Feasible Distributed CSP Models for Scheduling Problems

Miguel A. Salido, Adriana Giret
Universidad Politecnica de Valencia
Camino de Vera s/n, 46022 - Valencia (Spain)

Abstract

A distributed constraint satisfaction problem (DisCSP) is a CSP in which variables and constraints are distributed among multiple automated agents. Many researchers have developed techniques for solving DisCSPs. They assume for simplicity that each agent has exactly one variable. For real planning and scheduling problems, these techniques require a large number of messages passing among agents, so these problems are very difficult to solve. In this paper, we present a general distributed model for solving real-life scheduling problems. This distributed model is based on the idea of Holonic Systems. Furthermore, we propose some guidelines for distributing large-scale problems. Finally, we present two case studies in which two scheduling problems are distributed by using our model.

Key words: distributed CSP, constraint satisfaction, holonic system, multi-agent system, scheduling.

1. Introduction

In recent years, we have seen increasing interest in Distributed Constraint Satisfaction Problem (DisCSP) formulations to model combinatorial problems (see the special issue on Distributed Constraint Satisfaction in the Artificial Intelligence journal, vol 161, 2005). There is a rich set of real-world distributed applications, such as network systems, planning, scheduling, resource allocation, etc., for which the DisCSP paradigm is particularly useful. In such distributed applications, privacy issues, knowledge transfer costs, robustness against failure, etc, preclude the adoption of a centralized approach (Faltings and Yokoo, 2005).

Briefly, a Constraint Satisfaction Problem ***CSP*** consists of the following:

- a set of variables $X = \{x_1, x_2, ..., x_n\}$;
- each variable $x_i \in X$ has a set D_i of possible values (its domain);
- a finite collection of constraints $C = \{c_1, c_2, ..., c_p\}$ restricts the values that the variables can simultaneously take.

A solution to a CSP is an assignment of values to all the variables so that all constraints are satisfied. A problem with a solution is termed *satisfiable* or *consistent*.

A typical example of a CSP is the graph coloring problem. The objective is to color each region so that adjacent regions have different colors (Figure 1). A problem of this kind is called a constraint satisfaction problem since the objective is to find a configuration that satisfies the given conditions (constraints). Even though the definition of a CSP is very simple, a surprisingly wide variety of Artificial Intelligence problems can be formalized as CSPs. Therefore, the research on CSP has a long and distinguished history in Artificial Intelligence (Mackworth, 1992).

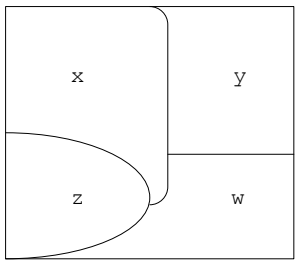

Fig. 1. Example of a Constraint Satisfaction problem (graph-coloring).

For example, in the graph coloring problem presented in Figure 1, it is obvious that region x must be colored with a different color than region y, and so on. Therefore, we can formalize this problem as a CSP, in which there are four variables x, y, z, w each of which corresponds to a regions. The domain of each variable is the set of available

[1] Tel.: +34 963877007; fax: +34 963877359.
E-mail addresses: msalido@dsic.upv.es (corresponding author), agiret@dsic.upv.es

colors: red, blue, green. The constraints that force that the adjacent regions must be colored with different colors are $x \neq y$, $x \neq z$, $x \neq w$, $y \neq w$.

CSP (graph coloring problem)
Variables: $\{x, y, z, w\}$
Domain of each variable: $\{red, blue, green\}$
Constraints: $x \neq y$, $x \neq z$, $x \neq w$, $y \neq w$.

A solution is a valid combination of values for these variables: $x = red$, $y = blue$, $z = blue$ and $w = green$.

A distributed CSP is a CSP in which the variables and constraints are distributed among automated agents (Yokoo and Hirayama, 2000). Finding a value assignment to variables that satisfies inter-agent constraints can be viewed as achieving coherence or consistency among agents.

The above example of the graph coloring problem can be modeled as a distributed CSP, where each variable in the resulting DisCSP is owned by one particular agent who ensures that the variable has a value assigned to it. Thus, agent x owns variable x and so on (see Figure 2). The actual search for solving the DisCSP can be carried out by a central agent, or in a distributed manner through message exchange among the agents. This approach, which is illustrated in Figure 2, was pioneered by Yokoo et al. (Yokoo *et al.*, 1992)(Yokoo *et al.*, 1998*b*) in their asynchronous backtracking algorithm. In asynchronous backtracking, the DisCSP is solved by asynchronous message exchange. It assumes a priority ordering among agents (e.g., a unique serial number assigned to each agent) and an agent is responsible for enforcing all constraints between itself and all variables owned by higher agents in this ordering. The problem is solved through an exchange of messages that does not need to be synchronized among agents.

The advantage of distributed search is that an agent only needs to know about agents that own a variable that it has a constraint with, but not about the entire problem. Thus, in the example of Figure 2, the agent that owns variable z only has to communicate with the agents that owns x and w, but not with the agent that owns y. In certain algorithms, it can happen that new connections are created dynamically during search; thus, z might become connected to y. The approach is most suitable when the overall problem is large, but not very densely connected. However, when the problem is densely connected or the number of variables is very high, the cost of communication during the solving process is likely to become very high, too.

The most cited papers related to DisCSP make the following assumptions for simplicity in describing their algorithms:

(i) Each agent has exactly one variable.
(ii) All constraints are binary.
(iii) Each agent knows all constraint predicates that are relevant to its variable.

Although the great majority of real problems are naturally modelled as non-binary CSPs, the second assumption is comprehensible due to the fact that there exist some techniques that translate any non-binary CSP into an equivalent binary one (Bacchus and van Beek, 1998).

However, the first assumption is too restrictive, and the main basic research focuses on small instances. Also, little work has been done to solve real-life problems.

