# Interactive Configuration using Constraint Satisfaction Techniques[1]

**Esther Gelle and Rainer Weigel**

Swiss Federal Institute of Technology (EPFL)
IN-Ecublens, CH-1015 Lausanne
Switzerland

E-mail : { Gelle / Weigel } @lia.di.epfl.ch

## Abstract

In this paper, we focus on techniques for incremental constraint-based configuration with discrete and continuous variables. We show how to formalize constraint knowledge using compatibility and activity constraints (Mittal 1990) and how this knowledge is used for reasoning within an intelligent CAD system. Most technical as opposed to spatial constraint configuration systems use algorithms for solving discrete problems (Haselboeck 1994). We claim that configuration is both *discrete and continuous* in nature and that new methods for handling both constraints in a unified way must be integrated in configuration systems. Visualization of the globally consistent configuration problem space allows for a systematic exploration of the space in an interactive fashion (Haroud 1995).

Knowledge maintenance in configuration systems must be simplified, because configuration knowledge of todays products evolves over the whole product life cycle. The knowledge representation in deductive rule-based systems as often used in intelligent CAD systems will always be context dependent; maintenance problems resulting from this context dependency are often insurmountable. We have identified the context independence of constraint-based knowledge representation as an important feature for facilitating the incremental development and maintenance of large evolving knowledge bases (Faltings & Weigel 1994).

**Keywords:**
Rule-based vs. constraint-based configuration, incremental constraint satisfaction, interactive configuration and design

[1] Complete version in E. Gelle and R. Weigel, Interactive Configuration using Constraint Satisfaction Techniques, *Second International Conference on Practical Application of Constraint Technology*, PACT-96, London, April 1996

## 1 Introduction

In recent years, manufacturing trends have changed from pure mass-production to a more customer oriented one-of-a-kind production. The main reason for this change is that today's customers have very specific and individual requirements, which can no longer be satisfied by mass-products. The one-of-a-kind production of many consumer and investment products requires powerful modeling techniques and representation methods combined with features which facilitate maintenance and extendability. We claim that the framework of incremental constraint satisfaction offers these features.

**Knowledge formalization:** The advantage of using constraints to formalize design knowledge is that relations between design parameters can be stated without explicitly mentioning the context in which these relations hold. This advantage will be illustrated in section 2. In section 3 we present our framework for dynamic constraints over discrete and continuous variables. In the framework of dynamic constraint satisfaction of Mittal (1990) one can reason about the introduction and retraction of variables respectively constraints during problem solving. This modeling technique is, for reasons of modularity and efficiency, especially useful when large amount of constraints must be handled.

**Interactivity:** Often configuration systems work in a batch-like manner which means that the customer requirements must all be known a priori and are then fed into the configurator to generate for example the bill-of-material of the product. The interactivity in our system leads the user from a rough to a more detailed specification. Furthermore, since we based the reasoning within the system on global consistency of the constraints, we can guarantee that the user cannot move into regions of the search space without solution. Although global consistency is computationally expensive, it is especially useful in interactive systems when working with continuous constraints where an enumeration of the single feasible solutions is no longer possible. In section 4 we will describe a small example showing how consistency techniques are integrated in framework of incremental constraint satisfaction.

```
R1 IF Package = Deluxe
     and Frame = convertible
     THEN Engine = A

R2 IF Package = Deluxe
     and Frame = hatchback
     THEN Engine = B

R3 IF Package = Std
     and Frame = convertible
     THEN Engine = A

R4 IF Engine = A
     THEN Transmission = manual

R5 IF Engine = B
     THEN Transmission = automatic

R6 IF Type = Sportscar
     THEN Frame = convertible

R7 IF Type = Familycar
     THEN Frame = sedan

R8 IF Type = Sportscar
     THEN Transmission = manual
```

Table 1: Rules for Car Configuration.

## 2 Maintaining configuration knowledge

Todays products evolve during their whole life-cycle. This implies that new knowledge must be integrated and old knowledge must be removed constantly from the configuration system. By using a small example we will show that building and maintaining a constraint knowledge base is much easier than building and maintaining a rule-base. Our fictive car company decided to develop a new *funcar* variant of its product line. The effects of adding this new knowledge to a rule-base respectively to a constraint-base will be studied and analysed.

