Constraint Techniques for Collaborative Design

Claudio Lottaz*, Djamila Sam-Haroud*,
Boi Faltings* and Ian Smith**
*AI-Lab, Computer-Science Department,
**IMAC-ISS, Department of Civil Engineering,
Swiss Federal Institute of Technology,
CH-1015 Lausanne (Switzerland)

Abstract

*This paper presents **SpaceSolver**, a constraint satisfaction tool-box providing access to constraint satisfaction techniques on continuous variables through an intuitive, web-based user interface. Moreover, we describe possible applications of such a platform to collaborative design and conclude that Internet-based use of constraint satisfaction techniques has the potential of increasing productivity in several fields in engineering.*

Keywords: *Constraint satisfaction, collaborative design, WWW-interface*

1 Introduction

In complex design projects collaboration is unavoidable. Knowledge of several designers with different backgrounds is employed concurrently. However, collaboration also implies problems as illustrated in the following (simplified) example: An architect, an engineer, and a contractor collaborate on constructing a computer-building. Since such buildings have special requirements for ventilation, they agree on drilling holes into the steel beams of the steel-framed building in order to pass ventilation ducts. Now they must decide about the spacing e and the diameter d of these holes, as well as the distance between the support and the first hole x (see Figure 1).

During the construction of the building, the values for d, e, and x evolved as shown in Table 1. For maximum flexibility, the architect proposes big holes at small intervals. In contrast, the engineer would prefer small holes in order to maintain structural integrity. Now the needs for ventilation are not satisfied and this leads to negotiation that may iterate several times.

One solution at a time: When designers collaborate, the design-task is divided into sub-tasks, which

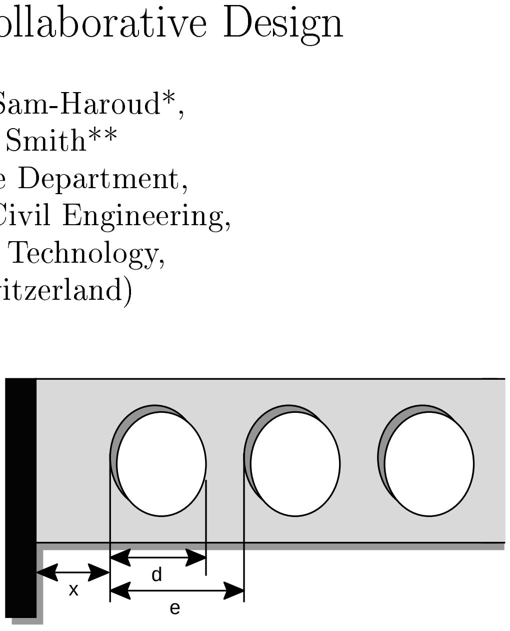


Figure 1: Beam with holes to pass ventilation ducts. d, e and x define the geometry.

	d	e	x
Architect	500mm	800mm	600mm
Engineer	**250mm**	**900mm**	**500mm**
Contractor	250mm	900mm	**700mm**
Architect	**300mm**	**800mm**	700mm
⋮	⋮	⋮	⋮

Table 1: When single values for parameters are fixed, artificial conflicts arise and negotiations may iterate. Variables d, e,and x are defiined in Figure 1.

are first treated separately, before the results to the sub-tasks are combined to form a solution to the whole task. Traditionally, designers provide one or few solutions to their sub-task. In order to do so, they may take decisions too early even though the task is influenced by decisions of other collaborators. Information related to alternatives and possible adaptation is thus lost. This may cause conflicts during the combination of sub-solutions, even though design-goals are not contradictory. The resolution of such *artificial conflicts* is performed during negotiation.

