CHAPTER 3, SECTION 7 AND CHAPTER 4, SECTION 4.4

Constraint Satisfaction Problems
Outline

- Heuristics for CSPs
- Forward checking
- Backtracking
- General search applied to CSPs
- CSP examples
than standard search algorithms
Allows useful general-purpose algorithms with more power

Simple example of a formal representation language

 allowable combinations of values for subsets of variables

goal test is a set of constraints specifying

state is defined by variables \( y \) with values from domain \( \mathcal{D} \)

\begin{align*}
\text{CSP:} & \quad \text{that supports goal test, eval, successor} \\
\text{state is a "black box"—any old data structure} \\
\text{Standard search problem:} & \quad \text{Constraint satisfaction problems (CSPs)}
\end{align*}
Example: 4-Queens as a CSP

\[(2,3) (4,1) (1,3) (4,2) (1,4) (3,1) (2,4) (3,2) \]

Valued for \( [q_1, q_2] \)

Translate each constraint into set of allowable values for its variables

\[ \begin{align*}
q_1 & \in \{1, 2, 3, 4\} \\
q_2 & \in \{1, 2, 3, 4\}
\end{align*} \]

Constraints:

\( q_1 - q_2 \neq |q_1 - q_2| \) (cannot be in same row)

\( q_1 \neq q_2 \) (or same diagonal)

Domains: \( D(q_1) = \{1, 2, 3, 4\} \)

Variables: \( q_1, q_2, q_3, q_4 \)

Assume one queen in each column. Which row does each one go in?
Constraint Graph: nodes are variables, arcs show constraints.

Binary CSP: each constraint relates at most two variables.
Example: Cryptarithmetric

\[
\begin{array}{c}
\text{SEND} \\
+ \text{MORE} \\
\hline
\text{MONEY}
\end{array}
\]

\[
\begin{aligned}
D & \neq 0, S & \neq 0, M & \neq 0, D & \neq E, E & \neq N, E & \neq L, E & \neq R, E & \neq E, \text{ etc.}
\end{aligned}
\]

\[
\begin{aligned}
\text{Constraints:} \\
0 \neq S, 0 \neq M, 0 \neq D
\end{aligned}
\]

\[
\begin{aligned}
\text{Domains:} \\
D \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
E \in \{0 \ldots 9\}
\end{aligned}
\]

\[
\begin{aligned}
\text{Variables:} \\
S \in \{0 \ldots 9\} \\
M \in \{0 \ldots 9\} \\
N \in \{0 \ldots 9\} \\
O \in \{0 \ldots 9\} \\
R \in \{0 \ldots 9\}
\end{aligned}
\]
Example: Map Coloring

Constraint Graph:

\[ C_1 \neq C_2, C_1 \neq C_3, \text{ etc.} \]

Domains

\{ \text{Red, Blue, Green} \}

Variables

Color a map so that no adjacent countries have the same color.
Notice that many real-world problems involve real-valued variables.

Floorplanning
Factory scheduling
Transportation scheduling
Spreadsheet
Hardware configuration

e.g., which class is offered when and where?

Timetabling problems
Assignment problems

Real-World CSPs
Notice that this is the same for all CSPs!

Goal: Test: all variables assigned, no constraints violated

Operators: assign a value to an unassigned variable

Initial state: all variables unassigned

States are defined by the values assigned so far

Let's start with the straightforward, dumb approach, then fix it.

Applying standard search
Implicitly by a function that tests for satisfaction of the constraint
explicitly as sets of allowable values, or
Constraints can be represented

VALUE, current value (if any)
DOMAIN, a list of possible values

component NAME, for i/o purposes
datatype CSP-VAR

ASSIGNED, a list of variables that have values
UNASSIGNED, a list of variables not yet assigned
datatype CSP-STATE

Each variable has a domain and a current value
CSP state keeps track of which variables have values so far

Implementation
Standard search applied to map-coloring
Adding assignments cannot correct a violated constraint

1) Order of assignment is irrelevant, hence many paths are equivalent.

