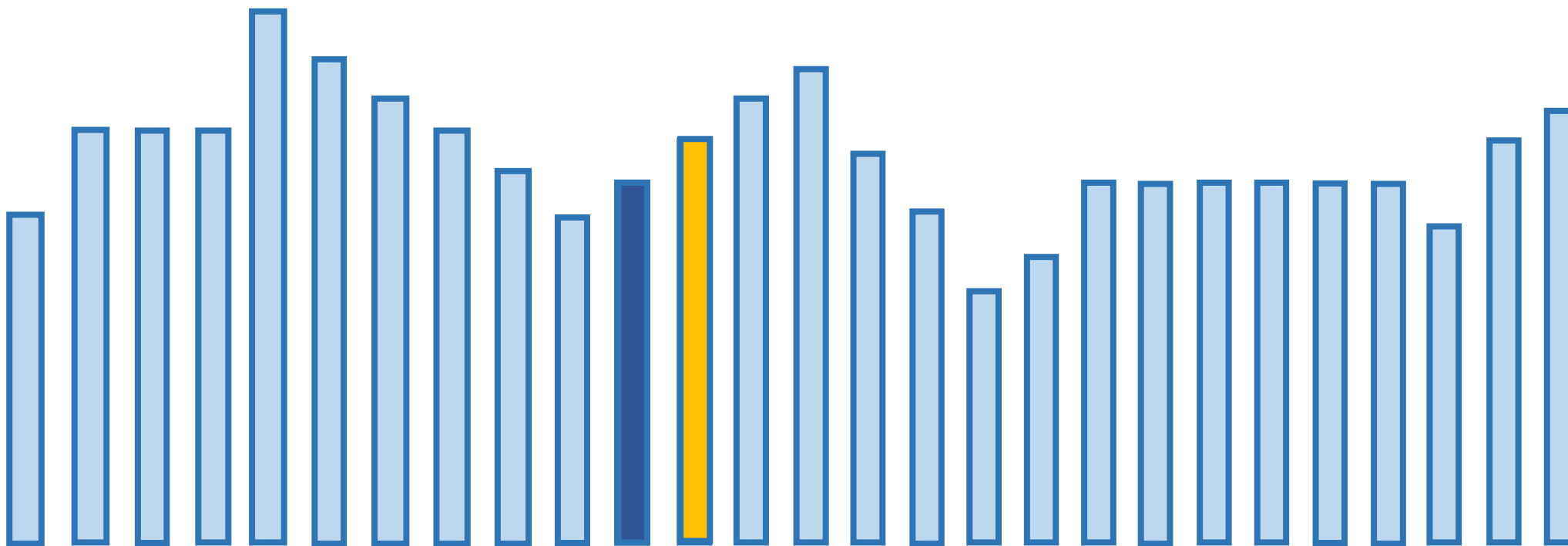
模拟退火 Simulated Annealing

- 退火 Annealing
 - 加热金属并使其缓慢冷却的过程
- 模拟退火 Simulated Annealing
 - 从一个比较高的温度开始
 - 更有可能做出随机选择
 - 随着温度的降低
 - 更低可能做出随机选择



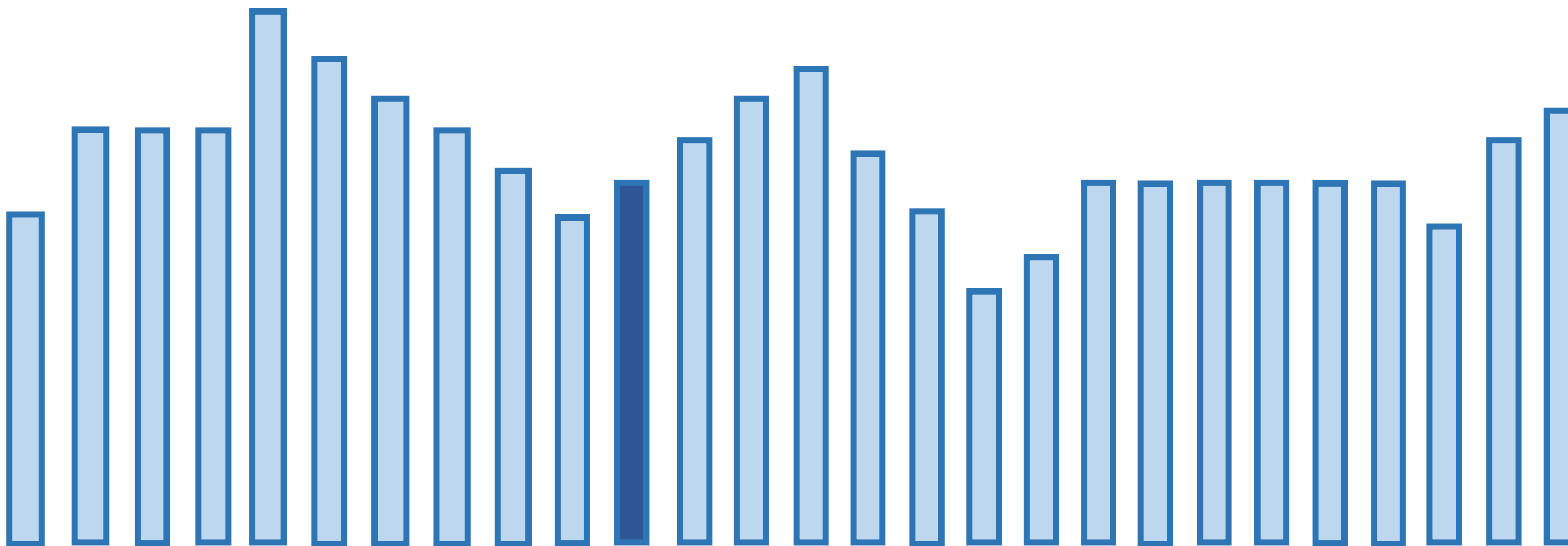
模拟退火 Simulated Annealing

- 爬山算法 Hill climbing: 总是选择更好的状态



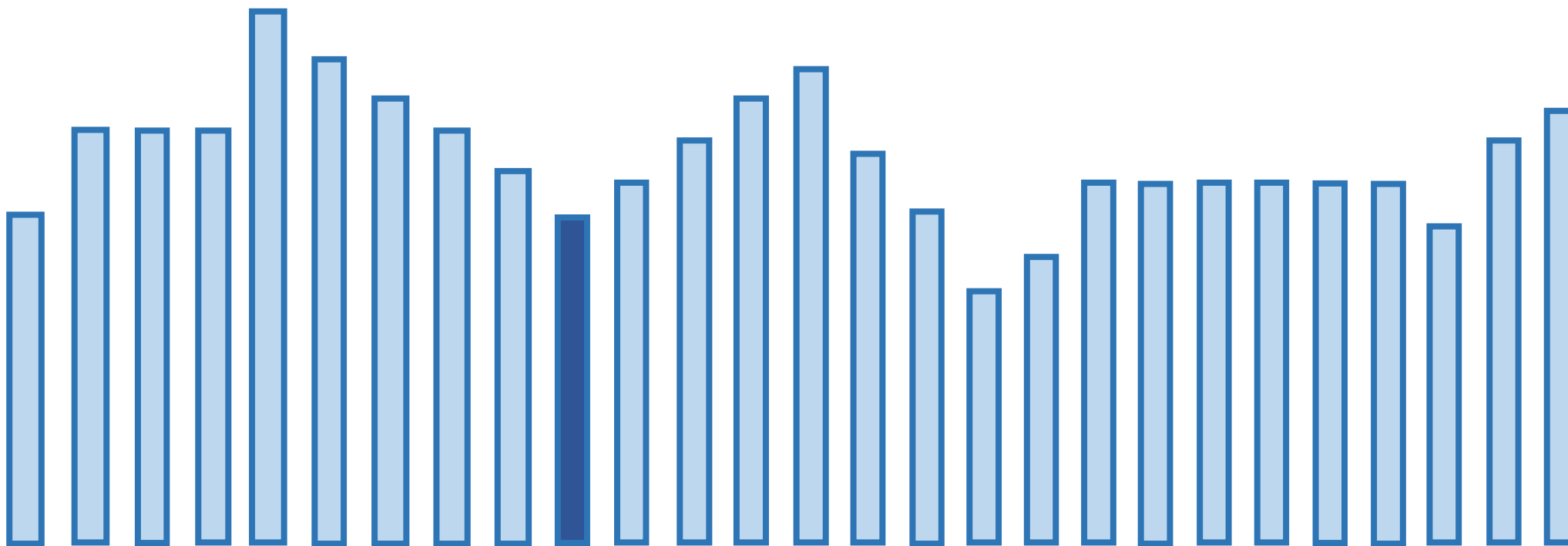
模拟退火 Simulated Annealing

- 模拟退火 Simulated annealing: 允许做出**不好**的选择的可能



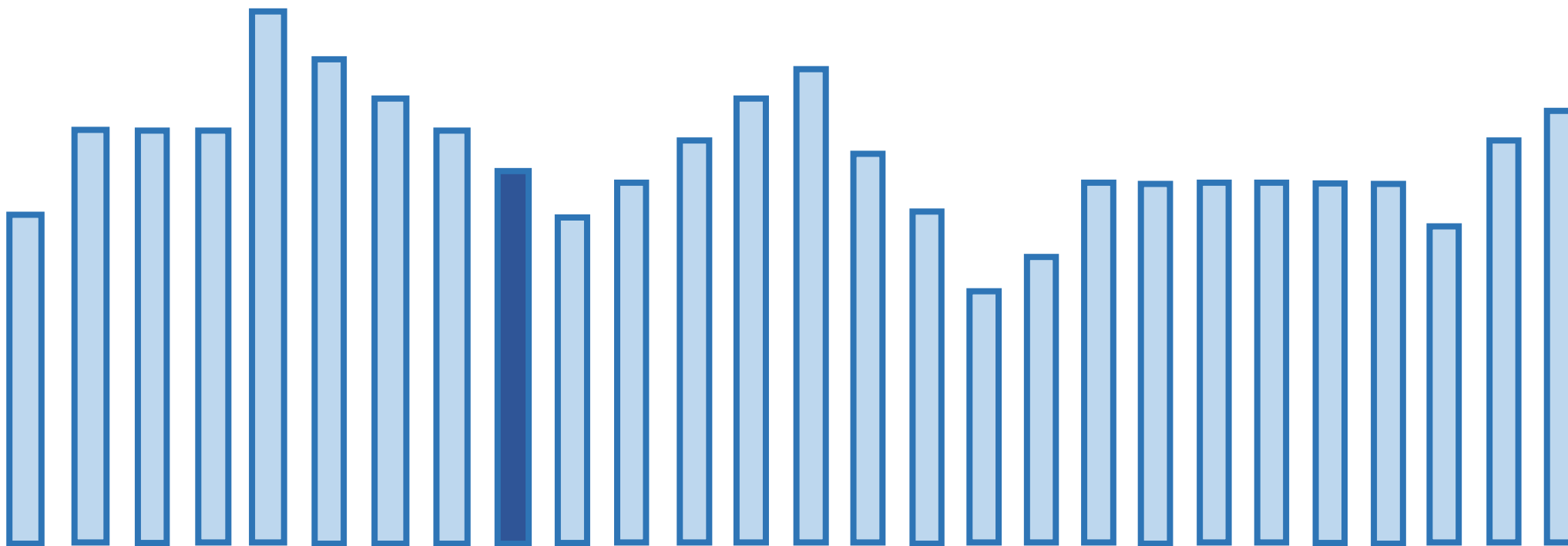
模拟退火 Simulated Annealing

- 模拟退火 Simulated annealing: 允许做出**不好**的选择的可能



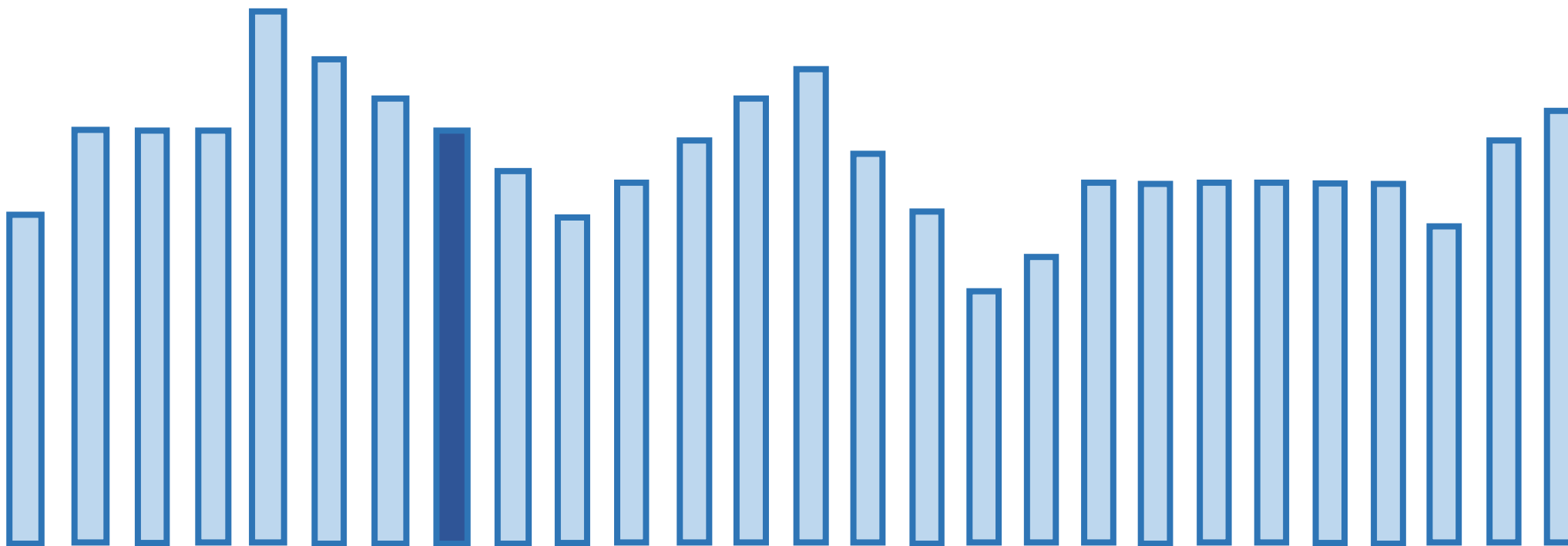
模拟退火 Simulated Annealing

- 模拟退火 Simulated annealing: 允许做出**不好**的选择的可能



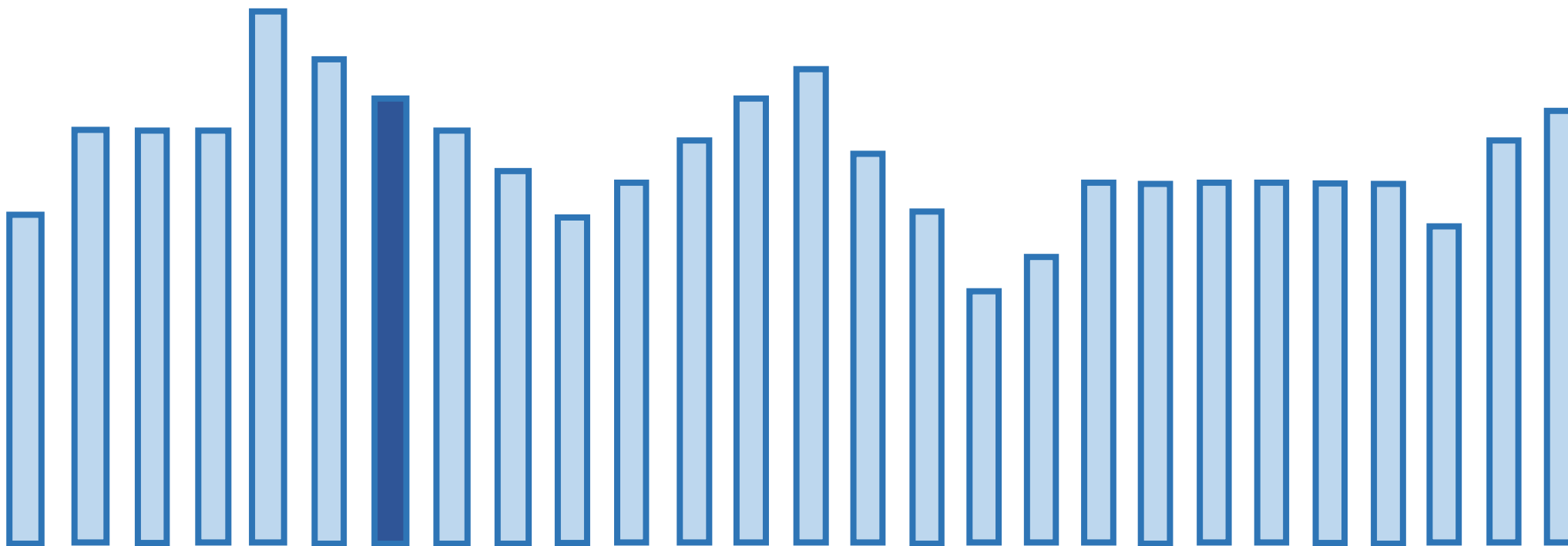
模拟退火 Simulated Annealing

- 模拟退火 Simulated annealing: 允许做出**不好**的选择的可能



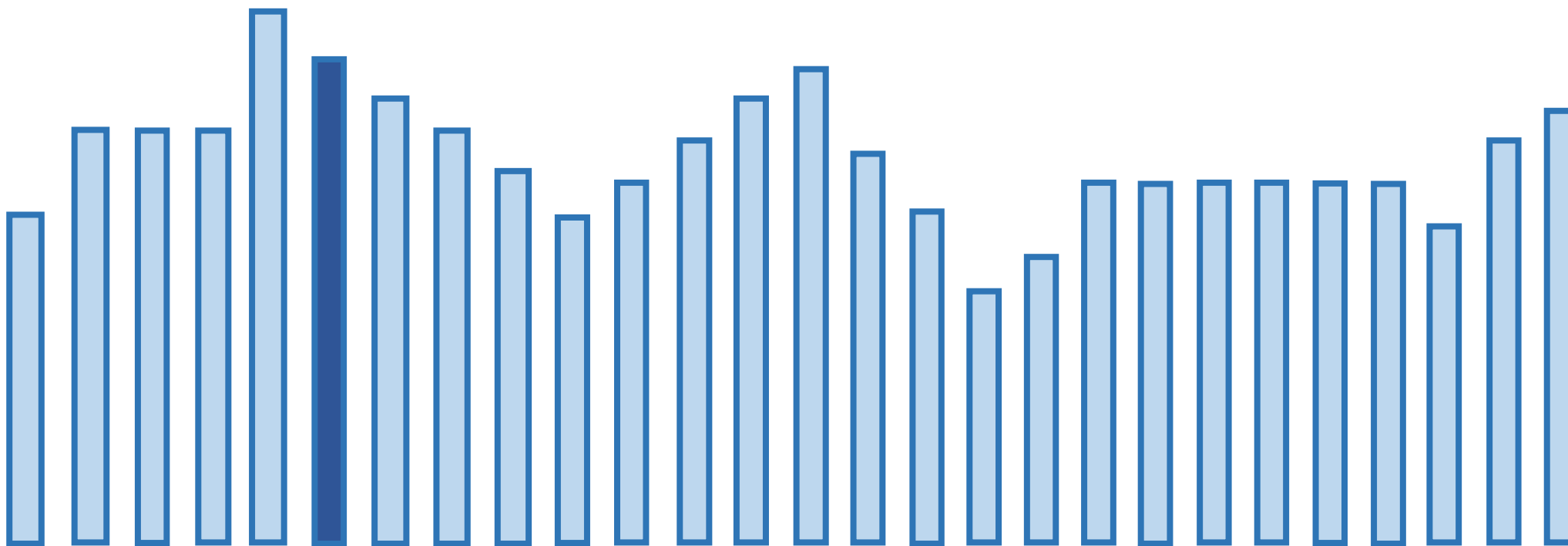
模拟退火 Simulated Annealing

- 模拟退火 Simulated annealing: 允许做出**不好**的选择的可能



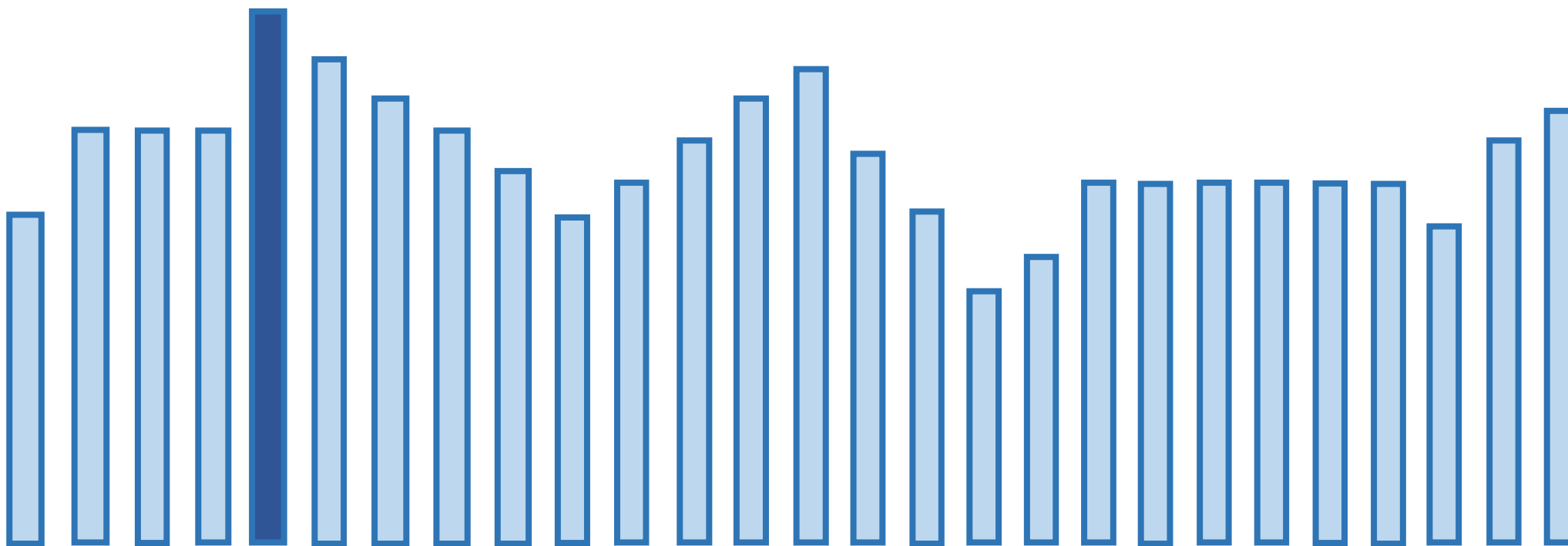
模拟退火 Simulated Annealing

- 模拟退火 Simulated annealing: 允许做出**不好**的选择的可能



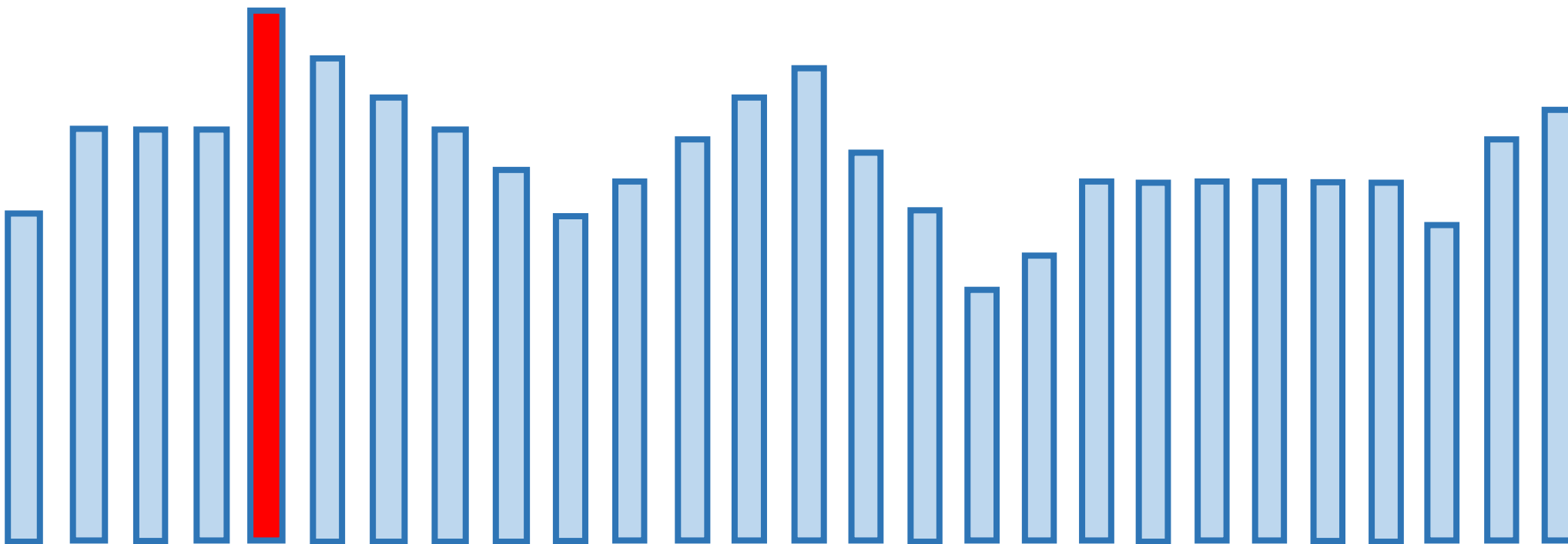
模拟退火 Simulated Annealing

- 模拟退火 Simulated annealing: 允许做出**不好**的选择的可能



模拟退火 Simulated Annealing

- 一旦达到全局最优，就不要再尝试更差的相邻状态



模拟退火 Simulated Annealing

- 早期，“温度”更高
 - 更有可能接受比当前状态更糟糕的相邻状态
- 后期，“温度”较低
 - 更不可能接受比当前状态更糟糕的相邻状态

模拟退火 Simulated Annealing

function Simulated-Annealing(*problem*, *max*):

current = *problem* 的初始状态

For $t = 1$ to *max*:

$T = \text{Temperature}(t)$ (例如, 剩余时间的比例)

neighbor = *current* 的相邻状态中的随机一个

$\Delta E = \text{neighbor}$ 比 *current* 好多少

if $\Delta E > 0$:

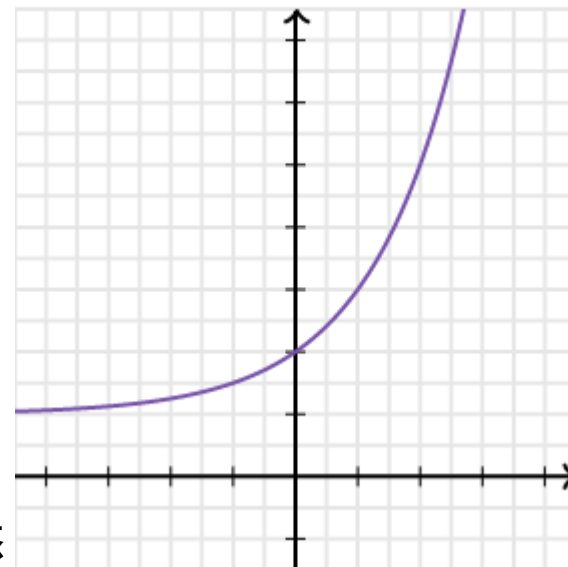
current = *neighbor*

概率 $e^{\frac{\Delta E}{T}}$ 的情况下, *current* = *neighbor*

return *current*

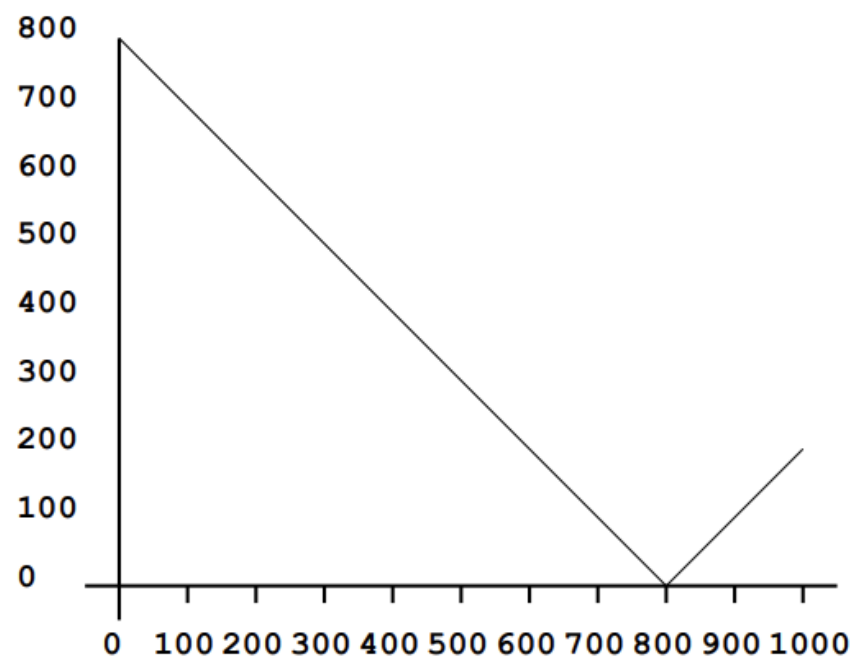
T 越大, 越有可能移动到不好的相邻状态

ΔE 负的越小, 越有可能移动到不好的相邻状态



练习 #7

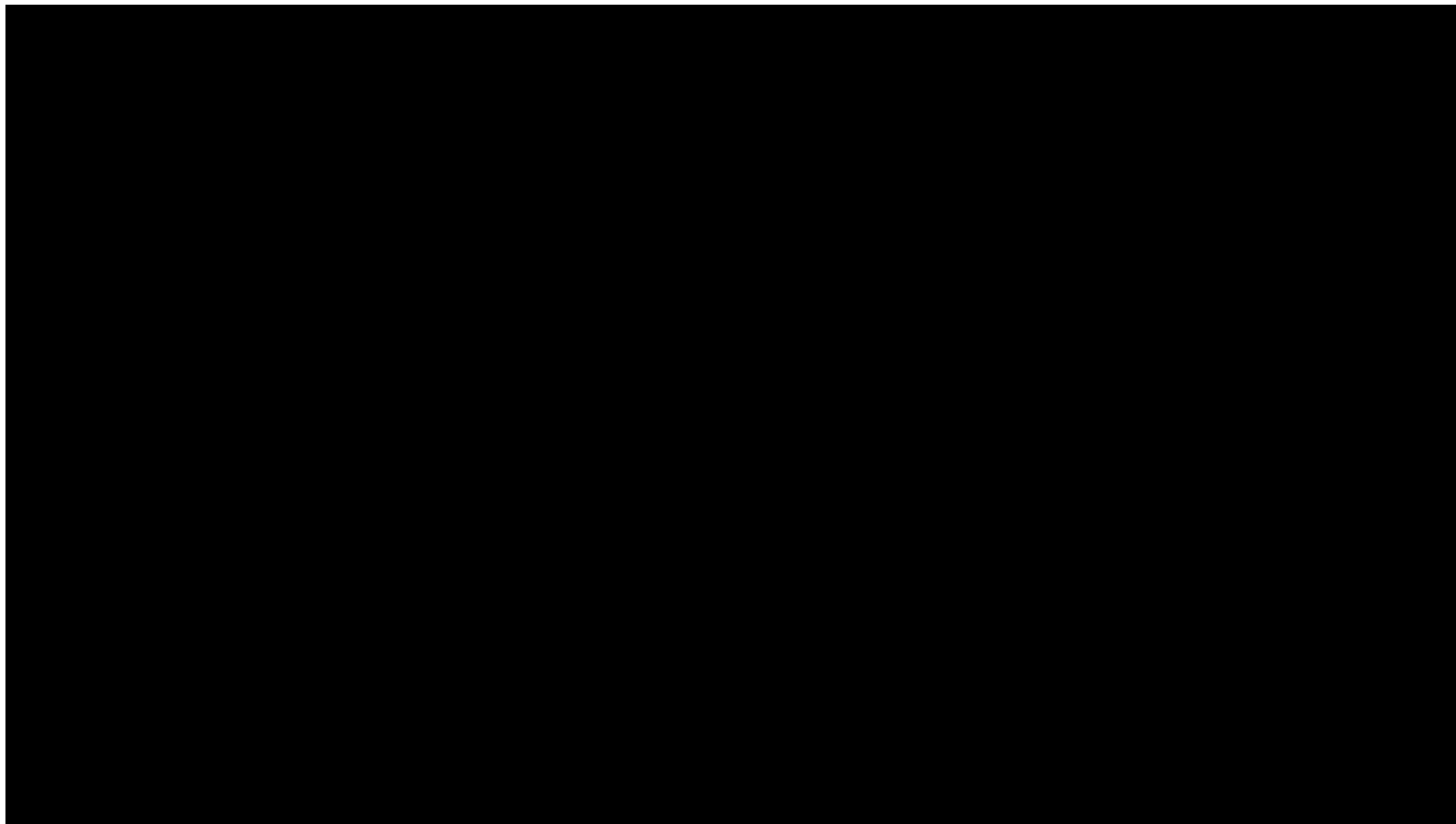
- 考虑右侧的目标函数 $f(x) = |x - 800|$
 - 状态 x 的相邻状态为 $\{x - 1, x + 1\}$
 - 当 $x = 0$, 相邻状态为 $x = 1$
 - 当 $x = 1000$, 相邻状态为 $x = 999$
 - 当初始状态为 $x = 900$ 时, 初始温度为 1
- ① 用爬山算法, 能否找到全局最大?
 - ② 用模拟退火, 假设温度函数为 $T_t = 0.8T_{t-1}$
 - a. 20步之后, 向左移动的概率是?
 - b. 能否找到全局最大?