Using design spaces: An alternative to the above approach is to use *design spaces.* Instead of determin-

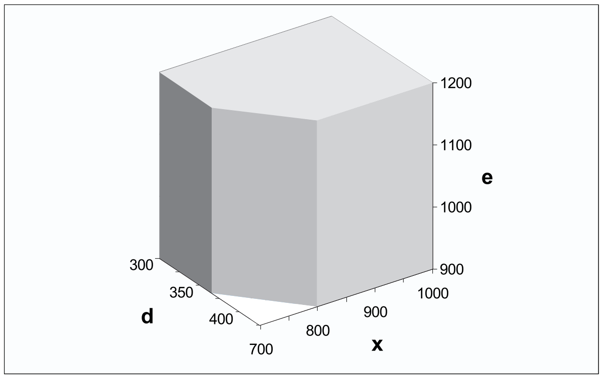


Figure 2: Solution spaces show the impact of decisions on other parameters. Choosing d very high in the space shown above requires high values for x.

ing single solutions to a sub-task, designers provide all acceptable solutions. Decisions are not taken if the needed information is not available, no arbitrary choices for parameters are made, and all alternative solutions can be used in subsequent negotiation. Solution spaces for sub-tasks can be combined to find a solution space for the whole task. Since all acceptable solutions to sub-tasks are considered, conflicts only arise when designers have incompatible design-goals. In this way less negotiation is necessary.

Simplifying the steel-framed building example mentioned above, the collaborators involved might impose the following constraints:

- Architect: $x < 1000$, $d > 300$, $600 < e < 1200$
- Engineer: $x > 2d$, $d < 400$, $e > 900$
- Contractor: $x > 700$, $e > d + 50$

Such a formalisation of the needs for every participating designer allows them to determine a solution space for the whole task as shown in Figure 2. Instead of negotiating about single solutions without knowing whether a valid solution to the problem exists, designers negotiate within feasible spaces in order to find the best solution. Table 2 shows a selection of the constraints which model the ventilation ducts example.

Constraint satisfaction techniques: In order to employ solution spaces as described, techniques to determine and combine solution spaces are needed, together with an adequate communication platform. Many problems in several domains such as civil engineering and mechanical design can be modelled as constraint satisfaction problems. A constraint satisfaction problem (CSP) is specified in terms of variables and their domains, as well as constraints on these variables. Variables represent properties of the artifact to be designed, each variable is assigned a domain, which contains the a priori possible values of its variable, and the constraints specify the restrictions which must hold for any valid solution of the problem. The solution space of a CSP which corresponds to a design task contains all feasible designs for the task at hand. When collaborating designers provide a CSPs corresponding to their sub-task, it is enough to combine the solution spaces of these CSPs, to determine the solution space for the whole design task.

Techniques to approximate solution spaces of CSPs have been under development for over two decades. An important class of CSPs are those defined for finite domains. Efficient methods for finding approximations of various accuracy [1, 9, 10] are available..

The task to find solution spaces for CSPs on continuous variables (CCSPs) is more difficult. However, tasks in design usually involve continuous parameters. Most successful systems which treat CCSPs concentrate on finding single solutions, possibly considering some optimisation criteria [15, 14]. Little previous research into finding solution spaces for continuous CSPs [3, 8, 12] is available.

Communication platform: In order to collaborate using solution spaces, designers need a common communication platform to exchange information on parameters and constraints. ***Space*Solver** is a platform. It interfaces with the Internet and provides access to constraint solving algorithms on continuous numerical CSPs, i.e., constraints are given as closed mathematical expressions over continuous variables.

This paper contains a brief description of ***Space*Solver**'s constraint satisfaction techniques in the next section. Section 3 illustrates its architecture and the WWW interface. Section 4 gives a summary of our first experiences and Section 5 provides conclusions.

2 Constraint satisfaction techniques

Resolution techniques for numerical constraint satisfaction problems encompass areas that include linear and non-linear programming, numerical analysis, hill-climbing and stochastic techniques. Most of these techniques attempt to compute a single solution, optimal according to some criterion. When the constraint satisfaction problem is embedded in a larger decision process, as it is often the case in collaborative applications, it becomes desirable to compute the space of *all* solutions. This section describes how ***Space*Solver**

