FAI LAB 6
Constraint satisfaction problems

Paolo Morettin

2024-25

Constraint Satisfaction Problems

Constraint Satisfaction Problems

- **Factored** state representation
  - variables $X_1, ..., X_n$
  - with domains $dom(X_i) = \{v_1^{(i)}, ..., v_{k_i}^{(i)}\}$
  - and constraints $C_1, ..., C_m$

Constraint Satisfaction Problems

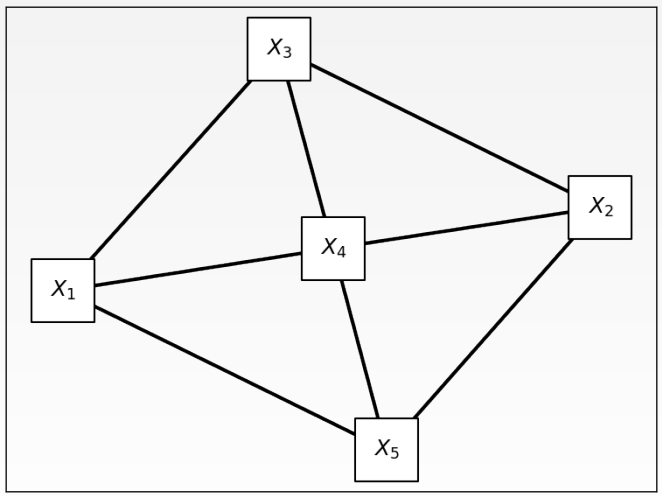
- **Factored** state representation
  - variables $X_1, ..., X_n$
  - with domains $dom(X_i) = \{v_1^{(i)}, ..., v_{k_i}^{(i)}\}$
  - and constraints $C_1, ..., C_m$
- **State**: an assignment to some $(X_i = v)$ s.t. $v \in dom(X_i)$
  - they can be [*partial*/*total*] [*consistent*/*inconsistent*]

Constraint Satisfaction Problems

- **Factored** state representation
  - variables $X_1, ..., X_n$
  - with domains $dom(X_i) = \{v_1^{(i)}, ..., v_{k_i}^{(i)}\}$
  - and constraints $C_1, ..., C_m$
- **State**: an assignment to some $(X_i = v)$ s.t. $v \in dom(X_i)$
  - they can be [*partial*/*total*] [*consistent*/*inconsistent*]
- **Solution**: a total **and** consistent assignment

Constraint Satisfaction Problems

- **Factored** state representation
  - variables $X_1, ..., X_n$
  - with domains $dom(X_i) = \{v_1^{(i)}, ..., v_{k_i}^{(i)}\}$
  - and constraints $C_1, ..., C_m$
- **State**: an assignment to some $(X_i = v)$ s.t. $v \in dom(X_i)$
  - they can be [*partial*/*total*] [*consistent*/*inconsistent*]
- **Solution**: a total **and** consistent assignment
- **CSPs as graphs**: nodes are vars, edges are (binary) constraints



Solving CSPs

- **What is new?** (w.r.t. classical/local search)
  There, we could only move from complete state to complete state

Solving CSPs

- **What is new?** (w.r.t. classical/local search)
  There, we could only move from complete state to complete state

- Now we *incrementally* build it. **How?**

Solving CSPs

- **What is new?** (w.r.t. classical/local search)
  There, we could only move from complete state to complete state

- Now we *incrementally* build it. **How?**

  - **Constraint propagation** (aka **inference**): removing inconsistencies in $dom(X_i)$

Solving CSPs

- **What is new?** (w.r.t. classical/local search)
  There, we could only move from complete state to complete state

- Now we *incrementally* build it. **How?**

  - **Constraint propagation** (aka **inference**): removing inconsistencies in $dom(X_i)$

  - **Search**: assigning X_i values from their domain

Solving CSPs

- **What is new?** (w.r.t. classical/local search)
  There, we could only move from complete state to complete state
- Now we *incrementally* build it. **How?**
  - **Constraint propagation** (aka **inference**): removing inconsistencies in $dom(X_i)$
  - **Search**: assigning X_i values from their domain
- Propagation and search are usually interleaved

Constraint propagation

Constraint propagation

- Node consistency (1-consistency): preprocessing

Constraint propagation

- Node consistency (1-consistency): preprocessing
- **Arc consistency** (2-consistency): forward checking or AC3

$$\forall v_i \in dom(X_i) \quad . \quad \exists v_j \in dom(X_j) \text{ s.t. } C_{i,j} \text{ is satisfied}$$

Constraint propagation

- Node consistency (1-consistency): preprocessing
- **Arc consistency** (2-consistency): forward checking or AC3

$$\forall v_i \in dom(X_i) \quad . \quad \exists v_j \in dom(X_j) \text{ s.t. } C_{i,j} \text{ is satisfied}$$

- There is more: path consistency (3-consistency), k-consistency

Constraint propagation: forward checking vs. AC3

Constraint propagation: forward checking vs. AC3

- FC: only propagates to neighbors of the last assigned X_i

Constraint propagation: forward checking vs. AC3

- FC: only propagates to neighbors of the last assigned X_i

- AC3: full propagation

  each time an arc is made consistent, (re)check its neighbors

function AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise
  inputs: *csp*, a binary CSP with components $(X,\ D,\ C)$
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    $(X_i,\ X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REVISE(*csp*, $X_i,\ X_j$) **then**
      if size of $D_i\ =\ 0$ **then return** *false*
      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**
        add $(X_k,\ X_i)$ to *queue*
  return *true*

---

function REVISE(*csp*, $X_i,\ X_j$) **returns** true iff we revise the domain of X_i
  revised $\leftarrow$ *false*
  for each x **in** D_i **do**
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**
      delete x from D_i
      revised $\leftarrow$ *true*
  return *revised*

Backtracking Search

Like DFS but:

Backtracking Search

Like DFS but:

- reasons on **partial assignments** (assigns 1 variable at the time)

Backtracking Search

Like DFS but:

- reasons on **partial assignments** (assigns 1 variable at the time)
- checks for **inconsistencies** along the way

Backtracking Search

Like DFS but:

- reasons on **partial assignments** (assigns 1 variable at the time)
- checks for **inconsistencies** along the way

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment then
          add {var = value} to assignment
          inferences ← INFERENCE(csp, var, value)
          if inferences ≠ failure then
              add inferences to assignment
              result ← BACKTRACK(assignment, csp)
              if result ≠ failure then
                  return result
          remove {var = value} and inferences from assignment
  return failure
```

Backtracking Search

Like DFS but:

- reasons on **partial assignments** (assigns 1 variable at the time)
- checks for **inconsistencies** along the way

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment then
          add {var = value} to assignment
          inferences ← INFERENCE(csp, var, value)
          if inferences ≠ failure then
              add inferences to assignment
              result ← BACKTRACK(assignment, csp)
              if result ≠ failure then
                  return result
          remove {var = value} and inferences from assignment
  return failure
```

