J. S. Gero (ed.), *Artificial Intelligence in Design'00*, 311-331.

STRUCTURAL ENGINEERING DESIGN SUPPORT BY CONSTRAINT SATISFACTION

ESTHER M. GELLE, BOI V. FALTINGS
Artificial Intelligence Laboratory (LIA)
EPFL – Swiss Federal Institute of Technology

AND

IAN F.C. SMITH
Structural Engineering and Mechanics (ISS-IMAC)
EPFL – Swiss Federal Institute of Technology

Abstract. Design tasks in structural engineering have always involved the use of constraints to formulate design requirements. Most existing algorithms for constraint satisfaction require input consisting of binary constraints on variables that have discrete values. Such restrictions limit their use in structural engineering since typical structural design tasks involve discrete and numerical variables. This paper provides an approach for decision support through approximating solution spaces of such constraint problems by local consistency. The approach is demonstrated for the selection of appropriate wind bracing for single story steel-framed buildings involving more than hundred variables. Finally, extension to conditional constraint satisfaction using different combinations of activation conditions is straightforward.

1. Introduction

Many important structural-engineering design requirements are formulated using constraints. For example, a structural design task is driven by functional criteria such as structural safety and serviceability or geometry. These criteria are usually represented in terms of inequality constraints. Examples of inequality are: 1) the stresses due to loads must be *less than* the resistance provided by the structural system and ii) the available clearance beneath a bridge must be *greater than* the minimum required clearance.

Constraints are therefore the natural language of structural engineering designers. They define spaces where feasible solutions may exist. Currently, single point solutions are employed in engineering because traditional media,

such as engineering drawings, require fixed-value assignments for variables. Nevertheless, the representation of possible solutions as solution spaces enhances the exploration of solution alternatives. supports a more effective negotiation between engineers since artificial conflicts emerging from partial single-point solutions are avoided (Lottaz et al., 1999).

Research into constraint satisfaction (CSP) is resulting in techniques for representing spaces of all solutions. Consistency techniques are employed during search to remove inconsistent values and to transform CSPs in order to make them simpler to solve (Waltz, 1975; Mackworth, 1977; Montanari, 1974; Freuder, 1982; Dechter, 1990; VanBeek, 1995). Other work has concentrated on applying consistency techniques during search in order to prune parts of the search space early (Haralick et al., 1980; McGregor, 1979; Bessière et al., 1996).

Search techniques that were developed for discrete problems have only recently been successfully applied to numeric domains (numeric CSPs). For a numeric CSP, value combinations are not enumerable and therefore, interval analysis is employed to achieve partial degrees of consistency (Benhamou et al., 1994; van Hentenryck et al., 1995; Lhomme, 1993; Hyvönen, 1992; Davis, 1987). Recently, Sam-Haroud et al. (1996) have extended the results of van Beek et al. (1995) to numeric CSPs and presented convexity conditions under which a numeric CSP becomes backtrack free.

Some relationships in a typical engineering task are not relevant until other decisions are taken. An extension of the original static CSP is proposed in (Mittal et al., 1990). Their definition of conditional CSP allows the activation of new variables during problem solving. However, conditional CSPs are only defined for discrete variables and constraints and the proposed resolution algorithm relies on an explicit enumeration of values.

Furthermore, most existing approaches for solving mixed CSPs, which involve discrete and numeric variables, are based on a cooperation between constraint solvers (Tinelli et al., 1996). Such an approach does not make use of mixed constraints to prune the solution space and is thus not able to solve effectively typical structural engineering tasks.

This paper addresses the problem of solving engineering tasks represented as CSPs. The advantage of using CSPs in this field is that consistency techniques provide a means of representing an approximation of solution spaces instead of single point solutions. This provides the engineer with a better basis for decision making and even improves the efficiency of further search and optimisation algorithms since it results in those parts of the search space which are likely to contain solutions. In this context, we propose the following new methods:

The approximation of solution spaces is achieved by a new local consistency method for discrete and numeric variables providing goods results in times of pruning and execution time.(Faltings, 1994; Gelle, 1998).

A new search method embedding local consistency for discrete and numeric variables is proposed. It integrates the local consistency methods for discrete and numeric variables into the search process and also makes use of the mixed constraints to prune the search space.

Finally, a new algorithm is introduced for solving conditional constraint problems over numeric and discrete variables. From the conditional problem formulation a tree of static CSPs is generated. Our local consistency methods are applied in order to eliminate inconsistent branches in the tree.

The paper is organised as follows. First, constraint satisfaction problems are introduced. After a review of the existing techniques, a new local consistency and an improved search method are presented. Their performance is demonstrated using a design task in structural engineering. Finally, an extension of the example to conditional CSPs is described.

2. Constraint Satisfaction Problems (CSPs)

A CSP $<V,C,D>$ is defined by the variables of interest V, each variable $X_i \in V$ with a domain $D_i \in D$ of possible values, and a set of constraints C defining allowed value combinations over subsets of the variables. The goal is to find either one or all *solutions* to the CSP. A solution is a consistent assignment of values to all variables such that all constraints are satisfied. A constraint can be represented either extensionally by specifying all valid value combinations or intensionally by a logical predicate. We distinguish between three different types of constraints:

A *discrete* constraint $C^{Dis}_{X1..,Xk}$ defines a set of allowed value tuples $(x_{1j},..,x_{kj})$ either extensionally or by a predicate such that $(x_{1j},..,x_{kj}) \in D_1 x .. x D_k$

A *numeric* constraint $C^{Num}_{X1..,Xk}$ is a relation $E \cdot 0$ where $\cdot \in \{=,\neq,\leq\}$ and E is an expression built from constants, variables and operations $\{+,-,/,*,exp,..\}$ over the reals.

A *mixed* constraint is defined over a set of discrete V^{Dis} and a set of numeric variables V^{Con} where the constraint is a relation defined on $V^{Dis} \cup V^{Con}$.

