Dynamic Constraint Satisfaction with Conflict Management in Design

Esther Gelle and Ian Smith

Laboratoire d'Intelligence Artificielle
Ecole Polytechnique Fédérale de Lausanne (EPFL)
IN-Ecublens, CH-1015 Lausanne

Abstract. In this paper, we focus on techniques for constraint-based design tasks with *continuous variables* in dynamic environments. We show how design spaces can be explored within an intelligent CAD system using a dynamic constraint satisfaction framework and a conflict resolution paradigm. Navigation between different design spaces is controlled by assumptions the designer makes in situations of incomplete knowledge. These assumptions divide design constraints into defaults, preferences and fixed constraints. Design strategies associate the constraints with the overall goals that the designer wishes to satisfy and allow for a more detailed ordering on the constraints. Reasoning within different belief sets which correspond to the designer's notion of assumptions is related to normal default logic.

1 Introduction

Synthesis tasks such as configuration and design are characterized by an imprecise, often ill structured initial state. At the time a design problem is specified, requirements are not completely identified. When the system is under-constrained, the designer makes assumptions in order to advance the design process and to focus on interesting design alternatives. In this situation, the designer does not always make optimal decisions. Factors and parameters affecting a decision may interact in such a complex manner that the designer becomes unable to decide the best course of action. He may be unable to process good choices or even to generate all alternative courses of action. The best design process is therefore a *dynamic* one where, at any stage of design development, decisions involved can be changed and intermediate solutions are re-evaluated. These requirements show the need for a framework which generates various design alternatives in order to explore the design space in a rational way.

The paradigm of constraint satisfaction is useful for expressing design tasks and subtasks since it accomodates variables and mathematical relations. However finding the relevant constraints and expressing them is much harder. Therefore a constraint-based framework for design should support the exploration of multiple solution spaces and give the designers the possibility to introduce new constraints and their preferences. To satisfy these requirements our system uses

1. Dynamic constraint satisfaction on continuous and discrete variables. Mittal & Falkenhainer [9] have developed a framework for treating constraints on

discrete variables in a dynamic environment. We show how to extend this framework in order to include continuous variables and constraints. Certain aspects of the design process are related to normal default logic.
2. Default and preference assumptions on constraints in order to guide the search of consistent constraint combinations, especially in early stages of the design process.
3. Conflict management methods for detecting and solving inconsistencies arising in over-constraint situations.
4. A general hierarchy on design criteria inducing an ordering on the constraints important for conflict management.

2 Related Work

Sketchpad [11] was the first constraint-based drawing system using constraint satisfaction and relaxation methods on real values. Borning et al. [2] define hierarchies of constraints in their graphics-based drawing system in order to satisfy constraints on graphical objects in an intuitive manner. The functionalities of such an interactive graphics system allow for a simple classification of constraints into required, preferential and default constraints. Another requirement is that the system's response to user requests should be fast. Different types of comparators are implemented to filter out the best possible solution. In a design system, such a simple hierarchy is no longer useful. Comparators are difficult to define for a given design task because often there is more than one hierarchy of preferences. In this cases, users should be able to define their own hierarchies based on design considerations such as cost or esthetics. They might also be interested in several solution spaces; not just a single solution. Freuder et al. [5] describe a related notion of partial constraint satisfaction on discrete variables. Their branch and bound algorithm finds values for a subset of variables that satisfy a subset of constraints. It uses a metric measuring the distance between the current values and the best solution found so far. Preferences in this system can be expressed by assigning weights to the constraints. However, in a design system it is preferable to define a hierarchy based on design criteria rather than on individual constraints.

In the domain of configuration, Mittal and Falkenhainer [9] have defined a framework for dynamic constraint satisfaction based on discrete variables. For design examples however, their definition of activity and compatibility constraints must be adapted in order to treat continuous as well as discrete variables. Other systems allowing for dynamic introduction of constraints include constraint logic programming (CLP) languages. CLP describes a set of languages based on logic programming with embedded constraints and defined on different domains, such as finite, real or interval domains. A CLP program consists of a set of rules. Each rule has a head, the goal, and a body consisting of subgoals and/or constraints. During the execution of the rules, a goal is blocked until its condition is satisfied by the active constraints. One of the most powerful languages for computing numeric systems is CLP(intervals) [1]. It uses box-consistency,

an approximation of arc-consistency, to solve numeric constraint systems. Other CLP($\Re$) systems use the simplex algorithm to treat linear constraint systems and delay nonlinear systems until they become linear. One of the major problems in the domain of numeric constraints is the resolution of nonlinear systems which may comprise inequalities. In order to solve a design problem, a unique solution must be found within the solution space. Haroud et al. [6] have developed an algorithm guaranteeing global consistency for continuous constraint satisfaction problems (CCSPs). The advantage of this algorithm is that feasible regions are represented explicitly in the three-dimensional space. The designer can choose a single feasible solution within the solution space. This algorithm needs to know beforehand the set of constraints involved in a problem, i.e. the problem is *static*. Further work will be necessary to adapt it to a dynamic design framework.

