Constraint Propagation

[backtracking] [forward checking] [look ahead] [comparison]

In the previous sections we presented two rather different schemes for solving the CSP: backtracking and consistency techniques. A third possible scheme is to embed a consistency algorithm inside a backtracking algorithm as follows.

As a skeleton we use the simple backtracking algorithm that incrementally instantiates variables and extends a partial solution that specifies consistent values for some of the variables, toward a complete solution, by repeatedly choosing a value for another variable. After assigning a value to the variable, some consistency technique is applied to the constraint graph. Depending on the degree of consistency technique we get various constraint satisfaction algorithms.

Backtracking

Even simple backtracking (BT) performs some kind of consistency technique and it can be seen as a combination of pure generate & test and a fraction of arc consistency. The BT algorithm tests arc consistency among already instantiated variables, i.e., the algorithm checks the validity of constraints considering the partial instantiation. Because the domains of instantiated variables contains just one value, it is possible to check only those constraints/arcs containing the last instantiated variable. If any domain is reduced then the corresponding constraint is not consistent and the algorithm backtracks to a new instantiation.

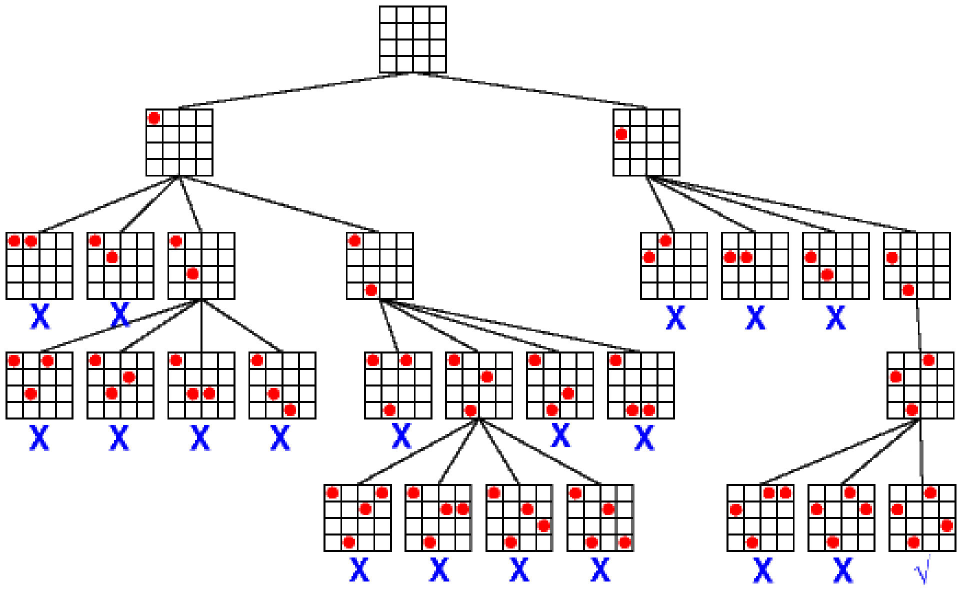
The following procedure AC3-BT is called each time a new value is assigned to some variable V_{cv} (cv is the consecutive number of the variable in the order of instantiating variables)

Algorithm AC-3 for Backtracking

```
procedure AC3-BT(cv)
  Q <- {(Vi,Vcv) in arcs(G),i<cv};
  consistent <- true;
  while not Q empty & consistent
    select and delete any arc (Vk,Vm) from Q;
    consistent <- REVISE(Vk,Vm)
  endwhile
  return consistent
end AC3-BT
```

The BT algorithm detects the inconsistency as soon as it appears and, therefore, it is far away efficient than the simple generate & test approach. But it has still to perform too much search.

Example: (4-queens problem and BT)



The BT algorithm can be easily extended to backtrack to the conflicting variable and, thus, to incorporate some form of look-back scheme or intelligent backtracking. Nevertheless, this adds some additional expenses to the algorithm and it seems that preventing possible future conflicts is more reasonable than recovering from them.

Forward Checking

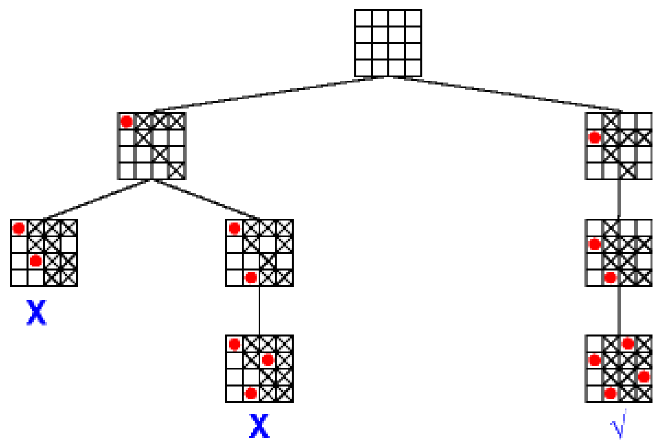
Forward checking is the easiest way to prevent future conflicts. Instead of performing arc consistency to the instantiated variables, it performs restricted form of arc consistency to the not yet instantiated variables. We speak about restricted arc consistency because forward checking checks only the constraints between the current variable and the future variables. When a value is assigned to the current variable, any value in the domain of a "future" variable which conflicts with this assignment is (temporarily) removed from the domain. The advantage of this is that if the domain of a future variable becomes empty, it is known immediately that the current partial solution is inconsistent. Forward checking therefore allows branches of the search tree that will lead to failure to be pruned earlier than with simple backtracking. Note that whenever a new variable is considered, all its remaining values are guaranteed to be consistent with the past variables, so the checking an assignment against the past assignments is no longer necessary.

