Distributed Splitting of Constraint Satisfaction Problems

F. Arbab, E.B.G. Monfroy

Software Engineering (SEN)

SEN-R0027 October 31, 2000

Report SEN-R0027
ISSN 1386-369X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Distributed Splitting of Constraint Satisfaction Problems

Farhad Arbab and Eric Monfroy
CWI
P.O. Box 94079, 1090 GB, Amsterdam, The Netherlands
{Farhad.Arbab,Eric.Monfroy}@cwi.nl

ABSTRACT
Constraint propagation aims to reduce a constraint satisfaction problem into an equivalent but simpler one. However, constraint propagation must be interleaved with a splitting mechanism in order to compose a complete solver. In [13] a framework for constraint propagation based on a control-driven coordination model was presented.
In this paper we extend this framework in order to integrate a distributed splitting mechanism. This technique has three main advantages: 1) in a single distributed and generic framework, propagation and splitting can be interleaved in order to realize complete distributed solvers, 2) by changing only one agent, we can perform different kinds of search, and 3) splitting of variables can be dynamically triggered before the fixed point of a propagation is reached.

2000 ACM Computing Classification System: D.1.3, D.1.m (Cooperative Constraint Solving), D.2.13, D.3.2, D.3.m (Constraint Programming), I.1.3
Keywords and Phrases: Constraint solving, domain splitting, solver collaboration, coordination models and languages.

1. Introduction

Constraint propagation aims to reduce a constraint satisfaction problem (CSP) into an equivalent but simpler one by narrowing domains of variables until a fixed-point is reached. However, constraint propagation must be interleaved with a splitting mechanism in order to compose a complete solver. This mechanism works by splitting the domain of a variable (i.e., the values the variable can assume) into (sub)domains, creating in this way sub-CSP's. After several splittings, we obtain a tree of sub-CSP's.

In [13] a framework for constraint propagation based on a control-driven coordination model was presented. In this paper we extend this framework in order to integrate a distributed splitting mechanism.

Intuitively, with every split, we would like to duplicate the entire network of agents in a CSP, one replica dedicated to each resulting sub-CSP. However, this idea is not conceivable in practice because the resulting network replicas quickly exhaust any reasonable amount of resources. Thus, by correctly indexing domains of variables in sub-CSP's, and by adding some more control agents, we perform a distributed splitting, preserving the same original network of agents, their connections, and their control for propagation.

At every moment in time, each variable agent has several domains corresponding to several sub-CSP's, and each function agent can compute in several domains, reducing several sub-CSP's. Using domain indices, we can associate variable values in each domain with its respective sub-CSP, and thus we are able to select which sub-CSP is reduced. The search agent aims to coordinate function and variable agents in order to guide the search in the search space: if variables send both domains after a split and functions compute with all the domains they receive, then we obtain a breadth first search; if variables and functions focus on certain indexed domains coordinated by the search agent, then we perform a depth first search in the branch selected by the search agent.

In our framework, communication (for constraint propagation, splitting, and search) is totally asynchronous. Contrary to other methods, the split of a variable is not broadcast to all variables, but only to the concerned variable. Subsequently, the result of this split is propagated to other agents through computation of domain reduction functions.

This technique has three main advantages. First, in a single distributed and generic framework, propagation and splitting can be interleaved, and thus, complete distributed solvers can be realized. Second, by changing only the search agent, we can perform different kinds of search. Finally, splitting of variables can be dynamically triggered before the fixed point of a propagation is reached, and thus, in many cases (e.g., when reductions are not strong enough) computation of solutions can be more efficient.

2. Constraint solving and coordination languages

2.1 *Constraint solving*

As claimed in [1], many algorithms for constraint solving can be described using a simple component-based framework based on two main interleaving processes: constraint propagation and splitting (i.e., a kind of enumeration mechanism).

Constraint propagation is one of the most important techniques for solving constraint satisfaction problems (CSP's). It attempts to reduce a CSP into an equivalent but simpler one, i.e., the solution space is preserved while the search space is reduced. A CSP is given as a set of constraints, and for each variable that occurs in each constraint, a domain of values that the variable can assume, e.g., $\langle\{X+Y=Z, Z<5\}, \{X \in [0..10], Y \in [2..8], Z \in [1..17]\}\rangle$ is a CSP.

Constraint propagation algorithms usually aim at computing some form of "local consistency" described as a common fixed point of some *domain reduction functions*. These algorithms are instances of a more general mathematical framework: the framework of *chaotic iterations* (CI) [2]. CI is a basic technique used for computing limits of iterations of finite sets of functions. By "feeding" domain reduction functions into a chaotic iteration algorithm, we generate an algorithm that enforces a local consistency.

Domain reduction functions (drf's) are related to domains of constraints (see Section 5 for examples of drf's for the *and* and *not* Boolean constraints), and they have been widely studied for standard domains (e.g., Boolean constraints [9], integers, interval arithmetic [8, 12]). When considering less usual domains, these functions must either be hand-crafted, or, for finite domains, techniques such as in [4] can automate generation of the reduction functions.