N_1	$<$	$f_y\, A_1/G_r$	V_1	$<$	$(f_y\, A_{1w}) / \left(\sqrt{3}\, G_r\right)$
N_1	$=$	$1/\left(8\, a_N\, qL^2\right)$	V_2	$<$	$(f_y\, (e-d)\, t_w) / \left(\sqrt{3}\, G_r\right)$
q	$=$	$(1.3\, G_m + G_q\, Q_r)\, b_s$	$11\, t_w$	$<$	$a - 1/2\, t$
$a_N\, A_1$	$=$	$2\, ctb + t_w\, (a-t)\,(b-a)$	$32\, L\, E\, I_y$	$>$	$175\, L_{sc}\, b_s\, L^4$
$\frac{N_1}{A_1}$	$<$	$\frac{\sqrt{{f_y}^2 - 3\,(V_1/A_{1w})^2}}{G_r} - \frac{V_1\, d}{4\, Z_1}$	I_y	$=$	$ctb^2 + t_w\, (a-t)\,(b-a)^2/2$
A_1	$=$	$2\, ct + t_w\, (a-t)$	$2\, ct$	$<$	$t_w\, (a-t)$
A_{1w}	$=$	$(a - 1/2\, t)\, t_w$	c_n	$=$	$A_1/\,(2\, t_w)$
V_1	$=$	$(q\, L - q\,(2\, x + d))\,/4$	Z_1	$=$	$\frac{{c_n}^2 t_w}{2} + \frac{(a - c_n - t_w)^2}{2\, t_w}$
V_2	$=$	$(e\, q\,(x + d/2 - e + L))\,/a_N$		$+$	$(2\, a - 2\, c_n - t)\, ct$

Table 2: The constraints in the above table partially model a civil engineering problem concerning ventillation in a steel-framed computer-building

uses and implements *consistency techniques* to compute compact descriptions of the complete solution space of a numerical CSP.

2.1 Consistency techniques

From a general perspective, local consistency techniques can be seen as a means of computing approximate descriptions of the *complete* solution space for a CSP. These descriptions serve as simplified bases for further search or reasoning tasks. In practice, they are constructed by pruning *locally* inconsistent values from the domains of the variables.

One can distinguish different orders of consistency: 1-, 2- and in general k-consistency, depending on the degree, k, of locality taken into consideration. A k-consistency algorithm furnishes an approximation of the solution space where each sub-problem of k variables is guaranteed to be consistent. This amount to ensuring that each partial solution of k-1 variables can be extended consistently to a partial instantiation of k variables. In practice, this is achieved by assigning a label to each subset of k-1 variables which contains their legal value assignments.

Constraint satisfaction problems are generally represented using a graph (or hyper-graph) called the constraint network. In such a graph, nodes represent variables involved in the problem while arcs or hyper-arcs stand for constraints between variables.

Global consistency When the labeling constructed by a consistency algorithm contains only those values or value combination which occur in at least one solution, it is said to be *globally consistent.* A globally consistent labeling is a compact and conservative representation of all solutions admitted by a CSP. It is *sound* in the sense that the labeling never admits any value which does not lead to a solution. It is *conservative* in the sense that all solutions are represented in it.

Backtrack-free search Extracting a particular solution from a consistent labeling is an iteration of two steps in which values are assigned to variables sequentially. In the first step, an unassigned variable is selected and assigned a value within its label. In the second step, the labels of all remaining unassigned variables are updated so that they contain only values which are consistent with those already assigned. If the initial labeling is globally consistent and non-empty, every partial assignment of variables can be extended to a full solution. Consequently, the assignment procedure never requires backtracking. In general, a globally consistent labeling may require explicitly representing constraints for all variables in the problem (i.e. computing n-dimensional labels for a problem of size n), a task which has exponential time complexity in the worst case.

***Space*Solver's Consistency techniques** While in general computing a consistent labeling is NP-hard, recent results show that for special classes of problems, low orders of consistency are equivalent to global consistency. These results lead to polynomial time algorithms for computing *globally consistent* labelings and can be summarized as follows:

- 2-consistency (arc-consistency) is equivalent to global consistency when the constraint network is a tree [4],
- 3-consistency (path-consistency) is equivalent to global consistency when the CSP is convex[1] and binary (constraint involves at most two variables) [2, 13, 12],
- 3,2-relational-consistency (a variant of 5-consistency defined in [11, 12]) is equivalent to global consistency when the CSP is convex and ternary. Note that a numerical CSP, stated analytically, can always be transformed into a ternary one without loss of information using syntactic transformations.