A CSP defined on numeric and discrete variables is called thus a mixed CSP.

In general, solving a CSP, that is deciding if there is at least one solution, is NP-complete since, in the worst case, all combinations of values for the variables have to be tested. Lower levels of complexity are possible for specific cases (van Beek et al., 1995; Freuder, 1982). An example of a binary discrete CSP is given in Figure 1. With *backtrack search*, a CSPs is solved by instantiating variables and testing partial instantiations against relevant constraints (*consistency checks)*. The search space of a CSP can be represented as a tree, in which each node corresponds to a partial instantiation of the variables and an edge between two nodes represents the choice of a value for the next variable (Figure 1). If a node is reached where

the tuple of values becomes inconsistent, no future choices will lead to a solution and therefore, the node is pruned. In Figure 1, node *(rrr)* is pruned and its subbranch neglected because it does not satisfy the constraint between variables *Hat* and *Jeans*. There is only one solution to this problem: *{Hat = r, Shirt = g, Jeans = b, Shoes = r}*. Note also that inconsistency is rediscovered, e.g. inconsistency between *Hat = r* and *Jeans = r* in Figure 1, a phenomenon called *thrashing* (Mackworth, 1977).

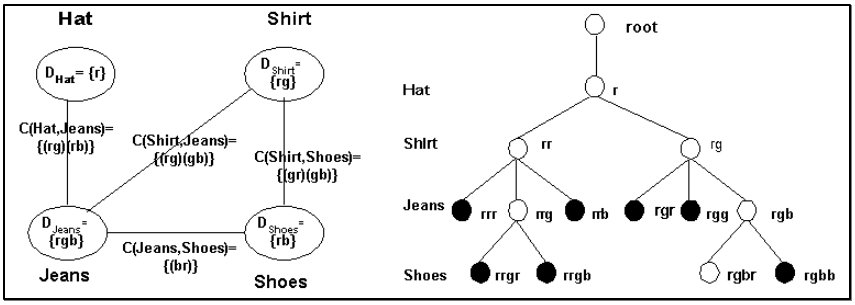

Figure 1. Left side: each node corresponds to a variable with its domain indicated in brackets and edges between the nodes represent constraints between pairs of variables with the allowed value tuples. Right side: search tree solving a small CSP by backtracking. The black nodes have been pruned from the search space.

2.1 LOCAL CONSISTENCY

Local consistency techniques transform the given CSP by removing inconsistent value combinations from subsets of variables and result in combinations of variable values called *labels*. Initially the labels are set to the value domains of the variables. This transformation step is complete, i.e. it does not loose any of the solutions. The set of refined labels corresponds to an approximation of the solution space for the given problem. In engineering tasks, it may be important to know at least an approximation of the solution space:

- In the traditional approach, engineering tasks are solved using some optimisation method which provides single point results. This may lead to artificial conflicts when different engineers solve subtasks of the problem and then try to integrate their partial solutions to a global solution (Lottaz et al., 1999).
- The result of local consistency can help engineers to visualise solution spaces and support them during decision making (Lottaz et al., 1999).
- Although locally consistent solution spaces may still contain infeasible value combinations, they provide useful input into further search or optimisation algorithms since they have already narrowed down the search space.
- Local consistency with search is often the only alternative for solvin g engineering problems as it can be very difficult to provide a good objective function for this type of problem (Gelle, 1998).

In this paper, we are interested in the consistency between pairs of variable values, also termed local consistency or arc-consistency (Mackworth, 1977). Local consistency on a CSP is achieved by examining all ordered pairs of variables, making the label of one variable in the pair consistent with the other, and adding the variable pairs that are dependent on the changed label to the queue. Changes are so propagated through the CSP. When no more changes occur, the set of pruned variable labels is returned. A single step of making the label of a variable X_j consistent with the label of another variable X_i is denoted by **refine(**$X_i X_j, C_{Xi,Xj,..}$**)**. It checks if it is possible to instantiate a second variable such that all constraints between both variables are satisfied. If not, the value is removed from the variable. Local consistency for discrete domains can be achieved in polynomial time in the size of the variable domains (Mackworth, 1977).

Global consistency, on the other hand, results in labels in which each value combination is part of a solution and is thus a much stronger condition. Solution spaces calculated in this way provide greater engineering support. However, achieving global consistency is exponential in the general case.

In Figure 1, the labels L_{Hat} and L_{Jeans} are initially set to D_{Hat} and D_{Jeans} respectively. One step of refine, making L_{Jeans} locally consistent with label L_{Hat} consists of removing the value r from L_{Jeans} because this value does not participate in any tuple of the constraint $C_{Hat,Jeans}$. In the example, the unique solution is obtained simply by enforcing local consistency.

2.2 SEARCH TECHNIQUES FOR CSPS

Backtrack search is improved through removing inconsistent value combinations either prior to or during search. During backtrack search, each instantiation is propagated further to not-yet-assigned variables using a refine step. The amount of local consistency performed at each node determines the type of search algorithm. *Forward checking* (Haralick et al., 1980), for example, applies a refine step to all the neighbours of an instantiated variable, that is, to all those future variables which are connected to the current variable by a constraint. Another algorithm, called *maintaining arc-consistency* (Sabin et al., 1994; Bessière et al., 1996) applies local consistency to the entire CSP after each domain reduction (Figure 2).