相关应用

- 设施/位置类型的问题
 - 例如，城市规划
- 数据分析
 - 特征选择、参数调整 (机器学习)
- 组合优化问题
 - 旅行商问题、投资组合

优化问题的几种方法对比



局部搜索 Local search

- 解决以下问题
 - 不关心如何找到解
 - 不关心采取几步才找到解
 - 我们只关心解本身
- 理论上，模拟退火可以找到最优解
 - 但可能时间是永远
- 局部搜索 Local search
 - 可以找到一个不是最佳但“足够好”的解决方案
 - 节省计算资源

BUSS 3620.人工智能导论

#2.2 代码示例：社区医院选址

刘佳璐

安泰经济与管理学院

上海交通大学

模拟退火 Simulated Annealing

function Simulated-Annealing(*problem*, *max*):

current = *problem* 的初始状态

For $t = 1$ to *max*:

$T = \text{Temperature}(t)$ (例如, 剩余时间的比例)

neighbor = *current* 的相邻状态中的随机一个

$\Delta E = \text{neighbor}$ 比 *current* 好多少

if $\Delta E > 0$:

current = *neighbor*

概率 $e^{\frac{\Delta E}{T}}$ 的情况下, *current* = *neighbor*

return *current*

Simulated Annealing

```
145 def simulated_annealing(self, maximum, image_prefix=None, log=False):
146     """Performs simulated annealing to find a solution."""
147
148     # Start by initializing testCenters randomly
149     self.testCenters = set()
150     for i in range(self.num_testCenters):
151         self.testCenters.add(random.choice(list(self.available_spaces())))
152     if log:
153         print("Initial state: cost", self.get_cost(self.testCenters))
154     if image_prefix:
155         self.output_image(f"{image_prefix}{str(0+1).zfill(3)}.png")
156     current_cost=self.get_cost(self.testCenters)
157
158     # Continue until we reach maximum number of iterations
159     for t in range(1,maximum):
160         T = (maximum - t)/maximum
161
162         # Randomly pick one testCenter and one neighbors from the testCenter
163         testCenter=random.choice(list(self.testCenters))
164
165         neighbor = self.testCenters.copy()
166         replacement = random.choice(self.get_neighbors(*testCenter))
167         neighbor.remove(testCenter)
168         neighbor.add(replacement)
169
170         # Calculate cost increase
171         new_cost = self.get_cost(neighbor)
172         cost_diff = current_cost - new_cost
```

```
174
175     # if new neighbor is better
176     if cost_diff>0:
177         if log:
178             print(f"Found better neighbor: cost {new_cost}")
179         self.testCenters = neighbor
180         current_cost = new_cost
181
182     # with probability  $e^{\text{cost\_diff}/T}$  set current = neighbor
183     else:
184         random_num = random.random()
185         if random_num < math.exp(cost_diff/T):
186             if log:
187                 print(f"Found a slightly worse neighbor: cost {new_cost}")
188             self.testCenters = neighbor
189             current_cost = new_cost
190         else:
191             if log:
192                 print(f"Did not find neighbor: cost {new_cost}")
193
194     # Generate image
195     if image_prefix:
196         self.output_image(f"{image_prefix}{str(t + 1).zfill(3)}.png")
197
198     return self.testCenters
```

有问题吗？

- 请随时举手提问。



BUSS 3620.人工智能导论

#4. 约束满足问题

刘佳璐

安泰经济与管理学院

上海交通大学

约束满足问题 Constraint Satisfaction Problems

- 问题有一组变量,每个变量都有自己的取值范围。当每个变量都有自己的赋值, 且**所有关于变量的约束都被满足时**, 问题就被解决。

学生:



1



2



3



4

课程:



A



B



C



B



D



E



C



E



F



E



F



G

考试时间:

星期一

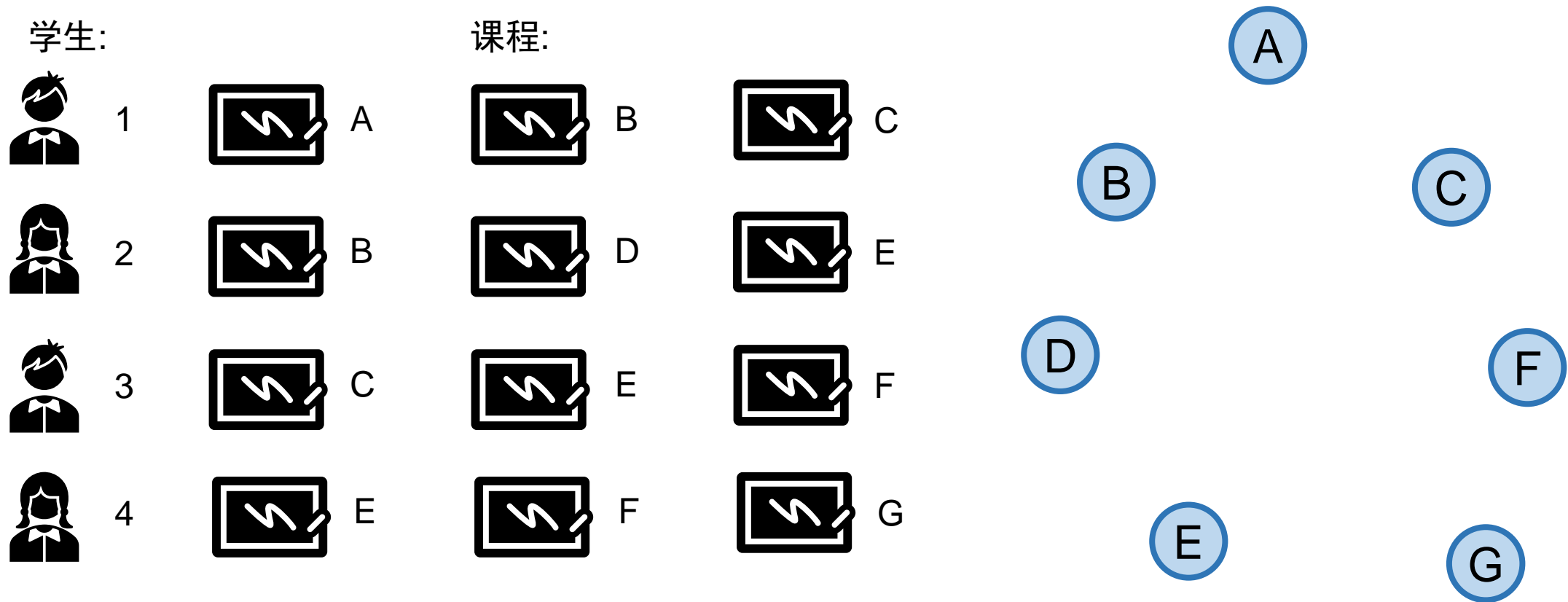
星期二

星期三

约束:

同一学生**不能**在同一天进行两次考试

约束满足问题示例：考试时间



将课程表示为节点

如果两个课程之间存在约束，则在两个节点(课程)之间创建边

约束满足问题示例：考试时间

学生:



1



A



2



B



3



C



4



E

课程:



B



C



D



E



E



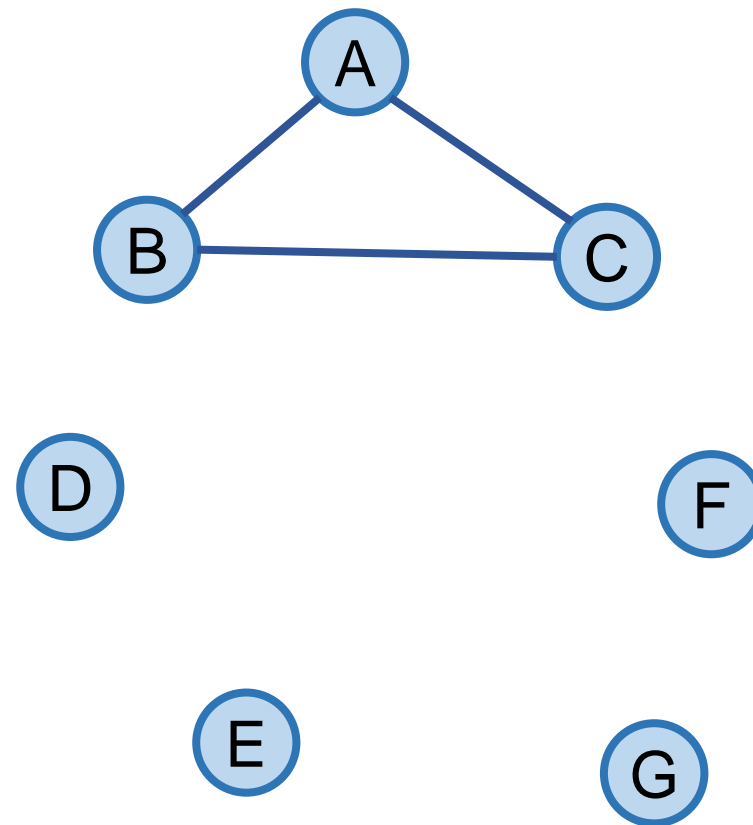
F



F



















G

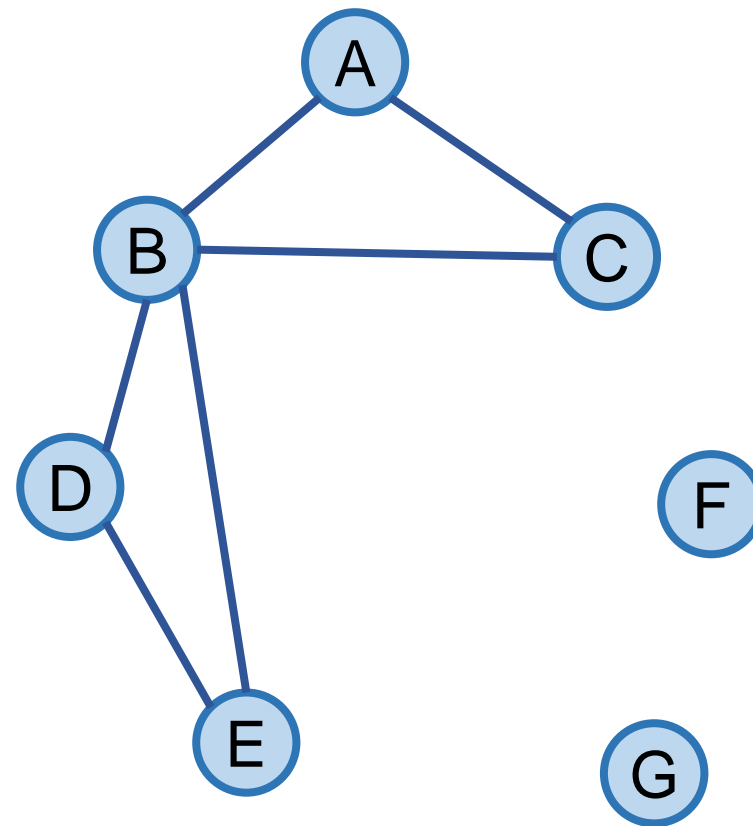


将课程表示为节点

如果两个课程之间存在约束，则在两个节点(课程)之间创建边

约束满足问题示例：考试时间

















学生:		课程:		
	1	 A	 B	 C
	2	 B	 D	 E
	3	 C	 E	 F
	4	 E	 F	 G

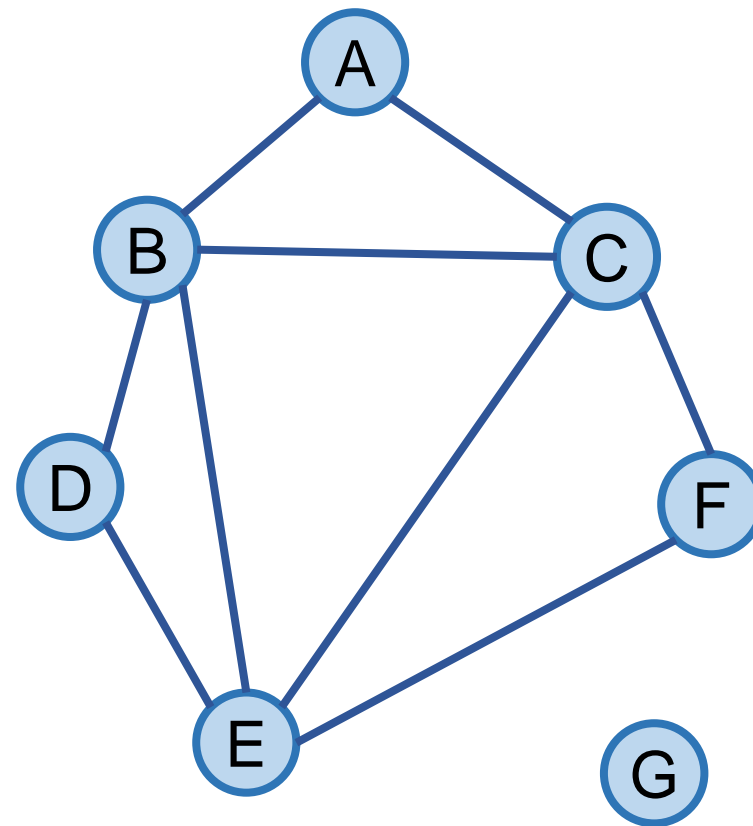


将课程表示为节点

如果两个课程之间存在约束，则在两个节点(课程)之间创建边

约束满足问题示例：考试时间

















学生:		课程:		
	1	 A	 B	 C
	2	 B	 D	 E
	3	 C	 E	 F
	4	 E	 F	 G

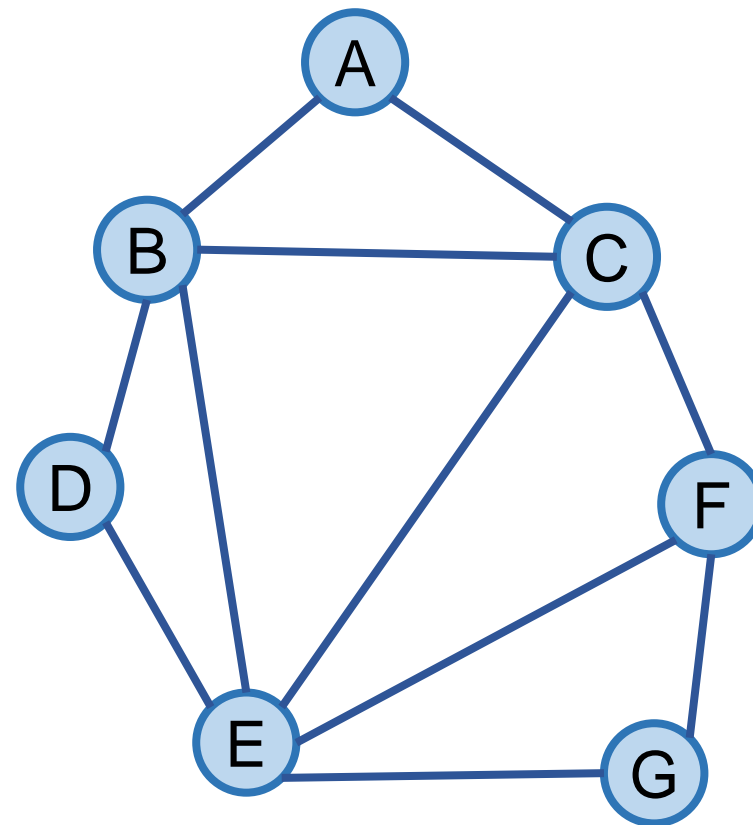


将课程表示为节点

如果两个课程之间存在约束，则在两个节点(课程)之间创建边

约束满足问题示例：考试时间

学生:		课程:		
	1	 A	 B	 C
	2	 B	 D	 E
	3	 C	 E	 F
	4	 E	 F	 G

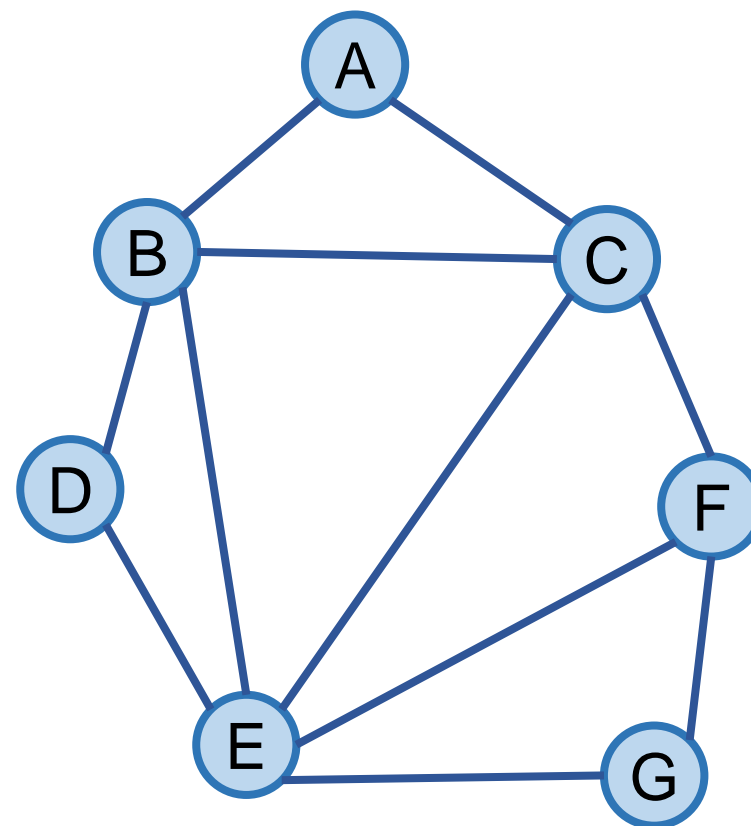


将课程表示为节点

如果两个课程之间存在约束，则在两个节点(课程)之间创建边

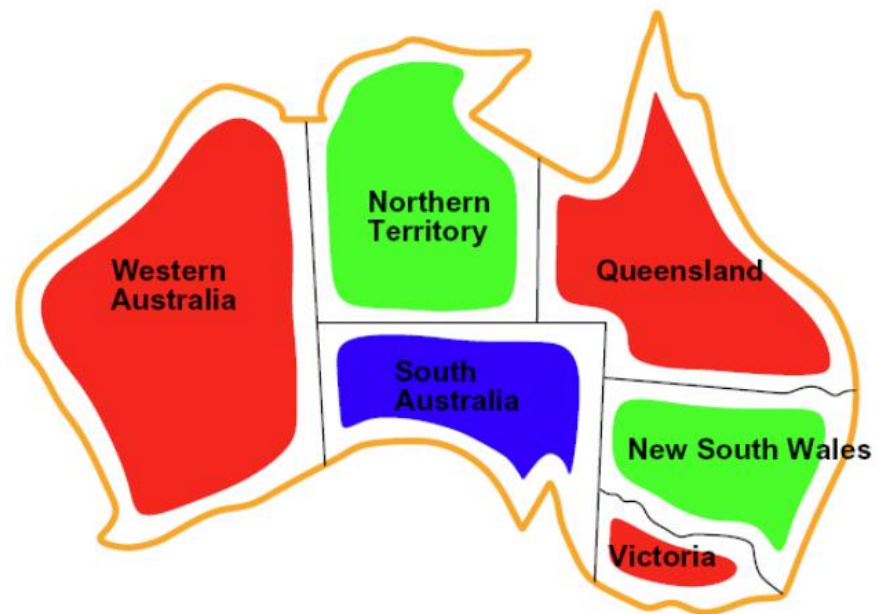
约束图 Constraint Graph

- 表示变量以及变量间的约束关系的图形
 - 例如，刚刚的考试时间的示例中的每个约束都是不等式约束
 - B 的值不能等于 D 的值



练习 #8

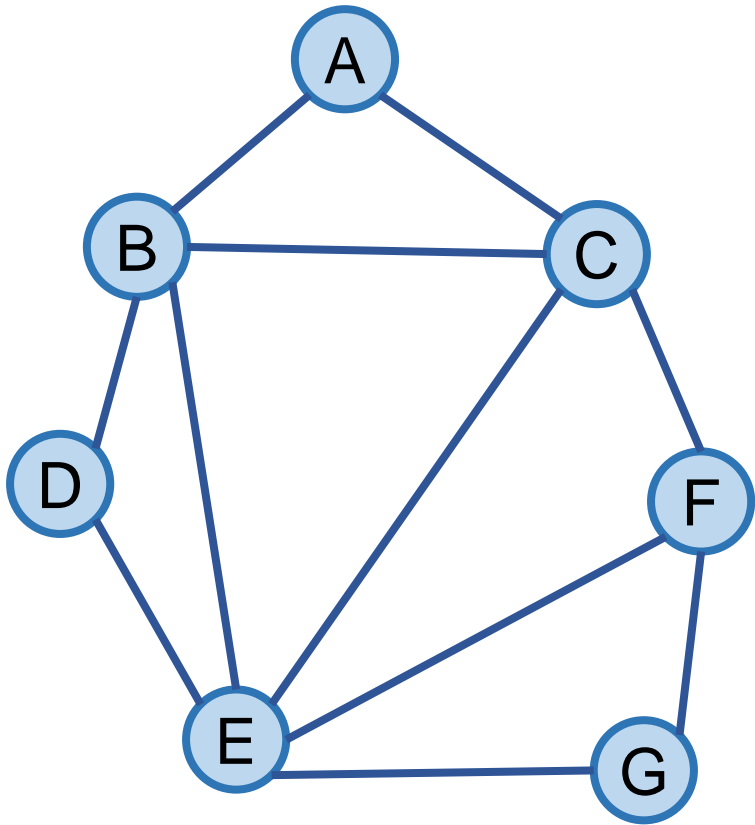
- 地图填色问题：画出约束图
 - 填的颜色为：红绿蓝
 - 相邻区域的颜色不能一样



约束满足问题 Constraint Satisfaction Problems

- 变量集 $\{X_1, X_2, \dots, X_n\}$
- 变量的定义域集 $\{D_1, D_2, \dots, D_n\}$
- 约束集 C
 - $X_1 \neq X_2$
 - $X_1 = X_2 + 1$

考试时间安排 Exam Schedule



变量 Variables

$\{A, B, C, D, E, F, G\}$

定义域 Domains

{周一、周二、周三}

for each variable

约束 Constraint

$\{A \neq B, A \neq C, B \neq C, B \neq D, B \neq E, C \neq E, C \neq F, D \neq E, E \neq F, E \neq G, F \neq G\}$

解 Solution

$\{A = \text{Monday}, B = \text{Tuesday}, C = \text{Wednesday}, \dots\}$

练习 #9. 数独 Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

变量 Variables

定义域 Domains

约束 Constraint

解 Solution

练习 #10. Cryptarithmic

$$\begin{array}{r} \text{S E N D} \\ + \text{M O R E} \\ \hline \text{M O N E Y} \end{array}$$

$$\begin{array}{r} 9567 \\ + 1085 \\ \hline 10652 \end{array}$$

变量 Variables

定义域 Domains

约束 Constraint

解 Solution

约束满足问题的现实例子

- 时间安排问题
 - 面试
- 时间表问题
 - 排课时间和地点、体育比赛时间和地点
- 分配问题
 - 老师课程分配、外卖员分配、滴滴司机分配
- 运输调度
 - 飞机、火车时刻表

现实例子 #1：多城市航班搜索

The screenshot shows the 'Cheap multi-city travel planning' interface on the PrestoTrip website. It includes fields for 'Departing city' (San Francisco, CA, USA), 'Depart date' (12/01/2019), 'Flexible date' (checked), 'Return city' (San Francisco, CA, USA), and 'Return date'. There are also dropdowns for 'Cheapest Flights' and a range of '2 nights' to '5 nights'. A 'SEARCH' button is visible. Below the main form, there are 'Trip suggestions' and a 'NEAR MY DESTINATIONS' button.

<http://prestotrip.com/>

变量 Variables

{SF-LDN, LDN-SF...}

定义域 Domains

{U901, BA284, ...}

约束 Constraint

$\{\text{Date}(\text{SF-LDN}) = 2019-12-01,$

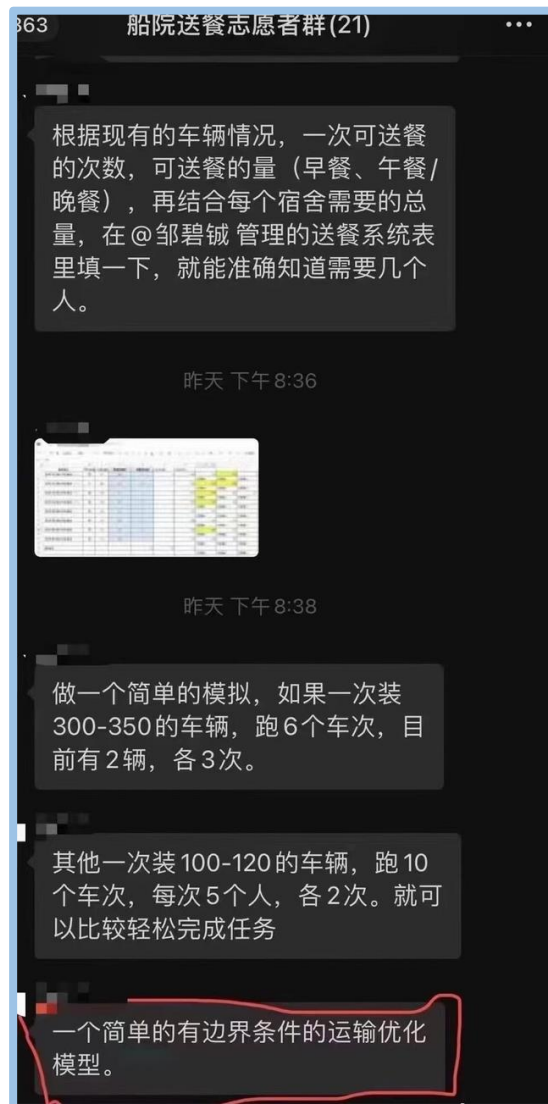
$\text{Date}(\text{SF-LDN}) + 2 < \text{Date}(\text{LDN-SF}) < \text{Date}(\text{SF-LDN}) + 5,$

$\text{Minimize}(\text{Price}(\text{SF-LDN}) + \text{Price}(\text{LDN-SF}))\}$

解 Solution

{SF-LDN = U901, LDN-SF = U949, ...}

现实例子 #2: 疫情期间送餐调度



约束 Constraint

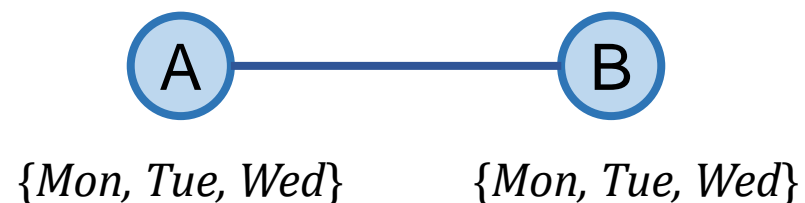
- 硬约束 Hard constraint
 - **必须**满足的约束
 - 例如，数独：一行不能有两个相同的数字
- 软约束 Soft constraint
 - 一些解**优于**其他解的约束
 - 例如，首选更便宜的航班

约束 Constraint

- 一元约束 Unary constraint
 - 仅涉及一个变量的约束
 - 例如, $\{A \neq Monday\}$
- 二元约束 Binary constraint
 - 涉及两个变量的约束
 - 例如, $\{A \neq B\}$

节点一致性 Node consistency

- 所有变量的赋值均满足每个变量的一元约束



$$\{A \neq Mon, B \neq Mon, B \neq Tue, A \neq B\}$$

节点一致性 Node consistency

- 所有变量的赋值均满足每个变量的一元约束



$$\{A \neq Mon, B \neq Mon, B \neq Tue, A \neq B\}$$

节点一致性 Node consistency

- 所有变量的赋值均满足每个变量的一元约束



$$\{A \neq Mon, B \neq Mon, B \neq Tue, A \neq B\}$$

弧一致性 Arc consistency

- 所有变量的赋值均满足所有变量的**二元约束**
 - 为了使 X 对 Y 满足弧一致性，需要从 X 的定义域中删除不满足二元约束的值，直到 X 的定义域中的每个值都在 Y 的定义域中存在一个值满足弧一致性



$$\{A \neq Mon, B \neq Mon, B \neq Tue, A \neq B\}$$

弧一致性 Arc consistency

- 所有变量的赋值均满足所有变量的**二元约束**
 - 为了使 X 对 Y 满足弧一致性，需要从 X 的定义域中删除不满足二元约束的值，直到 X 的定义域中的每个值都在 Y 的定义域中存在一个值满足弧一致性



$$\{A \neq Mon, B \neq Mon, B \neq Tue, A \neq B\}$$

弧一致性 Arc consistency

- 通过修改定义域满足一个二元约束

function Revise(*problem*, *X*, *Y*):

revised = *False*

 for *x* in *X.domain*:

 if no *y* in *Y.domain* 满足 (*X*, *Y*) 的约束:

 从 *X.domain* 删除 *x*

revised = *True*

 return *revised*

弧一致性 Arc consistency

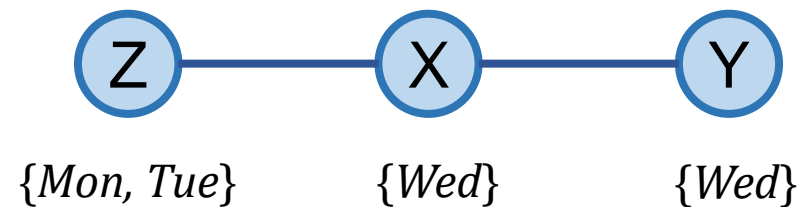
function AC-3(*problem*): 通过修改定义域满足整个问题的全部二元约束

queue = *problem* 中的每个弧 (二元约束)

while *queue* non-empty:

$(X, Y) = \text{Dequeue}(\textit{queue})$ 从*queue*中选择并移除一个二元约束

if $\text{Revise}(\textit{problem}, X, Y)$ is *True*:



弧一致性 Arc consistency

function AC-3(*problem*): 通过修改定义域满足整个问题的全部二元约束

queue = *problem* 中的每个弧 (二元约束)

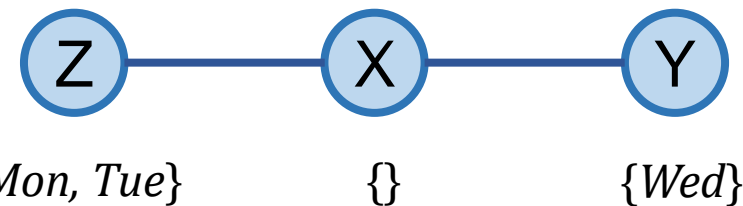
while *queue* non-empty:

$(X, Y) = \text{Dequeue}(\textit{queue})$ 从 \textit{queue} 中选择并移除一个二元约束

if $\text{Revise}(\textit{problem}, X, Y)$ is *True*:

if size of $X.\textit{domain} == 0$:

return *False*



弧一致性 Arc consistency

function AC-3(*problem*): 通过修改定义域满足整个问题的全部二元约束

queue = *problem* 中的每个弧 (二元约束)

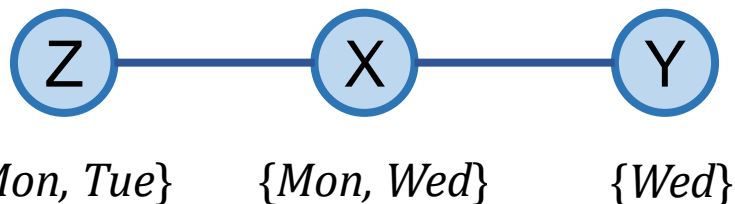
while *queue* non-empty:

(X, Y) = Dequeue(*queue*) 从*queue*中选择并移除一个二元约束

if Revise(*problem*, *X*, *Y*) is *True*:

if size of *X.domain* == 0:

return *False*



弧一致性 Arc consistency

function AC-3(*problem*): 通过修改定义域满足整个问题的全部二元约束

queue = *problem* 中的每个弧 (二元约束)

while *queue* non-empty:

$(X, Y) = \text{Dequeue}(\textit{queue})$ 从*queue*中选择并移除一个二元约束

if $\text{Revise}(\textit{problem}, X, Y)$ is *True*:

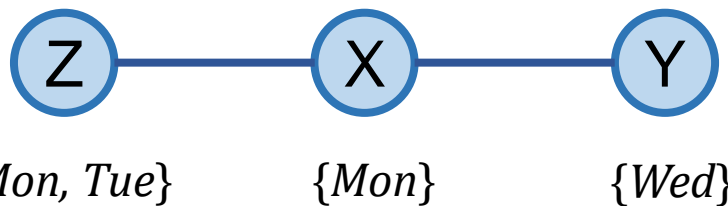
if size of $X.\textit{domain} == 0$:

return *False*

for each Z in $X.\textit{neighbors} - \{Y\}$: 修改X的定义域后, X可能和其他变量不再弧一致

$\text{Enqueue}(\textit{queue}, (Z, X))$ 在*queue*中添加还需要检查的二元约束

return *True*



练习 #11

- 有一个四位数

- 四位数是奇数

- 四位数的每个位置的数字由1-5构成

- 四位数每个位置的数字都比它左侧位置的数字更大

A	B	C	D

- 问题:

- ① 写出这个约束满足问题的变量、定义域、和约束

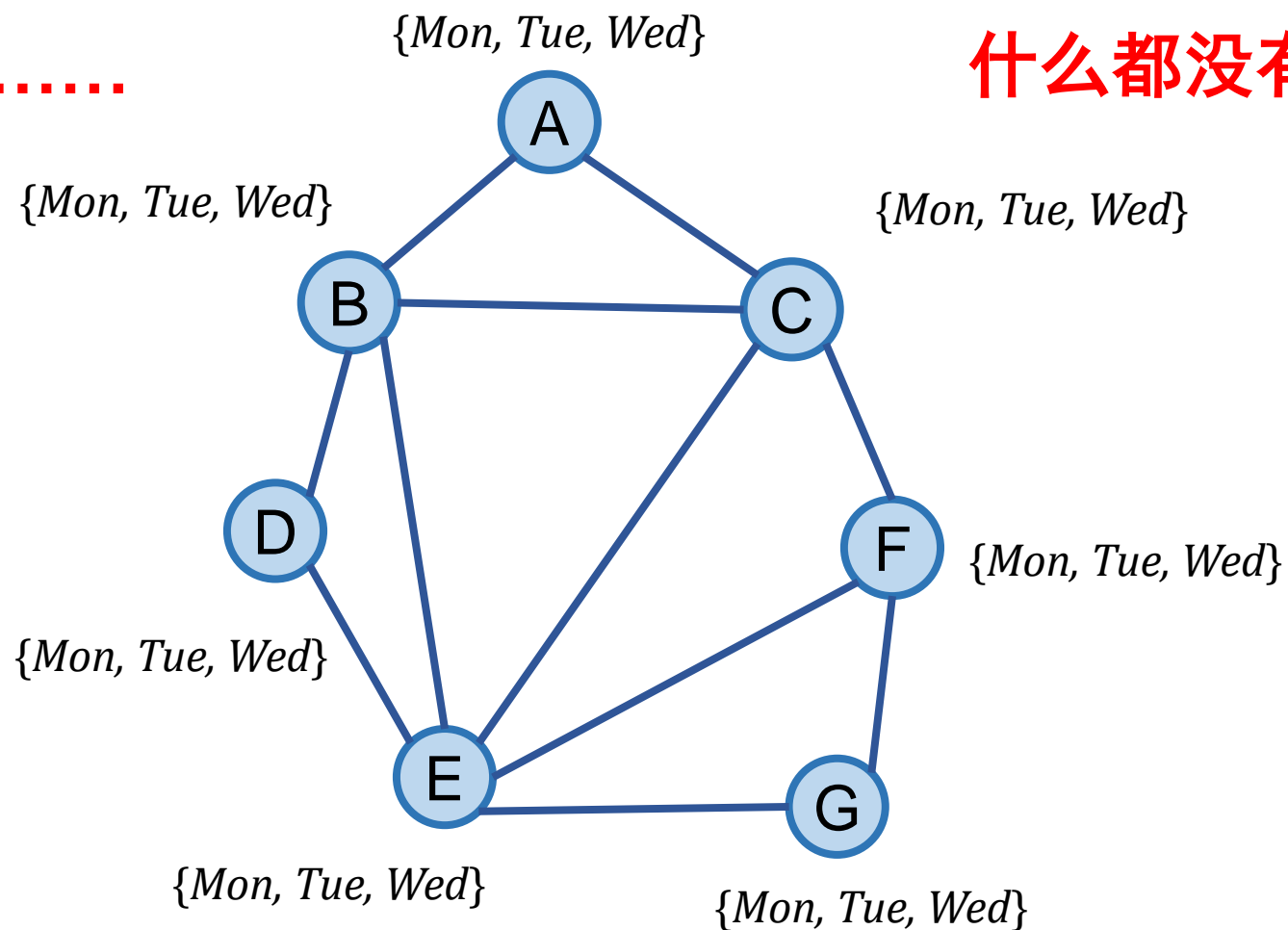
- ② 执行了节点一致性后，变量的定义域变成了什么样子？

- ③ 执行了弧一致性(AC-3)后，变量的定义域变成了什么样子？

考试时间安排

运行了AC-3后.....