2) This can be improved dramatically by noting the following:

\[ \overset{\text{Branching factor}}{i} \overset{q}{=} \overset{\text{Search algorithm to use}}{i} \overset{\text{Depth of solution state}}{i} \overset{p}{=} \overset{\text{Max. depth of space}}{m} \overset{?}{=} \]

Complexity of the dumb approach
Adding assignments cannot correct a violated constraint
Order of assignment is irrelevant so many paths are equivalent

This can be improved dramatically by noting the following:

- Branching factor \( \geq q \) (at top of tree)
- Search algorithm to use? depth-first
- Depth of solution state \( p \) (all vars assigned)
- Max. depth of space \( m \) (number of variables)

**Summary of the dumb approach**
Can solve n-queens for n ≤ 15

Backtracking search is the basic uninformed algorithm for CSPs

or 2) check constraints are satisfied before expanding a state
    are allowed, given the values already assigned
    modify SUCCESSORS to assign only values that
The constraint violation check can be implemented in two ways:

2) check for constraint violations
(can be done in the SUCCESSORS function)

1) fix the order of assignment, but
   use depth-first search, but

Backtracking search
Can solve $n$-queens up to $n \approx 30$

Simplified map-coloring example:

Terminate search when any variable has no legal values

Idea: Keep track of remaining legal values for unassigned variables
Can solve \( n \)-queens for \( n \approx 1000 \)

\[ \text{Given } C_1 = \text{Red}, C_2 = \text{Green}, \text{ what next?} \]
\[ \text{Given } C_1 = \text{Red}, C_2 = \text{Green}, \text{ choose } C_3 = \text{?} \]

which variable to assign next

which value to choose for each variable

More intelligent decisions on

Heuristics for CSPs
Can solve $n$-queens for $n \approx 1000$

$C_3$ is most-constrained variable
$C_3 = \text{Red}$, $C_2 = \text{Green}$, what next?

$C_3 = \text{Green}$: least-constrained value
Given $C_1 = \text{Red}$, $C_2 = \text{Green}$, choose $C_3 = \underline{?}$

Which variable to assign next
Which value to choose for each variable

More intelligent decisions on

Heuristics for CSPs
Iterative algorithms for CSPs
Example: 4 Queens

Goal Test: no attacks

Operators: move queen in column

States: 4 queens in 4 columns (4^4 = 256 states)

Evaluation function: \( h(u) = \text{number of attacks} \)

\[ h = 0 \quad h = 2 \quad h = 5 \]
The same appears to be true for any randomly-generated CSP
for arbitrary n with high probability (e.g., n = 10,000,000)
given random initial state, can solve n-queens in almost constant time

Performance of min-conflicts
complexity of reasoning.

An important example of the relation between syntactic restrictions and

This property also applies to logical and probabilistic reasoning:

\[ (\mathbb{u} | d) \mathcal{O} (\mathbb{u} | d) \mathcal{O} \]

Compare to general CSPs, where worst-case time is \( O(\sqrt{|d|}) \) time

in time

Theorem: if the constraint graph has no loops, the CSP can be solved

Tree-structured CSPs
Filtering example:

Remove values of \( \Lambda \) that are inconsistent with ALL values of \( \Lambda \).

\[ \text{FILTER}(\Lambda, \Lambda) \]

Basic step is called filtering.

Algorithm for tree-structured CSP
Algorithm cont'd.
Tree-structured CSPs can always be solved very efficiently. Iterative min-conflicts is usually effective in practice. Variable ordering and value selection heuristics help significantly. Forward checking prevents assignments that guarantee later failure.

2) only legal successors
1) fixed variable order

Backtracking = depth-first search with

goal test defined by constraints on variable values
states defined by values of a fixed set of variables

CSPs are a special kind of problem.

Summary