These 3 algorithms constitute the core of ***Space*Solver**'s consistency module. They have been chosen for their ability to construct complete and sound descriptions of the entire solution space at low cost under well-identified conditions.

When the problem has no special simplifying properties, these algorithms will be used as pre-processing tools for reducing the complexity of the search space. They must then be interleaved with backtrack-search algorithms based on interval to generate consistent sub-regions of the solution space [7].

In implementing these algorithms, a key problem is representing and reasoning about continuous labels. In discrete domains, labels are represented simply as enumerations of values or value combinations. A label for a single continuous variable can be represented by a small collection of intervals. However, representing and manipulating labels of several variables is more involved since they may represent complex geometric shapes. The next section describes how this issue is addressed in ***Space*Solver**.

2.2 Constraint and label representation

***Space*Solver** uses a quadtree/octree representation of continuous constraints as developed in computer vision and proposed in [11, 12] for computing a labeling of any degree of consistency in continuous domains. The quad/oct-trees representation of constraints is based on the observation that in most practical applications

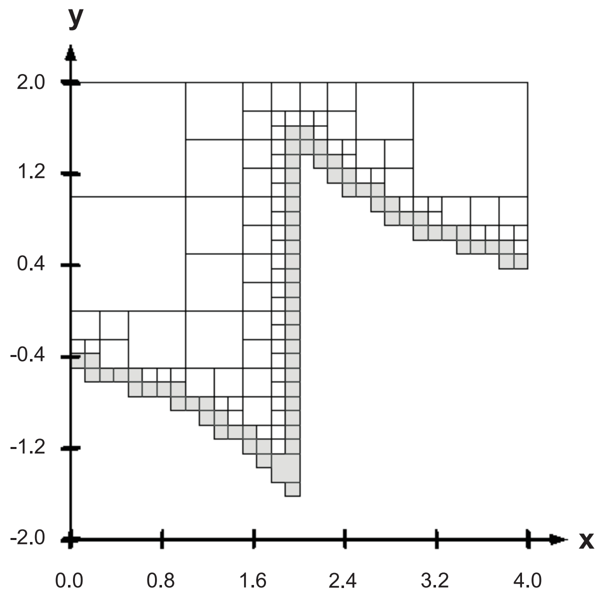


Figure 3: A continuous relation can be approximated by carrying out a hierarchical binary decomposition of its solution space into a 2^k-tree the nodes of which represent completely and partially legal regions. This figure shows $y \geq arctan(\frac{1}{x-1})$.

each variable takes its values in a bounded domain (bounded interval) and there exists a maximum precision with which results can be used.

Provided that these two assumptions hold, a relation can be approximated by carrying out a hierarchical binary decomposition of its solution space into quadtrees for binary relations and octrees for ternary ones etc ... (see Figure 2.2).

Numerical constraints generally arise as algebraic or transcendental equalities and inequalities involving several variables. Traditional CSP approaches process constraints directly in this form and therefore they encounter serious analytical difficulties related to intersecting surfaces and finding extrema. This partly explains why the most prominent advances in numerical constraint satisfaction are related to 2-consistency. [3, 5, 8, 14] [2]. As opposed to the implicit representation of numerical constraints, the 2^k-tree representation allows a logical rather than numerical handling of continuous solution spaces. It conveys a simple implementation even for higher degrees of consistency in continuous domains (see [11] for further details on the 2^k-tree construction and on the implementation of con-

[1]A CSP is convex when all its constraints are. A constraint is convex if the straight line between any two feasible points entirely lays within the feasible region.