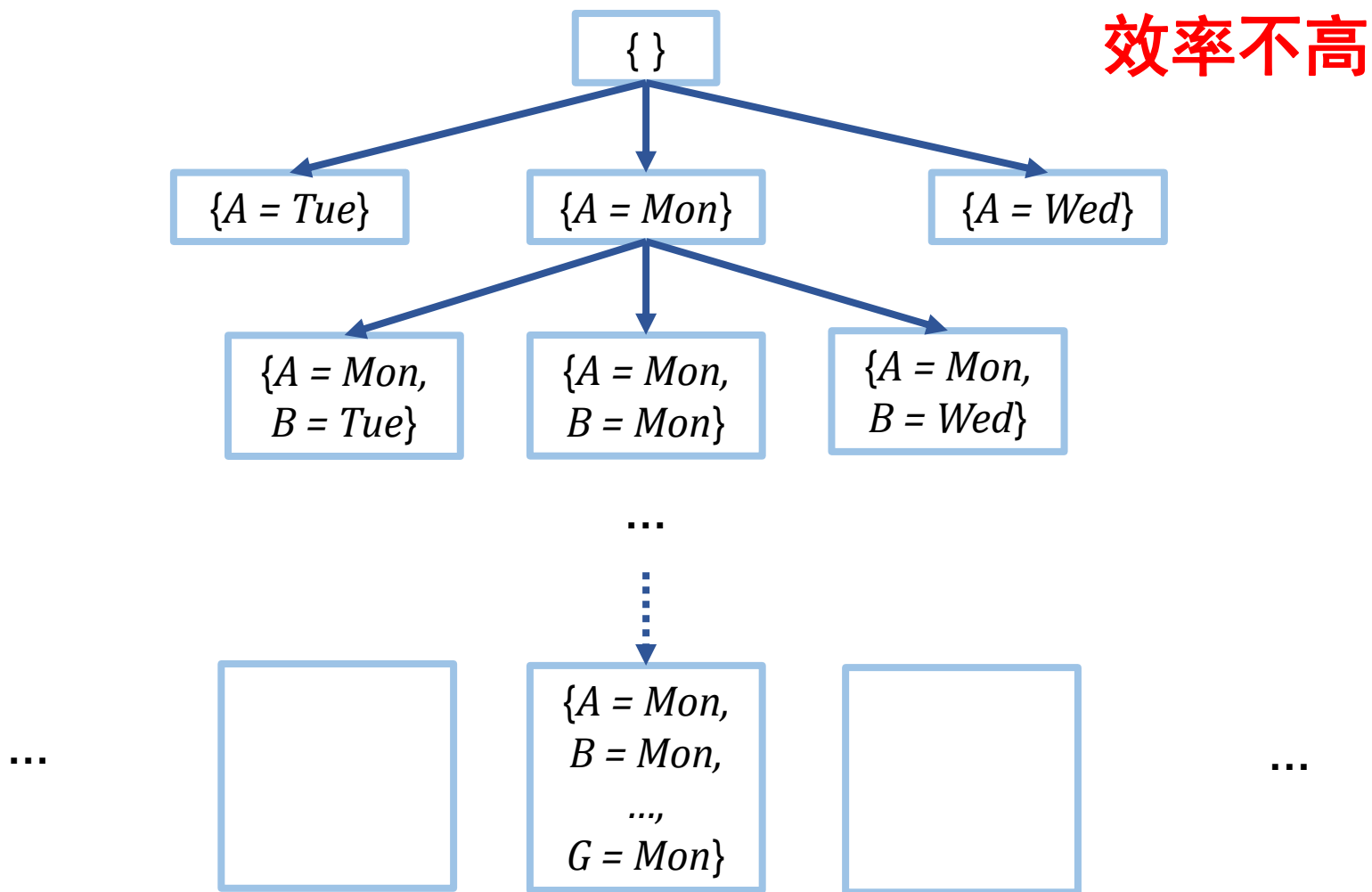
什么都没有改变



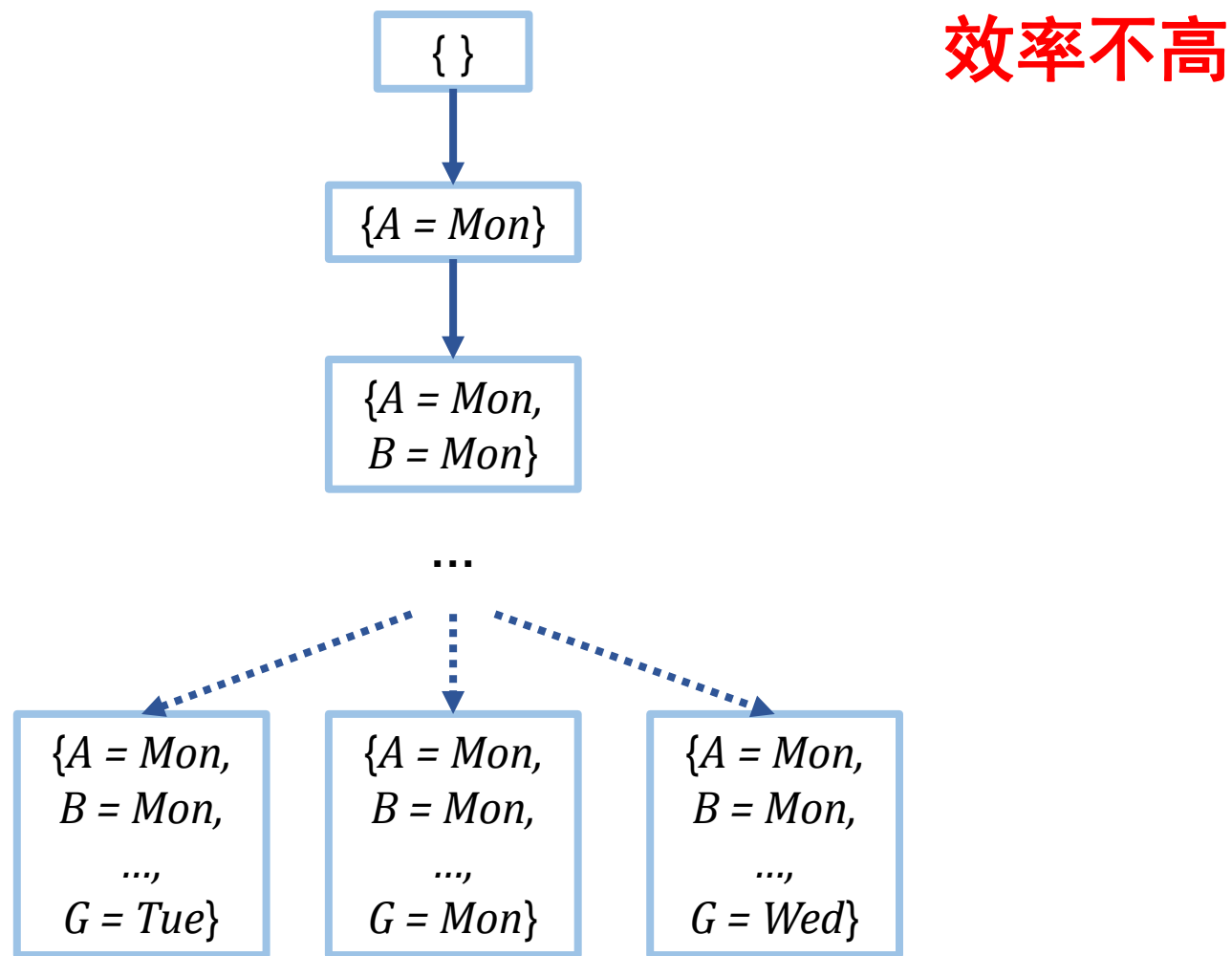
搜索问题 Search Problems

- 初始状态 Initial state
 - 每个变量都还没有被赋值
- 行动 Actions
 - 对变量进行赋值, 添加 {variable = value} 到考试时间安排中
- 转换模型 Transition model
 - 展示赋值后的新的考试时间安排
- 目标测试 Goal test
 - 所有变量都已经被赋值并且所有约束都被满足了
- 路径耗散函数 Path cost function
 - 每个路径的成本是一样的 All paths have same cost

考试时间安排 – Breadth first search



考试时间安排 – Depth first search

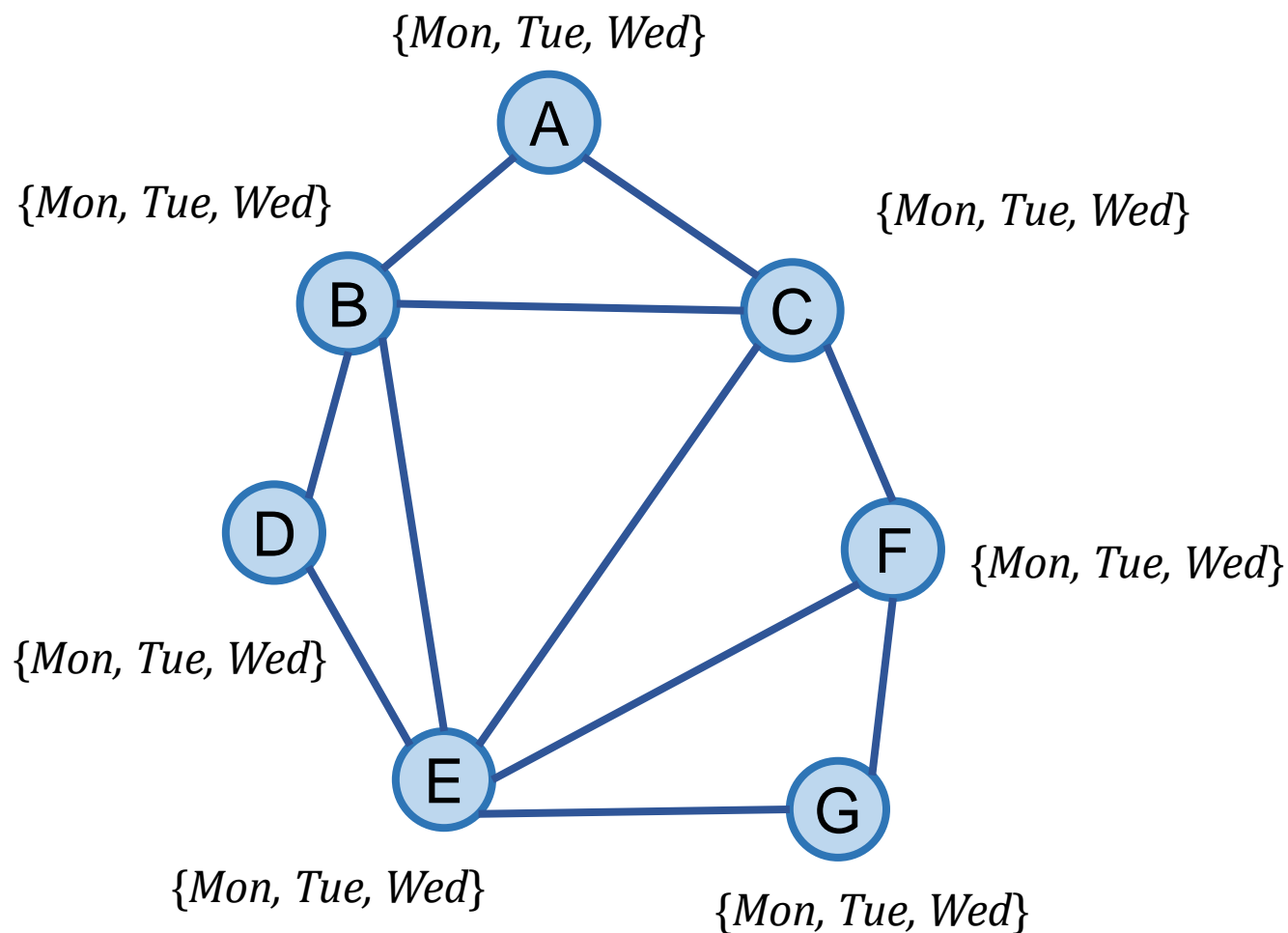


回溯搜索 Backtracking search

- 使用深度优先搜索
- 每次对一个变量赋值
 - 每步都需要目标测试
 - 对一个变量赋值后，检查约束是否满足
 - 如果没有违背任何约束条件，对下一个变量赋值
 - 如果有约束条件没有满足
 - 尝试赋值定义域中的另一个值
 - 都尝试过了也没有合适的值，返回上一个变量重新赋值