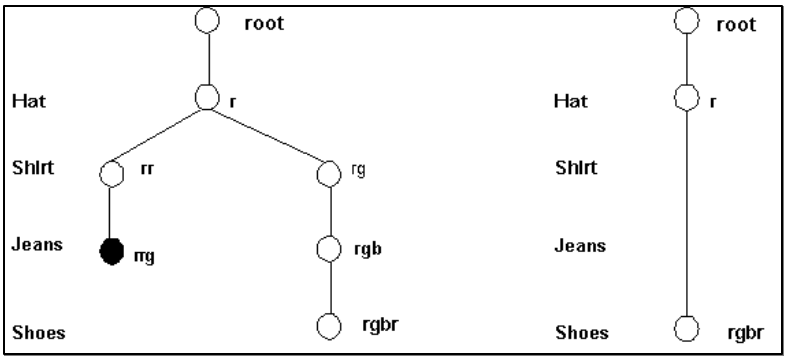

Figure 2. Search tree of the CSP in Figure 1 solved by a) forward checking and b) maintaining arc-consistency

While are a great deal of research results related to solving discrete CSPs, reliable numeric CSP resolution has proven to be more difficult. This is partly due to the infinitely many possible values the variables may take in their domains. Search methods using interval analysis (Moore, 1979) are often used for nonlinear optimisation and for resolution of equation systems in order to produce single solutions. In a typical branch and bound approach, the constraint region is subdivided into a finite number of cubic subregions (sets of intervals) that are then tested for the optimum using interval analysis. If a test fails on a cubic region, it is guaranteed not to contain a solution and can be discarded. In Figure 3a, branch and bound with interval analysis is applied to find the maximum in Y of the shaded region. Boxes 1,2 and 3 can be discarded by interval analysis. Further splits in box 4 are necessary to find a reasonable approximation of the optimum.

Van Hentenryck et al. 1995) have developed a prototype called Newton and its successor, Numerica (1997), enhancing interval methods with consistency techniques. The system consists of a branch and bound algorithm applying box-consistency to improve the assignment of variable labels. Hyvönen (1992) uses an interval analysis approach called tolerance propagation. This algorithm refines solution sets in a top-down manner in order to create a lattice of solution sets. Lhomme (1993) uses bound-consistency in order to improve the assignment of variable labels.

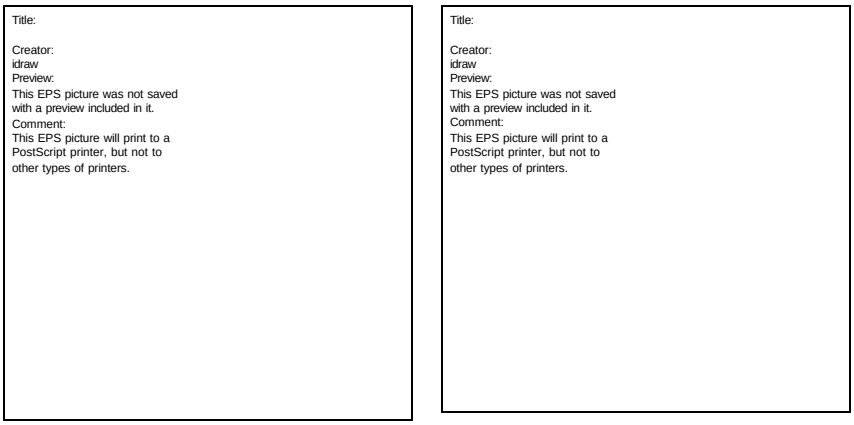

Figure 3. Search in numeric CSPs: a) interval method b) backtrack search instantiating the mid-value of an interval

3. Improved Algorithms for Local Consistency Over Numeric and Mixed Constraints

In order to simplify the discussion of local consistency over numeric constraints, we consider only binary numeric constraints in the following example, i.e. numeric constraints defined over two variables. Such a constraint defines a feasible two-dimensional region. It can be observed that **refine(**$X_iX_j,C_{Xi,Xj,..}$**)** computes the projection onto X_j of the region defined by all constraints between X_iX_j and given the label of X_i (Faltings, 1994).

Most existing refine functions for numeric constraints approximate the projection of this region onto the variable axes. They compute an approximation of the projection for each constraint by accounting for intersections of interval bounds with the constraints (2B-consistency, Lhomme (1993)) or by applying interval analysis (box-consistency, Benhamou et al. (1994), van Hentenryck et al. (1995). Consider the projection onto the Y-axis of the feasible region defined by the constraints in Figure 4 such that all values of X are in I_X. Any algorithm propagating such constraints individually will result in the single interval I_Y for the label of Y.

The true projection of a feasible constraint region is computed using local extrema of the region (Faltings, 1994). A refine function combines all constraints defined on the same pair of variables simultaneously in a *total constraint*. The local extrema are defined as local extrema of individual constraints, intersections between constraints and intersections of an interval boundary with a constraint. When this algorithm is applied to the example in Figure 4, the true projection, consisting of two intervals I_1 and I_2, is obtained.

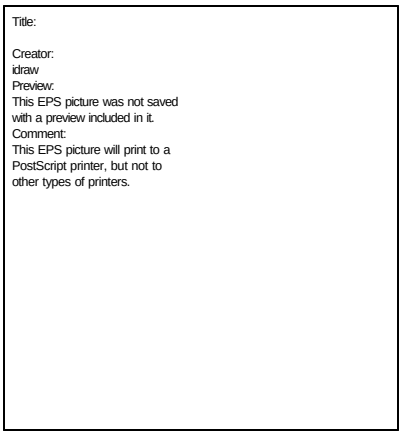

Figure 4. Propagating constraints individually results in the single interval IY whereas the exact projection of the feasible region onto the Y-axis are the intervals I_1 and I_2

The crosses in Figure 4 indicate intersections between constraints; these points were calculated once and then used to compute the projection.

For constraints with higher arity, this approach can directly be applied as described above during search when all but two variables of the constraint network have been instantiated. Another possibility is to use an extension of the local consistency algorithm to ternary numeric constraints described in Faltings et al. (1997) and Gelle (1998). The generalised algorithm is valid for topologically simply connected feasible regions[1]. This condition excludes three-dimensional regions with channels, i.e. holes that connect to the surface of a region. Since any constraint with an arbitrary number of variables can be decomposed into an equivalent network of ternary constraints, local consistency for a wide range of n-ary constraint networks can be computed.

