Constraint-based support for negotiation in collaborative design

C. Lottaza, I.F.C. Smith, Y. Robert-Nicoud, B.V. Faltings

Abstract

When constraints are used to represent engineering requirements, enhanced support for collaboration becomes possible. More specifically, if important engineering design specifications are represented as groups of inequalities on continuous variables, solving these constraints results in spaces of feasible values. Such spaces improve efficiency through avoiding artificial conflicts, improving design flexibility, enhancing change management and assisting conflict resolution. This paper describes an implementation that employs algebraic reformulation, which includes an efficient approach for transformation to ternary expressions. This approach reduces limitations related to computational complexity that were inherent in previous implementations. Important features are then illustrated using a full-scale example. Carrying out collaborative design using solution spaces (CDSS) with this new implementation called SpaceSolver. Such software is expected to enhance future collaboration tools.

Keywords: Negotiation; Collaborative design; Solution spaces; Constraint satisfaction; Algebraic reformulation

1. Introduction

Most engineering tasks require collaboration between many partners. Collaboration tasks are complicated by factors such as losses during information exchange, misunderstandings and iterative negotiation when sub-task solutions conflict. Moreover, changes in context, costs, requirements, deadlines, etc. require constant re-negotiation of issues that may modify important project characteristics.

Although communication and collaboration tools recently became more attractive with the success of the Internet, little work proposes support for maintaining project consistency. For example, many projects including Refs. [5,8,11,31,32,36] contain proposals for communication and information-management facilities such as graphical databases, shared project models, knowledge bases, bulletin boards and video conferencing. Much of this work is founded on the assumption that improved communication helps engineers carry out collaborative tasks such as negotiation.

However, improved communication techniques also present risks. It is too simple to suggest changes. The responsibility for maintaining design consistency is often taken away from the initiator of a change. Instead, recipients become solely responsible for approving changes. In complex projects, recipients typically become overwhelmed; under the volume of information they are no longer able to verify all suggestions. In some cases partners become very restrictive when confronted with new ideas in order to reduce the risk of error and inconsistency. In other cases, the integrity of the design project drops and possibilities for missed deadlines and additional costs increase.

Inconsistent construction projects are undesirable and therefore, there has been research into explicit computer support for negotiation and conflict mitigation. Beginning with work into design rationale [26], Peña-Mora has recently proposed a combination of negotiation and game theory to support negotiation between partners [27,28]. Ndumu and Tah [25] suggest a computational market model to resolve conflicts. Mokhtar et al. [24] focus on change management to provide an information model that assists in planning and scheduling design changes. None of this work proposes constraint solving or solution spaces for negotiation support, although such methods from artificial intelligence can provide interesting facilities for collaboration.

When constraint solving is used, only local consistency is usually assured. Bahler et al. [2,4] proposed a design advice tool that uses constraints to support negotiation and conflict resolution. An exception handling approach studied by Klein [16] also uses local methods for enhancing consistency. Realising the need for constraint solving in the Redux system, Petrie has enhanced the framework to include a constraint manager that would plug into remote solvers [30]. Khedro and Genesereth [15] have developed a progressive negotiation strategy for conflict resolution where...
locally consistent solutions are used to converge on global consistency. However in these studies, no explicit use of solution spaces has been found for constraints expressed in terms of continuous variables.

Currently in practice, much negotiation effort can be traced to partners that determine only one solution for individual subtasks. Integration of partial solutions often leads to artificial conflicts. Such conflicts may be less related to conflicting project goals than to premature decision-making. Decision-making is often required to specify single solutions that only demonstrate satisfaction of subtask requirements. Instead of such demonstration of feasibility, an explicit representation of many feasible solutions, a solution space for each subtask would be desirable.

Constraints provide a useful format for representing solution spaces. Constraint satisfaction techniques compute approximations of solution spaces of constraint satisfaction problems (CSPs). A CSP is defined by a set of variables, their domains (a priori possible values for each variable) and a set of constraints. In engineering, constraints are typically numerical relationships (equalities and inequalities) that influence feasible values of continuous and discrete variables. This paper focuses on constraints expressed in terms of continuous variables.

The task of the CSP is to identify variable values that satisfy its constraints and that are within allowable variable domains. A measure of success when carrying out this task is called consistency. In general, CSP algorithms that ensure low degrees of consistency over-estimate the solution space but have low computational complexity. CSP algorithms that ensure high degrees of consistency provide a tight estimation of the solution space but suffer from high complexity. Complexity is defined in the computer science sense, thereby referring to the sensitivity of execution time to the amount of input data. Researchers from the AI community have developed polynomial algorithms to compute approximations of solution spaces, which allow for backtrack-free search for solutions [33,34]. When no backtrack-free search is possible, many heuristics to accelerate search exist [17].

If collaborating partners specify CSPs instead of single solutions to their subtasks, solutions to the complete project task are obtained through solving the CSP formed by the conjunction of sub-CSPs. Solution spaces defined by CSPs have the potential to improve efficiency in collaboration. In addition, better solutions would emerge because project partners were less restrictive when confronted with innovative ideas. A key challenge to the adaptation of CSPs in collaborative engineering is the difficulty associated with solving very large CSPs on continuous variables.

CSPs have been already proposed to describe solution spaces by Darr and Birmingham in the system ACDS [6]. A distributed agent architecture solves collaborative tasks using catalogues and restrictions instead of single propositions by collaborators. Component catalogues and requirements specified on parameters are considered as a CSP on continuous variables. Domains of the variables are deduced from the catalogues and the requirements collected from collaborators. After imposing local consistency on parameter ranges, many components can be eliminated before starting the search for the best solution using utility functions. The combinatorial explosion during search is thus weakened.

Lottaz et al. have recently suggested the use of continuous CSPs for collaborative design [20]. We improve upon this work through proposing new methods for supporting decision-making in the context of collaborative design. We give further evidence related to the performance limitations noted in Ref. [20] and we show that the effect of these limitations can be reduced through new methods for algebraic reformulation. Finally, we illustrate the usefulness of our implementation and the importance of solution spaces to designers within a full-scale civil engineering project. This paper also extends the initial work through focusing on the use of solution spaces for decision-making during negotiation.

This article is structured as follows: Section 2 contains theoretical considerations on the benefits achieved by the use of solution spaces in collaborative design and negotiation. Section 3 describes an Internet-based implementation of our approach and Section 4 illustrates the results achieved in an example from civil engineering. Some limitation and future work is shown in Section 5 before we draw our conclusions.

2. Collaborative design using solution spaces

When communication between collaborators includes solutions spaces instead of single solutions, collaboration systems may provide additional engineering support. More specifically, system using solution spaces may assist in avoiding artificial conflicts, detecting real conflicts, maintaining project consistency, making informed decisions and guiding negotiation. Constraints provide a convenient means for representing, exchanging and manipulating information about solution spaces.

