Constraint Reasoning and Interactive Planning

*An Application to Forest Fire Fighting

Anna Perini and Francesco Ricci

Istituto per la Ricerca Scientifica e Tecnologica, 38050 Povo (TN), Italy
e.mail: perini@irst.it, tel ++39+461-314330

Abstract. Constraint Satisfaction Problem (CSP) for variables with real domains has been proposed as a framework for dealing with temporal reasoning on metric information [9]. A temporal reasoning system based on this framework has been developed inside a system aimed at supporting the planning of an initial attack to forest fires. The system implements a constraint reasoning language through an extension of a Object Oriented language. More generally the interactive planning system integrates case based reasoning techniques with constraint reasoning. Temporal constraints are used in two steps of the planning process: plan fitting and adaptation, and resource scheduling. This planning system is part of a decision support system aimed at supporting the user in the whole process of the forest fire management including both situation assessment and planning activities. A first prototype of the planning system is being tested.

1 Introduction

This paper presents the application of constraint technologies to the development of an interactive system that shall be used for planning the initial attack to forest fires [3, 22, 23]. This system is based on a hybrid architecture for planning that integrates a Case-Based Planner [12] and a Constraint Reasoner [13].

The case based module retrieves partial plans from memory using a similarity metric that takes into account not only a set of predictive features, as is usually done, but also the plan structure itself. Plans are represented in a hierarchy of parts. Partial plans can be completed using the similarity metric by searching for expansions of the leaves in the plan tree structure.

The constraint module adapts the retrieved plan, attaching and propagating constraints. Constraint reasoning is used mainly for managing temporal relations defined on sub-plans or between actions in plans. Constraints are also used in the identification of the sets of applicable actions in relation to particular scenario data. The temporal dimension of the plan is managed with a quantitative approach based on temporal variables over continuous domains. The constraint module also supports the resource allocation process for the chosen plan.

* This work has been partially supported by the EspritIII project #6095 CHARADE (Combining Human Assessment and Reasoning Aids for Decision Making in environmental Emergencies).

The planner is integrated in an Intelligent Decision Support System that includes Situation Assessment and Resource Manager subsystems. The Situation Assessment subsystem is based on a Geographical Information System and on a fire spreading model. During the assessment of a fire alarm the fire spreading model computes physical parameters of the fire given the geographical co-ordinates of the fire and some meteorological data and it forecasts the fire evolution in the following hours. The user can also draw on the map sector lines that represent the zones in which some actions (water spraying, control line, back fire, etc.) have to be executed. The information produced by the situation assessment is used both in plan retrieval and temporal reasoning. For example, the duration of an action is computed taking into account the length of the sector line where it has to be executed, the type of work and the number of men in the squad involved. The Resource Manager subsystem can show the displacement of all the resources (human and means) both in the bases and in the operating zones. The plans produced by the system can be scheduled and resources are allocated to them in accordance with plans' demands, availability of resources in the bases and optimising some objective functions (time, resource consumption, etc.).

The following concentrates on the use of constraint reasoning within this architecture. Both modelling issues are addressed, mainly how temporal information is coded into constraints, and the constraint reasoning tasks that are supported by the system. This application is being developed in an Object Oriented framework so all the abstractions here presented (plans, constraints, actions, etc.) are implemented as objects. In the following the word "object" is often used to refer to both the concept and the implementation. This paper does not present a methodology for translating application problems into constraint satisfaction problems (see for example [19] for a contribution in this direction). Instead, a case is presented in which complex heterogeneous knowledge was modeled, and constraints, among other forms of knowledge, play a central role. Nevertheless a well known and applied object-oriented analysis and design methodology was relied upon [5]. Using this methodology both the constraint reasoning subsystem and the whole application were designed in a uniform way. The Booch methodology and the object-oriented target language features helped in many ways:

- Constraints are one tool for knowledge representation among many. A unifying language and design methodology enables simple integration of different tools (methods, frames, data-base, etc.).
- A common design methodology provides an intermediary language for exchanging domain knowledge and in particular constraint definition and applicability.
- It is relatively simple to change constraint definition and propagation techniques used for specific constraint types. The system can be developed in an evolutionary way as is common in an object-oriented framework.
- Polymorphism enables the definition of constraint hierarchies in which more specific constraints adopt more specific constraint propagation techniques. For example, the `revise` procedure, namely the projection of one variable,

can be defined as a polymorphic method on the constraint type. So standard projection can be used by default but more efficient implementation can be developed when possible.