In addition, refine functions for mixed constraints involving both discrete and numeric variables have to be provided. Different types of mixed constraints are given in TABLE 1.

Discrete constraints with interval values can be treated by the discrete refine function if a discretisation of all numeric variables involved is also defined. A discretisation of a numeric domain involves defining landmarks.

The open intervals between landmarks are taken as discrete values. For variable *beam_depth* in the constraint given in TABLE 1, the intervals *[0.1,1]* and *[1,15]* are two discrete values covering the variable domain *[0.1,15]*. The constraint can thus be represented internally as a discrete constraint and treated by a discrete refine function. Before activating refine functions, *transformation functions* convert numeric variable labels into

[1] A region is simply connected if any closed curve inside the region can be contracted to a single point.

discretised sets of values. After the call to refine, resulting labels are translated back into sets of interval values in order to be available for other numeric constraints (Benhamou, 1996). In general, transformation steps might be necessary before and after a call to the refine function in order to combine results from mixed constraints.

TABLE 1. Different types of mixed constraints

Type 1: Discrete constraints with interval values		
if beam_type = rolled **then** beam_depth ∈ [0.1,1] **if** beam_type = plate_girder **then** beam_depth ∈ [1,15]		the expected depth of the beam according to its type
Type 2: Numeric constraints that use discretisation operators or that contain integer variables		
*B = (Nft - 1)*Eft*		width of the structure as a function of the number of joists and the joist spacing
Type 3: Mixed constraints that link discrete value tuples or interval values to numeric constraints such as discontinuous functions		
$Q_n = 320*(1 + (H_0/350)^2)$ $Q_n = 900$	if $Q_n \geq 900$ otherwise	snow load Q_n on a flat roof as a function of the altitude H_0 at which a structure is built in Switzerland in N/m^2. The minimum value for Q_n is 900 N/m^2

Numeric constraints with discretisation operators are approximated by a set of numeric constraints according to Table 2. The numeric refine function treats the constraint as all other numeric constraints. Additionally, if a new interval value *[a,b]* with *a,b* $\hat{I}\hat{A}$ is derived for the variable N_g in TABLE 1, it is rounded to the next integer interval applying the transformation function *f([a,b]) = [éaù , ëbû]*.

Mixed constraints associating discrete values with numeric constraints are constraints in which discrete tuples are related to numeric constraints, as for example in the piecewise defined function on Q_n in TABLE 1. These constraints are propagated during search. The strategy is to enumerate first discrete values of variables in such constraints in order to define those subspaces where the numeric part of the constraint is valid.

4. A Generic Search Algorithm with Refine Functions for Mixed Constraints

A generic way to solve a mixed CSP is to exploit similarities between search in discrete and numeric CSPs. In general, numeric CSPs are solved using

interval methods. These methods create a tree of intervals through bisection as shown in Figure 3 and then intervals are pruned using interval analysis. Discrete CSPs, on the other hand, are solved by enumerating combinations of values for variables. We combine these two methods and propose a systematic enumeration of values as follows:

Bisection generates a tree of intervals from which a tree of midpoint values is obtained.

The definition of midpoint value depends on the domain representation: discrete variable labels can be represented by a set of integer intervals, e.g. the label *{a,b,d}* of a discrete variable with domain *{a,b,c,d}*, ordered by *a:1*, *b:2*,*c:3*, *d:4*, is represented by the list of intervals *[1,2],[4,4]* The midpoint value for a discrete interval *[a,b]*, a,b ∈ *N* is then computed by the formula *ë(a+b)/2û*. Continuous variable labels are represented by a set of real intervals. The midpoint value is thus the midpoint of one of the intervals *[c,d]* with c,d ∈ Â given by the formula *(c+d)/2*.

The tree of midpoint values is searched in a depth-first manner starting, for example, with the first interval in a variable label. However, the locally consistent intervals are kept for each variable. If a consistency check fails, the algorithm can directly continue with the next interval in a label.

TABLE 2. A list of operators and their approximations; r is a real, i, p, and q are integers

Operator	**Name**	**Approximations**
i = érù	ceiling	*r £ i < r+1*
i = ërû	floor	*r-1 < i £ r*
i = round(r)	round	*r-1/2 £ i < r+1/2*
i = trunc(r)	trunc	*r-1 < i £ r if r ³ 0* *r £ i < r+1 if r<0*
i = mod(p,q)	mod	*r/q = div(r,q)+i*
i = div(p,q)	div	*i = trunc(r/q)*

The algorithm split instantiates discrete and numeric variables successively to a single value m, the midpoint, of the current interval I of their label L and checks the obtained assignments against the constraints in treat-value. When a consistency check fails, the algorithm first backtracks to the interval to the left of the failed value [left(I),m) and then to the interval to the right (m,right(I)] and searches them recursively. Domain splitting continues until when a consistency check fails, the algorithm first backtracks to the interval to the left of the failed value [left(I),m) and then to the interval to the right (m,right(I)] and searches them recursively. Domain splitting continues until a given distance w between adjacent values is reached. w is 1 for discrete domains and is chosen for numeric domains such that a reasonable number of values in the label are checked. The generic search algorithm is enhanced by consistency techniques that propagate instantiated values to the

future variables. The function check in Algorithm 1 applies forward checking to prune neighbour labels using the refine functions defined in this paper. It takes the set of labels L, the constraint set C and the already obtained assignments S as input and returns the changed labels and an ok-status indicating if an inconsistency has been found in the assigned variables and the constraints or not. If the problem is still consistent (ok is true), the algorithm continues calling recursive-search on the next variable, otherwise the last assignment added is removed from the solutions and the labels, which may have been changed by check, are reset to the state before the last assignment (unwind-labels). Mixed constraints of type 3 are treated by adding them to the constraint set whenever the new variable assigned is part of that (add-mixed-constraints). They are removed again when the algorithm backtracks on this variable. Additionally, this algorithm integrates dynamic variable ordering to improve search (function reorder-variables, Haralick et al., 1980; Bessière et al., 1996). Before the next variable is instantiated, variables are reordered taking into account the following strategies: i) variables occurring in discontinuous mixed constraints should be enumerated first since they add numeric constraints to the problem, ii) other discrete variables are enumerated subsequently so constraining numeric variables through mixed constraints.