However, constraint propagation is generally not strong enough to provide the user with convenient solutions. Thus, a CSP can be split into sub-CSP's (whose union contains all the solutions of the original CSP) by splitting the domain d of a variable X into several (sub)domains $d_1, \ldots, d_n$ such that $\bigcup_i d_i = d$. Thus, instead of searching for solutions in the initial CSP with d as the domain of X, we now search in n CSP's, each with its respective sub-domain for X. Since in each sub-CSP the domain of X is smaller, propagation is generally applicable again. Each of these sub-CSP's is then reduced again using propagation, split again, and so on, until convenient solutions are found. Different splitting techniques exist, such as splitting into domains of similar sizes, or labeling (i.e., enumeration) which splits the domain into a singleton, and the rest of the domain. Consider the abovementioned CSP. We can split it into the two following CSP's: $\langle\{X+Y=Z, Z<5\}, \{X \in [0..10], Y \in [2..5], Z \in [1..17]\}\rangle$ and $\langle\{X+Y=Z, Z<5\}, \{X \in [0..10], Y \in [6..8], Z \in [1..17]\}\rangle$

As soon as a CSP is split, the issue of search arises, i.e., how to explore the resulting search sub-spaces (or branches). Standard methods either explore one branch at a time (depth-first search), or all branches (breadth-first search).

Thus, by combining constraint propagation and splitting, one obtains a complete solver. In [13], a distributed version of the CDa algorithm (i.e., one algorithm for computing chaotic iterations [3]) is presented. We now extend this framework in order to integrate a distributed splitting mechanism into it.

2.2 Coordination model and language

To realize constraint propagation using a data-driven coordination model, we can consider a shared data-space (representing the variables and their values) used by agents (the reduction functions) that asynchronously post and retrieve values until they reach a fixed-point. However, we have chosen a control-driven coordination model to realize chaotic iteration techniques. Here, variables become coordinators that request computing agents to perform reductions. The reasons are quite simple. First, this allows us to easily change strategies by just changing coordinators. Second, splitting and search will become just two extra agents coordinating the variables. For more details about this choice, see [14].

Hence, we have chosen the IWIM (Ideal Worker Ideal Manager) model [5, 6] to realize our framework. The IWIM model is based on a complete symmetry between and decoupling of producers and consumers, as well as a clear distinction between the computational and the coordination/communication work performed by each process. A direct realization of the IWIM model in terms of a concrete coordination language, namely **MANIFOLD** [7], already exists.

MANIFOLD is a language for managing complex, dynamically changing interconnections among sets of independent, concurrent, cooperative processes [5]. The basics concepts in the IWIM model (thus also in **MANIFOLD**) are *processes*, *events*, *ports*, and *channels*. A **MANIFOLD** application consists of a (potentially very large) number of processes running on a network of heterogeneous hosts, some of which may be parallel systems. Processes in the same application may be written in different programming languages. **MANIFOLD** has been successfully used in a number of applications.

3. The Framework

We now explain constraint propagation, splitting mechanisms, and search techniques as coordination of cooperative agents. In this section, we thus map the main components of constraint propagation to specific IWIM processes (see [13] for more details). Splitting mechanisms and search techniques are realized by certain specific agents that either observe computation, or are queried by agents involved in propagation. The task of these two agents is thus to coordinate global computation, and to manage exploration of the search space.

3.1 Overview

We consider two types of agents and a special agent to perform constraint propagation: variables, drf's, and a Master agent. The general skeleton of the coordination-based constraint propagation framework is as follows. Each variable of the CDA algorithm is represented by one coordination variable, and each drf is represented by one worker. The Master agent builds the network of variables and drf's, and collects solutions. We do not detail the termination agent in this paper: we assume that termination of a set of agents is detected by using a standard coordination pattern of **MANIFOLD**.

The splitting mechanism is embodied in one agent which observes the states of variables (using an inquiry protocol). This agent decides which variable must be split, and how. It then communicates to the splitting variable all of its new sub-domains. Contrary to [13], a variable now has several domains, corresponding to several sub-CSP's.

In order to guide the search for solution, we consider a Search agent. Its task is to select the order for exploring sub-CSP's. An example of a network is illustrated in Figure 3.

3.2 Behavior of the network

Let us detail the process. A variable X is modified when it receives a new value for a given sub-CSP $\mathcal{P}$ whose intersection with its current domain in $\mathcal{P}$ (or in a compatible sub-CSP) is smaller than the current domain in $\mathcal{P}$. Each time a variable X is modified, it requests the drf's $F_1, \ldots, F_n$ (all the drf's containing x in their input) to work with its new domain value. $F_1, \ldots, F_n$ have thus new tasks to perform. When done, they send their results to each of the variables $X_1 \ldots, X_m$ of their output. $X_1, \ldots, X_m$ can eventually be modified by these values, and this will iterate the process.

In parallel, the Split agent observes the computation by observing the states of the variables, and may split one or more variables. In this case, a split variable is informed of its new domains corresponding to the new sub-CSP's. It is not necessary to broadcast these new CSP's to all agents in the network: the split is propagated through all the network by successive applications of drf's and modifications of variables.

When a variable X gets a value for a new sub-CSP (i.e., either X was split, or another variable was split and the split was subsequently propagated to X), it requests the Search agent to define which sub-domains must be explored first.

The process terminates when all solutions have been extracted.

3.3 Worlds and MCSP

Contrary to [13], at a given moment in time, the network of agents represents and solves several CSP's that are sub-CSP's of the initial CSP. Thus, we now solve a Multiple Constraint Satisfaction Problem (MCSP), i.e., a union of CSP's derived by splitting and constraint propagation from the original CSP that was to be solved. Note that the set of solutions of the union of such CSP's is equal to the solutions of the original CSP. Intuitively, we change the domain of computation. Previously, we had a single domain for each variable. Now we have a set of *indexed domains* for each variable. An indexed domain *(domain, world)* for a variable represents the *domain* of the variable in the sub-CSP denoted *world*.