[2]In this case, the goal is to compute unary labels. Constraints are therefore approximated by enclosing boxes (or set of boxes) and the combination of constraints then reduces to operations on intervals

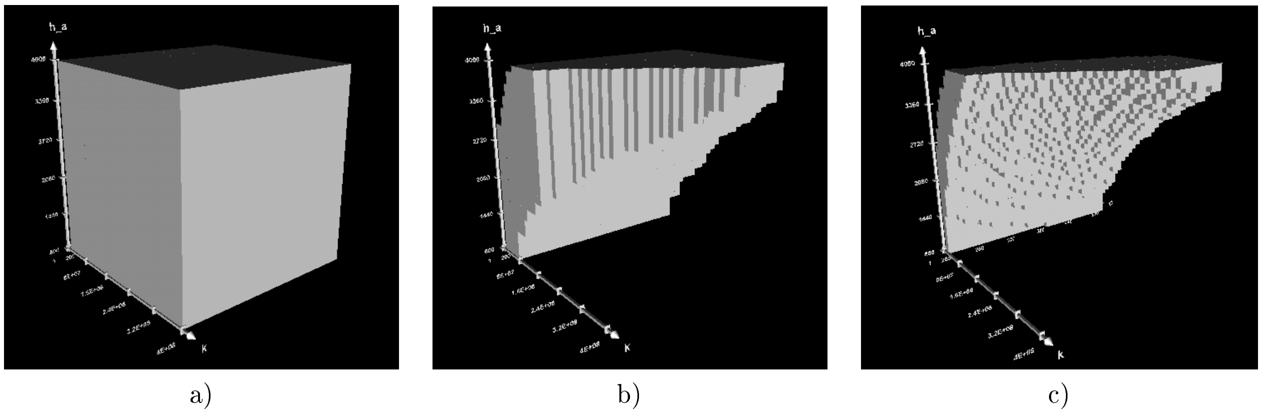


Figure 4: Consistency techniques can approximate the solution space of a CSP. This figure shows 3d-projections onto the same variables of a) the arc-, b) the path-, and c) the 3,2-relational consistent space of the same problem.

sistency algorithms). Figure 4 shows a ternary label computed by arc-, path- and 3,2-relational consistency respectively.

3 The *Space*Solver implementation

***Space*Solver** is an Internet-based software package that provides consistency techniques for CCSPs. It is available at the URL

http://liawww.epfl.ch/˜lottaz/SpaceSolver/

This section describes ***Space*Solver**'s architecture and modules.

3.1 System architecture

***Space*Solver** is conceived as an Internet application, in order to make consistency techniques on continuous variables available world-wide. The system architecture chosen places most modules on the server-side, thus avoiding client-side plug-ins or Java-Applets, thus making the solver more independent of the user's machine and configuration.

Figure 5 illustrates ***Space*Solver**'s system architecture. On the client side any JavaScript-enabled HTML-browser communicates through standard Internet protocols with the WWW-server where ***Space*Solver** is installed. Whenever a user is manipulating CSPs, asking for calculation or visualising constraints and solution spaces, the WWW-server calls the corresponding CGI-scripts to perform the demanded operation. These scripts are written in PERL and communicate through UNIX-sockets with the ***Space*Solver**-server. The ***Space*Solver**-server has the rights to perform activities such as writing files containing definitions of CSPs and executing ***Space*Solver**-modules. These modules include the *symbolic manipulator*, the *constraint converter*, the *constraint solver*, and the *VRML-Generator*.

Several scripts for Maple V make up the *Symbolic Manipulator* which is used to eliminate unnecessary auxiliary variables and bring the CSPs into ternary form. The *Constraint Converter* is used to convert constraints from symbolic form into an explicit spatial representation as illustrated in Section 2.2. The *Constraint Solver* module implements the consistency algorithms such as arc-, path- and 3,2-relational consistency. To achieve best performance, this module and the constraint converter are written in C++. Finally, the *VRML-Generator* dynamically generates 3D VRML models of constraints and solution spaces available in order to view constraints and solution-spaces on the Internet.

3.2 WWW interface

When users connect to the ***Space*Solver**-WWW-page, they are prompted for a user-name and a password. The purpose of this user authentication is not to prevent people from using ***Space*Solver**; but it allows us to keep data from different users apart and to control access to projects. Registration is free and access is given immediately.

