, , 1?? ()

Consistency Techniques for Continuous Constraints

D. SAM-HAROUD AND B. FALTINGS haroud/faltings@lia.di.epfl.ch
Artificial Intelligence Laboratory(LIA)
Computer Science Department (DI)
Swiss Federal Institute of Technology (EPFL)
IN-Ecublens, 1015 Lausanne
Switzerland

Abstract. We consider constraint satisfaction problems with variables in continuous, numerical domains. Contrary to most existing techniques, which focus on computing one single *optimal* solution, we address the problem of computing a compact representation of the space of *all* solutions admitted by the constraints. In particular, we show how globally consistent (also called *decomposable*) labelings of a constraint satisfaction problem can be computed.

Our approach is based on approximating regions of feasible solutions by 2^k-trees, a representation commonly used in computer vision and image processing. We give simple and stable algorithms for computing labelings with arbitrary degrees of consistency. The algorithms can process constraints and solution spaces of arbitrary complexity, but with a fixed maximal resolution.

Previous work has shown that when constraints are *convex* and binary, path-consistency is sufficient to ensure global consistency. We show that for continuous domains, this result can be generalized to ternary and in fact arbitrary n-ary constraints using the concept of (3,2)-relational consistency. This leads to polynomial-time algorithms for computing globally consistent labelings for a large class of constraint satisfaction problems with continuous variables.

Keywords: consistency algorithms, convexity, global consistency, constraint satisfaction, constraint propagation, interval arithmetic, forward checking, numerical constraints, continuous constraints

1. Introduction

Many problems, ranging from resource allocation and scheduling to fault diagnosis and design, involve numerical constraint satisfaction as an essential component. A constraint satisfaction problem (CSP) can have one, several or no solutions. Most commonly used constraint solvers attempt to compute a single solution, optimal according to some criterion. Most solvers are based on linear and non-linear programming, others use numerical analysis, hill-climbing or stochastic techniques.

In many applications, the constraint satisfaction problem is embedded in a larger decision process. In this case, it may be desirable to compute the space of *all* solutions. This has the advantage that no optimization criterion must be formulated beforehand, and that the space of possibilities can be explored systematically. In certain applications, such as diagnosis, design and configuration, these are important advantages. In this paper, we explore algorithms which construct compact descriptions of the complete solution space.

In computing the complete solution space for a CSP, the main problem is how to represent it in a compact manner: in the worst case, the number of solutions to a CSP may be exponential in the number of variables and it is clearly not practical to simply give a list of all

possibilities. When variable domains are continuous, the solution space may be a complex shape in many dimensions which is difficult to represent and reason about.

In *consistency* techniques, the solution space is represented compactly by *labels* assigned to individual variables and constraints. When the labeling is *globally consistent*, each label contains only values or value combinations which occur in at least one solution. To narrow down the space, the user can restrict or fix the value of any one of the variables. This restriction is then be propagated through the constraints (more precisely, the consistent labels of the constraints) and will result in corresponding restrictions on all other variables. Such a labeling is a compact, sound and complete representation of the solution space admitted by the constraint satisfaction problem. In this paper, we focus on the problem of how to compute such a labeling for constraint networks with continuous variables.

In discrete domains, labels are represented simply as enumerations of values or value combinations. In continuous domains, sets of individual values are often compact and can be represented by one or a small collection of intervals. However, representing and manipulating labels of several variables is more involved as they may be complex geometric shapes. In this paper, we present consistency algorithms using the quadtree/octree representation developed in computer vision to compute a labeling of any degree of consistency in continuous domains.

One can distinguish different orders of consistency according to the size of subnetworks taken into consideration: 1-consistency for networks involving 1 variable only, 2-consistency for 2 variables, and in general k-consistency for networks involving k variables. Freuder (Freuder, 1978) provides an algorithm for computing k-consistent labelings for constraints in discrete domains whose runtime is exponential in k.

While in general, computing a consistent labeling is NP-hard, recent results show that in the case where constraints are *convex*, low orders of consistency are equivalent to global consistency. For the cases of (i) binary discrete and (ii) binary temporal constraints (involving at most 2 variables), it has been shown that *path*-consistency (also called 3-consistency) is equivalent to global consistency (van Beek, 1992), (Dechter *et al.*, 1990). In this paper, we show that similar results can be stated for continuous constraints of arbitrary types and arities, contrary to the discrete case where generalization to non-binary constraints loses the polynomial time complexity bounds (van Beek and Dechter, 1995).

Practical motivation

This work has been motivated by a project on intelligent design systems (Faltings *et al.*, 1992), (Haroud *et al.*, 1995) which revealed the shortcomings of existing techniques and brought to light technological requirements inherent to a wide variety of practical problems.

Many practical continuous constraint satisfaction problems are embedded in decision processes. The set of variables and constraints is not independent of particular solutions and may change depending on the value chosen for certain variables. Figure 1 shows an example from civil engineering which illustrates this point. The problem is to design a floor consisting of a concrete slab on steel beams. Depending on the values chosen for depth and span of the beam, different design alternatives are possible, shown in Figure 2:

- as the depth to span ratio becomes higher, susceptibility to vibrations increases. This requires installing bridging (lateral reinforcements) for damping the floor (Figure 1, option -a-). This is the case when the beam dimensions fall into regions 3 and 4.

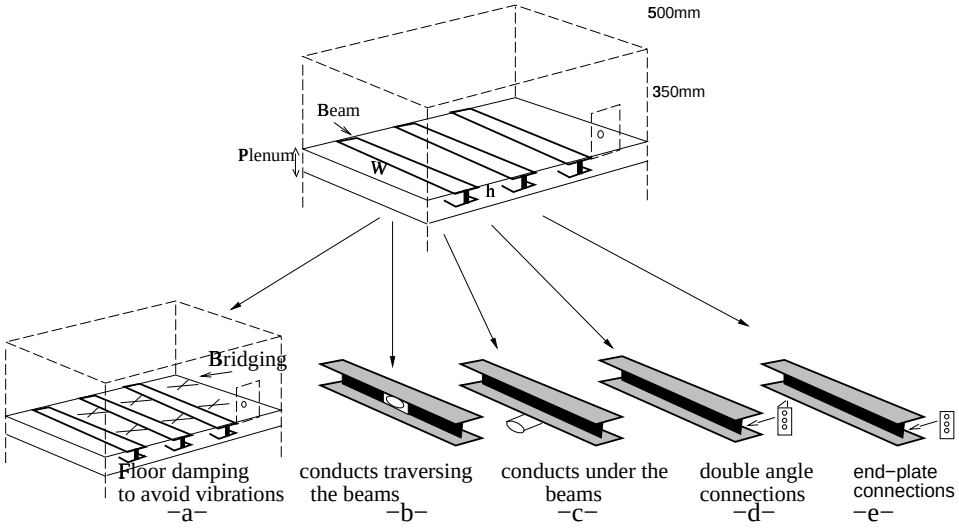


Figure 1. A civil engineering design problem presenting various design options.

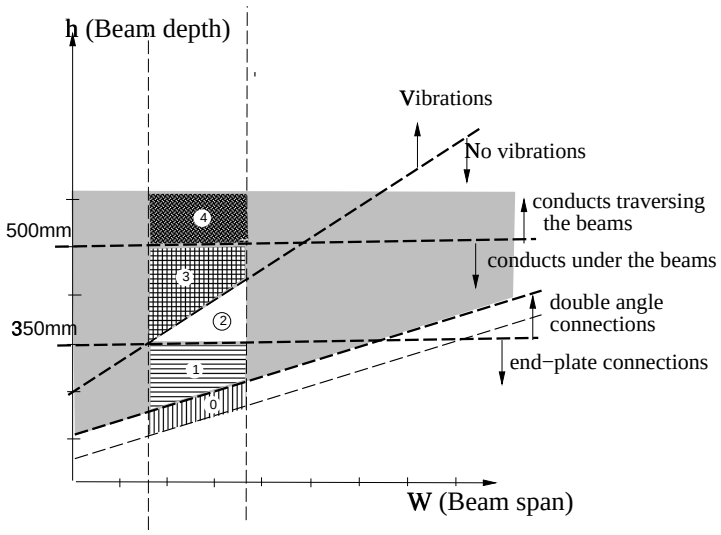


Figure 2. Dependence of design options on values chosen for beam depth and span.

- The plenum is the space situated between the floor of one story and the ceiling of the story below it (Figure 1). The plenum is often a fixed height, and there is a general preference for having the ventilation ducts go below the beams within this plenum(Figure 1, -c-). However, for a relatively deep beam where the height (h) is greater than 500mm (region 4), it is generally no longer possible for those ducts to go under the beams. At this point, rectangular openings must be made in the beams (Figure 1, -b-) to allow passage of the ventilation ducts.

- For connecting beams at their ends, the most economical connection is by end-plates (Figure 1, -e-), but this is only feasible for a depth of less than 350mm (regions 0 and 1). For greater depths, a double-angle connection (Figure 1, -d-) might be more advantageous (regions 2, 3 and 4).

When numerical values of parameters serve as the basis for decisions and impact the whole resolution process, the identification of *single point solutions* satisfying the set of active constraints usually fails to identify all options. In the example of Figure 1, a traditional mathematical method such as linear programming would identify a single solution within one of the regions 0,1, 2, 3 or 4. This forces the designer into a particular choice of design alternatives purely on the grounds of idiosyncrasies of the constraint solving procedure and his optimization criterion.

Using consistency techniques provides the user (or another program) with *ranges* of feasible values as sound and as complete as possible which allow rational decisions. In this example, it turns out that *global* consistency is important: the beam's depth and span are linked to other variables (slab thickness, beam spacing) through several non-linear equations. Region 0 eventually turns out to be inconsistent with these constraints, but this is not detectable using only local consistency methods. Thus, a designer might choose solutions which ultimately would turn out to be infeasible.

In practice, continuous variables admit an infinite set of values. By restricting the form of constraints, it becomes possible to reason about admissible values algebraically. In this paper, we pursue an alternative solution. In practice, most continuous variables can only be fixed with a certain maximal precision. For example, it would be unimaginable to construct a bridge of 10 km whose length would be precise to the micrometer range. Thus, solution spaces are only useful when they require no more than some maximal precision. We therefore discretize variable ranges to the maximal useful resolution. This leads to algorithms which are stable and applicable to constraints of arbitrary form.

We start this paper by reviewing related work. We go on to present the constraint and solution space representations, and algorithms for making them consistent. Finally, we show how and in what circumstances we can efficiently compute globally consistent labelings, and give a concrete example of an application.

2. Processing continuous constraints

While extensively applied to discrete problems, consistency techniques have had few results for of continuous constraints. This is not surprising when we consider the fact that

consistency algorithms are mainly used as preprocessing procedures for enabling search, which is most useful for discrete problems. Although continuous feasibility spaces preclude enumeration in the strict sense, the notion of constraint propagation for local pruning is appealing for continuous problems by providing a rapid and flexible approximation technique.

In this work we investigate the contribution of propagation techniques to constraint satisfaction problems in continuous domains.

In the following, a *continuous CSP* (CCSP),$(P = (V, D, R))$, is defined as a sequence V of *variables* $x_1, x_2, \ldots x_n$, taking their values respectively in a set D of continuous *domains* $D_1, D_2, \ldots, D_n$ and constrained by a set of *relations* $R_1, \ldots, R_m$. A *domain* is an interval of $\Re$ and a *relation* is defined intensionally by a set of arbitrary equalities and inequalities. Constraints are usually given as numerical *equalities* and *inequalities* of arbitrary types and arities.