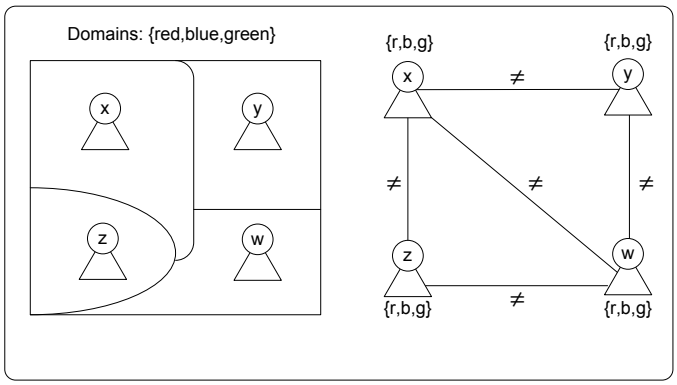

Fig. 2. Example of a Distributed Constraint Satisfaction problem (graph-coloring). Variables are nodes and constraints are arcs.

2. Main Features in DisCSPs

In this section, we present the main features that make the use of DisCSPs appropriate. It is well known that if all knowledge about the problem could be gathered into one agent, this agent could solve the problem alone using traditional centralized constraint satisfaction algorithms. However, such a centralized solution is often inadequate or even impossible. Faltings and Yokoo (Faltings and Yokoo, 2005) present some reasons why distributed methods may be desirable:

– The cost of creating a central authority. A CSP may be naturally distributed among a set of agents. In such cases, a central authority for solving the problem would require adding an additional element that was not present in the architecture. Examples of such systems are sensor networks or meeting scheduling.
– The knowledge transfer costs: In many cases, constraints arise from complex decision processes that are internal to an agent and cannot be articulated to a central authority. Examples of this range from simple meeting scheduling, where each participant has complex preferences that are hard to articulate, to coordination decisions in virtual enterprises that result from complex internal planning. A centralized solver would require such constraints to be completely articulated for all possible situations. This would entail prohibitive costs.
– Privacy/Security concerns: Agents involve constraints that may represent strategic information that should not be revealed to competitors, or even to a central authority. This situation often arises in many enterprises. Privacy is easier to maintain in distributed solvers.
– Robustness against failure: The failure of the centralized server can be fatal. In a distributed method, a failure

of one agent can be less critical and other agents might be able to find a solution without the failed agent. Such concerns arise, for example, in sensor networks, but also in web-based applications where participants may leave while a constraint solving process is ongoing.

These reasons have motivated significant research activity in distributed constraint satisfaction. Up to now, the field has reached a certain maturity and has developed a range of different techniques. Nevertheless, most of the works are focused on developing new techniques which are evaluated using toy problems and random benchmarks.

3. Open Issues in DisCSPs

In spite of significant progress, there are many important open issues in distributed CSPs. The six main open issues for using distributed CSPs are the following:

- While distributed algorithms eliminate the need for a central authority, the currently known algorithms pay a high price in efficiency. In general, the message traffic even for a single agent can be higher than what would be required to communicate the entire problem to a leader agent that could solve it centrally. More research is required to significantly reduce the communication requirements, possibly with radically different algorithms that are better suited for distribution.
- Many DisCSP algorithms assume an agent has enough knowledge to evaluate constraints that are related to its variables. If this is not true, some constraints may still have to be communicated or additional communication may be needed. Also, more research is needed on algorithms that minimize the number of constraint evaluations when evaluating constraints is costly.
- While there are algorithms using cryptographic techniques to ensure complete privacy of agent constraints, their message complexity is very high. For most other DisCSP algorithms, there is no good characterization of how much information is revealed to other agents. More research is needed on measures of privacy loss and on algorithms that balance the trade-off between privacy loss and efficiency.
- While most distributed algorithms tolerate certain kinds of agent failures, there is no good characterization of the kind of failures that are allowed for each algorithm. In general, this issue has not yet been given significant attention in research.
- While most distributed algorithms manage only one variable per agent, there are no specific distributed algorithms to solve real-world problems in an efficient way. This issue is currently receiving more attention in research. This paper focuses on this issue.
- While distributed algorithms are only designed for solving distributed problems, many real and centralized problems can be modeled as a distributed CSP and can be solved more efficiently using distributed search techniques. Usually, real problems imply models with a great number of variables and constraints, causing dense networks of inter-relations. Problems of this kind can be handled as a whole only at overwhelming computational cost. Thus, it could be an advantage to decompose/partition the problem into several simpler interconnected sub-problems which can be more easily solved. This issue has not yet been given enough attention in research. This paper also focuses on this issue.

Other issues that are important but have received little attention so far include openness, i.e., the possibility to add and remove agents dynamically during execution, and incentive-compatibility, i.e., making algorithms safe against manipulation by self-interested agents.

4. Non-efficient Traditional Methods

One of the pioneer researchers in DisCSP said "*So far, we assume that each agent has only one local variable. Although the developed algorithms can be applied to the situation where one agent has multiple local variables by the following methods, both methods are neither efficient nor scalable to large problems*" (Yokoo and Hirayama, 2000).

The following two general methods have been traditionally used to manage several variables per agent. Both methods have serious drawbacks that make them inefficient for solving real-life problems.

- Method 1: each agent finds all solutions to its local problem first. By finding all solutions, the original problem can be re-formalized as a distributed CSP, in which each agent has one local variable whose domain is a set of obtained local solutions. Then, agents can apply algorithms for the case of a single local variable.

  The main drawback of this method is that when a local problem becomes large and complex, finding all the solutions of a local problem becomes virtually impossible.
- Method 2: an agent creates multiple virtual agents, each of which corresponds to one local variable and simulates the activities of these virtual agents.

  However, since communicating with other agents is usually more expensive than performing local computations, it is wasteful to simulate the activities of multiple virtual agents without distinguishing the communications between virtual agents within a single real agent, and the communications between real agents.

In spite of significant progress in distributed CSP, the following question is straightforward: *Why is only one variable per agent assumed?* (Salido, 2007). Only some works include a set of variables into a single agent (Silaghi and Faltings, 2005),(Ezzahir and Bouyakhf, 2007),(Salido and Barber, 2006). Nevertheless, few works have been focused on distributed techniques for solving large-scale problems (Yokoo *et al.*, 1998*a*).

We argue that the problem of multiple local variables per agent can be managed efficiently. In this paper, we present different alternatives for managing large-scale problems. The alternatives are based on a general distributed model

and a set of guidelines for distributing large-scale problems. The general model is inspired in holonic structures (Koestler, 1971) and the Abstract Agent notion (Giret and Botti, 2004*b*). Each agent will be committed to a large number of variables and constraints and several subproblems can be executed concurrently depending on the internal structure. These methods are both efficient and scalable.

