Review of constraint satisfaction problems (CSPs)

We have:

- A set of \( n \) variables \( V_1, V_2, \ldots, V_n \).
- For each \( V_i \), a domain \( D_i \) specifying the values that \( V_i \) can take.
- A set of \( m \) constraints \( C_1, C_2, \ldots, C_m \).

Each constraint \( C_i \) involves a set of variables and specifies an allowable collection of values:

- A state is an assignment of specific values to some or all of the variables.
- An assignment is consistent if it violates no constraints.
- An assignment is complete if it gives a value to every variable.

A solution is a consistent and complete assignment.

Example

We will use the problem of colouring the nodes of a graph as a running example.

Each node corresponds to a variable. We have three colours and directly connected nodes should have different colours.

Caution required: later on, edges will have a different meaning.

Example

This translates easily to a CSP formulation:

- The variables are the nodes
  \[ V_i = \text{node } i \]
- The domain for each variable contains the values black, red and cyan
  \[ D_i = \{B, R, C\} \]
- The constraints enforce the idea that directly connected nodes must have different colours. For example, for variables \( V_1 \) and \( V_2 \) the constraints specify
  \[(B, R), (B, C), (R, B), (R, C), (C, B), (C, R)\]
- Variable \( V_3 \) is unconstrained.

Different kinds of CSP

This is an example of the simplest kind of CSP: it is discrete with finite domains. We will concentrate on these.

We will also concentrate on binary constraints; that is, constraints between pairs of variables.

- Constraints on single variables—unary constraints—can be handled by adjusting the variable’s domain. For example, if we don’t want \( V_i \) to be red, then we just remove that possibility from \( D_i \).
- Higher-order constraints applying to three or more variables can certainly be considered, but...
- … when dealing with finite domains they can always be converted to sets of binary constraints by introducing extra auxiliary variables.

How does that work?
The state-variable representation

Another planning language: the state-variable representation.
Things of interest such as people, places, objects etc are divided into domains:

\[ D_1 = \{ \text{climber1, climber2} \} \]
\[ D_2 = \{ \text{home, jokeShop, hardwareStore, pavement, spire, hospital} \} \]
\[ D_3 = \{ \text{rope, inflatableGorilla} \} \]

Part of the specification of a planning problem involves stating which domain a particular item is in. For example

\[ D_1(\text{climber1}) \]

and so on.

Relations and functions have arguments chosen from unions of these domains.

\[ \text{above}(x, y) \subseteq D_1^{\text{above}} \times D_2^{\text{above}} \]

is a relation. The \( D_i^{\text{above}} \) are unions of one or more \( D_i \).

Note:

- For properties such as a \textit{location} a function might be considerably more suitable than a relation.
- For locations, everything has to be \textit{somewhere} and it can only be in one \textit{place at a time}.

So a function is perfect and immediately solves some of the problems seen earlier.

Actions as usual, have a name, a set of preconditions and a set of effects.

- \textit{Names} are unique, and followed by a list of variables involved in the action.
- \textit{ Preconditions} are expressions involving state variables and relations.
- \textit{ Effects} are assignments to state variables.

For example:

<table>
<thead>
<tr>
<th>Name</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{buy}(x, y, l)</td>
<td>at(x, s) = l</td>
<td>has(y, s) = x</td>
</tr>
<tr>
<td>\texttt{sell}(l, y)</td>
<td>aells(l, y)</td>
<td></td>
</tr>
</tbody>
</table>
The state-variable representation

Goals are sets of expressions involving state variables. For example:

Goal:
\[ \text{at(climber, s)} = \text{home} \]
\[ \text{has(rope, s)} = \text{climber} \]
\[ \text{at(gorilla, s)} = \text{spire} \]

From now on we will generally suppress the state \( s \) when writing state variables.

The state-variable representation

Considering all the ground actions consistent with the rigid relations:

- An action is applicable in \( s \) if all expressions \( \nu = c \) appearing in the set of preconditions also appear in \( s \).

Finally, there is a function \( \gamma \) that maps a state and an action to a new state

\[ \gamma(s, a) = s' \]

Specifically, we have

\[ \gamma(s, a) = \{ (\nu = c) | \nu \in X \} \]

where either \( c \) is specified in an effect of \( a \), or otherwise \( \nu = c \) is a member of \( s \).

Note: the definition of \( \gamma \) implicitly solves the frame problem.

The state-variable representation

We can essentially regard a state as just a statement of what values the state variables take at a given time.

Formally:

- For each state variable \( sv \) we can consider all ground instances such as—\( sv(climber, \text{rope}) \)—with arguments that are consistent with the rigid relations.

Define \( X \) to be the set of all such ground instances.

- A state \( s \) is then just a set

\[ s = \{ (\nu = c) | \nu \in X \} \]

where \( c \) is in the range of \( \nu \).

This allows us to define the effect of an action.

