**Exercise 6: Designing Factored States & CSPs**

**Problem 1:**

**Consider the problems of Vacuum-Cleaner World (module 2) and the Eight-Puzzle problem. For each of these two problems, illustrate your specification of them as CSPs. Explain why, or why not, these problems are appropriate to consider as a CSP.**

**Vacuum-Cleaner World as a CSP:**

Let’s say we have 2 locations A and B.

* Variables:
  + L\_A\_dirt (is location A dirty)
  + L\_B\_dirt (is location B dirty)
  + L\_vacuum (location of vacuum)
* Domains:
* Constraints:
  + <(L\_A\_dirt, L\_B\_dirt), {F, F}>

We could consider this as CSP but it’s not necessary. It is because we already know the desired state for the problem, so we don’t need CSP to search for it. Since the problem is quite simple and has few states, we can use brute force with if-else statements instead of CSP.

**Eight-Puzzle as a CSP:**

|  |  |  |
| --- | --- | --- |
| L\_00 | L\_01 | L\_02 |
| L\_10 | L\_11 | L\_12 |
| L\_20 | L\_21 | L\_22 |

* Variables:
  + L\_00, L\_01, L\_02, L10, L\_11, L\_12, L\_20, L\_21, L\_22
* Domains:
  + The domain of each variable is the same as below:
    - D­L = {None, 1, 2, 3, 4, 5, 6, 7, 8}
* Constraints:
  + <(L\_00, L\_01, L\_02,…, L\_22), Alldiff>
  + <(L\_00, L\_01, L\_02,…, L\_22), {None, 1, 2, 3, 4, 5, 6, 7, 8}>

This is not a good fit for CSP. CSP is about finding a goal state, not a sequence of actions that reaches the goal state. For this problem, we already know the desired state. The challenge is not finding a valid state, but finding the sequence of moves to reach it. In CSP problems, we usually ask “is there some state that satisfies our criteria?” Here, we already know which state satisfies the constraints

**Problem 2:**

**Consider the problem of placing *k* knights on an *n × n* chessboard such that no two knights are attacking each other, where *k* is given and *k ≤ n2*.**

1. **Choose a CSP formulation. In your formulation, what are the variables?**
2. **What are the domains of each variable?**
3. **What sets of variables are constrained, and how?**
4. **Consider the problem of putting "as many knights as possible" on the board without any attacks. Explain how to solve this with local search, by defining appropriate *ACTIONS* and *RESULT* functions and a sensible objective function.**
5. Variables:
   * We treat each knight themselves as variable. Each variable represents the position of a knight.
   * {K1, K2, K3, K4, …, Kk}
6. Domains:
   * Each knight can be placed on position of the board
   * DK\_i = {(x, y) | 1 <= x <= n, 1 <= y <= n}
7. Constraints:
   * All knights have to be in different positions
     + <( K1, K2, K3, K4, …, Kk), Alldiff>
   * Knights must be within the board
     + <( K1, K2, K3, K4, …, Kk), {(x, y) | 1 <= x <= n, 1 <= y <= n} >
   * No two knights should attack each other

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | O |  | O |  |  |
| O |  |  |  | O |  |
|  |  | K |  |  |  |
| O |  |  |  | O |  |
|  | O |  | O |  |  |
|  |  |  |  |  |  |

* + - For the board above, let’s say K is the knight and O’s are the positions where the knight can go. If we have another knight, it cannot be on position O.
    - <(Ki, Kj), {(x1, y1), (x2, y2) | (|x1 – x2|, |y­1 – y2|) (2, 1) AND (x1, y1), (x2, y2) | (|x1 – x2|, |y­1 – y2|) (1, 2)}>