2.1. Augmenting point solutions by solution spaces

In traditional approaches to collaborative design, responsibility for subtasks is distributed among several project partners. Collaborators determine a single solution for their subtask and meet in order to negotiate for the integration of partial solutions. In this process, collaborators have to provide exactly one solution even though many solutions to their subtask may be acceptable. We suggest the use of solution spaces instead of single solutions in order to assist the negotiation process.

Solution-space determination can be modelled by CSPs. The definition of a CSP contains a set of variables and a set of constraints. Each variable has a domain of possible values. A solution to a CSP is an instantiation of a value
to each variable from its domain such that all constraints are satisfied. Through departing from traditional point solutions (one value for each variable), CSPs augment the amount of information available for subsequent decisions. For example, CADRE [13] and IDIOM [37] use constraint solving on constraints on geometric parameters to enhance apartment floor layout plans, thereby facilitating adaptation. The use of CSPs helps delaying decisions for variable values until they become essential for the completion of the project. When premature decisions are reduced, information related to possible alternatives is retained. This is a variant of the least commitment paradigm often employed for planning tasks. In the automotive industry, major manufacturers have already adopted such decision delay strategies [40].

2.2. Artificial and real conflicts

Many conflicts in traditional collaboration arise from forced early, uninformed decisions related to the values of variables. Collaborators are required to suggest one single solution to their subtask although in many cases, many solutions may be acceptable. This loss of information related to other solutions causes artificial conflicts. Sometimes there is no real conflict even though a negotiated change between two partners does not converge to an acceptable solution for all partners without compromising a constraint. When negotiating over single values for parameters, collaborators provoke artificial conflicts that lead to needless and sometimes unsuccessful iterations of negotiation over all constraints.

For example, during the design, fabrication and erection of a steel-framed building in Geneva (Switzerland), collaborators negotiated over geometrical parameters of a beam with holes for passing ventilation ducts. Since all collaborators were defending their point of view by choosing conservative values for their parameters, much iteration was necessary during negotiation. In fact, the steel contractor had to assume values before negotiation had terminated in order to satisfy the construction schedule. Several thousand dollars were wasted because another partner eventually refused the assumed values. This refusal made tens of fabricated beams worthless [20].

When solution spaces are considered instead of point solutions, any subtask allows calculation of a local solution space for common variables. The intersection of all local solution spaces contains the feasible value combinations for the whole task. No early decisions on single values for parameters are taken and thus no artificial conflicts arise. Solution spaces can then be visualised and used as aid for negotiation. In the steel-framed building example mentioned above, it turned out that in fact many solutions were possible. Had the solution-space approach been available to the partners involved in this project, no iterative negotiation would have been necessary and no wasted fabrication effort would have occurred.

Use of constraints for a collaborative design task can also help to detect real conflicts at early stages. When important requirements are expressed as mathematical relations, conflicts due to diverging design goals can be detected as soon as the intersection of solution spaces is empty, even though the formalised information may not yet be complete.

In contrast to traditional point solution approaches, we suggest that collaborators first negotiate about design goals represented as constraints before searching for single values for design parameters. During immediate negotiation about parameter values, the detection of contradicting goals is very difficult. After a certain number of iterations, collaborators just stop the process, assuming that no solution for the problem exists; although in fact, they have no possibility to proof this. On the other hand, when a real conflict has been detected using solution spaces, collaborators must compromise on a higher level before the work can proceed, i.e. project partners have to revise project goals represented by the constraints specified so far, such that contradicting requirements are avoided.

2.3. Decision-making during negotiation

Ultimately, negotiation over single solutions is unavoidable. After all important project requirements are modelled using constraints and no contradiction has been detected, project partners must decide which solution should actually be built. A solution space is a representation of the space of feasible alternatives. During negotiation over the final solution, knowledge of the form of solution spaces usually encourages partners to be less restrictive. Therefore, solution spaces provide useful support for negotiation and may enhance the quality of the product as well as improve the efficiency of the negotiation process.

In engineering, modelling of complete project knowledge is rarely feasible and therefore, global optimisation is an unattainable goal. However within solution spaces, it may be useful to provide support for identifying solutions that are better according to selected criteria. This is called optimally directed decision-making and many algorithms exist, for example [29], to support such effort. Solution spaces improve the efficiency of these algorithms through defining the sets of possible point solutions.

Moreover, many engineering tasks have several optimisation criteria. Automatic multi-criteria optimisation is not always adequate because priorities or weights for each criterion cannot always be given. Visualisation of projections of solution spaces onto a set of optimisation criteria is a means for illustrating tradeoffs between these criteria. Thus, more informed decisions can be taken during negotiation.

Solution spaces also make hidden dependencies between parameters explicit. This helps to understand the impact of a decision about one parameter on the other parameters during negotiation. Interactive exploration of such multi-dimensional dependencies provides collaborators with more information about the decision to be made.
2.4. Change management

When working with point solutions, values for parameters change frequently over the life of a project. Early collaboration systems failed because responsibility for maintaining design consistency shifted to the receiver of the change. If there is such a shift in responsibility, partners lose control of the project when they become unable to verify project requirements. The result is a project that is more expensive, has lower overall quality and takes longer to complete than a project without any computer support at all. Therefore, collaboration systems have the potential to make matters worse.

When solution spaces are computed after successful negotiation about design goals, the initiator of a change to a point solution has information related to consistent values before the change is decided upon and communicated to others. Therefore, it becomes possible to require, for example, that all modifications should fit into the declared solution space in order to avoid re-negotiation about design goals. In this way, solution spaces are able to forewarn the change initiator of consistency problems and subsequent extra work if changes are not within solution spaces. As a result, recipients of changes are much less likely to be overwhelmed by verification tasks as the project progresses.

Moreover, solution spaces represented as CSPs simplify adoption of changed requirements. Within traditional negotiation systems changes in requirements often mean that much negotiation must be revisited. When a constraint-based approach is used, changes to requirements lead to the introduction of new constraints and modifications to existing constraints. Re-calculation of solution spaces is then be done automatically, thus supporting better change management.

3. Implementing CDSS using constraint satisfaction techniques

Constraint satisfaction techniques have the potential to implement the collaborative design using solution spaces (CDSS) approach described in Section 2. SpaceSolver is an Internet-based software package that provides constraint techniques for numerical CSPs. It also contains many features that are needed in collaboration, such as data-management and protection facilities for shared and private data for collaborative projects. It is available at the URL:

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

SpaceSolver relies on the Common Gateway Interface (CGI) and performs most of the computation on the server. Its system architecture is illustrated in Fig. 1. Any Web-browser can be used on the client-side, while on the server-side, a dedicated Web-server handles data-management tasks and permanently communicates with various Space Solver-modules. These modules include a Symbolic Manipulator for rewriting CSPs, a Constraint Converter for the generation of spatial representations of constraints, a Consistency Solver to compute several degrees of consistency and a VRML Generator to visualise constraints and approximations of solution spaces.