Rules in this example are described in the format "IF variable1 = value THEN variable2 = value" and a simple forward chainer will be used for reasoning. Constraints are represented using tables and the search could be done by a standard backtracking algorithm. Rules and constraints are shown in Table 1 respectively Table 2.

The marketing department of the company decides to introduce a new funcar type and it is the task of the knowledge engineer to enter rules *R9 and R10* shown below into the rule-base.

```
R9 IF Type = Funcar
     THEN Frame = convertible

R10 IF Type = Funcar
     THEN Transmission = half-automatic
```

Simply adding these two rules will render the rule base inconsistent. This can be seen when configuring a *funcar deluxe*. The rule sequence *R9, R1, R4* and *R11* leads to the conflict that the transmission should be manual and half-automatic at the same time. Therefore one needs to modify the rule-base.

```
Step 1 removing Rule 4
     IF Engine = A
     THEN
     Transmission = manual

Step 2 adding Rule 4a
     IF Engine = A and
     Type = Funcar
     THEN
     Transmission = half-automatic

Step 3 adding Rule 4b
     IF Engine = A and
     Type = Sportcar or Type = Familycar
     THEN
     Transmission = manual
```

In the constraint formulation however, only the new tuples arising from the definition of a new funcar type must be added: (funcar convertible) to the **(type frame)** constraint and (funcar half-automatic) to the **(type transmission)** constraint.

**Comparison:** Constraints and rules must be interpreted differently. Consider for example the allowed tuple (A manual) in the constraint between engine and transmission. The constraint must be interpreted as follows: "engine A is compatible with manual transmission" while the interpretation of rule 4 is "every car with engine A will get a manual transmission". The scope of the constraints is local in the sense that new knowledge about funcars for example can not invalidate the constraint knowledge. The scope of the deductive rule on the other hand is global and new knowledge can invalidate the rule as described above.

In systems built using deductive rules, in particular expert systems, the context-dependence results in severe problems of *maintenance* of knowledge in the face of a dynamic world. Even minute changes of technology or changes in the marketing policy require revision of the entire rule set, which can be very costly. In the rule-based approach, adding a

| Package | Frame | Engine |
|---|---|---|
| Deluxe | convertible | A |
| Deluxe | hatchback | B |
| .. | .. | |

| Engine | Transmission |
|---|---|
| A | manual |
| B | automatic |
| A | half-automatic |
| .. | .. |

| Type | Frame |
|---|---|
| Sportscar | convertible |
| Familycar | sedan |
| .. | .. |

| Type | Transmission |
|---|---|
| Sportscar | manual |
| Familycar | half-automatic |
| .. | .. |

Table 2: Constraints for Car Configuration.

new car-type forced us to create a new relation between engine, type and transmission. Expressing knowledge without looking at the context is one of the major advantages of constraint based reasoning. No relation between engine, type and transmission is necessary.

One must furthermore realize, that assigning a single source (e.g. marketing regulations) to the rules 4a and 4b is not possible. Rules from different sources [2] are mixed together in new rules only to get a consistent chaining behaviour. These new rules can be considered as "interface rules" between knowledge sources and they are artificial in the sense that only the chaining behaviour is responsible for their existence. Proving the correctness of those rules becomes cumbersome. The problem of systems built using deductive rules is therefore not only the revision of the rules, which can be very costly. The revisions itself will make the maintenance of the rule-base even more difficult, leading to systems which are no longer manageable.

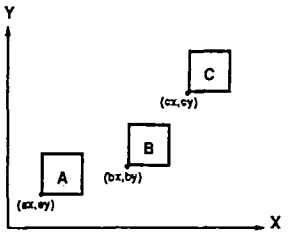
Mechanising the knowledge engineering process is straightforward within the constraint-based framework, because whenever a relation between variables is "established", the knowledge engineer must enter all valid variable-value combinations for that relation. Knowledge engineering in the rule-based framework on the other hand resembles a more hand-crafted approach, since it is possible to delay the engineering of rules until they are needed for a specific configuration. Consider the relation between engine and transmission in the above examples, where the tuple (A half-automatic) in the constraint-based system was "valid" from the very first moment. In the rule-based system on the other hand one could find this piece of knowledge only in "decoded" form within rule 4b, which was added after a contradiction was detected.