The notion of *world* is equivalent to the notion of sub-CSP. The name of a world is a string of symbols. We denote by $\top$ the initial world, i.e., the initial CSP (before any splitting). Consider a world w. When a variable x is split into n sub-domains, w gives rise to n sub-worlds denoted respectively as $w.x_1, \ldots, w.x_n$. Furthermore, we can compare such worlds in order to perform correct reductions. Consider the world w, and the worlds $w.x_1, \ldots, w.x_n$ derived from w after the split of the variable x. We say that w is compatible with each $w.x_i$, and we denote it as $w.x_i \preceq w$. The relation $\preceq$ is a partial order: $w.x_i.y_j \preceq w$ since $w.x_i.y_j$ is derived from $w.x_i$ after the split of y, but we cannot compare $w.x_i$ with $w.x_j$, nor $w.x_i$ with $w.y_j$. We can thus easily compare worlds:

$$w' \preceq w \;\text{ iff }\; w' = w.w''$$

In a world w such that $w' \preceq w$, the domain of a variable X is larger than or equal to the domain of X in the world w'. This is obvious: the difference between w and w', if any, is because at least either X or some other variable has been split, and this may have resulted in a reduction of X in the sub-CSP w'.

Since we do not require domain reduction functions to be contracting, and that we do not enforce synchronization of reductions (a drf can send a new domain to a variable which is larger than the current domain already reduced by another function), a variable must always intersect its current domain with the new domain it receives from a drf. For this reason, two cases can arise:

- The received domain is in a world w not yet known by the variable X (the set of worlds X knows about is denoted as *Worlds*). Then, X intersects this new domain with a domain in the smallest world w' greater than w known to X, i.e., $w' = min_{\preceq}\{w'' \in Worlds \mid w \preceq w''\}$. The intersection is then the domain of X in the world w.

- The received domain is in a world w already known to the variable X. Then, for each worlds w' such that $w' \preceq w$, X must intersect this new domain with the domain in w'. Hence, we update the domain in each w' with the intersection of the domain in w and the domain in w'.

This computation is correct, because we always update a domain in a world w by intersecting it with a domain in a world w' such that $w \preceq w'$. Thus, we can be sure that we do not lose a solution. In this way, some computation may be useless (e.g., if the Search agent directs variables to different branches), but this is the price we pay to avoid a costly synchronization of all reductions performed by drf's (which would also lead to sequential reduction of the CSP). Moreover, considering a correct Search agent, all variables will work on the same world, and such useless work will not happen.

4. The agents

We now describe the different agents, their connections, their tasks, and their coordination. Tasks of agents are guarded actions of the following form:

name: guard
    actions_1, ..., action_n.

where a guard is an event, such as the reception of a message on an input port, or a notification by another agent. When the agent is in its waiting state (i.e., not executing a task), as soon as the guard guard is satisfied, the guarded action name is triggered and action_1, ... , action_n are executed sequentially to the end without interruption. The process then returns to its waiting state.

4.1 Variables

Each variable in a CSP is implemented by a *generalized variable* of MANIFOLD (i.e., extensions of variables with call back functions) whose possible domains in each world are updated via an assignment operation, i.e., one of its call back functions.

Assume a CSP $\mathcal{P}$ over a set of variables $\mathcal{X}$, and a set $\mathcal{F}$ of r domain reduction functions $f_1, \dots, f_r$. Consider x a variable in $\mathcal{X}$. Hence, the generalized variable X implementing x has the following features and connections (see Figure 1):

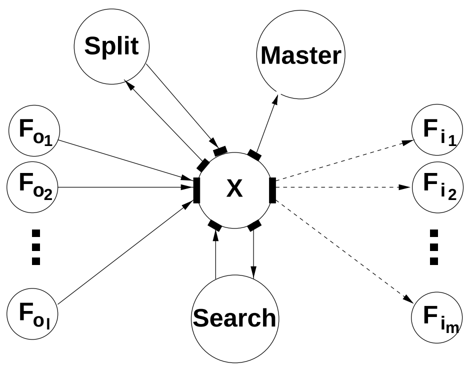

Figure 1: A variable agent

- one output port *Out* connected via channels to the implementations $F_{i_1}, \dots, F_{i_m}$ of $f_{i_1}, \dots, f_{i_m}$, i.e., the drf's that accept x as an input variable;
- one input port *In* connected via channels to the implementations $F_{o_1}, \dots, F_{o_l}$ of $f_{o_1}, \dots, f_{o_l}$ (i.e., the drf's that accept x as an output variable), and connected to the Split agent;
- one output port *OutSplit* connected via a channel to the Split agent in order to send it the current state of the variable;
- an output port *OutMaster* for forwarding to the Master agent the indexed domains of the variable when a solution is computed;
- an output port *OutSearch* to query the Search agent to select which current worlds (i.e., branches of the search space) must be explored;

- a set *Domains* of indexed domains to keep current domains and the worlds they are associated with;
- a set *Worlds* to store the worlds known by X. This set is used by the Search agent to select on which world X must concentrate;
- a set *Splits* to store the worlds known by X, and resulting after a split of X. This store is used for detecting when all solutions have been computed;
- a set *SearchWorlds* to store the worlds that are currently being explored (i.e., worlds that have been selected by the Search agent); and
- three call back functions: a domain intersection function, a domain comparison function, and an assignment function.

We now describe the guarded actions of X. Note that when sending a message on a port with multiple connections, the message is replicated into each connected channel.

Updating domains When the variable X gets an indexed domain (v', w') from one of the DRF's $F_{o_1}, \ldots, F_{o_l}$, the world w' may be unknown for X. This happens if the indexed domain (v', w') for X was computed in a DRF using other variables known in w' (i.e., w' results from the split of variables other than X).