Backtracking Search

`SELECT-UNASSIGNED-VARIABLE`

Two heuristics (*in this order*):

- **Minimum Remaining Values (MRV)** $argmin_X |dom(X)|$
- **Degree** $argmax_X |\{(X, n) \, \forall n \in UNeigh(X)\}|$

Backtracking Search

`SELECT-UNASSIGNED-VARIABLE`

Two heuristics (*in this order*):

- **Minimum Remaining Values (MRV)** $argmin_X |dom(X)|$
- **Degree** $argmax_X |\{(X, n) \, \forall n \in UNeigh(X)\}|$

`ORDER-DOMAIN-VALUES`

Least constraining value (LCS) heuristic

Pick $v \in dom(X)$ that has minimum impact on $\sum_{n \in UNeigh(X)} |dom(n)|$

Backtracking Search

`INFERENCE`

Two options:

- **Forward checking**
- **AC3** (init *queue* with arcs to *UNeigh*)

Backtracking Search

`INFERENCE`

Two options:

- **Forward checking**
- **AC3** (init *queue* with arcs to *UNeigh*)

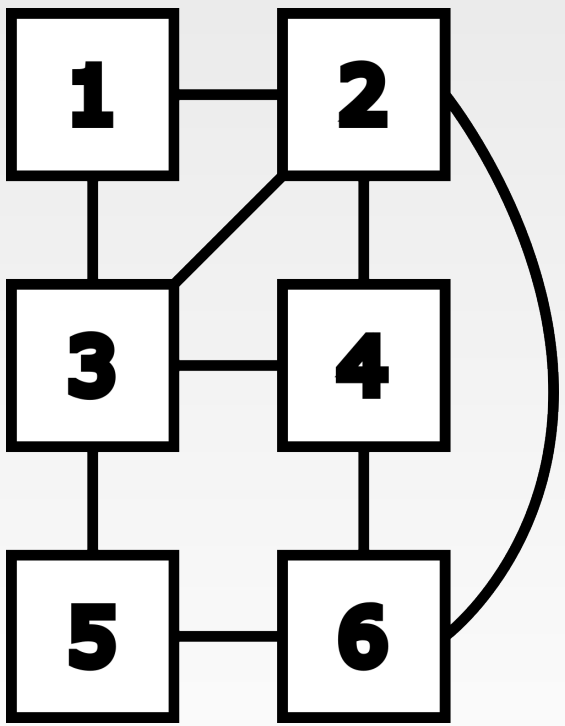
`BACKTRACK`

Standard Chronological Backtracking