In a k-ary CCSP, every subset of variables $\{x_1, \ldots x_k\}$ has a single relation $R_{x_1 \ldots x_k}$ which defines consistent value combinations. This relation may summarize several constraints involving the same variables and is also called a *total constraint*(Faltings, 1994).

Consistency techniques in continuous domains

Early work on constraint propagation for continuous domains, such as the work of Davis (Davis, 1987), has shown little success. The convergence of local propagation techniques, such as the Waltz algorithm, cannot be guaranteed. Furthermore, when applied in a straightforward manner, it is not even possible to guarantee that local propagation ensures arc-consistency! A good insight into the problems encountered can be found in the paper of Davis (Davis, 1987).

For a long time these negative results have been attributed to the analytical complexity inherent to solving general systems of equations and inequalities. Recent advances, (Hyvönen, 1992), (Faltings, 1884), (Lhomme, 1993) and (Benhamou *et al.*, 1994), have however questioned these conclusions and proposed various remedies and significant improvements.

Yet, since composing continuous constraints numerically is a complex task, most of the techniques for continuous CSPs work with the unary projection of constraints rather than constraints themselves and the vast majority of propagation techniques are variations of the same theme — fixed-point iteration via arc-consistency algorithms — with emphasis put on improving the tightening the outer bounds on variable domains. Constraints are generally roughly approximated by the feasibility domains of the variables they involve and the combination of constraints then reduces to elementary operations on intervals.

The first practical implementations limited themselves to basic interval arithmetic operations for refining the intervals labels, through fixed point iterates. Since interval arithmetic often produces unrealistic estimates, these implementations are relatively weak. Improvements have been proposed based on:

- A specific adaptation to particular types of constraints like in CLP(F), (Hickey, 1994), which is devoted to functional constraints

- A capability of handling multiple solutions like in ECHIDNA, (Sidebottom and Havens, 1992), where real domains are represented by interval hierarchies allowing control

of the constraint processing precision and providing a hierarchical version of consistency algorithms.

The most significant improvements of arc-consistency techniques in continuous domains are given by the works of Lhomme (Lhomme, 1993), Benhamou *et al.* (Benhamou *et al.*, 1994) and Faltings (Faltings, 1994). The two former works share some similarities with our work, while the latter tackles the issue from a different perspective and analyzes the cause behind the bad performance of traditional arc-consistency techniques in continuous domains. In (Lhomme, 1993), Lhomme proposes an interval propagation formalism based on bound propagation. A new consistency concept, weaker than arc-consistency and called 2-B-consistency, is introduced that assumes the convexity of variables' domains. In the case where the effective domains are disjunctive, 2-B-consistency will consequently admit local inconsistencies within the labels. In Lhomme's approach, constraints are decomposed into primitive (basic) constraints allowing an easy definition of extrema functions. The relaxation step of the 2-B-consistency algorithm — analog to AC-3 — uses these extrema functions to refine variables' domains upon fixed-point iterate. In (Benhamou *et al.*, 1994) is presented a logic programming language called `Newton` extended to handle systems of non-linear equations and inequalities as well as constrained and unconstrained optimization. `Newton` implements a relaxed version of arc-consistency called box-consistency using the combination of interval arithmetic with a binary search technique. The goal of box-consistency is to determine an outer enclosing approximation of each variable's domain, as tight as possible. Once the approximation is determined, `Newton` uses a branching mechanism for isolating solutions (van Hentenryck *et al.*, 1995).

Both 2-B and box-consistency assume that variable domains are convex and focus on optimizing the tightening of the feasibility space outer bounds. Their principal difference lies in the fact that 2-B consistency requires interval tools for computing the unary projections of constraints, while box-consistency works directly on the original constraints and approximates the projections using the combination of interval Newton iterates and bisection.

In (Faltings, 1994), the author shows that some undesirable features of propagation algorithms with interval labels must be attributed to the inadequacy of the propagation rule and to a lack of precision in the solution space description. Faltings demonstrates that by using a network of total constraints along with a relaxation rule — based on the identification and classification of local extrema — sound and locally complete propagation becomes possible. Faltings' algorithm has been implemented for both binary and ternary constraints with good practical results.

In the presence of *cyclic* constraint networks, fixed-point iterations cannot guarantee convergence. Lhomme addresses the problem by introducing the notion of 2-B(w)-consistency — a partial form of 2-B-consistency — characterizing the degree of consistency obtained when the refinement process terminates abnormally (looping problems). w characterizes the imprecision of the computed bounds and 2-B(w)-consistency is equivalent to 2-B-consistency when $w = 0$. Using 2-B(w)-consistency, the complexity of the consistency technique can also be tuned by fixing the precision desired for the resulting interval bounds.

Path-consistency algorithms such as PC-1 and PC-2 have been applied successfully to some

restricted classes of continuous constraints (temporal constraints (Dechter *et al.*, 1990) and spatial constraints (Tanimoto, 1993). In the general case however, and in the absence of reliable tools for combining numerical constraints, it is difficult to reach even arc-consistency. Thus, only a few authors have investigated the issue of algorithms for higher degrees of consistency in continuous domains.

One of the first attempts, by Hyvönen (Hyvönen, 1992), introduced the notion of *tolerance propagation*(TP) which generalizes the idea of numerical exact propagation into interval propagation. A local tolerance propagation algorithm, closely related to the Waltz filtering algorithm, is proposed that can be generalized into global TP for determining globally consistent solutions. The idea behind global tolerance propagation is to use global solution functions instead of local ones during propagation. Since the evaluation of complicated interval functions is often computationally expensive, the author proposes a partial globalization schema which determines global solution functions only with respect to some critical variables and some subnets of the original constraint net. Lhomme (Lhomme, 1993) proposed bound propagation algorithms whose complexities can be tuned by fixing the precision of the resulting interval bounds. These algorithms can reach higher degrees of consistency than arc-consistency but refine uniquely the outer bound of the feasibility domains.

Backtrack-free search

A globally consistent (or decomposable) labeling is a compact and conservative representation of all solutions admitted by the constraint network. 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.

In a *globally consistent* network (also called *decomposable* or strongly n-consistent), any partial consistent instantiation of a subset of variables can be extended to a *solution* with no backtracking (Dechter, 1990), a process which can generally be carried out in linear time. Minimality as defined by Montanari (Montanari, 1974) for binary networks, guarantees that each pair of values allowed by the constraints participates in at least one solution. A decomposable network being always minimal while the opposite is not true, so minimality is a weaker property.

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 will never require any backtracking.

In general, a globally consistent labeling may require explicitly representing constraints for all variables in the problem, a task which has exponential time complexity in the worst case. However, under certain conditions local consistency algorithms are sufficient to compute a globally consistent solution space or at least to bound the remaining search effort.

For example, it has been shown, (Freuder, 1982), for discrete constraints that when the constraint network is a tree, arc-consistency also guarantees global consistency. This is an example of how the topology of the constraint *network* can simplify the computation of

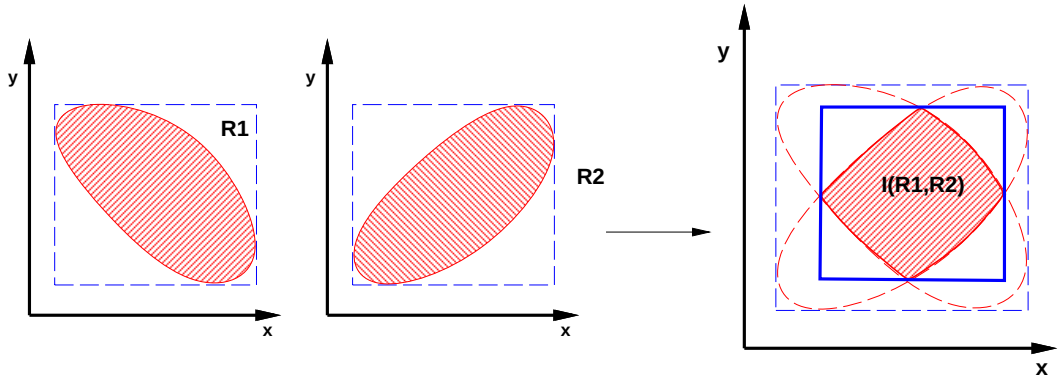


Figure 3. The enclosing rectangle of an intersection of regions R_1 and R_2 is in general different from the intersection of the enclosing rectangles of R_1 and R_2.

a globally consistent labeling (Freuder, 1978). Similarly, path-consistency is sufficient to ensure backtrack-free search in networks whose topology is *series-parallel*.

It has also been shown that when all constraints satisfy a *convexity* condition and are binary, path-consistency ensures global consistency irrespective of the topology of the network (van Beek, 1992), (Dechter *et al.*, 1990). In this paper, we use similar convexity conditions on the constraints which allow efficient algorithms independently of the network topology. These will allow tractably computing decomposable labelings of continuous CSPs.

Related fields

Interval arithmetic methods compile compact enclosing approximations of the solution space. They are consequently also appealing when it is needed to reason on a spectrum of feasible alternatives rather than on isolated solutions. In interval arithmetic, real numbers are approximated by intervals and arithmetic expressions are computed by applying the operators of the formula to the endpoints of the intervals of its arguments. Systems of linear and non-linear interval equations can be solved using the interval counterparts of iterative algebraic or numerical techniques. From the efficiency and robustness standpoints, interval arithmetic presents roughly the same advantages and shortcomings as the iterative techniques it derives from. While consistency techniques are based on the construction of *partial solutions*, interval arithmetic handles the entire set of constraints as an indivisible whole. It is consequently less scalable to large problems. Moreover, both linear and non-linear interval methods require preconditioning steps or may impose rigid applicability conditions (regular matrices, globally dominant Jacobians etc…). In practice, consistency techniques and interval arithmetic are often used together (Older and Vellino, 1993), (Benhamou and Older, 1994), (Lee and Emden, 1993), (Hickey, 1994).

3. Constraint and Label Representation

In continuous domains, constraint generally arise as algebraic or transcendental equalities and inequalities involving several variables. Simple propagation algorithms, such as the local propagation algorithm investigated by Davis (Davis, 1987), process constraints directly in this form. This has several important disadvantages:

- when there are several constraints involving the same variables, each is represented by a separate relation. This creates unnecessary cycles in the network which complicate the propagation process. Thus, applying the Waltz algorithm in general does not even result in an arc-consistent labeling (see (Davis, 1987)). Note that if only arc-consistency is required, there exists a better propagation rule which is guaranteed to find it (see (Faltings, 1994)). It is based on propagating all constraints between the same set of variables simultaneously and thus avoids the artificial loops introduced by simpler algorithms.
- local propagation of continuous constraints may never terminate (see (Davis, 1987) and (Faltings, 1994)). This is unavoidable since constraint propagation can be (ab-)used to solve systems of nonlinear equations by fixed point iteration. However, termination can be guaranteed by imposing a maximum precision beyond which no further refinement takes place.
- it is very difficult to give labelings for intersected or composed constraints. For example, the intersection of the solutions allowed by the constraints $f(x, y) > 0$ and $g(x, y) > 0$, where f and g are some algebraic or transcendental functions of x and y, can in general not be specified in a similar form. Such intersections and compositions are required for computing higher order consistent labelings.

