Distributed Ordering Heuristics

Ordering heuristics, for both variables and values, play an important role in centralized CSP search [14]. Orders of variables can be either static or dynamic. If variables are ordered before the start of the run of the search algorithm and the order is not changed during the run of the algorithm, it is called static. Such a heuristic does not take into consideration changes in the relations among variables that occur during search. A typical example of a static ordering heuristic is to use the features of the constraints graph. A popular example is to select variables with a high degree to be higher in the order [15]. A higher degree means that the variable is constrained by a larger number of variables. A common search heuristic is to try earlier branches on the search tree that have a higher chance to fail. Variables that are constrained by a larger number of other variables seem more likely to fail, so they are selected to be higher in the static order that is based on the constraint graph. This general approach, of trying earlier variables that have a higher chance to fail, is called the fail-first principle (cf. [16]). It is considered to lead to a more effective pruning since it potentially prunes (fails) nodes earlier.

Since the early 1990s researchers in standard (centralized) CSPs have found that dynamic ordering heuristics are far more successful than static heuristics. The standard use of dynamic ordering is to select the next variable to be assigned. The idea is that during search many parameters of the variables change dynamically. If search takes the form of assigning values to variables one after the other, a natural place to insert the ordering heuristics is to pause for computation immediately after the assignment of some variable. Based on the results of the computation, the next variable to be assigned is selected. A very simple and very successful general heuristic is to select the variable with the smallest domain size among the unassigned variables. A small domain size seems intuitively to represent a faster way to fail (smaller number of trial assignments), so it conforms with the above principle of Fail First (FF). In fact, the FF principle is nowadays used synonymously with selecting the smallest domain.
The use of ordering heuristics in distributed search is a much more complex problem. The best known algorithm - asynchronous backtracking (ABT) - uses static ordering of agents, simply by the agents’ IDs \[64\]. As we have seen, the static order of agents is essential for the proof of correctness of ABT (see Chapter \[5\]).

There are two main points that need consideration for ordering a distributed search algorithm:

- Agents in a DisCSP do not know the parameters of other agents during search and therefore miss information that is needed in order to make ordering decisions.
- When the search algorithm is asynchronous the meaning of ordering of assignments is not clear. Would a heuristic change anything? What should be the rational?

The first point raises the need to accumulate knowledge about the state of other agents during search. As we will show in the sections below, this can be addressed by either additional computation or by additional information that is sent by messages. The second point raises a very complex issue. When the DisCSP search algorithm is either completely synchronous or performs assignments sequentially, the intuitive meaning of ordering is still analogous to that of centralized CSPs. A consistent partial assignment exists and a \textit{next agent} has to be selected for assignment. This selection process, though distributed, is similar to the centralized process. As we will see in the following two sections, the main difference from standard CSPs is the means of processing the distributed information about states of agents. The heuristics themselves are very similar to standard ones (e.g., Min-Domain).

For asynchronous backtracking, the situation is completely different. The ABT algorithm uses strictly static ordering, that is essential for its correctness proof \[9\]. A pioneering attempt to introduce dynamic ordering into asynchronous backtracking, was proposed by Yokoo as early as 1995. The idea was to implement a specific ordering, in which a failing (e.g., backtracking) agent was always moved to be first in the global order \[61, 62\]. The algorithm was called Asynchronous Weak-commitment (AWC) search in its original paper, but is very similar to ABT. However, AWC needs exponential storage space in order to store all potential \textit{Nogoods} (cf. \[61\]). In 2005 a generalized version of ABT was published, enabling a general dynamic order for agents \[71, 73\]. This version of asynchronous backtracking, that includes dynamic ordering - \textit{ABT\_DO}, includes a special time-stamping mechanism that enables agents to change their order and retain the correctness of asynchronous backtracking. The issue of asynchronous ordering heuristics is presented in detail in Chapter \[9\], including both the ABT\_DO algorithm and several innovative asynchronous heuristics.