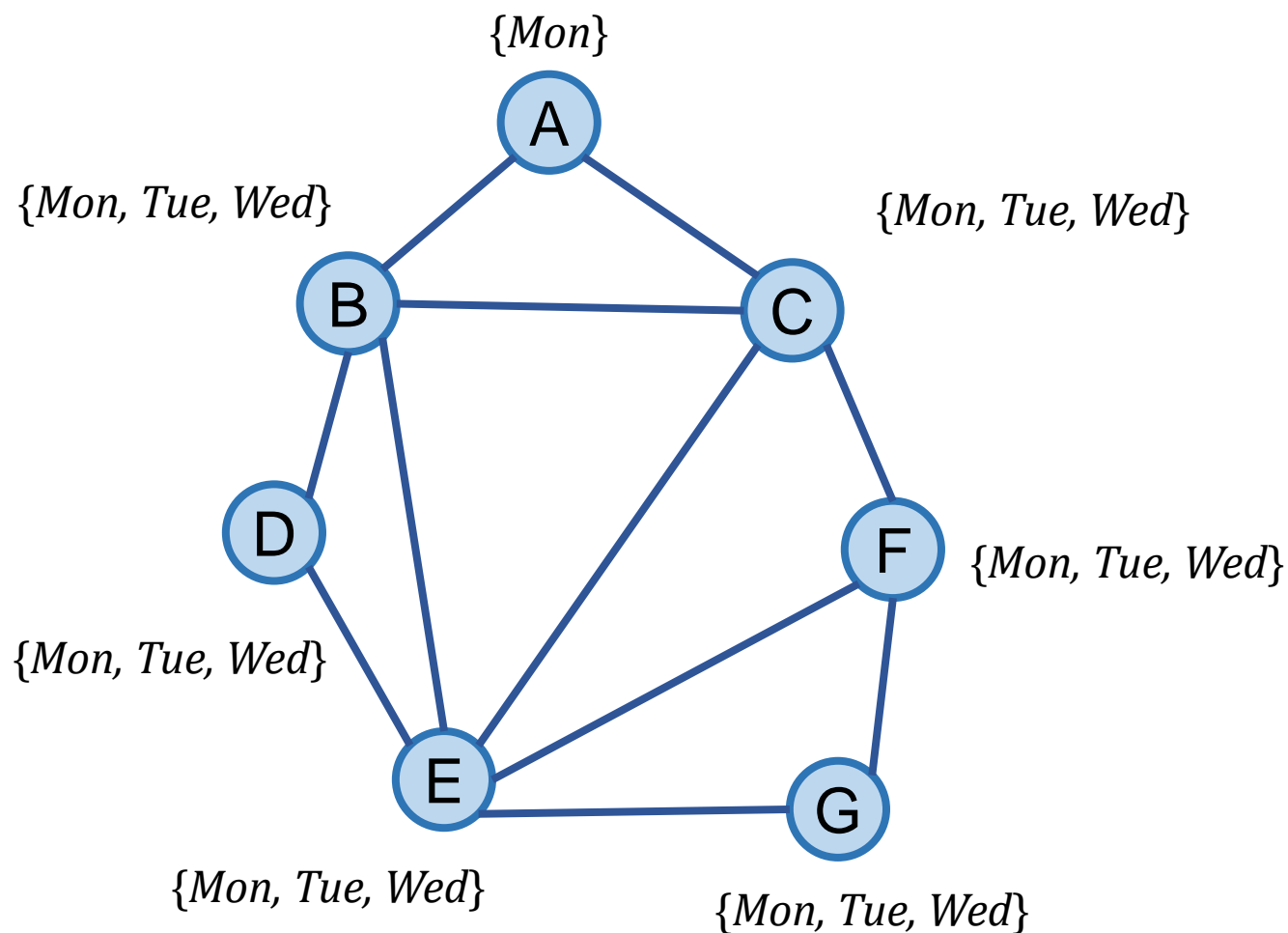
考试时间安排

- 对变量A赋值



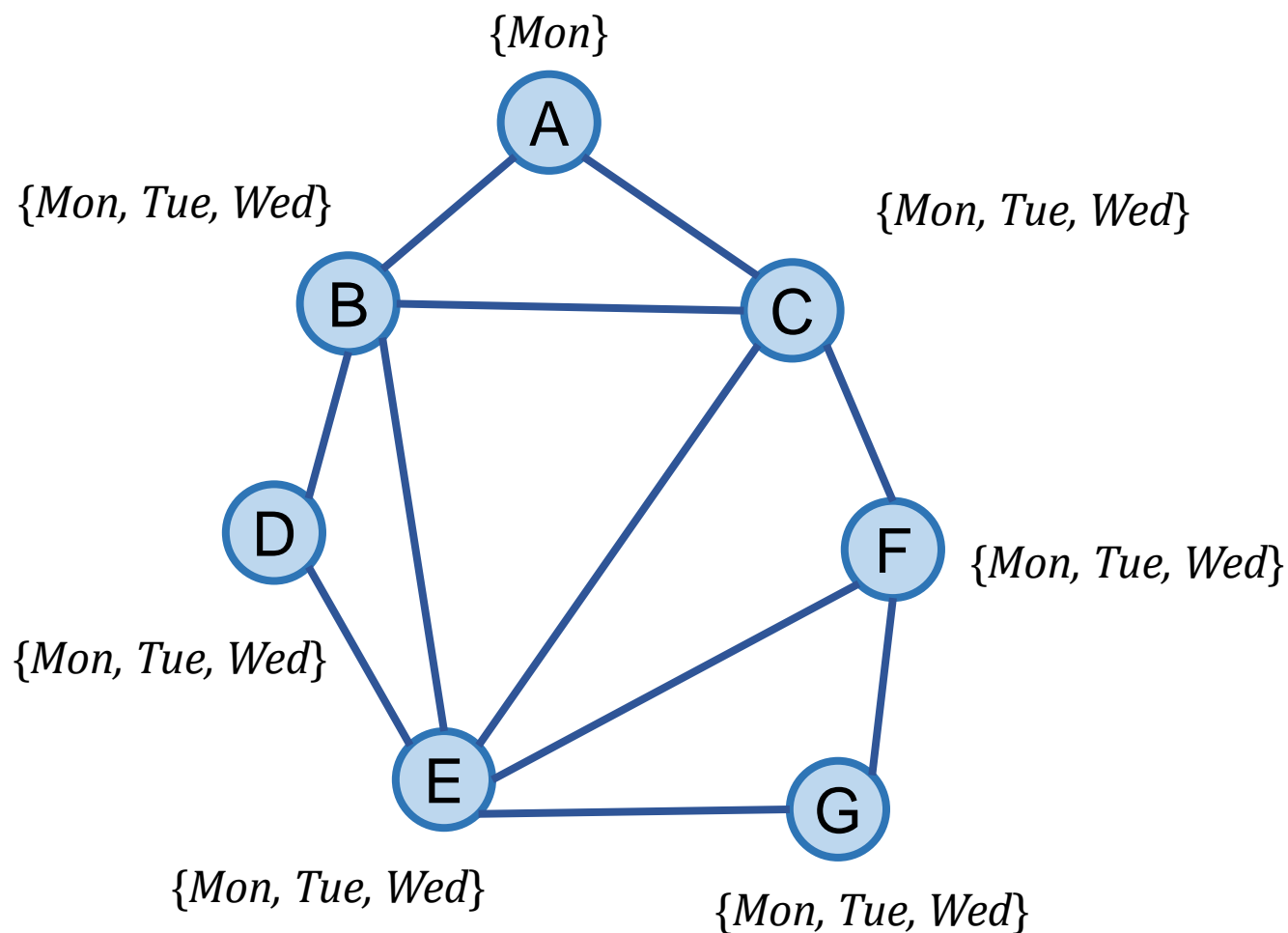
考试时间安排

- 对变量A赋值
 - 违背约束条件?
 - 没有
 - 赋值下一个变量



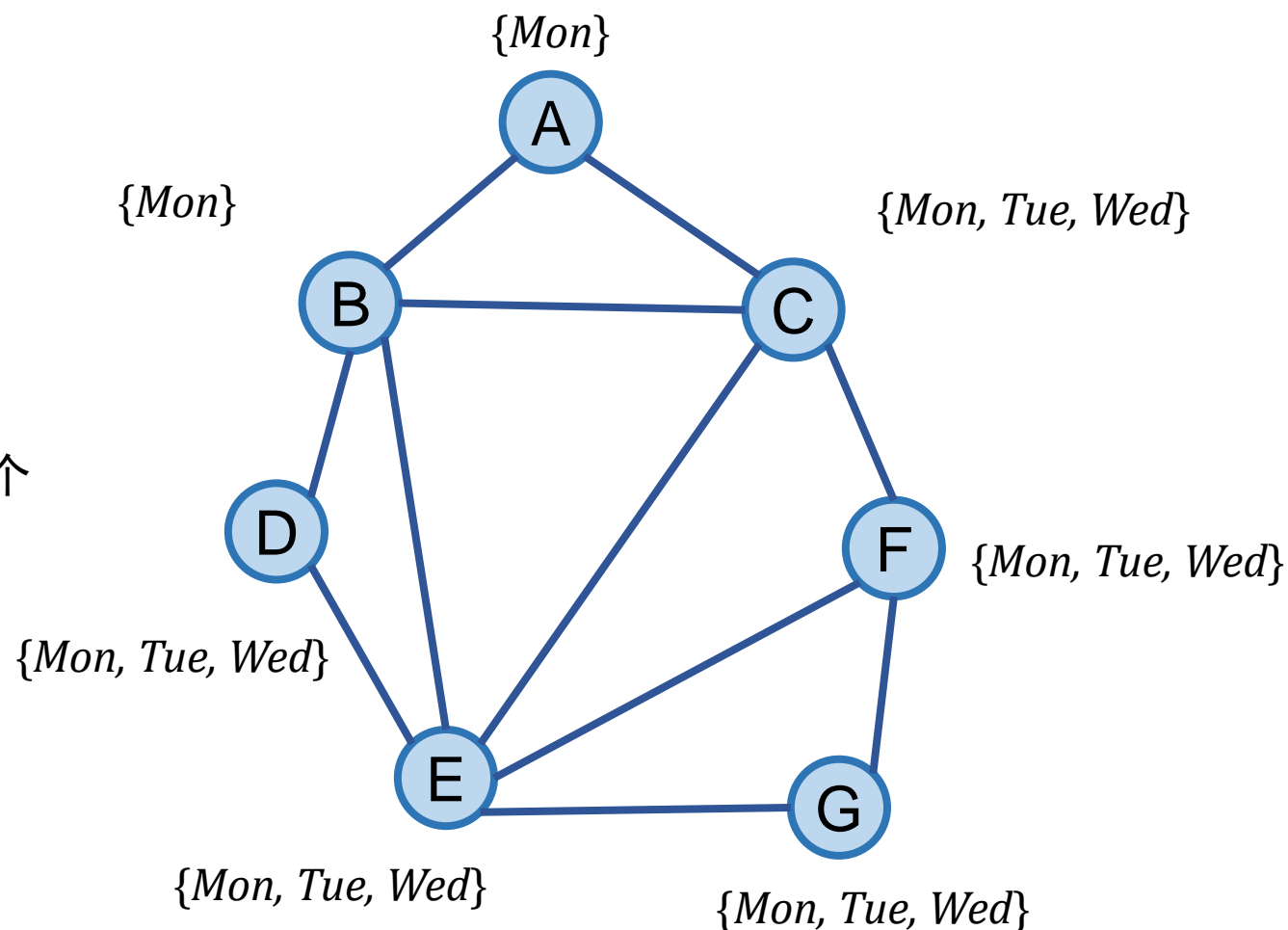
考试时间安排

- 对变量 B 赋值



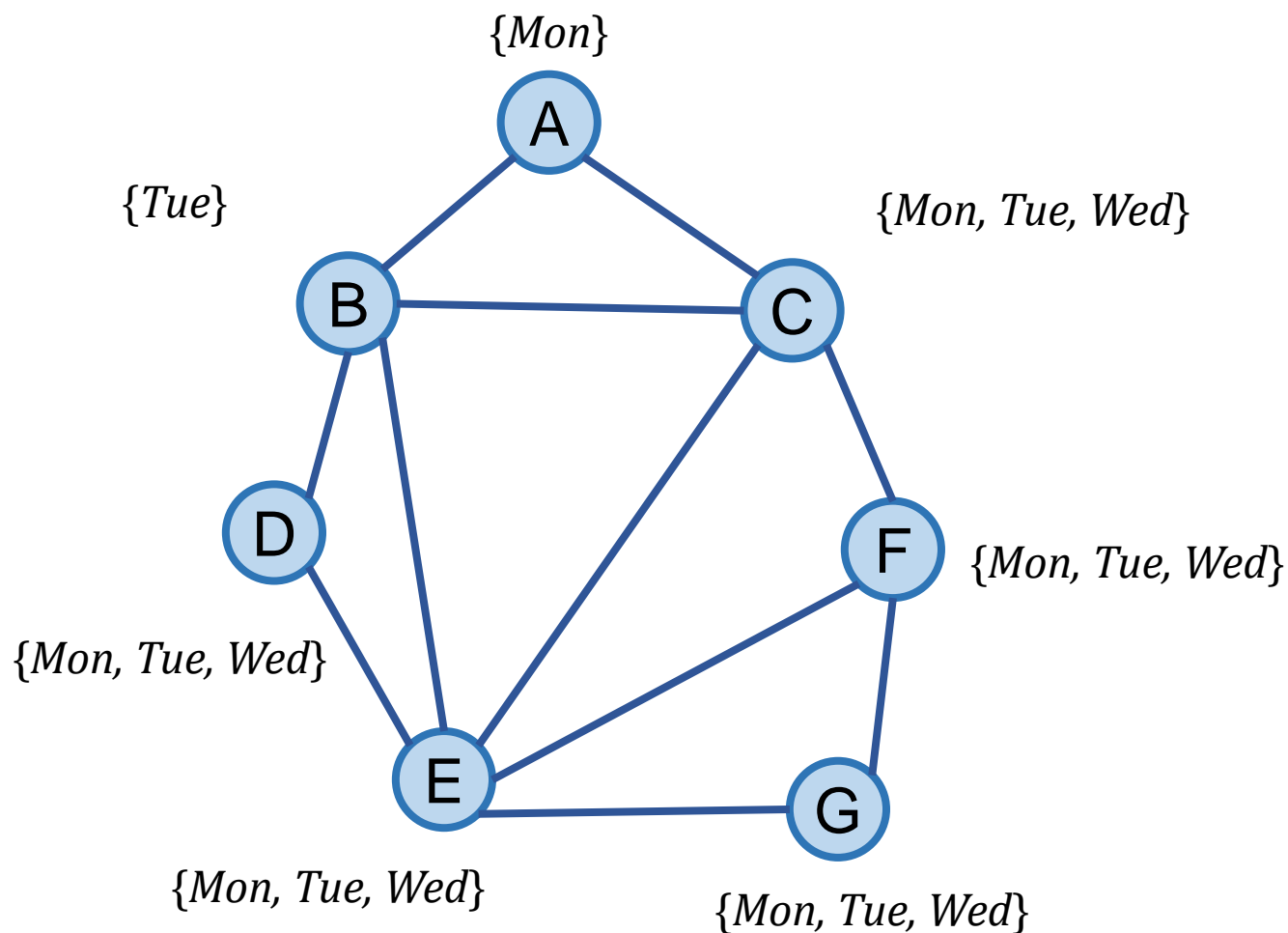
考试时间安排

- 对变量 B 赋值
 - 违背约束条件?
 - 有
 - 尝试定义域里另外一个值



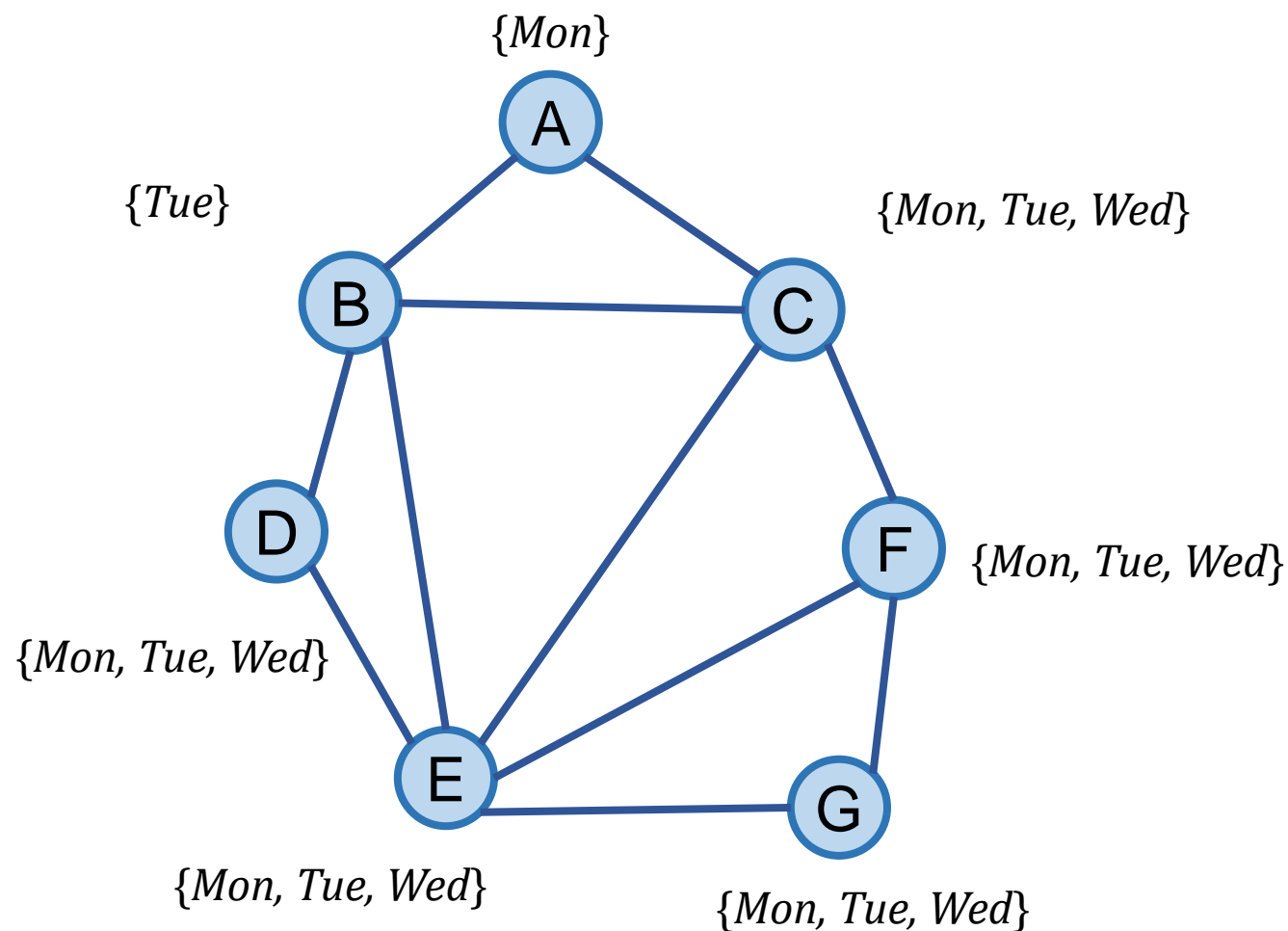
考试时间安排

- 对变量 B 赋值
 - 违背约束条件?
 - 没有
 - 赋值下一个变量



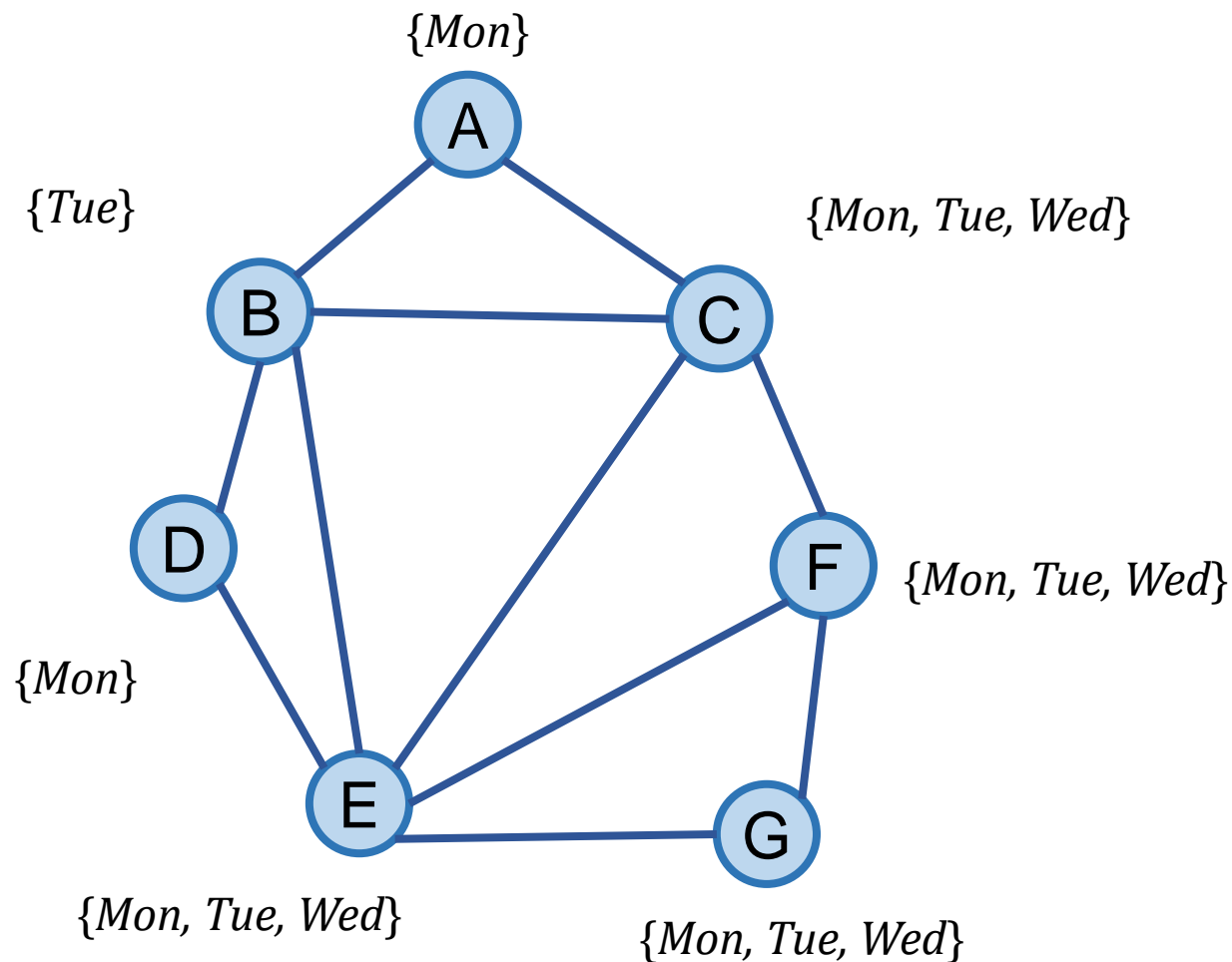
考试时间安排

- 对变量 D 赋值



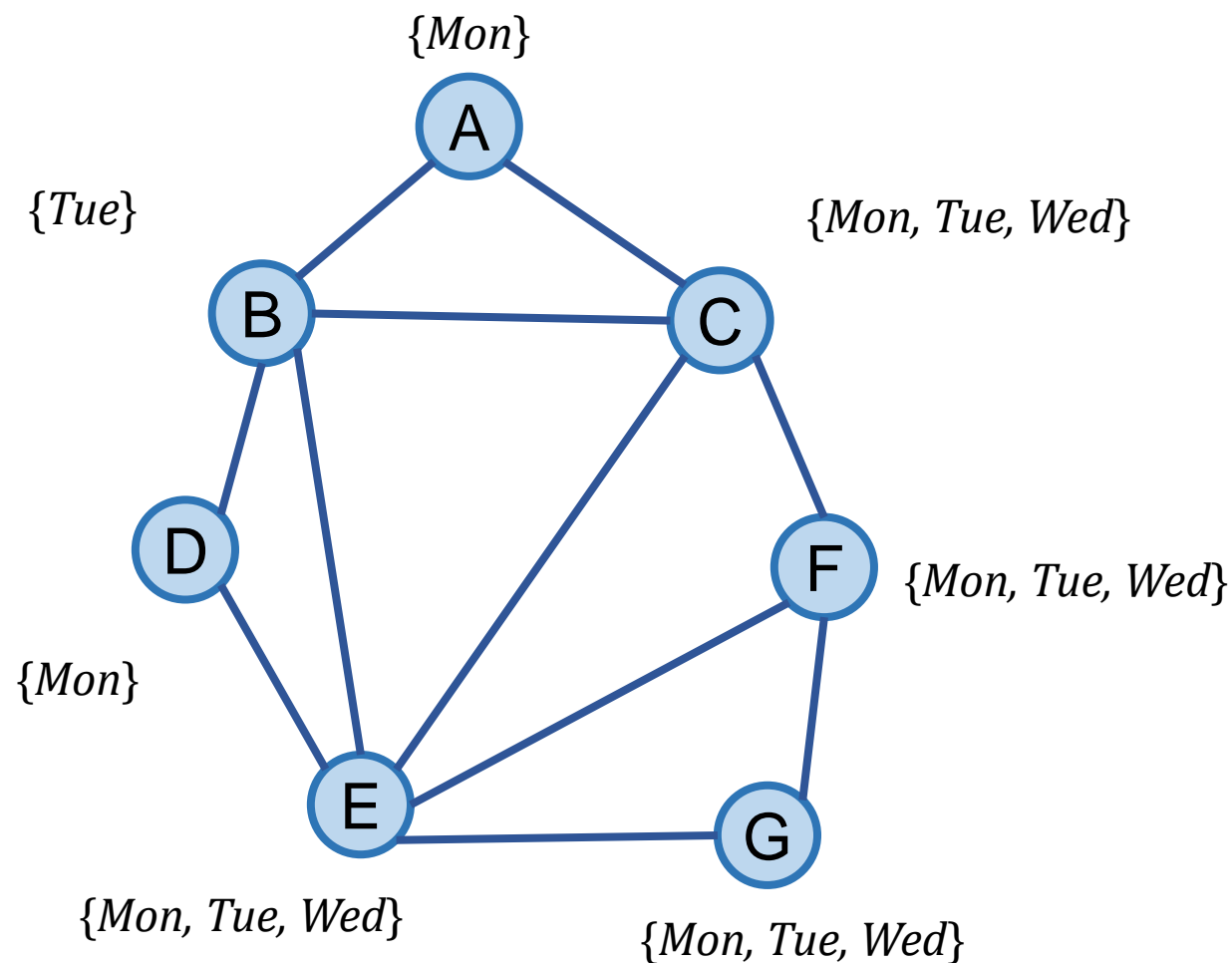
考试时间安排

- 对变量 D 赋值
 - 违背约束条件?
 - 没有
 - 赋值下一个变量



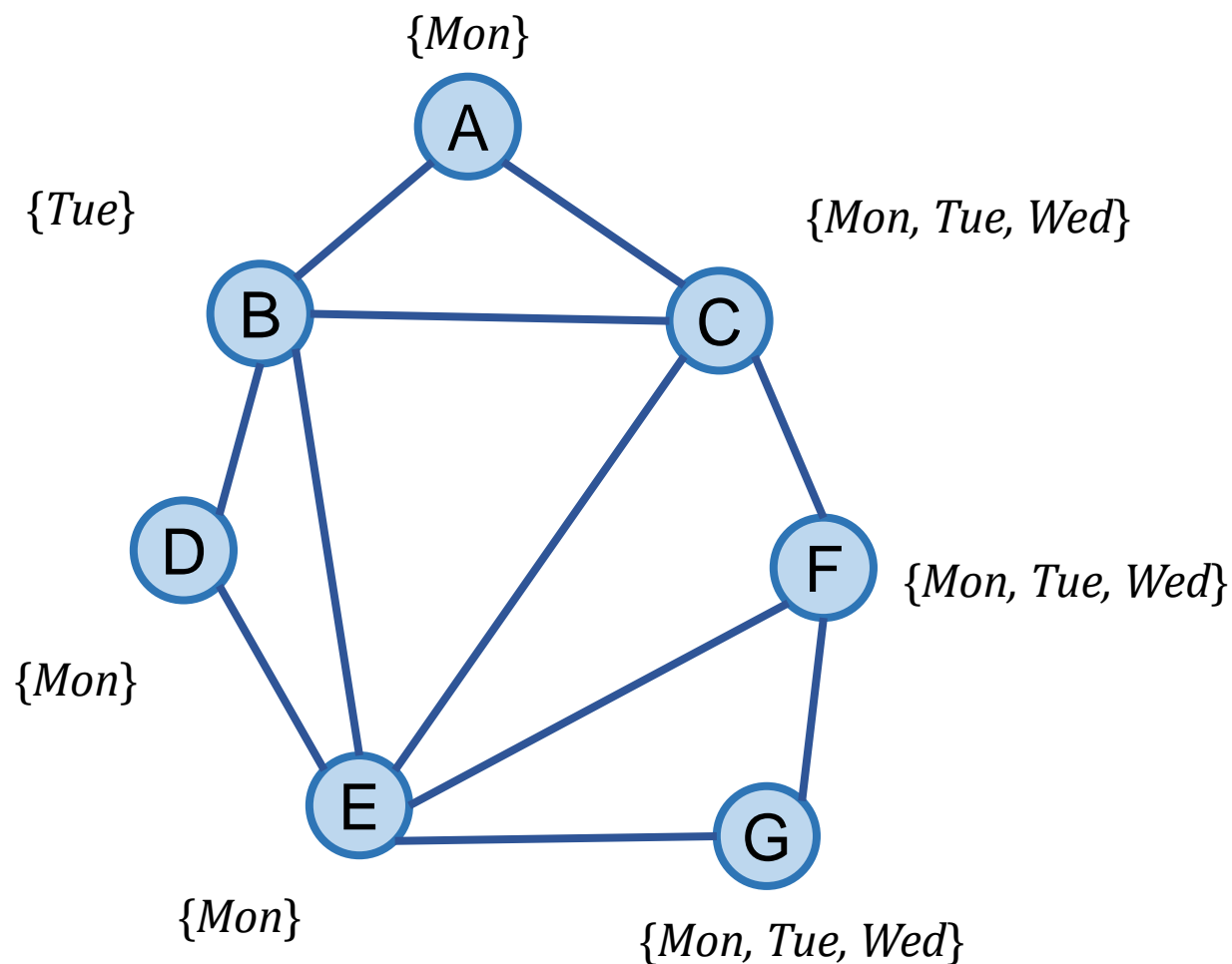
考试时间安排

- 对变量 E 赋值



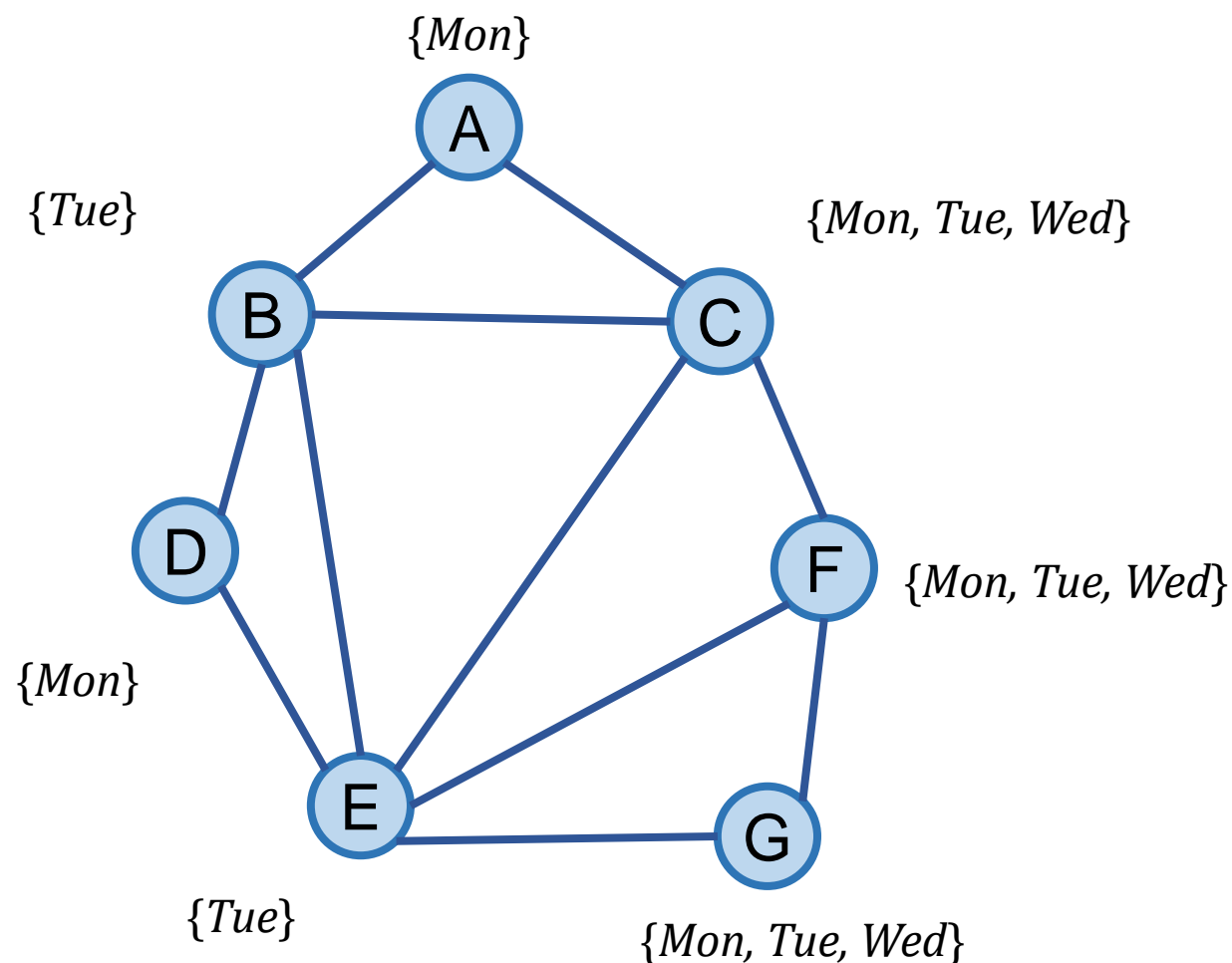
考试时间安排

- 对变量 E 赋值
 - 违背约束条件?
 - 有
 - 尝试定义域里另外一个值



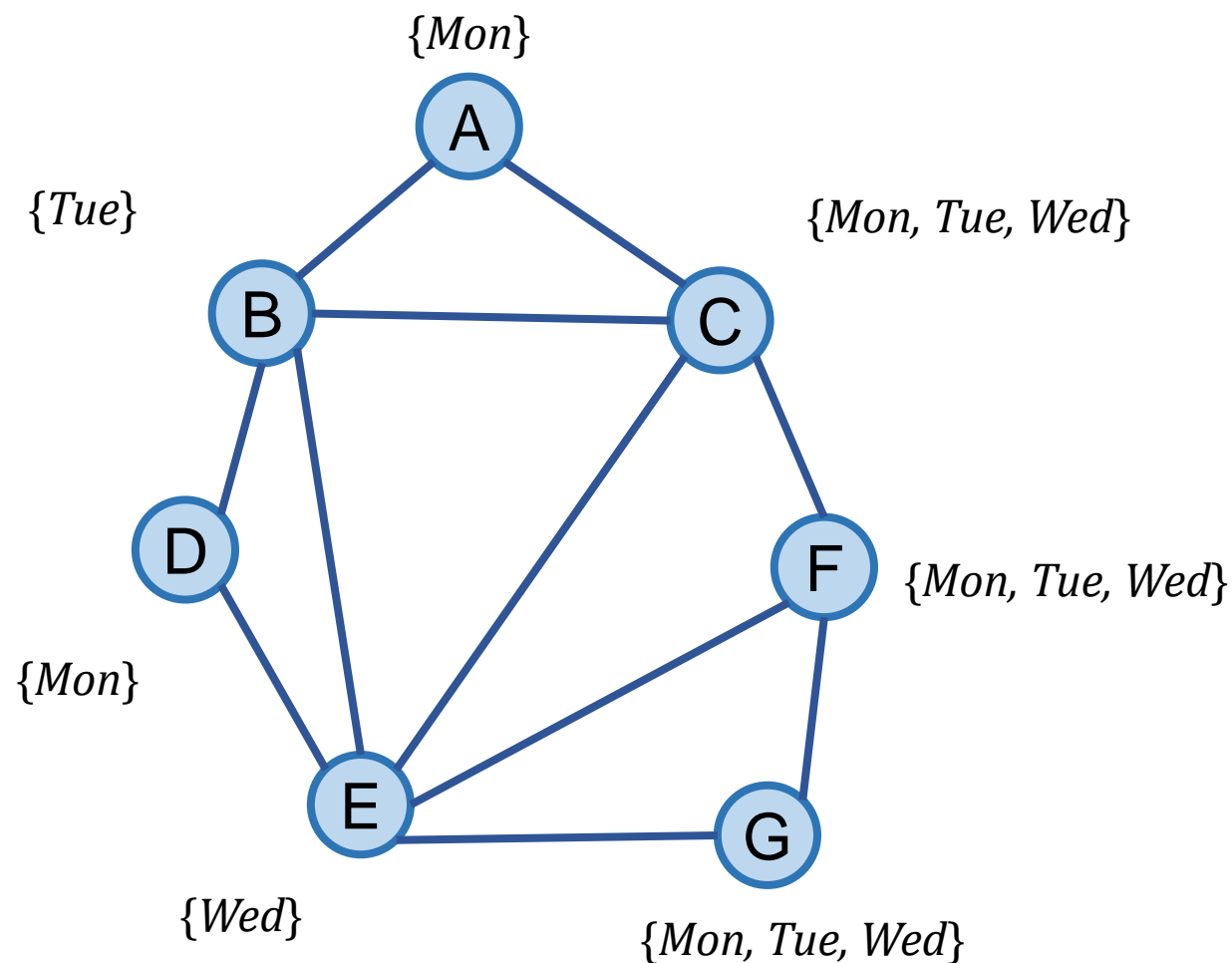
考试时间安排

- 对变量 E 赋值
 - 违背约束条件?
 - 有
 - 尝试定义域里另外一个值



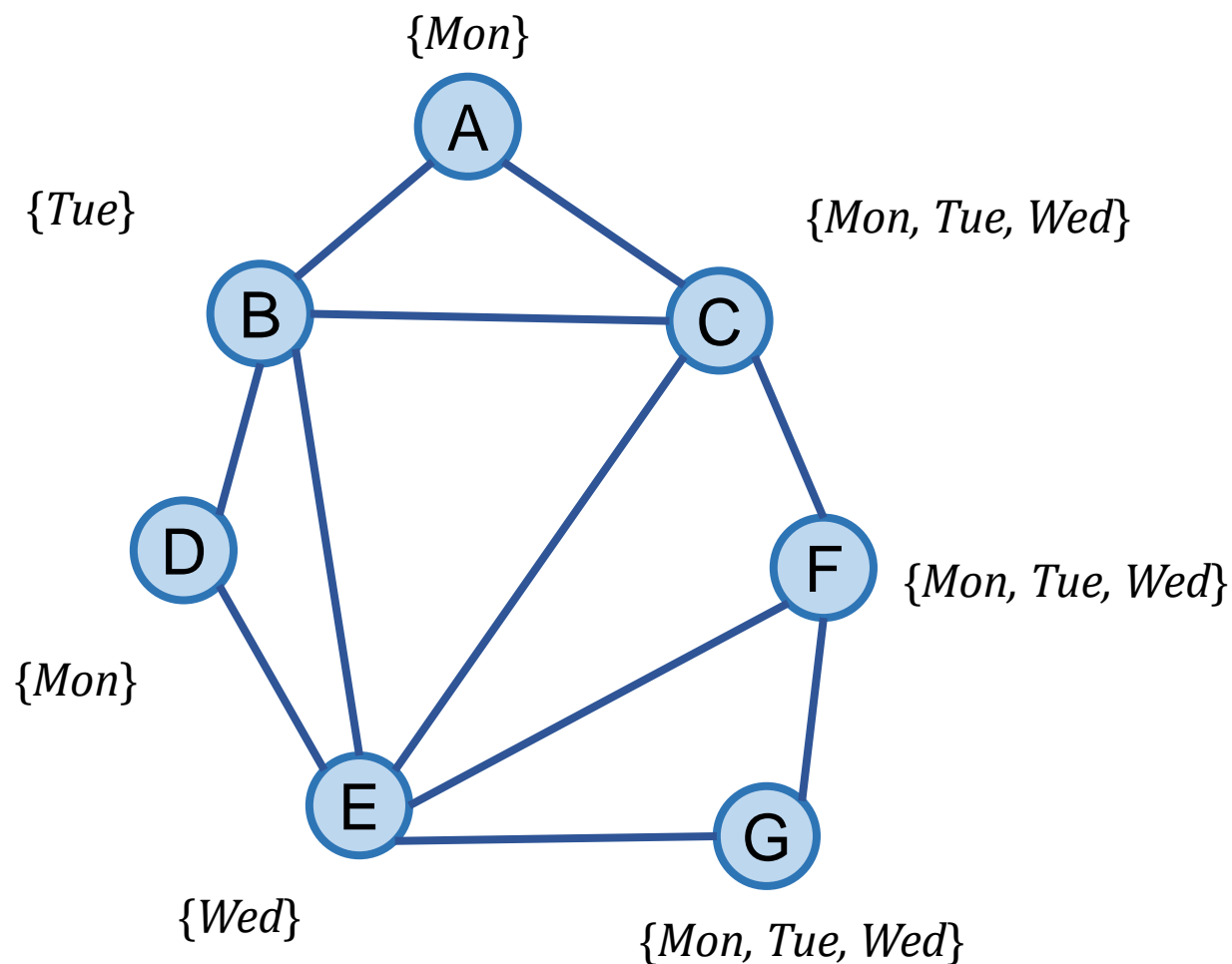
考试时间安排

- 对变量 E 赋值
 - 违背约束条件?
 - 没有
 - 赋值下一个变量



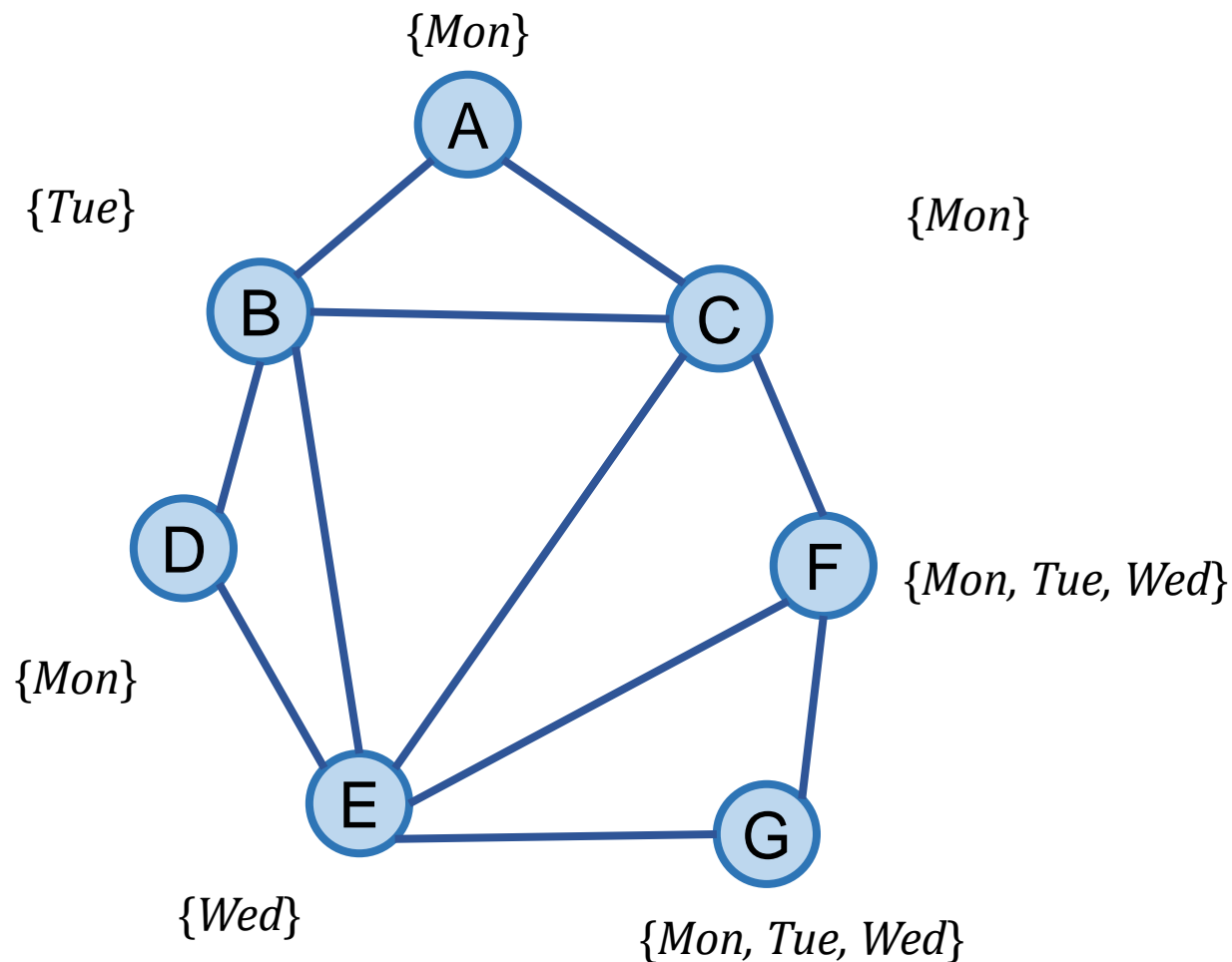
考试时间安排

- 对变量 C 赋值



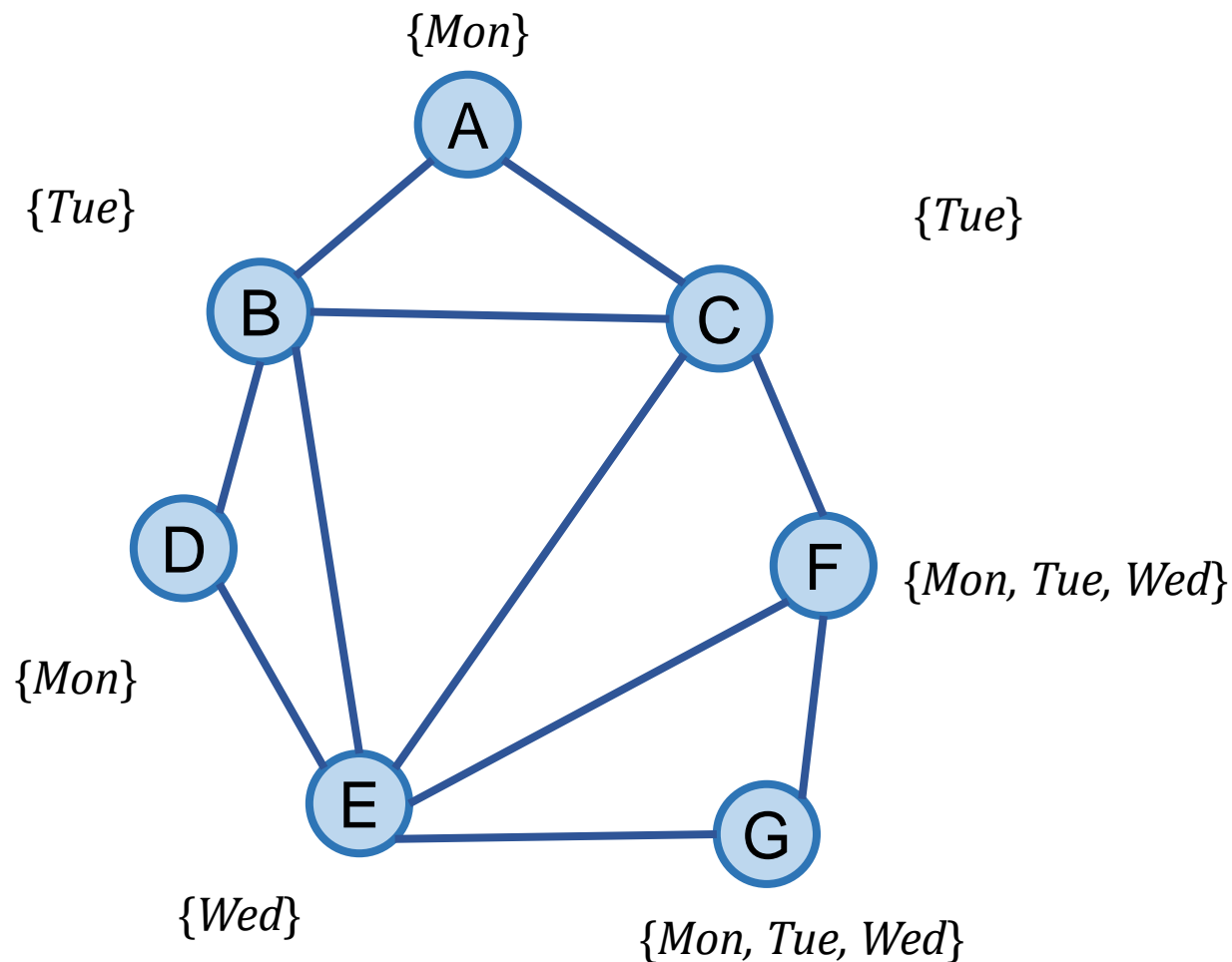
考试时间安排

- 对变量 C 赋值
 - 违背约束条件?
 - 有
 - 尝试定义域里另外一个值



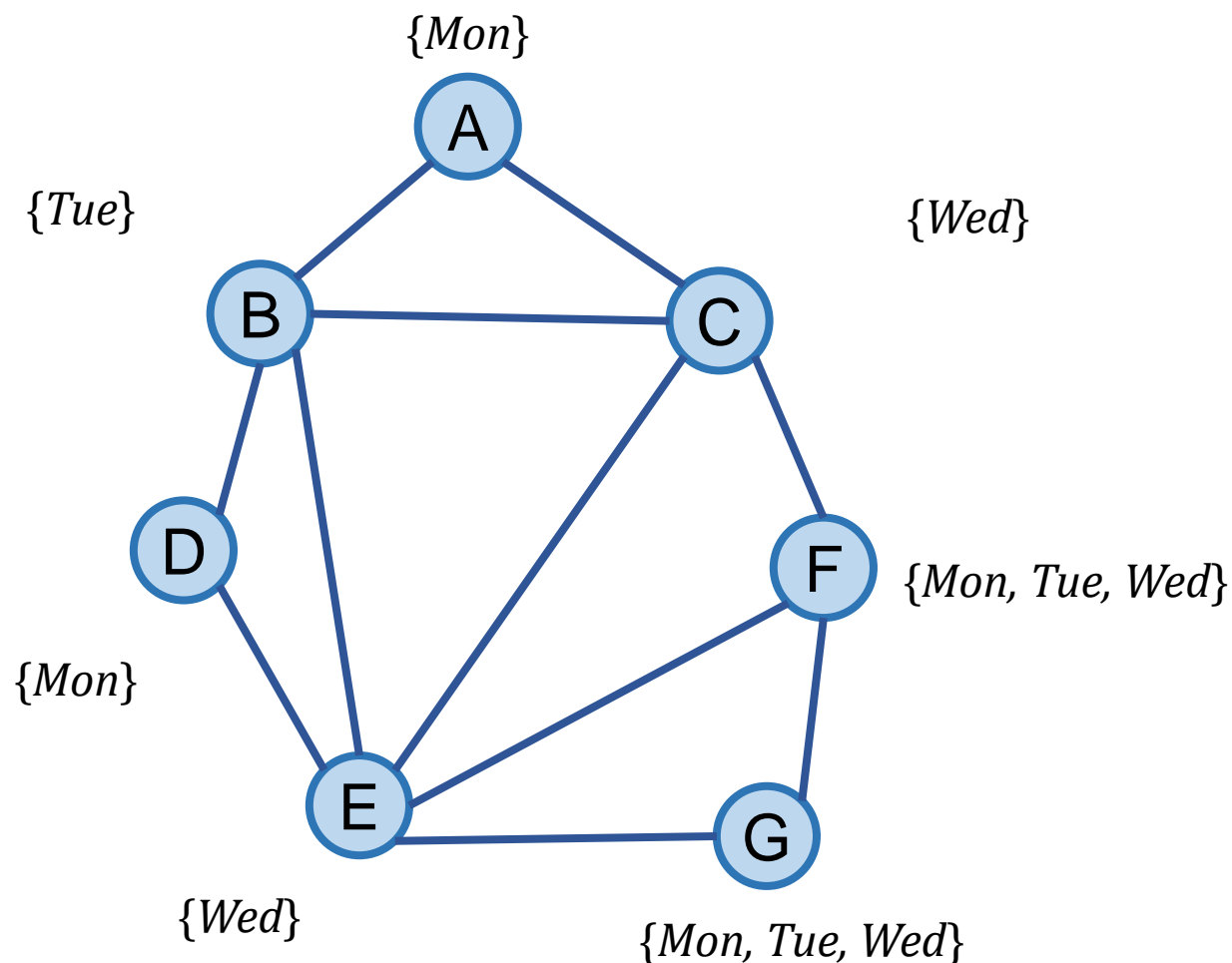
考试时间安排

- 对变量 C 赋值
 - 违背约束条件?
 - 有
 - 尝试定义域里另外一个值



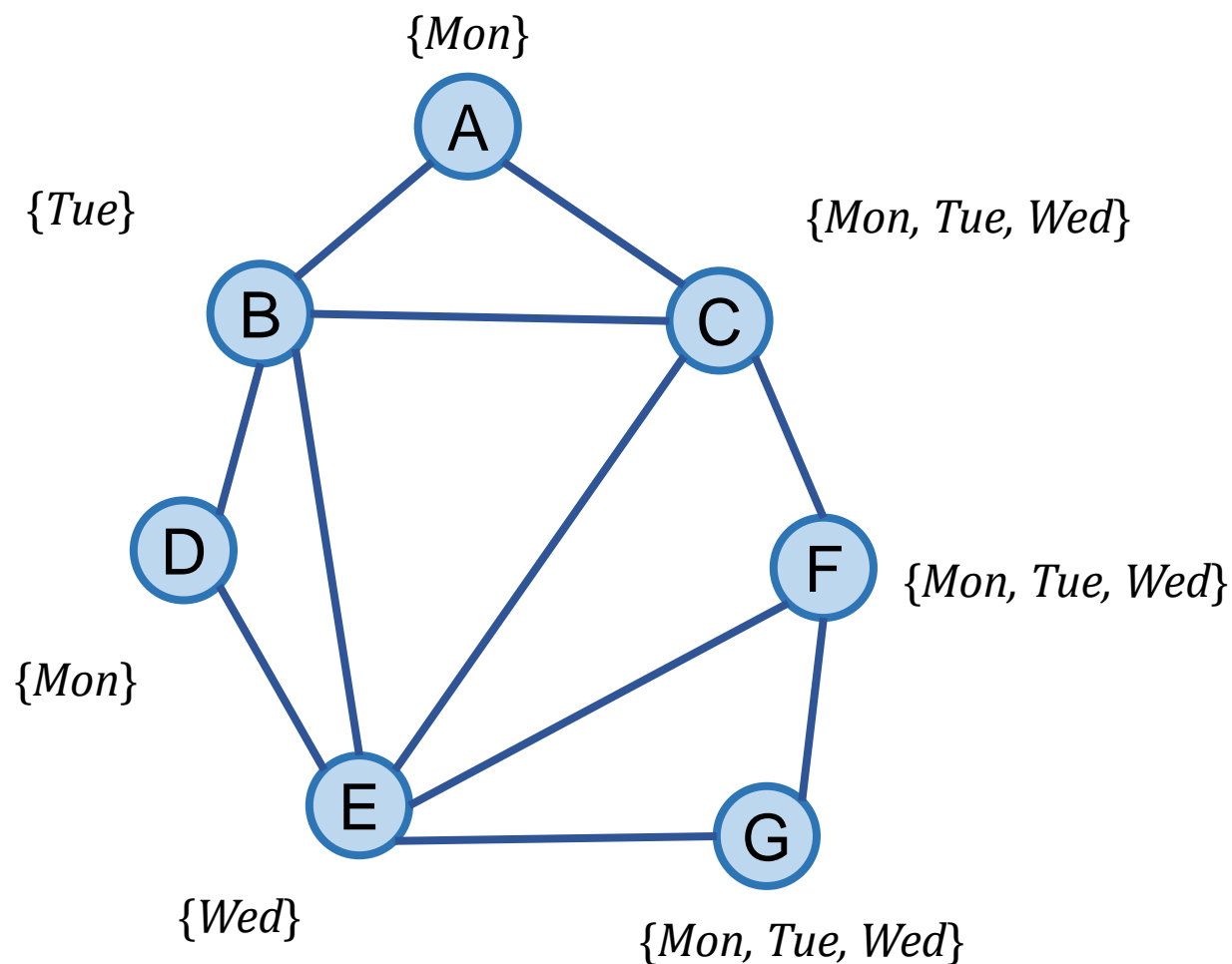
考试时间安排

- 对变量 C 赋值
 - 违背约束条件?
 - 有
 - 尝试定义域里另外一个值
 - 都尝试过了也没有合适的值，返回上一个变量重新赋值



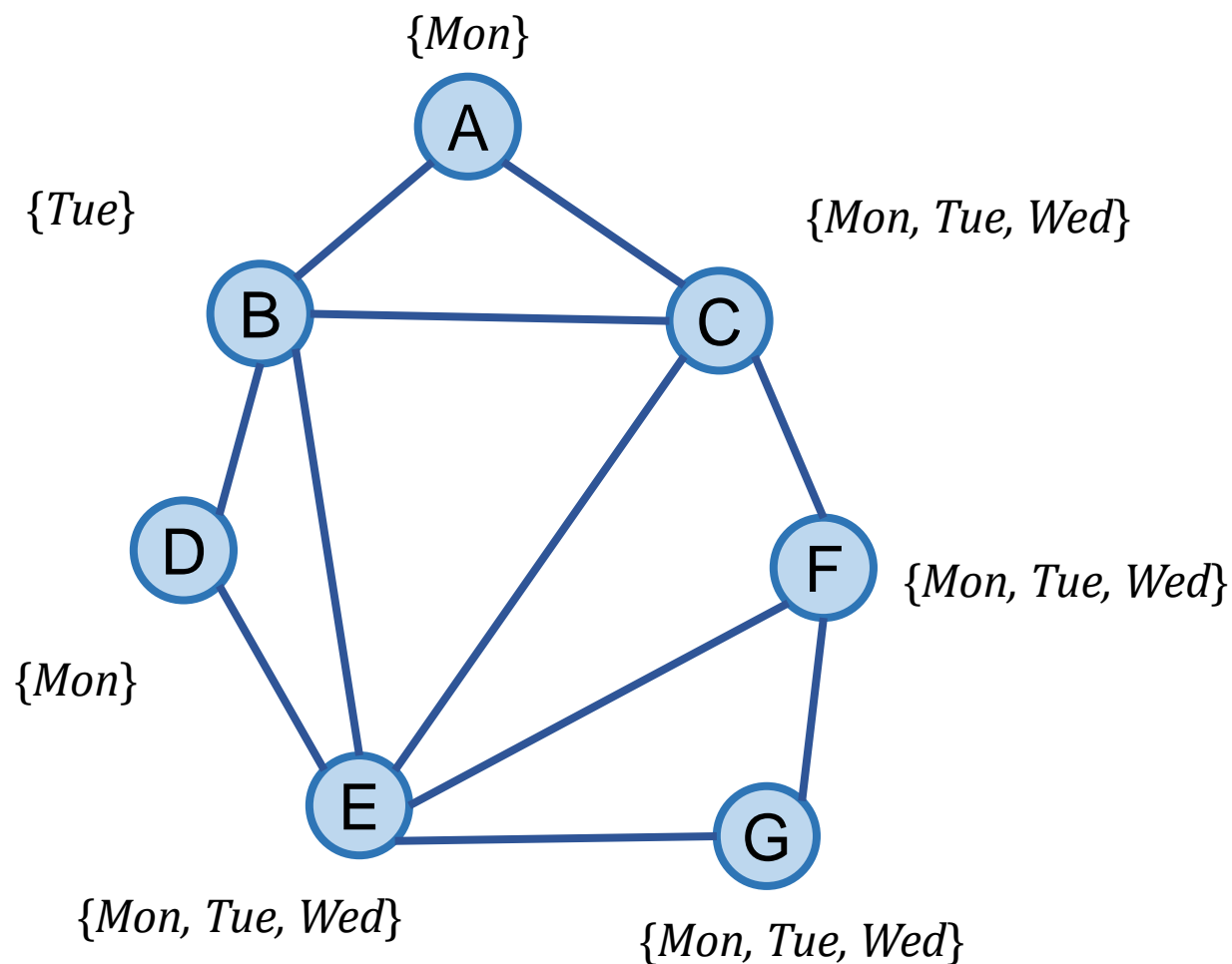
考试时间安排

- 对变量 E 赋值
 - 违背约束条件?
 - 有
 - 尝试定义域里另外一个值
 - 都尝试过了也没有合适的值，返回上一个变量重新赋值



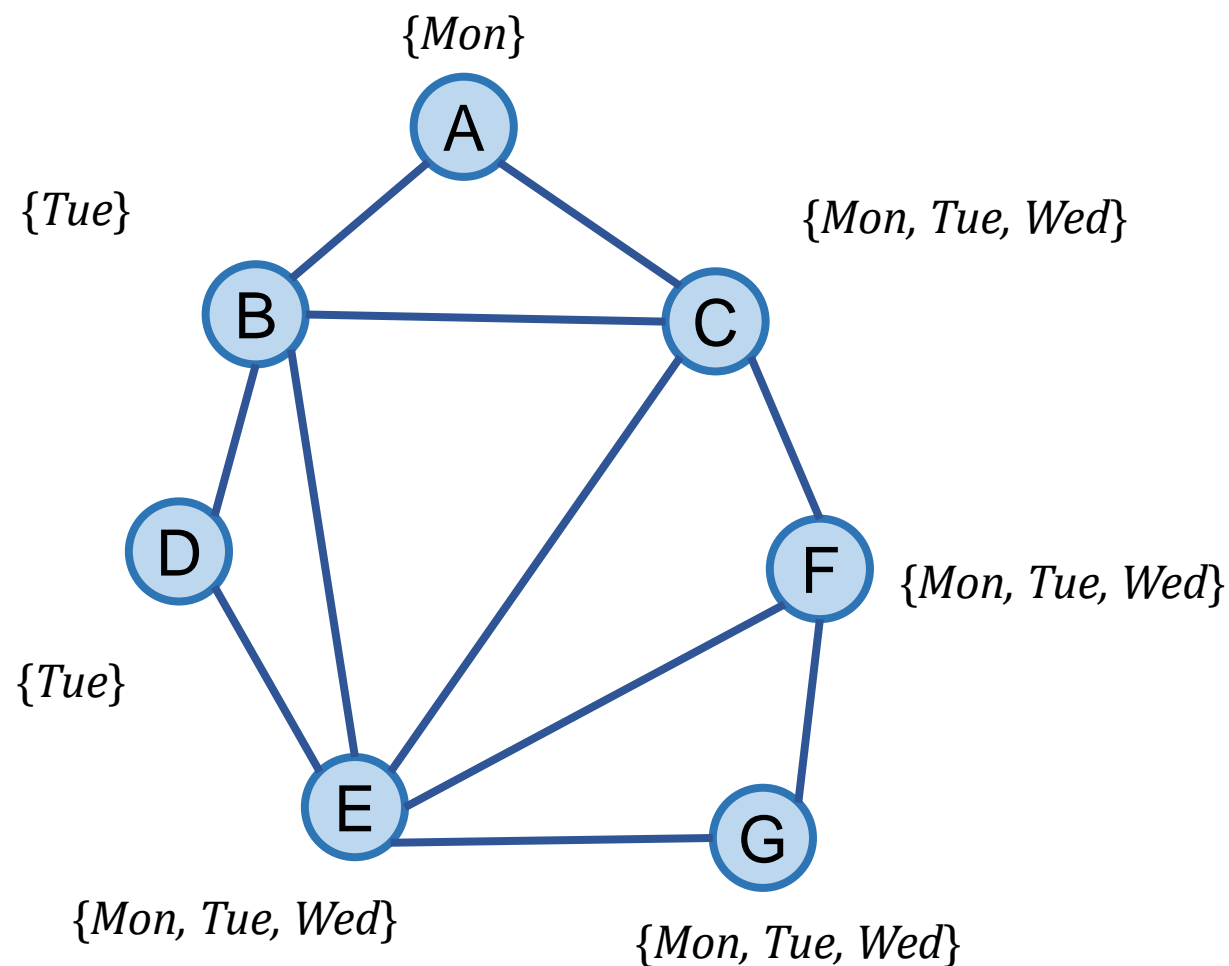
考试时间安排

- 对变量 D 赋值



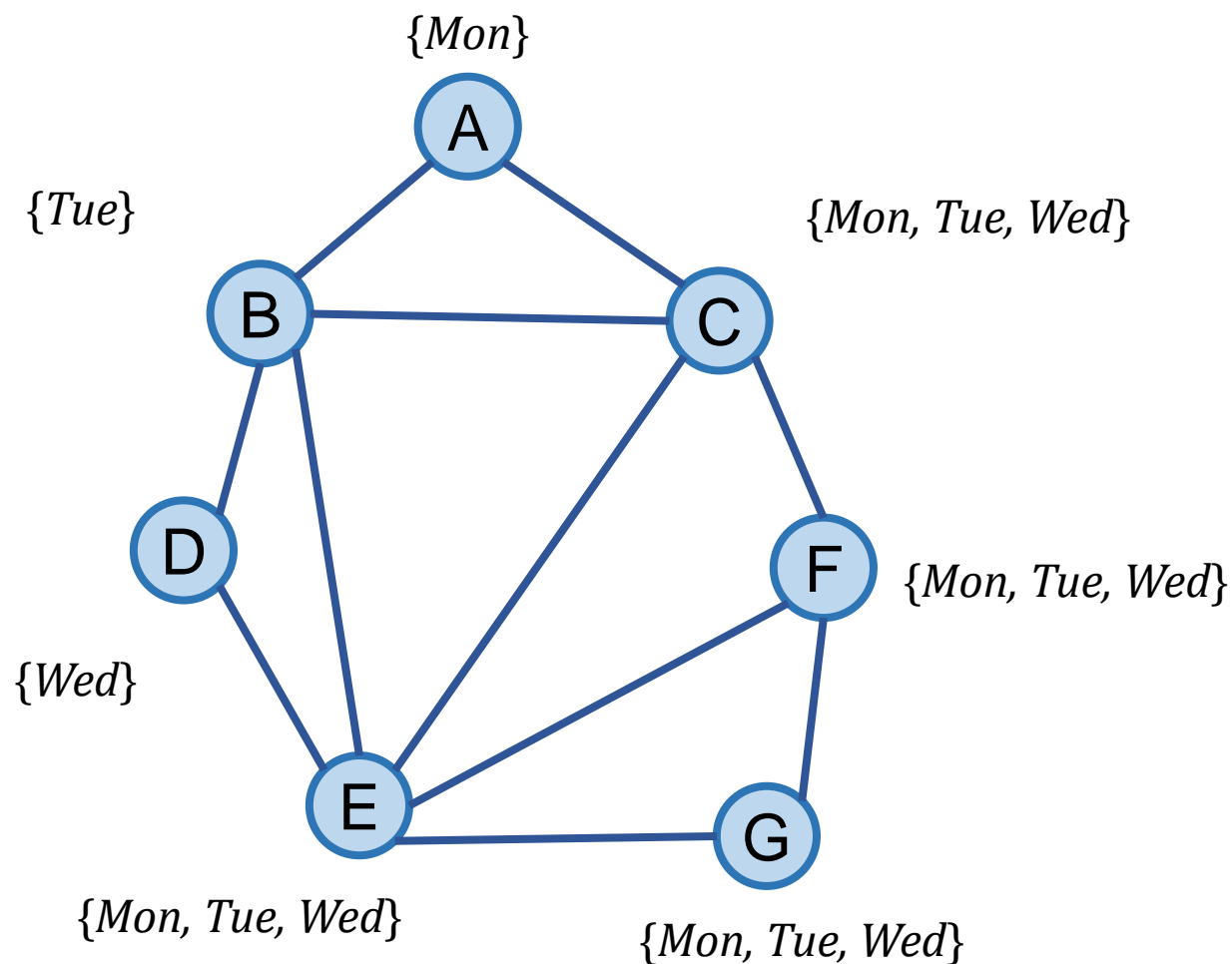
考试时间安排

- 对变量 D 赋值
 - 违背约束条件?
 - 有
 - 尝试定义域里另外一个值



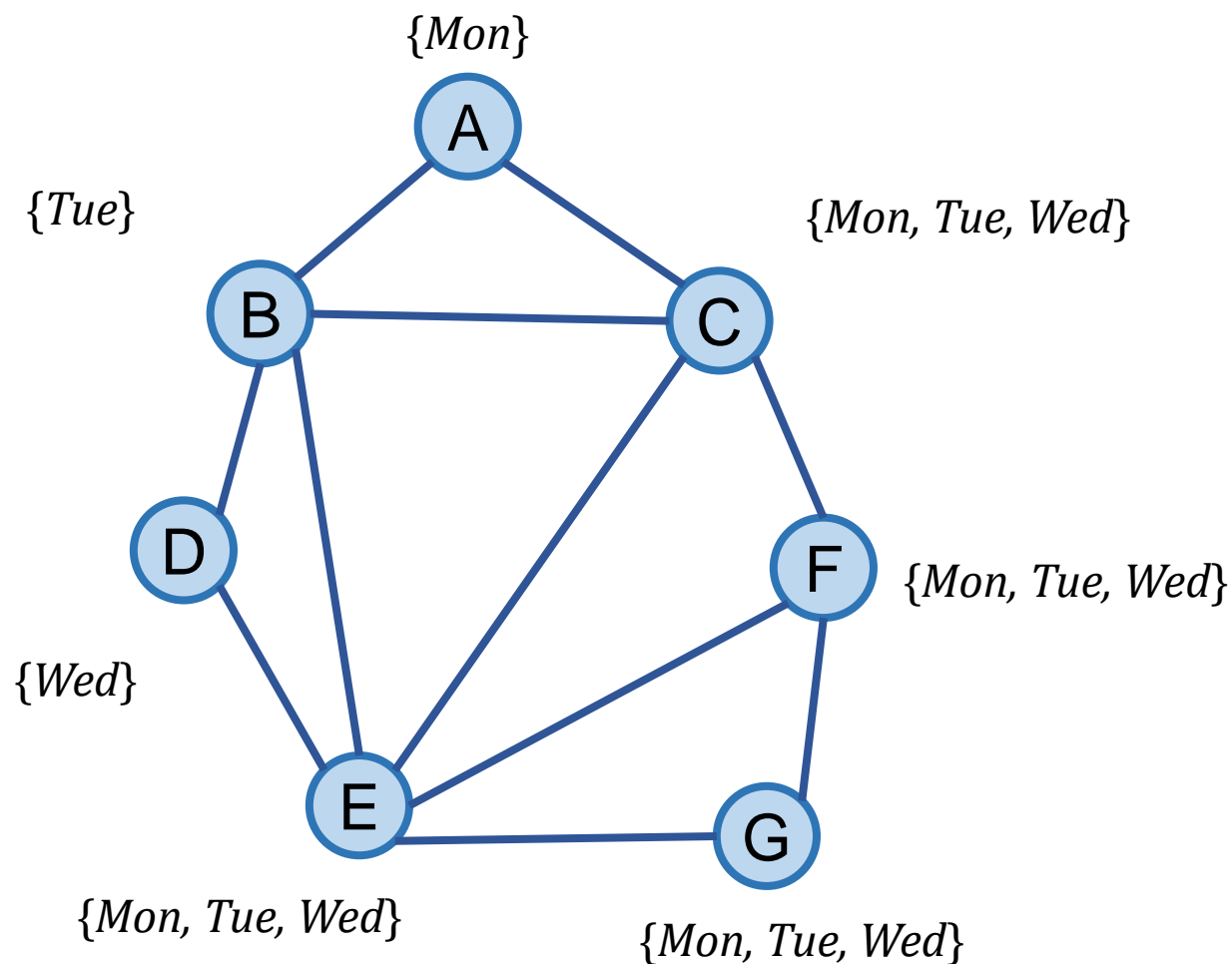
考试时间安排

- 对变量 D 赋值
 - 违背约束条件?
 - 没有
 - 赋值下一个变量



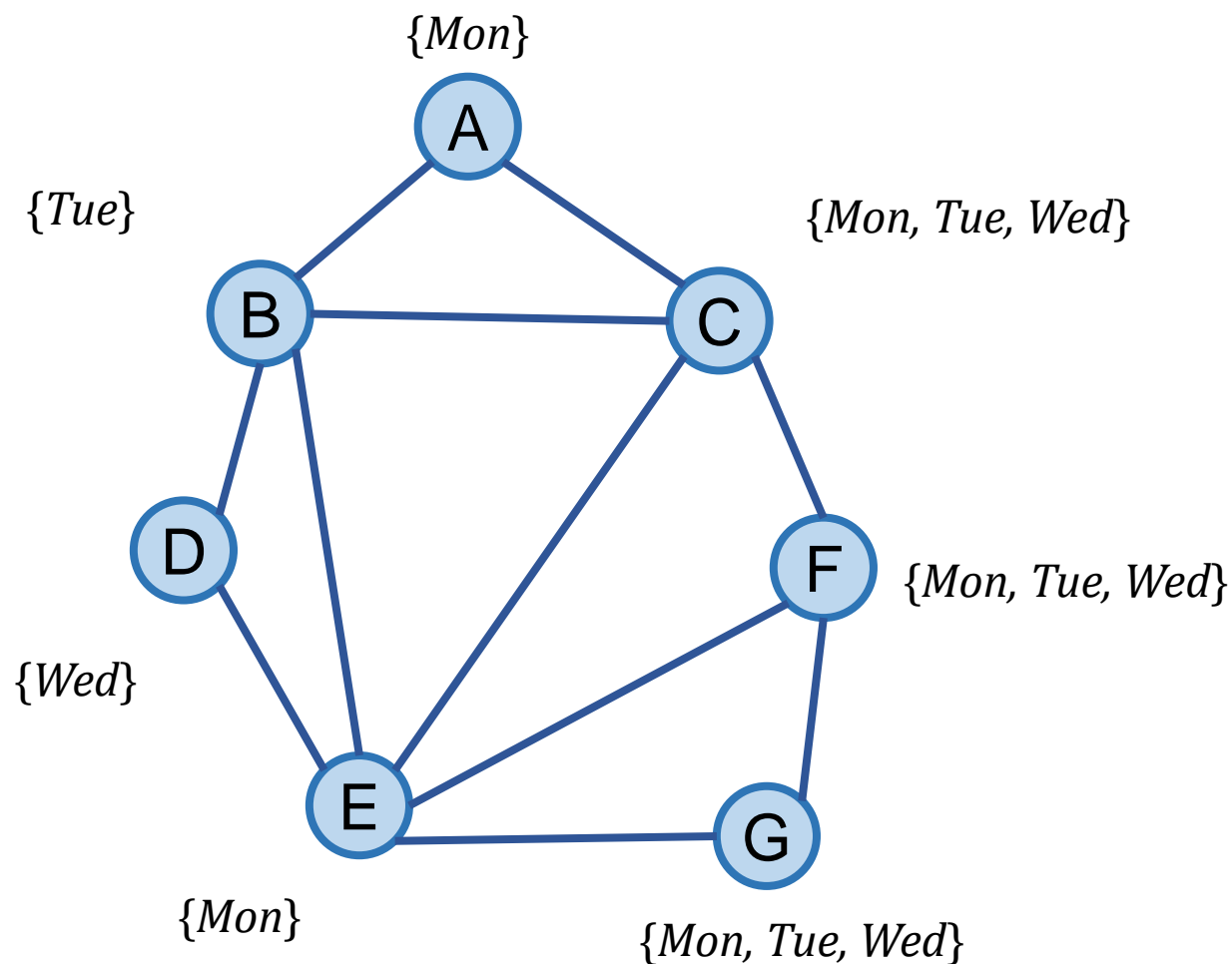
考试时间安排

- 对变量 E 赋值



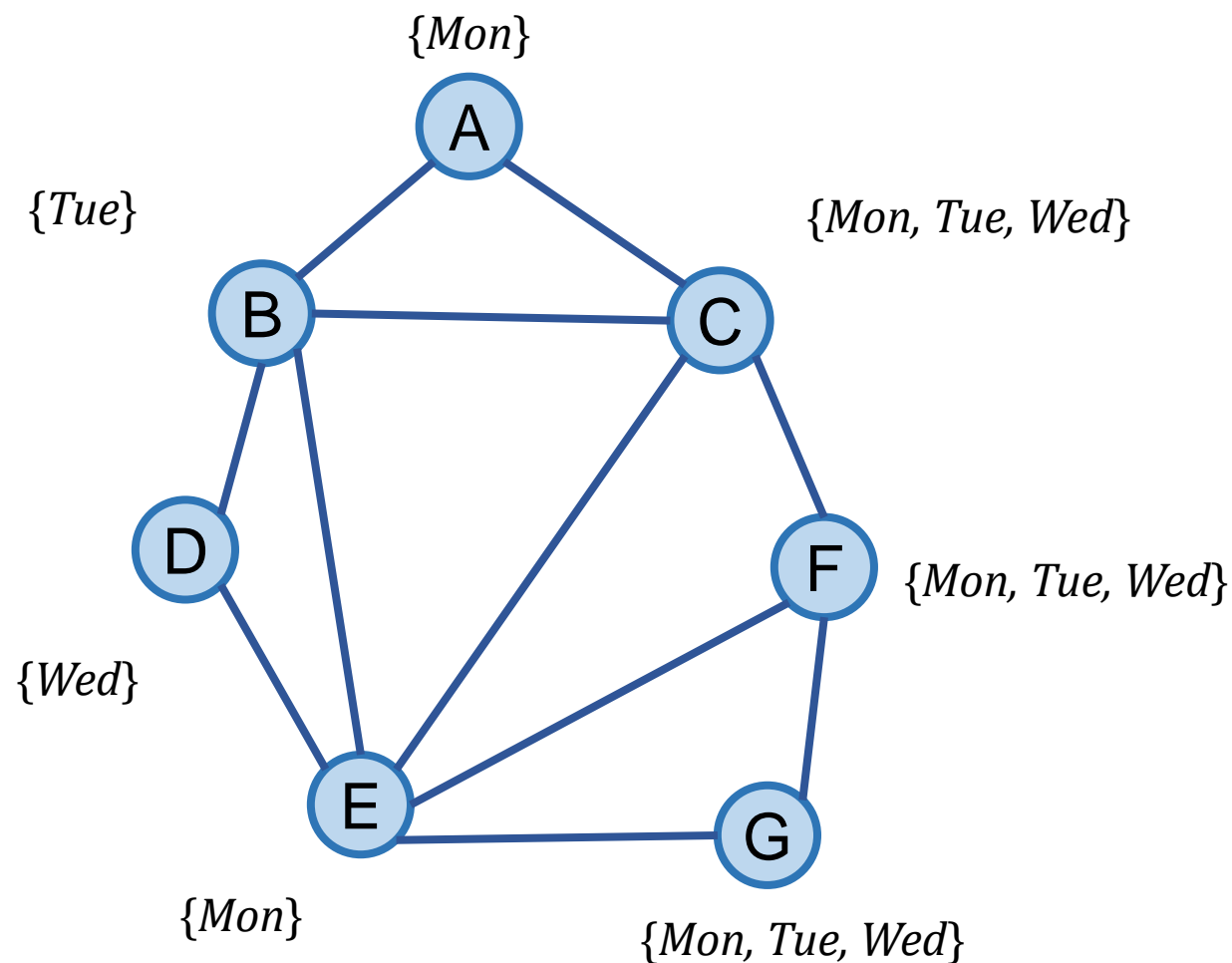
考试时间安排

- 对变量 E 赋值
 - 违背约束条件?
 - 没有
 - 赋值下一个变量



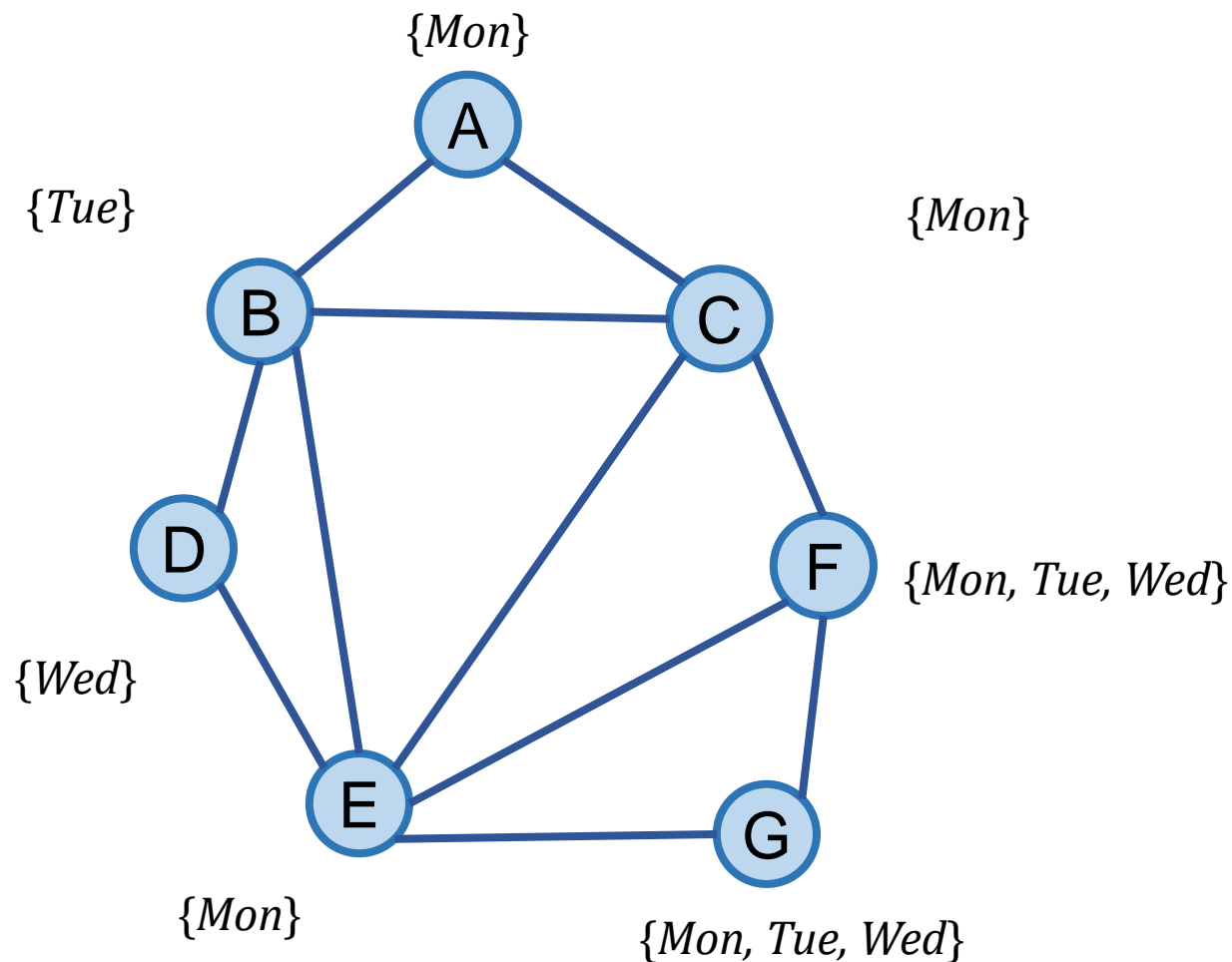
考试时间安排

- 对变量 C 赋值



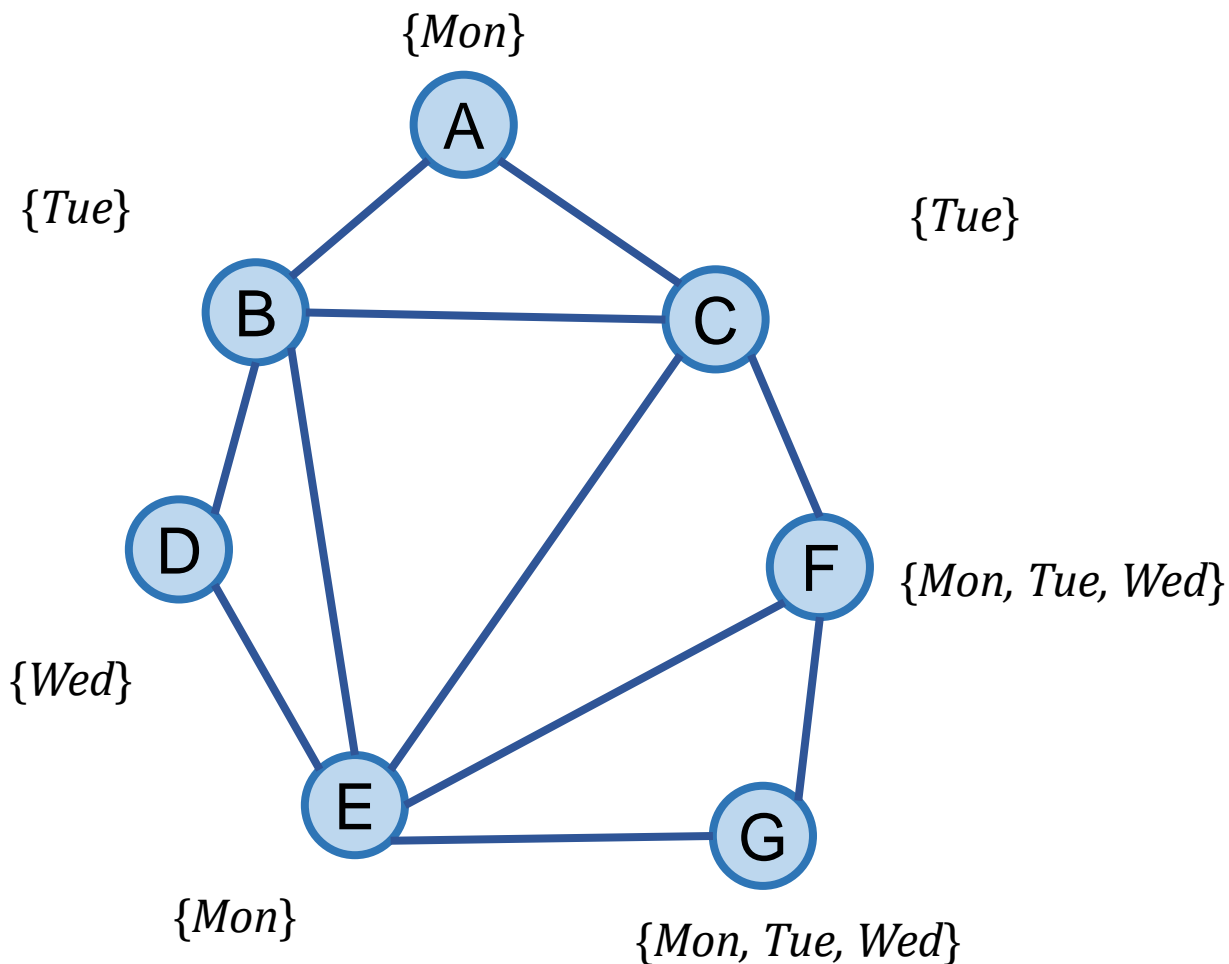
考试时间安排

- 对变量 C 赋值
 - 违背约束条件?
 - 有
 - 尝试定义域里另外一个值



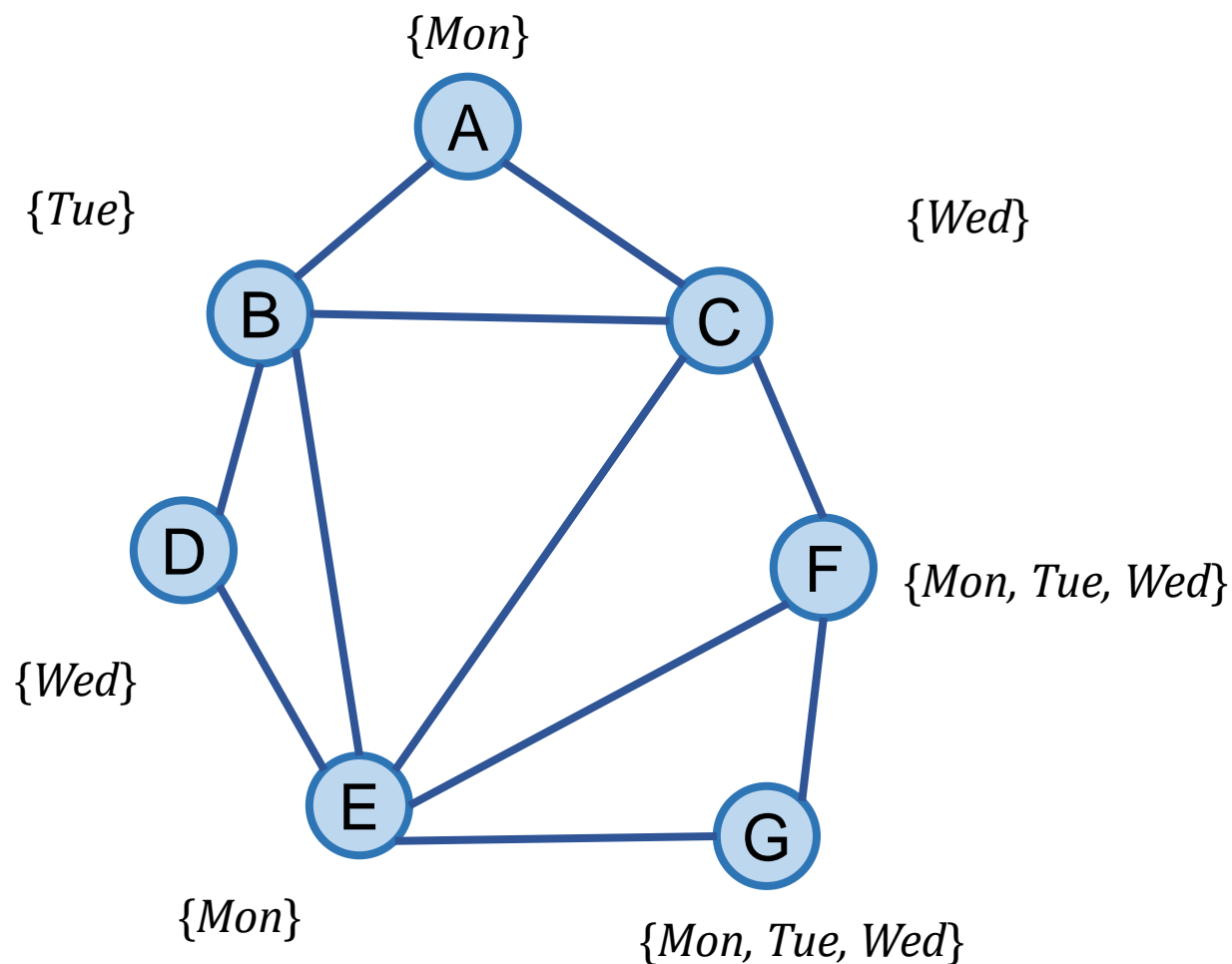
考试时间安排

- 对变量 C 赋值
 - 违背约束条件?
 - 有
 - 尝试定义域里另外一个值



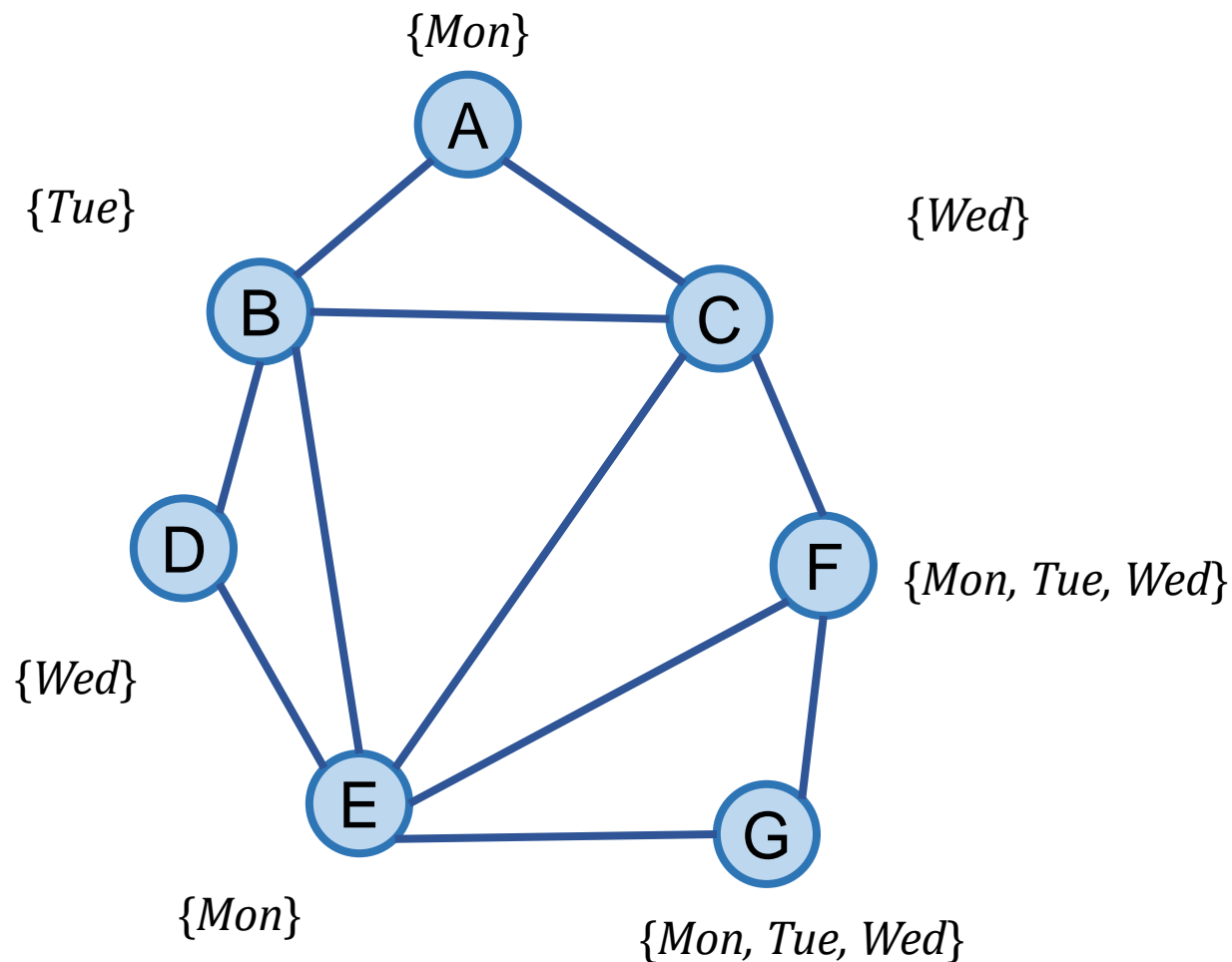
考试时间安排

- 对变量 C 赋值
 - 违背约束条件?
 - 没有
 - 赋值下一个变量



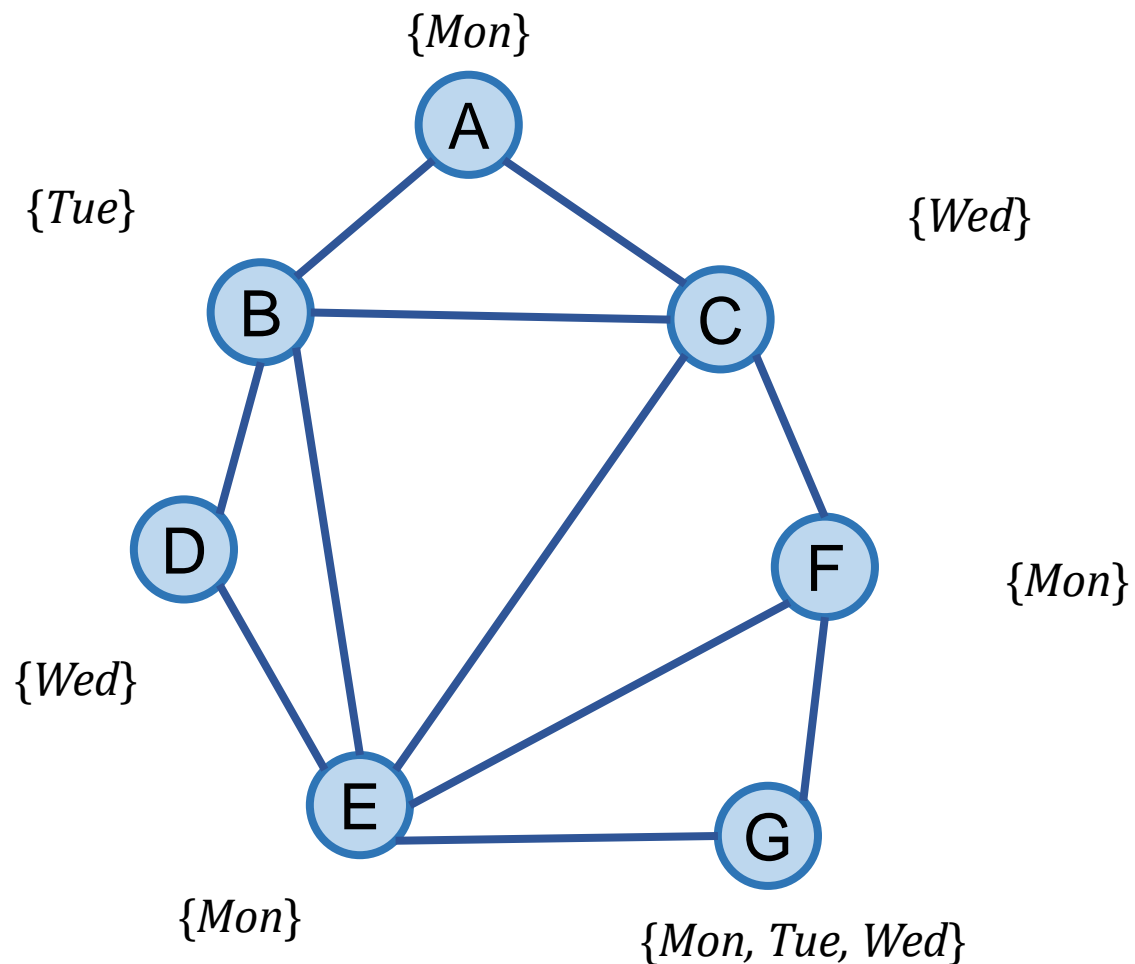
考试时间安排

- 对变量 F 赋值



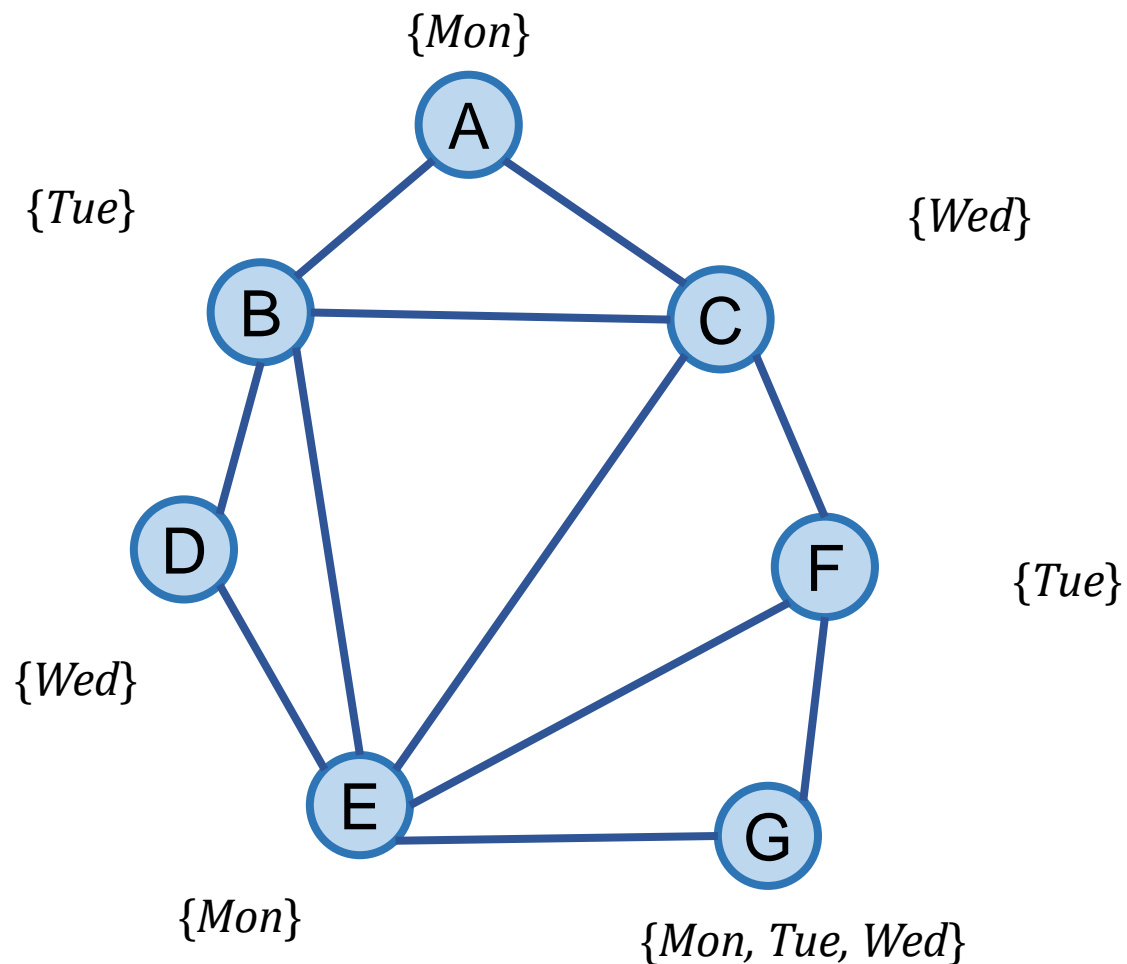
考试时间安排

- 对变量 F 赋值
 - 违背约束条件?
 - 有
 - 尝试定义域里另外一个值



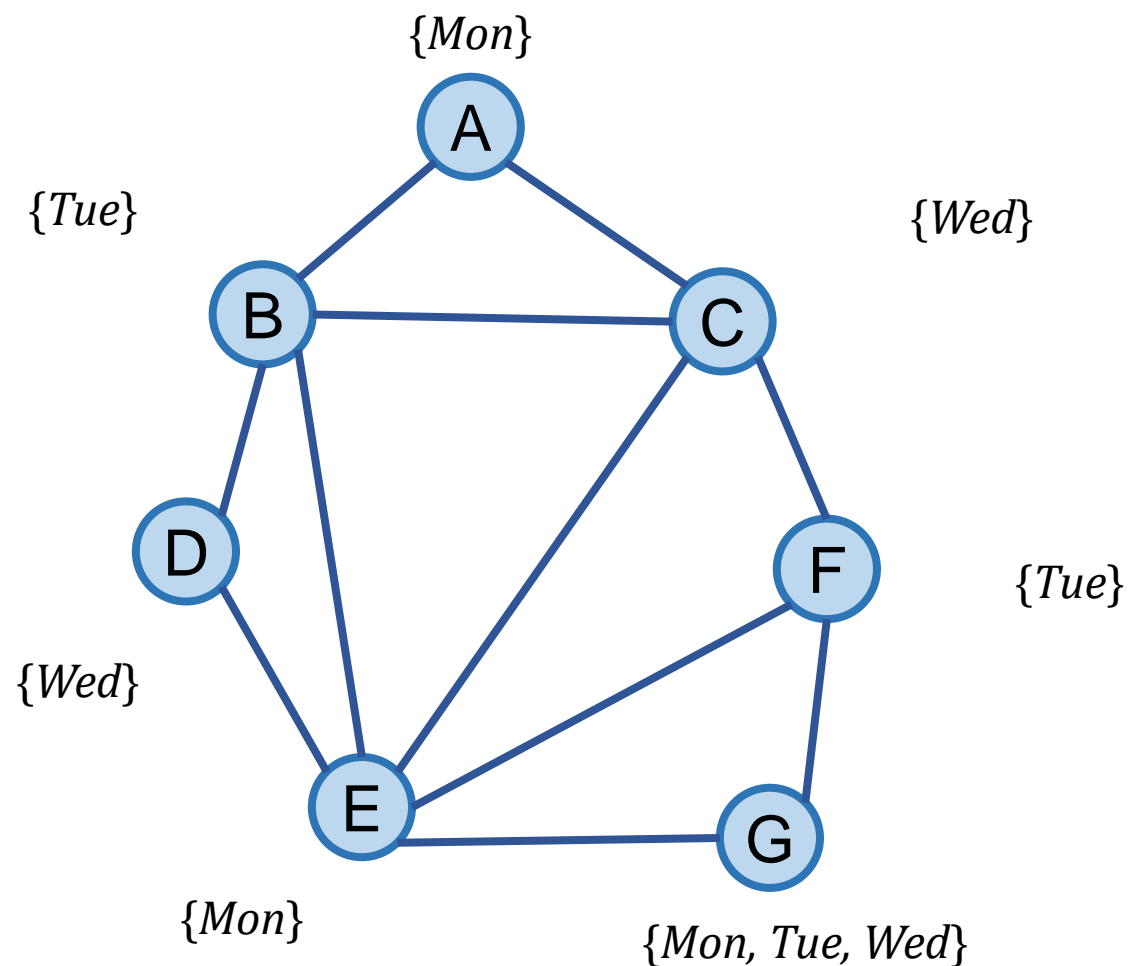
考试时间安排

- 对变量 F 赋值
 - 违背约束条件?
 - 没有
 - 赋值下一个变量



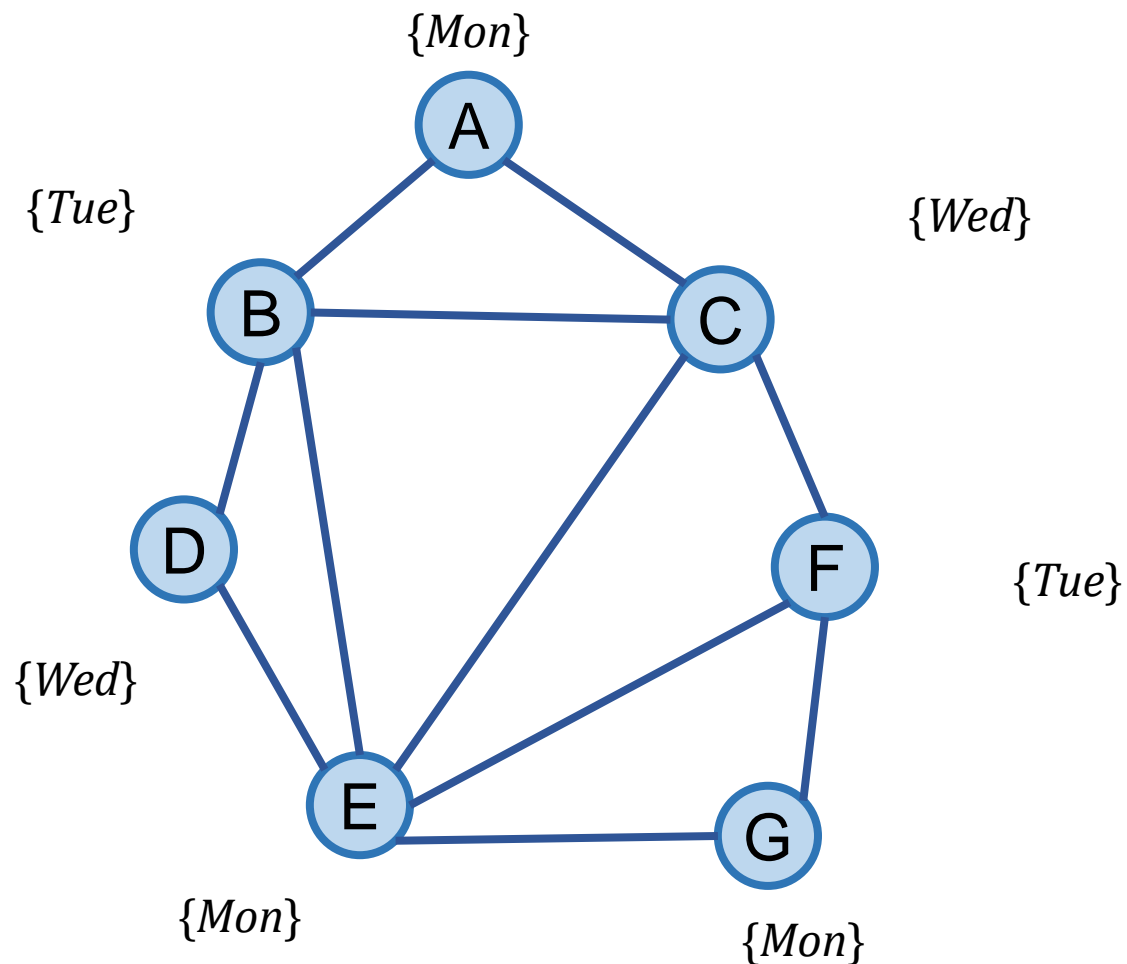
考试时间安排

- 对变量 G 赋值



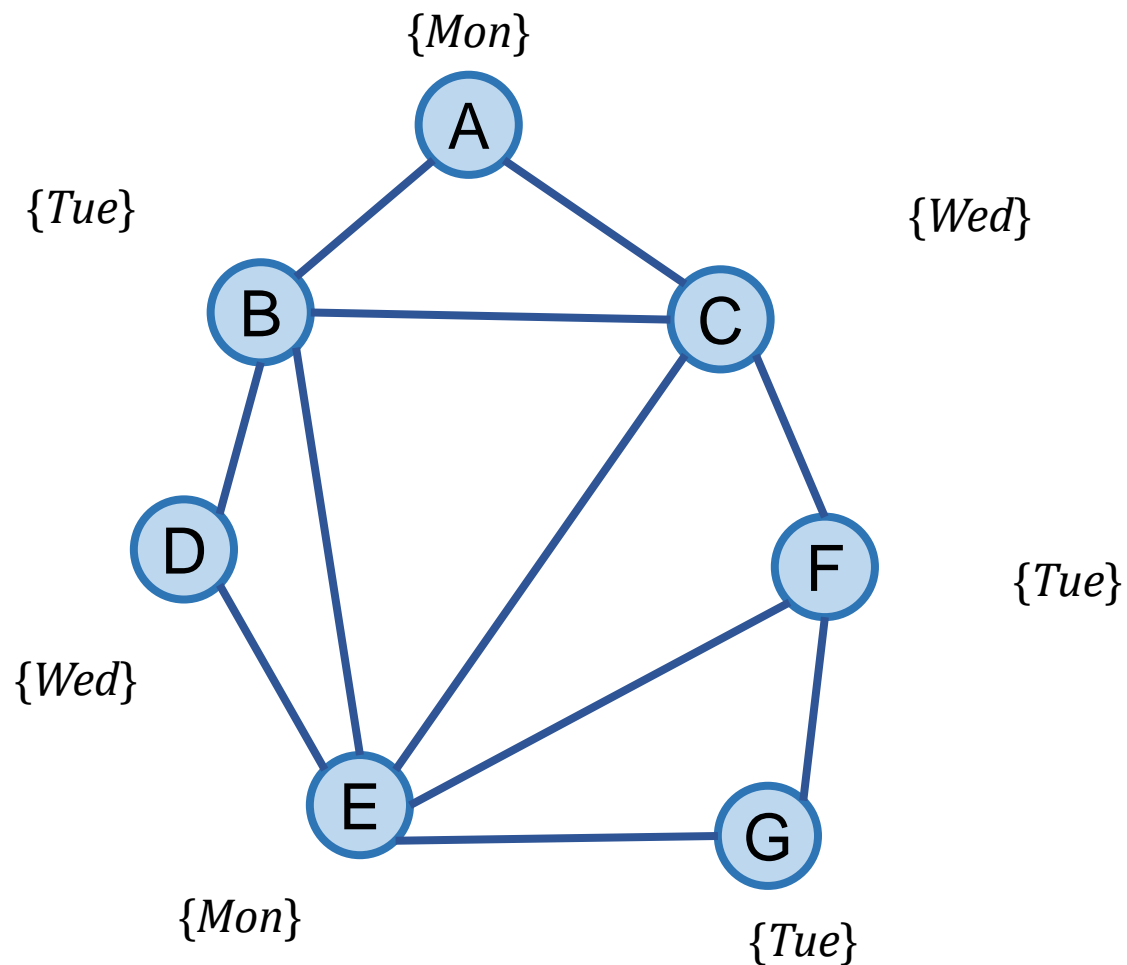
考试时间安排

- 对变量 G 赋值
 - 违背约束条件?
 - 有
 - 尝试定义域里另外一个值



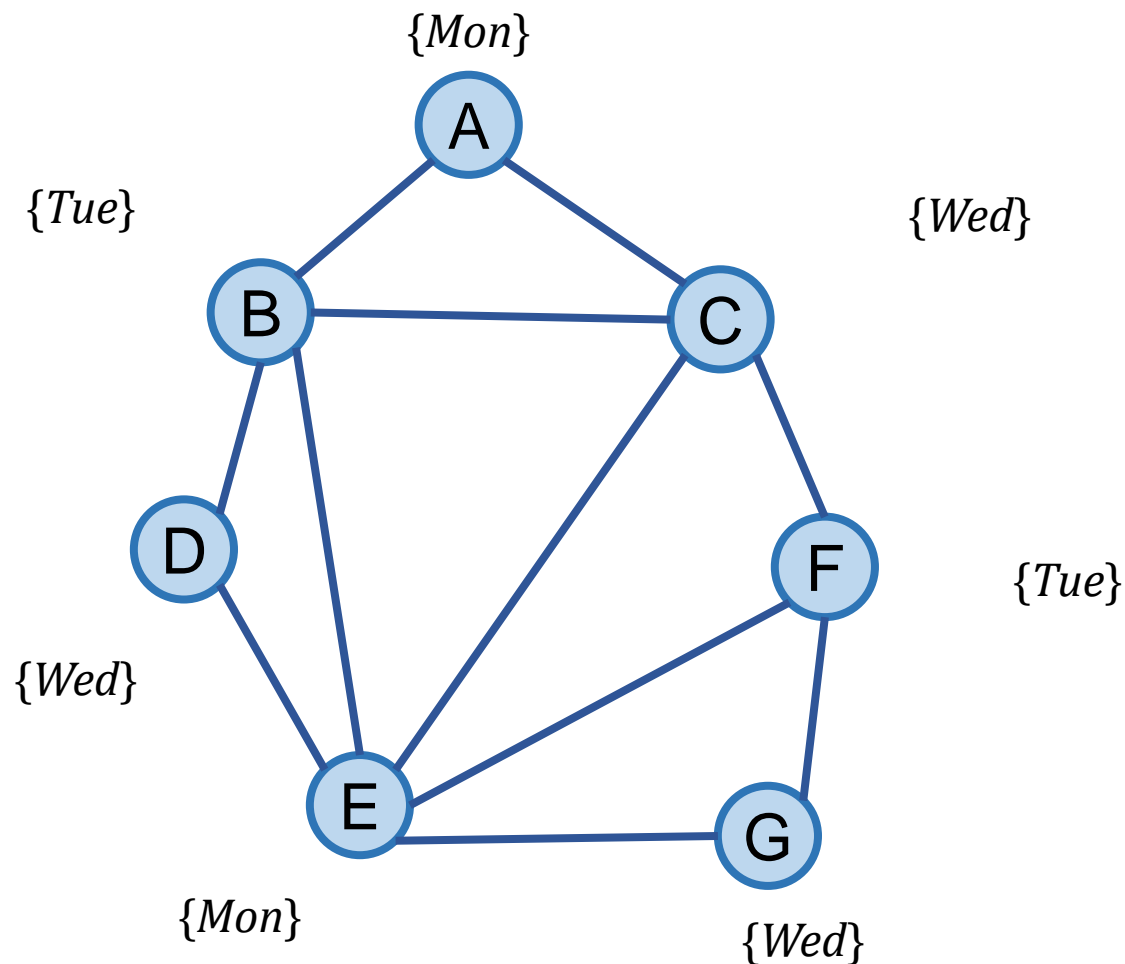
考试时间安排

- 对变量 G 赋值
 - 违背约束条件?
 - 有
 - 尝试定义域里另外一个值



考试时间安排

- 对变量 G 赋值
 - 违背约束条件?
 - 没有
 - 赋值下一个变量
- 所有变量已经赋值完毕
 - 所有约束都满足



回溯搜索 Backtracking search

function Backtrack(*assignment*, *problem*):

 if *assignment* is complete: return *assignment*

 所有变量都已经赋值完毕

var = Select-Unassigned-Var(*assignment*, *problem*)

 选择一个还没有被赋值的变量