3 Dynamic Constraint Satisfaction

Synthesis tasks can be represented naturally as dynamic constraint satisfaction problems (DCSP). In design and configuration, variables define components and properties of the artifact to be created and constraints represent relationships between objects. The sets of variables and the constraints defined on them change dynamically in response to decisions taken during the design process. Typically, the components of an artifact are interconnected: For instance when configuring a laptop, if an efficient secondary cache is required, a cooling fan has to be built in. Thus, the weight of the laptop will change as well as its price. Consequently, the existence of some variables, such as the fan, or constraints is entirely determined by earlier decisions. Simultaneously considering the entire set of possibly relevant constraints would result in a large number of constraints and variables to treat. A large set of conflicts would arise, which would be difficult to solve. In dynamic constraint satisfaction, the activation of a new variable or constraint is *conditional*. A variable or a constraint is called *active* when it must be part of a solution. Not all variables have to be assigned a value to solve the problem and not all constraints are relevant for a given problem. To determine a variable or constraint's activity the *activation conditions* are matched against the current context. A *design context* is defined as the set of active variables and their values, i.e. a partial solution space.

Activation conditions may split up the solution space into disjoint regions. A further difficulty arises, when activation conditions are defined on continuous variables; the subspaces they define are no longer enumerable. In Figure 1 for example, the width W and the height H of a steel structure element of shape I depend on several constraints, in this case linear inequalities, which define multiple solution spaces (shaded regions). The type of the section, which can be plastic, compact or non-compact, induces three different classes of ratio of H and W, denoted Class 1, 2 and 3. The constraint $W \leq w1$ defines sections of elements that must be enforced by stiffeners, whereas no stiffeners are needed when $W \geq w2$. In this example, $w1$ and $w2$ are constants. In general, a constraint satisfaction problem on continuous variables is can be solved in polynomial time

when the regions defined by the constraints are arc-wise connected [6]. In the presence of explicit splits in the problem definition as for example shown in Figure 1, it is wise to combine constraint satisfaction with search. Consistent constraint combinations are found and values are then propagated in each subset.

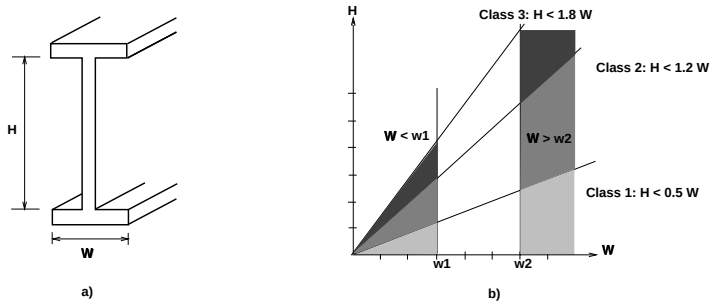

Fig. 1. *a) A structural I element, b) disjoint solution spaces defined by linear constraints on the width W and the height H for a structural element.*

3.1 A Formal Definition of DCSP

In terms of definitions by Mittal et al. [9] *compatibility* constraints define static relations on variables whereas *activity* constraints specify constraints on the activity of variables depending on possible values of other variables. In order to formalize a design problem, we need to enhance the definitions of activity constraints. Not only variables but also constraints may be activated. Consider the following example of calculating the surface of a shape which can be a rectangle or a circle:

R1 $shape = rect \rightarrow active(surface = length * width)$
R2 $shape = circ \rightarrow active(surface = \pi * radius^2)$
R3 $shape = rect \rightarrow active(length, width)$
R4 $shape = circ \rightarrow active(radius)$