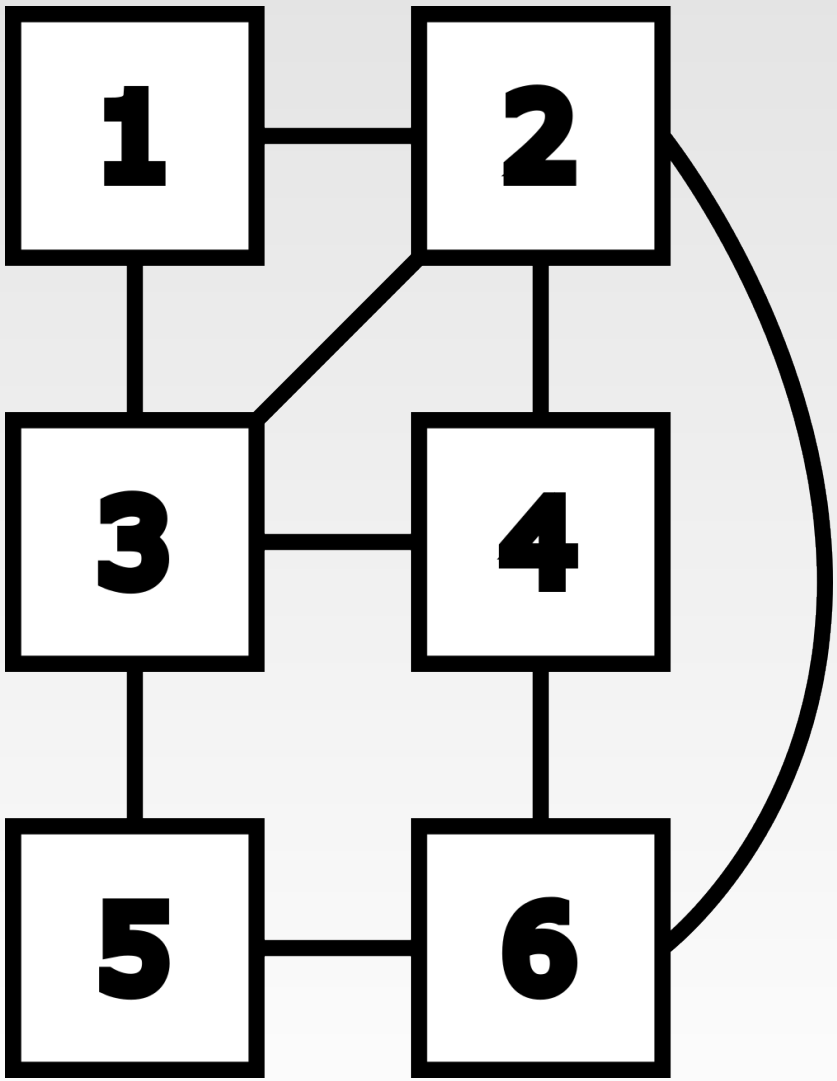
backtracks to the previous assignment

Graph coloring

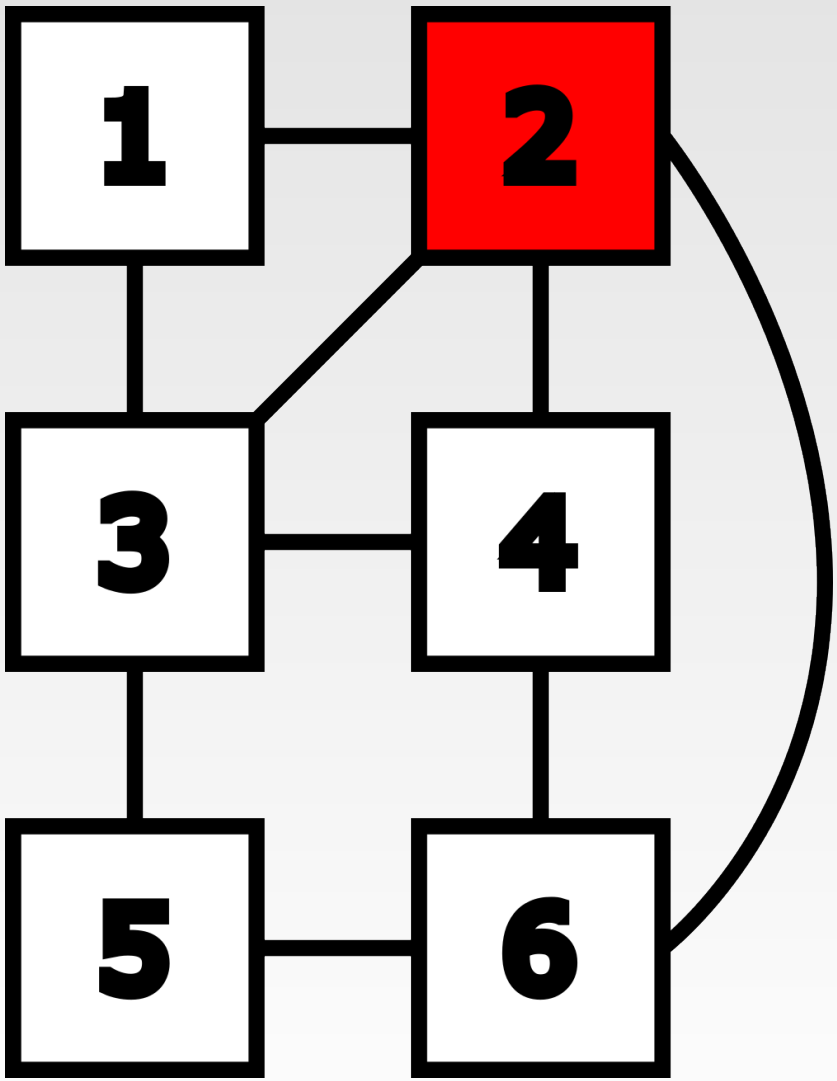
- $dom(X_i) = \{R, G, B\}$ (in this order)
- Edges represent $X_i \neq X_j$
- Multiple edge types are possible (e.g. $X_i = X_j$)



GC with backtracking search



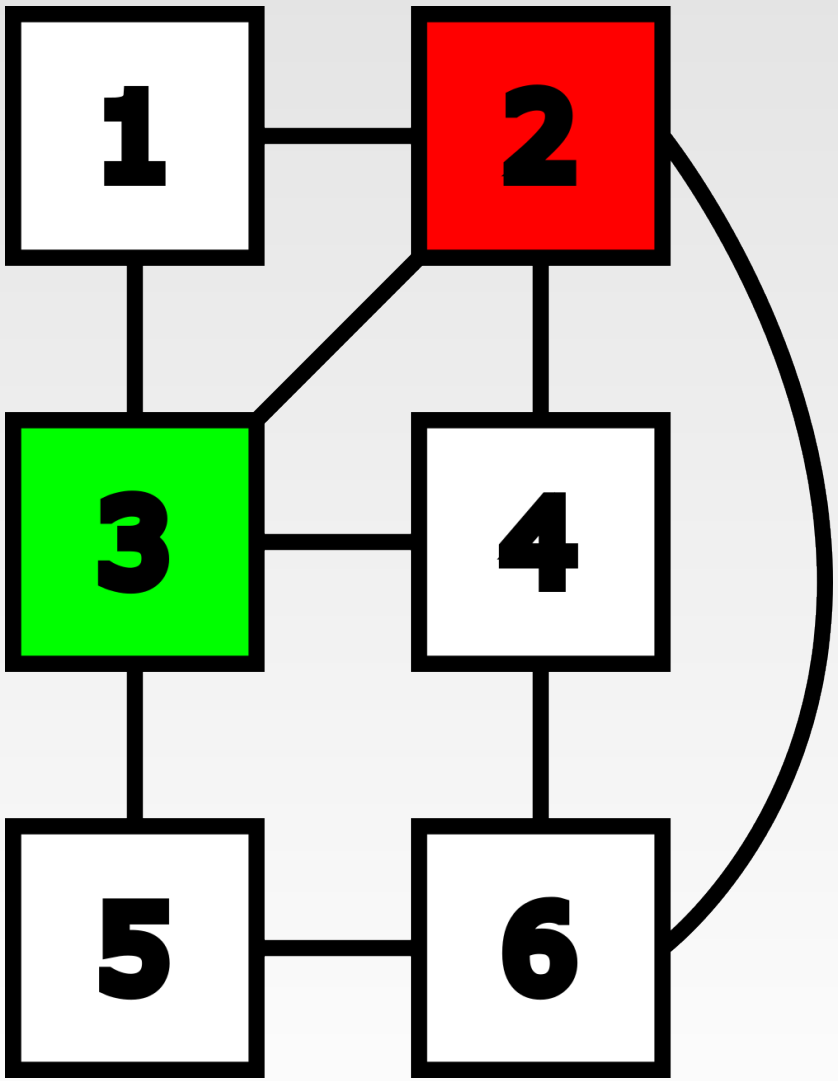
GC with backtracking search



$(2 = \mathrm{R})$

$\mathrm{dom}(1) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(3) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(4) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(6) = [\mathrm{G}, \mathrm{B}]$

GC with backtracking search



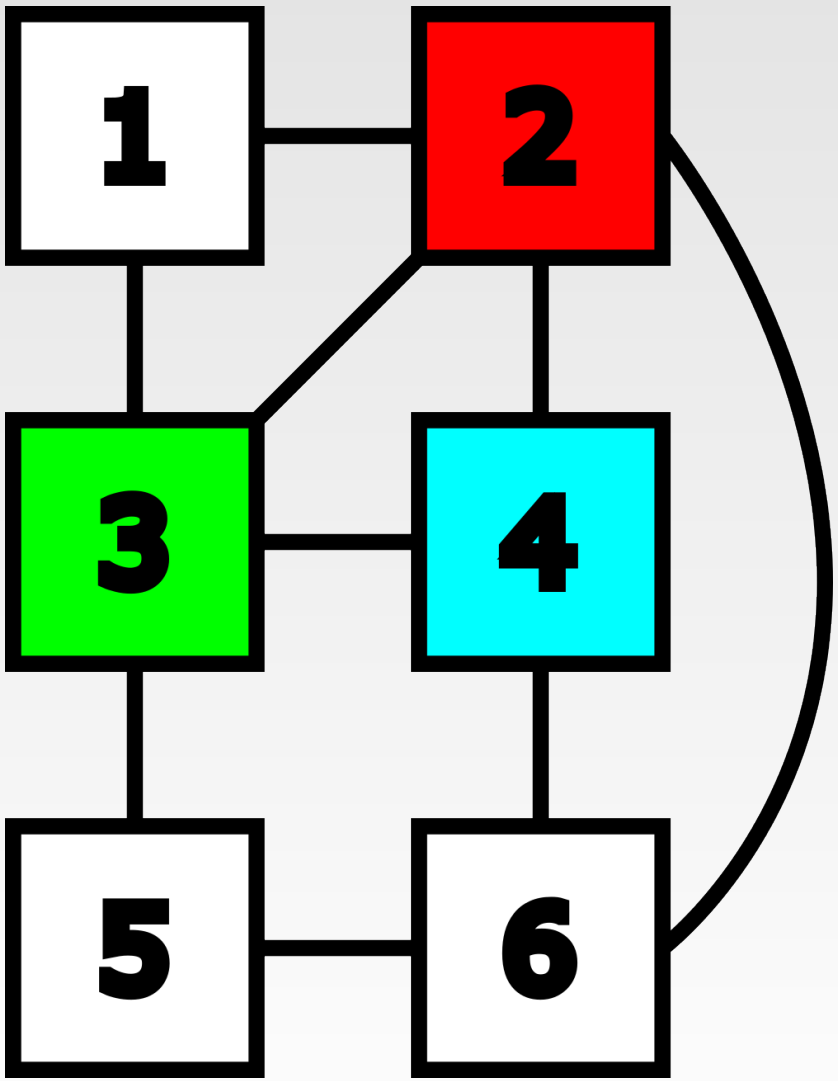
$(2 = \mathrm{R})$

$\mathrm{dom}(1) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(3) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(4) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(6) = [\mathrm{G}, \mathrm{B}]$