The remainder of this section describes the five phases SpaceSolver suggests to support CDSS:

- Specify parameters and constraints
- Perform algebraic reformulation
- Convert symbolic constraints into a spatial representation
- Compute consistency to approximate solution spaces
- Visualise and explore approximations of solution spaces.

3.1. Communication with constraints

CSPs have several advantages in the context of collaborative design. They are simple to manipulate, they occur naturally in engineering tasks, they are adapted to implement the CDSS approach described above and they provide an unambiguous formal way of specifying project requirements.

![Fig. 1. SpaceSolver system architecture.](image-url)
Much engineering knowledge is expressed in terms of constraints. Building codes, design criteria, behaviour models, cost restrictions and planning strategies all employ explicit declarations of constraints. Computer systems that propose support for engineering tasks often involve a transformation of this knowledge into other forms such as rules, directed relationships and post facto tests. Such transformations reduce clarity for engineers who use such systems and this results in systems that are hard to maintain and often incomprehensible to use. Since much engineering knowledge is already in constraint format, direct use of constraint representations is obviously desirable. Constraint-based systems have the potential to be one of the most understandable and easiest to maintain of all reasoning systems proposed for engineering tasks such as design.

Moreover, CSPs are also adapted to implement the CDSS approach described in Section 2, since any CSP represents a solution space, i.e. the set of all its solutions. Constraint satisfaction techniques provide the technology to approximate these solution spaces with tractable complexity.

Some problems in collaboration arise due to misunderstandings and imprecise communication. When collaborators use a formal language such as constraints, variables and variable domains to specify their needs, they necessarily specify their requirements in an unambiguous, precise way. In SpaceSolver, we use constraints given as closed mathematical formulas to express project requirements on an Internet-based communication platform.

When users connect to the SpaceSolver — WWW page, they are prompted for a user-name and a password. This user authentication allows us to maintain separation of data and control of access to collaboration projects. Registration is free.

Upon selection of an existing CSP network or creation of a new one, users are presented a page similar to Fig. 2. On this page, users specify equalities and inequalities in ASCII-text using Maple V’s syntax, as well as minima, maxima, default values and short descriptions for each variable. Maple V is the algebraic computation package we use to interpret the specified constraints. The table that specifies variables, shown in the lower part of Fig. 2 is generated automatically by detecting the variables in the user’s constraints.

SpaceSolver facilitates collaboration on engineering projects because several collaborators are allowed to participate in the same project. Although collaborators

![Fig. 2. SpaceSolver’s Internet-based user interface.](image-url)
maintain their own set of constraints, they may share variables, i.e. a variable can be implied in constraints by different collaborators. The creator of a project specifies, who is allowed to contribute to a project (Fig. 3). Collaborators can be added and removed at any time.

The constraints and variables submitted by other collaborators can be analysed through links generated by SpaceSolver. For every collaborator a link is provided that brings up a list of all constraints posted by the corresponding project partner. Since certain variables will be shared, collaborators must also be able to find which variables are already defined. SpaceSolver provides a summary of all variables defined with their minimum, maximum, and default value, as well as a short description.

Whenever a user starts computation, constraints and variables that are defined in this way are collected into one constraint satisfaction problem. All results obtained are thus valid for the conjunction of the CSPs defined by all collaborators. The process of solving the CSP is divided into algebraic rewriting, conversion into spatial representation and computation of consistency.

3.2. Rewriting constraint satisfaction problems for consistency algorithms

Rewriting CSPs is useful for two reasons. Firstly, rewriting a CSP can be used to enforce a normalised form of the CSP as it is needed by certain constraint satisfaction techniques. Secondly, algebraic reformulation can produce important simplifications and thus better performance for constraint satisfaction algorithms. More details related to this aspect are given in Ref. [19].

SpaceSolver focuses on numeric CSPs. The variable domains of a numeric CSP are continuous intervals in real numbers and its constraints are expressed as closed mathematical expressions (equalities and inequalities). The arity of a constraint is the number of variables it involves. The arity of a CSP is equal to the arity of the highest-arity constraint. Reformulation of numeric CSPs in lower arity before applying constraint satisfaction techniques is a common procedure because CSPs of lower arity are considerably simpler to treat.

It has been shown that rewriting numeric CSPs in terms of ternary constraints is possible as long as only unary and binary operators occur in the constraints. This condition holds for many practical applications. Therefore, certain consistency algorithms such as 2-consistency [9,12] only accept ternary constraints. 2B- as well as 3B-consistency [18] are based on primitive constraints which are also ternary and the complexity of other consistency algorithms such as (r, r − 1)-relational consistency [33] are exponential in the arity of the given CSP and therefore low-arity CSPs are treated much more efficiently.

3.3. Eliminating unnecessary intermediary variables

Designers and engineers use constants and intermediary variables in order to ensure that mathematical formulas are easy to read and adaptable to other contexts. Through changing constants, constraint sets reflect requirements of many similar (but not equivalent) situations. However, some of these variables should be eliminated from the CSP by substitution before computing consistency for computational efficiency.

An intermediary variable $a$ is defined in the CSP as the result of a functional expression $a = f(x_1, \ldots, x_n)$. It can be removed from the CSP by eliminating its definition from the CSP and by substituting $a$ wherever it occurs in the remaining constraints of the CSP by $f(x_1, \ldots, x_n)$. However, substitution of $a$ is only enough to keep the CSP equivalent, when $a$ is a constant. Otherwise, the information contained in the domain of $a$ is lost and thus the new CSP is less restrictive. Therefore, the constraints $a_u \geq f(x_1, \ldots, x_n)$ and $a_l < f(x_1, \ldots, x_n)$, must be added where $a_l$ and $a_u$ are the lower and upper bounds of the domain of the variable $a$.

Candidates for potentially unnecessary variables can be found in equalities. Solving an equality for one of its variables $a$ yields the definition $f(x_1, \ldots, x_n)$ for $a$. However, this

![Fig. 3. SpaceSolver’s facilities for collaboration.](image-url)
is only valid if \( f(x_1, \ldots, x_n) \) is functional. Otherwise, for instance if the original equation was quadratic in \( a \), substitution of \( a \) by \( f(x_1, \ldots, x_n) \) in the CSP is not equivalent.

In the case of a quadratic expression, substitution of \( a \) by \( f(x_1, \ldots, x_n) \) leads to the loss of one of the two possible solutions for \( a \).