Algorithm AC-3 for Forward Checking

```
procedure AC3-FC(cv)
  Q <- {(Vi,Vcv) in arcs(G),i>cv};
  consistent <- true;
  while not Q empty & consistent
    select and delete any arc (Vk,Vm) from Q;
    if REVISE(Vk,Vm) then
      consistent <- not Dk empty
    endif
  endwhile
  return consistent
end AC3-FC
```

Forward checking detects the inconsistency earlier than simple backtracking and thus it allows branches of the search tree that will lead to failure to be pruned earlier than with simple backtracking. This reduces the search tree and (hopefully) the overall amount of work done. But it should be noted that forward checking does more work when each assignment is added to the current partial solution.

Example: (4-queens problem and FC)



Forward checking is almost always a much better choice than simple backtracking.

Look Ahead

Forward checking checks only the constraints between the current variable and the future variables. So why not to perform full arc consistency that will further reduces the domains and removes possible conflicts? This approach is called **(full) look ahead** or **maintaining arc consistency** (MAC).

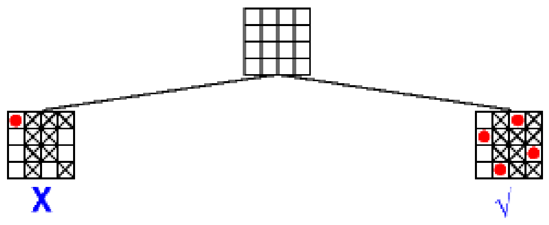
The advantage of look ahead is that it detects also the conflicts between future variables and therefore allows branches of the search tree that will lead to failure to be pruned earlier than with forward checking. Also as with forward checking, whenever a new variable is considered, all its remaining values are guaranteed to be consistent with the past variables, so the checking an assignment against the past assignments is no necessary.

Algorithm AC-3 for Look Ahead

```
procedure AC3-LA(cv)
  Q <- {(Vi,Vcv) in arcs(G),i>cv};
  consistent <- true;
  while not Q empty & consistent
    select and delete any arc (Vk,Vm) from Q;
    if REVISE(Vk,Vm) then
      Q <- Q union {(Vi,Vk) such that (Vi,Vk) in arcs(G),i#k,i#m,i>cv}
      consistent <- not Dk empty
    endif
  endwhile
  return consistent
end AC3-LA
```

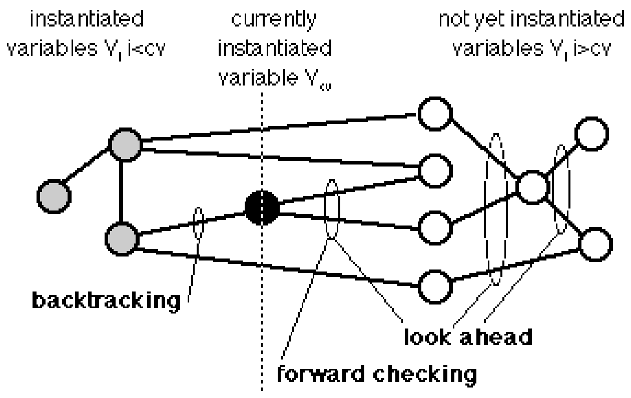
Look ahead prunes the search tree further more than forward checking but, again, it should be noted that look ahead does even more work when each assignment is added to the current partial solution than forward checking.

Example: (4-queens problem and LA)



Comparison of propagation techniques

The following figure shows which constraints are tested when the above described propagation techniques are applied.


instantiated variables V_i i<cv
currently instantiated variable V_{cv}
not yet instantiated variables V_i i>cv
backtracking
look ahead
forward checking

More constraint propagation at each node will result in the search tree containing fewer nodes, but the overall cost may be higher, as the processing at each node will be more expensive. In one extreme, obtaining strong n-consistency for the original problem would completely eliminate the need for search, but as mentioned before, this is usually more expensive than simple backtracking. Actually, in some cases even the full look ahead may be more expensive than simple backtracking. That is the reason why forward checking and simple backtracking are still used in applications.

[backtracking] [forward checking] [look ahead] [comparison]

Contents **Prev** **Up** **Next**

Designed and maintained by Roman Barták