Value and Variable Ordering

[variable ordering] [value ordering]

A search algorithm for constraint satisfaction requires the order in which variables are to be considered to be specified as well as the order in which the values are assigned to the variable on backtracking. Choosing the right order of variables (and values) can noticeably improve the efficiency of constraint satisfaction.

Variable Ordering

Experiments and analysis of several researchers have shown that the ordering in which variables are chosen for instantiation can have substantial impact on the complexity of backtrack search. The ordering may be either

- a *static ordering*, in which the order of the variables is specified before the search begins, and it is not changed thereafter, or
- a *dynamic ordering*, in which the choice of next variable to be considered at any point depends on the current state of the search.

Dynamic ordering is not feasible for all search algorithms, e.g., with simple backtracking there is no extra information available during the search that could be used to make a different choice of ordering from the initial ordering. However, with forward checking, the current state includes the domains of the variables as they have been pruned by the current set of instantiations, and so it is possible to base the choice of next variable on this information.

Several heuristics have been developed and analyzed for selecting variable ordering. The most common one is based on the "**first-fail**" principle, which can be explained as

"*To succeed, try first where you are most likely to fail.*"

In this method, the variable with the fewest possible remaining alternatives is selected for instantiation. Thus the order of variable instantiations is, in general, different in different branches of the tree, and is determined dynamically. This method is based on assumption that any value is equally likely to participate in a solution, so that the more values there are, the more likely it is that one of them will be a successful one.

The first-fail principle may seem slightly misleading, after all, we do not want to fail. The reason is that if the current partial solution does not lead to a complete solution, then the sooner we discover this the better. Hence encouraging early failure, if failure is inevitable, is beneficial in the long term. On the other end, if the current partial solution can be extended to a complete solution, then every remaining variable must be instantiated and the one with smallest domain is likely to be the most difficult to find a value for (instantiating other variables first may further reduce its domain and lead to a failure). Hence the principle could equally well be stated as:

"*Deal with hard cases first: they can only get more difficult if you put them off.*"

This heuristic should reduce the average depth of branches in the search tree by triggering early failure.

Another heuristic, that is applied when all variables have the same number of values, is to choose the variable which participates in most constraints (in the absence of more specific information on which constraints are likely to be difficult to satisfy, for instance). This heuristic follows also the principle of dealing with hard cases first.

There is also a heuristic for static ordering of variables that is suitable for simple backtracking. This heuristic says: choose the variable which has the largest number of constraints with the past variables. For instance, during solving graph coloring problem, it is reasonable to assign color to the vertex which has common arcs with already colored vertices so the conflict is detected as soon as possible.

Value Ordering

Once the decision is made to instantiate a variable, it may have several values available. Again, the order in which these values are considered can have substantial impact on the time to find the first solution. However, if all solutions are required or there are no solutions, then the value ordering is indifferent.

A different value ordering will rearrange the branches emanating from each node of the search tree. This is an advantage if it ensures that a branch which leads to a solution is searched earlier than branches which lead to death ends. For example, if the CSP has a solution, and if a correct value is chosen for each variable, then a solution can be found without any backtracking.

Suppose we have selected a variable to instantiate: how should we choose which value to try first? It may be that none of the values will succeed, in that case, every value for the current variable will eventually have to be considered, and the order does not matter. On the other hand, if we can find a complete solution based on the past instantiations, we want to choose a value which is likely to succeed, and unlikely to lead to a conflict. So, we apply the **"succeed first"** principle.

One possible heuristic is to prefer those values that maximize the number of options available. Visibly, the algorithm AC-4 is good for using this heuristic as it counts the supporting values. It is possible to count "promise" of each value, that is the product of the domain sizes of the future variables after choosing this value (this is an upper bound on the number of possible solutions resulting from the assignment). The value with highest promise should be chosen. Is also possible to calculate the percentage of values in future domains which will no longer be usable. The best choice would be the value with lowest cost.

Another heuristic is to prefer the value (from those available) that leads to an easiest to solve CSP. This requires to estimate the difficulty of solving a CSP. One method [Dechter] propose to convert a CSP into a tree-structured CSP by deleting a minimum number of arcs and then to find all solutions of the resulting CSP (higher the solution count, easier the CSP).

For randomly generated problems, and probably in general, the work involved in assessing each value is not worth the benefit of choosing a value which will on average be more likely to lead to a solution than the default choice. In particular problems, on the other hand, there may be information available which allows the values to be ordered according to the principle of choosing first those most likely to succeed.

[variable ordering] [value ordering]

based on Barbara M. Smith: A Tutorial on Constraint Programming, TR 95.14, University of Leeds,1995

Designed and maintained by Roman Barták