If \( a \) is a valid candidate for substitution, it should be substituted if its substitution does not increase the arity of any constraint to more than three. Removing a variable when its substitution renders a constraint \( C \) non-ternary is unlikely to be useful in the context of ternarisation, because it would imply an additional subsequent generation of at least one auxiliary variable in order to rewrite \( C \) in ternary form. For illustration consider the following small example, a simplified problem from civil engineering:

\[
u < (3.18e^{-5}H_b + 0.0054)S
\]

\[
H_b > 137.7 - 0.8633S + 5.511e^{-5}S^2 - 8.358e^{-9}S^3
\]

\[
p = u + 9.62e^{-5}(0.0417W)^{1.5161}
\]

\[
H_b > 0.077(pW^2)^{0.3976}
\]

\[
H_b > 0.0168(SW^3)^{0.2839}
\]

\( H_b, H_s \) and \( W \) thereby define the geometry of a beam (beam depth, slab depth and beam span, respectively) while \( S \) is the beam spacing in a steel-framed school building. \( u \) and \( p \) are intermediary variables and the constraints are the result of several aggregations. This set of constraints contains the intermediary variable \( p \) defined as \( u + 9.62e^{-5}(0.0417W)^{1.5161} \). Thus, the definition of \( p \) only involves the variables \( u \) and \( W \). The only occurrence of \( p \) is in \( H_b > 0.077(pW^2)^{0.3976} \). Substituting \( p \) in this ternary constraint leaves it ternary because \( W \) is involved in both, the definition of \( p \) and the constraint where \( p \) is to be substituted. Therefore, \( p \) is an unnecessary intermediary variable and its elimination accelerates computing consistency.

Since the substitution of a constant or an intermediary variable in the above described manner may decrease the arity of a constraint, unnecessary variables are eliminated iteratively until no more changes occur.

3.4. Making constraints ternary

\( \text{SpaceSolver} \) applies consistency techniques, which were developed for ternary numeric CSPs. Therefore a reformulation of CSPs in ternary form is necessary. Any mathematical expression built using unary and binary operators can be rewritten in ternary form through use of auxiliary variables as illustrated in Fig. 4. Introducing an auxiliary variable for each intermediary result generated by a binary operator obviously guarantees that no constraints involving more than three variables are left. However, this method possibly generates many auxiliary variables. So far, few procedures for reformulating numeric CSPs in ternary form have been suggested; rewriting is often done by hand. Since the performance of consistency algorithms depends on the number of variables involved in the given CSP, algorithms that introduce a small number of auxiliary variables are needed.

A simple algorithm to rewrite a given CSP in terms of ternary constraints is suggested in Refs. [33,12]. It replaces any binary operator in a constraint by a new auxiliary variable, which represents its result. This step is iterated until all constraints have arity three or less. The operands of the chosen operator do not contain binary operators in order to avoid introducing non-ternary constraints when defining auxiliary variables.

This algorithm shows that it is always possible to rewrite a numeric CSP expressed using unary and binary constraints in terms of ternary form but it generates far too many auxiliary variables. There are three main reasons for this. Firstly, it unnecessarily introduces binary constraints if one of the replaced operator’s operands is constant or if both operands involve the same variable. Secondly, it does not allow the introduction of complex definitions for auxiliary variables, since only one binary operator is allowed in such definitions. Finally, it does not reuse auxiliary variables in other constraints or sub-expressions. Some implementations improve upon the weakness of not reusing auxiliary variables through avoiding duplicate definitions. This allows for some optimisation. However, current algorithms do not choose the definition of auxiliary variables such that they can be reused in several places. Therefore, many opportunities for reusing auxiliary variables are missed.

We suggest a more general algorithm to perform the task of rewriting numeric CSPs in ternary form: In the first step the constraints, which already have ternary form are sorted out and are no longer manipulated. In the second step the algorithm searches for an expression in two variables, which occurs in one of the non-ternary constraints. The third step is to substitute the expression found in step two in all non-ternary constraints. These three steps must be repeated until the list of non-ternary constraints is empty. Fig. 5 illustrates this procedure. In step two sub-expressions involving exactly two variables are chosen because these expressions
have the potential to decrease the arity of a constraint, and at the same time they do not add non-ternary constraints to the system.

In order to find expressions for defining auxiliary variables (step 2), we must find sub-expressions in two variables occurring in the CSP. This is performed by traversing the expression tree defined by the CSP. Whenever the traversing algorithm visits sub-expressions involving exactly two variables, it stores them into a list instead of descending further into the expression tree. Thus no sub-expressions of expressions in two variables are considered.

As soon as the candidate expressions are determined we must decide which is the best expression to be used. Since we want to decrease the arity of all constraints below four, the sum of the arities of all non-ternary constraints seems to be a reasonable criterion to minimise. Therefore, we choose the candidate expression which decreases the arity of the most of the non-ternary constraints, breaking ties in favour of the candidates which generate the simplest constraints after substitution, i.e. the constraints with the fewest operands. In order to determine the best candidate expression, all candidates are substituted in the non-ternary constraints and the one yielding the best result is chosen.

This approach for selection of expressions to define auxiliary variables and heuristics to find reoccurring sub-expressions for defining auxiliary variables can treat full-scale practical examples. In this manner, large sets of constraints involving many variables can be rewritten automatically. Examples involving 50 and more variables are rewritten in 3 min on a SUN Ultra/10 workstation. Tests showed that our automatic ternarisation does not introduce many more variables than a ternarisation by hand in most cases [19].

3.5. Convert symbolic constraints into spatial representation

We use a spatial representation of the feasible regions of constraints for computing a labelling of several degrees of consistency. A hierarchical data-structure called \( 2^k \)-tree is employed to represent regions and volumes. This data-structure was originally developed in computer vision [35] and was later proposed for engineering design in Refs. [33,34]. This representation is based on the assumptions that in most practical applications, each variable takes its values in a bounded domain (bounded interval) and that results with limited precision are of interest.

Provided that these two assumptions hold, a relation can be approximated through a hierarchical binary decomposition of its solution space into quadtrees for binary relations and octrees for ternary ones (see Fig. 6). Graphically in two-dimensions (2D), this means that rectangles that are not completely in the solution space are subdivided into four smaller rectangles and re-tested iteratively until the desired precision is obtained.

Numerical constraints generally arise as algebraic or transcendental equalities and inequalities involving several variables. Many CSP approaches process constraints directly in this form and therefore they encounter analytical difficulties related to intersecting surfaces and finding extrema. Such analytical complexity quickly increases when computing higher degrees of consistency. This partly explains why the most prominent advances in numerical constraint satisfaction are related to 2-consistency [9,18,39].