When users select a project, i.e. a CSP, or create a new one, they are presented a page similar to Figure 6. On this page they can specify equalities and inequalities in ASCII-text using Maple V's syntax, as well as minima, maxima, default values and short de-

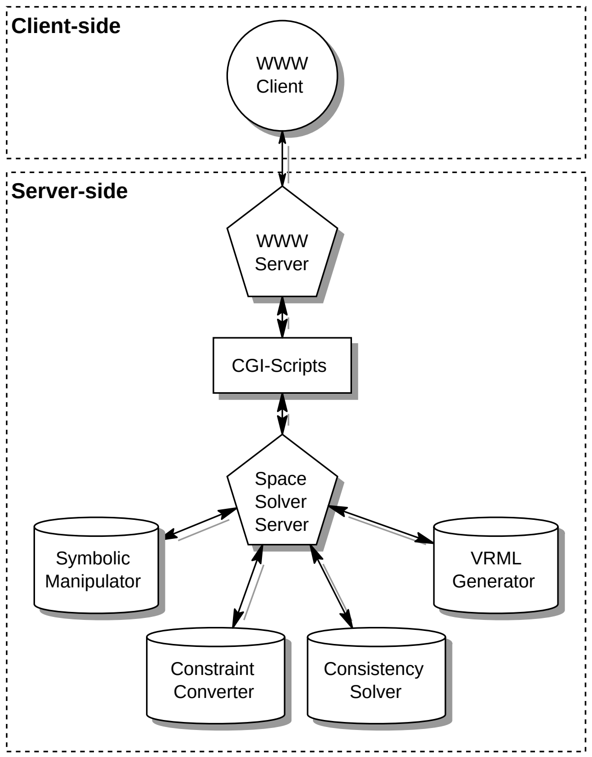


Figure 5: ***SpaceSolver****'s architecture avoids requiring users to install plug-ins or run java-applets by performing most tasks on the server-side.*

scription for each variable. The table that specifies variables, shown in the lower part of Figure 6, is generated automatically by detecting the variables found in the constraints.

For collaborative work ***Space*****Solver** provides the possibility, that several users can participate in the same project and specify their own constraints on partially shared variables. Users can be added and removed from a project by the creator of the project.

The visualisation of constraints and solution spaces helps understand CSPs and this supports decision making. Suppose for instance that an engineering task has three optimisation criteria. By visualising the projection of the solution space on these criteria, trade-offs can be illustrated and possible alternatives can be examined. In order to provide visualisation, VRML-scenes representing constraints and solution spaces are generated dynamically (see Figure 4). VRML is a 3D modelling language for the Internet. Several plug-ins to WWW-browser and stand-alone VRML-browsers allow Internet-users to examine and walk scenes specified as VRML.

3.3 Symbolic pre-treatment

Although ***Space*****Solver**'s consistency techniques are based on spatial representations of constraints and solution spaces, it performs symbolic pre-treatment for two reasons. First, the number of variables involved in the CSP is important for the efficiency of the consistency algorithms. Therefore avoidable auxiliary variables should be eliminated. Second, the consistency algorithms described in Section 2 treat CSPs in ternary form, i.e., each constraint must involve at most three variables. Thus ***Space*****Solver** substitutes constants and auxiliary variables defined by functional equations, and makes CSPs ternary.

Substituting constants: Equalities of the form $v = const$ are used to substitute v by $const$ in the whole CSP. This kind of equality may occur when designers commit to a value for a parameter, or to make the CSP-specification more readable.

Substituting auxiliary variables: Certain auxiliary variable are defined as an expression of only on one other variable x, i.e. $v = expr(x)$. Since this dependency attributes to each value of x exactly one value of v, v is substituted by $expr(x)$ in the whole CSP.