[2] The knowledge of rules 4a and 4b stems from the marketing and the engineering department!

# 3 Interactivity and solution spaces

Traditionally, configuration tasks have been reduced to the activity of assembling components of predefined dimensions. Haselboeck (1994) sees the difference between configuration and design tasks in the fact that in configuration tasks the generation of parameter values entirely depends on the structure of the final configuration. There exists no interdependence between the structure of a solution and the final values of system parameters. Therefore, the number of ways the components can be combined is enumerable and the configuration task can thus be modeled as a constraint satisfaction problem on finite, discrete variables. Most practical tasks, however, include objects without predefined ranges of dimensions and continuous variables are needed to describe their properties. Furthermore, such continuous parameters and discrete system parts may be interdependent. Consider, for example, the spatial configuration of 3 objects A, B and C. The position of the objects are described by continuous variables.

**Example: Spatial configuration with continuous variables** Let $a_i, b_i, c_i$ for $i = x, y$ be the $x$ respectively $y$ coordinates of the objects A, B and C. We can formulate a set of constraints involving $a_i, b_i, c_i$.




**C1** $a_i, b_i, c_i \in [0, 10]$ for $i = x, y$

**C2** $b_x \leq {a_x}^2 + 2$

**C3** $b_x \leq 2c_x$

**C4** ${a_x}^3 + 4 \leq c_x b_x$

**C5** $b_y \leq a_y$

**C6** $a_y + {b_y}^2 \leq c_y$

**C7** $c_x \leq c_y$

**C8** $2{b_y}^2 + 2c_x \leq 5b_x$

In order to solve these equations, engineers currently apply a particular sequence of calculations, but never consider the entire space of solutions. They will first solve subsets of constraints and then try to combine these partial solutions by picking one feasible point in a subregion corresponding to a subset of constraints and checking it against the resting constraints.

In constraint-based systems, consistency algorithms are used in order to refine the possible solution space for each variable. In case of an enumerable solution space, search then finds single feasible solutions within the refined space. Depending on the structure of the problem, applying a certain degree of consistency results in a globally consistent solution space. *Global consistency* in a constraint network ensures that a value can be found for each variable so that the entire constraint set is satisfied. Haroud (1995) has developed an algorithm guaranteeing *global consistency* for continuous constraint satisfaction problems (CCSP). In this algorithm, cubes approximate the region defined by each constraint in the tree-dimensional space. The algorithm calculates consistent solution spaces by combining these regions. Users can interactively restrain the feasible solution space and focus on regions of interest within. It is now possible for them to explore feasible space for preferable solutions.

Figure 1 visualizes the solution space for the constraints **C1**... **C4** of the spatial configuration example. In this Figure, the user decided to restrict the intial solution space to the region described by $6.5 \leq b_x \leq 7.2$. All the values dependent on $b_x$ ($a_x$ and $c_x$) are recalculated with respect to the new value of $b_x$.