When X encounters w' for the first time, it adds w' in its set *Worlds* of known worlds, and requests the Search agent to select which set of worlds (i.e., branches of the search space) must be explored. Note that X does not wait for the new decision of the Search agent before continuing. This can lead to useless work, e.g., when looking for a single solution. Nevertheless, this scheme avoids a costly synchronization with the Search agent (we can however consider this synchronization when looking for a single solution). Then, the new indexed domain must be integrated in the set of indexed domains of X after intersecting the domain with the domain of the "smallest" world "bigger" than w'.

When w' is already known, the domain of each world that is "smaller" than or "equal" to w' must be updated. For each such world w, X intersects the current domain with v'. When the result is different than the current domain, the current domain in the world w is updated, and consequently, X requests all the DRF's $F_{i_1}, \ldots, F_{i_m}$ to execute again.

These tasks are realized by the following guarded action:

```
update: $(v',w')$ on input port $Inp$
    if $w' \notin Worlds$
        % the world $w'$ is unknown to $X$
        then $Worlds := Worlds \cup \{w'\}$
                % request Search to define the search
                send $Worlds$ on output port $OutSearch$
                % intersects the domain $v'$ in $w'$ with the domain of the
                % "smallest" world known by $X$ strictly "greater" than $w'$
                $(v, w)$ s.t. $w = min_{\preceq}\{w'' \in Worlds \mid w' \preceq w''\}$
                $Domains := Domains \cup \{ (v' \cap v, w') \}$

        % the world $w'$ is already known by $X$
        else % eventually modifies domains of worlds known by $X$ "smaller"
                % than or "equal" to $w'$ using new domain in $w'$
                foreach $(v,w) \in Domains$ s.t. $w \in Worlds$ and $w \preceq w'$
                do $v'' := v \cap v'$
                    % updates the domain of $X$ in $w$ if modified
                    if $v'' \neq v$
                    then $Domains := Domains \setminus \{(v,w)\}$
                         $v = v''$
                         $Domains := Domains \cup \{(v,w)\}$
                         % requests associated DRF's if search is set for w
                         if $w \in SearchWorlds$
                            then send $(v,w)$ on port $Out$
                         fi
                  fi
              od
      fi
```

Note that updating domains of a world "smaller" than or "equal" to w' can be optimized by considering worlds in decreasing order: if a world is not modified, smaller worlds cannot be modified.

Splitting When the Split agent decides to split the variable X, it sends to X a set V of new indexed domains. X must then update the set of worlds it knows about, its set of indexed domains, and its set of splits.

```
split: $V$ on input port $InpSplit$
      $Domains := Domains \cup V$
      $Worlds := Worlds \cup \{w' \mid (v',w') \in V\}$
      $Splits := Splits \cup \{w' \mid (v',w') \in V\}$
```

Since splitting and reductions are asynchronous, we keep in *Domains* the domain of X in the world that has just been split. If we don't keep this information, the following problem can arise: consider x the domain of X in $\top$, x' the domain of X in $\top$ after reduction by the drf d_1, and x'_1, x'_2 domains of X after a split on x'. Then, consider y the domain of Y in $\top$, and y_1 a domain of Y after a split of Y from $\top$. Then, a drf d_2 computing with x and y_1 can compute a domain x'' larger than x' for X in the world of y_1. If we don't keep x', we cannot intersect x'' with any domain, and thus we obtain the domain x'' for X in the world of y_1. If we consider that domain reduction functions are always contracting domains, this is not a real problem: x is larger than or equal to x'', some reductions may be lost, but they will be computed again from x'', and we will not loop. But if we don't require any special properties for domain reduction functions (and this is the case in our framework), then we cannot ensure termination anymore, because x'' can be larger than x.

Orienting the search The next guarded action collects a set of worlds W sent by the Search agent. W represents the worlds on which a variable will focus in its next steps:

```
search: $W$ on input port $InpSearch$
      $SearchWorlds := W$
```

Jumping to another sub-space of the search When the solutions in the worlds defined by the Search agent are found, the Termination agent (see Section 4.5) raises the "solution_ack" event to inform all the variables. Then, the variable X sends its domains for the current worlds to the Master agent, updates its own structures (especially *Splits* that is used for termination detection), and requests the Search agent to change its set of worlds to concentrate on:

```
solution: solution_ack
      foreach $w \in SearchWorlds$
         do find $(v,w)$ in $Domains$
            send $(v,w)$ on output port $OutMaster$
            $Domains := Domains \setminus \{(v,w)\}$
            $SearchWorlds := SearchWorlds \setminus \{w\}$
```

```
        $Worlds := Worlds \setminus \{w\}$
        $Splits := Splits \setminus \{w\}$
    od
  % *asks for some new worlds to search in*
  send Worlds on output port $OutSearch$
```

Reporting state The Split agent observes the variables and their domains in order to decide which variable to split, and when. For this purpose, it needs to be informed of the state of the variables, i.e., their domains, and their *SearchWorlds*, in order to work in cooperation with the Search agent:

```
state_requested: domain? on input port $InSplit$
    send $Domains$ on output port $OutSplit$
    send $SearchWorlds$ on output port $OutSplit$
```

4.2 Domain reduction functions
A DRF implements a domain reduction function given as input to the CDA algorithm. Thus, as many DRF's as drf's fed in the CDA algorithm are created by the Master agent.