4.1. *A General Distributed Model*

Depending on the problem to be considered, the distributed model will maintain different properties. Our general model for solving scheduling problems can be considered as a synchronous model. It is meant to be a framework for interacting and cooperating holons/agents to achieve a consistent state. The main idea of our holonic/multi-agent system model is based on (Salido *et al.*, 2003) in which the problem is partitioned into a set of subproblems; then the subproblems are classified in the appropriate order and are solved concurrently.

In this section, we introduce the notion of holon as a complementary idea of agent. Depending on the scheduling problem, we will use holons instead of agents. That is, we will use different organizational structures: hierarchy, heterarchy or holarchy.

A hierarchical structure is a horizontal structure in which there is a central head or controller (Figure 3). The only type of communication allowed is between the controller and its controlled entities. This yields a deterministic system in which control is achieved by means of the use of global information. Nevertheless, disturbances and changes cannot be coped with.

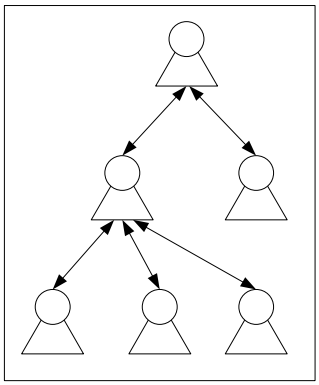

Fig. 3. A hierarchical structure

A heterarchical structure is a flat structure that is composed of independent entities (agents) where there is no central authority (Figure 4). The allocation of tasks to these agents is done using dynamic market mechanisms. This yields a simple and fault-tolerant system since none of the agents need a-priori information about the other agents. As a consequence, several disturbances and changes can easily be coped with. Nevertheless, the basic assumption of this architecture paradigm gives rise to its principle drawbacks: the independence of agents prohibits the use of global information. Therefore, the global system performance is very sensitive to the definition of the market rules; the system cannot guarantee a minimum performance level in case of unforseen circumstances.

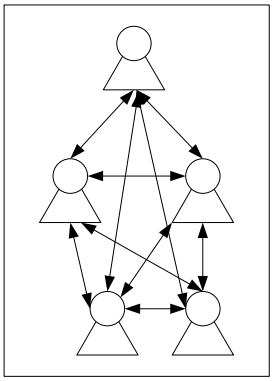

Fig. 4. A heterarchical structure

A holarchy structure is a combination of the two previous ones. In it, there are temporal and relaxed hierarchies that are dynamically created and deleted when required (Figure 5). Moreover, the participating entities called holons are autonomous and cooperating entities that can be seen as a whole and a part. Therefore, a holarchy is a group of basic holons and/or recursive holons that are themselves holarchies. The holonic concept was developed by the philosopher Arthur Koestler (Koestler, 1971) in order to explain the evolution of biological and social systems. The strength of a holonic organization, or a *holarchy*, is that it enables the construction of very complex systems that are efficient in the use of resources, highly resilient to disturbance (both internal and external), and adaptable to the changes in the environment in which they exist. Within a holarchy, holons may dynamically create and change hierarchies. Moreover, holons can participate in multiple hierarchies at the same time. Holarchies are recursive in the sense that a holon may itself be an entire holarchy that acts as an autonomous and cooperative unit in the first holarchy.

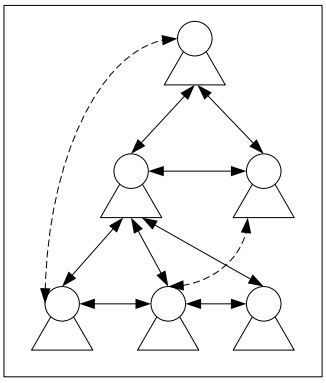

Fig. 5. A holarchy structure

The stability of holons and holarchies stems from holons being self-reliant units, which have a degree of independence and can handle circumstances and problems on their

particular level of existence without asking higher level holons for assistance. Holons can also receive instruction from and, to a certain extent, be controlled by higher level holons. This self-reliant characteristic ensures that holons are stable and are able to survive disturbance. The subordination to higher level holons ensures the effective operation of the larger whole.

In summary, a holonic architecture is committed to organizing entities (holon or agent (Giret and Botti, 2004*a*)) that are responsible for solving each subproblem. In order to unify the concepts of holon and agent, we will use the Abstract Agent notion (for short, A-Agent) (Giret and Botti, 2004*b*) to refer to any of these entities indistinctly.

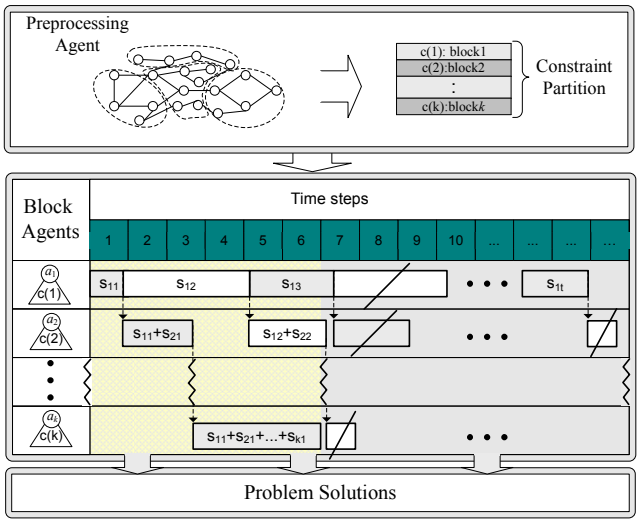

Fig. 6. General Distributed Model.

Depending on its features, the problem is partitioned into k blocks or clusters in order to be studied by A-Agents called *block agents*. Furthermore, a partition agent is committed to classifying the subproblems in the appropriate order, depending on the selected proposal.