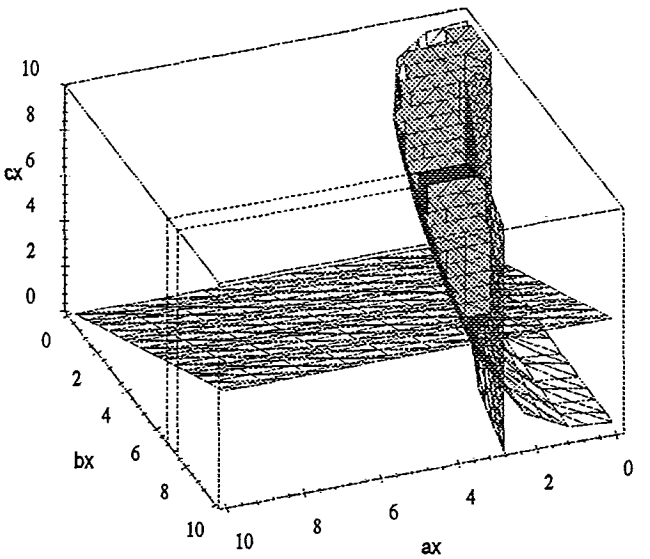



Figure 1: *The solution space corresponding to the constraints C1... C4.*

# 4 Incremental constraint satisfaction

In general design and configuration tasks, the problem space is often huge and interaction between components very complex. This is due to interactions between the variables - objects of that task - and their values defined by constraints. To reduce computational complexity, the task can be modeled as an incremental constraint satisfaction problem (ICSP). In an ICSP, the set of variables and constraints are not defined statically. Instead, so-called activity constraints extend a given set of initially active variables. An ICSP is defined by a set of variables $X$, a set of constraints $C$ and a set of initial conditions $W$. $W$ defines the set of variables that have to be part of every solution. $C$ consists of two types of constraints: *compatibility* and *activity constraints*, noted $CC$ respectively $AC$. Not all the variables need to be part of a solution: $X$ only defines the space of potentially active variables. The activity of variables and the constraints depending on them are reasoned on: the introduction of new variables and constraints depends on *activation conditions*. This dependency can be formulated as a so-called activity constraint[3] according to the definitions of Mittal (1990):

**AC:** $C_i(Y_1) \rightarrow \mathit{active} : Y_2$

[3] An activity constraint should not be mistaken for a rule, it defines a constraint on the activity of variables in the problem space

The activity constraint $AC$ activates the subset $Y2$ of variables if the activation condition (or precondition) $C(Y_1)$ is satisfied. The precondition $C_i$ defines a mathematical relation on $Y_1$. Compatibility constraints define the relations that must hold between active variables:

CC1: $C_i(Y_2)$

CC2: $C_i(Y_3) \rightarrow C_j(Y4) \qquad Y_3 \cap Y_4 = \emptyset$

If all the variables of $Y_2$ are active in **CC1**, $C_i$ has to be satisfied. The constraint $C_i$ is a mathematical (in)equality on the subset of variables $Y_2$ or, in the discrete case, a relation between variables where allowed tuples are enumerated. In **CC2**, it depends on the values of $Y_3$ as well as on the existence of $Y_3 \cap Y_4$ if $C_j$ is relevant or not.

Solving an ICSP completely implies finding all the solution spaces $S$ so that for each solution $s \in S$: $s$ satisfies all the constraints defined on a set of active variables in $X$ and no more variables can be activated.

For reasoning on the surface of the objects in the spatial configuration problem in paragraph 3, the objects are classified into different types such as rectangles, circles etc. and they are given additional dimensional properties. Depending on the type of an object, the variables width, height or radius are relevant and its surface will be calculated differently. Initially, $W$ is $\{A = rectangle, B = circle\}$, and the constraint set $C$ is defined by

AC1 $X = rectangle \rightarrow X.length$ AND $X.width$ AND $X.surface$

AC2 $X = circle \rightarrow X.radius$ AND $X.surface$

CC3 $X.surface = X.radius^2 * \pi$

CC4 $X.surface = X.length * X.width$

CC5 $R(X, Y) = (rectangle, circle)(circle, rectangle)$

When $A = rectangle$, the variables $A.surface$, $A.length$ and $A.width$ are generated and $A.surface$ will be calculated according to **CC4**. $B.Shape$ is restrained to a circle by **CC5** and its surface is calculated according to **CC3**.

In the following, we would like to detail how searching is performed in an ICSP. Search for solution spaces involves an *activate-propagate cycle*: From the given set of active variables, all activity constraints are checked in order to activate new variables. This step defines the new *problem space*, i.e. the space of currently active variables. In the propagate step, the compatibility constraints defined on active variables are checked for global consistency. Feasible partial *solution spaces*, i.e. regions in N-space defining value bounds for the variables, are found. At each cycle, values of currently active variables are either refined or an inconsistency is detected. Such a set of inconsistent constraints is discarded and the algorithm either backtracks to another solution space or to the next problem space not yet treated. It halts when no new problem spaces can be created, i.e. all the problem spaces have been searched and no new variables can be activated. The final solution spaces are those in the leaves of the problem space tree.