In our experience the main difficulty in applying constraints resides in the balance between the expressiveness required by the application and the efficiency and tractability of the constraint reasoning task required. So often it was necessary to redesign some parts because of the constraint reasoner inefficiency in tackling those specific computational tasks. For example the hierarchical representation of plans (an important feature of plan representation) introduced great complexity for the temporal reasoner so that we were obliged to revise this feature. But, at the same time, one should always avoid the risk of being overdetermined by the constraint solver capabilities, for example by introducing functions to the application that are simply supported by the constraint reasoner but that are of minor interest for the user. In our experience, a thorough investigation of the user tasks, even if difficult and extremely time consuming, always suggests interesting aspects also from a theoretical point of view. The following illustrates how some part of the domain knowledge was represented with constraints and how user functions were modeled with constraint reasoning tasks. Section 2.1 describes the plan representation. In Section 2.2 the two levels of temporal data representation (user and system) are presented. Section 2.3 illustrates the user interaction with planner and the impact on the constraint reasoner. Section 3 briefly describes the main abstractions in the constraint reasoner module and the main features of the solver that was implemented for supporting the temporal reasoner. Finally Section 4 briefly reiterates the main results of the work and lists some open problems.

2 Plan Representation and Interactive Planning

The main function of the system is to build intervention plans when new forest fires are detected and an assessment of the situation has been performed by the operator. An intervention plan specifies the set of fire fighting techniques, such as spraying water, building control lines or using back fire, that are to be performed within a given time in order to suppress the fire. The attack can be performed on different fire fronts (sectors) in parallel or according to a given temporal sequence. Here the plan representation tailored to the hybrid planning approach chosen is described.

2.1 Plan Representation

A plan is an object that is essentially characterized by a descriptive, domain dependent, component and by a structure representing a 'part of' hierarchy and the temporal relations between plan components. The descriptive component contains a set of scenario data resulting from the assessment of the fire alarm (physical parameters of the fire, wind speed and direction, accessibility of the

burning area, sectors). These data justify the choice of the fire fighting actions used in the plan and allow their dimensioning in terms of needed resources and temporal duration.

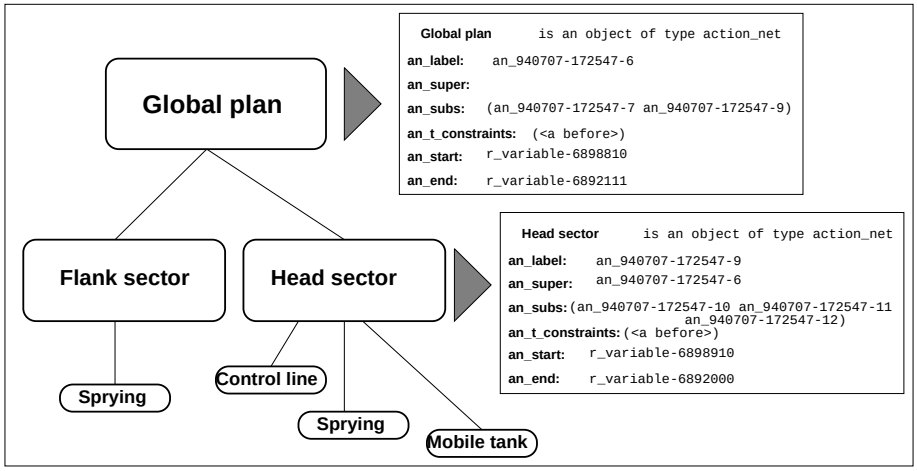

Fig. 1. Part of hierarchy in plan representation. Two examples of `action_net` object are shown, only the main attributes are reported.