Once the problem is divided into k blocks by a *preprocessing agent*, a group of *block agents* concurrently manages each block of constraints. Each *block A-Agent* is in charge of solving its own subproblem by means of a search algorithm. Each *block A-Agent* is free to select any algorithm to find a consistent partial state. It can select a local search algorithm, a backtracking-based algorithm, or any other, depending on the problem topology. In any case, each *block A-Agent* is committed to finding a solution to its particular subproblem. This subproblem is composed of its CSP, but it is subject to the variable assignment generated by the previous *block A-Agents*. Thus, *block A-Agent* 1 works on its group of constraints. If *block A-Agent* 1 finds a solution to its subproblem, it then sends the consistent partial state to *block A-Agent* 2, and they both work concurrently to solve their specific subproblems; *block A-Agent* 1 tries to find another solution and *block A-Agent* 2 tries to solve its subproblem knowing that its *common variables* have been assigned by *block A-Agent* 1. This second solution found by *block A-Agent* 1 is stored to be sent to *block A-Agent* 2 if necessary. In this case, *block A-Agent* 2 will not wait for *block A-Agent* 1 to search for a new solution.

Thus, *block agent* j, with the variable assignments generated by the previous *block A-Agents*, works concurrently with the previous *block A-Agents* and tries to find a more complete consistent state using a search algorithm. Finally, the last *block A-Agent* k, working concurrently with *block A-Agents* $1, 2, ...(k-1)$, tries to find a consistent state in order to find a problem solution.

Figure 6 shows the holonic system, in which the *preprocessing agent* carries out the network partition and, the *block A-Agents* (a_i) are committed to concurrently finding partial problem solutions (s_{ij}, where i denotes the number of *block A-Agents* and j the jth solution). Each *block A-Agent* sends the partial problem solutions to the next *block A-Agent* until a problem solution is found (by the last *block A-Agent*). For example, state: $s_{11} + s_{21} + ... + s_{k1}$ is a problem solution. The concurrence can be seen in Figure 6 in *Time step* 6 in which all *block A-Agents* are concurrently working. Each *block A-Agent* maintains the corresponding domains for its *new variables*. The *new variables* are the variables that are not involved in the previous *block A-Agent*. The *block A-Agent* must assign values to its *new variables* so that the block of constraints is satisfied. When a *block A-Agent* finds a value for each *new variable*, it then sends the consistent partial state to the next *block A-Agent*. When the last *block A-Agent* assigns values to its *new variables* satisfying its block of constraints, then a solution is found. It must be taken into account that if a *block A-Agent* maintains too many variables or constraints, it can be decomposed into a set of new agents/holons.

4.2. *Some Guidelines for Distributing Large-Scale Problems*

To manage large-scale problems, we must answer some modeling questions:

- Is the problem distributed by nature?
- Does the problem have a clustering appearance?
- Is the problem structure hierarchical?
- It a central authority necessary?

To answer these questions, we give some guidelines that can be taken into account to distribute and solve the problem.

(i) *The number of subproblems (agents).* This number must be in concordance with the size of the problem. As we have pointed out in the previous sections, one agent per variable is unmanageable. A problem with thousands of variables and constraints cannot be modelled as a distributed model with thousands of agents due to the high computational cost in the solving process. Generally, when the problem has privacy issues, the number of subproblems is straightforward, due to the nature of the problem. Nevertheless, some centralized problems are too large and should be divided/decomposed into smaller ones in

order to be solved in a more efficient manner. A reasonable way to divide/decompose the problem is by means of graph partitioning techniques (Salido and Barber, 2006). Thus, if the problem has a clustering appearance, the decomposition process will be guided by these clusters. However, in many real problems, the best way to partition the problem is by carrying out a domain dependent partition. Following, we present an example of a railway scheduling problem distributed by types of trains and sets of stations.

(ii) *The order in which each subproblem is executed (agent priority).* Sometimes all subproblems can be executed concurrently and partial states are sent to their neighbours. In many other cases, the subproblems can be ordered using a selected criterion.
- From the subproblems with the most neighbours to the subproblems with the least neighbours.
- From the tightest subproblems to the loosest ones.
- From the subproblems that have hard constraints to the subproblems with soft constraints.
- etc...

This ordering does not mean that the subproblems are solved sequentially, but rather that some subproblems are executed first and then all neighbours are concurrently executed (see Figure 6). Furthermore, this ordering, which is represented by priorities among agents, can change dynamically depending on many factors such as number of nogoods, number of backtracks, etc.

(iii) *The management of backtracking.* When an agent does not find a partial solution, it must communicate its current state to the related agents in order to avoid unnecessary searches. Some works decompose the subproblem into a set of subproblems in such a way that the resultant problem is represented as a tree. Each node in the tree is a subproblem (Abril *et al.*, 2007). Once a node/agent finds a partial solution, it is sent to its children, and the subproblems are solved concurrently. However, if a node/agent does not find a partial solution, a message is sent to its parents, and, depending on the management of the backtracking (which parent it backtracks to), the search space will be pruned more efficiently.

(iv) *The need for a central authority.* As we have pointed out, depending on the problem features, a central regulatory authority may or may not be required. Many researchers suggest, that for some multi-agent systems, no central regulatory authority is needed and can be replaced by a virtual representation where each agent is responsible for maintaining its own partial view of the relevant institutional state. This is due to the fact that, if the central authority fails, the problem cannot be solved. However, many real problems are hierarchical by nature and it is necessary to generate different levels of abstraction. In these cases, a holonic structure is very useful because temporal hierarchies are maintained whenever they are needed. Despite the hierarchical structures, the horizontal communications (pair to pair communication) in a holarchy are not forbidden. Thus, when the head or controller of the holarchy fails, a new one can be chosen by means of a voting mechanism that selects one holon of the holarchy as the new controller. In the next section, we will present a global road transportation system: a hierarchical system that maintains as many levels as necessary, depending on the problem complexity.

5. Distributed Models for Distributed Scheduling Problems: Two case studies

Many real problems are distributed by their nature. However, this distribution does not imply a single variable per agent. For instance, by their nature, many scheduling problems are decentralized and the problem is decomposed into clusters. Each cluster (composed by variables and constraints) is solved by an agent, and it communicates a consistent partial state. In this way, a large-scale scheduling problem can be solved with reasonable efficiency, maintaining all privacy issues. Most of these problems are very complex and traditional distributed techniques (such as the techniques in Section 4) are not appropriate.

In this section, we present two real-life scheduling problems, which by their nature, are distributed and very complex. To solve them in a distributed way, we group several compatible variables per agent so that the problem can be solved in a reasonable time.