The example shows that constraints can be conditional too (R1,R2). In a dynamic CSP, we are given a set of possibly active variables X, a set D defining the union of all domains of X so that each variable $x_i \in X$ has associated a domain $d_i \in D$ and C, the set of potentially active constraints. An instantiation of a set of variables $X' \subset X$ is a tuple representing an assignment for each variable. A solution is a consistent instantiation, so that all the constraints between the variables of X' are satisfied and no more variables and constraints can be activated. A solution space S is a subset of D comprising all solutions. Contrary to a static CSP, we distinguish between active variables and such which are not active. A variable is called *active* when it must be part of the solution. A variable

x_i which is active, has to be assigned values from its domain $d_i \in D$. Similarly, a constraint which is active must be satisfied by a solution s of the solution space S:

$$active(Y_i) \Leftrightarrow \{y_i = d_i | d_i \in D(y_i), y_i \in Y_i\}$$
$$active(constraint(Y_i)) \Leftrightarrow \exists s \in S\{active(Y_i) \wedge satisfies(s, constraint(Y_i))\}$$

We define as activity constraints C^A rules of the form

1. $Y_1 \rightarrow active(Y_2) \quad Y_1, Y_2 \subset X$
2. $predicate(Y_1) \rightarrow active(Y_2)$
3. $predicate(Y_1) \rightarrow active(constraint(Y_2))$

and compatibility constraints C^C

4. $constraint(Y_1) \quad Y_1 \subset X$

More generally, a dynamic constraint satisfaction problem (DCSP) is defined as a set $\langle W, R, X, D \rangle$ with a set of rules R and a set of initial assignments W. R is divided into activity constraints C^A and compatibility constraints C^C. A rule consists of an *activation condition* P_i and its body introducing a new constraint $C_i \in C$ (1). X is the set of variables that may be part of a solution.

$$P_i(Y_1) \rightarrow active(C_j(Y_2)) \Leftrightarrow P_i(Y_1) \stackrel{act}{\rightarrow} C_j(Y_2) \quad Y_1 \subset X, Y_2 \subset X^1 \qquad (1)$$

In its most general formulation, the activation condition P_i is a conjunction of constraints on a subset of variables Y_1 which have to be active. The constraint C_j is a mathematical expression, linear or non-linear, on the subset of variables Y_2. In order to express a compatibility constraint, the consequence of the rule simply evaluates to true. A rule $r \in R$ is satisfied by an assignment s if and only if:

$$active(Y_1) \wedge satisfies(s, P(Y_1)) \wedge active(C_j(Y_2))$$

A solution space S for a given problem $\langle W, R, X, D \rangle$ is a subset of variables $X' \subset X$ each of which has been assigned a range of values consistent with respect to $C^A \cup C^C$ so that $W \subset S$. A solution space S is therefore consistent with respect to all active constraints and no more variables and constraints can be activated.

[1] The notation $P_i(Y_1) \stackrel{act}{\rightarrow} C_j(Y_2)$ is equivalent to IF $P_i(Y_1)$ THEN $C_j(Y_2)$.

3.2 Hierarchy of Constraints

When the given problem is underconstrained, several solution spaces exist in the search space. In order to avoid enumerating all solution spaces, we aim at finding a solution space which fulfills some given design criteria and assumptions of the designer. When the problem is overconstrained, no solution space exists. In this case, we would like to find solution spaces violating as few important constraints as possible. We introduce therefore a hierarchy of constraints, which helps us to guide search in the case of several solution spaces and which aims at relaxing constraints which are the least preferred when the problem is overconstrained.

We define a hierarchy $H : h_1 \ldots h_n$ on the constraints by labeling the rules of R, so that $r_i < r_j$ with $r_i, r_j \in R$ if $r_i \in h_k$ and $r_j \in h_m$ with $k < m$, i.e. r_i is preferable to r_j. Each h_i consists of a set of rules which are at the same hierarchical level.

A solution s in the solution space S is defined to satisfy as many levels of the hierarchy as possible, starting with h_1 being the rules introducing required constraints [2]. The other levels $h_j, j = 2, \ldots n$ are optional and satisfied if possible. The rules are treated in the order of the overall hierarchy H. If a conflict has to be solved between two constraints initiated by two different rules r_i and r_j, the constraint which is less preferable, i.e. the constraint initiated by r_j if $r_i < r_j$, is chosen for relaxation.

3.3 A Comparison with Default Logic - Open Issues

In this section, we want to show how dynamic constraint satisfaction approximates the paradigm of default logic. In a dynamic constraint satisfaction problem, rules add constraints which first have to be checked against the current constraint network. The underlying reasoning process in a DCSP is *non-monotonic*. Suppose that a given precondition P_1 satisfies the current context (the values of variables found so far) and a new constraint C_1 is introduced. Adding a new rule $P_2 \rightarrow C_2$ where P_2 also satisfies the current context does not automatically result in the addition of the constraint C_2. Adding the constraint C_2 may result in a conflict. Hence, the addition of new rules does not automatically mean the addition of more constraints. The general rule structure $P_i(Y_1) \stackrel{act}{\rightarrow} C_j(Y_2)$ is not just a logical implication but incorporates a consistency check for the relevant constraint. It can be read as:

IF $P(Y_1)$ is satisfied by the current network AND $C(Y_2)$ does not result in a contradiction THEN add $C(Y_2)$ and propagate its results.