 for *value* in Domain-Value(*var*, *assignment*, *problem*):

 if *value* is consistent with *assignment*:

 没有违背任何约束

 add {*var* = *value*} to *assignment*

result = Backtrack(*assignment*, *problem*)

 继续赋值下一个变量

 if *result* \neq failure: return *result*

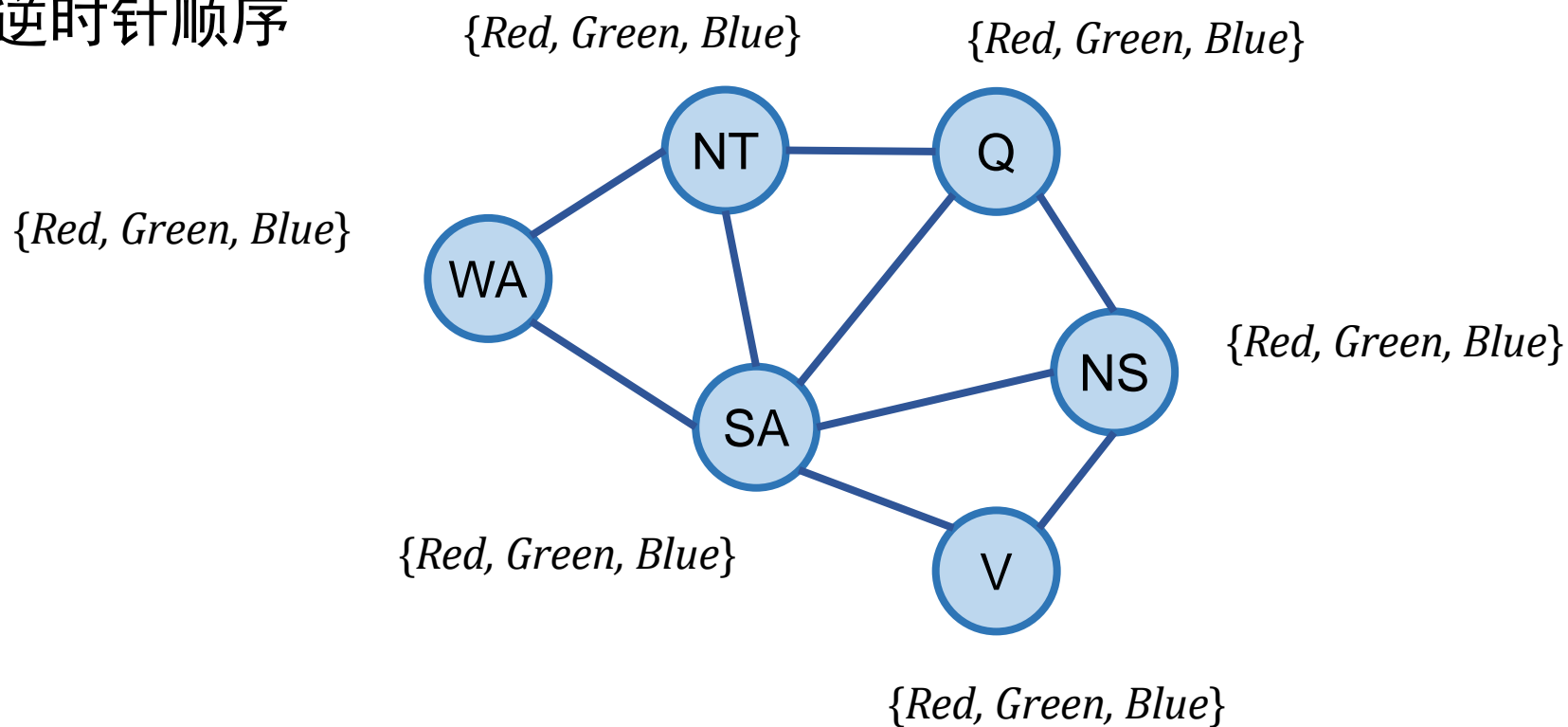
 remove {*var* = *value*} from *assignment*

 return failure

练习 #12

- 地图填色问题

- 逆时针顺序



有问题吗？

- 请随时举手提问。



BUSS 3620.人工智能导论

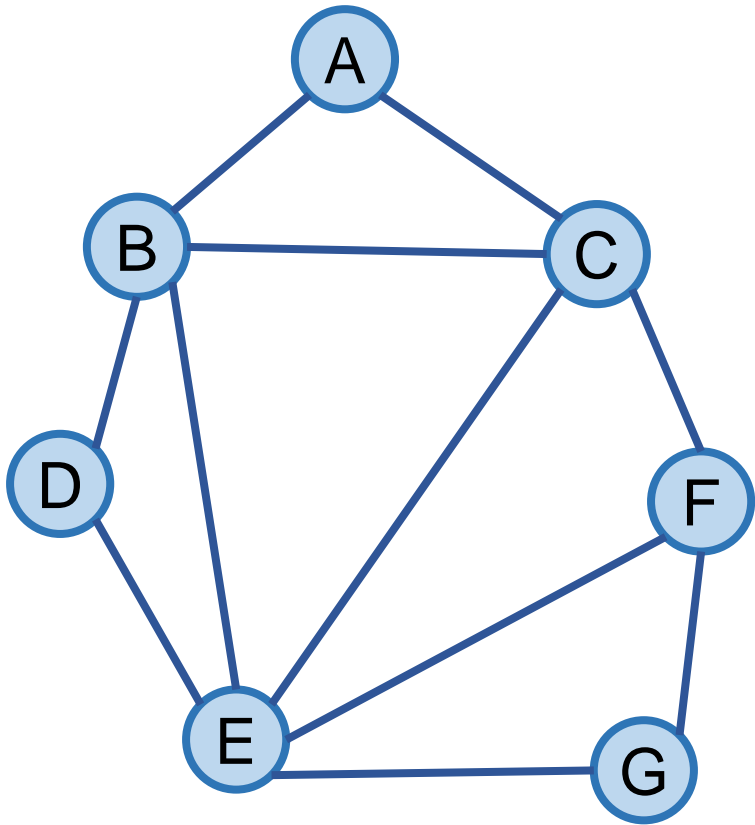
#5 代码示例：考试时间安排

刘佳璐

安泰经济与管理学院

上海交通大学

考试时间安排 Exam Schedule



变量 Variables

$\{A, B, C, D, E, F, G\}$

定义域 Domains

$\{Monday, Tuesday, Wednesday\}$

for each variable

约束 Constraint

$\{A \neq B, A \neq C, B \neq C, B \neq D, B \neq E, C \neq E, C \neq F, D \neq E, E \neq F, E \neq G, F \neq G\}$

解 Solution

$\{A = Monday, B = Tuesday, C = Wednesday, \dots\}$

考试时间安排 Exam Schedule

变量 Variables

$\{A, B, C, D, E, F, G\}$

定义域 Domains

$\{Monday, Tuesday, Wednesday\}$

for each variable

约束 Constraint

$\{A \neq B, A \neq C, B \neq C, B \neq D, B \neq E, C \neq E, C \neq F, D \neq E, E \neq F, E \neq G, F \neq G\}$

解 Solution

$\{A = Monday, B = Tuesday, C = Wednesday, \dots\}$

```
1 VARIABLES = ["A", "B", "C", "D", "E", "F", "G"]
2 DOMAINS = ["Monday", "Tuesday", "Wednesday"]
3 CONSTRAINTS = [
4     ("A", "B"),
5     ("A", "C"),
6     ("B", "C"),
7     ("B", "D"),
8     ("B", "E"),
9     ("C", "E"),
10    ("C", "F"),
11    ("D", "E"),
12    ("E", "F"),
13    ("E", "G"),
14    ("F", "G")
15 ]
```

回溯搜索 Backtracking search

function Backtrack(*assignment*, *problem*):

if *assignment* is complement: return *assignment* 所有变量都已经赋值完毕

var = Select-Unassigned-Var(*assignment*, *problem*) 选择一个还没有被赋值的变量

for *value* in Domain-Value(*var*, *assignment*, *problem*):

 if *value* is consistent with *assignment*: 没有违背任何约束

 add {*var* = *value*} to *assignment*

result = Backtrack(*assignment*, *problem*) 继续赋值下一个变量

 if *result* \neq failure: return *result*

 remove {*var* = *value*} from *assignment*

return failure

回溯搜索 Backtracking search

function Backtrack(*assignment*, *problem*):

if *assignment* is complement: return *assignment* 所有变量都已经赋值完毕

var = Select-Unassigned-Var(*assignment*, *problem*) 选择一个还没有被赋值的变量

for *value* in Domain-Value(*var*, *assignment*, *problem*):

new_assignment = add {*var* = *value*} to *assignment* 继续赋值下一个变量

 if *value* is consistent with *new_assignment*: 没有违背任何约束

result = Backtrack(*new_assignment*, *problem*)

 if *result* \neq failure: return *result*

 如果failure, 直接循环下一个value

return *failure*

回溯搜索 Backtracking search

function Backtrack(*assignment*, *problem*):

 if *assignment* is complement: return *assignment*

var = Select-Unassigned-Var(*assignment*, *problem*)

 for *value* in Domain-Value(*var*, *assignment*, *problem*):

new_assignment = add {*var* = *value*} to *assignment*

 if *value* is consistent with *new_assignment*:

result = Backtrack(*new_assignment*, *problem*)

 if *result* \neq failure: return *result*

 return failure

```
def backtrack(assignment):  
    """Runs backtracking search to find an assignment."""  
  
    # Check if assignment is complete  
    if len(assignment) == len(VARIABLES):  
        return assignment  
  
    # Try a new variable  
    var = select_unassigned_variable(assignment)  
    for value in DOMAINS:  
        new_assignment = assignment.copy()  
        new_assignment[var] = value  
        if consistent(new_assignment):  
            result = backtrack(new_assignment)  
            if result is not None:  
                return result  
  
    return None
```

回溯搜索 Backtracking search

function Backtrack(*assignment*, *problem*):

 if *assignment* is complement: return *assignment*

var = Select-Unassigned-Var(*assignment*, *problem*)

 for *value* in Domain-Value(*var*, *assignment*, *problem*):

new_assignment = add {*var* = *value*} to *assignment*

 if *value* is consistent with *new_assignment*:

result = Backtrack(*new_assignment*, *problem*)

 if *result* \neq failure: return *result*

 return failure

```
def backtrack(assignment):  
    """Runs backtracking search to find an assignment."""  
  
    # Check if assignment is complete  
    if len(assignment) == len(VARIABLES):  
        return assignment  
  
    # Try a new variable  
    var = select_unassigned_variable(assignment)  
    for value in DOMAINS:  
        new_assignment = assignment.copy()  
        new_assignment[var] = value  
        if consistent(new_assignment):  
            result = backtrack(new_assignment)  
            if result is not None:  
                return result  
  
    return None
```


练习 #13

- 构造函数 `select_unassigned_var(assignment)`
 - 功能：选取一个还没有被赋值的变量
 - 如果所有变量都被赋值完毕，返回None
 - 输入：assignment 字典，其中key是变量，value是变量的赋值
 - 输出：变量的名字

练习 #14

- 构造函数 `consistent(assignment)`
 - 功能：判断已经被赋值的变量是否满足他们的二元约束
 - 仅考虑二元约束中两个变量均被赋值完毕的二元约束
 - 输入：assignment 字典，其中key是变量，value是变量的赋值
 - 输出：True or False

Python 包

- constraint

```
1 from constraint import *
2
3 problem = Problem()
4
5 # Add variables
6 problem.addVariables(
7     ["A", "B", "C", "D", "E", "F", "G"],
8     ["Monday", "Tuesday", "Wednesday"]
9 )
10
11 # Add constraints
12 CONSTRAINTS = [
13     ("A", "B"),
14     ("A", "C"),
15     ("B", "C"),
16     ("B", "D"),
17     ("B", "E"),
18     ("C", "E"),
19     ("C", "F"),
20     ("D", "E"),
21     ("E", "F"),
22     ("E", "G"),
23     ("F", "G")
24 ]
25 for x, y in CONSTRAINTS:
26     problem.addConstraint(lambda x, y: x != y, (x, y))
27
28 # Solve problem
29 for solution in problem.getSolutions():
30     print(solution)
```

有问题吗？

- 请随时举手提问。



BUSS 3620.人工智能导论

#4.2: 推断

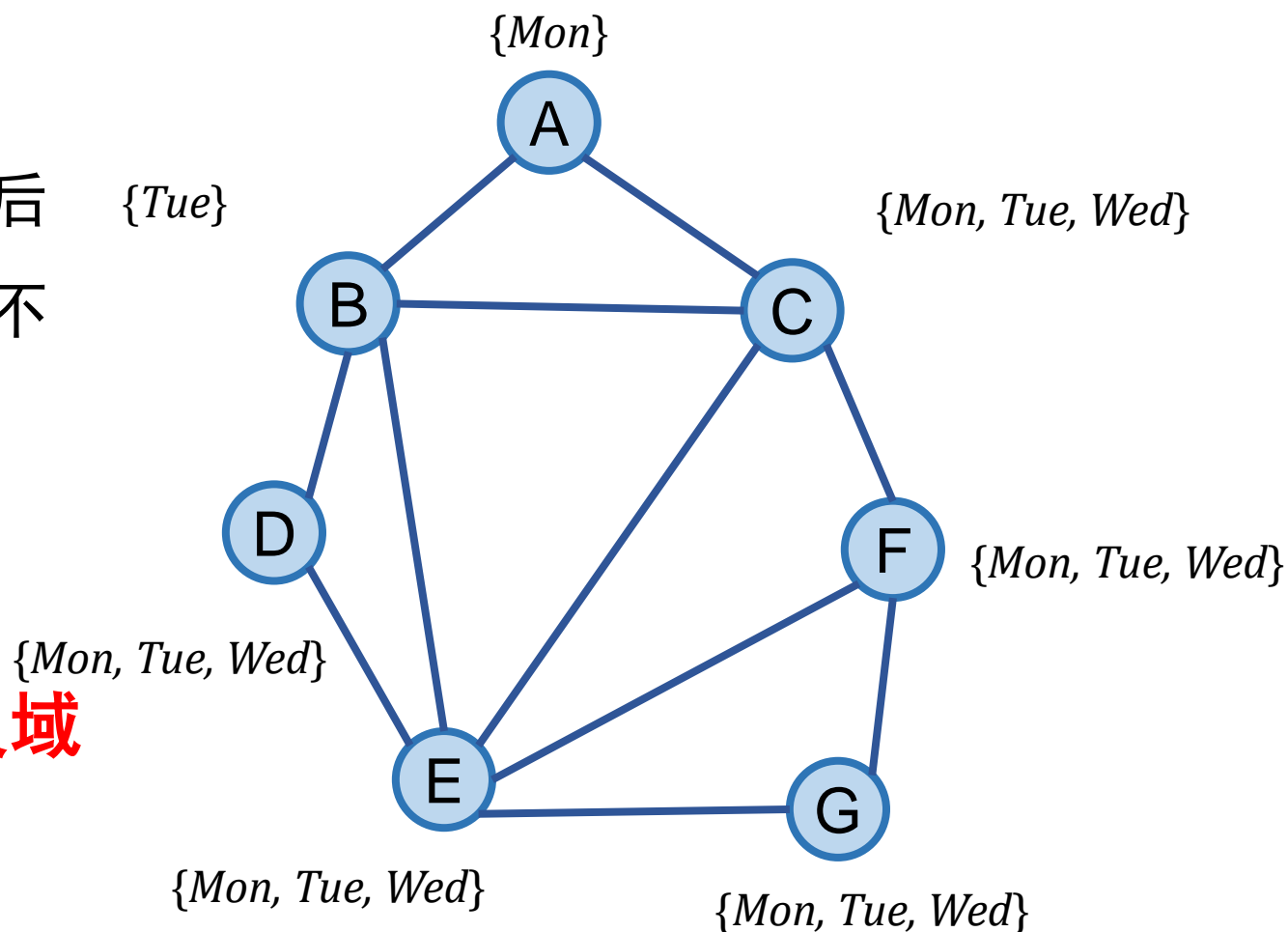
刘佳璐

安泰经济与管理学院

上海交通大学

考试时间安排

- 对变量 D 赋值
 - Mon 满足约束，但是之后其他节点的赋值使 Mon 不是一个合适的赋值
- 如何避免这种情况？
- 约束图中的变量的定义域不满足弧一致性



考试时间安排

- 对变量 B 赋值

- 违背约束条件?