$(3 = \mathrm{G})$

$\mathrm{dom}(1) = [\mathrm{B}]$
$\mathrm{dom}(4) = [\mathrm{B}]$
$\mathrm{dom}(5) = [\mathrm{R}, \mathrm{B}]$

GC with backtracking search



$(2 = \mathrm{R})$

$\mathrm{dom}(1) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(3) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(4) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(6) = [\mathrm{G}, \mathrm{B}]$

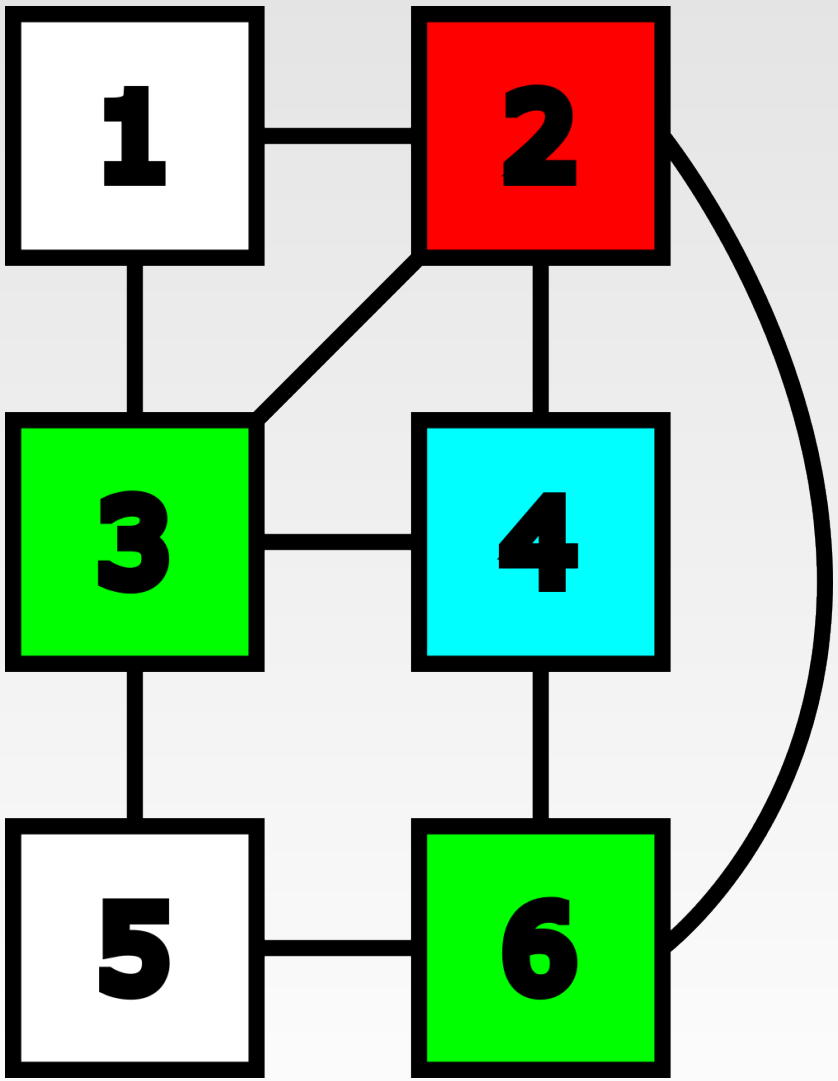
$(3 = \mathrm{G})$

$\mathrm{dom}(1) = [\mathrm{B}]$
$\mathrm{dom}(4) = [\mathrm{B}]$
$\mathrm{dom}(5) = [\mathrm{R}, \mathrm{B}]$

$(4 = \mathrm{B})$

$\mathrm{dom}(6) = [\mathrm{G}]$

GC with backtracking search



$(2 = \mathrm{R})$

$\mathrm{dom}(1) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(3) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(4) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(6) = [\mathrm{G}, \mathrm{B}]$

$(3 = \mathrm{G})$

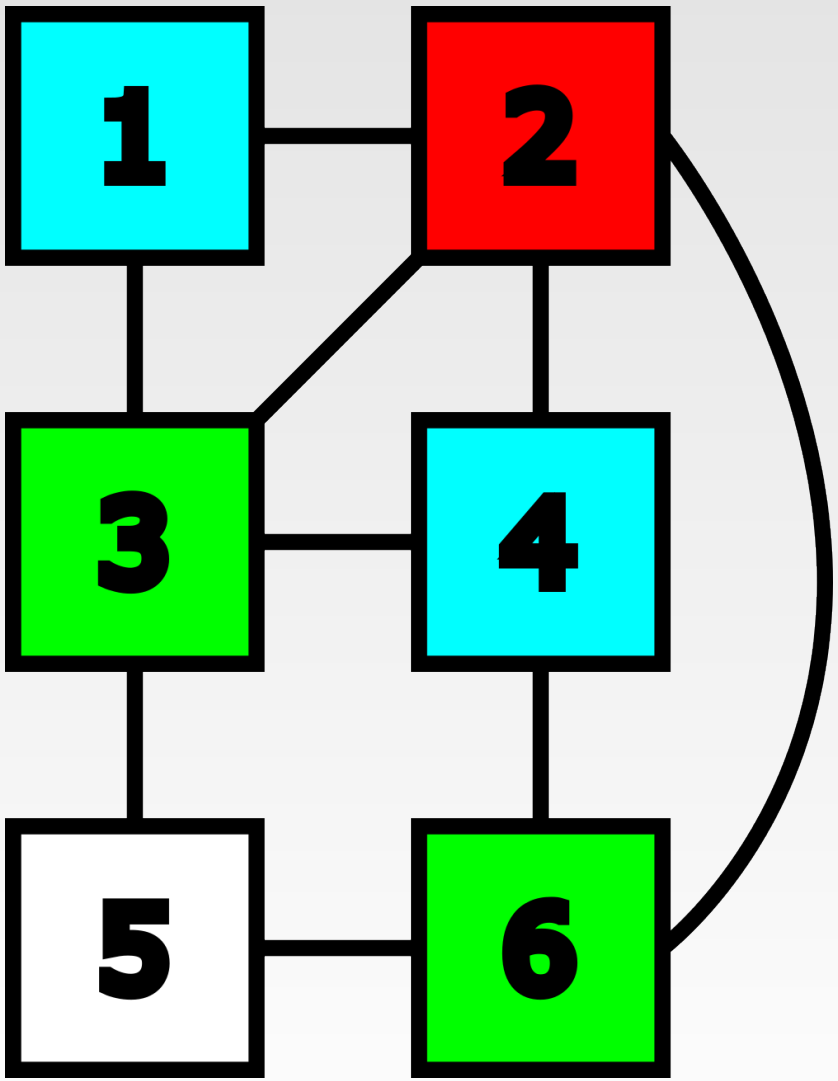
$\mathrm{dom}(1) = [\mathrm{B}]$
$\mathrm{dom}(4) = [\mathrm{B}]$
$\mathrm{dom}(5) = [\mathrm{R}, \mathrm{B}]$