The search process within a DCSP augmented with a constraint hierarchy can be compared with techniques used in default logic [10, 8]. A default logic is a proof system $< W, R >$ where W is an initial theory and R consists of default rules of the form

$$\frac{\phi : M\beta_1 \ldots M\beta_n}{\psi} \tag{2}$$

which can be read as *under the assumption that* $\beta_1 \ldots \beta_n$, *if* ϕ *then* ψ. Default rules not only consist of a precondition (here the clause ϕ) but also of a justification $\beta_1 \ldots \beta_n$. The role of this justification is that it describes the exceptions blocking the applicability of the default rule in (1). As long as none of the $\neg\beta_i$ belong to the context S, the rule is applicable. S is a belief set, a context, from which so-called *extensions*, i.e. solutions are derived. Such a belief set S determines a subset of the default rules which are applicable. The existence of a proof for a formula f from W using R and S means that f itself is believed and therefore belongs to S. It represents the assumptions under which an extension is derived. Finding consistent solutions in a default theory $< W, R >$ consists of:

1. Guessing a belief set S.
2. Proving that all the consequences derived from W and R with respect to S equal S.

The solutions derived from belief sets using the rules are themselves part of the belief set and can be found using different techniques. One technique generates all elements of the power set of the rule consequences and checks them for consistency with the belief set S and the theory W and R. Another technique uses a well-ordering[2] to treat default rules with given priorities (prioritized defaults). User-defined priorities $\leq$ on the rule set R allow for generating a well-ordering $\preceq$ so that for each r_i,$r_j \in R$ if $r_i \leq r_j$ then $r_i \preceq r_j$. Given the order $\leq$: $r_1 < r_2$ and $r_1 < r_3$ among three rules $\{r_1, r_2, r_3\}$, for example, two possible well-orderings $\preceq$ can be generated, $\{r_1, r_2, r_3\}$ and $\{r_1, r_3, r_2\}$. The rules are processed sequentially with respect to the given well-ordering using one-step operators.

The rules of a DCSP can be modeled in *normal default theory*, a special case of default theory. The rule justifications in a normal default theory coincide with the consequent of the rule. For example:

$$\frac{\phi : M\psi}{\psi} \tag{3}$$

The rule is read as *under normal circumstances, if* ϕ *then* ψ. The justifications have to be proved within the current belief set. The rules in a dynamic constraint satisfaction can be given the same logic representation:

$$\frac{P(Y_1) : MC(Y_2)}{C(Y_2)} \tag{4}$$

The current belief set is the assumption that the constraints belong to the solution, i.e. they are *active*. In a DCSP, consistency checking among the active constraints provides a proof for believing in a constraint added to the active ones. Our hierarchy defined on rules is used in a similar way as in default logic to derive a consistent solution with respect to the given ordering.

[2] A well-ordering is a partially ordered set which is connected and each subset of which has a least element. There is a homomorphism between a well-ordered set and the set of ordinal numbers.

4 Bridge Design

In this section, we describe our implementation in the domain of preliminary bridge design. We sequentially process rules using dynamic constraint satisfaction and an ordering of the rules induced by a hierarchy on design criteria and by assumptions.

4.1 Knowledge Representation

Structural knowledge of the bridge is represented by generic components in the CAD system ICAD[3], which uses a parametric model of the artifact to be created. Design knowledge is represented by design rules each as described in section 3.1. The variables are given an initial range of possible values (intervals in case of real variables). The rules are clustered in consistent rule sets representing different design criteria. Typical criteria are esthetic, structural, economic or safety considerations. They allow the designer to define an ordering on the rules by which the evolution of the design process is controlled. Table 1 shows an example in preliminary bridge design. Initial conditions on the region where the bridge is to be built are entered into the ICAD interface. These include the section of the region, the obstacle over which the bridge is built etc. A geometrical interpretation of the section results in further qualitative information, for example $opposing\ slopes = similar$ or $soil\ condition = good$ illustrated in Figure 3.