The hierarchical structure of the plan allows different levels of abstraction to be represented in a uniform way: the global plan, the plans on different sectors, the specific actions. So actions, which are the leaves of the hierarchy, are represented by the same type of object (`action_net`) that represents sector or global plans, differing in the type of scenario data associated with them, that is the descriptive component is specialized according to the hierarchy level. For instance the scenario data of an action contains information on the fire fighting technique such as the type and number of resources to be used, the scenario data of a sector plan contain information on the fire situation, on the accessibility, on the buildings to defend, etc., relative to the sector line. The scenario data and the plan structure enter in the computation of the similarity metric that drives the search of the plan in memory. Figure 1 depicts an example of hierarchical structure and how it is represented in objects of type `action_net`.

2.2 Temporal Data

Focusing now on the temporal dimension of a plan shows two levels at which temporal information is represented:

- **The system's user level.** Here time intervals are considered and constraints between time intervals are handled.

– **The constraint reasoner level.** Here time points are defined and time constraints representing bounded differences between time points are implemented.

The distinction between the user level and the constraint reasoner level points out an issue raised in the introduction. The temporal information to be dealt in the application is essentially of metric type. Constraints on time points allow to represent and reason on quantitative information, but cannot be easily grasped by the user. Intervals and constraints between intervals are more easily managed by the user.

Actions and plans are characterized by a start time and an end time that define the time interval during which the action (plan) is performed. From a user point of view, the temporal structure of a plan is represented through the temporal relations that can be defined between these time intervals in the form of single Allen's interval relations [1]. That is, given two plan components, one of the thirteen Allen's temporal relations can be stated for them. Not all the disjunctions are allowed in the current version of the system. Mainly qualitative temporal relations are represented in this way with the exception of the interval relations "before" and "overlaps" (and their inverses) the extent of which can be further specified by a minimum and a maximum numerical parameter. As seen previously, the interval relations represent the temporal dimension of a plan at a higher level (the user level) and they are mapped to a set of metric constraints representing bounded differences between the interval endpoints. Figure 2 shows

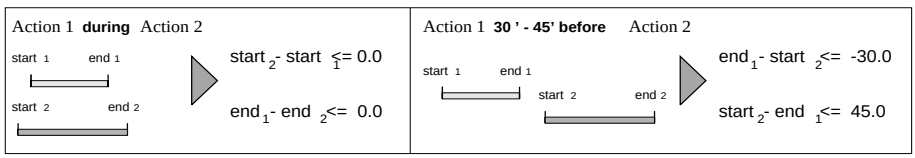

Fig. 2. Examples of correspondence between interval relations and bounded difference constraints between the points representing the start time and the end time of actions.

an example of how Allen's relations are translated into bounded difference constraints between time points. A bounded difference constraint between two time points x and y can be represented by the inequality $y - x \leq a$. Qualitative relationships can be represented using infinity values as distance bounds. For example "x before or equal to y" is expressed constraining the distance between x and y to between zero and infinity ($0 \leq y - x \leq \infty$). Other temporal information of the plan represented at this level is that relative to the action duration depending on the work capacity and the number of resources employed and those relative to plan sector deadlines that are derived from the requirement of having completed the plan before the fire reaches that sector. So estimates on minimum and maximum action duration bound the distance between the start and end times of the action, while a sector deadline gives an upper bound to the distance

between the start and end times of the sector plan. Note that all this information can be represented with bounded difference constraints. A set of bounded difference constraints defined on a set of time points can be cast into a CSP problem defined on variables with continuous domains [9, 25]. This approach (as described in the next section) was followed so the temporal dimension of a plan is represented by a constraint network whose variables are the time endpoints of each plan component. Time variables take values into real closed intervals. The directed arcs between variables x and y, y and x respectively, are labeled with the bounded difference constraints $a \leq y - x \leq b$ that bound the distance of the two variables on which they are defined.