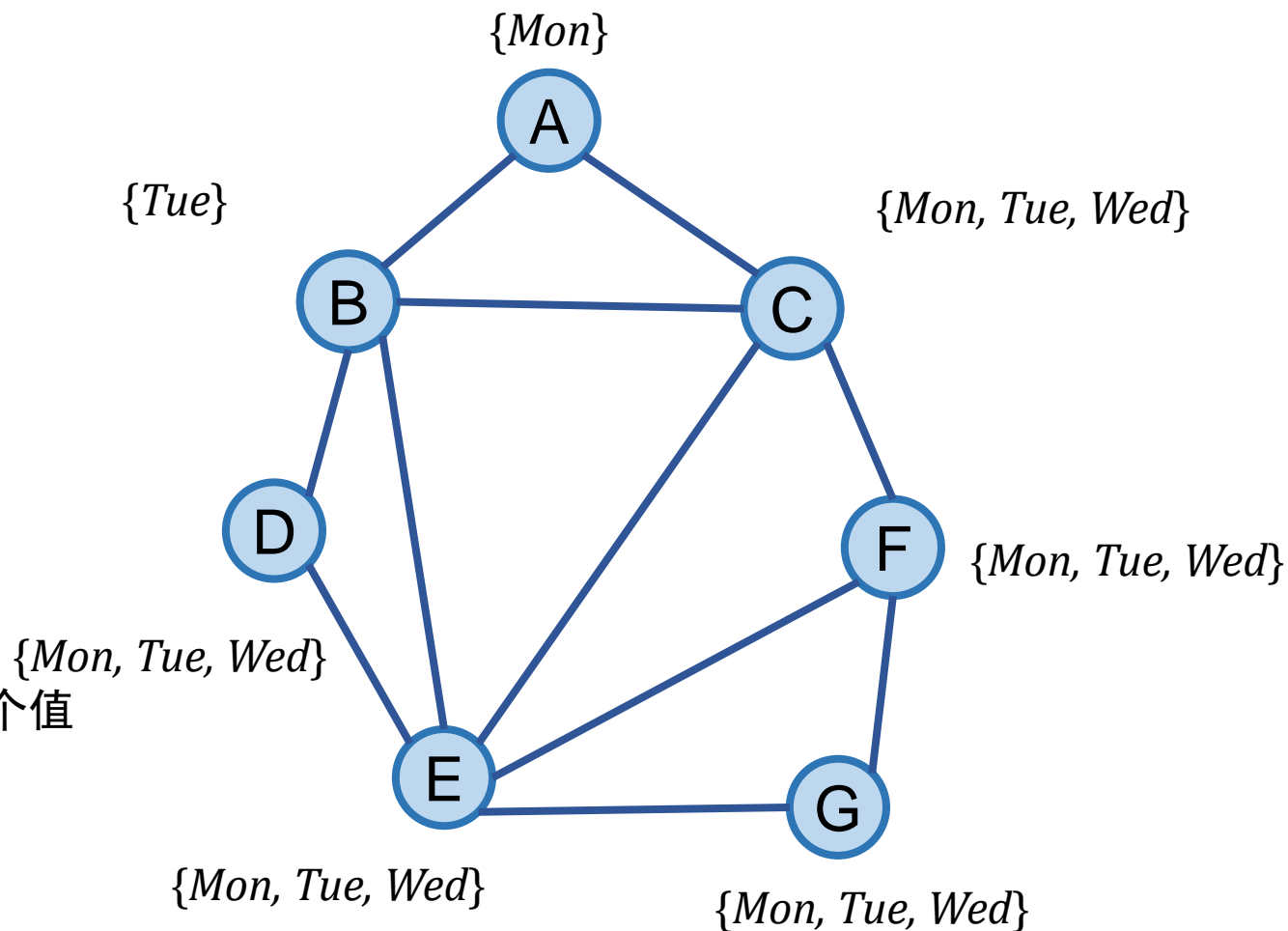
- 没有

- 执行算法使弧一致

- 赋值下一个变量 D

- 有

- 尝试定义域里另外一个值



考试时间安排

- 对变量 B 赋值

- 违背约束条件?

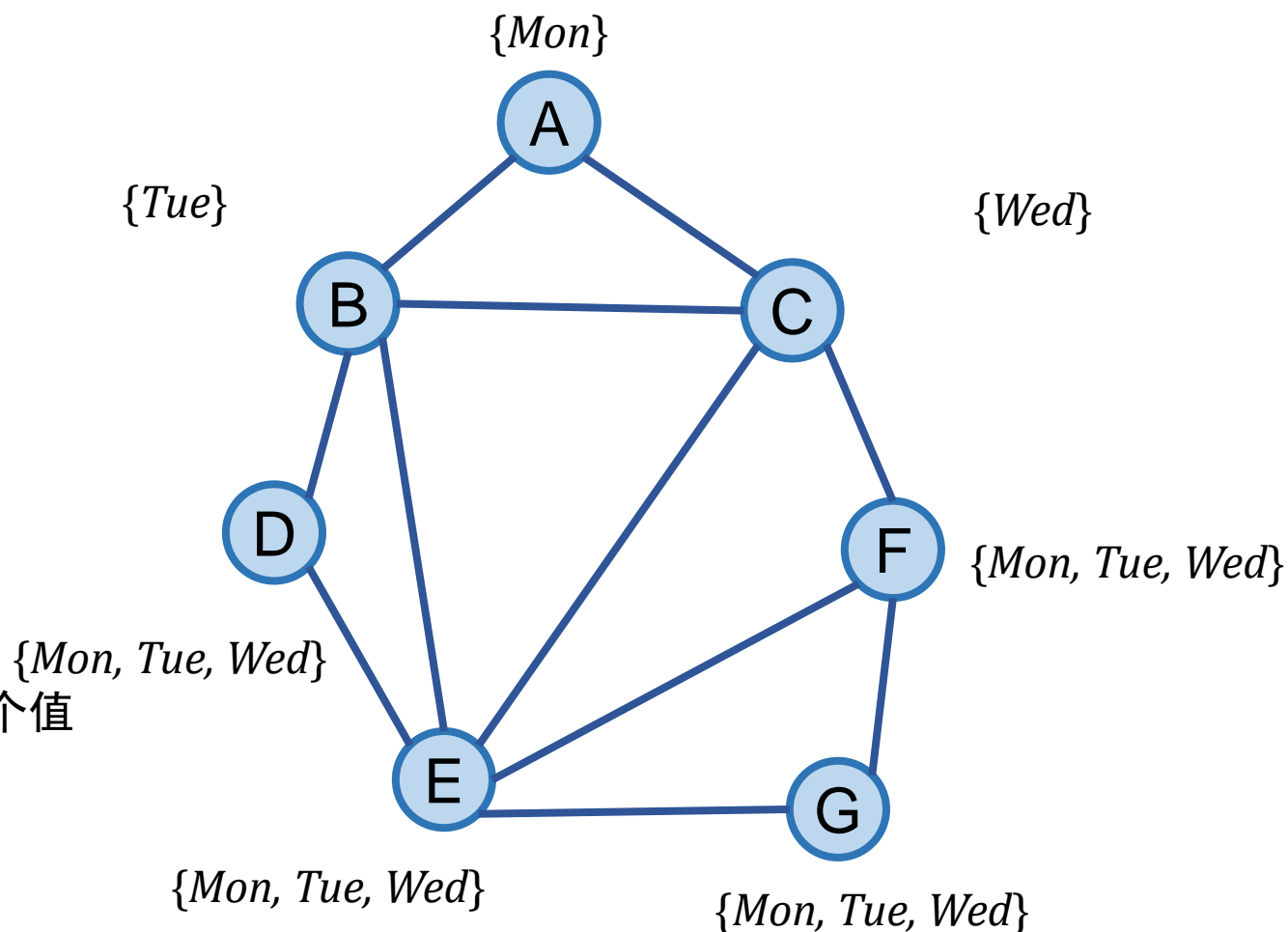
- 没有

- 执行算法使弧一致

- 赋值下一个变量 D

- 有

- 尝试定义域里另外一个值



考试时间安排

- 对变量 B 赋值

- 违背约束条件?

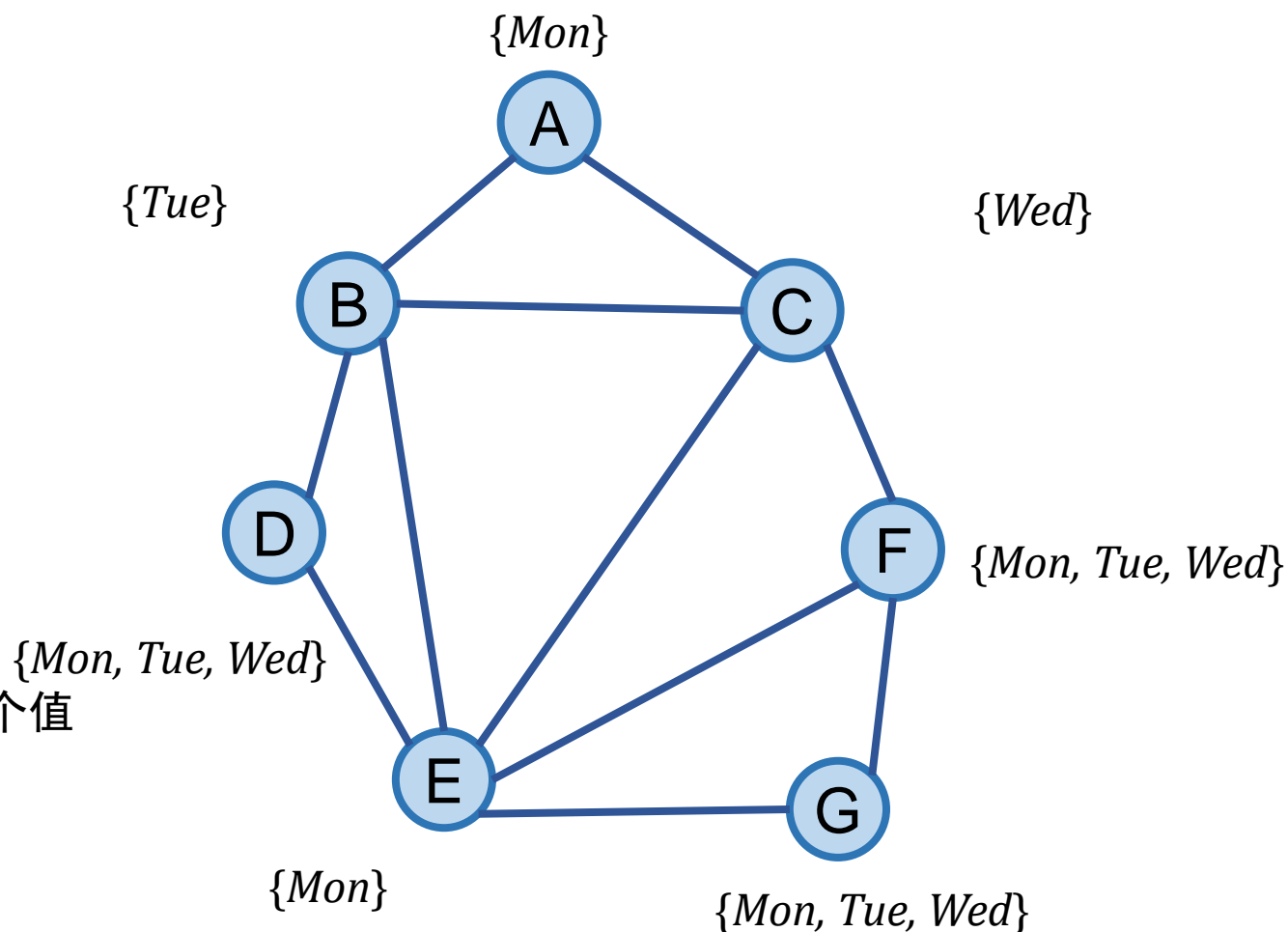
- 没有

- 执行算法使弧一致

- 赋值下一个变量 D

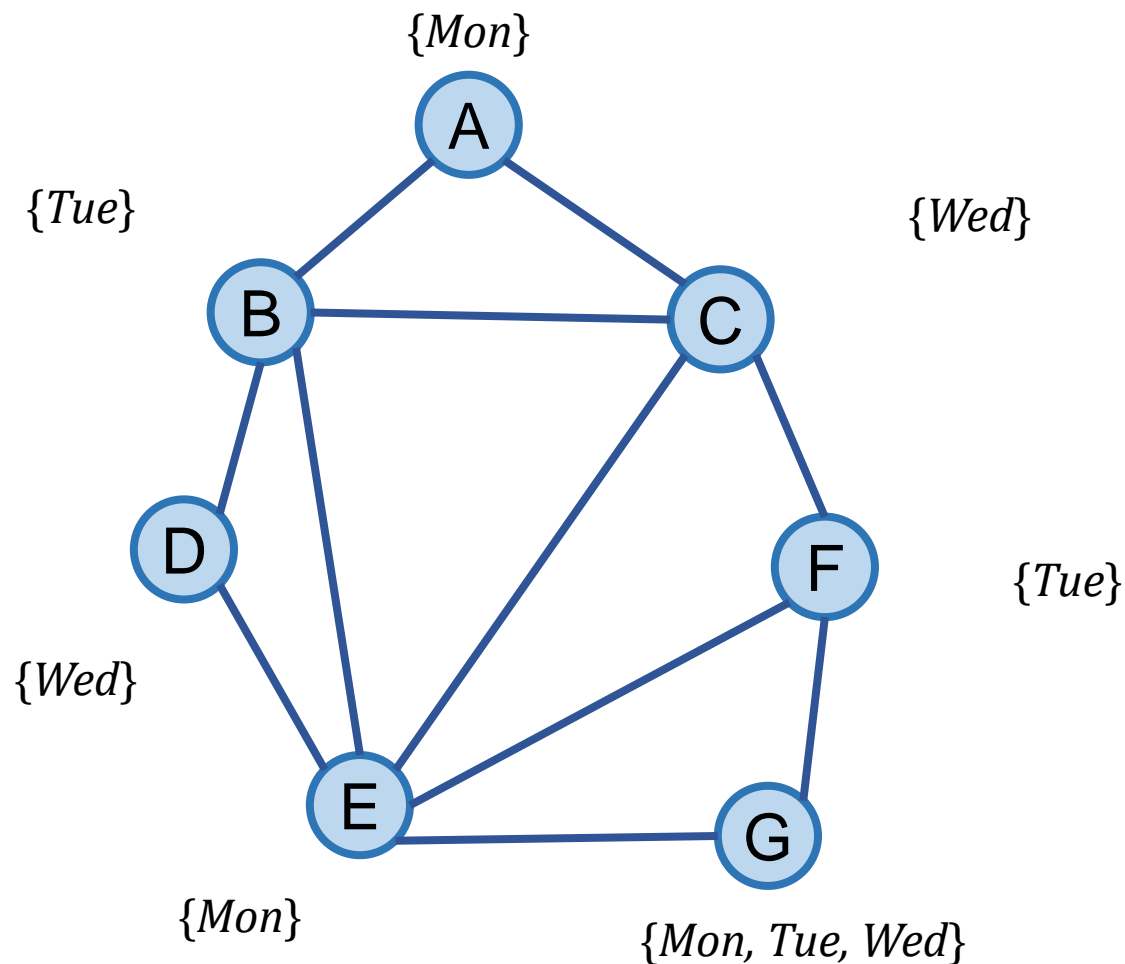
- 有

- 尝试定义域里另外一个值



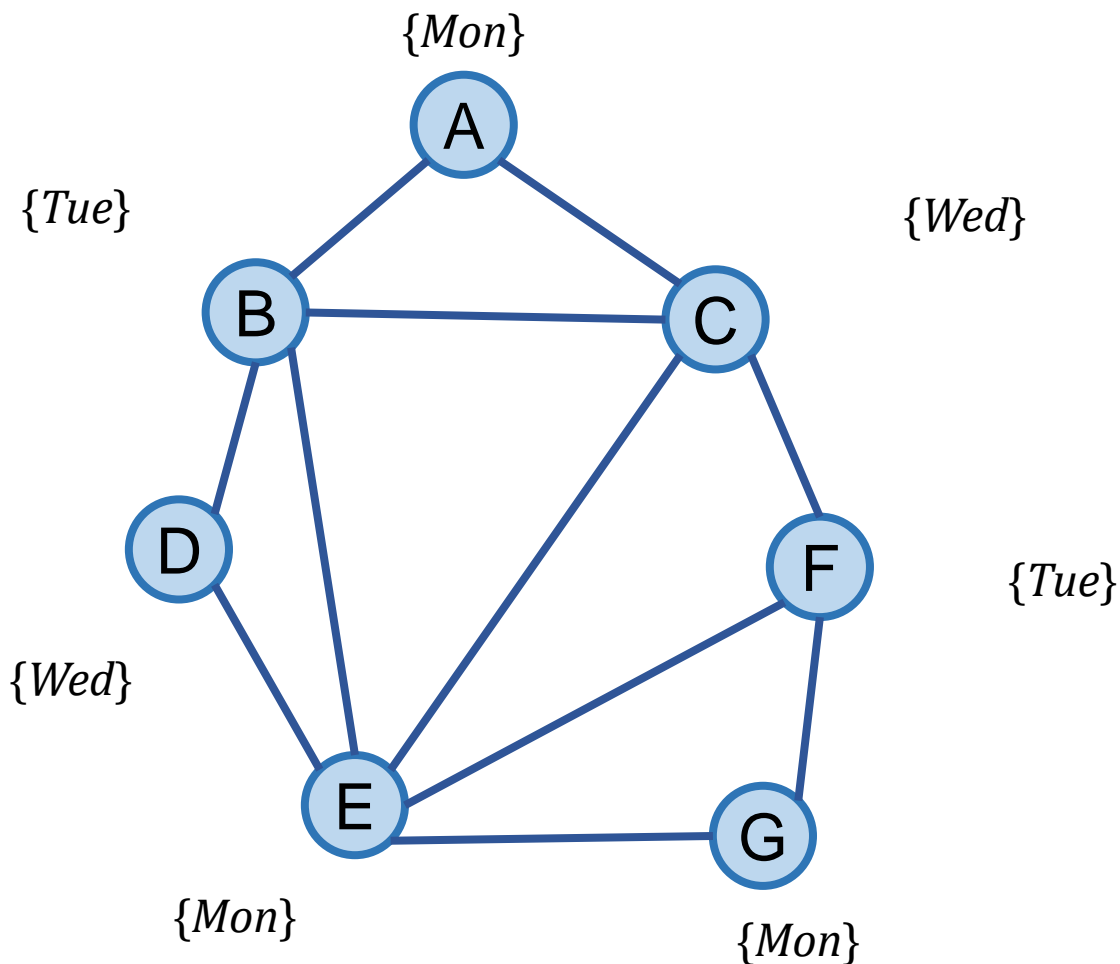
考试时间安排

- 对变量 B 赋值
 - 违背约束条件?
 - 没有
 - 执行算法使弧一致
 - 赋值下一个变量 D
 - 有
 - 尝试定义域里另外一个值



考试时间安排

- 对变量 B 赋值
 - 违背约束条件?
 - 没有
 - 执行算法使弧一致
 - 赋值下一个变量 D
 - 有
 - 尝试定义域里另外一个值



执行算法使弧一致

- 每次对变量赋值时都执行使弧一致性的算法
 - 对 X 进行赋值时, 运行 **AC-3** 算法。
 - 从 X 与其所有相邻结点 Y 的二元约束 (Y, X) 开始

回溯搜索 Backtracking search

function Backtrack(*assignment*, *problem*):

if *assignment* is complement: return *assignment* 所有变量都已经赋值完毕

var = Select-Unassigned-Var(*assignment*, *problem*) 选择一个还没有被赋值的变量

for *value* in Domain-Value(*var*, *assignment*, *problem*):

new_assignment = add {*var* = *value*} to *assignment* 继续赋值下一个变量

 if *value* is consistent with *new_assignment*: 没有违背任何约束

inferences = Inference(*new_assignment*, *problem*) 执行AC-3, 推断其他变量的值

 if *inferences* ≠ failure:

 add *inferences* to *new_assignment* 将推断的值进行赋值

result = Backtrack(*new_assignment*, *problem*)

 if *result* ≠ failure: return *result* 如果failure, 直接循环下一个value

return *failure*

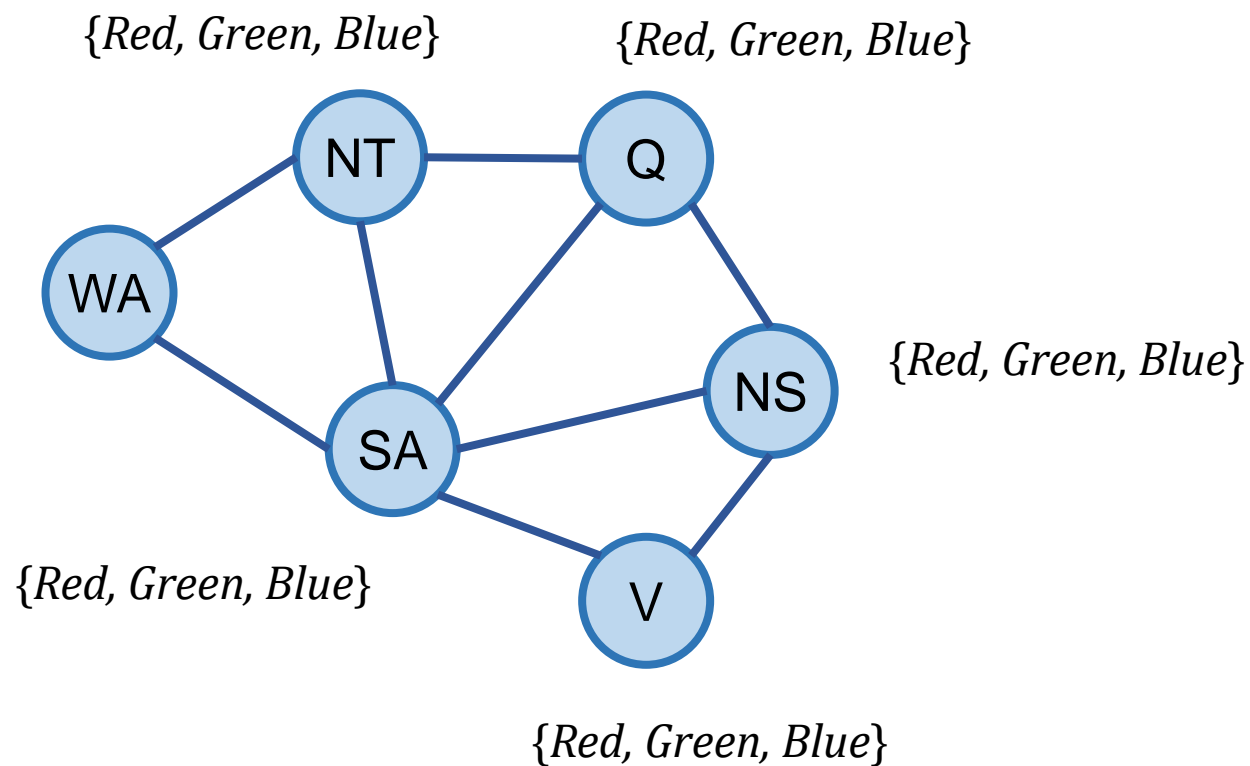
练习 #13

- 地图填色问题

- 顺时针顺序

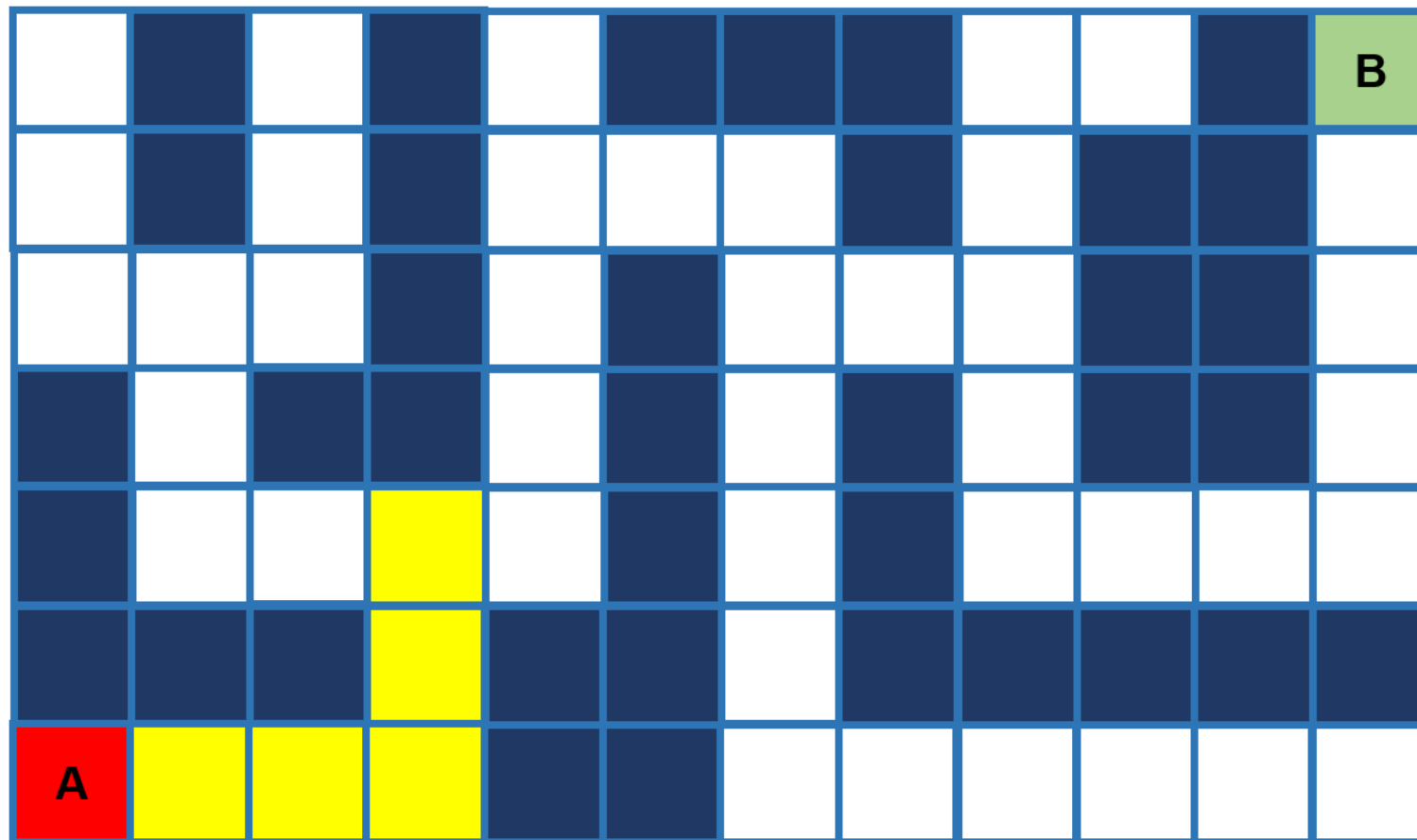
- 每次赋值后执行AC-3

{Red, Green, Blue}



下一步选择探索哪个节点?

- 可能向右
 - 假设:
 - 知道坐标
 - 知道距离



改进回溯搜索

function Backtrack(*assignment*, *problem*):

if *assignment* is complement: return *assignment* 所有变量都已经赋值完毕

var = **Select-Unassigned-Var**(*assignment*, *problem*) 选择一个还没有被赋值的变量

for *value* in Domain-Value(*var*, *assignment*, *problem*):

new_assignment = add {*var* = *value*} to *assignment* 继续赋值下一个变量

 if *value* is consistent with *new_assignment*: 没有违背任何约束

inferences = Inference(*new_assignment*, *problem*) 执行AC-3, 推断其他变量的值

 if *inferences* ≠ failure:

 add *inferences* to *new_assignment* 将推断的值进行赋值

result = Backtrack(*new_assignment*, *problem*)

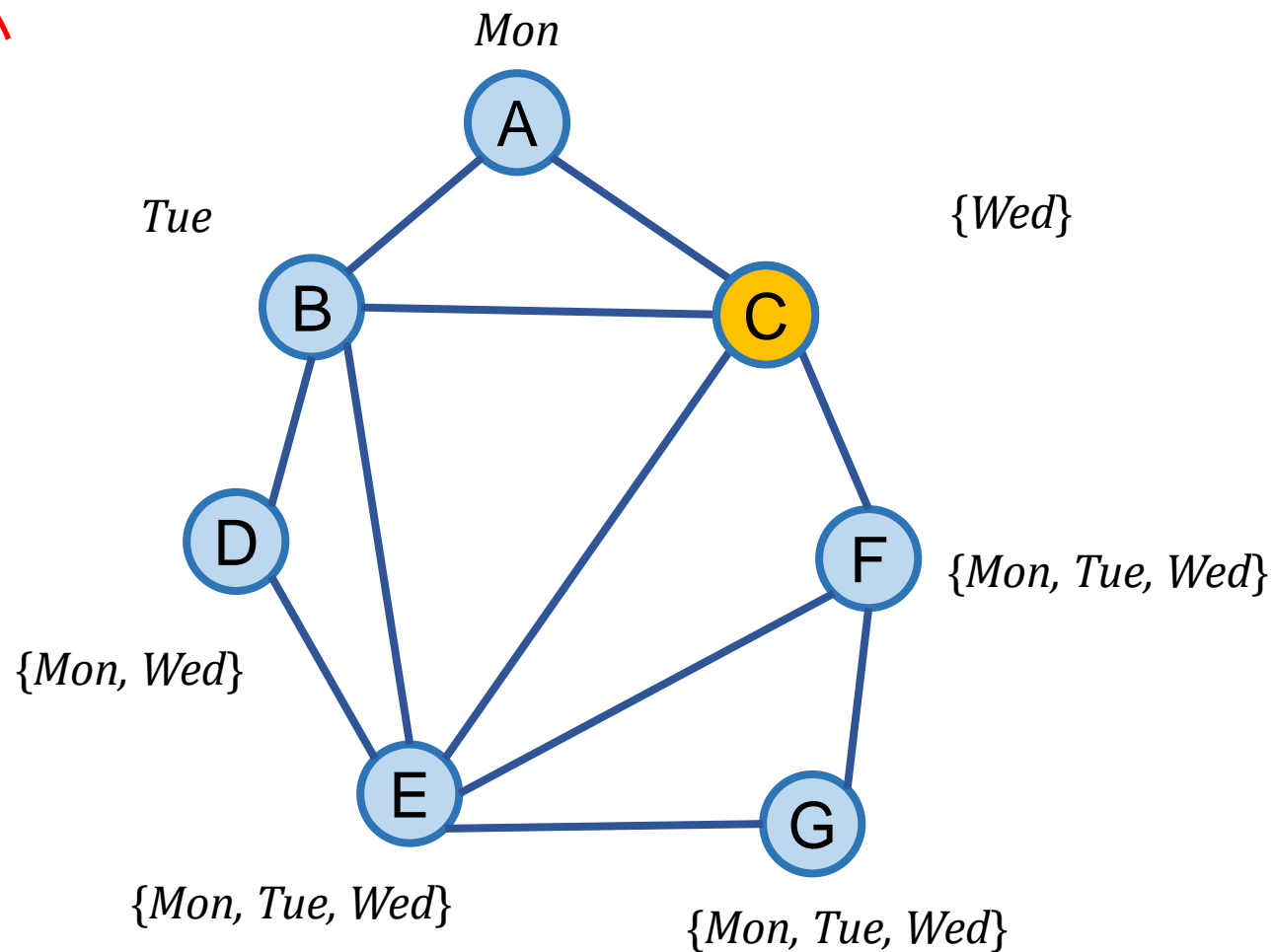
 if *result* ≠ failure: return *result* 如果failure, 直接循环下一个value

return failure

Select-Unassigned-Var 函数

- 选择定义域中**可选值的数量最小**的变量

- 最小剩余值 Minimum Remaining Values (MRV) heuristic
- 最容易被限制或不满足
 - 减少之后可能需要回溯的次数



练习 #14

- 有一个四位数

- 四位数是奇数

- 四位数的每个位置的数字由1-5构成

- 四位数每个位置的数字都比它左侧位置的数字更大

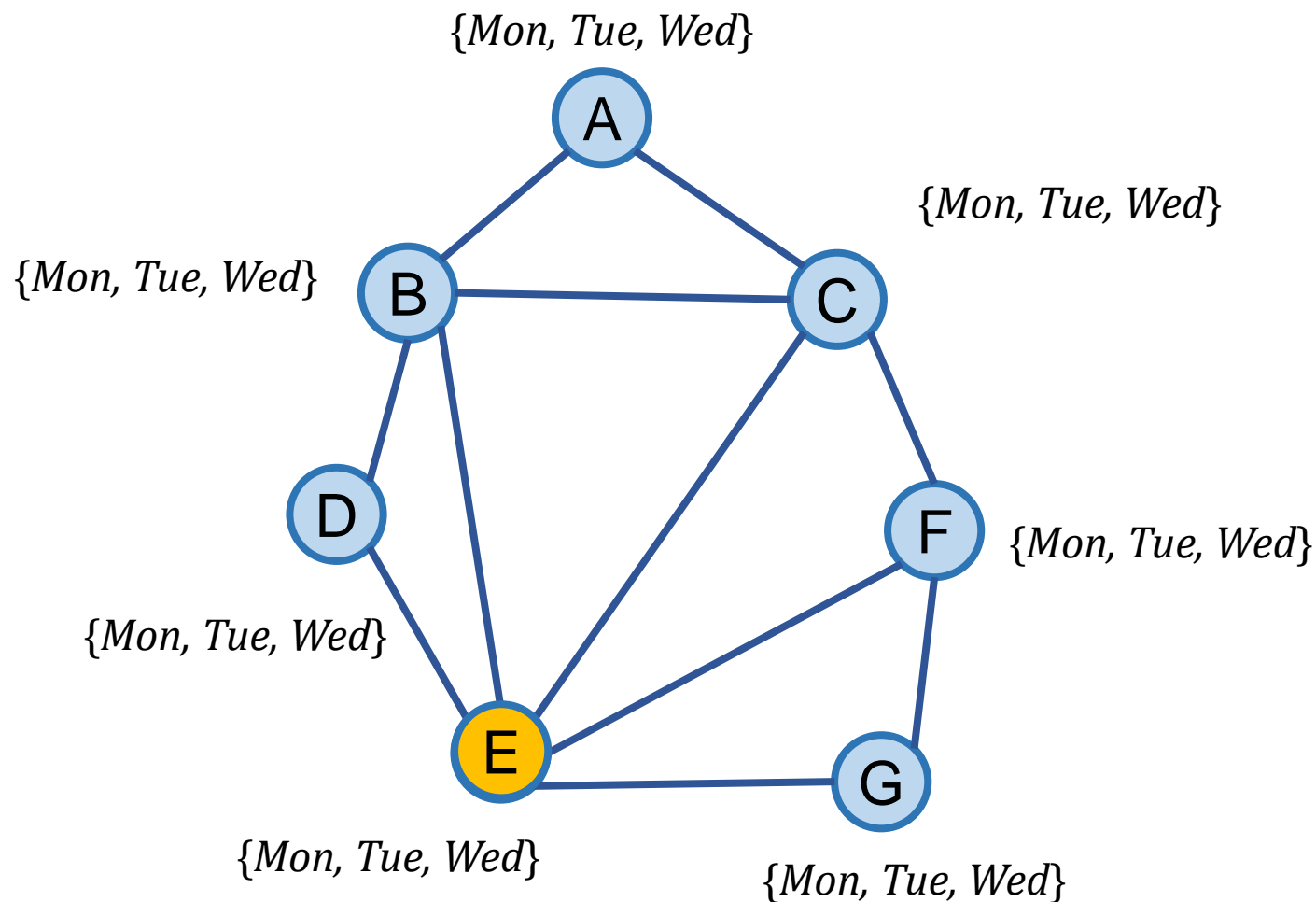
A	B	C	D
1 2	2 3	3 4	5

- 问题：

- 如果我们依据最小残留值 Minimum Remaining Values (MRV) heuristic 来选择下一个赋值的变量，我们会先赋值哪一个变量？(都可以的情况下，优先选择左侧数字)