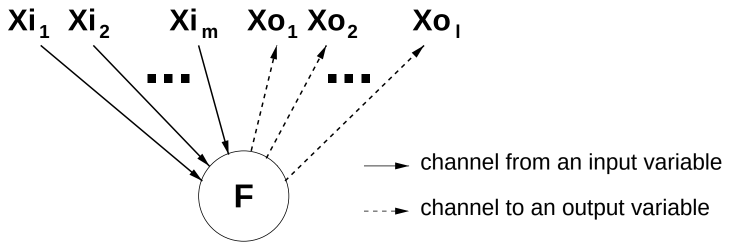

Figure 2: a DRF agent

Assume a CSP over a set $\mathcal{X}$ of variables, and a drf f such that:

$$f : xi_1, ..., xi_m \longrightarrow xo_1, \dots, xo_l$$

where $xi_1, ..., xi_m, xo_1, \dots, xo_l$ are variables in $\mathcal{X}$. Then, the DRF F implementing the function f has the following structure (see Figure 2):

- m input ports $Inp\_Xi_1, \dots, Inp\_Xi_m$ connected respectively to the m variable processes $Xi_1, \dots, Xi_m$ implementing $xi_1, \dots, xi_m$;
- l output ports $Out\_Xo_1, \dots, Out\_Xo_l$ connected, respectively, to the l variable processes $Xo_1, \dots, Xo_l$ implementing $xo_1, \dots, xo_l$;
- m sets $Domains\_Xi_1, \dots, Domains\_Xi_m$ of indexed variables for each input variable. These stores are initialized with the initial domains (i.e., in the $\top$ initial world) of input variables during the creation of F by the Master agent;
- m sets $Worlds\_Xi_1, \dots, Worlds\_Xi_m$ of worlds known for the input variables;
- the code of the function f,

We now present the guarded actions a *DRF* can execute.

Reduction request When a DRF receives a reduction request from X_i, one of its input variables (i.e., it receives a new indexed domain (v,w) for this variable), it updates the domain of X_i: either the new indexed domain is added to the set $Domains_{X_i}$ (when the DRF does not know yet the world w for X_i), or the indexed domain is replaced in the set $Domains_{X_i}$. The DRF then selects the smallest w_compatible domains of each other input variables. Then, it reduces its output variables using the drf it represents. Finally, it sends their new indexed domains to each of its output variables. For the DRF F, mentioned above, we obtain the following guarded action for each of its input variables:

```
reduction_request: $(v,w)$ on input port $Inp\_Xi_j$
        % Update of the indexed domains of $Xi_j$
        if $w \in Worlds\_Xi_j$
           then find $(v',w')$ in $Domains\_Xi_j$ s.t. $w' = w$
                   $Domains\_Xi_j := Domains\_Xi_j \setminus \{(v',w')\}$
           else  $Worlds\_Xi_j := Worlds\_Xi_j \cup \{w\}$
        fi
        $Domains\_Xi_j := Domains\_Xi_j \cup \{(v,w)\}$
        % Find the smallest compatible domains for each input variable
        $I_j = v$
        foreach $k \in [1..m]$ s.t. $k \neq j$
            do $w'' = min_{\preceq}\{w \in Worlds\_Xi_k \mid w \preceq w'\}$
                find $(v'',w'')$ in $Domains\_Xi_k$
                $I_k = v''$
            od
        % Compute new domains for output variables using drf
        $(O_1,\ldots,O_l) = drf(I_1,\ldots,I_m)$
        % send new indexed (by w) domains of output variables
        foreach $k \in [1..l]$ sends $(O_k,w)$ on output port $Out\_Xo_k$
```

Note that we can optimize this guarded action by 1) directly updating the set $Domains\_X_i$ and reducing again new domains when the output variable X_i is also an input variable and the drf is not idempotent, and 2) sending new indexed domains only to variables that have effectively been modified.

Initializing the propagation process Each function must be applied at least once, i.e., in the CDA algorithm the set G of functions still to be applied is initialized with the set F of drf's. We consider the *start* event raised by the Master agent when the network is installed. We can start reduction with the initial domains (i.e., domains indexed by the initial world $\top$) of $Xi_1,\ldots,Xi_m$ given at creation time with the following guarded action:

```
starting: start
        % find initial domains of input variables
        foreach $j \in [1..m]$
            do find $(v'',\top)$ in $Domains\_Xi_j$
                $I_j = v''$
            od
        % compute and send new indexed domains to output variables
        $(O_1,\ldots,O_l) = drf(I_1,\ldots,I_m)$
        foreach $j \in [1..l]$ sends $(O_j,\top)$ on output port $Out\_Xo_j$
```

4.3 The Split agent

This agents aims to dynamically (i.e., depending on the progress of constraint propagation) split the domain d of a variable into several (sub)domains such that their union is equal to d. In terms of constraint solving, this means that a CSP $\mathcal{P}$ is split into several sub-CSP's such that their union is equivalent to (i.e., correctness and completeness of the set of solutions) $\mathcal{P}$.

This agent observes all the variables (by periodically inquiring their states), analyzes the evolution of the propagation, and decides which variable must be split, and splits the variable. Thus, this agent needs a local memory in order to store pieces of global information involved in its decision making.

The Split agent is connected to each variable with two streams: one to receive the states of variables, the other, to request states, and to send the split domains. Moreover, this agent is linked to the Termination agent that detects its termination, and informs it when constraint propagation for some branch of the search-space is finished. Hence, the Split agent is able to establish one of its strategies: either wait for the end of propagation before splitting, or split as soon as domain reduction becomes tedious and slow.