$(4 = \mathrm{B})$

$\mathrm{dom}(6) = [\mathrm{G}]$

$(6 = \mathrm{G})$

GC with backtracking search



$(2 = \mathrm{R})$

$\mathrm{dom}(1) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(3) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(4) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(6) = [\mathrm{G}, \mathrm{B}]$

$(3 = \mathrm{G})$

$\mathrm{dom}(1) = [\mathrm{B}]$
$\mathrm{dom}(4) = [\mathrm{B}]$
$\mathrm{dom}(5) = [\mathrm{R}, \mathrm{B}]$

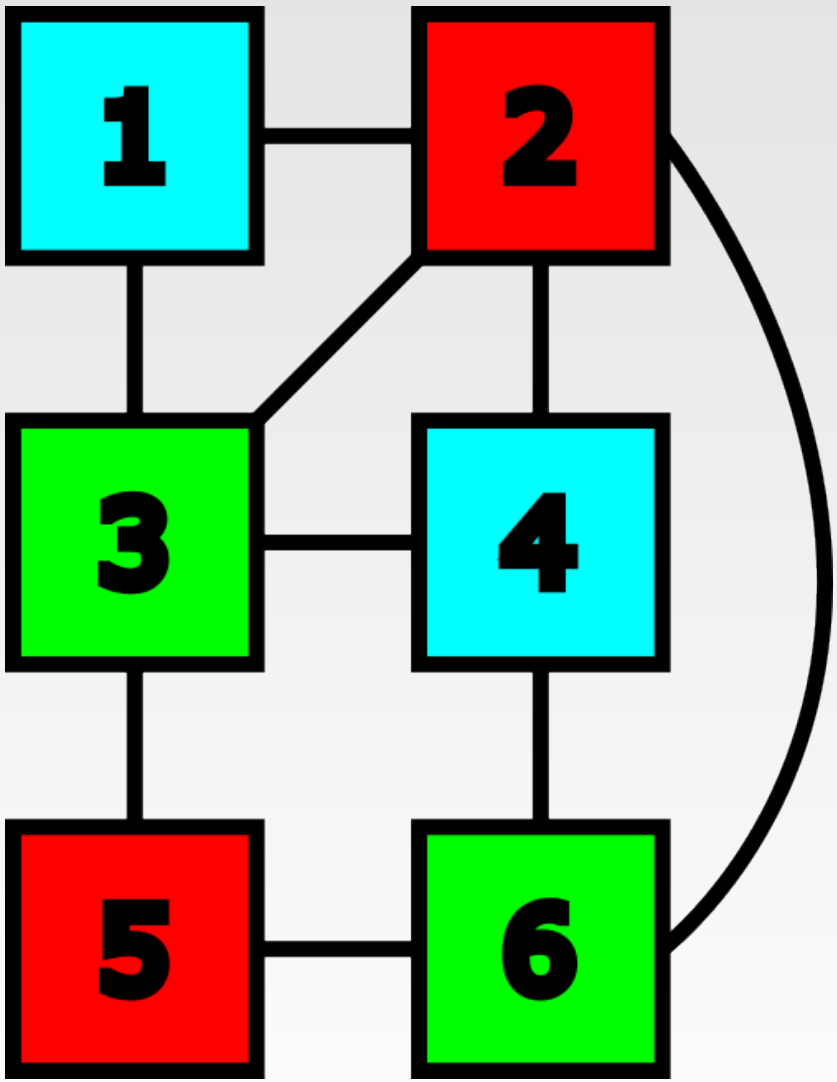
$(4 = \mathrm{B})$

$\mathrm{dom}(6) = [\mathrm{G}]$

$(6 = \mathrm{G})$

$(1 = \mathrm{B})$

GC with backtracking search



$(2 = \mathrm{R})$

$\mathrm{dom}(1) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(3) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(4) = [\mathrm{G}, \mathrm{B}]$
$\mathrm{dom}(6) = [\mathrm{G}, \mathrm{B}]$

$(3 = \mathrm{G})$

$\mathrm{dom}(1) = [\mathrm{B}]$
$\mathrm{dom}(4) = [\mathrm{B}]$
$\mathrm{dom}(5) = [\mathrm{R}, \mathrm{B}]$