In the example presented in the following section, we choose to enumerate all values for discrete variables and to find only one consistent value for numeric variables. The reason is that an enumeration of values in the numeric labels results in very similar solutions at a distance w of the previous solution.

An example of a simple backtrack algorithm with value instantiation is presented in Figure 3b. The dot indicates one possible midpoint solution. Here a solution is found after three levels of splitting, i.e. in the worst case the centers of sixteen cubes have to be tested.

In numeric domains, pruning the intervals from inconsistent values using local consistency is important since interval size and number of splitting iterations can be reduced drastically. Local consistency algorithms for numeric domains, such as the one described here have improved computation times, especially if many variables are already instantiated since, in that case, the search space becomes highly constrained.

Algorithm 1. Search algorithm for a mixed CSP with interval splitting on backtracking

```
procedure recursive-search (V,L,C,width,S)
  if all variables of V are assigned in S then
    return on first solution: S
```

```
    else
        V ← reorder-variables(V)
        X ← first variable in V unassigned in S
        I_X ← next interval from L_X ∈ L
        split(I_X,X,V,L,C,width,S)
    fi

procedure split(I_X,X,V,L,C,width,S)
    m ← midpoint(I_X)
    treat-value(m,X,V,L,C,S,width)
    if right(I) - left(I) > width then
      split(X,[left(I_X),m),V,L,C,width)
      split(X,(m,right(I_X)],V,L,C,width)
    fi

procedure treat-value(m,X,V,L,C,S,width)
    oldLabels ← L
    L_X ← ¬ m , L_X Î L
    S ← add X = m to S
    C ← add-mixed-constraints( C,X=m )
    newLabels ← check (L,C,S)
    if newLabels ≠ ∅ then
     recursive -search(V,newLabels,C,width,S)
    fi
    S ← remove X=m from S
    C ← remove-mixed-constraints (C,X=m)
    unwind-labels(newLabels, oldLabels)
```

Our approach is very generic in that the search engine is the same for discrete and numeric variables and that special *refine functions* can be associated to each constraint type (discrete, numeric, and mixed) to prune variable labels.

5. A Full-Scale Example

A simple example on the design of a single story steel building is presented in Figure 5. Non-rigid connections (pin connections) are envisaged. The building has to resist to external loads such as wind and snow (Swiss Code SIA 160) as well as internal loads caused by an overhead crane. In order to resist horizontal loads, a longitudinal and transversal bracing system is necessary (crosses in Figure 5). Constraints from the Swiss Code (SIA 161) that have to be respected are: bending of joist (*Ft*) and girder (*Fl* and *Ff*), biaxial bending of girts, buckling of columns (*Cl* and *Cf*) and axial forces in diagonals (*Dlt*, *Dll*, *Dtt*, *Dtl*) of the bracing systems. Characteristics of steel

sections available in Switzerland are introduced in form of tables. Geometric constraints on volume, surface, width, length, height, width-length ratio, spacing of the elements (*E*, *Eft*, *Ecf*, *Efl*), frontal (*Lpf*) and lateral door (*Lpl*) width are also considered. A small set of more than 100 variables and over 50 constraints applied in the example are shown in Table 3.

Typical input parameters are snow and wind load, the altitude at which the building is situated and the minimal surface it occupies (user constraints in Table 4). Applying local consistency methods at this point results in reduced domains for the numeric variables. However, there is no effect on the choice of the cross sections since there are too many degrees of freedom. When additional constraints are imposed on variables (building dimensions and spacing between elements in the user constraints of Table 4), their effect on other variables is observed through reduced solution for the choice of cross sections for the girts, joists, columns, and frames.

The average execution time of the prototype implementation is four minutes on a two-processor Sun Sparc for achieving local consistency on the example with the additional constraints posted by the user. In both solution spaces, the choices of cross-sections for the girder and the column were reduced from twenty-five possibilities to one and two respectively. The spacings and the number of elements are also reduced. Although some values within these intervals may not be part of the exact solution space, the presented spaces provide good approximations.

TABLE 3. Constraints employed in the example with the variables Cl (frame column), E (frame spacing, B (building width), Qtot1 (total load), Eft (joist spacing), Cvt (No of braced bays in the roof widthwise), Cvl (No of braced bays in the roof lengthwise), and A_d, (cross section of the frame column)

$N_{k1} = K_1 * 235 * A_{Cl}$
$K_1 = 1/(f_1 + sqrt(f_1^2 - l_{k1}^2)) \leq 1$ If $K_1 > 1$ then $K_1 = 1$
$f_1 = 0.5*(1+0.34*(l_{k1}^2 - 0.2) + l_{k1}^2)$
$l_{k1} = (1000*H)/(93.91*i_{Cl})$
$N_{q1} = 0.5 * Q_{tot1} * B * E$
$1.1 * N_{q1} \leq N_{k1}$
$235 * A_{Dlt} \geq 1.1*1.5*F_{lt}*sqrt(1+(E*Cvt)^2/(Eft*Cvl)^2)$