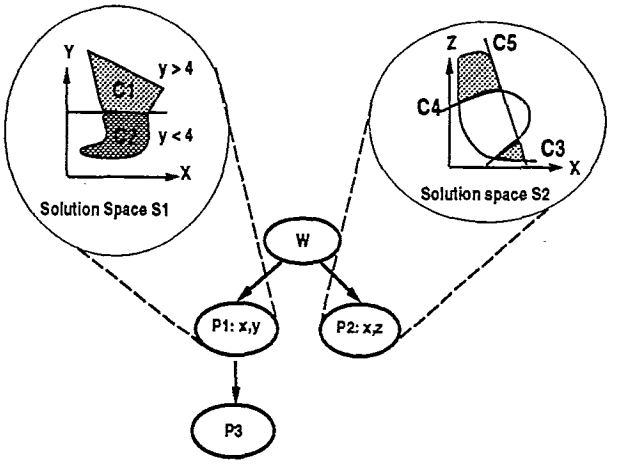



Figure 2: *A problem space tree. In problem space P1, the variables x and y are active; in P2, x and z. In the solution space S1, the set of the constraints explicitly depends on the value of y. In S2, the global consistency algorithm finds two separate feasible regions satisfying the same set of constraints with C3:* $y - 1/x \geq 0$, *C4:* $(x - y)^2 + 2x + 2y - 8 \geq 0$, *C5:* $y + 1.3x - 4 \leq 0$.

An incremental CSP can then be viewed as a sequence of static problems (Figure 2): $P_0, \ldots P_n$ with $P_0 =< X_0, C_0, D >$ and $P_i =< X_i, C_i, D >$ where $C_i = C_{i-1} \pm \{C_j\}$ with $\{C_j\} \subset C$, $X$ is the set of variables and $D$ the variables' domains. Activity constraints may split up one problem space into several each containing different sets of active variables ($P1$ and $P2$ in Figure 2). Constraints may split a solution space further by creating separate regions of consistent values. In S1, the relevant constraints explicitly depend on values of y. In S2, the intersection of $C3$, $C4$ and $C5$ create two separate feasible regions.

## 4.1 Formalisation of Configuration and Design Tasks

Objects to be modeled in configuration and design tasks are structured. Each object has a type, for example circle or rectangle and a set of properties. The set of types T is known in advance either from some catalogue of components or by the definition

of the task. The structure of an object can be modeled by an activity constraint (Haselboeck 1994).

$X = T_1 \rightarrow X.p_1 = D_1, \ldots, X.p_i = D_i$

If $X$ is an object of type $T1$, its instance expands into an object with attributes $p_1, \ldots, p_i$. Each of these attributes has a domain $D_i$ which can again be the set of component types. E. g. $p_3 = T$.

In order to be able to describe a configuration taks generically, it is important to express constraints on types of objects and not on the objects themselves. This renders them indpendent of specifically structured solutions.

$X = T_1 \wedge Y = T_2 \rightarrow C(X.p_i, Y.p_j)$

Here a constraint $C(X.p_i, Y.p_j)$ is stated between two attributes of objects of type $T_1$ and $T_2$ that will be applied to any pair of objects of the specified types without considering the structure of the specific configuration.

The structure of a specific configuration can be modeled by so-called ports, attributes that have as domain the component set $T$

$X = T_1 \rightarrow X.p_j = T$

$C(X.p_j, Y.p_i)$

Such ports allow for modeling part-of relationships also called component hierarchies (Mittal 1990, Haselboeck 1994).

## 4.2 Example in Bridge Configuration

We would like to show on a small example of bridge configuration how components of the configuration product can be added incrementally. Adding components leads to new design parameters and values that activate new constraints.

The aim in bridge configuration is to find bridge designs that satisfy design specifications as well as building codes and other requirements as described in Haroud and Boulanger (1995). Given the section of the valley in which the bridge has to be built, a set of initial conditions $W$ and a set of constraints $C$, we want to enumerate the solution spaces. In the following example, the designer already decided on a beam bridge type. The initial conditions $W$ are thus:

| Variable | Definition | Domain | Value |
|---|---|---|---|
| $L$ | valley length | real number | 200 |
| $B$ | bridge | {beam..} | beam |

and the constraint set $C$ is defined by:

**AC1** [4] $X = beam \rightarrow \left\{ \begin{array}{ll} X.ns & [1,20] \\ X.span: & \forall_{i=1}^{X.ns}\{X.span_i : [20, L]\} \\ X.maximal\ span: & [20, L] \end{array} \right\}$

[4] NS is an abbreviation of number of spans.