$(4 = \mathrm{B})$

$\mathrm{dom}(6) = [\mathrm{G}]$

$(6 = \mathrm{G})$

$(1 = \mathrm{B})$

$(5 = \mathrm{R})$

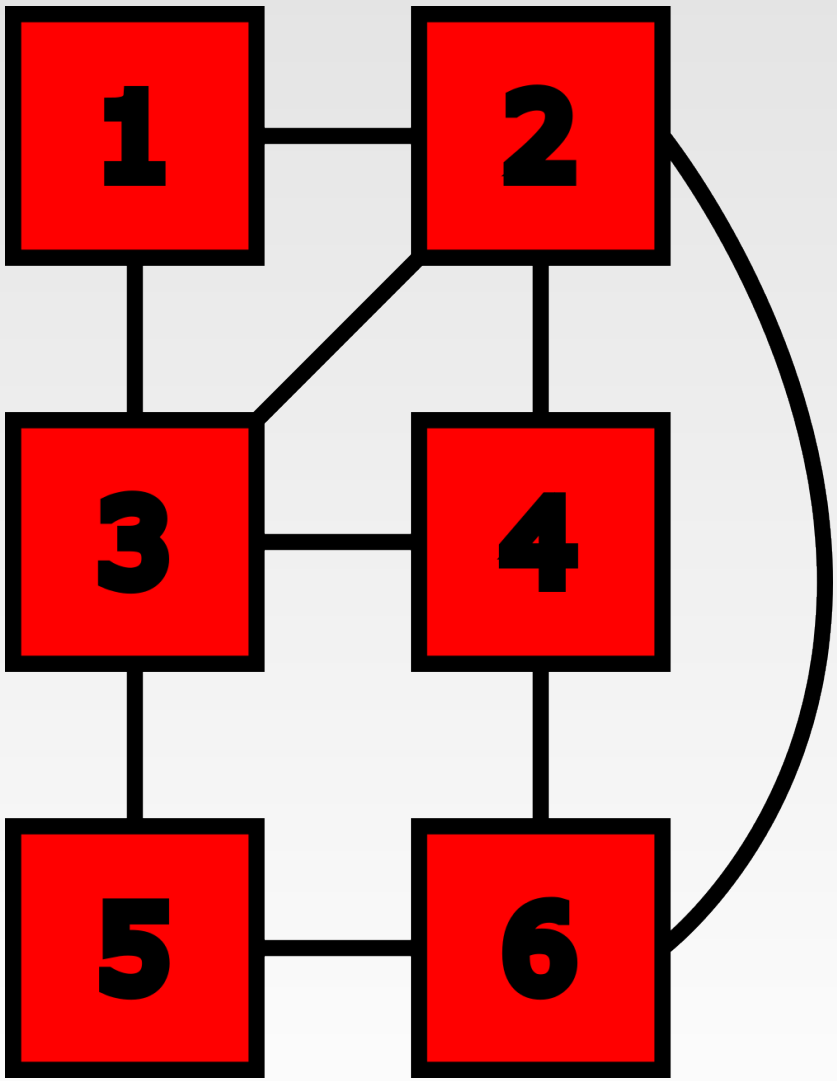
Local Search with CSPs

Hill-climbing + MCH:

function MIN-CONFLICTS(csp, max_steps) **returns** a solution or failure
  inputs: csp, a constraint satisfaction problem
    max_steps, the number of steps allowed before giving up

  $current \leftarrow$ an initial complete assignment for csp
  for i = 1 to max_steps **do**
    if $current$ is a solution for csp **then return** $current$
    $var \leftarrow$ a randomly chosen conflicted variable from csp.VARIABLES
    $value \leftarrow$ the value v for var that minimizes CONFLICTS($var, v, current, csp$)
    set $var = value$ in $current$
  return $failure$

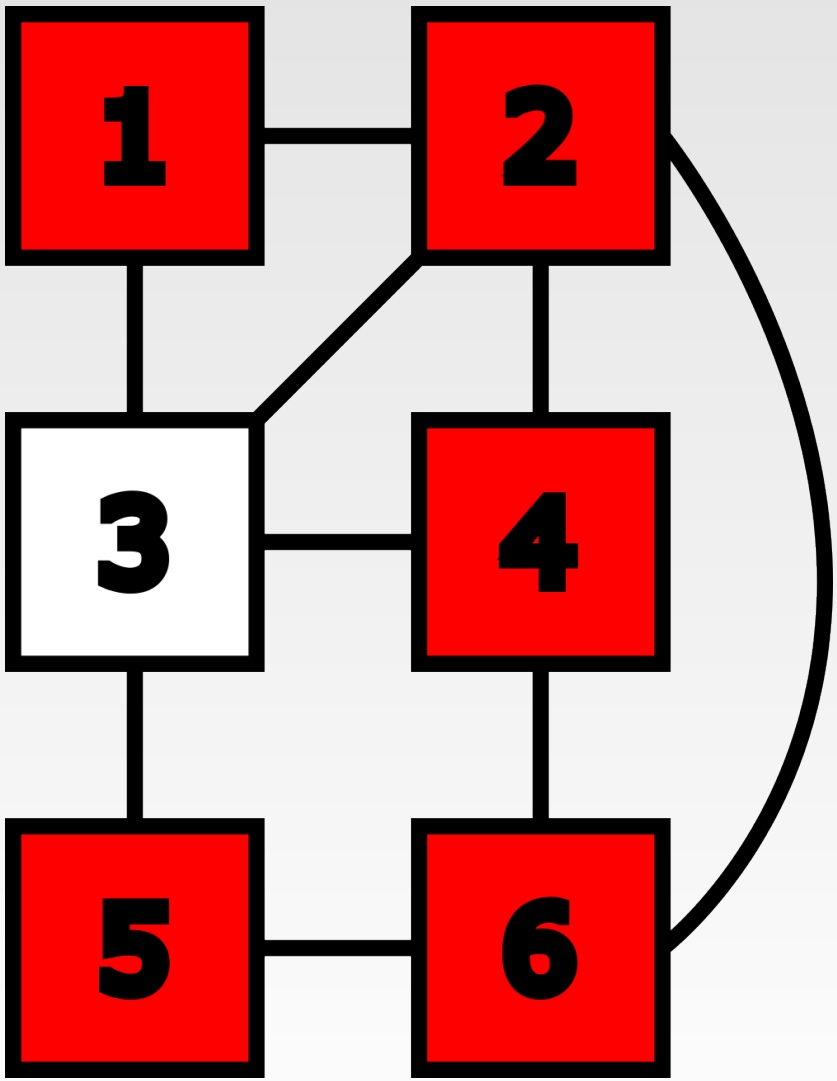
GC with HC + MCH search



conflicts = [1, 2, 3, 4, 5, 6]

choices = [0.44, 0.81, 0.76, 0.88]

GC with HC + MCH search



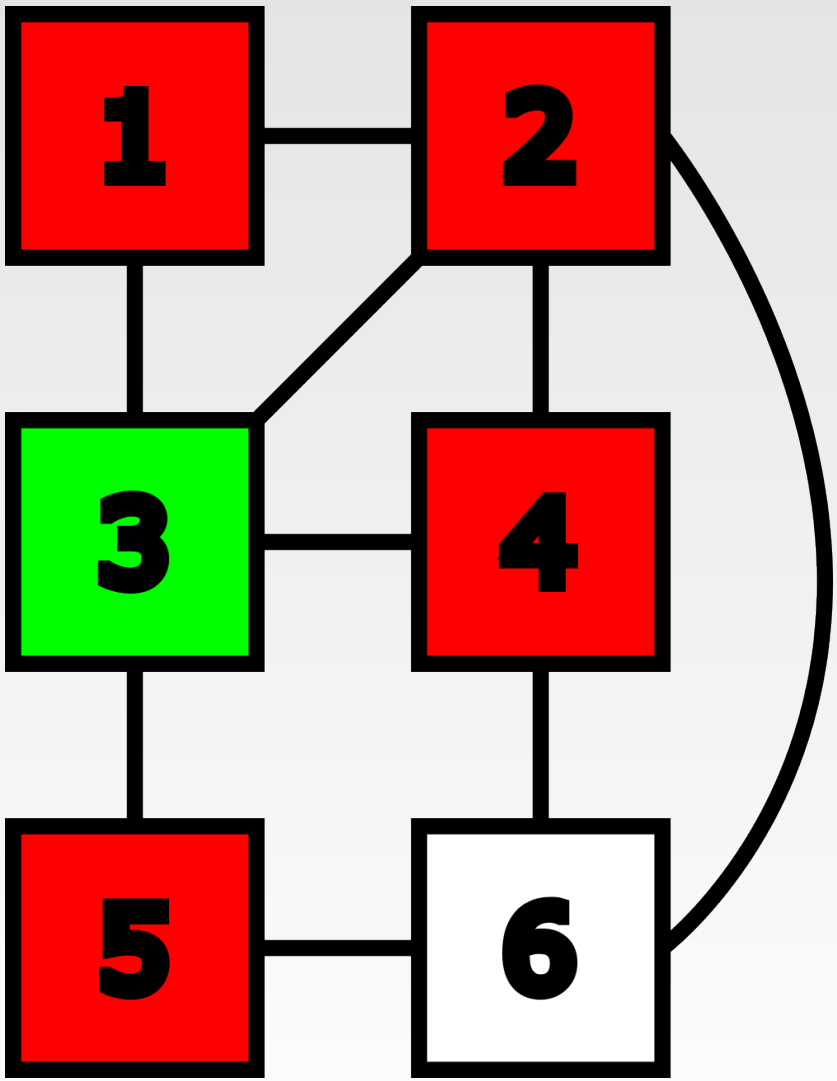
conflicts = [1, 2, **3**, 4, 5, 6]

