CS410: Artificial Intelligence 2021 Fall
Homework 2: Constraint Satisfaction Problems
Due date: 23:59:59 (GMT +08:00), October 22 2021

Wendi Chen 519021910071

1. **A* graph search**. Consider the following undirected graph shown in Figure 1 where we are searching from start state A to goal state G. The number over each edge is the transition cost. Additionally, we are given a heuristic function h as follows: $\{h(A) = 7, h(B) = 5, h(C) = 6, h(D) = 4, h(E) = 3, h(F) = 3, h(G) = 0\}$. Assume that, in case of ties, the search procedure uses an alphabetical order for tie-breaking.

   Find the sequence of nodes expanded by A* graph search algorithm, with problem-solving steps (i.e., updates for the frontier and explored set).

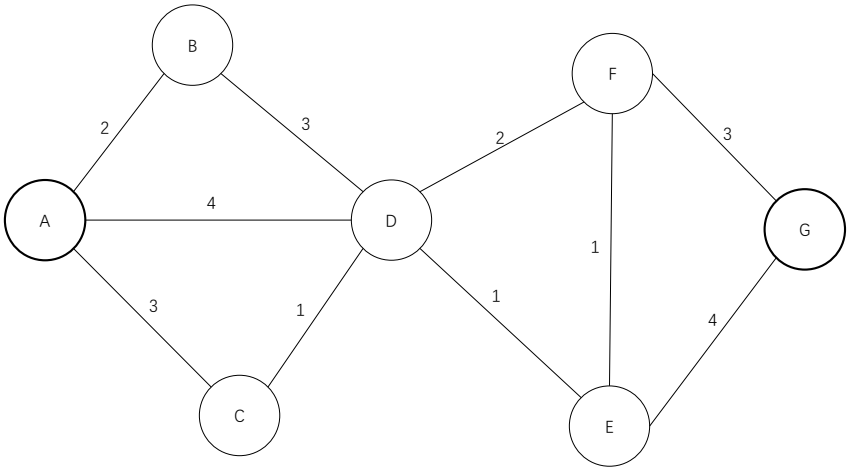


Figure 1: Problem 1.

Solution. According to the definition of A* graph search, we denote F as the frontier set and E as the explored set. Then we show the process of algorithm execution by showing the elements in these two sets for each iteration.

- $F = \{A : 7\}$, $E = \emptyset$
- $F = \{B : 7, C : 9, D : 8\}$, $E = \{A\}$
- $F = \{C : 9, D : 8\}$, $E = \{A, B\}$
- $F = \{C : 9, E : 8, F : 9\}$, $E = \{A, B, D\}$
- $F = \{C : 9, F : 9, G : 9\}$, $E = \{A, B, D, E\}$
- $F = \{F : 9, G : 9\}$, $E = \{A, B, C, D, E\}$
- $F = \{G : 9\}$, $E = \{A, B, C, D, E, F\}$
- $F = \emptyset$, $E = \{A, B, C, D, E, F, G\}$

So the sequence of nodes expanded by A* graph search is $A, B, D, E, C, F(, G)$. Besides, I also write some code to verify my answer.

```python
from queue import PriorityQueue

graph = {
    "A":{"B":2, "C":3, "D": 4},
    "B":{"A":2, "D":3},
    "C":{"A":3, "D":1},
    "D":{"A":4, "B":3, "C":1, "E":1, "F":2},
    "E":{"D":1, "F":1, "G":4},
    "F":{"D":2, "E":1, "G":3},
    "G":{"E":4, "F":3}
```

```
}

h = {"A":7, "B":5, "C":6, "D":4, "E":3, "F":3, "G":0}

explored, frontier = set(), PriorityQueue()
info, sequence = {}, []

frontier.put((h["A"]+0, "A"))
info["A"] = h["A"]+0

while not frontier.empty():
    print("F:"+str(["{}:{}".format(k,v) for k, v in info.items()]),
        "E:" + str([e for e in explored]))
    f, node = frontier.get()
    if node in explored: continue
    del info[node]
    sequence.append(node)
    explored.add(node)

    if node == "G":
        print("F:"+str(["{}:{}".format(k,v) for k, v in info.items()]),
            "E:" + str([e for e in explored]))
        print("Sequence:", sequence)
        break
    for k, v in graph[node].items():
        if k not in info and k not in explored:
            info[k] = f - h[node] + h[k] + v
            frontier.put((f - h[node] + h[k] + v, k))
        elif k in info and info[k] > f - h[node] + h[k] + v:
            info[k] = f - h[node] + h[k] + v
            frontier.put((f - h[node] + h[k] + v, k))
```