**AC2** $X = beam \wedge X.maximal\ span > 80 \rightarrow X.beam\ type : \{variable\ height, constant\ height\}$

**CC1** $X.beam\ type = variable\ height$

**CC2** [5] $X.ns \geq 4 \rightarrow \forall_{i=1}^{X.ns}\{X.span_i = \frac{L}{X.ns}\}$

**CC3** $X.ns < 4 \rightarrow \left\{ \begin{array}{l} X.span_1, X.span_{X.ns} < \frac{L}{X.ns} - 10 \pm 10\% \\ \forall_{i=2}^{X.ns-1}\{X.span_i > \frac{L}{X.ns} + 20 \pm 10\%\} \end{array} \right\}$

**CC4** $\sum_{i=1}^{X.ns} X.span_i = L$

**CC5** $X.maximal\ span = max_{i=1 \ldots X.ns} X.span_i$

The activity constraints in this example show how new components are added (**AC2**) and how the structure of the artifact is built (**AC1**). We simplified the structure by representing beams and piers by the unique notion of spans[6].

Starting with $B = beam$ and $L = 200$ the **AC1** is activated and the attributes $B.nb\ of\ spans = [1, 10]$, $B.span = ([20, L])$, $B.maximal\ span = [20, L]$ are added. The constraints **CC2**, **CC3**, **CC4** and **CC5** are propagated. They split the solution space into two spaces S1 and S2: S1 with $B.nb\ of\ spans < 4$ and S2 with $B.nb\ of\ spans \geq 4$. In S1, the constraints **CC3**, **CC4** and **CC5** are considered. In S2, **CC2**, **CC4** and **CC5** are propagated. In the second cycle, a new problem space P2 is created by adding the attribute $B.beam\ type$ according to the activity constraint **AC2**. Within P2, the constraint **CC1** is propagated adding a new component to the bridge.

This is an example of how configuration can be formalised as an incremental process of adding new components and their attributes and checking relevant constraints. Inconsistencies and splits in the solution space are detected during constraint propagation (solution spaces in P1). After each constraint propagation the user has the possibility of interactively restricting values. In S1, for example, the user could set $B.nb\ of\ spans$ to 3.

Our implementation is based on a forward chaining rule engine for activating constraints, a justification-based truth maintenance system (JTMS) and currently a low-level constraint satisfaction algorithm for checking consistency. During constraint propagation, new feasible regions inferred are justified by a JTMS-label linking design variables and constraints. Each constraint has a JTMS-label as well. Reasoning on the relevancy of a constraint can so be made explicit. After each cycle, the partial result is visualized in ICAD, an intelligent CAD system. It provides the user-interface with graphical representation and a product model of the bridge. New components are incrementally

[5] CC2 and the following constraints could be applied to any bridge type.

[6] A span is the distance between two adjacent piers.

