We describe a way to solve problems more efficiently that have an internal structure.

**Factored Representation** = for each state: a set of variables, with a value

The problem is solved when each variable has a value satisfying all the constraints on the var.

Hence, Constraint Satisfaction Problems.

Main Idea – To eliminate large portions of each search space all at once by identifying variable/value combinations that violate the constraints.

CSP Problems consist of

3 components: X is a set of variable {Xsub 1, … , Xsub n}

D is a set of domains {Dsub 1, … , Dsub n}, one for each variable

C is a set of constraints that specify allowable combinations of values

In each domain there is a set of allowable values {Vsub 1, … Vsub k} for each variable.

In each constraint there is a pair of <scope, rel>

where scope is a tuple of variable in the constraint

where rel is a relation that defines the values that those variable can take on

\*Tip – it can be helpful to visualize a CSP , as a constraint graph

The nodes of the graph correspond to variables of the problem , and a link connects any two variables that participate in a constraint

CSP yields a natural representation for a wide variety of problems.

(Quickly eliminate large parts of the search space)

The simplest kind of CSP involves variable that have discrete, finite domains.

(e.g. map coloring and scheduling with time limits)

* It can be shown that no algorithm exists for solving nonlinear constraints.
* There are special solution algorithms for linear constraints.
* Linear programming problems do Constraint Optimization Problems where CSP’s with preferences can be solved by optimization search methods, either path-based or local.
* CSP with continuous domains are common in the real world.

**Cutset conditioning = reduce a CSP to a tree form**

Tree structured graphs can always be solved in linear time. You can add Tree Composition techniques to make a tree into subproblems and are efficient if the tree width is small.

CSP’s =- states are defined by value of a fixed set of variable

=- goal test is defined by constraints on variable values

Unary constraints involve a single variable (simplest type of CSP)

Binary constraints involve pairs of variables (easily represented as a graph)

Global Constraint (an arbitrary number of variables)

^Higher-order constraints involve 3 or more variables^

Preferences are soft constraints involve being representable by a cost for each variable assignment

CSP don’t have to just search they can apply constraint propagation - repeatedly enforcing constraints locally. **results are reduced number of legal values for a variable** , which can in result reduce the values for another variable …. And so on. Providing LOCAL consistency

General Purpose strategy can give huge gains in speed:

We look for which variable should be assigned next, what order should its values be tried, can we detect failure early, can we take advantage of its internal structure.

So we use methods of

Variable ordering and value selection heuristics to help.

\*Tip – a variable ordering that chooses a variable with the minimum number of remaining values helps minimize the number of nodes in the search tree by pruning large parts of the tree earlier.

* **Minimum-remaining-values (MRV) heuristic** aka “most constrained variable” or “fail-first” heuristic

(it picks a variable that is most likely to cause a failure soon), pruning the search tree

* + If there are several MRV variables, we can use the degree heuristic:
  + -By applying the **degree heuristic** – choose the variable with the most constraints on remaining variables
    - solving the Map coloring problem with no false steps ( no backtracking ) you can choose a color at each choice point.
  + When we have selected a variable using MRV and degree heuristic, we choose the least constraining value: – the one that rules out the fewest values in the remaining variables
* **Least-constraining-value heuristic** – in general , it is trying to leave the maximum flexibility for subsequent variable assignments.
* **Forward checking** will help prevent assignments that guarantee failure later
  + Key Concept: Keep track of remaining legal values for unassigned variables Terminate search when any variable has no legal values

**MAC (maintaining arc consistency) is more powerful**

It will recursively propagate constraints when changes are made to the domains of variables.

-While Forward Checking detects many inconsistencies it does not find all of them , hence MAC will detect all of the inconsistencies.

* + - -Constraint propagation for example arc consistency does additional work to constrain values and detect inconsistencies.
* **Backtracking search** is the basic uninformed algorithm for CSP’s

Backtracking = depth-first search with one variable assigned per node

The idea of the backtracking search began in 1965 with Golomb and Baumert.

Its application to constraint satisfaction was done by Bitner and Reingold in 1975.

Bitner and Reingold introduced the MRV heuristic, “ the most-constrained-variable” heuristic

Even though it’s a simple algorithm, it is still the best method for k-coloring arbitrary graphs.

https://youtu.be/vPluRAznFPw