The result is the same.

```
F:['A:7'] E:[]
F:['B:7', 'C:9', 'D:8'] E:['A']
F:['C:9', 'D:8'] E:['A', 'B']
F:['C:9', 'E:8', 'F:9'] E:['D', 'A', 'B']
F:['C:9', 'F:9', 'G:9'] E:['E', 'D', 'A', 'B']
F:['F:9', 'G:9'] E:['C', 'B', 'D', 'A', 'E']
F:['G:9'] E:['F', 'C', 'B', 'D', 'A', 'E']
F:[] E:['F', 'C', 'B', 'D', 'A', 'E', 'G']
Sequence: ['A', 'B', 'D', 'E', 'C', 'F', 'G']
```

Figure 2: The Result of A* Graph Search.

2. **CSP formulation**. Consider the following three problems:

   (a) Rectilinear floor-planning: find non-overlapping places in a large rectangle for a number of smaller rectangles.

   (b) Class scheduling: There is a fixed number of professors and classrooms, a list of classes to be offered, and a list of possible time slots for classes. Each professor has a set of classes that he or she can teach.

   (c) Hamiltonian tour: given a network of cities connected by roads, choose an order to visit all cities in a country without repeating any.

   Determine each as a planning problem or an identification problem and explain why. Give precise formulations for each problem as constraint satisfaction problems.

   Recall a CSP consists of three components, X, D, and C. Note that there are many valid formulations, but you only need to provide one.

   Solution.

   (a) This problem is an identification problem. That's because we care about the place of each rectangle and the order (path) in which we place these rectangles doesn't matter. For a problem whose goal is important instead of path, it is an identification problem. Here's one formulation.

   - **Variables** X. For the i-th smaller rectangle X_i, $X_i = (x_i, y_i)$, where (x_i, y_i) means the position of the center of the i-th rectangle.
   - **Domains** D. For the i-th rectangle, we denotes its height and width as h_i and w_i, and the height and width of the large rectangle is H and W. Then the domain of X_i is $X_i \in [\frac{w_i}{2}, W - \frac{w_i}{2}] \times [\frac{h_i}{2}, H - \frac{h_i}{2}]$.
   - **Constraints** C. For every two different rectangles X_i and X_j $(i \neq j)$, we have

     $$[(x_i + \frac{w_i}{2} \leq x_j - \frac{w_j}{2}) \text{ or } (x_i - \frac{w_i}{2} \geq x_j + \frac{w_j}{2})] \text{ and } [(y_i + \frac{h_i}{2} \leq y_j - \frac{h_j}{2}) \text{ or } (y_i - \frac{h_i}{2} \geq y_j + \frac{h_j}{2})]$$

   (b) Similar to problem (a), this is an identification problem. That's because we want to assign professors, classrooms and times for each class, which means we care about the goal instead of the order (path). Here's one formulation.

   - **Variables** X. For the i-th class X_i, $X_i = (p_i, c_i, t_i)$, where p_i means professor p_i teaches this class, c_i means this class is held in classroom c_i and t_i means the time slot for this class is t_i.
   - **Domains** D. Denote the set of professors, classrooms and time slots as P, $Class$ and T respectively. Then the domain of X_i is $X_i \in P \times Class \times T$.
   - **Constraints** C. Denote the set of professors that can teach X_i as $P(X_j)$. Then for every class X_i, we have

     $$p_i \in P(X_j)$$

     Besides, for every two different classes X_i and X_j $(i \neq j)$, we have

     i. If $t_i = t_j$, then $p_i \neq p_j$.

     ii. If $t_i = t_j$, then $c_i \neq c_j$.

   (c) It is an identification problem. Because we want to visit all cities in a country without repeating any, all paths has the same depth. By the definition of identification problem, for a problem whose goal itself is important, it's an identification problem. Here's one formulation.