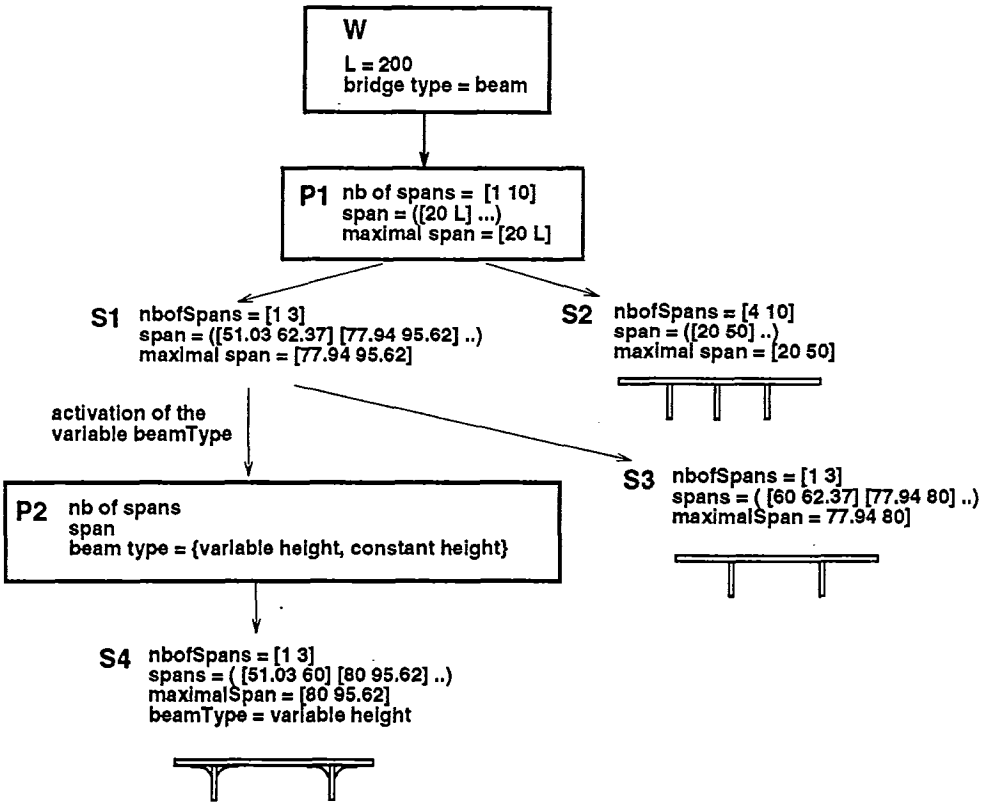



Figure 3: *A preliminary bridge design with two problem spaces and different solution spaces.*

added in the ICAD structure corresponding to our algorithm.

# 5 Conclusion

We have shown how configuration problems can be modeled in the framework of incremental constraint satisfaction problems with discrete and continuous variables. This is an extension of the purely discrete framework of Mittal (1990) and allows us to attack a broad range of configuration problems, which couldn't be solved with discrete variables only. Furthermore, we can guarantee maintenance and extendability of the system within this framework. We have based reasoning on algorithms of global consistency in the constraint network. Although this is computationally expensive, this gives the users the possibility to concentrate on the "solution spaces" of the problems $P_i$ after each propagation step. They are able to explore different partial solutions instead of searching in regions where no solutions can be found. The integration of algorithms for incremental constraint satisfaction into an intelligent CAD system, like ICAD, has shown to be very promising since the user can interact with the system by working with the graphical representations of the configuration objects instead of manipulating alpha-numeric symbols.

# Acknowledgments

Funding for this research was provided by the Swiss National Science Foundation. The authors would like to thank Boi Faltings, Ian Smith, Djamila Haroud, LIA, EPFL, and Sylvie Boulanger, ICOM, EPFL, for useful discussions.

# References

[1] Mittal, S. and Falkenhainer, B. (1990) Dynamic Constraint Satisfaction Problems. *Proc. of AAAI-90*, 25–32

[2] Faltings, B. and Weigel, R. (1994) Constraint-based knowledge representation for configuration systems. *Technical Report TR-94/95, LIA-DI, EPFL, Lausanne, CH*

[3] Haroud, D. (1995) Global Consistency for Continuous Constraints. *Phd thesis, LIA-DI, EPFL, Lausanne, CH*

[4] Haroud, D., Boulanger, S., Gelle, E. and Smith, I. (1995) Strategies for Conflict Management in Pre-

liminary Engineering Design. *AIEDAM Special Issue on Conflict Management in Design*, 313–323

[5] Stumptner, M., Haselböck, A. and Friedrich, G. (1994) COCOS, A Tool for Constraint-Based, Dynamic Configuration. *Proc. of the 10th IEEE Conference on AI Applications*, 373–380

[6] Mackworth, A.K. and Freuder, E.C. (1985) The Com-
plexity of some Polynomial Network-Consistency Algorithms for Constraint-Satisfaction Problems. *Artificial Intelligence*, **25**, 65-73

[7] Mohr, R. and Henderson, T (1986) Arc and Path Consistency Revisited. *Artificial Intelligence*, **28**, 225–233