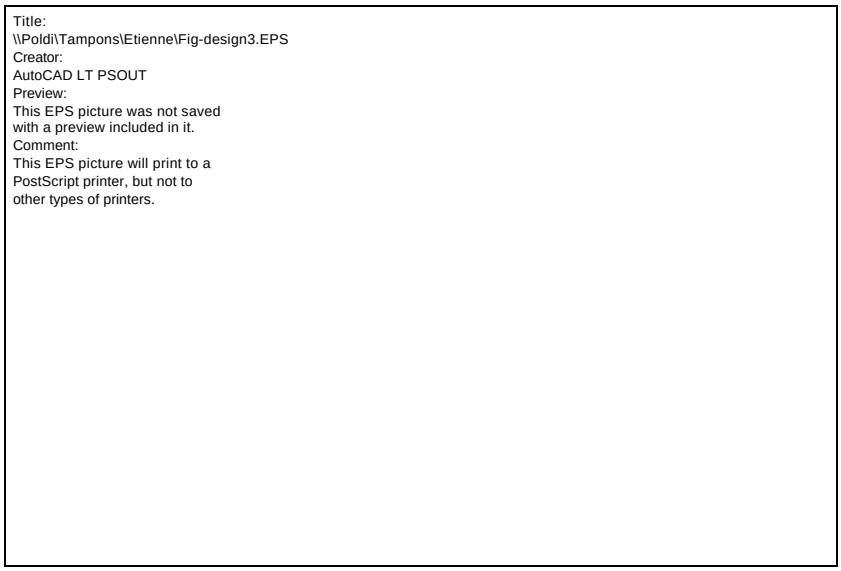

Figure 5. Design of a single story steel building. The crosses indicate the longitudinal and transversal bracing system

Also variations in the solution spaces with changes in user-posted constraints provide indications for sensitivity studies. In any case, when a solution is chosen from the solution spaces it needs to be checked against each constraint before it is adopted. A solution is derived for a structure with L = 36, B = 24, H = 10 at an altitude of 900 meters using the following cross sections: T = HEB300, Cl = HEB300, Cf = HEB300, Ft = IPE200, Fl = UNP280, Ff = UNP280, and LNP100*10 for all diagonals. The number of girts in the longitudinal bracing system is three (Cvl) and one in the transversal bracing system. This corresponds to the third possibility shown in Figure 6.

6. Conditional CSPs

Support for structural design of, for example, this industrial building can be enhanced by considering different configuration possibilities such as the bracing system (Figure 5). Some designs include only a transversal bracing system when girders are attached with moment-connections to the columns and when the frames are rigid enough to resist horizontal loads. Under these circumstances, the constraints related to the longitudinal bracing system are not relevant. This design variant is dependent on the height of the structure. When the values for height are refined during calculation of the solution

space, such a simplification may be revealed only during execution of the resolution algorithm. Complex engineering tasks often demonstrate such behaviour which can be formulated using conditional constraint satisfaction.

TABLE 4. Users may post additional constraints to reduce the size of the solution space

User constraint set 1		**User constraint set 2**	
altitude H_0	*800*	altitude H_0	*1000*
minimal surface S_{min}	*200*	minimal surface S_{min}	*400*
width *B*	*[10,40]*	length *L*	*[40,200]*
height *H*	*[8,10]*	height *H*	*[10,20]*
frame spacing *E*	*[4,5]*	Nb of frames *N*	*[1,9]*
joist spacing *Eft*	*[4,5]*	Nb of joists *Nft*	*[1,7]*
joist spacing *Efl*	*[4,5]*	Nb of windcolumns *Ncf*	*[1,10]*
joist spacing *Ecf*	[4,5]	Nb of girts *Nfl*	[1,7]
Solution set 1		**Solution set 2**	
Nb of joists *Nft*	*[3,4]*	frame spacing *E*	*[5,7.19]*
Nb of windcolumns *Ncf*	*[3,4]*	joist spacing *Eft*	*[1.16,5]*
Nb of girts *Nfl*	*3*	joist spacing *Efl*	*[1.67,3.45]*
Girder *T*	*HEB300*	joist spacing *Ecf*	*[3.45,10.19]*
Longitudinal girt *Fl*	*UNP240UNP280UNP320*	Nb of frames *N*	*[7,9]*
Transversal girt *Ff*	*UNP240UNP280UNP320*	Nb of joists *Nft*	*[3,7]*
Column *Cl*	*HEA240HEB300*	Nb of windcolumns *Ncf*	*[2,4]*
		Nb of girts *Nfl*	*[4,7]*
		Girder *T*	*HEB300*
		Column *Cl*	*HEA240 HEB300*

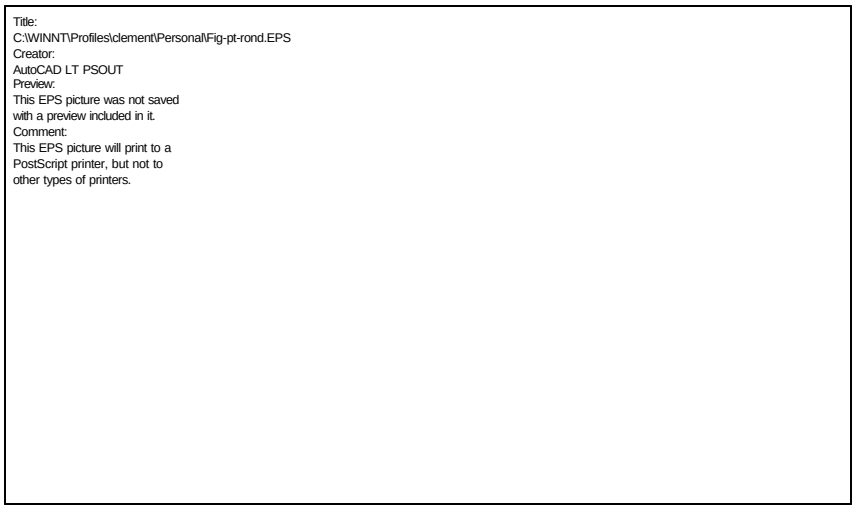

Figure 6. Different solutions for the longitudinal and transversal bracing system