We consider the Split agent as a coordinator, not as a simple worker. By this we mean that the termination of the network of agents depends on a correct and sensible agent, i.e., an agent able to take beneficial decisions, e.g., not to split numerous variables simultaneously, not to split again the same variable before noticing the effect of the previous split, etc.

We don't give the details of this agent because, depending on the desired splitting strategies (see Section 6), numerous algorithms are possible.

4.4 The Search agent

The task of this agent is to dynamically determine how the search space must be explored. When variables encounter a new world, they ask the Search agent to decide which branches (worlds) must be exploited first. Since this agent collects information from variables, the Split agent, and the Termination agent, it can also lead the search without being queried by any variable.

The behavior of this agent depends not only on the evolution of the computation in the network of agents, but also on the needs of the end user: should only one solution be computed, several, all, or only some with specific given properties? Depending on these strategies and requirements, the Search agent then decides whether functions and variables must use all their domains in different worlds (this implements a breadth first search), one single domain at a time (this implements a depth first search), or several domains simultaneously (i.e., a mixed search).

In order to manage the search of solutions, and the exploration of the search space, the Search agent is connected to every variable to receive the set of worlds they know about, and to inform them which branches to explore. This coordinator is not linked to the Termination agent, since as soon as a branch is completely explored the Termination agent will be warned indirectly by the variables involved through a decision request. However, we can imagine connecting it to the termination agent in order to avoid some intermediary data exchange, and to the Split agent in order to work in tighter collaboration with it. But then, the decoupling of propagation, termination, splitting, and search becomes less obvious.

We assume that this agent takes correct and compatible decisions for each variable, i.e., its decisions are global and not only local. An agent that forces a variable to explore a world, and another variable to concentrate on another (non-compatible) world can lead to non-termination: some solutions can be computed, but we can never be sure to compute all solutions of the initial CSP.

4.5 The Termination agent

This agent is responsible to detect four types of termination. Note that the framework we present is very generic and allows numerous strategies, explorations of the search space, and splitting techniques. Thus, detection of termination in such a case assumes that the Split agent and the Search agent are mutually correct, i.e., they do not loop, and they give compatible information (such as *SearchWorlds*) to every variable.

Termination of propagation Constraint propagation terminates when 1) no domain reduction function is busy anymore, 2) no variable is busy anymore with domains associated with worlds they must currently concentrate on (i.e., worlds of *SearchWorlds* as determined by the Search agent), and 3) no message is pending in a stream between a variable and a domain reduction function. When constraint propagation is finished, the Termination agent signals it to the Split agent, because the latter may need this information to fully realize its strategies.

Current branches totally explored This case happens when the solutions in one/several branches have been found. This means that propagation for these branches is finished, and the Split agent can no longer split their corresponding domains of variables. This termination is thus detected when both propagation and the Split agent terminate.

The whole search space has been explored In this case, all solutions of the initial CSP have been computed. The Termination agent detects this state when current branches are totally explored and the *Splits* set of every variable (worlds directly derived from a split of the variable itself) is empty.

Termination required by the Master agent When a user is not interested in all solutions, the Master agent can decide to stop resolution as soon as a/several satisfying solution(s) have been collected. In this case, the Master agent can request the Termination agent to stop all activity in the network of agents. Note that this feature can be used for optimization when computing a "good" solution (not necessarily the best) within a given time, or when the ratio of solution quality over elapsed time becomes sufficient.

We do not give here the details of the mechanism used by this agent. Informally, using features of MANIFOLD, we can easily implement a generic termination protocol scanning activities of agents, and the presence of messages pending in streams. We just require this agent to be connected to variables, drf's, the Split agent, and to be able to observe streams inbetween these agents.

4.6 The Master agent

The task of this agent is rather static. The Master agent is mainly concerned with initialization and creation of the network of agents, and collecting solutions. Given a CSP, a set of "meta" domain reduction functions, a search algorithm, and a split algorithm, the Master agent derives the drf's needed for the resolution of the CSP, and establishes the network of agents with their connecting streams. Moreover, the Master agent is connected to each variable (to receive its values when necessary), and to the termination agent (to stop the resolution when sufficently many solutions or sufficiently good solutions have been computed).

5. An example

We now illustrate our framework by solving an example of Boolean constraints. We assume three types of constraints: 1) $and(x, y, z)$ with the usual meaning $z = x \wedge z$, 2) $not(x, y)$ meaning $x = \overline{y}$, and 3) the standard equality "=". We now consider solving the CSP:

$$\langle\{and(x, y, z), not(x, z), y = t\}, \{x \in \{0, 1\}, y \in \{0, 1\}, z \in \{0, 1\}, t \in \{0, 1\}\}\rangle$$

The Master agent creates 4 generalized variables X, Y, Z, and T implementing x, y, z, and t, with initial domains $\{0, 1\}$. Assume that, using some meta domain reduction functions, the Master agent identifies 5 drf's f_1, f_2, f_3, f_4, and f_5[1]:

[1]To simplify the example, we consider here only 5 functions, a subset of the complete set of functions that can be automatically generated using algorithms such as in [4]. Constraint propagation based on these four functions, together with the Split agent, are sufficient for solving our example CSP.