Instead of an implicit representation of numerical constraints, the quad/octree representation leads to a logical rather than analytical treatment of continuous solution spaces, thereby avoiding problems with singularities and other analytical anomalies. This allows for a simple

![Fig. 5. General algorithm for rewriting CSPs in ternary form.](image)

![Fig. 6. Quadtree-representation for \( y > \arctan(1/x - 2) \).](image)
implementation even for high degrees of consistency in continuous domains (see Ref. [33] for further details of the quad/octree construction and consistency algorithms).

During hierarchical decomposition, generation of these $2^k$-trees relies on a procedure involving a test to determine whether a rectangular or cubic region is feasible or unfeasible. Interval arithmetic is used to avoid symbolic solving of equalities and to improve performance compared with simple sampling. Interval arithmetic provides exact results as long as no variable reoccurs in the expressions to be analysed. In this case, interval evaluation of the expression $f(x_1, ..., x_n)$ computes the minimum and maximum value of the expression and thus allows to determine whether the inequality $f(x_1, ..., x_n) > 0$ is feasible or unfeasible in the given intervals for $x_1, ..., x_n$. Otherwise, SpaceSolver falls back on simple sampling.

Constraints involving the same set of parameters are stored in one total constraint. The feasible regions of the original constraints are intersected to form one single feasible region of the corresponding total constraint, thereby simplifying the CSP.

3.6. Constraint satisfaction techniques

CSPs do not only provide a means for manipulating and communicating about solution spaces, constraint satisfaction techniques also provide the necessary methods to approximate solution spaces in a tractable way and thus allow for the implementation of CDSS. SpaceSolver provide the techniques described in this subsection.

3.6.1. Definitions

Constraint problems are generally formalised in terms of variables, which represent properties of components, and constraints, which specify the restrictions that must hold for any valid solution of the task. Sets of constraints and variables together with their allowable values (domains) are called CSPs. CSPs are generally represented using graphs (or hyper-graphs) called constraint networks. In such graphs, nodes represent usually variables and arcs represent constraints between variables. A CSP is binary if constraints involve at most two variables. Note that all CSPs on finite domains can be rewritten as binary CSPs [1]. A CSP is ternary if any constraint contains at most three variables. Note that as described earlier, a numerical CSP, stated analytically, can always be transformed into a ternary one without loss of information using syntactic transformations. A solution to a CSP is an instantiation of all variables such that all constraints are satisfied simultaneously. A solution space of a CSP contains all its solutions.

CSP algorithms fall into two categories. Consistency techniques use a variety of methods to restrict the search space to interesting regions. They prune parts of the search space to discard those branches where no solutions can be found. The second category includes splitting and backtracking techniques for searching for solutions. The task of solving a CSP in its general form is NP complete.

An important class of CSPs exclusively involves variables that have finite domains. Efficient methods to achieve various degrees of consistency [3,22,23] are available and sophisticated techniques for identifying solutions through intelligent backtracking exist [17]. While in many fields such as planning and resource allocation it is often useful and efficient to consider variables with finite domains only, tasks in engineering employ variables that have continuous values.

A numeric CSP is defined similarly to ordinary CSPs but the domains of its variables are continuous intervals on reals. Constraints of numeric CSPs are usually defined using closed mathematical expressions like equalities and inequalities. A CSP is convex when all its constraints are convex. A constraint on continuous variables is convex if the straight line between any two feasible points entirely lies within the feasible region. A numeric CSP is convex if all its constraints are convex.

3.6.2. Finding single solutions for continuous CSPs

Resolution of CSPs involving variables with continuous domains (CCSPs) is different from solving CSPs on variables with finite domains. Most successful resolution techniques for numerical CSPs on continuous variables concentrate on finding single solutions, possibly considering optimisation criteria. Most techniques are not related to traditional CSP research but use principles from operations research instead. Popular techniques include linear and non-linear programming, numerical analysis, hill-climbing and stochastic techniques.

Such approaches are adequate when a problem is entirely specified; complete specification makes it reasonable to search automatically for the best solution although many solutions may exist. However, since complete specification cannot feasibly be achieved in full-scale engineering practice, more sophisticated support for decision-making is desirable.

3.6.3. Local consistency

Local consistency is an approach for approximating solution spaces. The original search-space of a CSP is defined by the cross product of the domains of all variables. Local consistency algorithms prune parts of this search-space through detecting local inconsistencies. Such inconsistencies are incompatibilities between values of closely related variables.

There are many ways to enforce local consistency and each method provides a different degree of reliability. It is possible to test for different orders of consistency: 1-, 2- and in general $k$-consistency, depending on the degree, $k$, of locality taken into consideration. A $k$-consistency algorithm furnishes an approximation of the solution space where each sub-problem of $k$ variables is guaranteed to be consistent. This amounts to ensuring that each partial solution of $k - 1$
variables can be extended consistently to a partial instantiation of k-variables.

In practice, consistency is verified through assigning a label to each subset of $k-1$ variables. Each label contains feasible value assignments. Thus a 3-consistency algorithm produces a 2D label for each pair of variables. For any combination of values contained within such a label, any third variable can be instantiated such that all constraints of the CSP are satisfied.

Although $k$-consistency ensures extensibility of partial solutions involving $k-1$ variables to any $k$-th variable, there is in general, no guarantee that such partial solutions are extensible to a complete solution if the CSP contains more than $k$ variables. A locally consistent labelling usually overestimates the solution space and it is even possible that a locally consistent label does not contain any value combination that is extensible to a consistent solution.

However, enforcing local consistency has the advantage that it has polynomial complexity and therefore, low degrees of consistency can be achieved very quickly. Moreover, if some local consistency algorithms result in empty labellings, it is guaranteed that no solution to the full CSP exists. As a result, local consistency methods are often used in practice to improve search through pruning inconsistent values from the domains of the variables and to detect conflicting constraint sets.

### 3.6.4. Global consistency

Construction projects require that all values for variables are globally consistent with project constraints. Inconsistencies cause project delays, extra costs and accidents. Therefore, a stronger notion than local consistency is desirable. When the labelling constructed by a consistency algorithm contains only those values or value combination that occur in at least one solution, it is said to be globally consistent. A globally consistent labelling is a compact and conservative representation of all solutions admitted by a CSP. It is sound in the sense that the labelling never admits any value combination, which does not lead to a solution. It is complete in the sense that all solutions are represented in it.

In general, a globally consistent labelling may require explicitly representing all implicit and induced constraints in the problem (i.e. computing $n-1$-dimensional labels for a problem of size $n$), a task, which has exponential time and space complexity in the worst case. Recent results show, however, that for special classes of problems, low orders of consistency are equivalent to global consistency. These results lead to polynomial time algorithms for computing globally consistent labellings and can be summarised as follows:

- 2-consistency (also called arc-consistency) is equivalent to global consistency when the constraint network is a tree graph [10],
- 3-consistency (also called path-consistency) is equivalent to global consistency when the CSP is convex and binary [7,38],
- (3,2)-relational-consistency (a variant of 5-consistency defined in Refs. [33,34]) is equivalent to global consistency when the CSP is convex and ternary.