choices = [**0.44**, 0.81, 0.76, 0.88]

conflicts:

(R $\rightarrow$ 6)
(G $\rightarrow$ 5)
(B $\rightarrow$ 5)

GC with HC + MCH search



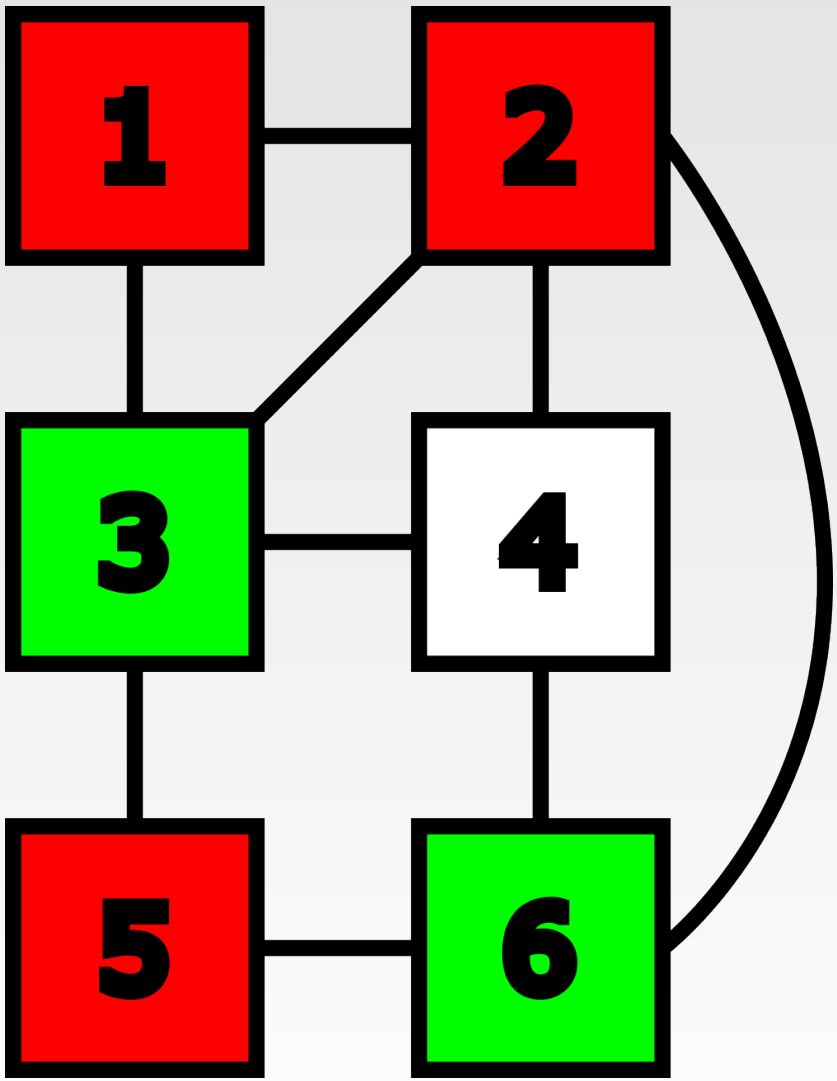
conflicts = [1, 2, 4, 5, **6**]

choices = [0.44, **0.81**, 0.76, 0.88]

conflicts:

(R $\rightarrow$ 5)
(G $\rightarrow$ 3)
(B $\rightarrow$ 3)

GC with HC + MCH search



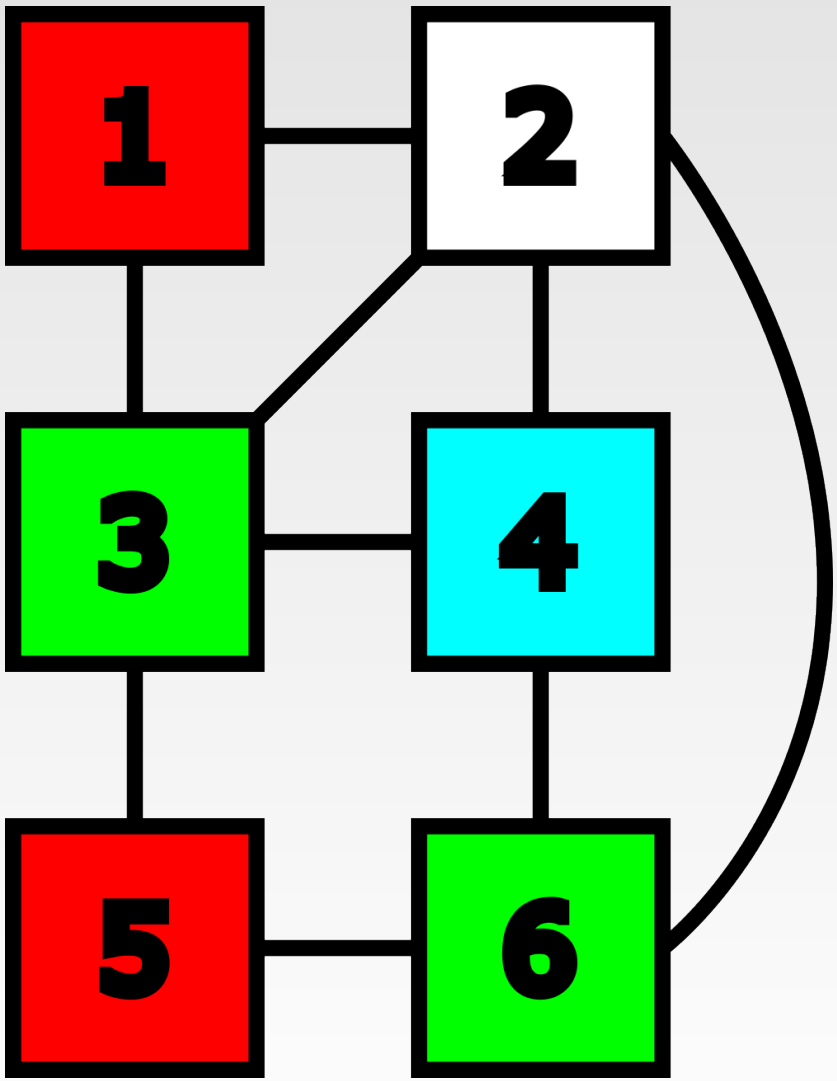
conflicts = [1, 2, **4**]

choices = [0.44, 0.81, **0.76**, 0.88]

conflicts:

(R $\rightarrow$ 3)
(G $\rightarrow$ 5)
(B $\rightarrow$ 2)

GC with HC + MCH search



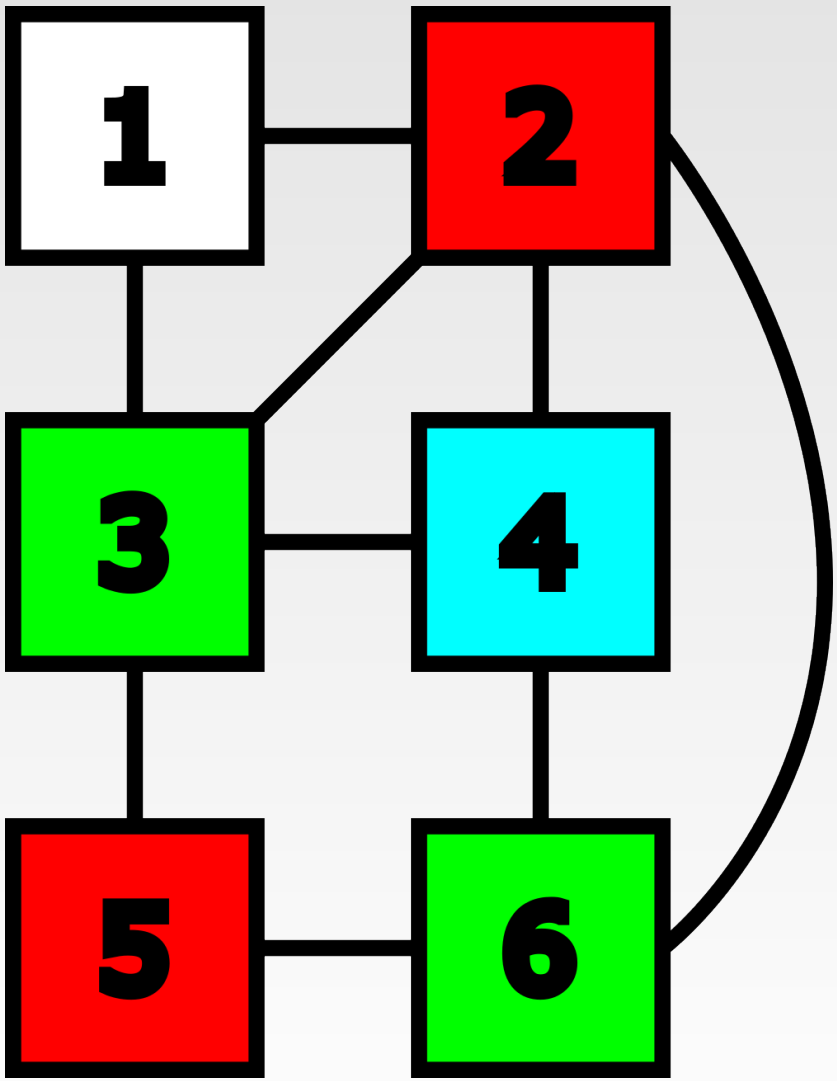
conflicts = [1, **2**]

choices = [0.44, 0.81, 0.76 **0.88**]

conflicts:

(R → 2)
(G → 3)
(B → 2)

GC with HC + MCH search



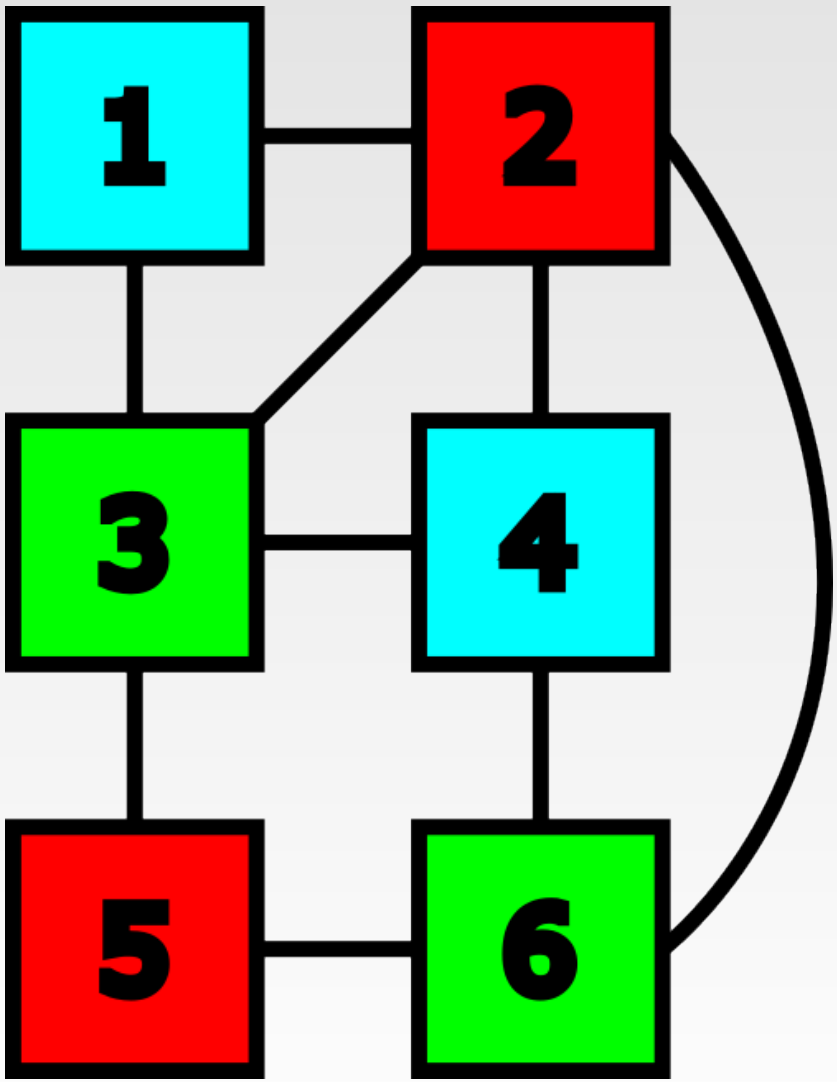
conflicts = [**1**, 2]

choices = [**0.44**, 0.81, 0.76 0.88]

conflicts:

(R $\rightarrow$ 2)
(G $\rightarrow$ 2)
(B $\rightarrow$ 0)

GC with HC + MCH search



conflicts = []

choices = [0.44, 0.81, 0.76 0.88]