$$\begin{array}{lll}
f_1: & y \rightarrow z & \\
 & f_1(y) := \{0\} & \text{if y=\{0\}} \\
 & f_1(y) := \{0,1\} & \text{otherwise} \\[6pt]
f_2: & z \rightarrow x & \\
 & f_2(z) := \{1\} & \text{if z=\{0\}} \\
 & f_2(z) := \{0\} & \text{if z=\{1\}} \\
 & f_2(z) := \{0,1\} & \text{otherwise} \\[6pt]
f_3: & t \rightarrow y & \\
 & f_2(t) := t & \\[6pt]
f_4: & x, z \rightarrow y & \\
 & f_4(x,z) := \{0\} & \text{if x=\{1\} and z=\{0\}} \\
 & f_4(x,z) := \emptyset & \text{if x=\{0\} and z=\{1\}} \\
 & f_4(x,z) := \{0,1\} & \text{otherwise} \\[6pt]
f_5: & x \rightarrow z & \\
 & f_5(x) := \{1\} & \text{if x=\{0\}} \\
 & f_5(x) := \{0\} & \text{if x=\{1\}} \\
 & f_5(x) := \{0,1\} & \text{otherwise}
\end{array}$$

Consider now that we want to perform a depth first search, and we want to perform a complete

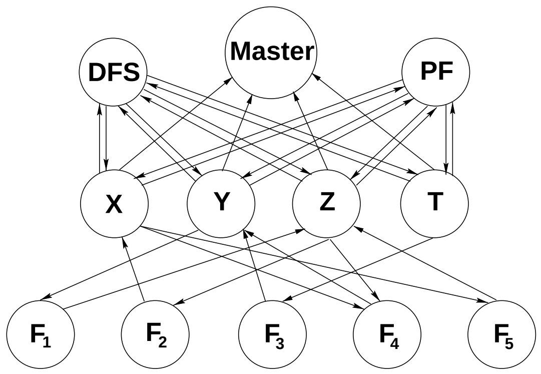

Figure 3: A Boolean Example

constraint propagation before splitting. We thus consider DFS as the Search agent that implements a depth first search, and PF as the Split agent that waits for the termination of propagation before splitting. We obtain the network illustrated in Figure 3. The agents F_1, F_2, F_3, F_4, and F_5 (respectively, X, Y, Z, and T) implement the functions f_1, f_2, f_3, f_4, and f_5 (respectively, the variables x, y, z, and t) as described in the previous section.

As soon as the start event is raised by the Master agent, the reduction process starts. Each function is applied only once, since none of them is able to modify the domains of the variables. Thus, propagation terminates without any change of domain.

Assume now that PF decides to split the variable T into $\{0\}$ in the world of T_1 and $\{1\}$ in the world of T_2. When requested by T, DFS has two possibilities: explore the world T_1 or the world T_2. Assume DFS first selects T_1. Then, T forwards $\{0\}$ to F_3, and the reduction starts again. When the propagation finishes, PF also terminates (no other variable can be split), and we obtain the first solution: $X = \{1\}$, $Y = \{0\}$, $Z = \{0\}$, and $T = \{0\}$. The Termination agent detects that current branches have been fully explored, and raises the solution_ack event. Variables catch this event, and thus forward (their respective parts of) this solution to the Master.

Next, the variables again query DFS, which can now decide to explore the world T_2. After propagation, the domains of T and Y are fixed to $\{1\}$. Another splitting is required for, say, X. We obtain two new worlds: the world of $T_2.X_1$ with the X value $\{0\}$, and $T_2.X_2$ with the X value $\{1\}$.

Suppose DFS selects $T_2.X_2$. Then, after the fixed point of propagation is reached, and no other split is possible, we obtain $X = \{1\}$, $Y = \emptyset$ (since F_4 deduces that $Y = \{0\}$, and then the Y agent intersects $\{0\}$ with $\{1\}$), $Z = \{0\}$, and $T = \{1\}$. This branch is totally explored, and the domains are forwarded to the Master, which deduces that this branch leads to an inconsistency (i.e., no valid solution since $Y = \emptyset$). DFS then selects $T_2.X_1$, which also leads to an inconsistency. At this point, the entire search space has been explored, and the computation stops.

6. Comments

We now discuss some important advantages of our framework with regards to its generality, its component-based aspect, and its dynamic behavior.

Interleaving of propagation and splitting In [13], a coordination-based chaotic iteration algorithm is presented. However, constraint propagation is generally not powerful enough to solve a CSP, i.e., reduction alone is not able to narrow domains of variables to singletons. Thus, such a framework cannot generally extract solutions. In this paper, we extend this framework and integrate splitting mechanisms and search techniques in a distributed environment, without requiring any synchronization, or any mutual exclusion among tasks. We thus obtain a single distributed and generic framework, in which propagation and splitting can be interleaved in order to realize complete distributed solvers. Furthermore, the decoupling of constraint propagation, splitting mechanism, and search technique is total. Thus, we can envisage designing in our framework most of the usual strategies realized for sequential computation at low cost: strategies are numerous, but they are all based on a mix of a small number of different splitting mechanisms and search techniques. We can also tackle new strategies, such as the ones based on the simultaneous exploration of several sub-spaces (worlds). Finally, we consider propagation and splitting at the same level, similarly to the sequential framework of [10].

Different types of splitting mechanisms Depending on the Split agent we plug in the network, we can realize several types of splitting strategies, such as:

- splitting a domain into two (or more) domains of the same size. This is a good strategy when we consider an even probability of solution containment in each zone of the domain, or when we want to favour propagation and avoid enumeration.
- splitting a domain into one value and the rest of the domain. This strategy is also known as "labeling" or enumeration. It is generally useful in search for one solution, when performing a depth-first search, or when one assignment can significantly ease reduction.
- shaving technique: a domain is split into three sub-domains, two narrow ones to include the bounds, and one for the rest. This is especially efficient when a domain reduction function is used that can push the left and the right bounds of the interval, until a local solution is reached. Then, one can hope that the global solutions are close to the bounds.