One way of overcoming these difficulties is to approximate solution spaces by enclosing rectangles or boxes. Although this representation is often useful to narrow down the space of solutions to be considered, it usually generates solution spaces which are much too large. This is illustrated by Figure 3.

In order to obtain a more powerful representation for general solution spaces which avoids the problems cited above, we make the following two assumptions:

- each variable domain is restricted to a bounded interval. This assumption is reasonable in most practical problems.
- for each variable, there is a maximum precision with which results can be used. This assumption is also very reasonable, reflecting limitations in manufacturing, measurement or control precision.

Provided that these two assumptions hold, a solution region $R_{x_1 \ldots x_k}$ can be approximated by a hierarchical decomposition of its solution space into 2^k-trees. For binary relations, these are called quadtrees, for ternary ones, octrees. Figure 4 shows an example of a quadtree representing a binary relation. Each node represents a k-dimensional sub-region of the original domain (i.e. the domain over which the decomposition is carried out). A node has one of three possible types:

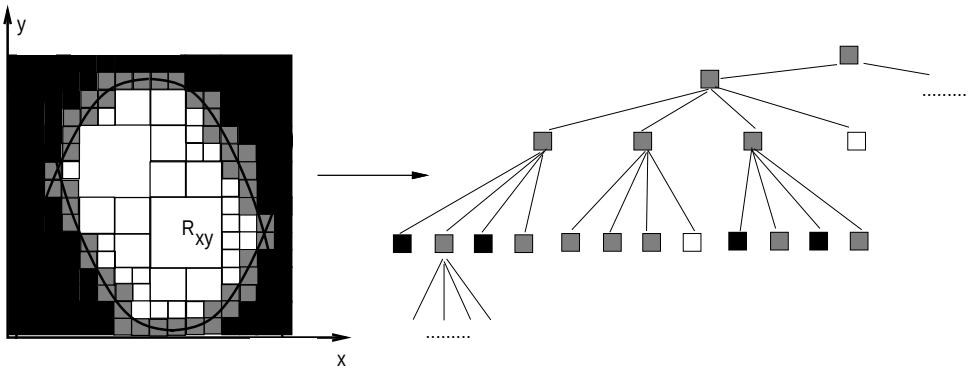


Figure 4. A binary relation R_{xy} can be approximated by a hierarchical binary decomposition of its solution space into a quadtree.

- *white*: if all points within the region are consistent
- *grey*: if some points within the region are consistent and some are inconsistent
- *black*: if all points within the region are inconsistent

Each grey k-dimensional cube is decomposed into 2^k smaller ones whose sides are one half the length of the original cube, and which form the children of the grey node in the hierarchy. When a node is black or white, the recursive division stops. Unless the boundaries of a region are parallel to the coordinates axes, infinitely many levels of representation would be required to represent a region precisely. However, when there is a fixed minimum resolution, any grey node with a smaller size than this resolution need not be decomposed any further. This looses some solutions, but as long as there are more solutions than the resolution limit it is guaranteed that some solutions remain.

In the case of *equality* constraints, a strict application of the binary decomposition into a 2^k-tree would leave only grey nodes at the resolution limit which would then be declared black. Consequently, there would be no legal values in this representation. We therefore introduce the notion of *toleranced equalities* where a strict equality constraint is translated into a weaker form where the final grey nodes are all replaced by white rather than black nodes. This amounts to replacing each equality by two inequalities close to each other. Since this might introduce spurious solutions around the fringes of the solution space, for each equality it must be carefully checked whether the effects of this tolerancing will be acceptable or not.

N-ary constraints (N>2)

In many realistic problems, the constraints are not binary, but n-ary. 2^k trees are completely general and can represent constraints of any arity. However, the complexity increases significantly when constraints of high arity are present. In the worst case, the space requirements grow exponentially since n-ary constraints are represented by 2^n-trees.

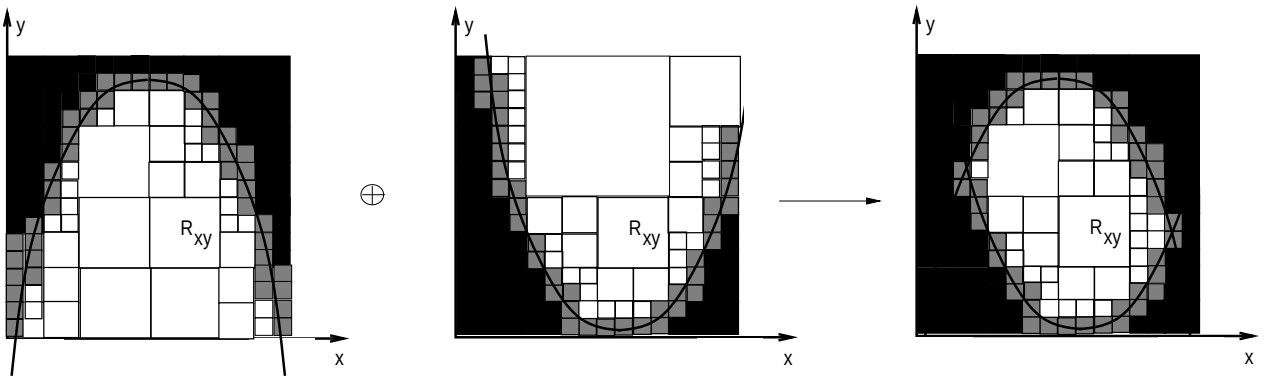


Figure 5. A total constraint between a pair of variables is constructed by superimposing the quadtree representations of all individual constraints among these variables.

In general, it is not possible to find binary constraints which are equivalent to an n-ary one. However, it is possible to syntactically transform an arbitrary n-ary constraint into a set of *ternary* equalities and inequalities. For example, the 5-ary CSP with one constraint, $(x-y)^2 + \frac{(z+t)}{u} > 2$, can be translated into a ternary one with three constraints: $w_1^2 + (w_2/u) > 2$, $w_1 = x - y$, $w_2 = z + t$. Thus, as long as representing the resulting equalities by toleranced equalities is acceptable, it is sufficient to solve systems of ternary constraints.

In this paper, as well as in the system we implemented using the algorithms, we only treat the case of binary and ternary constraints, represented by quad- and octrees.

2^k-tree construction

When there are several constraints curves between the same set of variables, the 2^k-tree representing the total constraint between them is constructed by first constructing 2^k trees for each constraint curve in isolation, and them intersecting the resulting trees into a single one. This is possible since the initial interval for the variables is fixed in the same manner for each constraint; thus the decomposition occurs at identical intervals and can be combined by logical operations. Given an ordering $white < grey < black$, the rule for determining the feasibility of a node obtained by intersection is expressed as:

$$color(node_1 \oplus node_2) = Max(color(node_1), color(node_2)) \quad (1)$$

Constructing the 2^k-tree representation of a single constraints only requires evaluating constraint equations at certain points in the space. For determining the type of a subregion (k-dimensional cube) created in the decomposition, three cases have to be distinguished:

i. if the constraint curve *traverses* the region, i.e. not all corners fall on the same side, the region is grey.

ii. if the constraint curve is *contained* in the region, a pre-processing phase must be carried out in order to split the curve into transverse segments.The role of the preprocessing phase is to check whether a solution exists within the initial approximation. This task

can be achieved for example, by carrying out a binary search for determining a division which intersects the constraint curve. The complexity of such a numerical search is reduced in practice by the fact that initial domains are the most often set to reasonable values and that a limited degree of precision is required.

iii. if the constraint curve passes *outside* of the region, the region is either white or black depending on which side it falls, which can be tested on any of its corners.

For more details about 2^k-tree construction, we refer the reader to (Haroud and Faltings, 1994) and (Sam-Haroud, 1995).

Correctness of the representation

The 2^k-tree representation of constraints provides two possible approximations of the actual solution space $\mathcal{S}$:

- the inner content approximation, $\mathcal{I}(\mathcal{S})$, is given by the white nodes (interior nodes) and is entirely enclosed within the solution space. Since all values within $\mathcal{I}(\mathcal{S})$ are consistent, it is a *sound* approximation. However, some solutions maybe missing from this representation.

- the closest outer content approximation, $\mathcal{O}(\mathcal{S})$, is given by the union of the white and grey nodes (interior nodes $\cup$ boundary nodes). This approximation is guaranteed to contain all solution, but it may be *not sound* since the grey nodes contain inconsistent values.

With regard to equalities, remember that this method only allows toleranced inequalities and soundness will only hold with respect to these tolerances.

For the initial 2^k-tree representations of *individual* constraints, we can always guarantee that $\mathcal{I}(\mathcal{S})$ and $\mathcal{O}(\mathcal{S})$ are as close as possible to the actual solution region. However, constructing total constraints and enforcing consistency involves composition and intersection operations on constraints.

Letting $\mathcal{S}_1 \oplus \mathcal{S}_2$ denote the solution space resulting from the intersection of $\mathcal{S}_1$ and $\mathcal{S}_2$. We then have the following Lemma (see (Sam-Haroud, 1995)):

LEMMA 1 *The inner content approximation is exact with respect to intersection:*

$$\mathcal{I}(\mathcal{S}_1) \oplus \mathcal{I}(\mathcal{S}_2) = \mathcal{I}(\mathcal{S}_1 \oplus \mathcal{S}_2)$$

The outer content approximation may contain spurious nodes after intersection:

$$\mathcal{O}(\mathcal{S}_1 \oplus \mathcal{S}_2) \subseteq \mathcal{O}(\mathcal{S}_1) \oplus \mathcal{O}(\mathcal{S}_2)$$

Proof. By definition, all points in a white node are consistent. A node is therefore white in the $\mathcal{I}(\mathcal{S})$ or $\mathcal{O}(\mathcal{S})$ representation of $R_1 \oplus R_2$ if and only if it is also white in the $\mathcal{I}(\mathcal{S})$ and $\mathcal{O}(\mathcal{S})$ representations of R_1 and R_2. However, due to imprecision considerations, a node computed by intersection can be grey even if it contains no solution:

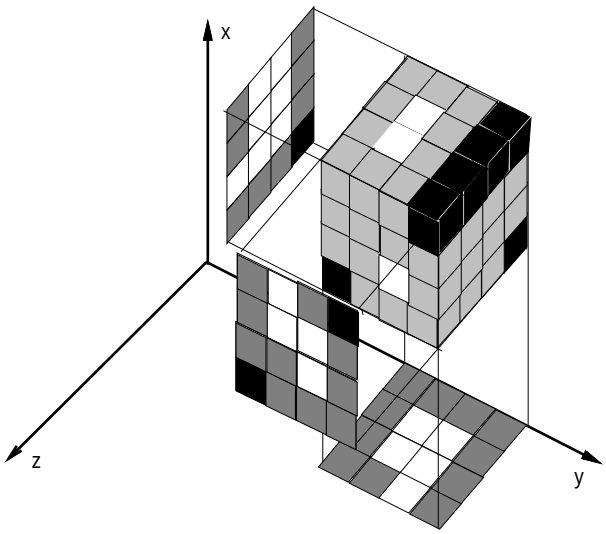
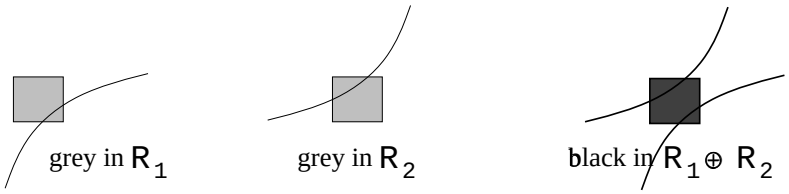


Figure 6. 3-dimensional nodes are computed by composing their facets (2-dimensional nodes), and vice versa, 2-dimensional nodes can be obtained by projecting the related 3-dimensional nodes over one of their facets



□.