4.2 Forward Reasoning with Assumptions

Rule sets are activated in the ICAD model according to the chosen order of design criteria. At each step (corresponding to one criterion), a new rule set is loaded. The design rules that satisfy the current design context are then sequentially processed activating constraints and variables. Each activated constraint is added to the current constraint network and new values are inferred through constraint propagation. If no conflict is detected and the whole rule block has been processed, the ICAD model is updated creating new components if necessary as shown in Figure 2.

A justification-based non-monotonic truth maintenance system (JTMS) reasons explicitly on constraint and variable activity. Variable value pairs are represented as JTMS-nodes and constraints are represented as assumption nodes. If such an assumption is `:in` the corresponding constraint is active and therefore in the constraint network. There are two types of assumptions: default assumptions and preferences. *Default assumptions* are important in preliminary stages of the design process when very little accurate information is available. *Preferences* control and guide search by focusing on interesting alternatives according to design criteria. *Fixed* constraints represent physical relations which always hold. The first rule of the start-up criterion in Table 1 for instance, introduces the default constraint C_3. The following data is recorded[4] :

[3] © Concentra Corporation

[4] The arrow shows data dependencies.

Level	Criteria	Type	Rule
0	physical	fixed	$bridge\ length = environment\ length$ **if** $nb\ of\ spans = n$ **then** $\sum_{i=1}^{n} span_i = bridge\ length$
1	start-up	default	**if** $\frac{1}{10} < aspect\ ratio < \frac{1}{3}$ **then** $nb\ of\ spans = \frac{environment\ length}{environment\ height}$ **if** $nb\ of\ spans = n$ **then** $\forall i(span_i) = \frac{bridge\ length}{nb\ of\ spans}$
2	construction	pref.	**if** $soil\ condition = good$ **then** $construction\ span = 55$ **and** $nb\ of\ spans = \frac{bridge\ length}{construction\ span}$
3	esthetics I	pref.	**if** $environment\ symmetry = \neg symmetric$ **and** $centre\ of\ gravity = centre$ **then** $nb\ of\ spans = 4$ **and** $inner\ spans = \frac{10}{38} * bridge\ length$ **and** $outer\ spans = \frac{9}{38} * bridge\ length$
4	cost	pref.	**if** $angle\ change = abrupt$ **then** $span\ 1 = point\ x - 15$
5	resistance	pref.	**if** $environment\ shape = V\ shape$ **then** $inner\ spans \geq outer\ spans$
6	esthetics II	pref.	**if** $opposing\ slopes = similar$ **then** $span_1 = span_n$

Table 1. *Sample of rules used in preliminary bridge design.*

default(C_3) :in
$(nb\ of\ spans\ [4,5]) \leftarrow (aspect\ ratio\ \frac{70}{300}), (default\ (C_3))$

It depends on the belief in constraint C_3 as well as on the other preconditions of this rule if the inferred values for the spans are :in (or active). Based on new data derived from constraint propagation and justified within the JTMS, the system may activate new rules. When a conflict is detected during constraint propagation, a tms-nogood is installed with the conflicting constraints and conflict management techniques are activated.

A general hierarchy on rule sets is defined in our system. Each rule set is labeled by a design criterion. Rule sets introduced earlier are considered more general than later ones. This order influences the general conflict management strategy and identifies among all the constraints in conflict the one that is preferably relaxed. Earlier default and preference constraints act on a less informed situation than later ones. Therefore the ones earlier introduced are more relaxable. Earlier fixed rules are more resilient. As the rule order proceeds from general to more detailed information later fixed rules are defined by more detailed knowledge. Thus fixed rules introduced later are more relaxable. The hierarchy of design criteria defines the order of activation of the rule sets, as well as the conflict resolution management with defaults, preferences and fixed constraints as explained in section 4.4.

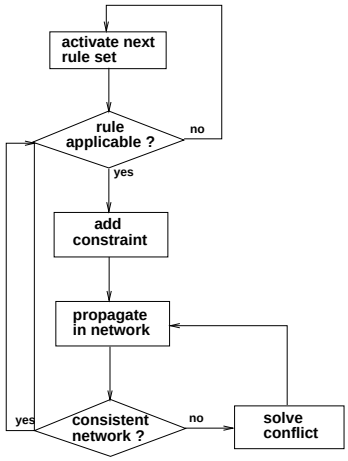

Fig. 2. *General algorithm for DCSP.*

4.3 An Example in Preliminary Bridge Design

The task is:

Given the section of the valley, build a beam bridge in the valley

shown in Figure 3. We will explore two different orders of design criteria presented in table 1: O_1: $0, \ldots, 6$ and O_2: $0, 1, 3, 2, 4, \ldots, 6$. This shows how slightly different rule orderings can result in completely different solutions and how solution spaces can be explored.