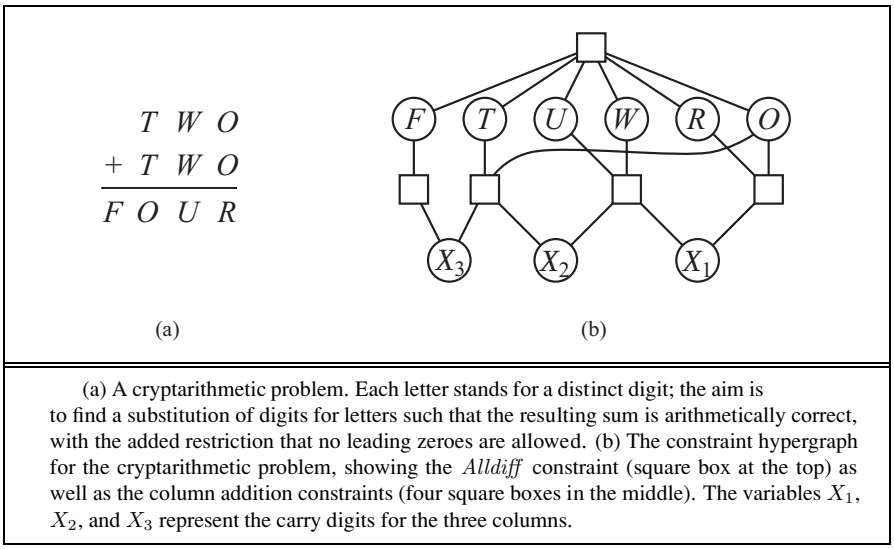
   - **Variables** X. If there are n cities in this country, we define $X = \{x_1, x_2, \cdots, x_n\}$, where x_i means the i-th city we visit.
   - **Domains** D. Denote the set of cities in this country as $City$. For every $x_i (1 \leq i \leq n)$, the domain of x_i is $x_i \in City$.
   - **Constraints** C. Denote the adjacent cities of x_i is $A(x_i)$. Then for every $x_i (1 \leq i \leq n-1)$, we have

     $$x_{i+1} \in A(x_i)$$

     Besides, for every two cities $x_i, x_j (1 \leq i \neq j \leq n)$, we have

     $$x_i \neq x_j$$

3. **Forward checking**. Solve the cryptarithmetic problem shown in Figure 3 step by step, using the strategy of backtracking with forward checking. Assume the variable order is $X_3 \to F \to X_2 \to X_1 \to O \to T \to R \to U \to W$, and the value order is increasing. Note that different variables have different domains (e.g., the domain for X_3 is $\{0, 1\}$, the domain for O is $\{0, 1, \ldots, 9\}$, and the domain for F is $\{1, 2, \ldots, 9\}$).



(a) A cryptarithmetic problem. Each letter stands for a distinct digit; the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct, with the added restriction that no leading zeroes are allowed. (b) The constraint hypergraph for the cryptarithmetic problem, showing the *Alldiff* constraint (square box at the top) as well as the column addition constraints (four square boxes in the middle). The variables X_1, X_2, and X_3 represent the carry digits for the three columns.

Figure 3: Problem 3.

Solution.
First, we give the constraints:

$$\begin{aligned} Alldiff(F, T, U, W, R, O) \\ 2 \times O = R + 10 \times X_1 \\ 2 \times W + X_1 = U + 10 \times X_2 \\ 2 \times T + X_2 = O + 10 \times X_3 \\ X_3 = F_1 \end{aligned}$$

Then, because no leading zeroes are allowed, we have the following domains

$$\begin{aligned} O, R, W, U, T &\in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\ F &\in \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \\ X_1, X_2, X_3 &\in \{0, 1\} \end{aligned}$$

After that, we use *forward checking* to solve the problem. The assigned variables are put into a set S.

- Start state.

$$\begin{aligned} S &= \{\} \\ O, R, W, U, T &\in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\ F &\in \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \\ X_1, X_2, X_3 &\in \{0, 1\} \end{aligned}$$

- We assign X_3 with 0 and find that F has no possible values.

$$\begin{aligned} S &= \{X_3 : 0\} \\ O, R, W, U &\in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\ T &\in \{0, 1, 2, 3, 4\} \\ F &\in \emptyset \\ X_1, X_2 &\in \{0, 1\} \\ X_3 &\in \{0\} \end{aligned}$$

- Then,we backtrack and assign X_3 with 1.

$$\begin{aligned} S &= \{X_3 : 1\} \\ O, R, W, U &\in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\ T &\in \{5, 6, 7, 8, 9\} \\ X_1, X_2 &\in \{0, 1\} \\ F, X_3 &\in \{1\} \end{aligned}$$

- Next, we assign F with 1.

$$\begin{aligned} S &= \{X_3 : 1, F : 1\} \\ O, R, W, U &\in \{0, 2, 3, 4, 5, 6, 7, 8, 9\} \\ T &\in \{5, 6, 7, 8, 9\} \\ X_1, X_2 &\in \{0, 1\} \\ F, X_3 &\in \{1\} \end{aligned}$$