For composition, it is possible to show that the projection of a constraint into a higher-dimensional space is exact for both inner and outer approximations. Thus, we have that:

- the $\mathcal{I}(\mathcal{S})$ representation of total constraints is exact, even after executing consistency algorithms. This means that it is both sound (not containing any spurious inconsistent values) as well as maximal in the sense that there is no larger sound $\mathcal{I}(\mathcal{S})$ approximation.
- the $\mathcal{O}(\mathcal{S})$ representation computed by logical combination of simultaneous constraints is complete but not sound with respect to the minimal enclosing approximation $\mathcal{O}(\mathcal{S})$ — spurious grey nodes can be created by intersections.

4. Consistency algorithms using 2^k-trees

The 2^k-tree constraint representation allows applying consistency algorithms such as AC-3 (Davis, 1987) for arc-consistency, PC-1 (Montanari, 1974) and PC-2 (Mackworth, 1977)

for path-consistency as well as their generalizations for higher degrees of consistency. Algorithms for computing a k-consistent labeling ((Freuder, 1978), (Mackworth, 1977), (Montanari, 1974)) compute labels for each subset of k-1 variables. Each label is computed so that it admits only value combinations of the $k-1$ variables which are consistent with a least one value in the label of a kth variable, for any kth variable. In the algorithm of Freuder (Freuder,1978), the label for the set of variables $x_1, x_2, ..., x_{k-1}$ is computed as the intersection ($\oplus$) of any existing constraint between $x_1, x_2, ..., x_{k-1}$ and all *induced* constraints involving these variables and one additional k-th variable. An induced constraint is given by the *composition* ($\otimes$) of a pair of (k-1)-ary labels, each of which involves all but one of the k-1 variables $x_1, .., x_{k-1}$, and one additional variable x_k. Composition involves extending the (k-1)-ary constraints into the k-dimensional space spanned by $x_1, .., x_{k-1}$ together with x_k, intersecting the resulting volumes, and *projecting* the intersection into the subspace $x_1, .., x_{k-1}$. Figure 6 illustrates this process for k=3.

Using the 2^k-tree representation, all three operators can be implemented using purely logical operations. Given the ordering $white < grey < black$, rules for determining the feasibility of a node obtained by one of these operators can be expressed as follows:

i. $color(node_1 \oplus node_2) = Max(color(node_1), color(node_2))$

ii. $color(node_1 \otimes node_2) = Max(color(node_1), color(node_2))$

iii. $color(\prod(node)) = Min(color(node_i))$
where $node_i$ are all nodes having $\prod(node)$ as facet.

Composition can be implemented by first extending the (k-1)-dimensional constraints into k-dimensional space and then projecting back the result into the (k-1)-dimensional subspaces, as shown in Figure 6. For example, the relaxation operation required by the arc consistency algorithm AC-3 can be implemented as follows:

$$T'_{ii} = T_{ii} \oplus \prod_i (T_{ij} \otimes T_{jj}) \tag{2}$$

and the one for the path consistency algorithm PC-2:

$$T'_{ij} = T_{ij} \oplus \prod_{ij} (T_{ik} \otimes T_{kk} \otimes T_{kj}) \tag{3}$$

where T_{xy} denotes the 2^k-tree representation of a constraint between variables x and y, and $\oplus$ and $\otimes$ denotes respectively intersection and composition of 2^k-trees. We show in (Haroud and Faltings, 1994) that PC-2 computes a path-consistent network representation of binary CCSPs in $O(n^3 w^2 log_4(w))$ where w is the number of feasibility nodes in the largest tree and n is the number of variables. We also propose in (Sam-Haroud, 1995) a variant of Cooper's k-consistency algorithm (Cooper, 1989), adapted to the 2^k-tree representation of constraints.

It may appear that the 2^k-tree representation only allows a rough approximation of the constraints. However, all known propagation algorithms for general constraints involve

numerical analysis techniques with a finite precision. For example, the Waltz filtering algorithm for achieving 2-consistency (arc-consistency), whose performance for continuous domains is analyzed by Davis (Davis, 1987), involves fixed-point iterations whose convergence is notoriously bad. In the example of arc-consistency, using the 2^k-tree decomposition amounts to replacing it by binary search, a much more efficient technique. In general, the 2^k-tree representation is an efficient and stable alternative to analytic representations of region boundaries.

Complexity

Table 1. The number of nodes effectively computed while constructing a 2^k-tree is generally smaller by far than the worst case estimate.

Constraint	k	d	Real # of nodes	Expected # of nodes
$\sqrt{x^2+y^2} < 50$	2	6	688	5461
$\frac{12y}{\sqrt{x^2+y^2}} < 10$	2	6	340	5461
$y = x^2$	2	7	685	21845
$\cos(\phi_2 - \phi_1) > 0$	2	7	1492	21845
$x = r\cos\phi$	3	5	4740	37449
$y > 0.086x - 5.51e^{-5}x^2 + 8.36e^{-9}x^3$	2	5	115	1365
$z = -y - 9.63e^{-5}(0.041x)^{1.51}$	3	5	4117	37449
$x < (y+z)^{1/3}$	3	5	2857	37449
$x > 2/5(y+z)$	3	5	2787	37449
$x < \sin(\phi_2 - \phi 1)$	3	4	988	4681
$x < \frac{y}{\sqrt{z}}$	3	4	1107	4681
$x < y + z$	3	4	1072	4681

The complexity of the 2^k-tree based consistency algorithms depends on the number of nodes of the hierarchical decompositions. In the worst case, the 2^k-tree resulting from the construction procedure is complete, i.e. all leaf nodes have maximal resolution. Letting d be the maximum depth of recursive divisions, the number of nodes in a 2^k-tree is given by the following formula:

$$\sum_{i=0}^{d} 2^{k(d-i)} = \frac{2^{k(d+1)} - 1}{2^k - 1} \tag{4}$$

Hence, the number of nodes in a 2^k-tree is roughly in $O(2^{k*d})$ in the worst case where k is the arity of tree and d the maximum granularity. For practical cases, d is given by the term s/ε where s is the maximum domain size (the largest interval size in D, for a CCSP (V, C, D)) and ε the tightest interval size accepted for variables of V. In most cases, the measurements given above strongly overestimate the complexity. A more realistic measure should be done in terms of the number of grey nodes generated, since the recursive quartering stops as soon as a node color is set to white or black. More precisely, we can show

that the number of grey nodes generated while decomposing a region is in $O(b^{k-1})$ (Tanimoto, 1993) where b measures the boundary size of the represented k-dimensional region. When the boundary delimits sufficiently wide regions ($>> \varepsilon^k$), the number of grey nodes generated is less than $\frac{2^{k*(d+1)}-1}{2^k-1}$ by far. Table 1 illustrates this point on a sample of binary and ternary constraints derived from practical problems.

Note finally that Tanimoto (Tanimoto, 1993) has shown the interest of 2^k-trees for handling a particular type of continuous constraints (spatial constraints). Tanimoto's work and ours, though carried out independently with different objectives, lead to related methodologies. This can be seen as favorably illustrating the relevance of discretized hierarchical decomposition for designing reliable continuous consistency techniques.

Advantages of 2^k-trees

In comparison with with other ways of representing continuous constraints, 2^k-trees have several advantages which are crucial for consistency algorithms:

1. *Discretization prevents infinite looping:* At each relaxation step performed by consistency algorithms on 2^k-trees, the intervals contained in the involved labels are constructed by an implicit binary search: each successive relaxation step refines the interval bounds to an interval half the size of the previous one until maximum granularity is reached. Consequently, the decomposition into 2^k-trees has the important advantage of ruling out infinite cycling of the propagation algorithm as observed for the Waltz algorithm applied to continuous domains. While the majority of consistency algorithms perform unstable fixed point iterations, the binary search method using the 2^k-tree decomposition guarantees stability and convergence. The methods presented are consequently robust even for cyclic networks, explicit cycles being eliminated by the computation of total constraints.

2. *Numerical tools are required only for evaluating individual equations:* the numerical handling of simultaneous set of constraints is replaced by logical operators on 2^k-trees. The only numerical tools needed are those used for determining the colors of nodes during the 2^k-trees construction.

3. *Distinguishing multiple k-consistent solutions does not increase the complexity* of 2^k-tree based consistency techniques: This is not the case using standard interval-based propagation techniques where each interval split generates a new sub-CSP to be checked for consistency separately.

4. *The precision of results can be improved deterministically:* results are refined by increasing the resolution parameter of the 2^k-tree decomposition and the accuracy does not depend upon convergence guarantees, as it is often the case when using standard numerical-based techniques.

These features provide a basis for designing k-consistency algorithms which are reliable in terms of stability and convergence, a question marginally addressed in the literature so far.

The 2^k-tree decomposition of a continuous relation imposes a uniform precision on all the variables it involves. Using this representation it is therefore sometimes necessary to

pursue the decomposition process over certain axes of a relation more finely than it is theoretically necessary. It would be interesting to study whether alternative structures (such as k^d-trees), enabling a distinct precision for each individual variable, lend themselves to an easy implementation of the logical constraint combination operators required by consistency techniques.

Note finally that the way in which 2^k-tree representation groups large groups of similar nodes into singles ones is similar to the cross-product representation of discrete constraints proposed by Hubbe and Freuder (Hubbe and Freuder, 1992). Both representations are interesting because of their ability to focus on aggregated subsets of data rather than on individual elements.

5. Reaching global consistency using convexity

The constraint representation and consistency algorithms we have described so far in principle allow computing labelings of any degree of consistency, but in the worst case would still require exponential time to do so. In this second part of the paper, we identify properties of constraints which guarantee that global consistency is equivalent to some degree of local consistency, thus allowing efficient algorithms. We first recall some results developed for convex discrete and temporal constraints and show their validity for binary convex continuous CSPs. Section 5.2 shows that a certain partial convexity feature, axis-convexity, is sufficient for efficient computation of global consistency. Finally, in Sections 5.3 and 5.4, we show that, contrary to the case of discrete constraints, these results are tractably generalizable to n-ary constraints $(n > 2)$ in continuous domains. A new concept of consistency called $(r, r-1)$-relational consistency is introduced that leads to polynomial time algorithms for computing globally consistent labelings for n-ary CCSPs.