A model of *conditional constraint* satisfaction (CCSP)[2] (Mittal et al., 1990) has been introduced to adapt CSPs to changing environments as illustrated by the constraints on the bracing system. The standard CSP model is extended to reason on the *presence* of variables in a solution. A variable in a CCSP can be active or not in a solution; i.e. $\forall X_i \in V$: *active*$(X_i) \rightarrow X_i = x$ *with* $x \in D_i$. Typical constraints restricting value combinations of variables are called *compatibility constraints* in this model. We extend the original definition of CCSP by allowing numeric and discrete compatibility constraints. In contrast to a static CSP, a compatibility constraint in a CCSP is only *relevant* to a problem if all the variables of this constraint are active. This means that a compatibility constraint has only to be considered in parts of the search space in which all variables of the constraint are active. Additionally, a CCSP can post constraints on a variable's activity in a given context of value assignments. Such an activity constraint has the form

$C_{X1,..,Xj} \rightarrow$ *active*(X_k) also written $C_{X1,..,Xj} \supset X_k$

where $C_{X1,..,Xj}$ is a single constraint expressing an *activation condition* under which the variable X_k becomes active. An activity constraint is *satisfied* by a set of values $\{X1 = x_1,..,Xj = x_j\}$ if $\exists\, C_{X1,..,Xj}(x_1,..,x_j) \wedge$ *active*(X_k) is true.

[2] Historically, the term used to designate constraint problems in which the set of variables in a solution varies was dynamic constraint satisfaction. However, this term is used with different meanings even within the CSP community. We therefore decided to follow the suggestion of researchers in this community and renamed it conditional constraint satisfaction.

In our model, we allow the formulation of an activation condition as a set of value assignments (a discrete constraint) and as numeric constraints. A CCSP thus consists of:

A set of variables V representing all variables that may potentially become active and be part of a solution. Each variable $X_i \in V$ has associated a domain $D_i \in D$ representing the set of possible values for the variable.

A non-empty set of *initial variables* $V_I \subseteq V$. These variables have to be part of every solution.

A set of *compatibility constraints* C^C on subsets of V representing allowed value combinations for these variables.

A set of *activity constraints* C^A on subsets of V specifying constraints between the activity of a variable and possible values of problem variables.

The goal of a CCSP is to find all solutions, where a solution S is

an assignment of values to the variables s. t. S satisfies all constraints in $C^C \cup C^A$.

minimal -- there is no solution S' satisfying all constraints such that $S' \subset S$.

complete -- all variables of V_I are assigned in S.

Minimal solutions of a CCSP contain only those assignments for which there exist no other assignment that has fewer identical variable-value pairs satisfying the same constraints. Starting from the set of initially active variables V_I, a CCSP incrementally defines spaces where different variables are active and values are only assigned to active variables.

In order to illustrate the formulation of a CCSP, the constraints associated with designing a steel structure are extended as follows. A new discrete variable *cgt* standing for column-girt connection is introduced with the domain *[moment_connection,pin_connection]* as well as a numeric variable *Hd* representing the horizontal displacement of the structure. The following activity constraints define different bracing systems:

$$cgt = pin\_connection \Rightarrow Dll \quad (1)$$

$$cgt = pin\_connection \Rightarrow Dlt \quad (2)$$

$$cgt = moment\_connection \Rightarrow Hd \quad (3)$$

$$Hd \geq f(H) \Rightarrow Dll \quad (4)$$

$$Hd \geq f(H) \Rightarrow Dlt \quad (5)$$

These constraints ensure that the longitudinal bracing system represented by the variables *Dlt* and *Dll* is only computed if either *cgt* is not a moment connection or the horizontal displacement is larger than a given function on the building height. As a consequence, the compatibility constraints over the longitudinal bracing system only become relevant when the variables *Dlt* and *Dll* are active.

In the original algorithm for solving a CCSP (Mittal et al., 1990), the condition of an activity constraint is restricted to a set of value assignments and the activation of new variable is based on partial solutions. This restriction does not allow treatment of problems in which the activation condition is i) a discrete constraint with several value combinations or ii) a numeric constraint defining infinitely many value combinations.

For solving mixed CCSPs, we propose the generation of subspaces where variables are active through consideration of different combinations of activation conditions. The activity constraints (1) and (2) lead to two subspaces, one where the longitudinal bracing system exists and the condition is satisfied with *cgt = pin-connection* and one where the condition is not satisfied (*cgt = moment-connection*). Solutions to a CCSP are found in the combinations of all these subspaces. This combination can be represented in a tree where for each constraint two sub-branches are added to each node, one with the condition satisfied and the variable active and one with the complement of the condition satisfied.

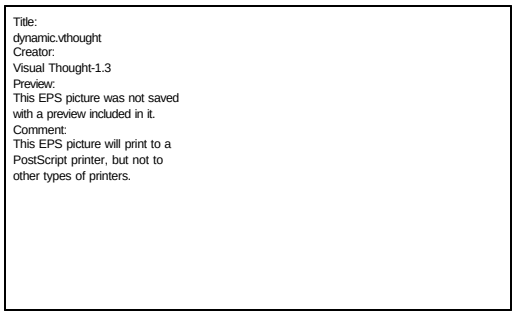

Figure 7. The constraints and variables added dynamically are represented in a combination tree. The numbers in brackets refer to the constraints in the bracing-system example

In Figure 7, each node shows only newly added variables and conditions. Trivially inconsistent subspaces are not shown. In the leaf nodes of this tree are represented the constraint combinations in which solution spaces are found. In our example, the leaf spaces P_i with additional variables V_i and constraints C_i, $i=1,2,3$ are:

P_1 : *V1 = {Dlt,Dll} C_1 ={cgt = pin_connection}*

P_2 : *V2 = {Hd,Dlt,Dll} C_2 = {cgt = moment_connection,Hd* $\geq$ *f(H)}*

P_3 : *V3 = {Hd} C_3 = {cgt = moment_connection,Hd < f(H)}*

Since each feasible space corresponds to a standard CSP, the methods presented in the earlier sections can be applied to define solution spaces. More detailed information on generalized CCSPs is given in Gelle (1998).

