
Dietter Fox

Sections 3.7 and 4.4, Exercise 6.15, Weld paper on AI planning

Constraint Satisfaction Problems

Introduction to Artificial Intelligence
Planning as satisfiability

Heuristics for CSPs

Forward checking

Backtracking

General search applied to CSPs

CSP examples

Outline
Standard search problem: state is a "black box"—any old data structure that supports goal test, eval, successor.

CSP: Constraint satisfaction problems (CSPs)
Assume one queen in each column. Which row does each one go in?

Example: 4-Queens as a CSP
Constraint graph: nodes are variables, arcs show constraints

Binary CSP: each constraint relates at most two variables
Color a map so that no adjacent countries have the same color.

**Example: Map Coloring**
Notice that many real-world problems involve real-valued variables.

- Factory scheduling
- Transportation scheduling
- Spreadshets
- Hardware configuration
- e.g., which class is offered when and where?
- Timetabling problems
- e.g., who teaches what class
- Assignment problems

Real-world CSPs
Let's start with the straightforward dumb approach, then fix it.

*States* are defined by the values assigned so far.

**Initial state:** all variables unassigned

**Operators:** assign a value to an unassigned variable

**Goal test:** all variables assigned, no constraints violated

Notice that this is the same for all CSPs!

**Applying standard search**
CSPstate keeps track of which variables have values so far. Each variable has a domain and a current value. CSP state keeps track of which variables have values so far.

Implementation

- Branching factor $b = \ldots$
- Search algorithm to use $\ldots$
- Depth of solution state $p = \ldots$
- Max. depth of space $w = \ldots$

Complexity of the dumb approach
Can solve $n$-queens for $n \approx 15$

Backtracking search is the basic uninformed algorithm for CSPs.

1. Fix the order of assignment
2. Check for constraint violations

The constraint violation check can be implemented in two ways:

- Modify the SUCCESSORS function to assign only values that are allowed, given the values already assigned.
- Check constraints are satisfied before expanding a state.

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

Use Depth-first search, but...
Keep track of remaining legal values for unassigned variables.

Terminate search when any variable has no legal values.

Idea: Keep track of remaining legal values for unassigned variables.

Forward checking.
Idea: Keep track of remaining legal values for unassigned variables.

Terminate search when any variable has no legal values.

Forward checking
Idea: Keep track of remaining legal values for unassigned variables.
Idea: Terminate search when any variable has no legal values.

Forward checking:
Simplified map-coloring example:

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>$C_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$C_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
</tr>
<tr>
<td>$X$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$C_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\land$</td>
</tr>
<tr>
<td>$X$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$C_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\land$</td>
</tr>
<tr>
<td>$X$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$C_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\land$</td>
</tr>
</tbody>
</table>

Red Blue Green

Idea: Terminate search when any variable has no legal values

Idea: Keep track of remaining legal values for unassigned variables

Forward checking
More intelligent decisions on which variable to assign next and which value to choose for each variable.

Given $C_1 = \text{Red}$, $C_2 = \text{Green}$, what next?

Choose $C_3 = \text{Green}$.

More heuristics for CSPs.
Can solve $n$-queens for $n \approx 1000$

Given $C_1 = \text{Red}, C_2 = \text{Green}$, what next?

$C_3$: least-constraining-value

$C_5$: most-constraining-variable

Choose $C_3 = \text{Green}$.

Which variable to assign next?

Which value to choose for each variable?

More intelligent decisions on heuristics for CSPs.
Hill-climbing, simulated annealing typically work with “complete” states, i.e., all variables assigned.

To apply to CSPs:

**Iterative Algorithms for CSPs**

---

Hill-climbing, simulated annealing typically work with “complete” states, i.e., all variables assigned.

Variable selection: randomly select any conflicted variable.

Operators: reassign variable values

allow states with unsatisfied constraints

**min-conflicts heuristic:**

Choose value that violates the fewest constraints, i.e., hillclimb with

\[ \text{min-conflicts} = \min \left( n \left( v \right) \right) \]

where \( n \left( v \right) \) is total number of violated constraints
4 queens in 4 columns (4^4 = 256 states)

Operators: move queen in column

Goal test: no attacks

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

Example: 4 Queens
Given random initial state, can solve n-queens in almost constant time for any randomly-generated CSP except in a narrow range of the ratio

\[
\frac{\text{number of variables}}{\text{number of constraints}} = R
\]

except for n with high probability (e.g., \(n = 10,000,000\)).

Given random initial state, can solve n-queens in almost constant time for any randomly-generated CSP except in a narrow range of the ratio

\[
\frac{\text{number of variables}}{\text{number of constraints}} = R
\]

Performance of min-conflicts
Determine whether a sentence in CNF (conjunctive normal form) is satisfiable.

Walksat: Add randomness

Function Gsat(sentence, max-clauses, max-threads) returns a truth assignment or failure

\[
\left( \neg J \land H \right) \lor \left( J \land H \land \neg d \right) \lor \left( \neg s \land \neg \neg \right) \land \neg d
\]

E.9.9
Based on AMI A Stiles ©5, Russell, and P. Norvig, 1998 Sections 3.7 and 4.4, Exercise 6.15, Weld paper on AI Planning

function Satisfiable(Problem, Initial State, Goal, Actions) returns Plan or Failure

1. Compile the planning problem (initial state, goal, actions) into CNF
2. Try to solve CNF (e.g., using GSAT, WalkSat)
3. If satisfying assignment is found then decode and return plan
4. Return failure

end
is instantiated to:

\[ (\forall \text{At}(A,B) \land \text{At}(B,A)) \]

\[ \text{goal} = \text{At}(\text{book, university}, n) \]

\[ \text{At}(@\text{objective, local}, i) \leftarrow \text{At}(\text{objective, local}, i) \land (\text{local} \neq \text{local}) \]

Compilation:

SatPlan cont'd.
Decoding, plan construction (if satisfying assignment is found):

Check successor states and find actions responsible for transitions

SatPlan contd.
CSPs are a special kind of problem: states defined by values of a fixed set of variables; goal test defined by constraints on variable values; states defined by values of a fixed set of variables; forward checking prevents assignments that guarantee later failure.

Variable ordering and value selection heuristics help significantly.

BlackBox [Kautz] combines SatPlan with GraphPlan.

Variable ordering and value selection heuristics are often more efficient than special purpose planners for planning as satisfiability are often more efficient than special purpose planners.