5.1. Exploiting convexity for binary constraints

For *binary* constraint networks, it has been observed by Dechter, Meiri & Pearl (Dechter *et al.*, 1990) and van Beek (van Beek, 1992) that when constraints are *convex*, a path-consistent labelling is also globally consistent. This result follows from Helly's theorem (illustrated in Figure 7):

THEOREM 1 (HELLY) *Let F be a finite family of at least $n+1$ convex sets in R^n such that every $n+1$ sets in F have a point in common. Then all the sets have a point in common.*

Helly's theorem can be applied to show that for any consistent assignment of n variables $x_1, x_2, ..., x_n$, there exists a consistent value which can be assigned to x_{n+1}. A partial assignment $x_1 = v_1, ..., x_n = v_n$ restricts x_{n+1} through the n constraints $c(x_1, x_{n+1}), ..., c(x_n, x_{n+1})$. Since they are convex, each of these constraints will restrict x_{n+1} to a single interval. Now consider a triple $(x_i, x_j, x_{n+1}), i, j \in 1..n$. Path consistency means that x_i, x_j can only be assigned values such that there is at least one value of x_{n+1} consistent with both. This means that every pair of intervals restricting x_{n+1} intersect. Thus, by Helly's theorem, there must exist a common intersection of all the intervals, i.e. at least one value for x_{n+1} which

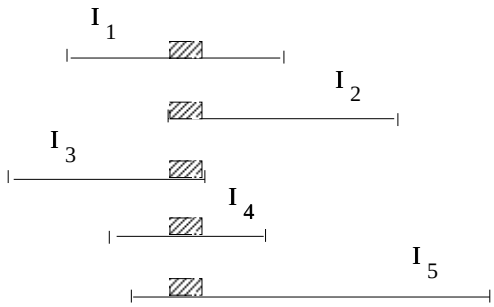


Figure 7. Helly's theorem in one dimension: if each interval is convex and if each pair of intervals has a non-null intersection, then the whole set of intervals has a non-null common intersection (shaded interval)

is consistent with all previous assignments, and consequently the assignment can be extended. We thus have the theorem:

THEOREM 2 *A binary constraint network which is convex and path-consistent is minimal and decomposable.*

The convexity property is conservative with respect to composition, intersection and projection of constraints. Hence, if a CCSP is convex its path-consistent counterpart remains so. The following a priori result can therefore be stated for convex CCSPs:

COROLLARY 1 *Path-consistency computes the minimal and decomposable network representation of any convex binary CCSP.*

Because of the difficulties of representing continuous constraints for higher order consistency algorithms, the convexity property has so far only been applied to temporal reasoning, where constraints are *unit difference constraints* of the form $x_i - x_j \leq c$. This class of constraints is closed under intersection and composition, so that path-consistency algorithms can be implemented directly on this representation. The representation techniques based on 2^k-trees we gave earlier allow implementing path-consistency algorithms for continuous constraints of arbitrary form. Another problem is that it may be difficult to check that constraints are in fact convex. (Sam-Haroud, 1995) shows a simple algorithm for reliably checking convexity of regions represented by 2^k-trees.

5.2. Partially convex binary CCSPs

Constraint convexity is a rather strong condition, but it turns out that weaker forms of convexity are often sufficient to satisfy the conditions of a globally consistent labeling. In fact, convexity has to hold only in the direction from x_i, $i \in 1..n$ to x_{n+1}. Calling this one-directional convexity *row-convexity*, Van Beek has shown the following Theorem for discrete domains (van Beek, 1992):

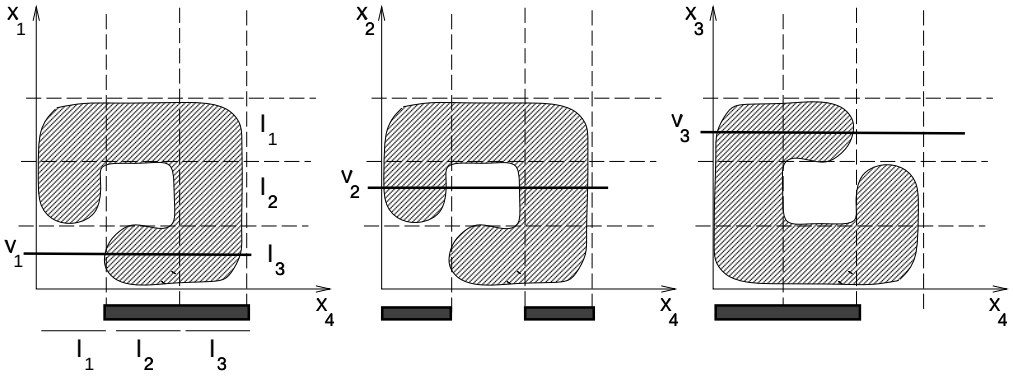


Figure 8. Path-consistency is not always sufficient for determining inconsistencies in simply connected solution spaces

THEOREM 3 (VAN BEEK) *Let R be a path-consistent binary constraint network. If there exists an ordering of the domains $D_1 \ldots D_n$ of R such that the relations are row convex, the network is minimal and globally consistent.*

where row-convexity might require reordering of the values within the domains of each variable.

Since the projection of any connected constraint region onto an axis yields a single convex interval, one might think that simple connectedness of the consistent region admitted by a constraint would be sufficient for row-convexity. However, the example of Figure 8 shows an example where this is not true: it is path-consistent, all constraints are connected, but the problem has no solution. Choosing $x_1 = v_1$, $x_2 = v_2$, $x_3 = v_3$ restricts x_4 to the intersection of the three intervals shown in Figure 8. However, these intervals do not have a single intersection, since the one in the center is not convex and thus does not satisfy the conditions of Helly's theorem.

The correct definition of row-convexity for the continuous case, which we call x-convexity, has to ensure convexity of any subprojection of a constraint:

Definition. axis-convexity
Let R be a binary relation defined by a set of algebraic or transcendental constraints on variables $x_1, x_2, \ldots, x_i$. R is said to be *axis-convex* with respect to x_k or (x_k)-convex if, for any two points q_1 and q_2 in R such that the segment $\overline{q_1 q_2}$ is parallel to x_k, $\overline{q_1 q_2}$ is entirely contained in the region delimited by R.

Note that axis-convexity is less restrictive than convexity: convexity allows q_1 and q_2 to be chosen with no restriction while axis-convexity requires choosing q_1 and q_2 within the set of points belonging to lines parallel to the x axis. A convex region is necessarily axis-convex but the converse is not true. A network is said to be axis-convex if all of its relations are axis-convex.

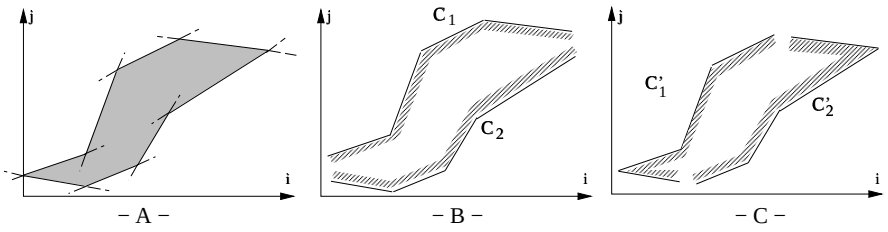


Figure 9. Figure -A- shows the polygonal solution space of a linear problem which is both (i)- and (j)-convex but not convex: it is monotone with respect to i and j since its boundaries can be decomposed into two monotone chains C_1 and C_2 (-b-), monotone with respect to i, or C_1' and C_2' (-c-), monotone with respect to j.

LEMMA 2 *Let R be a relation defined by a set of algebraic or transcendental constraints. R is (x_k)-convex if and only if any subprojection of R over the x_k axis yields a single convex interval.*

Proof: Apply Definition with q_1 and q_2 being the intersection points of the constraint boundaries with the plane identifying the subprojection.

Monotone and functional relations are axis-convex

The (x)-convexity property shares important similarities with the notion of monotonicity as defined in Euclidean geometry. For polygonal objects, we have the following definitions. A chain is defined as a set of vertices connected with segments of lines. It is said to be monotone with respect to a straight line l if a line orthogonal to l intersects C in exactly one point. A simple polygon is said to be monotone if its boundaries can be decomposed into two chains monotone with respect to the same line. These definitions can be extended to general curves as follows: a curve C of $\Re^2$ will be said to be monotone with respect to a straight line l, if a line orthogonal to l intersects C in exactly one point, and a region r of $\Re^2$ will be said monotone if its boundaries can be decomposed into two curves monotone with respect to the same line. This generalized definition of monotonicity is equivalent to axis-convexity. A linear problem for which the solution space is a monotone polygon with respect to x is consequently axis-convex with respect to any axis orthogonal to x. Note that even if axis-convexity holds for both axes, it is still weaker than convexity (see Figure 9). Note that testing for the monotonicity of a polygon with respect to a line can be done in $O(N)$ time, where N is the number of edges.

As another example, the solution spaces defined by functions also satisfy axis-convexity properties: the curve defined by a function $x_1 = f(x_2)$, functional in x_2, is by definition monotone with respect to x_2 and (x_1)-convex.

Directional axis-convexity

In the case where a discrete network does not satisfy the row-convexity property, van Beek (van Beek, 1992) shows that *directional* row-convexity is already sufficient for backtrack-free solutions. In directional row-convexity, a relation must be row-convex only with respect to not yet instantiated variables.

Similar results generalize to the case of axis-convex relations. The following theorem states that directional axis-convexity of the network is sufficient to ensure that a solution can be determined without backtracking.

THEOREM 4 *Let N be a path-consistent binary constraint network. If there exists an ordering of the variables $x_1, \ldots x_n$ such that each relation of N R_{x_i,x_j}, $1 \leq j \leq i$, is (x_i)-convex, then a consistent instantiation can be found without backtracking.*

Proof: We denote as $\prod_{v_j}^{i} R_{ij}$ the projection of the slice of relation R_{ij} at $x_j = v_j$ over the x_i axis. The backtrack-free instantiation algorithm proposed by van Beek (algorithm *instantiate* in (van Beek, 1992)) generalizes as follows:

1. Choose a value v_1 of x_1 that satisfies R_{11}
2. For $i \leftarrow 2$ to n do
3. $\quad v_i \leftarrow$ choose a value in $\bigcap_{j=1\ldots i-1}(\prod_{v_j}^{i} R_{ij})$

where $\bigcap$ denotes interval intersection. Since all relations are axis-convex with respect to all non-instantiated variables, the constraint they impose on any variable to be instantiated in step 3 are single intervals. By path-consistency, all pairs of these intervals intersect and so Helly's theorem guarantees at least one non-empty interval within which the new value can be chosen. This provides a constructive proof of the theorem.

Directional row-convexity imposes ordering conditions on both variables and variable domains. Since discrete CSPs do not have the strictly ordered domains characterizing continuous CSPs, the fact that a constraint network is row-convex can sometimes be hidden. This is obviously not the case for continuous CSPs concerning axis-convexity.

5.3. Convex ternary CCSPs

As shown earlier, N-ary CCSPs can be translated into ternary ones without loss of information, so we will only treat the case of ternary constraints here. The generalization of the results on axis-convexity to the case of ternary networks seems straightforward a priori. Similarly to the case of binary constraints, a relation R between variables x_1, x_2 and x_3 is called (x_k)-convex if, for any two points q_1 and q_2 in R such that the segment $\overline{q_1 q_2}$ is parallel to the x_k-axis, $\overline{q_1 q_2}$ is entirely contained in R.