2.3 System Functions

Plans are retrieved from memory by the case-based module. The temporal structure as stored in memory (constraints between plan's components) has to be installed, that is a temporal constraint network has to be defined on the time variables according to a script that is stored in memory. After that step, this data structure has to be fitted and modified to adapt it to the specific scenario data. Part of this process is done in batch mode by the system (fitting), part is performed in collaboration with the user (adaptation). The typical temporal reasoning tasks performed by the planner during plan fitting and adaptation can be stated as follows: recompute and propagate plan step durations, check if a plan can be performed before a given temporal deadline, check if a new action can be inserted in the plan network without violating the deadline, compute earliest and latest start (end) times for the action (the plan). This requires dealing mainly with metric information. Temporal variables and constraints are added to the plan by the deadlines of the sectors defined in the current scenario. Plan step durations and feasible values of the temporal variables are recomputed. The main interactive planning funtionalities provided by the system follow. They require updating the underlying temporal constraint network (a-d), modifications of variable domains (e), constraint modifications (f):

a. New actions can be inserted.
b. New interval relations can be inserted.
c. Actions can be deleted.
d. Interval relations can be deleted.
e. Start time and end time of a plan step can be anticipated or delayed.
f. The duration of an action can be modified.

A brief description is given of how the user can perform plan editing during a typical planning session. Figure 3 depicts the part of the man-machine interface devoted to plan selection and editing. After the user has selected one of the candidate plans retrieved from the memory, both the plan hierarchy (see the upper window in Figure 3) and the plan temporal structure (see the lower window in Figure 3) are displayed. Now the user can edit the network of actions that define the plan. Actions are displayed as dark bars in the editor window, earliest

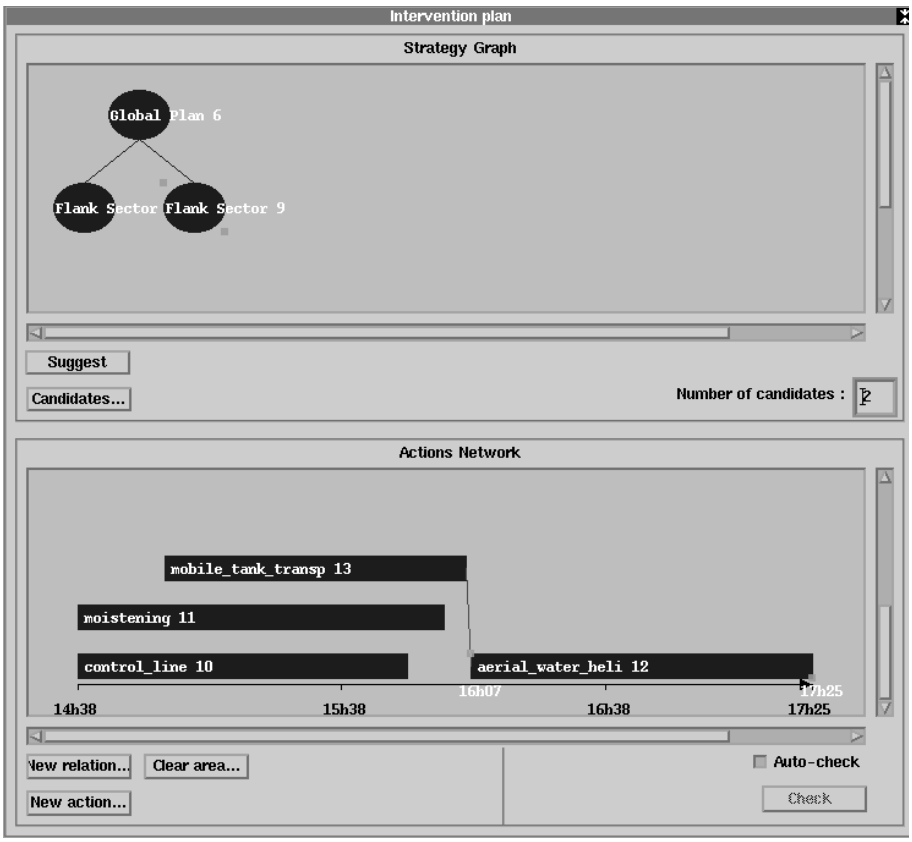

Fig. 3. A snapshot of the man-machine interface window reserved for plan editing.