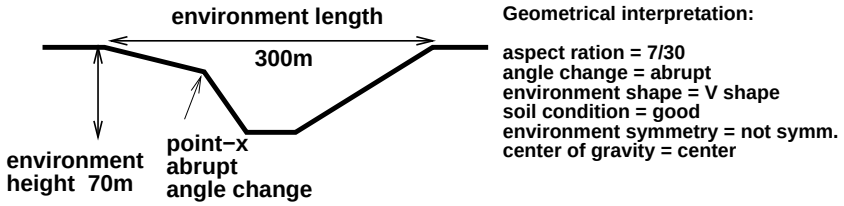

Fig. 3. *The valley entered in the interface of ICAD.*

Solutions for Ordering O_1 In the first ordering O_1 (Figure 4), the start up default rules create a beam bridge with 4 spans equally distributed. $Nbofspans = 4$ is chosen from $nbofspans = [4, 5]$. The construction preference of $nbofspans = [5, 6]$ overrides the default $nb\ of\ spans = 4$ to 5. The spans are still distributed

regularly. The esthetic preference, which has higher priority than the construction preference, weakens [5] $nb\,of\,spans = 5$ again to 4. The spans are distributed so that the inner spans are larger than the outer ones ($outer\ spans = 71$, $inner\ spans = 79$). The cost preference then weakens $span_1$ and $span_2$ to 75 in order to place the first pier near the abrupt angle change in the section. According to the second rule on esthetics, $span_3$ and $span_4$ are weakened to 75 in order to assure that the first and the last span are of equal length.

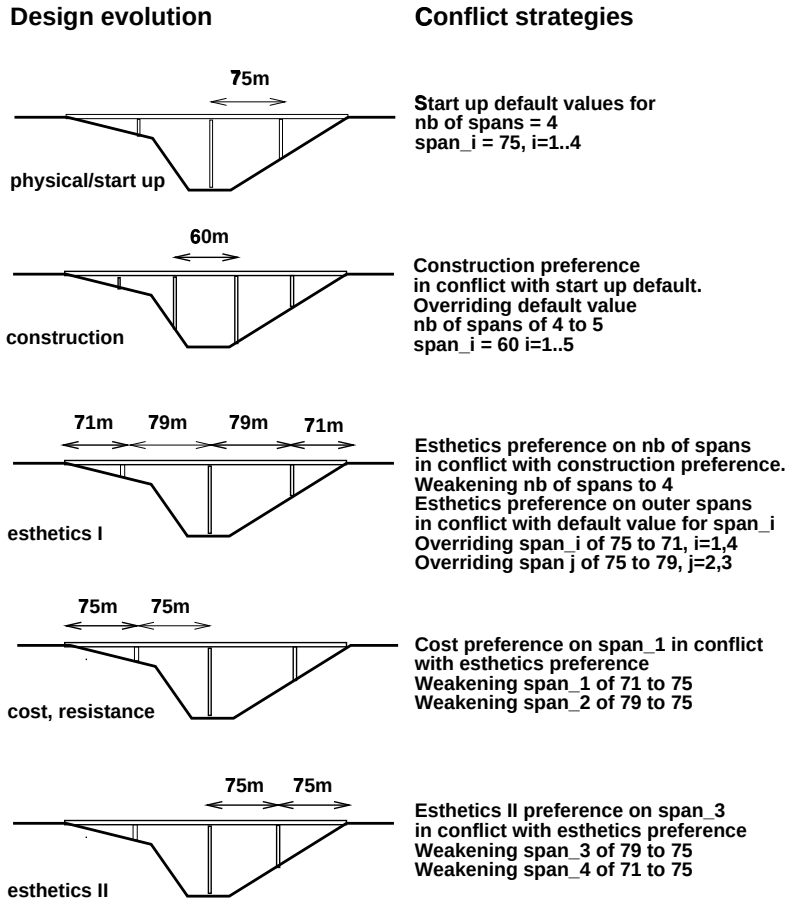

Fig. 4. *Conflict management for the first ordering* O_1.