In the case of ternary constraints, Helly's theorem can be used to prove the decomposability of the constraint network only if each pair of ternary relations have a non-null (x)-intersection. Indeed, building a proof analogous to the one used for demonstrating Theorem 2, requires proving strong k-consistency of the network for each k. Given that the network is strongly (k-1)-consistent, the following assertion must therefore be satisfied:

$$(\forall (i,j) \in \{1, \ldots k-1\}), R(X_k, X_i, X_j) \tag{5}$$

Verifying assertion(5) amounts to showing that the interval on x_k, $X_k^1 = \prod_{x_k} R(x_k, X_{i_1}, X_{j_1})$, intersects the interval on x_k, $X_k^2 = \prod_{x_k} R(x_k, X_{i_2}, X_{j_2})$, for any combination of $i_1, i_2, j_1,$

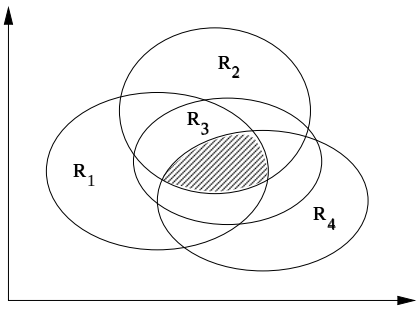


Figure 10. Helly's theorem in $\Re^2$: if a finite set of binary convex regions is such that each triplet of regions has a non-null intersection, then the whole set of regions has a non-null common intersection (shaded area)

and j_2 in $\{1 \ldots k-1\}$. Each subset of *five* variables $(x_{i_1}, x_{i_2}, x_{j_1}, x_{j_2}, x_k)$ has therefore to be consistent. This requirement corresponds to the definition of 5-consistency. Hence, a possible generalization of Theorem 2 is as follows:

THEOREM 5 *A ternary constraint network which is axis-convex and strongly 5-consistent is minimal and decomposable.*

Since all relations in a convex network are also axis-convex, a similar result hold for convex problems. Unfortunately, even if theoretically correct, this result has no significant practical impact. In analogy to the case of discrete row convex problems, enforcing 5-consistency may create additional constraints of arity *four*. This would mean that the constraint network is no longer ternary and would require even higher degrees of consistency. To overcome this limitation for discrete CSPs, van Beek and Dechter (van Beek and Dechter, 1995) have introduced the notion of relational path-consistency for discrete problems.

A relationally path-consistent network (of arbitrary arity) is characterized by the condition that every pair of relations having at least one variable in common is nonempty. By definition, relational path-consistency guarantees for each set of relations having a variable x in common that the pairwise intersections of their unary projections over the x axis is non-empty. Helly's theorem becomes thereby applicable, which results in the following theorem:

THEOREM 6 (VANBEEK & DECHTER) *Let R be a network of relations that is relationally path-consistent. If there exists an ordering of the domains $D_1 \ldots D_n$ of R such that the relations are row convex, the network is globally consistent.*

The problem with this approach is that enforcing relational path-consistency may pose complexity problems. In the case of ternary networks for example, composing *pairs* of ternary relations (with a variable in common) results in a relation of arity five. Relational path-consistency aims at ensuring extensibility properties of an arbitrary subset of variables to a third *single* variable (unary projection). Since arity five is inevitable for ternary constraints, it becomes necessary to guarantee that a set of four variables is extensible to a fifth one, so that the relation between the five variables (issued from extended-composition) is

```
Procedure 3-2-rel-con(V,C,D)
   repeat
   changed ← false
   for each pair (u,v), u,v ∈ V do
      for each ternary tuple (i,j,k), i,j,k ∈ V do
        begin
        c'(i,j,k) ← c(i,j,k) ⊕ ∏_(i,j,k) c(i,u,v) ⊗ c(j,u,v) ⊗
c(k,u,v)
        if c'(i,j,k) ≠ c(i,j,k) then
          begin
          c(i,j,k) ← c'(i,j,k)
          changed ← true
          end
        end
   until changed = false
```

Figure 11. Algorithm for computing a (3-2)-relationally consistent labeling.

not empty. This means that relational path-consistency must also be ensured for relations of arity four —and recurrently, for relations of arbitrary arity. This can result in an intractable complexity in the most general case.

We show in (Sam-Haroud, 1995) that a tractable alternative generalization can be proposed in continuous domains. This generalization is based on the observation that the extensibility of a ternary set of variables to a *binary* region (rather than to an *unary* one like proposed in van Beek and Dechter approach) does not involve relations with arity greater than 3 and thus removes the causes for this combinatorial explosion. This approach implies that Helly's theorem must be applied in two dimensions rather than one (see Figure 10).

For the case of ternary networks, we introduce the notion of (3,2)-relational consistency which guarantees that each *triplet of relations*, having *two variables in common*, has a non-null intersection.

Definition. Let N be a ternary network of relations over a set of variables X. Let $R_{I_1}(x_1, u, v)$, $R_{I_2}(x_2, u, v)$ and $R_{I_3}(x_3, u, v)$ be three relations of N which share two variables u and v, where u might be identical to v. R_{I_1}, R_{I_2} and R_{I_3} are (3,2)-relationally consistent relative to $\{u, v\}$ if and only if any consistent instantiation of the 3 variables in x_1, x_2, x_3 has an extension to $\{u, v\}$ that satisfies R_{I_1}, R_{I_2} and R_{I_3} simultaneously.

Since (3,2)-relational consistency only imposes constraints between at most three variables, a (3,2)-relationally consistent labelling still contains only ternary labels and thus does not add to the arity of a ternary constraint network. Provided that each binary projection is convex, (3,2)-relational consistency enables the application of Helly's theorem in two dimensions. However, Helly's theorem only guarantees in these conditions that each pair of variables (binary projections) has a non-empty domain. We prove in (Sam-Haroud, 1995) that this is sufficient to also guarantee that the constraints involving *individual* variables will be non-empty.

(3-2)-relational consistency can be computed by the simple algorithm shown in Figure 11. It takes as input a continuous CSP, $\mathcal{P} = (V, C, D)$, where V is the set of variables, C the set of constraints, and D is the set of variable domains. c denotes the label of a relation in C, and the algorithm terminates with a (3-2)-relational consistent set of labels.

Using Helly's theorem in two dimensions, in (Sam-Haroud, 1995) we prove the following result:

THEOREM 7 *For any convex ternary network $\mathcal{P}$, (3,2)-relational-consistency will either:*

- *decide that the network is inconsistent by generating an empty label, or*
- *compute an equivalent globally consistent labelling of $\mathcal{P}$*

in time $O(n^5)$ where n is the number of variables of $\mathcal{P}$.

Sketch of proof.

We first prove that when applied to a ternary network $\mathcal{P}$, the algorithm for (3,2)-relational-consistency results in an empty network if a given pair of variables has an empty label. Next, in order to show that non-empty labels on each pair of variables imply global consistency, we introduce a *dual* network $\mathcal{P}_d(V_d, C_d, D_d)$ such that:

- $V_d = \{\alpha_1, \ldots \alpha_m\}$. A variable of V_d, α_j, represents a pair of variables in the original network (an element $(x_{j,1}, x_{j,2})$ of V^2 so that $x_{(j,1)} \neq x_{(j,2)}$)
- a domain in D_d is a combination of two domains in D
- a relation between two variables, α_u and α_v, of V_d is the relation $R(x_{(u,1)}, x_{(u,2)}, x_{(v,1)}, x_{(v,2)})$ resulting from the intersection of the relations between $x_{(u,1)}, x_{(u,2)}, x_{(v,1)}, x_{(v,2)}$ in the original problem $\mathcal{P}$.

The original and dual networks are shown to be equivalent (each solution of the primal corresponds to a solution of the dual network and vice versa). (3,2)-relational consistency of the original network $\mathcal{P}$ is then shown to imply 4-consistency of the dual network $\mathcal{P}_d$. However, the dual network is binary. Since its variables are points in two dimensions, for proving extensibility we have to apply Helly's theorem in *two dimensions*: since $\mathcal{P}_d$ is 4-consistent, the constraints imposed by any triple of already instantiated variables intersect and thus there is a non-empty common intersection. Since this guarantees that any partial solution is extensible, $\mathcal{P}_d$ is globally consistent. Since it is equivalent to $\mathcal{P}$, the original problem must also be globally consistent.

□

5.4. Generalization of (x)-convexity

Directional (x,y)-convexity

So far, we have assumed full convexity of the constraint network. We now examine how, analogously to the case of binary constraints, a less restrictive convexity condition can be defined for n-ary constraints. We first propose the following generalization of the (x)-convexity property:

Definition. $(x_1, \ldots x_k)$**-Convexity**
Let R be an n-ary relation between n variables $x_1 \ldots x_n$. R is said to be $(x_1, \ldots x_k)$-convex in the domains $D_1 \times \ldots D_{x_k}$ if, for any two points q_1 and q_2 in the region r defined by R, such that the segment $\overline{q_1 q_2}$ is on a plane parallel to $x_1 \ldots \times x_k$, $\overline{q_1 q_2}$ is entirely contained in r

Informally, this means that a relation is $(x_1, \ldots x_k)$-convex if any subprojection over the subset $(x_1, \ldots x_k)$ yields a convex k-ary region.

In the proof of Theorem 7, convexity of the constraints is used only in the application of Helly's theorem to ensure extensibility of partial solutions of the dual network. Here, it would be sufficient to have convexity hold only in two of the three dimensions involved in the constraint. Thus, we have the following theorem:

THEOREM 8 *Let $\mathcal{N}$ be a (3,2)-relationally consistent ternary constraint network. If there exists an ordering of the variables $x_1, \ldots x_n$ such that for any $i, j, k : 1 \leq i < j \leq k \leq n$, $R(x_i, x_j, x_k)$ is (x_j, x_k)-convex, then the network is globally consistent and a consistent instantiation can be found without backtracking.*

Sketch of proof.

According to Helly's theorem in two dimensions, the fact that each ternary relation $R(x_i, x_j, x_k)$ is (x_j, x_k)-convex guarantees that the binary relations $R(x_j, x_k)$ derived from the problem are non-empty ($R(x_j, x_k) = \bigcap_{j:i..k} \prod_{x_i} R(x_i, x_j, x_k)$). By construction, these binary relations are convex and have non-null pairwise intersections. Consequently, a similar argument as the one given for the proof of theorem 2 hold and instantiation can be carried out backtrack-free □.

Generalization to general r-ary CCSP

In the case of networks of arity r, the composition of two maximal arity constraints having at least one variable in common, results in a relation of arity $2r - 1$. In analogy to the case of ternary networks, we observe that the extension of an r-ary set of variables to a region of arity $r - 1$ does not involve relations with arity greater than r. To apply Helly's theorem, we must introduce the notion of (r,r-1)-relational consistency which guarantees that each set of r relations having $r - 1$ variables in common has a non-null intersection:

Definition. Let $\mathcal{P}$ be a network of relations over a set of variables X, of arity r. Let $R_{I_1}(x_1, y_1, \ldots, y_{r-1}), \ldots R_{I_r}(x_r, y_1, \ldots, y_{r-1})$ be r relations of N sharing the r-1 variables $\{y_1, \ldots y_{r-1}\}$. The relations are (r,r-1)-relationally consistent relative to the shared variables if and only if any consistent instantiation of the variables in $\{x_1, \ldots, x_r\}$ has an extension to $\{y_1, \ldots y_{r-1}\}$ that satisfies all relations simultaneously. The network $\mathcal{P}$ is relationally (r,r-1) consistent if and only if all relations are (r,r-1)-consistent with respect to all subsets of shared variables.