Select-Unassigned-Var 函数

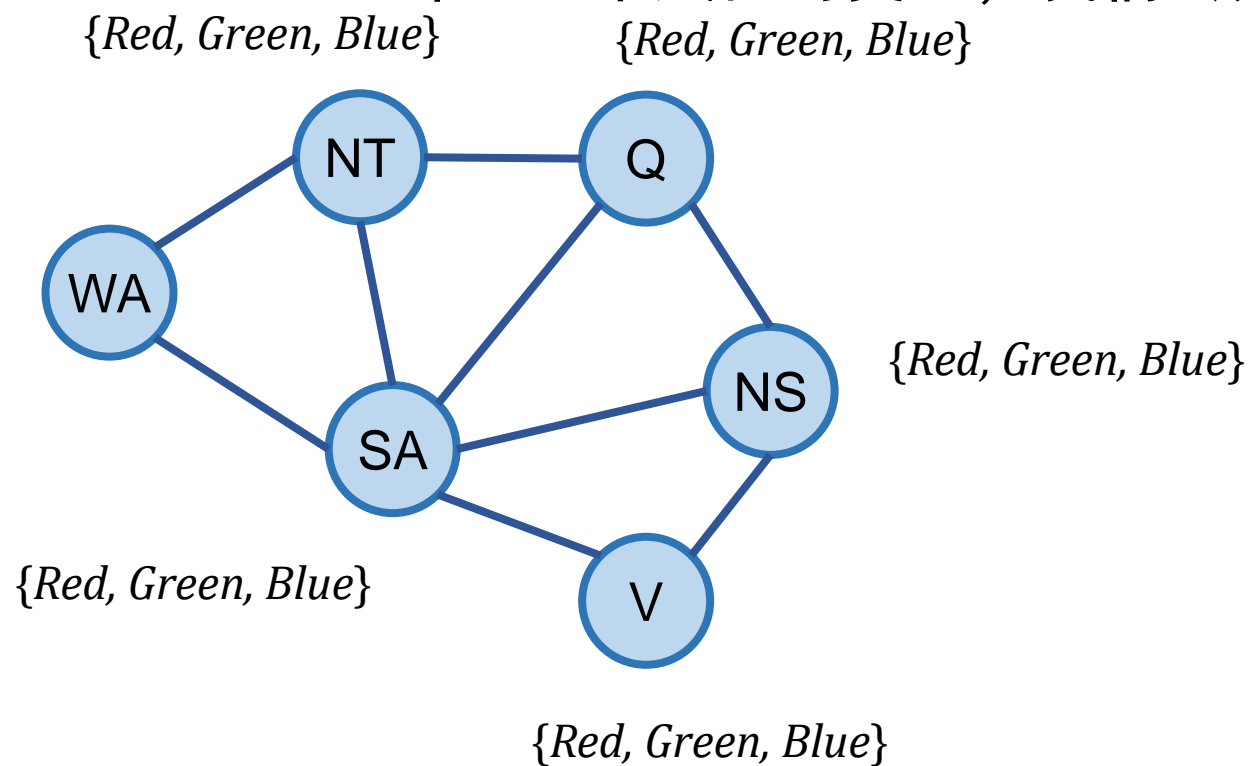
- 选择连接弧数最多的变量
 - 度启发式 Degree heuristic
 - 该变量影响很多其他变量的定义域，可以更快的执行算法



练习 #15

- 地图填色问题

- 如果我们依据度 degree heuristic 来选择下一个赋值的变量，我们会先赋值哪一个变量？



改进回溯搜索

function Backtrack(*assignment*, *problem*):

if *assignment* is complement: return *assignment* 所有变量都已经赋值完毕

var = Select-Unassigned-Var(*assignment*, *problem*) 选择一个还没有被赋值的变量

for *value* in Domain-Value(*var*, *assignment*, *problem*):

new_assignment = add {*var* = *value*} to *assignment* 继续赋值下一个变量

 if *value* is consistent with *new_assignment*: 没有违背任何约束

inferences = Inference(*new_assignment*, *problem*) 执行AC-3, 推断其他变量的值

 if *inferences* ≠ failure:

 add *inferences* to *new_assignment* 将推断的值进行赋值

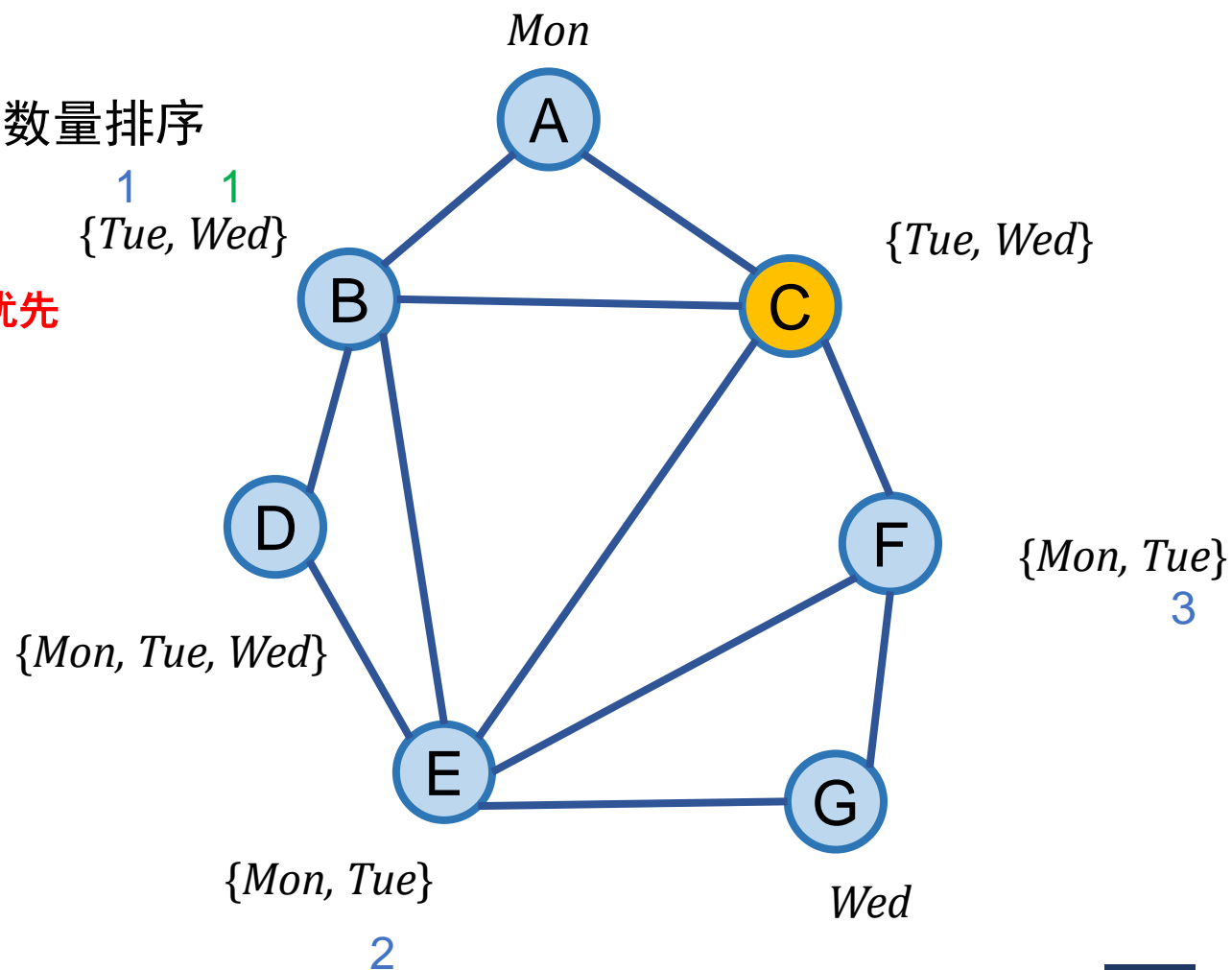
result = Backtrack(*new_assignment*, *problem*)

 if *result* ≠ failure: return *result* 如果failure, 直接循环下一个value

return failure

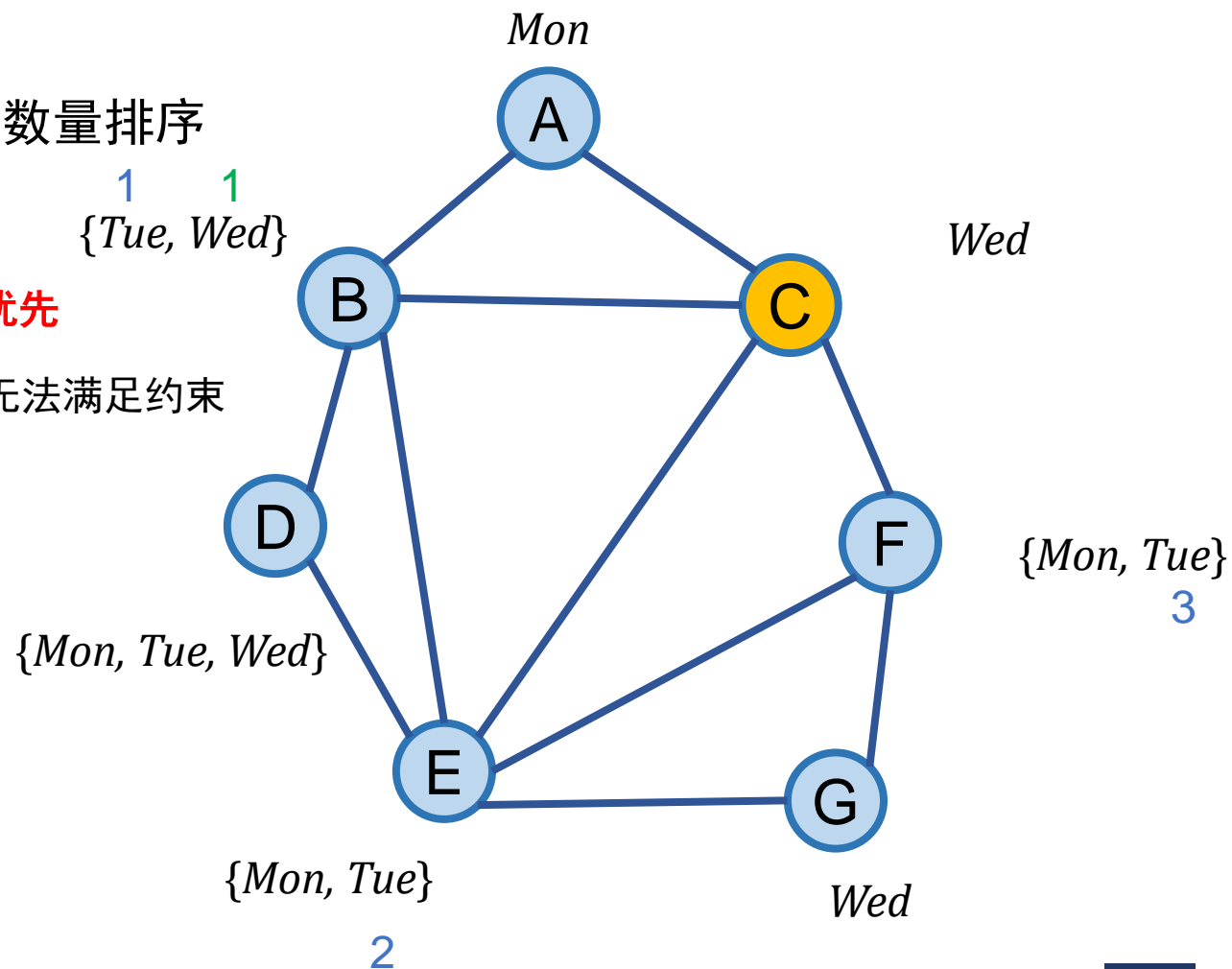
Domain-Value 函数

- 先选择一些更可能是答案的值
 - 按照相邻变量中各个值会被删除的数量排序
- 返回
 - 最少约束值(least constraint value)优先



Domain-Value 函数

- 先选择一些更可能是答案的值
 - 按照相邻变量中各个值会被删除的数量排序
返回
 - **最少约束值(least constraint value)优先**
 - 约束值多的可能在赋值别的变量时会无法满足约束
 - 仍然保留更多可能



练习 #16

- 有一个四位数

- 四位数是奇数

- 四位数的每个位置的数字由1-5构成

- 四位数每个位置的数字都比它左侧位置的数字更大

- 问题：

- 假设已经对D赋值5，下一个是对C进行赋值，那么赋值的顺序应该是？

A	B	C	D
1 2	2 3	3 4	5

有问题吗？

- 请随时举手提问。



BUSS 3620.人工智能导论

#4.3: 迭代改进

刘佳璐

安泰经济与管理学院

上海交通大学

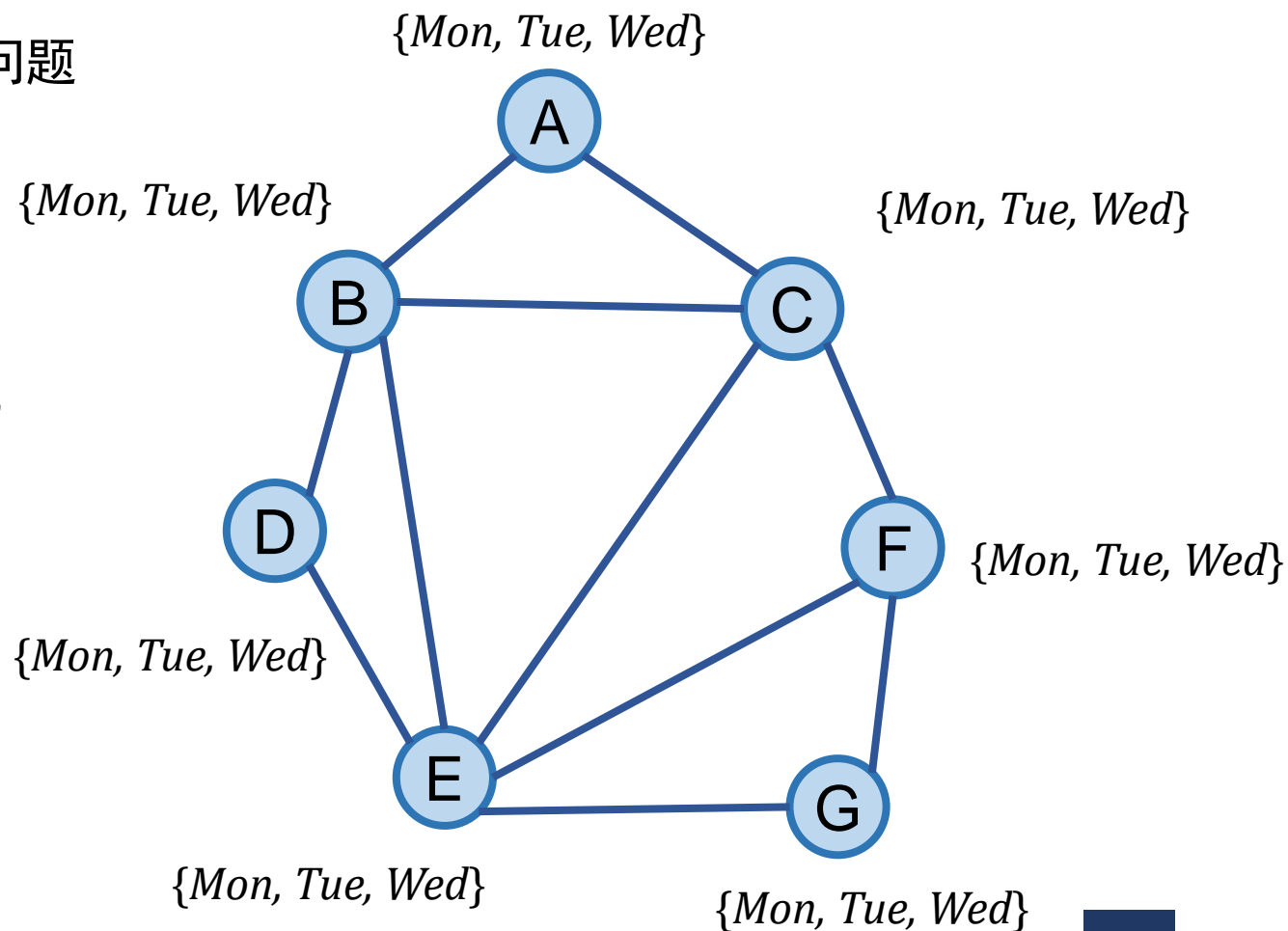
练习 #17

- 考试时间安排
 - 可看做一个可以用局部搜索解决的问题

① 每一个状态指什么？

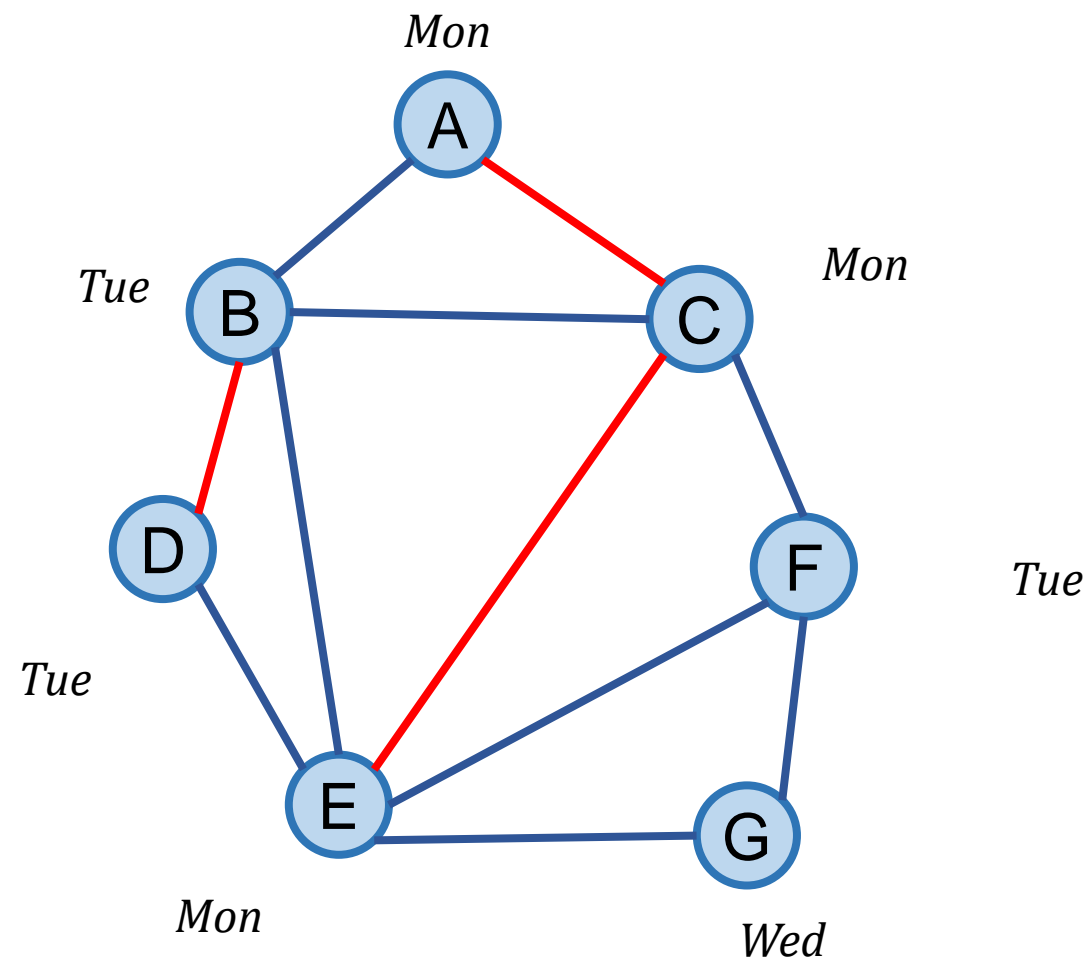
② 每一个状态的代价函数是什么？

③ 相邻状态是什么？



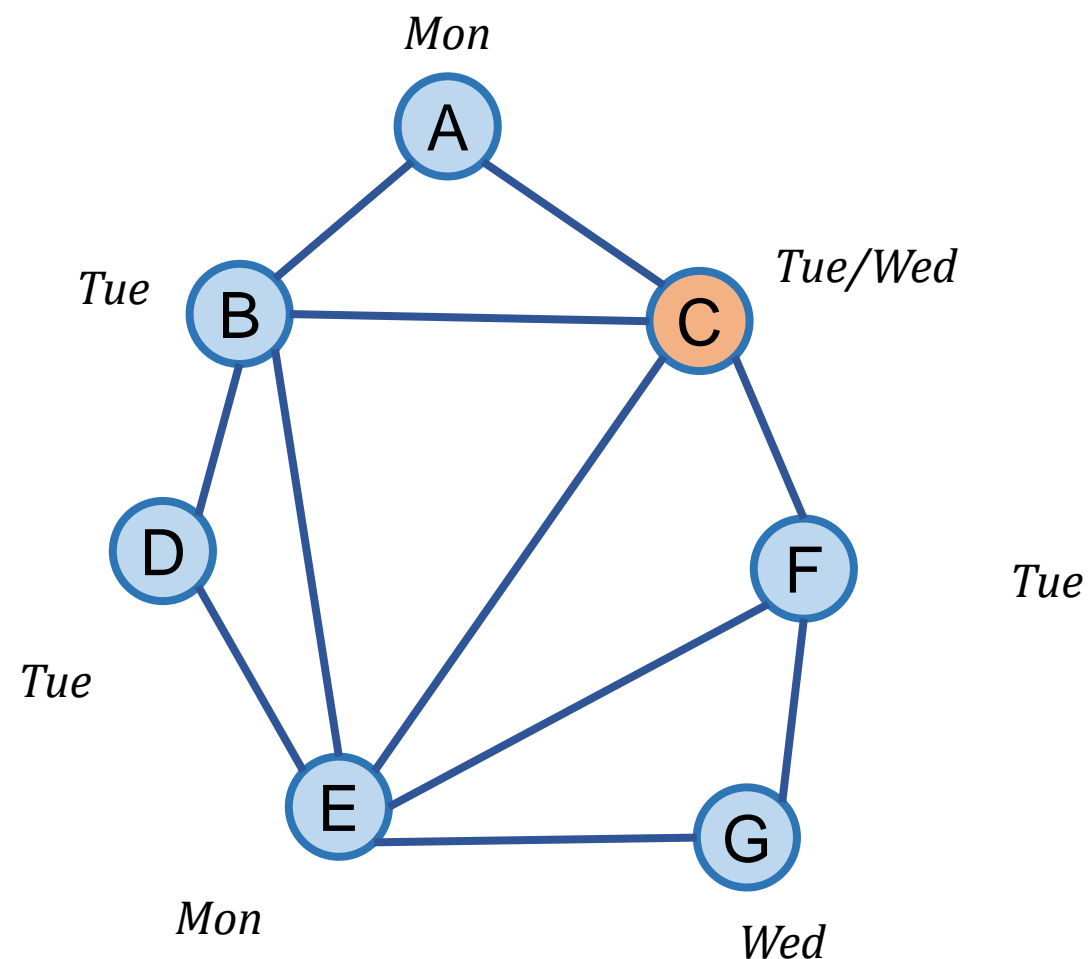
考试时间安排 – 局部搜索

- 初始状态
- 代价函数：3



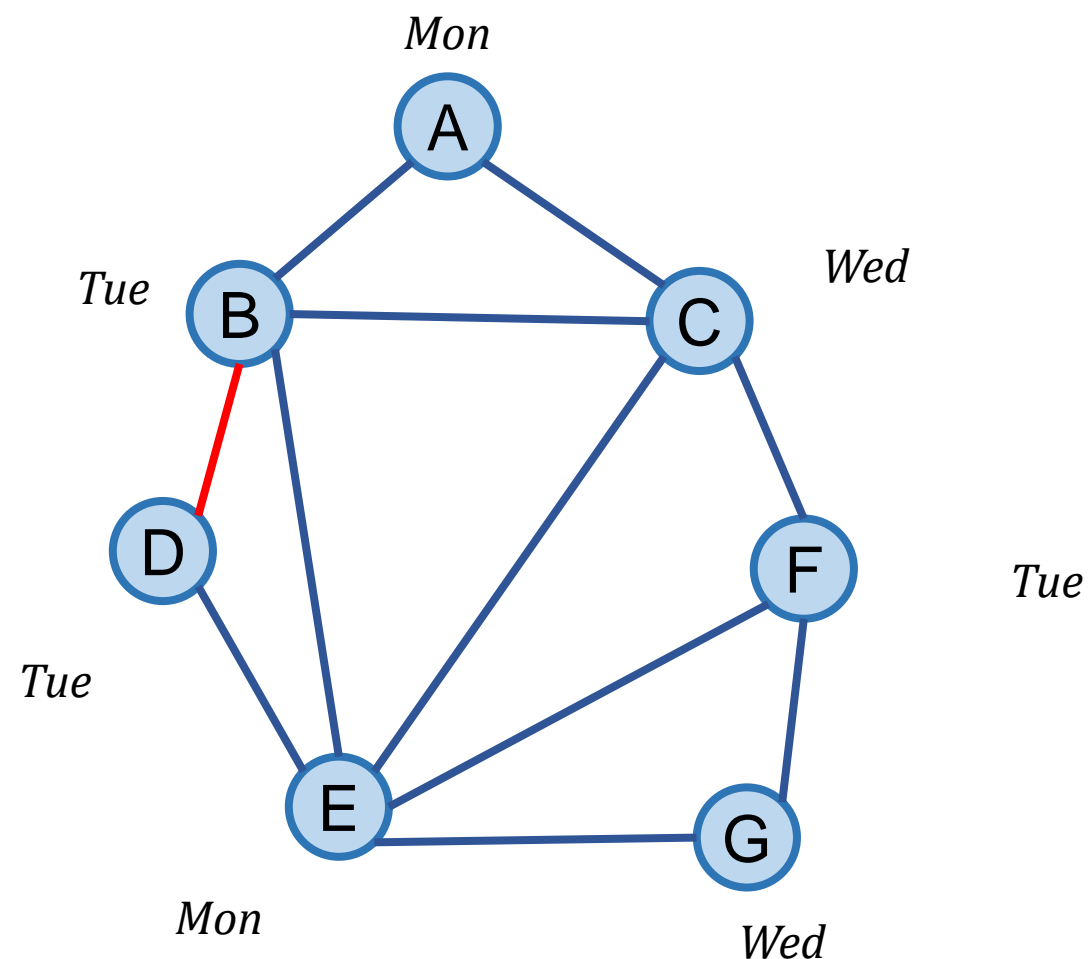
考试时间安排 – 局部搜索

- 初始状态
- 代价函数：3
 - Tue: 3
 - Wed: 1



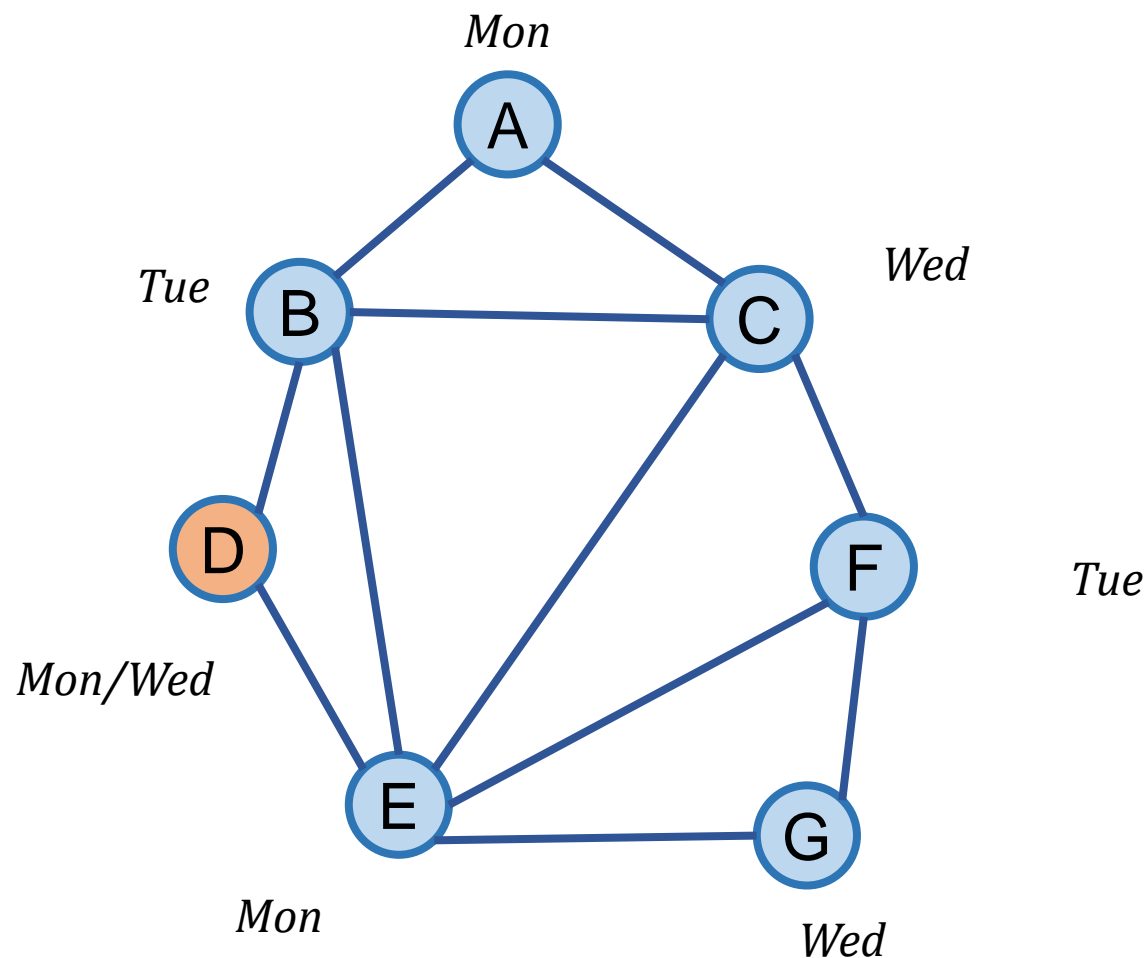
考试时间安排 – 局部搜索

- 相邻状态
- 代价函数：1



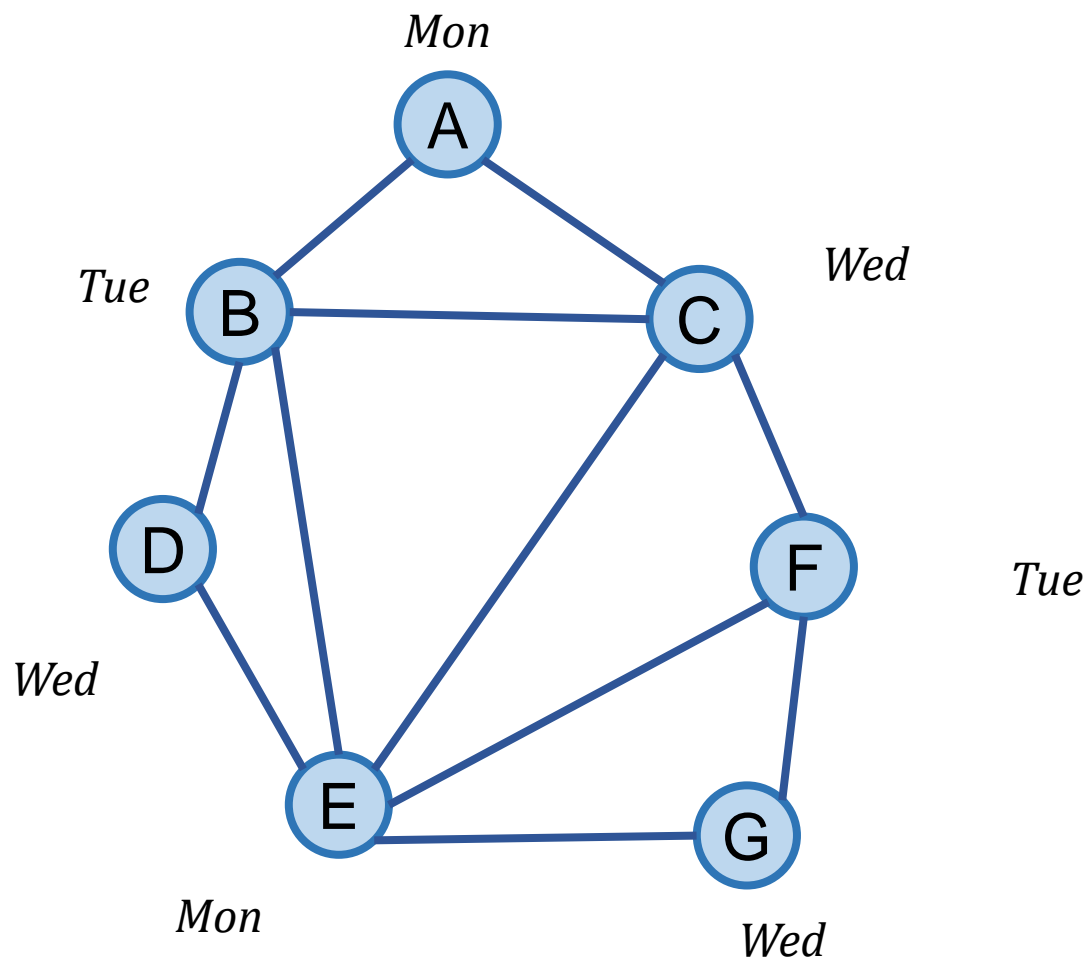
考试时间安排 – 局部搜索

- 相邻状态
- 代价函数：1
 - Mon: 1
 - Wed: 0



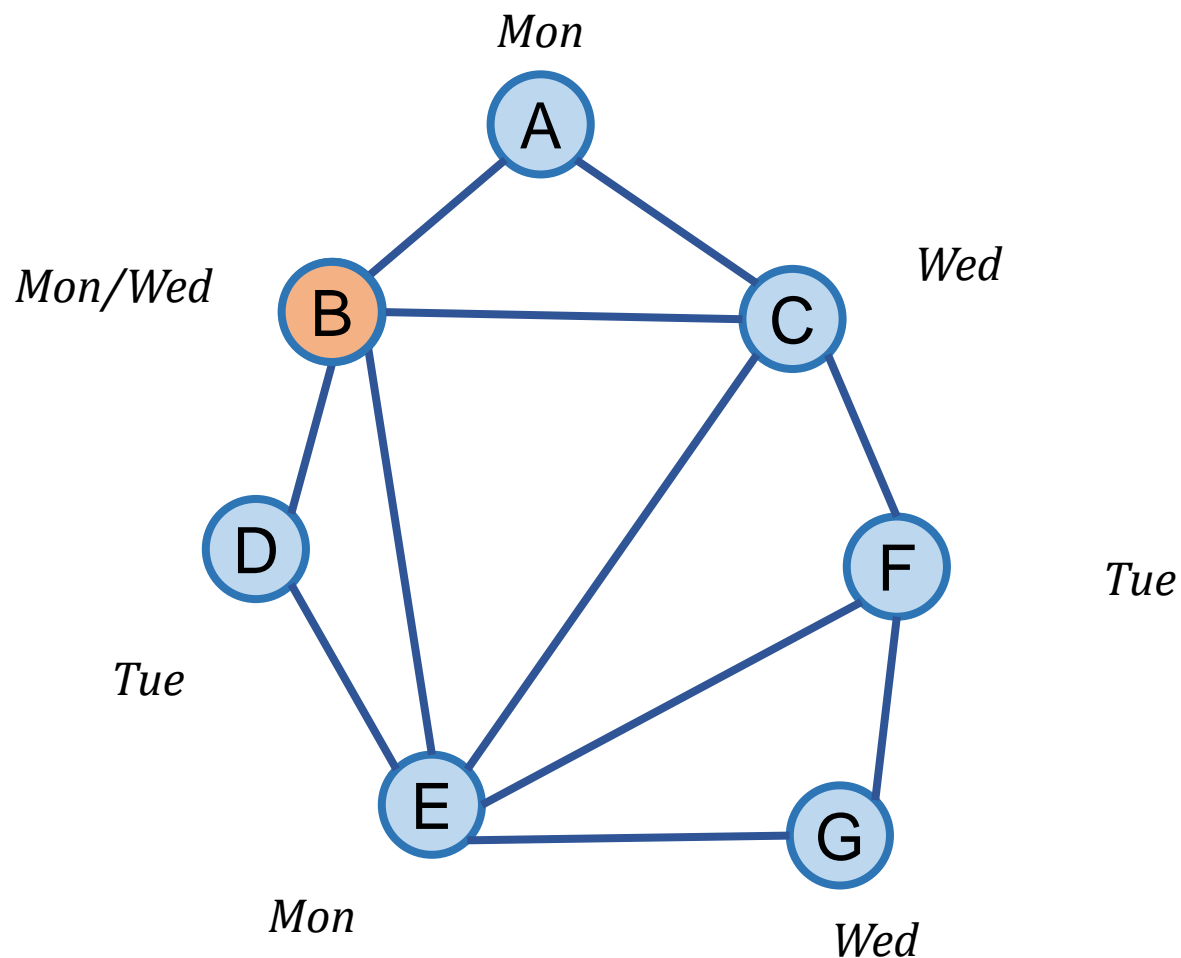
考试时间安排 – 局部搜索

- 相邻状态
- 代价函数：0



考试时间安排 – 局部搜索 – 可能出现的问题

- 相邻状态
- 代价函数：1
 - Mon: 1
 - Wed: 1
 - 可能会找不到解
- 局部搜索可能会运行很久
 - 但是通常都很快



小结

- 局部搜索 Local search
 - 城市规划
 - 旅行商问题 Traveling sales man's problem
 - 数独
 - 约束满足问题 Constraint satisfaction problem
 - 局部搜索 Local search
 - 回溯搜索 Backtracking search
- 如何抽象问题成状态、代价函数、相邻状态，使AI理解

有问题吗？

- 请随时举手提问。