5.1. *The Railway Scheduling Problem*

Train timetabling is a difficult problem, particularly in the case of real networks, where the number and the complexity of constraints grows drastically. A feasible train timetable should specify the departure and arrival time of each train to each location of its journey in such a way that the line capacity and other operational constraints are taken into account. Traditionally, train timetables are generated manually by drawing trains on the time-distance graph. The train schedule is generated from a given starting time and is manually adjusted so that all constraints are met. High priority trains are usually placed first, followed by lower priority trains. It can take many days to develop train timetables for a line, and the process usually stops once a feasible timetable has been found. The resulting plan of this procedure may be far from optimal.

The literature of the 1960s, 1970s, and 1980s related to rail optimization was relatively limited. Compared to the airline and bus industries, optimization was generally overlooked in favor of simulation or heuristic-based methods. However, Cordeau et al. (Cordeau *et al.*, 1998) pointed out greater competition, privatization, deregulation, and increasing computer speed as reasons for the more prevalent use of optimization techniques in the railway indus-

try. Our review of the methods and models that have been published indicates that the majority of authors use models that are based on the Periodic Event Scheduling Problem (PESP) introduced by Serafini and Ukovich (Serafini and Ukovich, 1989). The PESP considers the problem of scheduling as a set of periodically recurring events under periodic time-window constraints. The model generates disjunctive constraints that may cause the exponential growth of the computational complexity of the problem depending on its size. Schrijver and Steenbeek (Schrijver and Steenbeek, 1994) have developed CADANS, a constraint programming-based algorithm to find a feasible timetable for a set of PESP constraints. The train scheduling problem can also be modeled as a special case of the job-shop scheduling problem (Silva de Oliveira (Silva de Oliveira, 2001), Walker et al. (Walker and Ryan, 2005)), where train trips are considered as jobs that are scheduled on tracks that are regarded as resources.

We modelled the railway scheduling problem as a Constraint Satisfaction Problem (CSP), and it was solved by using constraint programming techniques. However, due to the distributed nature of the problem and the huge number of variables and constraints that this problem generates, a distributed model was developed to distribute the resultant CSP into semi-independent subproblems so that the solution could be found efficiently (Salido *et al.*, 2007).

Here, we present two ways for distributing the railway scheduling problem. It is partitioned into a set of subproblems by means of types of trains and by means of contiguous constraints.

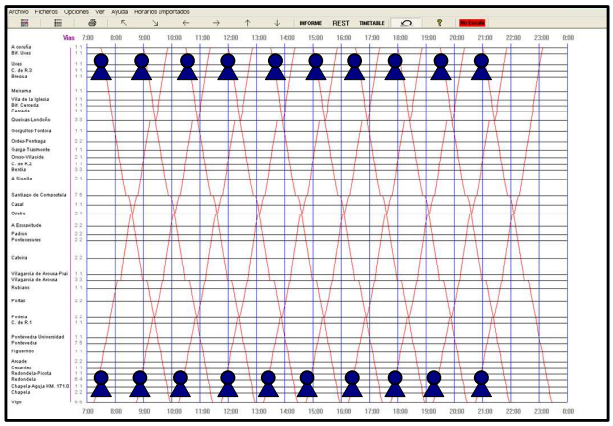

Fig. 7. Distributed Railway Scheduling Problem by type of trains.

5.1.1. *Distribution by types of trains*

This distributed model is based on dividing the original railway problem by means of train types. In this model, each agent is committed to assigning values to variables that are related to a train or sets of trains in order to minimize the journey time. This partition model takes into account some of the guidelines given above.

– Depending on the problem instance, the number of partitions will be given by the railway operator or by the number of trains. Figure 7 shows a running map with 20 partitions. Each agent manages one train. Each train generates a large number of variables and constraints, depending on the number of stations and the user requirements. Furthermore, this model allows us to improve privacy. Currently, due to the policy of deregulation in the European railways, trains from different operators work on the same railway infrastructure. Thus, the partition model also gives us the possibility of partitioning the problem so that each agent is committed to an operator. Therefore, different operators maintain privacy about strategic data.
– This model allows us to efficiently manage priorities between different types of trains (regional trains, high-speed trains, freight trains). In this way, agents committed to priority trains (high-speed trains) will first carry out the value assignment to variables in order to achieve better journey times. This ordering is inserted into our distributed model to solve the scheduling problem concurrently.

5.1.2. *Distribution by set of stations*

This distributed model is based on distributing the original railway problem by means of contiguous stations. The deregulation of European railway operators gives the opportunity to schedule long journeys. However, long journeys involve a large number of stations in different countries with different railway policies. Therefore, a logical partition of the railway network can be carried out by means of regions (contiguous stations).

Some of the guidelines presented in Section 4.2 that must be taken into account in this model are:

– The number of subproblems depends directly on several factors: distance of the journey, number of different regions (mainly countries) and railway topology. This distributed model divides the problem into a set of physical regions. It is important to analyze the railway infrastructure in order to detect restricted regions (bottlenecks). To balance the problem, each agent is committed to a different number of stations. An agent can manage many stations if they are not restricted stations; however, an agent can manage only a few stations if they represent bottlenecks.
– The order in which each subproblem is executed plays an important role. Agents committed with bottlenecks have preference to assign values to variables due to the fact that their domains are reduced (variable ordering). Once the agent with the tightest constraints solves its subproblems, both the previous agent and the following agent must concurrently solve their own subproblems.
– The management of backtracking is very important to avoid unnecessary constraint checking. For instance, if agent 4 in Figure 8 finds a partial solution to its subproblem, it communicates to agent 3 and agent 5. Both agents concurrently search to find their own partial solutions. However, if agent 3 does not find a partial solution,

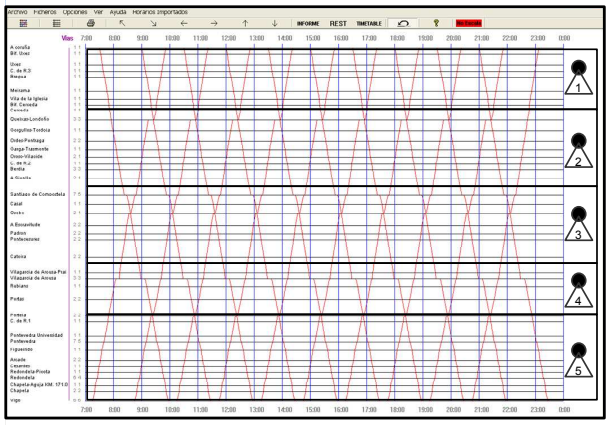

Fig. 8. Distributed Railway Scheduling Problem by set of stations.

it sends a nogood message to agent 4, and this last agent sends a message to agent 5 in order to stop its search. As we pointed out in Figure 6, while agent 3 and agent 5 work concurrently to find their partial solutions, agent 4 also works in the same time-step in order to find another partial solution. This last partial solution will be used if agent 3 or agent 5 backtracks due to inconsistency with the previous partial solution given by agent 4.