The following result can then be proven (see (Sam-Haroud, 1995)):

THEOREM 9 *Let $\mathcal{P}$ be a constraint network of arity r at most, $(x_1, \ldots x_{r-1})$-convex. If $\mathcal{P}$ is (r,r-1)-relationally consistent then it is globally consistent.*

6. Relevance for practical applications

We now discuss the relevance of the results of section 5 for solving practical CSPs in continuous domains. We first discuss, in terms of topological properties of the solution space, the theoretical complexity of solving CCSPs. We then present how these complexity issues translate when constraints are represented by 2^k-trees.

6.1. A classification of CCSPs

A general CCSP may admit no convex or axis-convex constraint network representation. Moreover, even if the initial problem is axis-convex, consistency algorithms may not preserve this property since intersecting two non-convex regions may results in an arbitrary number of distinct sub-regions. For these reasons, the result of Theorems 4, 8 is valid only on the a posteriori networks computed by consistency algorithms. Concerning the a priori networks, similar results can be formulated only for the classes of problems where axis-convexity is closed under composition and intersection of constraints. In order to bound a priori the complexity of solving CCSPs, it is desirable to identify the classes of problems for which the axis-convexity property is closed under intersection and composition.

We can distinguish three classes of CCSPs, as follows:

i. CCSPs where all the relations determine convex regions

ii. CCSP where each relation determining a non-path connected solution space consists of a set of convex regions.

iii. CCSPs where there exist non-convex regions

In problems of class *i*, since the intersection of two convex regions is necessarily convex (and hence axis-convex), consistency algorithms will preserve the convexity of the constraint network representation. Hence, problems of this first category can be solved efficiently using path consistency followed by backtrack-free instantiation. Problems of class ii require an additional search procedure but in general are still solvable efficiently. Problems of class iii can cause an explosion of the number of disjoint solution regions and remain difficult to solve.

Search

A multi-convex CCSP (type *ii*) can be decomposed into convex sub-problems, one for each possible combination of convex sub-region. We show in (Sam-Haroud, 1995) that solving a multi-convex CCSP is NP-Hard in the number of disjoint convex regions. However, if the task consists of finding any globally consistent region of the solution space, backtracking algorithms can be proposed as alternatives to the brute force enumeration process. Similarly to the case of temporal problems (Dechter *et al.*, 1990), a multi-convex CCSP, C, can be translated into a discrete CSP where the variables correspond to the edges of C and the values for the variables are the 2^k-tree approximations of the relation convex subregions. The assignment $(X_1 = Tree_1, ..X_i = Tree_i)$ is consistent if and only if the corresponding convex CCSP is consistent. Since the complexity of solving convex

subproblems is bounded (and polynomial), more sophisticated search schemes can be designed that incorporate local consistency checking with search.

Splitting

In problems of type iii, the splitting problem (similar to the one described in (Hyvönen, 1992) may occur and the complexity is difficult to estimate. In the best case, the consistency algorithm may create a convex constraint network from a set of non-convex relations. In the worst case however, the intersection of each pair of non-convex regions may result in an unbounded number of disjoint new sub-regions which can in turn split again. Practical solutions (such as stopping the splitting process when the maximum precision is reached) can be used to bound the combinatorial explosion, but in general the complexity remains high for CCSPs of type *iii*.

6.2. 2^k-trees and convexity

While the 2^k-tree representation of constraints presents many advantages in terms of expressiveness, simplicity and reliability, we have not yet considered the crucial issue of whether this representation preserves the topological properties of the original solution space (connectivity, convexity etc…). Since the 2^k-tree decomposition generates stepwise approxi-

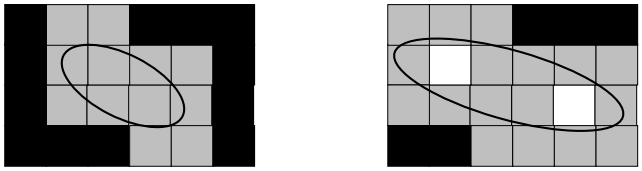


Figure 12. When the resolution in insufficient, a connected solution region can yield an empty (-A-) or a disconnected (-B-) $\mathcal{I}(\mathcal{S})$ representation.

mations of the boundaries, convexity is obviously not preserved in the strict mathematical sense. Moreover, as shown in Figure 12, when the resolution chosen is insufficient, situations may occur where a connected solution space is represented by disconnected or even empty 2^k-tree regions.

We show in (Sam-Haroud, 1995) that the 2^k-tree representation of any solution space $\mathcal{S}$:

- has a connected and nonempty $\mathcal{I}(\mathcal{S})$ representation if it its smallest diameter is at least two times the limit of resolution,
- can yield a disconnected $\mathcal{I}(\mathcal{S})$ representation only in the case where its smallest diameter is less than two times the limit of resolution. Moreover, a single additional level of decomposition is then sufficient to make the representation connected again.

Hence, the only risk that presents itself when running the propagation algorithms using the 2^k-tree representation of convex constraints is the risk of disconnection. It occurs when

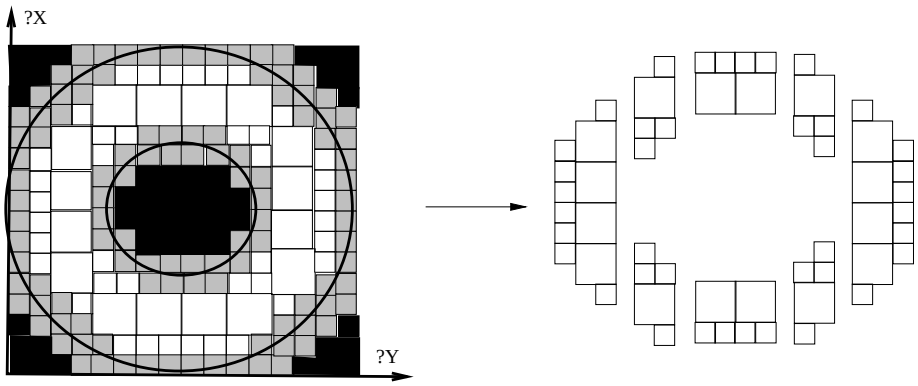


*Figure 13. A non-convex quadtree (***left***) decomposed into eight convex sub-quadtrees (***right***)*

the solution of the CCSP falls within the limit of resolution chosen for the 2^k-tree representation and is therefore restricted to limit cases. In the marginal cases where a disconnection occurs, it is then possible either to resort to further refinements of the quadtrees or to neglect the solution region within the disconnected area—considering that its identification requires a precision having no significance for the application. The problem will then be stated as non-convex.

Solving non-convex problems

For a fixed precision, we show in (Sam-Haroud, 1995) that 2^k-trees for a general CCSP can always be decomposed into convex ones (see Figure 13) and the problem can then be solved by search techniques, as those sketched out for multi-convex CSPs. However, in this case problems with insufficient resolution limits cannot be fixed by simply increasing the resolution, since a higher resolution might require a different subdivision. Thus, the resolution complexity after decomposition can only be bounded when the precision required is fixed a priori. This condition holds for almost all the engineering applications manipulating physical entities, making it possible to apply the methods we have described to almost any practical engineering problem.

Checking for convexity and axis-convexity

When constraints are approximated using quadtrees, we show in (Sam-Haroud, 1995) that the axis-convexity property can checked for in $O(Nlog_4(N)+2^{D_x/\varepsilon})$ where N is the number of feasible nodes, D_x is the domain size of variable x (i.e. interval length) and ε the minimal interval length of x in the quadtree decomposition. Similarly, convexity can be checked for in $O(2Nlog_4(N)+2^{\frac{D}{\varepsilon}+1})$ where N is the number of feasibility nodes, D is the maximal domain size in the quadtree (i.e. interval length) and ε the minimal interval length in the quadtree decomposition (for a fixed precision, this complexity is $O(Nlog_4(N))$. Similarly, we show in (Sam-Haroud, 1995) that the (x_1, x_2)-convexity property, useful for solving ternary problems, can be checked for in $O(N + Nlog_4(N))$, for a fixed precision.

Directional axis-convexity

In (van Beek and Dechter, 1995), van Beek and Dechter propose an algorithm for determining a directional row-convex ordering for discrete problems. This algorithm can be directly mapped into an equivalent one for determining an axis-convex ordering. We propose in (Sam-Haroud, 1995) a variant of van Beek and Dechter's ordering algorithm having a complexity of $O((n \cdot Nlog_4(N))$ where n is the number of variables and N is the number of feasible nodes in the quadtree of maximal size. In the case of ternary constraints, the appropriate variable ordering can be derived similarly to the case of axis-convexity with a complexity of $O(n \cdot (N + Nlog_4(N)))$ for a fixed precision (see (Sam-Haroud, 1995)).

Comparison with the discrete case — 2^k-trees and backtrack-free search:

It is worth mentioning that the results on n-ary constraints (see section 5.3) are not directly transferable to discrete domains. Ensuring backtrack-free search in ternary constraints requires convexity conditions to hold in $\Re^2$ rather that $\Re$. We have shown that (3,2)-relational consistency is equivalent to global consistency, but the backtrack-free instantiation might require refining the resolution of different variables. This is possible in continuous domains, but not possible if we use a continuous domain to represent a discrete problem.

Consider the following example where two matrices representing discrete relations are understood as showing convex solution regions. When we intersect the two regions, the result is:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \oplus \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

i.e. the intersection has been lost as it is smaller than the resolution limit. In a continuous problem, we can now refine the resolution to make this problem go away, and continue with the instantiation. But in a discrete problem, we do not have this possibility as the maximum resolution is fixed. A path-consistent labeling does not guarantee that we can in fact successfully complete a backtrack-free instantiation without need for increasing the resolution, and hence does not guarantee global consistency in a discrete problem where this possibility does not exist. Thus, our results can unfortunately not improve on the complexity of discrete convex CSPs.