Solutions for Ordering O_2 In the second solution (Figure 5), the criteria of esthetics I and construction are switched. First, the esthetics I criterion estab-

[5] Overriding and weakening are explained in section 4.4

lishes the same results as in ordering O_1 with 4 spans distributed according to the values of $spans = (71, 79, 79, 71)$. As the construction is more important than the start-up criterion $nb\,of\,spans = 4$ is weakened to 5. $Span_1$ is weakened to [20,51] so that the bridge length does not exceed 300 (second rule of level 0): $spans = ([20, 51], 79, 79, 71, [20, 51])$. The cost preference determines $span_1$ to be 75m and in turn $span_2$ is weakened: $spans = (75, [20, 55], 79, 71, [20, 55])$. The resistance preference then weakens $span_3$ in order to accommodate for the resistance criterion and $spans = (75, [75, 114], [20, 60], 71, [20, 60])$. When introducing the rules of esthetics II, the constraint on $span_4$ is weakened in order to be able to change $span_5 = [20, 60]$ to 75. The result of $span_5 = [20, 59]$, however, does not allow for this change: the cost preference of $span_1 = 75$ has to be weakened too. The design ends up with $span_i = [20, 220]$ for $i = 1 \ldots 5$. The only active constraints are then the constraints of the resistance and esthetics II rules. One possible solution respecting all active constraints might be for example $spans = (58.5, 61, 61, 61, 58.5)$

4.4 Conflict Management

Conflicts in design arise as a consequence of conflicting subgoals when assumptions are introduced in situations of incomplete knowledge or when the solution space is infeasible. Conflict management strategies are based on the nature of the constraints (default, preference and fixed) and their importance for the designer (ordering of design criteria). We distinguish two types of conflicts: *assumption conflicts* and *feasibility conflicts* [7]. In assumption conflicts default or preference constraints are involved. Feasibility conflicts arise when all the default and preference information has been retracted and the conflict is still unsolved. They indicate an infeasible region in the solution space. The three conflict management strategies are:

1. **Overriding defaults**. Default constraints are used to start the design process when only incomplete knowledge of the design object is available. When involved in a conflict they are dropped without consequence and information inferred from them is overridden. Example: $nb\,of\,spans$ [4,5] is overriden by the new value for $nb\,of\,spans$ [5,6] when the construction criterion is introduced.
2. **Weakening preferences**. If a conflict exists, after all the default information has been overridden, preference constraints involved in the conflict are examined. These constraints determine local optimal solutions and should not be dropped completely, therefore they are weakened, and can be reactivated at a later stage. Their activation conditions are not revised and search continues in the same space of activation conditions. Example: after the introduction of the esthetic I criterion $span_1 = 71$ is in conflict with the cost preference $span_1 = point\,x - 15 = 75$. It is weakened to 75 meters because of the higher importance of the cost criterion.
3. **Dependency directed backtracking**. If no assumed information is left in conflict, activation conditions have to be reviewed. In this case, the system backtracks to the last activation condition (in the sense of a defined