start/end times are visible. Temporal relations between actions are displayed as line segments connecting them. Additional information, like the type and number of resources associated to an action, can be shown in a panel that will be opened when double clicking with the mouse an action bar. The user can try to modify the start (end) time of an action by moving the left (right) corner of the box to the desidered time. Also the duration or the number of resources of an action can be modifyed overwriting the current values presented on the action information panel. A new action or a new temporal relation between two actions can be inserted. Each modification proposed by the user will be accepted if it doesn't cause any inconsistency and all the variables of the network will be updated correspondingly. When the editing phase is concluded the user can ask the system to allocate resources on the resulting plan.

3 The Constraint Reasoner Module

A partial view of the system architecture is depicted in Figure 4. This picture shows the main components: the case-based reasoner, which retrieves (stores) plans from (to) the plan memory; the plan space, which is the data space where plans are installed; the action manager, that manages the user requests to modify the plans installed in the plan space; the constraint reasoner, in which variables and constraints are created and constraint reasoning algorithms called. The con-

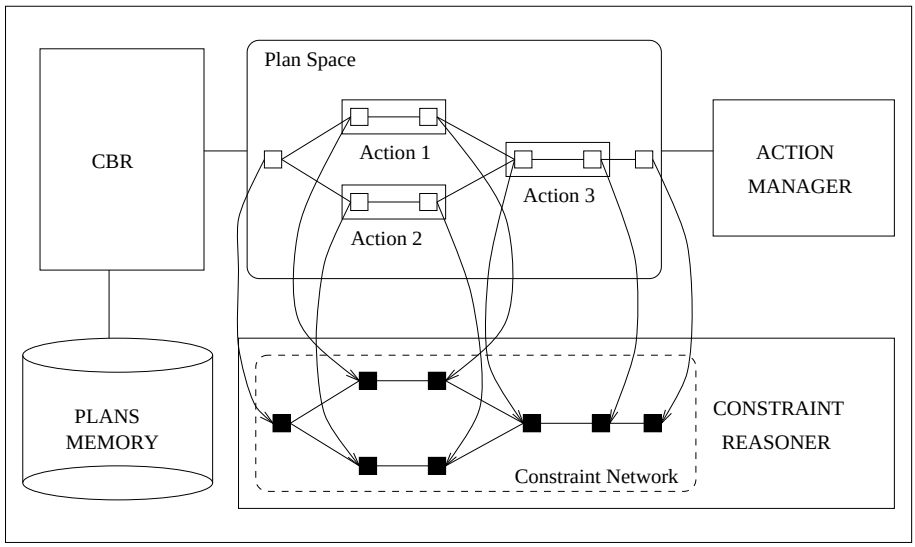

Fig. 4. The architecture of the planner.

straint reasoner module is implemented in an object oriented language, as are the other components of the system. The object hierarchy of the language was extended with a new set of classes that enable problem solving in a broad set of constraint satisfaction problems. This integration of constraints and objects was developed taking into account basic requirements such as that of maintaining all the flexibility of the object-oriented languages and that of constraints being objects themselves, as extensively discussed in [11]. This approach follows the line indicated in [2]. Here only the main abstractions of the system are presented. The reader interested in a more detailed description can refer to [24].