- After that, we assign X_2 with 0.

$$\begin{aligned} S &= \{X_3 : 1, F : 1, X_2 : 0\} \\ O &\in \{0, 2, 4, 6, 8\} \\ R &\in \{0, 2, 3, 4, 5, 6, 7, 8, 9\} \\ U &\in \{0, 4, 5, 6, 7, 8, 9\} \\ W &\in \{0, 2, 3, 4\} \\ T &\in \{5, 6, 7, 8, 9\} \\ X_1 &\in \{0, 1\} \\ X_2 &\in \{0\} \\ F, X_3 &\in \{1\} \end{aligned}$$

- After that, we assign X_1 with 0.

$$\begin{aligned} S &= \{X_3 : 1, F : 1, X_2 : 0, X_1 : 0\} \\ O &\in \{0, 2, 4\} \\ R &\in \{0, 4, 8\} \\ U &\in \{0, 4, 6, 8\} \\ W &\in \{0, 2, 3, 4\} \\ T &\in \{5, 6, 7, 8, 9\} \\ X_1, X_2 &\in \{0\} \\ F, X_3 &\in \{1\} \end{aligned}$$

- Then, we assign O with 0 and find that R has no possible values.

$$\begin{aligned} S &= \{X_3 : 1, F : 1, X_2 : 0, X_1 : 0, O : 0\} \\ R &\in \emptyset \\ U &\in \{4, 6, 8\} \\ W &\in \{2, 3, 4\} \\ T &\in \{5\} \\ O, X_1, X_2 &\in \{0\} \\ F, X_3 &\in \{1\} \end{aligned}$$

- We backtrack and assign O with 2.

$$\begin{aligned} S &= \{X_3 : 1, F : 1, X_2 : 0, X_1 : 0, O : 2\} \\ O &\in \{2\} \\ R &\in \{4\} \\ U &\in \{0, 4, 6, 8\} \\ W &\in \{0, 3, 4\} \\ T &\in \{6\} \\ X_1, X_2 &\in \{0\} \\ F, X_3 &\in \{1\} \end{aligned}$$

- Then, we assign T with 6.

$$\begin{aligned} S &= \{X_3 : 1, F : 1, X_2 : 0, X_1 : 0, O : 2, T : 6\} \\ O &\in \{2\} \\ R &\in \{4\} \\ U &\in \{0, 4, 8\} \\ W &\in \{0, 3, 4\} \\ T &\in \{6\} \\ X_1, X_2 &\in \{0\} \\ F, X_3 &\in \{1\} \end{aligned}$$

- Next, we assign R with 4.

$$\begin{aligned} S &= \{X_3 : 1, F : 1, X_2 : 0, X_1 : 0, O : 2, T : 6, R : 4\} \\ O &\in \{2\} \\ R &\in \{4\} \\ U &\in \{0, 8\} \\ W &\in \{0, 3\} \\ T &\in \{6\} \\ X_1, X_2 &\in \{0\} \\ F, X_3 &\in \{1\} \end{aligned}$$

- Then we assign U with 0 and find that W has no possible values.

$$\begin{aligned} S &= \{X_3 : 1, F : 1, X_2 : 0, X_1 : 0, O : 2, T : 6, R : 4, U : 0\} \\ O &\in \{2\} \\ R &\in \{4\} \\ U &\in \{0\} \\ W &\in \emptyset \\ T &\in \{6\} \\ X_1, X_2 &\in \{0\} \\ F, X_3 &\in \{1\} \end{aligned}$$

- We will find that if $U = 8$, W will also have no possible values.
- We backtrack again and assign O with 4.

$$\begin{aligned} S &= \{X_3 : 1, F : 1, X_2 : 0, X_1 : 0, O : 4\} \\ O &\in \{4\} \\ R &\in \{8\} \\ U &\in \{0, 6, 8\} \\ W &\in \{0, 2, 3\} \\ T &\in \{7\} \\ X_1, X_2 &\in \{0\} \\ F, X_3 &\in \{1\} \end{aligned}$$

- Then we assign T with 7.

$$\begin{aligned} S &= \{X_3 : 1, F : 1, X_2 : 0, X_1 : 0, O : 4, T : 7\} \\ O &\in \{4\} \\ R &\in \{8\} \\ U &\in \{0, 6, 8\} \\ W &\in \{0, 2, 3\} \\ T &\in \{7\} \\ X_1, X_2 &\in \{0\} \\ F, X_3 &\in \{1\} \end{aligned}$$