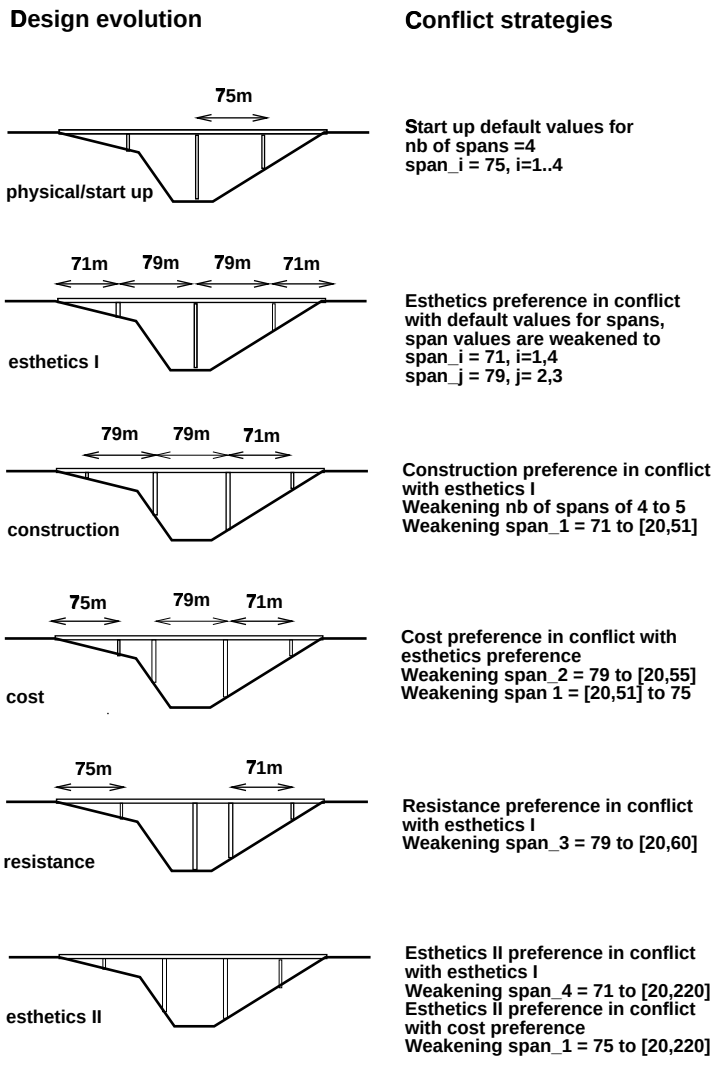

Fig. 5. *Conflict management for the second ordering* O_2.

order of rule sets) and completely changes the solution path. In our example several solutions exist. If none were feasible, the system would switch to *opposing slopes*¬*similar* in the esthetics II criterion.

4.5 Limits of our Implementation

A designer can explore solution spaces by defining an order on the design criteria. Limits of this implementation are:

1. When several choices are possible, the system takes the first one in the order. In the second solution, for example, the system continues with $nb\,of\,spans = 5$ chosen from [5,6] according to the construction rule. In order to get a second solution with $nb\,of\,spans = 6$ the system has to backtrack.
2. It can be quite difficult to evaluate in advance the influence of rule ordering on the evolution of the design. The designer has to explore different orderings to get an idea of the solution spaces.
3. Rules acting on a set of similar variables, as spans for example, introduce sets of similar constraints whereas others act on a specific variable, $span_1$ for example. In the example of the second start up rule, spans are distributed at equal distances. Acting only on $span_1$ and $span_2$ when a conflict arises with the cost rule, may lead to a fairly unesthetical design, where the spans are of unequal length.

Currently, we use the Waltz algorithm [3] for processing continuous constraints. It is known that this algorithm is neither complete nor sound [4]. Therefore the entire process of sequential rule processing is only an approximation of the sequential techniques applied in default logic. A theoretical development is under way in order to achieve soundness and completeness. Most of the current methods for nonlinear constraints only apply to local constraint satisfaction. For nonlinear systems, a complete and sound global constraint satisfaction algorithm is also under development [6].

5 Conclusion

We have shown how solution spaces can be explored using continuous constraint satisfaction algorithms in a dynamic environment. Exploration is guided by ordering the constraints into a hierarchy of criteria and by distinguishing between default, preference and fixed constraints. Theoretically, the problem of dynamic constraint satisfaction can be described in terms of normal default logic. Analogue algorithms of sequential rule triggering are used in dynamic constraint satisfaction where different rule orderings lead to different solutions.

Acknowledgments

Funding for this research was provided by the Swiss National Science Foundation. The authors would like to thank Boi Faltings, Djamila Haroud, Rainer Weigel, LIA, EPFL, and Sylvie Boulanger, ICOM, EPFL, for useful discussions. They would also like to thank the anonymous referees for their comments.

References

1. F. Benhamou, D. Mc Allester, and P. Van Hentenryck. Clp(intervals) revisited. *Technical Report CS-94-18*, University of Marseille, France, April 1994 1994.

2. Alan Borning, Bjorn Freeman-Benson, and Molly Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, 5(3):223–270, September 1992.
3. E. Davis. Constraint Propagation with Interval Labels. *Artificial Intelligence*, 32:281–331, 1987.
4. Boi V. Faltings. Arc-consistency for Continuous Variables. *Artificial Intelligence*, 65:363–376, 1994.
5. Eugene C. Freuder and Richard J. Wallace. Partial Constraint Satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
6. Djamila Haroud and Boi Faltings. Global Consistency for Continuous Constraints. *Proc. of the ECAI-94*, 1994.
7. D. Haroud, S. Boulanger, E. Gelle, and I. Smith. Strategies for Conflict Management in Preliminary Engineering Design. *AIEDAM Special Issue on Conflict Management in Design*, 1995, pages 313–323.
8. V.W. Marek and M. Truszczynski. *Nonmonotonic Logic* Springer-Verlag, 1993.
9. S. Mittal and B. Falkenhainer. Dynamic Constraint Satisfaction Problems. In *Proc. of AAAI-90*, pages 25–32, Boston, MA, 1990.
10. Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.
11. Ivan Sutherland. Sketchpad: A man-machine graphical communication system. In *Proceedings of the Spring Joint Computer Conference*, pages 329–346. IFIPS, 1963.

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