Asynchronous Ordering Heuristics

The case of asynchronous ordering is quite complex. Assignments of agents are performed concurrently and asynchronously, so the selection of a next agent to be assigned is not clear. The order of agents here is expected to have some impact, depending on the algorithm. However, unlike centralized or synchronous search it is not clear how the order changes the traversal of the search tree. Intuitively, a given order selects a subset of paths on the search tree - those that have the nodes in a similar enough order. Since nodes perform their computations asynchronously, two ordered agents $A_i < A_j$ can have their assignments performed in any order. The order just dictates that, when the assignments of these two agents are part of a NOGOOD, only agent $A_j$ can backtrack to $A_i$ and not vice versa.

An attempt to order the agents dynamically during the run of an asynchronous search algorithm was performed quite early. In 1995 Yokoo proposed an asynchronous search algorithm, similar to ABT, that does not use a static order of agents. This first attempt at asynchronous ordering is described in the following section. It uses a specific ordering, moving an agent that performs a backtrack (e.g., sends a NOGOOD) to the top place in the order.

9.1 Specific Asynchronous Heuristics

As mentioned before, the first attempt to introduce dynamic ordering into asynchronous backtracking was performed by Yokoo in 1995 (cf. [62, 63]). The idea was to take ABT and make a small change, so that some reordering would be done during search. The Asynchronous Weak Commitment (AWC) algorithm implements a specific ordering heuristic. Order can be changed only after a backtrack operation and only the agent sending the NOGOOD changes its position. The only change of position is for the NOGOOD sending agent to move to the first position and it must do so. This is a specific ordering heuristic, of moving all the way to the first position and it is performed everytime that a NOGOOD message is sent. The code needed for AWC, in
addition to standard ABT that sends complete Agent_Views as NOGOODS (Algorithm 5.1), is presented in Algorithm 9.1.

Algorithm 9.1: Additional code for the AWC algorithm

- procedure check_agent_view
  1. when Agent_View and current_value are not consistent do
  2. if no value in $D_i$ is consistent with Agent_View
  3. then backtrack;
  4. else select $d \in D_i$ where Agent_View and $d$ are consistent and
     d minimizes the number of constraint violations with lower priority agents;
  5. current_value ← $d$;
  6. send (ok?, $(x_i, d), current\_priority$) to neighbors;
  8. end_if; end_do;

- procedure backtrack
  1. nogood ← inconsistent_subset;
  2. when nogood is an empty set do
     broadcast to other agents that there is no solution;
  4. terminate; end_do;
  5. when nogood is a new NOGOOD do
     send nogood to the agents in the nogood
  7. current\_priority ← 1 + $p_{max}$ # $p_{max}$ is the maximal priority value over all neighbours
     select $d \in D_i$ where Agent_View and $d$ are consistent and
     $d$ minimizes the number of constraint violations with lower priority agents;
  9. current_value ← $d$;
  11. send (ok?, $(x_i, d), current\_priority$) to neighbors;

As can be seen in Algorithm 9.1, another major heuristic for ordering values is employed in AWC. Whenever a value is selected for assignment, either in the check_agent_view procedure or during backtrack, the value selected minimizes conflicts with lower priority agents. This is a very special change from standard ABT, since in any version of ABT agents do not know the value of lower-priority agents. The reason for this possibility in AWC is that ok? messages are sent to all neighboring agents (line 7 in procedure check_agent_view in Algorithm 9.1).

The interesting point about the AWC algorithm is that it can be easily proven correct even with its reordering procedure. To see why it is correct, all one needs is to observe that the correctness arguments for the ABT version with complete Agent_Views as NOGOODS applies to AWC as well. Consider the proof of correctness for ABT with complete Agent_Views as NOGOODS in Section 5.3. Soundness applies immediately because an idle state means...