- Then we assign R with 8.

$$\begin{aligned} S &= \{X_3 : 1, F : 1, X_2 : 0, X_1 : 0, O : 4, T : 7, R : 8\} \\ O &\in \{4\} \\ R &\in \{8\} \\ U &\in \{0, 6\} \\ W &\in \{0, 2, 3\} \\ T &\in \{7\} \\ X_1, X_2 &\in \{0\} \\ F, X_3 &\in \{1\} \end{aligned}$$

- After that, we assign U with 0 and that makes us backtrack. At last, we assign U with 6.

$$\begin{aligned} S &= \{X_3 : 1, F : 1, X_2 : 0, X_1 : 0, O : 4, T : 7, R : 8, U : 6\} \\ O &\in \{4\} \\ R &\in \{8\} \\ U &\in \{6\} \\ W &\in \{3\} \\ T &\in \{7\} \\ X_1, X_2 &\in \{0\} \\ F, X_3 &\in \{1\} \end{aligned}$$

- Finally, we assign W with 3 and get an answer.

$$\begin{aligned} S &= \{X_3 : 1, F : 1, X_2 : 0, X_1 : 0, O : 4, T : 7, R : 8, U : 6, W : 3\} \\ O &\in \{4\} \\ R &\in \{8\} \\ T &\in \{7\} \\ U &\in \{6\} \\ W &\in \{3\} \\ X_1, X_2 &\in \{0\} \\ F, X_3 &\in \{1\} \end{aligned}$$

As a conclusion, the answer found by *forward checking* is $S = \{X_3 : 1, X_2 : 0, X_1 : 0, F : 1, O : 4, U : 6, R : 8, T : 7, W : 3\}$ which means "734 + 734 = 1468".

4. **AC-3**. Consider the following CSP:

   - Variables: A, B, C, D
   - Domain: $\{1, 2, 3\}$
   - Constraints: $A \neq B, B > C, C < D$

   Solve the problem using the strategy of backtracking search with AC-3 algorithm. Give the problem-solving steps, specifying each assignment when backtracking and the consequence of each pop operation in AC-3 (i.e., what values you cross off and which arcs you push to the queue).

   Suppose the value order is descending and the variable order is alphabetical, which both queue initialization and neighbor consideration follow (i.e., the queue is initialized as $A \to B, B \to A, \dots$).

   Solution.
   As what we have done in problem 3, the assigned variables are put into a set S. We denote the queue used in AC-3 as Q.

   - The start state.

     $$S = \{\}$$
     $$A, B, C, D \in \{1, 2, 3\}$$

   - First, we assign A with 3 and put $B \to A$ into the Q.

     $$S = \{A : 3\}$$
     $$A \in \{3\}$$
     $$B, C, D \in \{1, 2, 3\}$$
     $$Q = \{B \to A\}$$

   - Then, pop $B \to A$, modify the domain of B and put $A \to B, C \to B$ into Q.

     $$S = \{A : 3\}$$
     $$A \in \{3\}$$
     $$B \in \{1, 2\}$$
     $$C, D \in \{1, 2, 3\}$$
     $$Q = \{A \to B, C \to B\}$$

   - Then, pop $A \to B$ and nothing happens. After that, Pop $C \to B$, modify the domain of C and put $B \to C, D \to C$ into Q.

     $$S = \{A : 3\}$$
     $$A \in \{3\}$$
     $$B \in \{1, 2\}$$
     $$C \in \{1\}$$
     $$D \in \{1, 2, 3\}$$
     $$Q = \{B \to C, D \to C\}$$

   - Next, pop $B \to C$, modify the domain of B and put $A \to B, C \to B$ into Q. After that, Pop $D \to C$, modify the domain of D and put $C \to D$ into Q.

     $$S = \{A : 3\}$$
     $$A \in \{3\}$$
     $$B \in \{2\}$$
     $$C \in \{1\}$$
     $$D \in \{2, 3\}$$
     $$Q = \{A \to B, C \to B, C \to D\}$$

   - Then, pop $A \to B$, pop $C \to B$, pop $C \to D$ and nothing happens.
   - Next, we assign B with 2 and put $A \to B, C \to B$ into the Q.