`c_variable` This class represents the concept of variable in a constraint network. If some attribute of an object has to be constrained it has to be filled with a `c_variable`. For example the starting time of an action_net is represented as an attribute of the class. This attribute can be filled only by an object of type `c_variable`, or of some subtype of this type (in fact it is declared `r_variable` because the system distinguishes between finite domain and

continuous domain variables). In Figure 4 this relation between constrainable attributes and variables of a constraint network is represented with an arrow (pointer) from a component of an action in the plan space to a constraint network variable in the constraint reasoner module.

`domain` This class represents the concept of domain of a variable in a constraint network. Currently two domain types have been implemented: `d_domain` and `r_domain`. `d_domain` implements the possible values of the variable with a sequence of objects. `r_domain` implements the real interval $[a, b]$ with a sequence of two floats.

`constraint` This class represents the abstraction of constraint. Specific constraint types are implemented as subtypes of this class. So for example all the bounded difference constraints are instances of the class `y-x<=a` which is a subtype of `constraint`. New constraint types can be defined quite simply if they have a finite domain. Essentially only the corresponding boolean test, which is a polymorphic method, has to be redefined. For continuous domain constraints the user has to define what are called the refine functions [8] or solution functions [14].

`cnetwork` This class represents the constraint network abstraction. Each time a new constraint satisfaction problem is to be defined an object of type `cnetwork` has to be created. Constraint instances and variable instances have to be linked to this new object (pointers are used extensively). The instances of constraints and variables are also linked. So a constraint instance maintains a pointer to the variables to which it is linked. Conversely each variable maintains a pointer to the constraints that are linked to the variable. The main constraint reasoning techniques are implemented as methods of this class. In this way polymorphism can be applied and the type of constraint network dispatches the correct solver. The system can deal with many types of networks: `tcnetwork` is a specific subtype of cnetwork having constraints of type `y-x<=a` and variables of type `r_variable`; `dcnetwork` are finite domain constraint satisfaction problems. A `tcnetwork` is associated to each `action_net`. It contains all the `r_variables` associated to start and end points of the `action_net` and of its components and all the `y-x<=a` constraints defined on this variable for representing temporal information such as duration of plan steps, precedence relations and so on.

Figure 5 depicts a simplified Booch diagram showing the abstractions previously introduced. Note that a simple arrow indicates class subtype relation and the double line indicates a 'use for' relation. It is out of the scope of this paper to describe all the algorithms that were implemented. Essentially they can be divided in two groups: the first devoted to constraint propagation and solution finding for discrete domain constraints, the second devoted to constraints checking on networks of bounded difference constraints defined on continuous domain variables. The first group includes a propagation algorithm based on arc consistency (AC3 [16]) that is generalized to handle also the continuous domain case in a similar way as other authors have done [10, 15]. Moreover other classical hybrid algorithms for solution finding were implemented, they perform

