Asynchronous Backtracking (ABT)

The *asynchronous backtracking* algorithm (*ABT*) first presented by Yokoo [18] was constructed to remove the drawbacks of *synchronous backtracking* (*SBT*) by allowing agents to perform assignments asynchronously. In all the presentations of ABT, the algorithm is presented for DisCSPs in which each agent holds exactly one variable. This avoids the problem of reconciling the total order of the agents, which is assumed by ABT, with the order of the variables. When agents hold multiple variables, these are two distinct orders that the algorithm has to address. Each agent assigns its variable and communicates the assignment it made to the relevant agents.

The ABT algorithm in its standard form assumes that a total order of priorities is defined among agents. In Chapter 9 a new and innovative version of the ABT algorithm will be presented, in which agents can change their order dynamically during search [74]. However, in order to both make the understanding of ABT simpler and follow its history, it will be presented here in the form in which it was used for most of the last decade [9, 63]. Each binary constraint is known to both of the constrained agents and is checked in the algorithm by the agent with the lower priority among the two. A link in the constraint network is *directed* from the agent with the higher priority to the agent with the lower priority among the two constrained agents.

Agents instantiate their variables concurrently and send their assigned values to the agents that are connected to them by outgoing links. All agents wait for and respond to messages. After each update of its assignment, an agent sends through all outgoing links its new assignment. An agent which receives an assignment (from the higher-priority agent of the link) tries to find an assignment for its variable which does not violate a constraint with the assignment it received.

ok? messages are messages carrying an assignment of an agent. When an agent A_i receives an **ok?** message from agent A_j, it places the received assignment in a data structure called $Agent\_View$, which holds the last assignment A_i received from higher-priority neighbors such as A_j. Next, A_i checks if its current assignment is still consistent with its $Agent\_View$. If it is consistent,

A_i does nothing. If not, then A_i searches its domain for a new consistent value. If it finds one, it assigns its variable and sends **ok?** messages to all lower-priority agents linked to it. Otherwise, A_i backtracks.

The *backtrack* operation is executed by sending a NOGOOD message that contains an inconsistent partial assignment. NOGOODs are sent to the agent with the lowest priority among the agents whose assignments are included in the inconsistent tuple in the NOGOOD. Agent A_i that sends a NOGOOD message to agent A_j assumes that A_j will change its assignment. Therefore, A_i removes from its $Agent\_View$ the assignment of A_j and makes an attempt to find an assignment for its variable that is consistent with the updated $Agent\_View$.

The issue of how to resolve the inconsistent partial assignment (NOGOOD) which will be sent in the backtrack message evolved through the different versions of ABT. A shorter NOGOOD would mean backjumping further up the search tree, but finding such a short NOGOOD can be wasteful in computational time. In the early versions of ABT ([18, 64]), Yokoo proposes to send the full $Agent\_View$ as a NOGOOD. The full $Agent\_View$ is in many cases not a minimal NOGOOD. In other words, it might contain assignments that, if removed, the remaining partial assignment still eliminates all values in the agent's domain. The reason that sending the whole $Agent\_View$ back is correct (but unsatisfactory) can be explained by the following example.

Consider an agent A_6 which holds an inconsistent $AgentView$ with the assignments of agents A_1, A_2, A_3, A_4, and A_5. If we assume that A_6 is only constrained by the current assignments of A_1 and A_3, sending a NOGOOD message to A_5 which contains all the assignments in the $Agent\_View$ seems to be a waste. After sending the NOGOOD to A_5, A_6 will remove its assignment from the $Agent\_View$ and make another attempt to assign its variable which will be followed by an additional NOGOOD sent to A_4 and the removal of A_4's assignment from the $Agent\_View$. These attempts will continue until a minimal subset is sent as a NOGOOD. In this example, it is the NOGOOD sent to A_3. The assignment with the lower priority in the minimal inconsistent subset is removed from the $Agent\_View$ and a consistent assignment can now be found. In this example the computation ended by sending a NOGOOD to the culprit agent, which would have been the outcome that would have been achieved if the agent would have computed a minimal subset.

Let us turn now to the code of ABT in its simplest form. Algorithm 5.1 presents Yokoo's code that assumes sending complete $Agent\_Views$ as NOGOODs. It can be rougfly divided into two parts - moving forward and moving back. Note that due to ABT's asynchronous nature, the moves forward or backward are performed at the same time. There is no synchronization among actions taken by agents. Assignments and revoking of assignments are performed concurrently. When an assignment is performed by an agent, it sends an **ok?** message and upon receiving it the receiving agent performs lines 1-2 of the code in Algorithm 5.1. The function $Check\_Agent\_View$