Different types of search By changing only one agent, we can perform different kinds of search, either a usual search such as depth-first or breadth-first, or unusual searches that our framework makes possible, such as simultaneously exploring several branches.

To perform a depth-first search, the Search agent selects a single world on which variables and domain reduction functions will focus. Breadth first search is realized by letting variables and function to simultaneously work on every world they know about. Searching in several branches is similar, but the Search agent selects a subset of all the worlds that are known at a given point in time.

In depth first (or when exploring several branches), we have the possibility of changing the branch, if desired. For instance, when exploring a branch takes too long (due to slow reduction), then the Search agent can decide to jump to another branch in order to extract a solution more quickly.

Splitting before termination of propagation It is generally accepted that splitting a CSP before reaching the termination of propagation can significantly improve the resolution speed (e.g., when reduction is converging slowly [11], it is generally better to split a variable first and then reduce each sub-CSP). Our framework is generic enough to allow this type of strategy. The Split agent gets enough information from variables to analyze the convergence, and it is free to act before the end of propagation. Thus, splitting of variables can be dynamically triggered if necessary before the fixed point of propagation is reached.

7. Conclusion and future works

In this paper we extend the framework of [13] in order to integrate a distributed splitting mechanism. This technique has three main advantages: 1) propagation and splitting can be interleaved in order to realize complete distributed solvers, 2) agents are decoupled, and thus by changing only one agent, we can perform different kinds of search and split, and 3) splitting of variables can be dynamically triggered before the fixed point of a propagation is reached.

We plan to establish the minimal properties jointly required of the Search and the Split agents (i.e., a kind of mutual agreement) in order to be able to ensure termination. This must ensure termination even for special cases such as when several branches are simultaneously explored, search sub-spaces to be explored are changed, variables not currently being explored split, an arbitrary number of search sub-spaces split, etc.

We also plan to tackle optimization problems by using the dynamic features of **manifold** and the properties of specific Search agents. When a solution is extracted, a new constraint (i.e., its reduction functions) can be added to state that the next solution must be better than the one just extracted.

Finally, we plan to extend this framework to constraint reduction, i.e., adding, changing, and removing constraints, and thus adding, changing and removing drf's. This is crucial when considering symbolic transformation of constraints (such as simplification of constraints, and addition of redundancies that can speed-up propagation) during propagation. This will open some other forms of splitting strategies such as partioning the search space with additional constraints (e.g., $X < Y$ or $X \geq Y$).

References

1. K. R. Apt. Component-based framework for constraint programming. Manuscript, 1999.
2. K. R. Apt. The Essence of Constraint Propagation. *Theoretical Computer Science*, 221(1–2):179–210, 1999.
3. K. R. Apt. The Rough Guide to Constraint Propagation". In J. Jaffar, editor, *Proc. of the 5th International Conference on Principles and Practi ce of Constraint Programming (CP'99)*, volume 1713 of *Lecture Notes in Computer Science*, pages 1–23. Springer-Verlag, 1999. Invited lecture.
4. K. R. Apt and E. Monfroy. Automatic Generation of Constraint Propagation Algorithms for Small Finite Domains. In J. Jaffar, editor, *Proceedings of Fifth International Conference on Principles and Practice of Constraint Programming, CP'99*, volume 1713 of *Lecture Notes in Computer Science*, pages 58–72, Alexandria, Virginia, USA, October 1999.
5. F. Arbab. Coordination of massively concurrent activities. Technical Report CS–R9565, CWI, Amsterdam, The Netherlands, November 1995. Available on-line `http://www.cwi.nl/ftp/CWIreports/IS/CS-R9565.ps.Z`.
6. F. Arbab. The IWIM model for coordination of concurrent activities. In Paolo Ciancarini and Chris Hankin, editors, *Coordination Languages and Models*, volume 1061 of *Lecture Notes in Computer Science*, pages 34–56. Springer-Verlag, 1996.
7. F. Arbab. *Manifold2.0 reference manual.* CWI, Amsterdam, The Netherlands, May 1997.
8. F. Benhamou and W. Older. Applying interval arithmetic to real, integer and Boolean constraints. *Journal of Logic Programming*, 32(1):1–24, March 1997.
9. P. Codognet and D. Diaz. A simple and efficient Boolean constraint solver for constraint logic programming. *Journal of Automated Reasoning*, 17(1):97–128, 1996.
10. L. Granvilliers. Résolution approchée de contraintes réelles par transformations symboliques et consistance de bloc. *Technique et Science Informatiques*, 18(2):209–232, 1999.
11. O. Lhomme, A. Gotlieb, and M. Rueher. Dynamic Optimization of Interval Narrowing Algorithms. *Journal of Logic Programming*, 37(1–2):165–183, 1998.
12. E. Monfroy. Using "Weaker" Functions for Constraint Propagation over Real Numb ers. In J. Carroll, H. Haddad, D. Oppenheim, B. Bryant, and G. Lamont, editors, *Proceedings of The 14th ACM Symposium on Applied Computing, ACM SAC'99, Scientific Computing Track*, pages 553–559, San Antonio, Texas, USA, March 1999.

13. E. Monfroy. A Coordination-based Chaotic Iteration Algorithm for Constraint Propagation. In J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim, editors, *Proceedings of the 2000 ACM Symposium on Applied Computing (SAC'2000)*, pages 262–269, Villa Olmo, Como, Italy, March 2000. ACM Press.

14. E. Monfroy and F. Arbab. *Constraints Solving as the Coordination of Inference Engines*, chapter in "Coordination of Internet Agents: Models, Technologies, and Applications". Springer-Verlag, 2000. To appear.