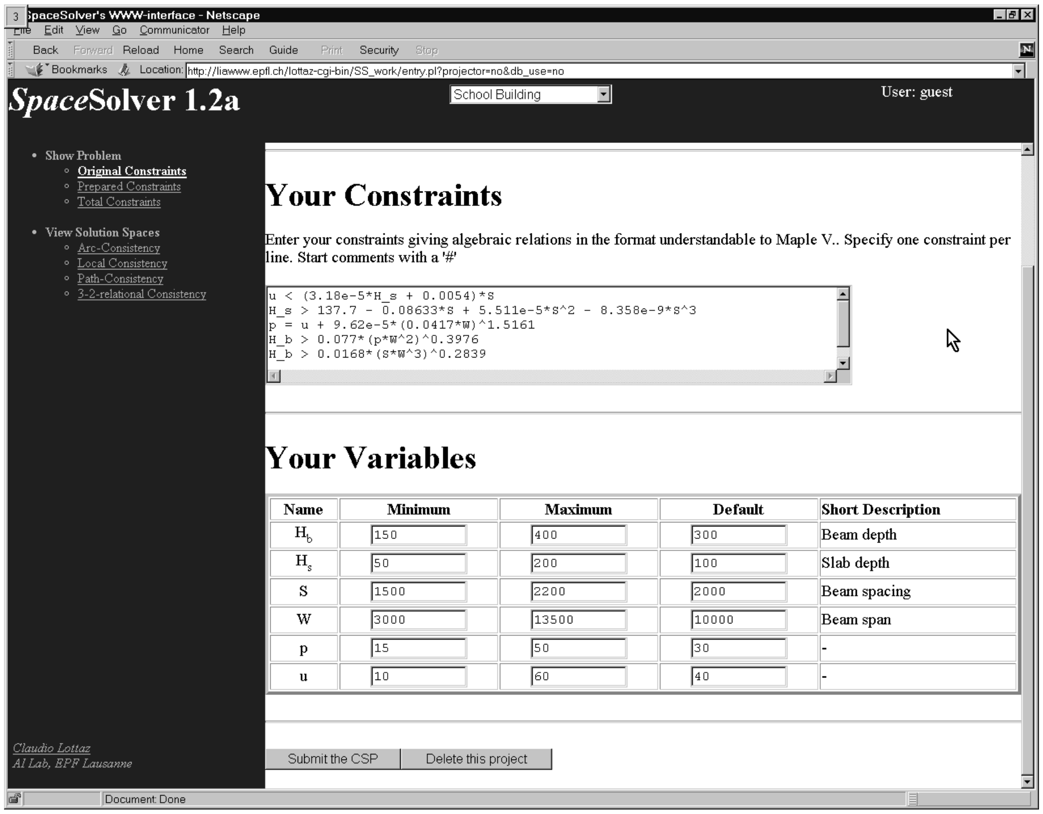
Making CSPs ternary: In general CSPs contain constraints which involve an arbitrary number of variables. For numeric CSPs, however, it is possible to introduce auxiliary variables such that all constraints are expressed using ternary constraints. The brute force method that involves introducing an auxiliary variable for each intermediate result generates many auxiliary variables. Therefore a more sophisticated algorithm introduces as few auxiliary variables as possible through reusing expressions as often as possible.

3.4 Consistency algorithms

Space**Solver** implements the consistency concepts introduced in Section 2. Two algorithms for arc-consistency are available: the standard version [11] and an improved one inspired by [3]. The original path- and 3,2-relational consistency algorithms have been substantially improved by avoiding the storage of large intermediate 2^k-trees.

4 First experiences

Space**Solver** has been used to model full-scale examples taken from the construction industry. In this field, it is possible to model most critical aspects related to building design and construction in terms of constraints. Moreover, it has been confirmed that ***Space-***



Your Constraints

Enter your constraints giving algebraic relations in the format understandable to Maple V.. Specify one constraint per line. Start comments with a '#'

```
u < (3.18e-5*H_s + 0.0054)*S
H_s > 137.7 - 0.08633*S + 5.511e-5*S^2 - 8.358e-9*S^3
p = u + 9.62e-5*(0.0417*W)^1.5161
H_b > 0.077*(p*W^2)^0.3976
H_b > 0.0168*(S*W^3)^0.2839
```

Your Variables

Name	Minimum	Maximum	Default	Short Description
H_b	150	400	300	Beam depth
H_s	50	200	100	Slab depth
S	1500	2200	2000	Beam spacing
W	3000	13500	10000	Beam span
p	15	50	30	-
u	10	60	40	-

*Figure 6: **SpaceSolver's** WWW-Interface allows users to specify CSPs in an intuitive way through the Internet*

Solver performs efficiently as a communication platform for assisting collaboration.