Therefore, algorithms, which achieve 2-, 3- and (3,2)-relational consistency can compute complete and sound descriptions of the entire solution space at low computational cost under certain conditions. When the problem has no special simplifying conditions, these algorithms are often useful pre-processing tools for reducing the size of the search space. They can be interleaved with interval-based backtrack-search algorithms to generate consistent sub-regions of the solution space [14]. Alternatively, users may wish to explore the solution space interactively, particularly when it is known that available formalised knowledge is incomplete.

#### 3.6.5. Visualising and exploring solution spaces

Visualisation of constraints and solution spaces helps obtain an understanding of the constraint network’s characteristics and this supports decision-making. For instance, suppose that an engineering task has three (partial) optimisation criteria. Through visualising the projection of the solution space on these criteria, tradeoffs may be illustrated and possible alternatives can thus be examined. In order to provide a visualisation, VRML-scenes representing constraints and solution spaces are generated dynamically.

VRML is a 3D modelling language for the Internet. Several plug-ins to WWW-browsers and stand-alone VRML-browsers allow Internet-users to examine scenes specified as VRML. Projections of the solution space on any triplet of variables can be generated and visualised in SpaceSolver as illustrated in Fig. 7. Projecting consistent spaces on up to 3D

As a further facility for decision making, SpaceSolver provides the possibility to explore the solution space approximations interactively as illustrated in Fig. 8. For each variable of the CSP a slider is provided. The positions of these sliders represent values for their attributed variable, thus all sliders together represent one point solution in the search space. The coloured regions besides the sliders show which values the corresponding variable can take without compromising the chosen degree of consistency given that no other variables are changed. Dark regions represent values which are outside the consistent space bright regions are within.

By interactively moving through the search space through manipulation of sliders, users observe complex multi-dimensional relations in a simple and intuitive manner. As a result, the impact of alternative decisions on other design parameters can be anticipated, thereby improving decision-making.
Fig. 7. 3D projection of a solution space approximation.

Fig. 8. Interactive exploration of solution space approximations. Grey regions in the grey and black vertical bars indicate approximations of ranges of feasible values for variables.
4. An illustration of CDSS in civil engineering

4.1. A steel-framed building

This example was employed initially to illustrate functionality of a collaborative environment that includes Space-Solver. This work is part of a project involving architects (CAAD-ETH) in Zürich as well as computer scientists (LIA) and civil engineers (IMAC) at EPFL in Lausanne [21]. The example is taken from a real construction case as proposed by our industrial partner, a steel fabricator and contractor (Zwahlen & Mayr SA, Aigle, Switzerland) represented by A. Flückiger. This project, a storage hall in Gösgen for a nuclear power plant, is a good test for collaborative support between partners. Apart from the steel fabricator, partners include the client, the civil engineer, who coordinates work, the wind expert as well as the crane designer and supplier. These partners work together to define the main structural elements of the building. Each of them defends specific requirements, and the civil engineer is responsible for providing consistent proposals for the steel frame structure.

For this example, partners have the following tasks:

- **Client** — choose the volume of the hall to contain generators and define the place and width of openings, and define its use in terms of loads, goods to store and handling of these goods.
- **Crane designer and supplier** — after discussions with the client, configure the crane and communicate important parameters to others, for example, the civil engineer needs to know its self weight.
- **Steel fabricator** — provide all information related to sections of steel elements and material properties.
- **Wind expert** — determine the impact of the wind on the structure including the influence of the nearby cooling tower.
- **Civil engineer** — design the steel frame structure considering requirements of the client, the crane designer, the steel fabricator, and the wind expert.

Only structural safety criteria, and not serviceability, have been modelled in certain places. Relevant parameters and the constraints are given in Tables 1 and 2. Parameter meanings are illustrated in Figs. 9 and 10.

4.2. Rewriting the CSP

The CSP associated with the storage hall example is of considerable size. It involves more than 100 variables. A detailed analysis using complex consistency algorithms of high degree is not possible for a CSP of this size. However, in our example the elimination of unnecessary intermediary variables proves to be very useful.

The collaborators employed many constants in order to keep the constraints adaptable and thus facilitate what-if analyses. SpaceSolver eliminates most of the variables leaving just eight original variables in the system for consistency analysis and adds only one auxiliary variable during ternarisation. This dramatic simplification of the problem is due to chained substitution of variables. Substitution of constants and unnecessary intermediary variables recursively induces many other such substitutions until the system is almost ternary and more than 90 variables are removed. This transformation is performed within a few seconds.

Chained removal of many unnecessary variables leads to complex expressions as shown in Fig. 11. This illustrates the necessity of allowing complex definitions of auxiliary variables. Rudimentary algorithms for ternarisation are unable to transform expressions as shown in Fig. 11 efficiently into ternary form, while our generalised algorithm for ternarisation as suggested in Section 3 only adds one auxiliary variable.

4.3. Change management and detection of real conflicts

SpaceSolver supports conflict detection using low degrees of consistency. We illustrate this possibility using the following scenario:

The civil engineer has defined the steel frame structure and has already asked the steel contractor to manufacture it. Then, the client asks for a crane, which is able to lift heavier loads. Since the manufacturing of the steel beams has already begun, a change of their cross-section is not possibly without important extra cost, but the civil engineer proposes to reduce the spacing of the frames in order to accommodate
Table 1
Definitions of variables for storage hall example (excerpt)