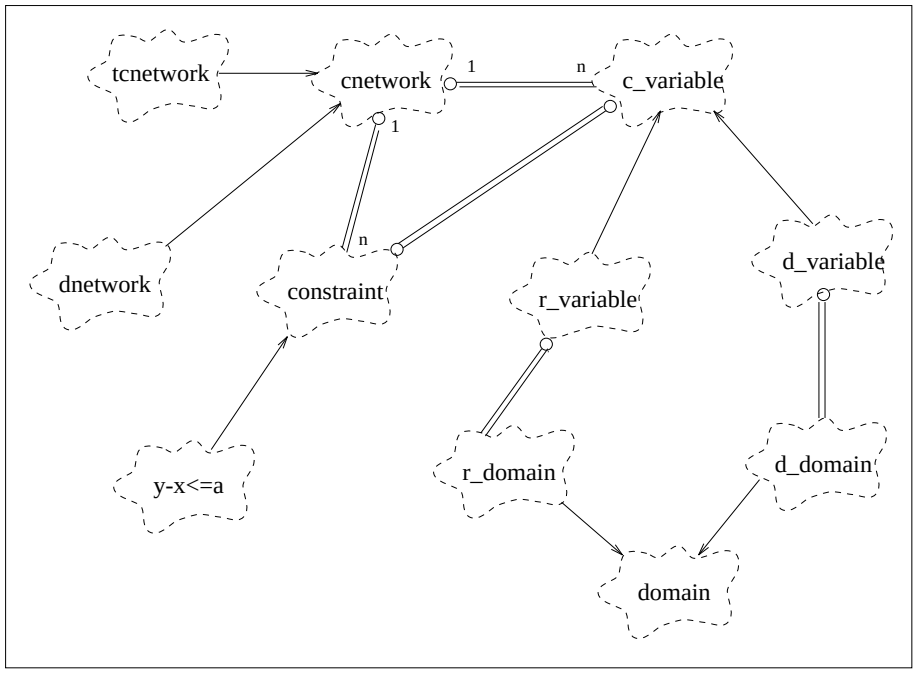

Fig. 5. A simplified Booch diagram of the constraint reasoner module.

propagation during search and dynamic variables reordering [17, 20] as forward checking, partial look ahead, full look ahead and most constrained search.

The second group of algorithms has been coded in a set of methods of the constraint reasoner that support the system functions described in the previous section, as explained in the following. In general the global consistency of a temporal network TCN with $\{X_1, \ldots, X_N\}$ variables and $\{X_j - X_i \leq a_{ij} \mid i = 1, \ldots, N,\ j = 1, \ldots, N\}$ constraints, is checked using shortest path algorithms on the corresponding distance graph. The graph is a directed edge-weighted graph $G = (V, E)$, in which each node $i \in V$ represents the temporal variable X_i, and each edge $(i, j) \in E$ is labeled by a weight a_{ij}, and represents the linear inequality $X_j - X_i \leq a_{ij}$ [9, 7]. In [9] it is shown that these techniques provide a global consistency check tool and produce the minimal network representation (a representation of the constraints that is minimal with respect to the natural order of constraint networks $\subseteq$).

The Floyd-Warshall's [7] all-pair shortest path algorithm is used for computing the minimal network and the Bellman-Ford's single-source shortest path for testing the network consistency and for computing earliest times respect to a reference time.

Each time a user function is called, like adding a new constraint or modifying the definition of a previously installed constraint, the TCN, implemented as a `tcnetwork` is checked against possible inconsistencies and the new domain ex-

tensions for the temporal variables are computed using shortest path algorithms. In such a way the user is enabled to incrementally define a plan, by adding or removing actions and constraints, while the system checks the correctness of these operations with respect to the temporal dimension.

The planner, including the constraint reasoning module, has been developed in the Object-Oriented language Spoke[2]. The complete system (Planner plus Situation Assessment and Resource Manager) have been developed on a software platform that integrates Spoke 3.3.0, URIAH 2.0[3] a Geographical Information System and the relational data-base Informix 5.0[4].

4 Conclusion and Future Work

This paper presents a partial view of a system for planning first intervention attacks to forest fires. Constraints play a central role in managing the temporal information related to a plan. Constraints provide a tool for modeling part of the knowledge involved in this application. A smooth integration of the knowledge modeled with constraints is attained using an object-oriented design methodology and an object-oriented implementation language.

The solvers implemented so far are well documented in the literature and they do not represent an advancement with respect to the current state of research. Now we are facing the problem of developing more specialized algorithms. The major line of improvements involves the algorithms for supporting temporal reasoning. Traditional algorithms do not support a process in which problem definition and problem solving interleave. The problem is generally defined and then solved. In the application here presented the plan is built interactively in collaboration with the user and furthermore replanning can occur when a schedule cannot be found or a resource allocation step fails. It is an interesting and open field of research to provide tools for dynamic updating of a constraint network and for checking the constraint satisfiability [18, 6]. Two lines of research are under investigation; first, to develop incremental graph based algorithms for shortest path; second, to separate the satisfiability of the network from the search for the minimal network representation. In this second approach one can use an algorithm for dynamically maintaining the loops in the constraint graphs for the first task, and arc-consistency for dynamic networks for the second task [18, 4, 21].