Figure 8 shows the journeys to be scheduled between two cities. They are decomposed into several shorter journeys. The set of stations is partitioned into blocks of contiguous stations and a set of agents will coordinate with each other to achieve a global solution. Thus, we can obtain important results such as railway capacity (Abril *et al.*, 2008), consistent timetables, etc.

5.2. *Global Road Transportation System*

During recent years, the development of automated traffic systems has received increased attention, and substantial effort has been invested in trying to find a solution to problems associated with road transport. Among these problems are road accidents caused by human-related factors, such as tiredness, loss of control, a slow reaction time, limited field of view, etc. A further transport-related problem is that of loss of time, which may be caused by slow driving speed due to weather conditions, road conditions, visibility, and traffic congestion. In this section, we present a global road transportation system, which is being developed by several European Universities. The main goal of the algorithmic section is to develop algorithms that are capable of creating driving schemes for a vehicle from any arbitrary address to any other arbitrary address (in the address space of the system), while considering, and if necessary, adapting the driving schemes of other vehicles travelling in the system at the same time according to priorities, driving and optimization rules. Thus, distributed techniques are necessary to solve these problems.

The Global Automated Transportation System (GATS) (Zelinkovskyn, n.d.) is a driverless, integrated transport system. It has the ability to simultaneously coordinate the macro and micro needs of road transport networks. Millions of vehicles can be optimally, simultaneously and automatically "driven" over a virtually unlimited geographic region, including whole continents, while the requirements of each individual vehicle and its passengers are attended to at the same time. It is an innovative concept, based on simple, recognized principles and proven technologies.

Due to the decentralized and modular nature of the architecture, it can be implemented with the same ease and simplicity in both small contained areas such as airports and theme parks and in larger areas such as local, national and international road systems.

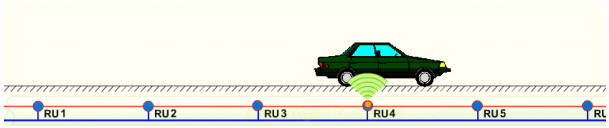

Fig. 9. Driving on the System.

Following, we summarize the architecture that is above and below the road. In the center of a traffic lane, 15-20 cm below the road surface, there is an "Intelligent Cable" of about 1 cm in diameter which is comprised of tiny intelligent transponders (Road-Units (RUs)) located at fixed distances (less than a vehicle length of 3m) from each other. While driving, the vehicle sends short radio transmissions down towards the RUs at regular time intervals (about every 30 milliseconds). The RU receives a transmission, processes it, and responds with a radio transmission back to the vehicle. The vehicle communicates continuously with the RUs one after the other incessantly. Thus, it has continuous radio communication with the RUs and the whole system connected to them. The interchange of radio transmissions between the vehicles and the RUs also facilitates lateral and longitudinal positioning of vehicles on the road, as presented in Figure 9.

The memory of each RU stores the specifications of the RU and the individual driving instructions that it will transmit to each vehicle above it. Several hundreds of consecutive RUs constitute a Segment, whose functions are administered by a *Segment Controller*. The *Segment Controller* is connected to its RUs through the Parallel Buses and is responsible for "driving" the vehicles passing in its domain; performing routine maintenance check-ups of the components in its segment; and monitoring and regulating their mutual performance. A group of adjacent *Segment Controllers* has a superior controller, the *Level-1 Controller*, which coordinates and controls its individual and mutual functions. A group of adjacent *Level-1 Controllers* has a superior controller: *Level-2 Controller*. This goes on hierarchically (Figure 10). The *Top Level Controller* coordinates and controls the functions of the whole system. There is virtually no limit to the number of levels and to the size of the geographic domain of the systems.

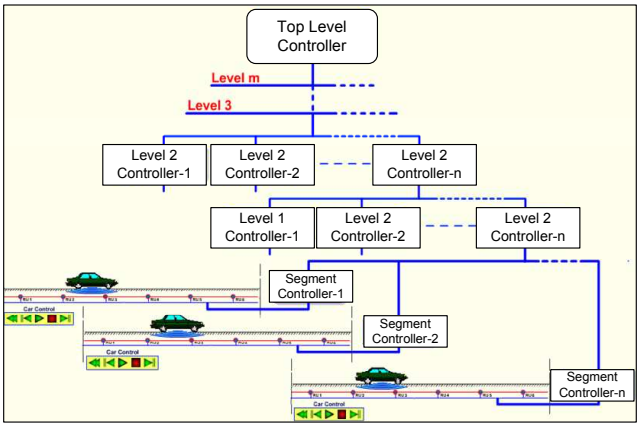

Fig. 10. Hierarchical Architecture of the System.

5.2.1. *Integrated Functioning*

Assume a vehicle is in a parking lot above a RU. The passengers turn the vehicle on, which begins to send short radio transmissions down towards the road. The RU detects those transmissions and responds with radio transmissions back to the vehicle. The RU initiates a communication session with the *Segment Controller* in order to inform it about the new event. The passengers in the vehicle enter their requirements such as destination, priority, preferred routes etc. The vehicle's processor sends a message that includes the requirements, the exact location of the vehicle relative to the RU and its own specifications to the *Segment Controller*. The *Segment Controller* processes the request while considering additional inputs from other RUs in the Segment and from its superior Controller. Finally, it prepares a driving instruction message for each RU in the Segment. The RUs will send these instructions to the vehicle when it passes above them. Each message includes an addressee (RU1 etc.), a vehicle ID, the expected arrival time of the vehicle to the RU, the speed that the vehicle should travel at and the driving direction. When the vehicle is driving from one RU to another, the active RU uses the Serial Bus to inform the next and previous RUs in the sequence about the exact timing, the ID number and other specifications of the moving vehicle. If the RUs detect intolerable deviation from the plan, they can initiate a so-called Emergency Braking Procedure. The active RU uses the Parallel Bus to inform the *Segment Controller* with information about the moving vehicle.