1. Maximize number of knights with local search
   * State Representation:
     + S = {(x1, y1), (x2, y2), …, (xk, yk)}
     + This is the initial state of the knights on the board
   * Objective Function:
     + f(S) =
     + The objective function measures the number of attacks. In this case, 0 is the optimal value meaning no knights can attack each other.
     + Our goal is to minimize f(S).
   * Actions:
     + Initialize k knights randomly on the board and compute initial attack count f(S)
     + Repeat until convergence:
       - Select a random K\_i knight
       - Generate a neighbor state S’ by moving or swapping knights
       - Compute f(S’)
       - If f(S’) < f(S), then accept the move
       - Otherwise, reject the move
     + If we stuck in a local minimum, then restart with a new random knight placement
     + Stop when f(S) = 0 or we’re done with the iteration
   * Result function:
     + Result(S, action) = S’
     + This function computes the new state after the action is applied

**Problem 3:**

**Consider the following logic puzzle.**

**In five houses, each with a different color, live five persons of different nationalities, each of whom prefers a different brand of candy, a different drink, and a different pet.**

**Consider the facts below. We would like to know where the zebra lives, and in which house resides the person that drinks water.**

* **The Englishman lives in the red house.**
* **The Spaniard owns the dog.**
* **The Norwegian lives in the first house on the left.**
* **The green house is immediately to the right of the ivory house.**
* **The man who eats Hershey bars lives in the house next to the man with the fox.**
* **Kit Kats are eaten in the yellow house.**
* **The Norwegian lives next to the blue house.**
* **The Smarties eater owns snails.**
* **The Snickers eater drinks orange juice.**
* **The Ukrainian drinks tea.**
* **The Japanese eats Milky Ways.**
* **Kit Kats are eaten in a house next to the house where the horse is kept.**
* **Coffee is drunk in the green house.**
* **Milk is drunk in the middle house.**

**Now, design a representation of this problem as a CSP. Try to produce two different designs. Describe at least two benefits and drawbacks of one design over the other.**

**Design 1:**

* Variables:
  + H\_1\_color, H\_2\_color, H\_3\_color, H\_4\_color, H\_5\_color
  + H\_1\_nationality, H\_2\_ nationality, H\_3\_ nationality, H\_4\_ nationality, H\_5\_ nationality
  + H\_1\_candy, H\_2\_ candy, H\_3\_ candy, H\_4\_ candy, H\_5\_ candy
  + H\_1\_drink, H\_2\_ drink, H\_3\_ drink, H\_4\_ drink, H\_5\_ drink
  + H\_1\_pet, H\_2\_ pet, H\_3\_ pet, H\_4\_ pet, H\_5\_ pet
* Domains:
  + Colors: {red, blue, green, yellow, ivory}
  + Nationalities: {English, Spaniard, Norwegian, Ukrainian, Japanese}
  + Candy: {Hershey, KitKat, Smarties, Snickers, MilkyWay}
  + Drinks: {coffee, tea, milk, orange juice, water}
  + Pets: {dog, fox, horse, snails, zebra}

Each house gets one unique value per attribute

* Constraints:
  + The Englishman lives in the red house
    - <(H\_i\_nationality, H\_i\_color), {(English, red)}>
  + The Spaniard owns the dog
    - <(H\_i\_nationality, H\_i\_pet), {(Spaniard, dog)}>
  + The Norwegian lives in the first house
    - <(H\_1\_nationality), {Norwegian}>
  + The green house is immediately to the right of the ivory house
    - <(H\_i\_color, H\_(i+1)\_color), {(ivory, green)}>
  + The man who eats Hershey bars lives next to the man with the fox
    - <(H\_i\_candy, H\_(i±1)\_pet), {(Hershey, fox)}>
  + Kit Kats are eaten in the yellow house
    - <(H\_i\_candy, H\_i\_color), {(KitKat, yellow)}>
  + The Norwegian lives next to the blue house
    - <(H\_i\_nationality, H\_(i±1)\_color), {(Norwegian, blue)}>
  + The Smarties eater owns snails
    - <(H\_i\_candy, H\_i\_pet), {(Smarties, snails)}>
  + The Snickers eater drinks orange juice
    - <(H\_i\_candy, H\_i\_drink), {(Snickers, orange juice)}>
  + The Ukrainian drinks tea
    - <(H\_i\_nationality, H\_i\_drink), {(Ukrainian, tea)}>
  + The Japanese eats Milky Ways
    - <(H\_i\_nationality, H\_i\_candy), {(Japanese, MilkyWay)}>
  + Kit Kats are eaten in a house next to the house where the horse is kept
    - <(H\_i\_candy, H\_(i±1)\_pet), {(KitKat, horse)}>
  + Coffee is drunk in the green house
    - <(H\_i\_drink, H\_i\_color), {(coffee, green)}>
  + Milk is drunk in the middle house
    - <(H\_3\_drink), {milk}>
  + Each attribute must have exactly one value per house
    - <(H\_1\_color, H\_2\_color, H\_3\_color, H\_4\_color, H\_5\_color), Alldiff>
    - <(H\_1\_nationality, H\_2\_nationality, H\_3\_nationality, H\_4\_nationality, H\_5\_nationality), Alldiff>
    - <(H\_1\_candy, H\_2\_candy, H\_3\_candy, H\_4\_candy, H\_5\_candy), Alldiff>
    - <(H\_1\_drink, H\_2\_drink, H\_3\_drink, H\_4\_drink, H\_5\_drink), Alldiff>
    - <(H\_1\_pet, H\_2\_pet, H\_3\_pet, H\_4\_pet, H\_5\_pet), Alldiff>