A planning problem also needs a start state \( s_0 \), which can be defined in this way.

The state-variable representation

A solution to a planning problem is a sequence \( (a_0, a_1, \ldots, a_n) \) of actions such that...

- \( a_0 \) is applicable in \( s_0 \) and for each \( i, a_i \) is applicable in \( s_i = \gamma(s_{i-1}, a_{i-1}) \).

- For each goal \( g \) we have

\[ g \in \gamma(s_n, a_n) \]

What we need now is a method for transforming a problem described in this language into a CSP.

We'll once again do this for a fixed upper limit \( T \) on the number of steps in the plan.
Converting to a CSP

Step 1: encode actions as CSP variables.
For each time step $t$ where $0 \leq t \leq T - 1$, the CSP has a variable

$\text{action}^t$

with domain $D^{\text{action}^t} = \{a | a \text{ is the ground instance of an action}\} \cup \{\text{none}\}$

Example: at some point in searching for a plan we might attempt to find the solution to the corresponding CSP involving

$\text{action}^5 = \text{attach(inflatableGorilla, spire)}$

WARNING: be careful in what follows to distinguish between state variables, actions etc in the planning problem and variables in the CSP.

Converting to a CSP

Step 2: encode ground state variables as CSP variables, with a complete copy of all the state variables for each time step.
So, for each $t$ where $0 \leq t \leq T$ we have a CSP variable

$\text{sv}_t[v_1, \ldots, v_n]$

with domain $D^{\text{sv}_t}$. (That is, the domain of the CSP variable is the range of the state variable.)

Example: at some point in searching for a plan we might attempt to find the solution to the corresponding CSP involving

$\text{location}^9(\text{climber1}) = \text{hospital}$.  

Converting to a CSP

Step 3: encode the preconditions for actions in the planning problem as constraints in the CSP problem.
For each time step $t$ and for each ground action $a(c_1, \ldots, c_n)$ with arguments consistent with the rigid relations in its preconditions:

For a precondition of the form $\text{sv}_t[v]$, include constraint pairs

$\text{action}^t = a(c_1, \ldots, c_n),$

$\text{sv}_t^t[v] = v$ .

Example: consider the action $\text{buy}(x, y, l)$ introduced above, and having the preconditions $\text{at}\{x\} = l$, $\text{sells}\{l, y\}$ and $\text{has}\{y\} = l$.

Assume $\text{sells}\{y, l\}$ is only true for

$l = \text{jokeShop}$

and

$y = \text{inflatableGorilla}$

(it’s a very strange town) so we only consider these values for $l$ and $y$. Then for each time step $t$ we have the constraints...

| action$^t$ = buy(climber1, inflatableGorilla, jokeShop) | paired with |
| at$^t$(climber1) = jokeShop |

| action$^t$ = buy(climber1, inflatableGorilla, jokeShop) | paired with |
| has$^t$(inflatableGorilla) = jokeShop |

| action$^t$ = buy(climber2, inflatableGorilla, jokeShop) | paired with |
| at$^t$(climber2) = jokeShop |

| action$^t$ = buy(climber2, inflatableGorilla, jokeShop) | paired with |
| has$^t$(inflatableGorilla) = jokeShop |

and so on...
Converting to a CSP

Step 4: encode the effects of actions in the planning problem as constraints in the CSP problem.

For each time step $t$ and for each ground action $a(c_1, \ldots, c_n)$ with arguments consistent with the rigid relations in its preconditions:

For an effect of the form $sv_i = v$ include constraint pairs

$$\{
\text{action}^t = a(c_1, \ldots, c_n),
sv_{t+1}^i = v\}$$

Example: continuing with the previous example, we will include constraints

<table>
<thead>
<tr>
<th>action\textsuperscript{t} = buy(climber1, inflatableGorilla, jokeShop)</th>
</tr>
</thead>
<tbody>
<tr>
<td>paired with</td>
</tr>
<tr>
<td>has\textsuperscript{t+1}(inflatableGorilla) = climber1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>action\textsuperscript{t} = buy(climber2, inflatableGorilla, jokeShop)</th>
</tr>
</thead>
<tbody>
<tr>
<td>paired with</td>
</tr>
<tr>
<td>has\textsuperscript{t+1}(inflatableGorilla) = climber2</td>
</tr>
<tr>
<td>and so on...</td>
</tr>
</tbody>
</table>

Finding a plan

Finally, having encoded a planning problem into a CSP, we solve the CSP.

The scheme has the following property:

A solution to the planning problem with at most $T$ steps exists if and only if there is a a solution to the corresponding CSP.

Assume the CSP has a solution.

Then we can extract a plan simply by looking at the values assigned to the action\textsuperscript{t} variables in the solution of the CSP.

It is also the case that:

There is a solution to the planning problem with at most $T$ steps if and only if there is a solution to the corresponding CSP from which the solution can be extracted in this way.

For a proof see:

Automated Planning: Theory and Practice