5.2.2. *A Distributed Model for GATS*

Traditional Centralized techniques fail to model and implement problems of this type due to their complex and large nature. Because of the decentralized and modular nature of the architecture, the algorithms must be distributed in order to obtain the scheduling of each vehicle. Figure 11 shows the map of Europe to be distributed/divided into regions (countries); each region is divided into sub-regions, and so on.

Briefly, the system is composed of a network, where nodes are locations and arcs are roads. Depending on the granularity, nodes are points in the road or regions in a country. In the lower level of the system, each RU is represented by a variable (see Figure 12). The system may be composed of millions of RUs. As we have explained in the first section, this problem cannot be managed by current distributed CP techniques by using a single variable per agent. By using these approaches, the problem generates millions of agents and messages passing in the interaction scenarios. This makes the resulting DisCSP unmanageable.

Fig. 11. Map of Europe to be distributed.

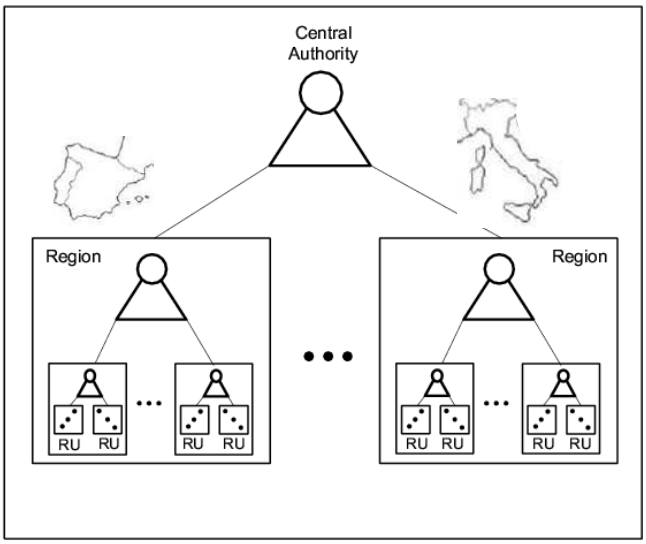

Fig. 12. Distributed model with a central authority.

To overcome these weaknesses, we use our distributed model in which the problem is partitioned into subproblems that represent regions, countries, etc (see Figure 12). Here, we use a Holonic architecture (HMS, 1994)(Koestler, 1971) to organize the entities (holon or agent (Giret and Botti, 2004*a*)) that are responsible for solving each subproblem. Due to the hierarchical structure of the problem and its changing conditions, the holonic architecture proposed in Section 4.1 is used. That is, whenever two regions need to

cooperate, a new holarchy is created for these two regions and it is managed by the corresponding high level *Segment Controller* (see Figure 13).

In order to get a better understanding of the holonic structure that we have used in the GAT problem, we explain a simplified execution case. Figure 13 shows a group of RUs and *Segment Controllers* for a given set of regions. At start time, there is a group of unconnected holons (Figure 13.A). When a journey plan is required from a given point X to a point Y, the corresponding holarchy is created among the RUs and the *Segment Controller* (see Figure 13.B). Let's suppose that a new journey plan is required for two other points. In order to calculate this plan, the corresponding new holarchy is created among the RUs and the *Segment Controller* responsible for the region (Figure 13.C). Later, a journey plan connecting two regions needs calculation. In this case, the previously created holarchies are connected with the new holarchy responsible of the path calculation (Figure 13.D). Finally, when there is no vehicle running in the segments of the previously created holarchies, these holarchies are deleted and the entire system returns to its initial state (Figure 13.A).

The distributed model generated for this scheduling problem follows the guidelines presented in the paper.

- The number of subproblems depends on the size of the system. A holon can represent a track between two traffic lights or represent a region or a country. Figure 12 shows two holons that represent two countries, Spain and Italy. Each of them is composed of a set of sub-holons that represent regions, and each sub-holon is composed by new sub-holons that represent sub-regions and so on. The base case is composed of individual variables that represent RUs.

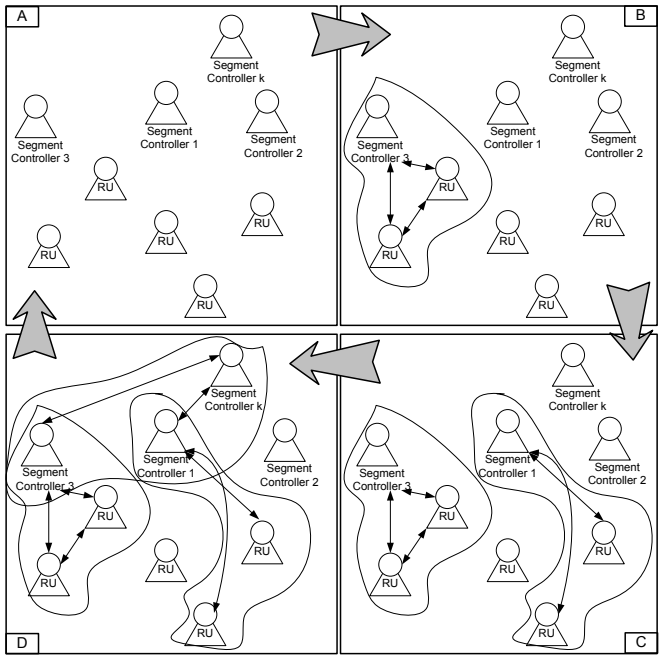

Fig. 13. The holonic structure for the GAT problem

- The execution of the subproblems is carried out in two steps. First, given the requirements of the passenger (with the destination being the most important requirement), the central authority is the *Level i Controller*, which involves both origin and destination. A holarchy is created with the regions and the *Level i Controller*. This *Level i Controller* is committed to solving the shortest path in a high level problem (each node is a region). This path is only a first approach to guides us to finding the shortest path. Thus, *Level i Controller* is executed first, then all *Level i-1 Controllers* are executed concurrently and so on. Depending on the size of the journey, several hierarchical levels are required, and the corresponding holarchies to solve the problem at hand are created. Finally, the calculated route is sent to the *Segment Controllers* that are involved.
- Due to the dynamic structure of the problem, some parts of the system may change and new schedules must be calculated. The rescheduling is only calculated from the incidence to the destination. The management of backtracking is carried out in a way similar to the railway scheduling problem distributed by stations.
- As we have pointed out, the nature of the system makes the presence of a central authority necessary. However, due to the scalability of the system, the central authority has the same behaviour as a level controller. The central authority is the minimal level controller that involves both origin and destination. This level depends on the problem instance. Thus, the holarchies are created as needed. Whenever there is no vehicle running between two given regions the corresponding holarchy is destroyed. The holons do not form the holarchy until a new journey planning or vehicle control among the regions is required.