7. A practical example

We now sketch out how the introductory example of Figure 1 is solved using our method. In this example, four main independent variables, beam depth(H_b), slab thickness(H_s), beam span(W) and beam spacing(S) are linked together through the following non-linear constraints:

$$\begin{aligned} H_s &> 137.70 - 8633.10^{-3}S + 5511.10^{-8}S^2 - 835810^{-10}S^3 \\ H_b &> 41.383 \left(\frac{1.1 * 2.35 * p * W^2}{10^6 C * f_y} \right)^{0.3976} \end{aligned}$$

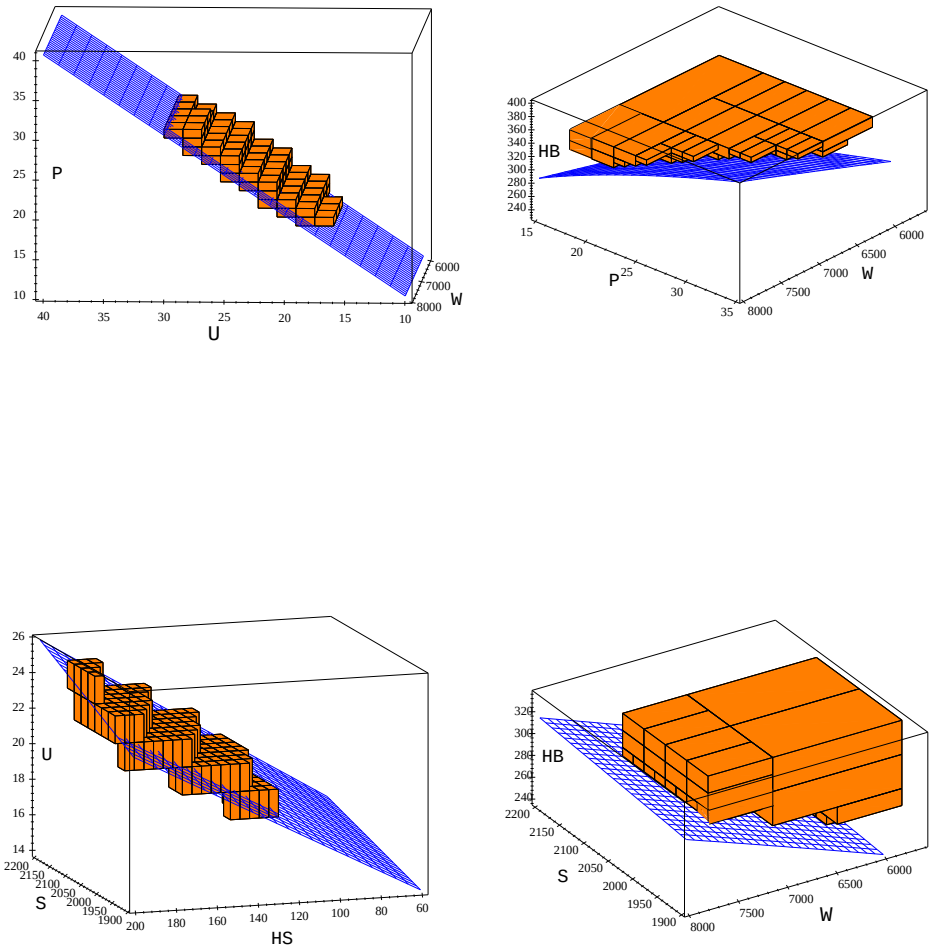


Figure 14. Some octrees constructed for the steel structure problem

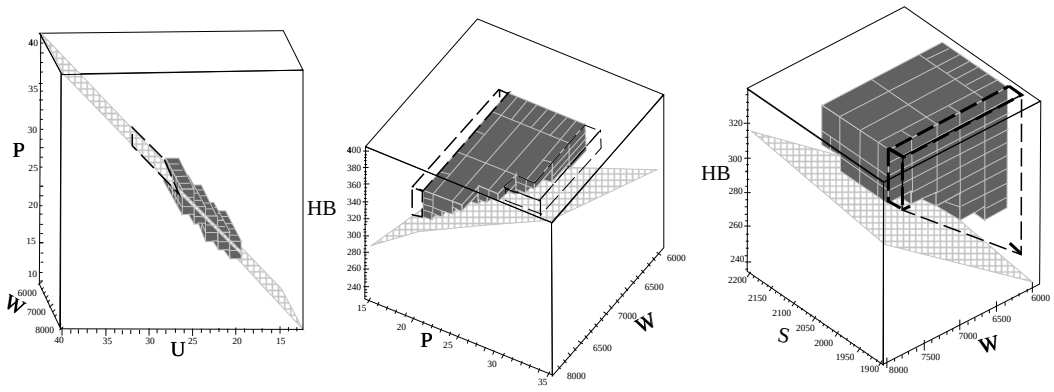


Figure 15. Three constraints from the steel structure problem. The areas within the dashed lines are those removed while enforcing global consistency

$$p = 1.3\left[\frac{(P_{sw(slab)} + P_{dl}) * S}{1000} + 0.000074\left(\frac{W}{24}\right)^{1.5161}\right]$$

$$H_b > \left[\frac{P_{llst} * S}{1000} \cdot \frac{D * 350 * W^3}{0.1545 * 384 * E_s}\right]^{0.2839}$$

P_{llst} is the short term live load which equals to $4kN/m^2$ and E_s equals to $210000N/mm^2$ in this example. Traditionally, an engineer works through these equations hierarchically; at no time is the complete solution set known. Exploration of possible solutions is done point by point according to the experience of the designer. Using our method, a large space of potential solutions can be made explicit for the first time.

We start by transforming the original problem into a ternary equivalent one:

$$\begin{aligned} u &< (3.18.10^{-5}H_s + 0.0054)S \\ H_s &> 137.70 - 8633.10^{-3}S + 5511.10^{-8}S^2 - 835810^{-10}S^3 \\ p &= u + 9.62.10^{-5}(0.0417W)^{1.5161} \\ H_b &> 0.77(p.W^2)^{0.3976} \\ H_b &> 0.0168(s.W^3)^{0.2839} \end{aligned}$$

Figure 14 shows the octree representation constructed for each relation. Figure 15 shows a set of constraints derived from the problem after global relaxation: the areas within the dashed lines are those removed by (3,2)-relational consistency. The (3,2)-relationally consistent octrees are not convex but (s, w, h_b, p, h_s, u) is identified as a directional (x,y)-convex ordering. The solutions of the problem can therefore be derived backtrack-free, which might produce for example the following very different instantiations:

```
solution 1:
-----------
?S = [1959.38 , 1981.25],
?W = [5953.12 , 6281.25],
?HB = [267.19 , 275.0],
?P = [18.28 , 19.38],
?HS = [120.31 , 125.0],
?U = (17.81 , 22.5)

solution 2:
-----------
?S = [1981.25 , 2003.12],
?W = [5953.12 , 6281.25],
?HB = [267.19 , 275.0],
?P = [18.28 , 19.38],
?HS = [120.31 , 125.0],
?U = [17.81, 22.5]
.................
```

This shows that the problem admits in fact a large space of potential solutions, of which current mathematical methods only find a single one at random. The new tool offers many new possibilities for optimization which have not been explored before.

A prototype Lisp implementation calculates the octrees and enforces global consistency for this problem in approximately 1800 seconds on a Silicon Graphics Indigo with an R4000 processor.

8. Conclusions

Our work has been motivated by the needs of practical engineering problems, where decision-making can benefit greatly from knowledge of the globally consistent ranges of variables. It is surprising that among the many tools available for solving numerical constraints, very few are capable of addressing this problem in a practically useful way. In this paper, we have shown two major results, namely that:

- an appropriate representation of numerical constraints allows reliable algorithms for higher degree of consistency than arc-consistency
- general conditions can be identified under which global consistency is tractable for continuous constraint satisfaction problems.

This has allowed solutions to many practical constraint satisfaction problems which are much appreciated by engineers.

There exist a myriad of methods for constraint solving, ranging from mathematical and operations research techniques to interval arithmetic and local consistency methods. Due to the difficulty of dealing with arbitrary numerical constraints, it is not surprising that none of these techniques provides satisfactory solutions in all cases: some methods are designed to respond efficiently to the needs of particular problems, others are more general but less

performing. The method we proposed is very general, but also makes a restrictive assumption. It is guaranteed to solve *any* continuous constraint satisfaction problem as long as an acceptable resolution limit can be specified. This restriction is very different from others, but quite readily satisfied in many practical problems.

Note that while the polynomial complexity bounds are valid only under certain convexity conditions, the algorithm is guaranteed to terminate with the correct solution no matter what the form of the constraints are. This is a significant improvement over traditional interval-based approximation techniques which handle constraints using iterative numerical methods are inevitably confronted with convergence and stability problems which make termination impossible to guarantee. Discretization is the price paid for this guarantee, but we believe it is a often a minimal price.

9. Acknowledgment

We would like to thank Sylvie Boulanger (Steel Structure lab., EPFL) for interesting discussions related to civil engineering, Peter van Beek (University of Alberta, CA), and Gaston Gonnet (ETHZ) for many helpful comments on this work, as well as the Swiss National Science Foundation for sponsoring this research under contract No.5003-034269. We also thank several anonymous reviewers for very helpful comments which led to significant improvements to this paper.

References

Benhamou, F., McAllester D. and van Hentenryck P., "CLP(Intervals) revisited," Proceedings of the International Logic Programming Symposium 94, 1994.

Benhamou, F. and Older W., "Applying interval arithmetic to real integer and boolean constraints," Journal of logic programming, 1994.

Cooper, M.C., "An optimal k-consistency algorithm," Artificial Intelligence, 41, 1989.

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

Dechter, R., "From local to global consistency," Proceedings of the 8th Canadian Conference on AI, 1990.

Dechter, R., Meiri I., and Pearl J., "Temporal constraint networks," Artificial Intelligence, 49, 1990.

Faltings, B., "Arc-consistency for continuous variables," Artificial Intelligence, 65(2), 1994.

Faltings, B., Haroud D., and Smith I., "Dynamic constraint propagation with continuous variables," Proceedings of the 10th European Conference on AI, 1992.

Freuder, E.C., "Synthesizing constraint expressions," Communications of the ACM, 21, 1978.

Freuder, E.C., "A sufficient condition for backtrack-free search," Journal of the ACM, 29, 1982.

Haroud, D., and Faltings B.V., "Global consistency for continuous constraints," Lecture notes in computer Science 874: Principles and Practice of Constraint Programming, Springer Verlag, 1994.

Haroud, D., Boulanger S., Gelle E., and Smith I., "Management of conflict for preliminary engineering design tasks," AIDEAM, 9(4), 1995.

Hickey, T. "CLP(F) and constrained ODE's," Proceedings of 1994 workshop on constraints and modeling, 1994.

Hubbe, D., and Freuder E.C., "An efficient cross-product representation of the constraint satisfaction problem search space," Proceedings of the 10th National Conference on AI, 1992.

Hyvönen, E., "Constraint reasoning based on interval arithmetic: the tolerance propagation approach," Artificial Intelligence, 58, 1992.

Lee, J.H.M, and van Emden M.H., "Interval computation as deduction in CHIP," Journal of Logic programming, 16(3-4), 1993.

Lhomme, O. “Consistency techniques for numeric CSPs,” Proceedings of the 13th International Joint Conference on AI, 1993.

Mackworth, A.K., “Consistency in networks of relations,” Artificial Intelligence, 8, 1977.

Montanari, U., “Networks of constraints: fundamental properties and applications to picture processing,” Inform. Scie.,7, 1994.

Older, W. and Vellino A., “Constraint arithmetic on real intervals,” Constraint logic programming —Selected research, The MIT Press, 1993.

Sam-Haroud, D. “Constraint consistency techniques for continuous domains,”, PhD thesis, Ecole Polytechnique Fédérale de Lausanne, Switzerland, 1995.

Sidebottom, G., and Havens W.S., “Hierarchical arc-consistency for disjoint real intervals on constraint logic programming,” Computational Intelligence, 8(4), 1992.

Tanimoto, S. “A constraint decomposition method for spatio-temporal configuration problems,” Proceedings of the 11th National Conference on AI, 1993.

van Beek, P., “On the minimality and decomposability of constraint networks,” Proceedings of the 10th National Conference on AI, 1992.

van Beek, P., and Dechter R., “ On the minimality and global consistency of row-convex constraint networks,” Journal of the ACM, 1995.

van Hentenryck, P., Mc Allester D., and Kapur D., “Solving polynomial systems using a branch and prune approach,” To be published, 1995.