

tuples of the Cartesian product $\{0..3\} \times \{0, 1\}$. Similarly, a solution is no more a list of individual assignments but a Cartesian product of intervals which represents a set of possible valuations. On random problems, the experiments show that AAS has an overall performance comparable to that of AS and that it is even slightly better on average. In scheduling and resource allocation problems with large domains, the savings allowed by the Cartesian product representation can be particularly interesting.

Figure 1 illustrates the behavior of AAS on our small example. Agent A^1 first selects the Cartesian product $x_1 = \{0..3\} \times x_3 = \{0\}$, and sends an **ok?** message with this information to A^2 , A^3 and A^4 who manage constraints sharing variables with A^1 . The algorithm now works in the same manner as AS, except that messages refer to Cartesian products and agents select different Cartesian products rather than value assignments. More specifically, A^4 finds that no combination in the Cartesian product $x_2 = \{0, 1\} \times x_3 = \{0\}$ is compatible with its constraint. It therefore generates a **nogood** for this combination which causes A^2 to select the next Cartesian product. Note that since this change selects a subrange of the values allowed for x_1 , it is not necessary to verify this change with A^1 . If it were not possible to find such a subrange, a **nogood** would be generated at A^2 and sent to A^1 in order to try another Cartesian-product there. There are several ways in which the agents can build the aggregations. In the current implementation, the aggregation mechanism guarantees a complete and non-redundant covering of the search space determined by the local constraints [1]. Variants such as the ones mentioned in [2] can also be considered.

In fact, AAS can be understood as a dynamic set of parallel AS, synchronized with each other and operating on the *dual* constraint graph, where the variables are relations of the original problem and the relations ensure that identical variables take on identical assignments. Variables are now tuple-valued and their domains are all the tuples allowed by the corresponding constraint, represented more compactly by a set of Cartesian products. Relations exist between any pair of agents that represent constraints sharing a variable. Thus, we can see that the execution of AAS is just a parallel set of ASs, an AS for each tuple of the current Cartesian products, on this dual problem, where: constraint redistribution is not required, local solutions are aggregated, non-redundant projections of aggregated dual values on the requested original variables are transmitted as Cartesian products. The guarantees of completeness, soundness and termination are preserved.

Finally, it is worth mentioning that we do not explicitly transform the original constraint graph into its dual form. That would lead to formulate huge domains for loosely constrained problems. We only consider at each node of the search, the local dual graph determined by the tuples active in the current context.

3. IMPLEMENTATION

AAS has been implemented using agents running as separate processes on different workstations. Our implementation uses a system agent to which the agents subscribe for the search. It decides the order of the agents and announces the termination and the result.

In AS, the messages must respect a FIFO order of delivery to be properly treated [3]. Our algorithm requires a stronger

condition to hold since the channel for each variable is no longer a tree but a graph. This means that several messages can arrive to the same agent, for changing the value of the same variable, through different paths of the graph. For example, in Figure 1 agent A^3 can receive messages concerning variable x_1 from both A^1 and A^2 . An order must therefore be established between these kind of messages. We associate a history of changes to any message, that allows the agent to properly order them. The history of changes is built by associating a chain of pairs $|a : b|$ to each variable of a message (see Figure 1). Such a pair means that a change of the variable's domain was performed by the agent with index a when its counter for the corresponding variable had the value b . To ensure termination, we use the next conventions: The history of changes where the agent with the smaller index or the counter with the larger value occurs first is the most recent. If a history is the prefix of the other, then the longer one is the most recent.

AAS has been evaluated on randomly generated problems with 10, 15 and 20 agents. The constraints have been distributed to the agents in the same way that they would have been in AS, so that the two algorithms can be compared. The size of domains is of 8 values and the problems are generated near the peak of difficulty with a density of 30% and a tightness of constraints of 35%. Three heuristics for building the Cartesian products have been evaluated. The cost of search is evaluated using the longest sequence of constraint checks. Each test is averaged over 50 instances. AAS performs slightly better than AS on average for problems with 20 variables. Similar results are obtained for 10 and 15 variables. Of course, depending on the problem generated and on the heuristic used for aggregation, there are specific cases where AS performs better for finding the first solution. However, for discovering that no solution exists AAS performs steadily better than AS since the whole search space needs to be expanded. AAS also reduces the longest sequence of messages as well as the number of nogoods stored by a factor of 2 on average.

We have presented AAS, a new asynchronous backtrack search algorithm. AAS is a generalization of asynchronous search (AS) [3]. It requires no artificial redistribution of constraints and allows for aggregating the information transmitted using the Cartesian product representation. AAS provides a natural support for enforcing privacy requirements on constraints. In the current implementation, the agents with the lower priority have to reveal more information about their constraints. If undesirable, such a behavior can be avoided using random or cyclic agent reordering.

4. REFERENCES

- [1] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Fractionnement intelligent de domaine pour CSPs avec domaines ordonnés. In *Proc. of RFAIA2000*, 2000.
- [2] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Ways of maintaining arc consistency in search using Cartesian representation. In *Proc. of ERCIM'99*, 99.
- [3] M. Yokoo and T. Ishida. *Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence*, chapter Search Algorithms for Agents, pages 165–199. MIT Press, 99.