The arc-consistency algorithm detects contradictions within a few seconds even in large examples (with about 100 parameters). Furthermore, 3,2-relational consistency can be obtained overnight for small examples. Such a response time is acceptable in many engineering contexts since numerical modelling is traditionally performed this way.

Limitations and future work: The computation needed for 3,2-relational consistency, though polynomial, remains costly for large problems. We plan to address this question using tree-structured decomposition of CSPs as described in [6]. The search-module will provide a navigation tool to explore the solution space interactively. It is worth mentioning that the 2^k-tree approach implies limited precision. This restriction is acceptable for many engineering applications.

5 Conclusions

***Space*Solver** provides an intuitive interface to specify numeric constraint satisfaction problems on continuous variables. Its Internet-based approach makes it available throughout the world and it provides useful facilities for collaborative work. State-of-the-art consistency techniques to determine approximations of solution spaces are available and the 3D-visualisation of constraints and solution-spaces is well suited to facilitate the analysis of a given CSP and its results.

In the context of collaborative design and concurrent engineering, ***Space*Solver** shows through its data management capabilities that using constraints has the potential to improve Mrommunication between collaborating designers.

Acknowledgements

This work was performed within a project founded by the Swiss Priority Programme in Computer Science (SPP-IF). The authors would also like to thank Denis Clément and André Flückiger for their help in providing the civil engineering examples, the project partners at CAAD (ETH Zürich), namely Gerhard Schmitt, Rudi Stouffs, Bige Tunçer, Kuk-Hwan Mieusset and David Kurmann for helpful discussions.

References

[1] C. Bessière. Arc-Consistency and Arc-Consistency Again. *Artificial Intelligence*, 65(1):179–190, January 1994.

[2] R. Dechter, I. Meiri, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49(1-3):61–95, January 1991.

[3] Boi V. Faltings and Esther M. Gelle. Local Consistency for Ternary Numeric Constraints. In *IJCAI-97*, volume 1, pages 392–397, 1997.

[4] Eugene C. Freuder. A sufficient condition for backtrack-free search. *Jrnl. A.C.M.*, 29(1):24–32, January 1982.

[5] Esther M. Gelle. *On the generation of locally consistent solution spaces in mixed dynamic constraint problems.* PhD thesis, Swiss Federal Institute of Technology in Lausanne, Switzerland, 1998.

[6] Marc Gyssens, Peter G. Jeavons, and David A. Cohen. Decomposing Constraint Satisfaction Problems using Database Techniques. *Artificial Intelligence*, 66(1):57–89, 1994.

[7] Narendra G. Jussien and Olivier Lhomme. Dynamic domain splitting for numeric CSPs. In *ECAI-98*, 1998.

[8] Olivier Lhomme. Consistency Techniques for Numerical CSPs. In *IJCAI-93*, volume 1, pages 232–238, 1993.

[9] Alan K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[10] R. Mohr and T. C. Henderson. Arc and Path Consistency Revisited. *Artificial Intelligence*, 28:225–233, 1986.

[11] Djamila Sam-Haroud. *Constraint consistency techniques for continuous domains.* PhD thesis, Swiss Federal Institute of Technology in Lausanne, Thesis No. 1423, Switzerland, 1995.

[12] Djamila Sam-Haroud and Boi Faltings. Consistency Techniques for Continuous Constraints. *Constraints*, 1(1&2):85–118, September 1996.

[13] Peter van Beek. On the Minimality and Decomposability of Constraint Networks. In William Swartout, editor, *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 447–452, San Jose, CA, July 1992. MIT Press.

[14] Pascal van Hentenryck, Laurent Michel, and Frédéric Benhamou. `Newton`: Constraint programming over nonlinear constraints. *Science of Computer Programming*, 30(1–2):83–118, January 1998.

[15] Pascal van Hentenryck, Laurent Michel, and Yves Deville. *Numerica - A Modelling Language for Global Optimization.* MIT Press, 1997.