<table>
<thead>
<tr>
<th>Geometric parameters</th>
<th>Materials property features</th>
<th>Loads</th>
<th>Structural analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_C$ Flange width of column</td>
<td>$E_{PB}$ Plastic modulus of the beam</td>
<td>$g$ Dead load of the roof</td>
<td>$\beta$ Effective length coefficient for buckling</td>
</tr>
<tr>
<td>$w_B$ Flange width of beam</td>
<td>$M_{BB}$ Moment causing buckling for the beam</td>
<td>$q_s$ Snow load</td>
<td>$k$ Relative stiffness coefficient of the frame</td>
</tr>
<tr>
<td>$L$ Length of the building</td>
<td>$M_{BC}$ Moment causing buckling for the column</td>
<td>$q_w$ West façade wind pressure</td>
<td>$M_{df}$ Design moment in the column footing</td>
</tr>
<tr>
<td>$w$ Width of the building</td>
<td>$I_{Bxx}$ Moment of Inertia of the beam</td>
<td>$q_{w_2}$ West roof wind pressure</td>
<td>$M_{dB}$ Design moment in the frame joint</td>
</tr>
<tr>
<td>$h_1$ Clear height from floor to the crane</td>
<td>$I_{cax}$ Moment of Inertia of the column</td>
<td>$q_{w_3}$ East roof wind pressure</td>
<td>$V_{dc}$ Design shear force in the column footing</td>
</tr>
<tr>
<td>$h_{tot}$ Height from crane to the roof</td>
<td>$A_C$ Area of column cross section</td>
<td>$q_{w_4}$ East façade wind pressure</td>
<td>$\gamma_C$ Resistance factor</td>
</tr>
<tr>
<td>$h_{tot}$ Total height of the building</td>
<td>$A_B$ Area of beam cross section</td>
<td>$g_{crane}$ Self weight of the crane</td>
<td>$M_{pl}$ Plastic moment of the beam</td>
</tr>
<tr>
<td>$s$ Spacing between frames</td>
<td>$r_{Cax}$ Radius of gyration of the column</td>
<td>$Q_{crane}$ Crane load</td>
<td>$\gamma_B$ Factor for cross-section strength</td>
</tr>
<tr>
<td>$d_1$ Minimum distance between the crane load and the rail supporting the crane</td>
<td>$f_y$ Elastic limit</td>
<td>$Q_{max}$ Maximum load on the rail supporting the crane</td>
<td>$\gamma_{crane}$ Factor for cross-section buckling</td>
</tr>
<tr>
<td>$s_C$ Span of the crane</td>
<td>$E$ Young’s modulus</td>
<td>$Q_T$ Braking load of the crane</td>
<td>$N_{cy}$ Critical elastic buckling load</td>
</tr>
<tr>
<td>$d_2$ Distance from column axis to the rail supporting the crane</td>
<td>$D$ Steel density</td>
<td>$\lambda$ Coefficient for horizontal strength of the crane [SIA 160]</td>
<td>$N_{ul}$ Ultimate buckling load</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\phi$ Dynamic amplification factor [SIA 160]</td>
<td>$\sigma_k$ Ultimate buckling stress</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\xi$ Lifting coefficient [SIA160]</td>
<td>$l_{GC}$ Effective length of the column</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\lambda_k$ Slenderness ratio</td>
<td>$l_{w_20}$ Second-order factor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$j_{2ndO}$</td>
<td>$\gamma_C$ Column security factor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$q_{steel}$</td>
<td>$q_{w_{east}}$ Quantity of steel</td>
</tr>
</tbody>
</table>
the new requirements, but this solution may conflict with an earlier requirement of the client, imposing that spacing must not be smaller than 5 m due to the size of the openings.

In traditional collaborative design, collaborators have to find a new solution by renegotiating large parts of the project without knowing if a solution exists and which part of the old solutions can be kept. Within construction projects, such problems often happen at the end of the process when needs for changes become obvious. However, last minute changes possibly lead to higher costs and lower quality.

### 4.4. Tradeoff analysis

In order to determine what are the best combinations of flange width of the column and the beam, the civil engineer can do the following analysis using SpaceSolver. The following optimisation criteria are involved in the appropriate consideration: security factors of beams and columns as well as cost. The security factors must be greater than one and define the level of security. Consequently, the higher

### Table 2

Constraints for the storage hall example (excerpt)

<table>
<thead>
<tr>
<th>Constraints given by the client</th>
<th>Constraints given by the civil engineer</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_1 = 12 )</td>
<td>( h_1 + h_2 = h_{tot} )</td>
</tr>
<tr>
<td>( L = 41 )</td>
<td>( s = 5.8 )</td>
</tr>
<tr>
<td>( w = 20.8 )</td>
<td>( g = 1.7x + 1.7 )</td>
</tr>
<tr>
<td>( Q_{\text{crane}} = 500 )</td>
<td>( q_{f} = 0.8s )</td>
</tr>
<tr>
<td>( s \geq 5 )</td>
<td>( Q_{\text{max}} = \frac{Q_{\text{crane}}(s_C - d_i)}{s_C} )</td>
</tr>
</tbody>
</table>

### Constraints given by the crane designer

\[
\begin{align*}
d_1 & = 1 \\
s_c & = 19.5 \\
h_1 & = 3.7 \\
g_{\text{crane}} & = (5.1sC/2) \\
d_2 & = 0.65 \\
q_{a1} & = 0.96st \\
q_{a2} & = 0.6x \\
q_{a3} & = 0.36st \\
q_{a4} & = 0.36st \\
\end{align*}
\]

### Constraints given by the wind expert

\[
\begin{align*}
q_{w1} & = 0.96st \\
q_{w2} & = 0.6x \\
q_{w3} & = 0.36st \\
q_{w4} & = 0.36st \\
\end{align*}
\]

### Constraints given by the steel contractor

\[
\begin{align*}
E_{\text{DB}} & = -1065260 + 6310w_b + 7.795w_b^2 \\
M_{\text{DB}} & = -298.9 + 1.71w_g + 0.00136w_g^2 \\
M_{\text{BC}} & = -298.9 + 1.71w_C + 0.00136w_C^2 \\
I_{\text{DB}} & = 855 - 4.66w_B + 0.00948w_B^2 \\
I_{\text{BC}} & = 855 - 4.66w_C + 0.00948w_C^2 \\
A_b & = 4247 + 31.08w_b \\
A_c & = 4247 + 31.08w_c \\
r_{\text{BC}} & = 20.8 + 0.385w_C \\
f_y & = 235 \\
E & = 210 \\
D & = 7850 \\
\end{align*}
\]

\[
\begin{align*}
\frac{1}{\gamma_c} & = \frac{V_{c}g_{y}}{N_k} + f_{\text{h00}} \left( \frac{M_{g}g_{y}}{M_{B}} \right) \\
\gamma_c & \geq 1 \\
q_{\text{steel}} & = \frac{L}{s} \left( \frac{w_{A_b}D}{10^6} + \frac{2h_{A_c}D}{10^6} \right) \\
\end{align*}
\]

\(^a\) Crane with load is the most critical situation.