6. Conclusions

In this paper, we question the common assumption made in DisCSP literature in which each agent has just a single variable. Many real problems can be modelled as a CSP, but they cannot be solved by using DisCSP techniques due to the enormous amount of message passing. Thus, new distributed techniques must be developed to solved large instances of real problems. In this paper, we present a general distributed model for solving large-scale problems. This distributed model is based on the idea of Holonic Systems. We also include some guidelines for distributing these problems by relaxing the above assumption. Finally, two real-life problems that can be modelled as a distributed problem are presented. They manage several variables per agent in order to solve these problems in a reasonable time. These problems follow some of the presented guidelines presented in the paper.

7. Acknowledgements

This work has been partially supported by the research projects TIN2007-29666-E and TIN2007-67943-C02-01 (Min. de Educacion y Ciencia, Spain-FEDER), CONSOLIDER-INGENIO 2010 under grant CSD2007-00022, FOM- 70022/T05 (Min. de Fomento, Spain), GV/2007/274 (Generalidad Valenciana), PAID-06-07/3191 (Universidad Politecnica de Valencia) and by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

References

Abril, M., F. Barber, L. Ingolotti, M.A. Salido, P. Tormos and A. Lova (2008). An assessment of railway capacity. *Transportation Research Part E-Logistics and Transportation Review,* p. to appear.

Abril, M., M.A. Salido and Barber (2007). DFS-tree based heuristic search. *In Proceeding of the Seventh Symposium on Abstraction, Reformulation and Abstraction (SARA'07), LNAI 4612,* pp. 5–19.

Bacchus, F. and P. van Beek (1998). On the conversion between non-binary and binary constraint satisfaction problems. *In proceeding of AAAI-98* pp. 311–318.

Cordeau, J., P. Toth and D. Vigo (1998). A survey of optimization models for train routing and scheduling. *Transportation Science* **32**, 380–446.

Ezzahir, R., Bessire C. Belaissaoui M. and El-H. Bouyakhf (2007). Dischoco: A platform for distributed constraint programming. *In Proceedings of IJCAI-07 Workshop on Distributed Constraint Reasoning* pp. 16–21.

Faltings, B. and M. Yokoo (2005). Introduction: Special issue on distributed constraint satisfaction. *Artificial Intelligence* **161**, 1–5.

Giret, A. and V. Botti (2004*a*). Holons and Agents. *Journal of Intelligent Manufacturing* **15**, 645–659.

Giret, A. and V. Botti (2004*b*). Towards an Abstract Recursive Agent. *Integrated Computer-Aided Engineering* **11**(2), 165–177.

HMS, Press Release (1994). *HMS Requirements*. HMS Server. http://hms.ifw.uni-hannover.de/.

Koestler, A. (1971). *The Ghost in the Machine*. Arkana Books.

Mackworth, A.K. (1992). Constraint satisfaction. *Encyclopedia of Artificial Intelligence* pp. 285–293.

Salido, M.A. (2007). Distributed CSPs: Why it is assumed a variable per agent?. *In Proceeding of the Seventh Symposium on Abstraction, Reformulation and Abstraction (SARA'07), LNAI 4612,* pp. 407–408.

Salido, M.A., A. Giret and F. Barber (2003). Distributing Constraints by Sampling in Non-Binary CSPs. *In IJCAI Workshop on Distributing Constraint Reasoning* pp. 79–87.

Salido, M.A. and F. Barber (2006). Distributed CSPs by graph partitioning. *Applied Mathematics and Computation.* **183**, 491–498.

Salido, M.A., M. Abril, F. Barber, L. Ingolotti, P. Tormos and A. Lova (2007). Domain-dependent distributed models for railway scheduling. *Knowledge Based Systems. Ed. Elsevier Science* **20**, 186–194.

Schrijver, A. and A. Steenbeek (1994). Timetable construction for railned. *Technical Report, CWI, Amsterdam, The Netherlands.*

Serafini, P. and W Ukovich (1989). A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics* pp. 550–581.

Silaghi, M.C. and B. Faltings (2005). Asynchronous aggregation and consistency in distributed constraint satisfaction. *Artificial Intelligence* **161**, 25–53.

Silva de Oliveira, E. (2001). Solving single-track railway scheduling problem using constraint programming. *Phd Thesis. Univ. of Leeds, School of Computing.*

Walker, C., Snowdon J. and D. Ryan (2005). Simultaneous disruption recovery of a train timetable and crew roster in real time. *Comput. Oper. Res* pp. 2077–2094.

Yokoo, M. and K. Hirayama (2000). Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems* **3**, 185–207.

Yokoo, M., E.H. Durfee, T. Ishida and K. Kuwabara (1992). Distributed constraint satisfaction for formalizing distributed problem solving. *In 12th International Conference on Distributed Computing Systems (ICDCS-92)* pp. 614–621.

Yokoo, M., E.H. Durfee, T. Ishida and K. Kuwabara (1998*a*). Distributed constraint satisfaction algorithm for complex local problems. *Third International Conference on Multiagent Systems (ICMAS-98)* pp. 372–379.

Yokoo, M., E.H. Durfee, T. Ishida and K. Kuwabara (1998*b*). The distributed constraint satisfaction problem: formalization and algorithms. *In IEEE Transactions on Knowledge and Data Engineering* **10**, 673–685.

Zelinkovskyn, R. (n.d.). Global Automated Transport System. *http://www.global-transportation.com.*