References

1. J. F. Allen. Mantaining knowledge about temporal intervals. *Communication of ACM*, 26(11):832–843, 1983.

[2] Spoke is a trademark of Alcatel/ISR.
[3] URIAH is a trademark of 3IG.
[4] Informix is a trademark of Informix Software, Inc.

2. P. Avesani, A. Perini, and F. Ricci. COOL: An object system with constraints. In J. Bézivin, B. Meyer, and J.-M. Nerson, editors, *Technology of Object-Oriented Languages and Systems*, pages 221–228, 1990.
3. P. Avesani, A. Perini, and F. Ricci. Combining CBR and constraint reasoning in planning forest fire fighting. In *Proceedings of the first european workshop on Case-Based reasoning*, pages 235–239, Kaiserslautern, 1993.
4. C. Bessière. Acr-consistecy in dynamic constraint satisfaction problems. In *Proceedings of the Nineth National Conference on Artificial Intelligence*, 1991.
5. G. Booch. *Object oriented design with applications.* The Benjamin/Cummings Publishing Company, 1991.
6. R. Cervoni, A. Cesta, and A. Oddi. Managing dynamic temporal constraint networks. In K. Hammond, editor, *Proceedings of the second international conference on artificial intelligence planning systems*, pages 196–201. 1994.
7. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms.* The MIT Press, 1990.
8. E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32:281–331, 1987.
9. R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49, 1991.
10. B. Faltings. Arc-consistency for continuous variables. *Artificial Intelligence*, 65:363–376, 1994.
11. B. N. Freeman-Benson and A. Borning. Integrating constraints with an object-oriented language. In *Proceedings of the 1992 european conference on object-oriented programming*, pages 268–286, 1992.
12. K. J. Hammond. *Case-Based Planning: Viewing Planning as a Memory Task.* Academic Press, Boston, 1989.
13. T. R. Hinrichs. Towards an architecture for open world problem solving. In J. L. Kolodner, editor, *Proceedings of the 1988 DARPA Workshop on Case-Based Reasoning*, pages 182–189, 1988.
14. E. Hyvönen. Constraint reasoning based on interval arithmetic: the tolerance propagation approach. *Artificial Intelligence*, 58:71–112, 1992.
15. O. Lhomme. Consistency techniques for numeric CSPs. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, Chambéry, France*, pages 232–238, 1993.
16. A. K. Mackworth. Consistency in network of relations. *Artificial Intelligence*, 8:99–118, 1977.
17. B. A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989.
18. B. Neveu and P. Berlandier. Arc-Consistency for dynamic constraint satisfaction problems: an rms-free approach. In *ECAI94 Workshop, Constraint Satisfaction Issues Raised by Practical Applications.* 1994.
19. M. Paltrinieri. Some remarks on the design of constraint satisfaction problems. In A. Borning, editor, *PPCP'94: Second Workshop on Principles and Practice of Constraint Programming*, Seattle WA, May 1994.
20. P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299, 1993.
21. P. Prosser, C. Conway, and C. Muller. A distributed constraint maintenance system. In *Proceedings of Expert Systems and their Appplications*, Avignon, France, 1992.

22. F. Ricci, S. Mam, P. Marti, V. Normand, and P. Olmo. CHARADE: a platform for emergencies management systems. Technical Report 9404-07, IRST, 1994.
23. F. Ricci, A. Perini, and P. Avesani. Planning in a complex real domain. In *proceedings of the italian planning workshop*, pages 55–60, Rome, 1993.
24. F. Ricci, A. Perini, and S. Pillorget. The constraint reasoner module specifications. Technical report, IRST, 1994. CHARADE Restricted Report #50A.
25. E. Schwalb and R. Dechter. Temporal constraint satisfaction problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 127–132, 1993.

This article was processed using the LaTeX macro package with LLNCS style