     $$S = \{A : 3, B : 2\}$$
     $$A \in \{3\}$$
     $$B \in \{2\}$$
     $$C \in \{1\}$$
     $$D \in \{2, 3\}$$
     $$Q = \{A \to B, C \to B\}$$

- Then, pop $A \to B$, pop $C \to B$ and nothing happens.
- Then, we assign C with 1 and put $B \to C, D \to C$ into the Q.

$$S = \{A : 3, B : 2, C : 1\}$$
$$A \in \{3\}$$
$$B \in \{2\}$$
$$C \in \{1\}$$
$$D \in \{2, 3\}$$
$$Q = \{B \to C, D \to C\}$$

- Next, pop $B \to C$, pop $D \to C$ and nothing happens.
- After that, we assign D with 3 and put $C \to D$ into the Q.

$$S = \{A : 3, B : 2, C : 1, D : 3\}$$
$$A \in \{3\}$$
$$B \in \{2\}$$
$$C \in \{1\}$$
$$D \in \{3\}$$
$$Q = \{C \to D\}$$

- Finally, pop $C \to D$ and nothing happens.
- All variables have been assigned.

In conclusion, the answer found by *backtracking search with AC-3 algorithm* is $A = 3, B = 2, C = 1, D = 3$.

5. **K-consistency & tree structure**. Consider the following three questions:

   (a) Give a concrete CSP example to show that k-consistency does not imply $(k+1)$-consistency for some $k \geq 2$.

   (b) Give a concrete CSP example to show that k-consistency does not imply $(k-1)$-consistency for some $k \geq 3$.

   (c) Why graphs with cycles can not be applied with the algorithm for tree-structured CSPs (introduced in Page 77-83, Lecture 4)? What step in the analysis fails? Why this step holds for trees? Give an example to explain the failure.

Solution.

(a) Let's think about the following situation in a map coloring problem. It's easy to verify that for every possible arc, for every color in the tail, there's some color in head which could be assigned without violating a constraint. That means this CSP satisfies 2-consistency (arc consistency). However, for one consistent assignment to two nodes (assign A with red and assign B with blue), there's no color in C's domain that could be assigned without violating a constraint, which mean the 3-consistency doesn't hold.

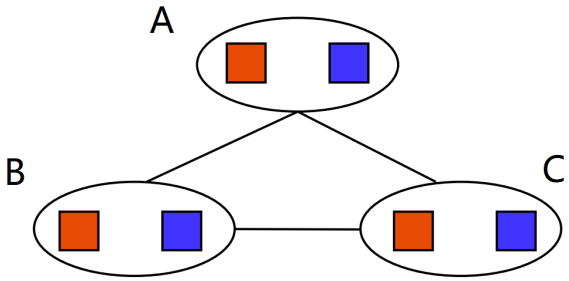


Figure 4: A Concrete CSP Example for Problem 5(a).

(b) Here's another situation in the map coloring problem. We can verify that for every consistent assignment to 2 nodes, there is a possible color in the third node's domain that could be assigned without violating a constraint. So, the CSP satisfies 3-consistency. However, if we assign B with red, there will be no color in A that could be assigned without violating a constraint. Thus, the CSP doesn't satisfy 2-consistency.

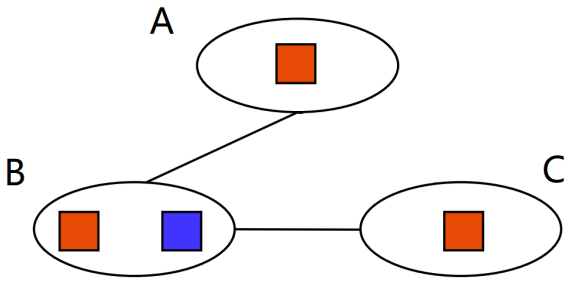


Figure 5: A Concrete CSP Example for Problem 5(b).

(c) In graphs with cycles, we can choose an arbitrary order and draw 'forward' arcs. What is different from trees is that here every node may have more than one parents. The reason why the algorithm for tree-structured CSPs can't be applied is that we may get fail when doing 'assign forward'. Because every node may have more than one parents, it may be impossible to make sure the constraints involving the current node and any parent node are satisfied. But in tree-structured CSPs, every node has at most one parents and the 'remove backward' promises the consistency of $Parent(node) \rightarrow node$, which make sure that we won't backtrack.

The example can also be Figure 4. Suppose the order is $A \rightarrow B \rightarrow C$. When we do 'remove backward', we have nothing to do. But when we do 'assign forward', we'll get stuck when facing C who has two parents.