\(^b\) Defined by regression.
Fig. 10. Loads (top) and hazard scenario (bottom) for storage hall example.

\[
\begin{align*}
1 \leq & 8545.45 \left( -213052 + 1262 \, w_B + 1.559 \, w_B^2 \right) \\
& \left\{ \left( -6591 \times 10^{18} \, w_B - .6208 \times 10^{18} \, w_C - .1774 \times 10^{16} \, w_B^2 + .1592 \times 10^{16} \, w_C^2 \\
& + .1363 \times 10^{10} \, w_C^4 + .2724 \times 10^{16} \, w_B \, w_C - .5542 \times 10^{13} \, w_B \, w_C^2 - .1341 \times 10^{13} \, w_C^3 - .1765 \times 10^{13} \, w_B^2 + .1795 \times 10^{10} \, w_B^4 \\
& + .6194 \times 10^{21} \, w_B^2 \, w_C^2 + .1127 \times 10^{11} \, w_B^2 \, w_C \right) \\
& \left( 76540 - 75.45 \, w_C + .1535 \, w_C^2 - 341.7 \, w_B + .6951 \, w_B^2 \right) (228800 - 905.4 \, w_C + 1.842 \, w_C^2 - 341.7 \, w_B + .6951 \, w_B^2) \right\} \\
\end{align*}
\]

\[
\gamma_{SR1} = 181.818 \left( \frac{.24535917318 \times 10^7 \%1}{\%1} + .2754 \times 10^{11} + 37148.72069 \frac{\%1}{\%1} \right) \%2 \\
\left( .004 \%1 + 11 \right) \%2 \\
\%1 = .7654 \times 10^{8} - 75450 \, w_C + 153.5 \, w_C^2 - 341700 \, w_B + 695.1 \, w_B^2 \\
\%2 = 55370. - 301.8 \, w_C + .614 \, w_C^2 
\]

Fig. 11. Two constraints after elimination of unnecessary intermediary variables.
Fig. 12. The criteria for the storage hall example.

Fig. 13. Binary tradeoffs for storage hall example.
these factors are, the better is the security on the corresponding element.

Considering each combination of beam and columns flange width with respect to each of the mentioned optimisation criteria as shown in Fig. 12 we can find the following:

- $w_C/w_B/q_{\text{steel}}$ — lowest costs are reached when values for column flange width and beam flange width are low.

These results demonstrate that the best combination for one criterion is not the best combination for the other criteria. For example, the criterion for the quantity of steel leads to a column with very small cross-section, while the criterion for the security factor of the column leads to the opposite. Comparing each pair of criteria (Fig. 13), the following conclusions are of interest:

- $q_{\text{steel}}/\gamma_C$ — using high quantities of steel provides a high factor of security for the column, $\gamma_C$ can only be high for solutions which use high quantities of steel.
- $\gamma_{BB}/\gamma_C$ — enforcing a very high factor of security for the beam leads to a low security factor for the column. Consequently, the factor on the column is more critical than the factor on the beam.
- $q_{\text{steel}}/\gamma_{BB}$ — solutions which use large quantities of steel do not provide the best security factor for the beam.

Finally, if all criteria are placed in a 3D graph (Fig. 14), an overall view is obtained. The points which correspond to the best compromises in the 2D considerations appear as extrema of the 3D feasible region. This 3D volume illustrates the necessity of compromises in such multi-criteria decision-making tasks.

4.5. Explore solution spaces interactively

Visualisation of tradeoffs as illustrated above supports understanding of relations of up to 3D. More complex multi-dimensional relations cannot be visualised as easily. Interactive exploration of solution spaces is an attempt to provide intuitive support for understanding relationships.

Fig. 15 shows SpaceSolver’s solution space explorer for...
the storage hall example. All sliders are within the bright regions, thus indicating a possible solution for the problem. Since many parameters are fixed in this problem, the relations between the parameters are tight and the ranges of feasibility for the variables are small.

Interactively exploring the solution space helps to understand multi-dimensional relations. Moving the slider controlling \( w_B \) for instance shows that this variable is linked to several variables. It augments both factors \( \gamma_C \) and \( \gamma_{\text{fill}} \) while augmenting \( q_{\text{steel}} \) as well. All these relations are shown simultaneously, while moving the slider. Fig. 15 shows the start-point of the mentioned move and Fig. 16 the end-point. On the other hand, this application reveals that there is no direct or indirect link to \( w_C \) in the range \( w_B \) was modified.

5. Limitations and outlook

While the solution space concept is generally applicable, the implementation presented in this paper is limited to the use of constraints of continuous variables or parameters. Although this kind of constraint is common in many fields of engineering there are almost always parameters that have finite domains. For example, there are integer parameters, such as number of beams or number of holes in a beam, and there are qualitative parameters such as type of bridge and type of material. Integer variables can be considered continuous in a first phase and chosen in from their integer domain during the determination of single solutions. In contrast qualitative parameters can completely change the system of constraints and are therefore more difficult to accommodate. Nevertheless, constraint satisfaction techniques have been developed which are able to treat such mixed CSPs [12]. Given some extensions for visualisation and definition of such constraints and solution spaces, mixed CSPs could be adopted in CDSS.

Certain restrictions in applicability are also given by computational complexity of our consistency algorithms and the mathematical restrictions required for tractably computing global consistency. However, we observe that low degrees of consistency, which already allow to determine many conflicts, can be enforced for large constraint sets quickly enough for interactive use. Higher degrees of consistency need more time but allow for a more sophisticated analysis. Therefore, engineers may accept a few hours of computation. Moreover, we have shown that the algebraic reformulation of constraint sets allows for high consistency analysis within large projects. Finally, the convexity restrictions needed for global consistency represent a theoretical drawback. Even though no back-track-free determination of solutions is possible, consistent spaces reveal hidden relations and thus support decision-making.

A challenging field for further development is how negotiation should be performed in practice. Although we propose support for informed decision-making during negotiation, we do not explore how negotiation over goals should take place? The formulation of design goals as constraints could be extended by the concept to attribute an importance to each goal. Constraint satisfaction techniques for overconstrained problems could help determine the best set of goals, which can be reached. Also, when collaborators accept certain protocols, results from research into agent technology could improve the efficiency of negotiation by controlling the behaviour of negotiating partners by motivating them to cooperate. However, further research is needed to draw specific conclusions.
6. Conclusions

Determination of solution spaces shows much potential for supporting negotiation in collaboration through: providing representations that help avoid artificial conflicts, offering early detection of real conflicts and through improving understanding of multi-dimensional dependencies. Such support greatly reduces the risk of bad decision-making. Constraint satisfaction techniques provide a means to approximate solution spaces with tractable complexity. Eliminating variables from CSPs improves efficiency. Evaluation using a full-scale engineering project indicates that the current implementation of SpaceSolver provides support for data-management, visualisation, tradeoff analysis and interactive exploration of solution spaces.

Acknowledgements

The work described in this paper was funded by the Special Measures Program (SPP – IuK) of the Swiss National Science Foundation. The authors would like to thank Prof. G Schmitt, ETHZ and his team at CAAD for collaboration within the scope of this project. D. Sam-Haroud of EPFL-LIA performed fundamental constraints research and provided advice on improvement of her algorithms. We are also grateful to Mr Flückiger of Zwahlen & Mayr SA Aigle, Switzerland for proposing the example and to Mr Vögeli of TSW Engineering, Olten, Switzerland for supplying relevant details.

References