**Design 2:**

* Variables: each nationality, color, drink, pet, and candy is treated as a variable
  + English, Spaniard, Norwegian, Ukrainian, Japanese
  + Red, Blue, Green, Yellow, Ivory
  + Hershey, KitKat, Smarties, Snickers, MilkyWay
  + Coffee, Tea, Milk, OrangeJuice, Water
  + Dog, Fox, Horse, Snails, Zebra
* Domains: each variable can be assigned to one of the five houses
  + D = {1, 2, 3, 4, 5}
* Constraints:
  + Each house must have exactly one nationality, one color, one candy, one pet, and one drink
    - <(English, Spaniard, Norwegian, Ukrainian, Japanese), Alldiff>
    - <(Red, Blue, Green, Yellow, Ivory), Alldiff>
    - <(Hershey, KitKat, Smarties, Snickers, MilkyWay), Alldiff>
    - <(Coffee, Tea, Milk, OrangeJuice, Water), Alldiff>
    - <(Dog, Fox, Horse, Snails, Zebra), Alldiff>
  + The Englishman lives in the red house
    - <(English, red), English = red>
  + The Spaniard owns the dog
    - <(Spaniard, Dog), Spaniard = Dog>
  + The Norwegian lives in the first house
    - <(Norwegian), Norwegian = 1>
  + The green house is immediately to the right of the ivory house
    - <(Green, Ivory), Green = Ivory + 1>
  + The man who eats Hershey bars lives next to the man with the fox
    - <(Hershey, Fox), |Hershey - Fox| = 1>
  + Kit Kats are eaten in the yellow house
    - <(KitKat, Yellow), KitKat = Yellow>
  + The Norwegian lives next to the blue house
    - <(Norwegian, Blue), |Norwegian - Blue| = 1>
  + The Smarties eater owns snails
    - <(Smarties, Snails), Smarties = Snails>
  + The Snickers eater drinks orange juice
    - <(Snickers, OrangeJuice), Snickers = OrangeJuice>
  + The Ukrainian drinks tea
    - <(Ukrainian, Tea), Ukrainian = Tea>
  + The Japanese eats Milky Ways
    - <(Japanese, MilkyWay), Japanese = MilkyWay>
  + Kit Kats are eaten in a house next to the house where the horse is kept
    - <(KitKat, Horse), |KitKat - Horse| = 1>
  + Coffee is drunk in the green house
    - <(Coffee, Green), Coffee = Green>
  + Milk is drunk in the middle house
    - <(Milk), Milk = 3>

|  |  |  |
| --- | --- | --- |
|  | **Benefits** | **Drawbacks** |
| **Design 1** | * Easier to understand * Easier to express adjacency constraints | * More variables to track for constraint propagation * Complex constraint dependencies (some rules rely on unassigned houses). |
| **Design 2** | * Faster constraint propagation because all variables have the same domain * Simpler AllDiff constraints | * Less intuitive for humans * Adjacency constraints require numerical checking |