7. Conclusions

Properly formulated local consistency techniques provide useful support for engineering tasks when many interdependent variables of discrete and numeric nature are involved. The local consistency techniques described in this paper are able to approximate globally consistent spaces rapidly and with a reasonable degree of accuracy for constraint networks that exceed twenty variables. However, local consistency algorithms are not completely reliable. When accuracy is essential, our approach provides an effective filter for infeasible values such that its results can be the input to subsequent search or optimisation methods. Our exension to conditional CSPs allows a systematic enumeration of solution spaces, which leads to a clarification of the overall complexity of solution-space sets. As in the context of standard CSPs, local consistency is applied to remove entire spaces that are inconsistent. An example of steel-structure construction motivates the results of this paper and also demonstrates the need for conditional CSPs. The proposed methods will enhance performance and quality of future computer-aided engineering systems.

Acknowledgements

This research was funded by the Swiss National Science Foundation. The authors are grateful to D. Clément for assistance in describing the full scale example.

References

Benhamou, F.: 1996, Heterogeneous constraint solving, *in* M. Hanus, and M. Rodriguez-Artalejo, (eds), *Algebraic and Logic Programming, 5th International Conference, ALP'96*, Lecture Notes in Computer Science Springer-Verlag 1139, Aachen, pp. 62-76.

Benhamou, F. and McAllester, D. and Hentenryck, P.V: 1994, CLP (intervals) revisited, *in* M. Bruynooghe, (ed.), *Logic Programming-Proceedings of the 1994 International Symposium*, The MIT Press, pp. 124-138.

Bessière, C. and Régin, J.-C: 1996, Mac and combined heuristics: Two reasons to forsake fc (and cbj) on hard problems, *in* E. Freuder, and M. Jampel (eds), *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science 1118. Springer-Verlag, 61-75.

Davis E: 1987, Constraint propagation with interval labels, *Artificial Intelligence*, **32**.

Dechter, R.: 1990, Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition, *Artificial Intelligence*, **41**(3), 273-312.

Faltings, B.: 1994, Arc consistency for continuous variables, *Artificial Intelligence*, **65**(2), 85-118.

Faltings, B. and Gelle, E.: 1997, Local consistency for ternary numeric constraints. *Proceedings of the 11th International Joint Conference on Artificial Intelligence* IJCAI-97, pp. 392-397.

Freuder, E. C.: 1982, A sufficient condition for backtrack-bounded search, *Journal of the ACM*, **32**(4), 755-761.

Freuder, E. C.: 1982, A sufficient condition for backtrack-free search, *Journal of the ACM*, **29**(1), 24--32.

Gelle, E.: 1998, On the generation of locally consistent solution spaces in mixed dynamic constraint problems, *PhD thesis No.1826*, Swiss Federal Institute of Technology (EPFL).

Haralick, R. M. and Elliott, G. L.: 1980, Increasing tree search efficiency for constraint satisfaction problems, *Artificial Intelligence*, **14**, 263-313.

Hyvönen, E.: 1992, Constraint reasoning based on interval arithmetic. The tolerance propagation approach, *Artificial Intelligence*, **58**, 71-112

Kondrak, G.: 1994, A theoretical evaluation of selected backtracking algorithms, *Technical Report TR-94-10*, University of Alberta.

Lhomme, O.: 1993, Consistency techniques for numeric CSPs, *Proceedings of the 9th International Joint Conference on Artificial Intelligence IJCAI-93*, pp. 232-238.

Lottaz, C., Clément D, Faltings, B. and Smith, I.: 1999, Constraint-based support for collaboration in design and construction, *Journal of Computing in Civil Engineering*, **13**(1), 23-35.

Mackworth, A.: 1977, Consistency in networks of relations, *Artificial Intelligence*, **8**, 69-78.

McGregor, J.J.: 1979, Relational consistency algorithms and their application in finding subgraph and graph isomorphisms, *Information Sciences*, 19, 229-250.

Mittal, S. and Falkenhainer, B.: 1990, Dynamic constraint satisfaction problems, *in* T. Dietterich, and W. Swartout, (eds), *Proceedings of the 8th National Conference on Artificial Intelligence*, The MIT Press, pp. 25-32.

Montanari, U.: 1974, Networks of constraints: Fundamental properties and applications to picture processing, *Information Sciences*, 7, 95-132.

Moore, R. E. (ed.): 1979, Methods and Applications of Interval Analysis, *Society for Industrial and Applied Mathematics*, Philadelphia, USA.

Sabin, D. and Freuder, E.: 1994, Contradicting conventional wisdom in constraint satisfaction, *in* A. Borning, (ed.), *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, 874, Springer-Verlag.

Sam-Haroud, D. and Faltings, B.: 1996, Consistency techniques for continuous constraints, *Constraints*, **1**, 85-118.

Tinelli, C. and Harandi, M.: 1996, Constraint logic programming over unions of constraint theories, *Lecture Notes in Computer Science*, 1118, 436-450.

Van Beek, P. and Dechter, R.: 1995, On the minimality and global consistency of row-convex constraint networks, *Journal of the ACM*, **42**(3), 543-561.

Van Hentenryck, P., McAllester, D. and Kapur, D.: 1995, Solving polynomial systems using a branch and prune approach, *SIAM Journal*, **34**(2).

Van Hentenryck, P., Micher, L. and Deville, Y. (eds.): 1997, *Numerica. A modeling language for global optimization*. The MIT Press.

Waltz, D. L.: 1975, Understanding line drawings of scenes with shadows, *in* P. H. Winston (ed.), *The Psychology of Computer Vision*, McGraw-Hill, pp. 19-91.