Satisfying interaction constraints

Alex Noort, Rafael Bidarra, and Willem F. Bronsvoort

Computer Graphics and CAD/CAM Group
Faculty of Information Technology and Systems
Delft University of Technology
The Netherlands

A.Noort/R.Bidarra/W.F.Bronsvoort@its.tudelft.nl

Key words: Feature modelling, Constraint solving, Interaction constraints, Parameter space, Monte Carlo sampling

Abstract: In feature modelling, constraints can be used to store design intent in a model. Interaction constraints are an important type of constraints, which limit the extent to which features may interact. This paper describes a solver for constraints on interactions that involve spatial overlap between feature shapes. It is based on sampling of the parameter space of the feature model. A Monte Carlo technique is applied to reduce the expected number of samples that are needed to find a valid model.

1. INTRODUCTION

In feature modelling, features are used to model a product to be developed. During the process of developing the product, multiple feature models can be built, each model with features that have a specific meaning for a particular phase of the development process. For example, in the manufacturing planning phase, features represent regions of the geometry of the product that can be created by some manufacturing operation, and in the assembly planning phase, features represent regions of the geometry of the components that are involved in some connection. A more comprehensive description of feature modelling can be found in Shah and Mäntylä (1995).

Constraints play a major role in feature modelling, because they offer the possibility to store design intent in a feature model. Constraints can be specified on an individual feature of a feature model, where they specify a requirement on that feature, e.g. that the radius of the cylinder shape of a feature should be 5. Alternatively, constraints can be specified between several features of a feature model, where they specify a relation between those features, e.g. that the value of a parameter of one feature should be equal to the value of a parameter of another feature.

Many different types of constraints are used in feature modelling. In feature models for manufacturing planning, as an example, dimension constraints are used to specify that shapes with certain dimensions cannot be created by the available manufacturing equipment. In feature models for assembly planning, as an example, geometric constraints are used to specify the direction in which components can be moved during the assembly process in order to connect them.

Interaction constraints are a relatively new type of constraints, which limit the extent to which features may interact. In this paper, only the important interactions involving spatial overlap between the shapes of features are dealt with. An example of such an interaction is that a blind hole completely overlaps with a larger through slot, because then the blind hole would be absorbed by the through slot.

This paper will describe an automatic constraint satisfaction algorithm for interaction constraints, i.e. an algorithm that automatically adjusts a feature model to satisfy an unsatisfied interaction constraint, and its implementation in the SPIFF multiple-view feature modelling system (Bronsvoort et al. 1997). This system supports automatic checking and satisfying many types of constraints. For interaction constraints, however, up to now it only supported automatic checking, i.e. it could automatically determine whether a specific model satisfied an interaction constraints (Bidarra and Bronsvoort 2000), but it could not automatically adjust the model in order to satisfy an unsatisfied interaction constraint. The new algorithm has been used to extend the SPIFF modelling system with capabilities to also satisfy interaction constraints, but is more generally applicable.

Section 2 will describe the constraint model used here. Section 3 will give an overview of the interaction constraints supported by the algorithm. Section 4 will present different approaches to automatically satisfy interaction constraints. Section 5 will describe the implemented approach to solve interaction constraints. Section 6 will give some results and conclusions of the work that has been done.

2. CONSTRAINT MODEL

A feature model is built from features, and is represented at a high level by a so-called feature dependency graph, and at a lower level by a constraint model. In the feature dependency graph, the features are related by so-called dependency relations. Dependency relations arise, for example, when a feature element of an existing feature is used to attach or position another feature; see Figure 1. In this case, the new feature depends on the existing feature. Also all constraints between features, and their dependencies on these features, are represented in the dependency graph (Bidarra and Bronsvoort 2000)

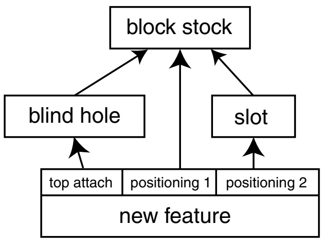

Figure 1: A new feature depends on existing features, because elements of these features are used to attach and position the new feature.

A constraint model is used to check whether the feature model satisfies the stored design intent. The constraint model is built from variables and constraints relating these variables.

Variables represent entities in the constraint model, and have a value that can be specified by constraints. Examples are geometric and algebraic variables.

Geometric variables represent the position and orientation of feature faces. They are related by geometric constraints, such as coincident, i.e. the position of the related variables should be the same, and in-plane, i.e. the position of one variable should be in a plane represented by the other variable.

Algebraic variables represent the real value of feature parameters, such as dimensions. They are related by algebraic constraints, such as equal, i.e. the real value of the related variables should be the same, and sum, i.e. the sum of the real values of two related variables should be equal to the third related variable.

Constraints are satisfied if the relation that they specify holds, and are unsatisfied otherwise. Unsatisfied constraints can be satisfied by changing the values of the variables on which they have been specified.

Solving a constraint model involves satisfying all constraints, and consists of two steps: checking whether the constraints are satisfied, and satisfying the constraints that have not yet been satisfied.

Checking a constraint involves performing a boolean test to determine whether the current values of its variables satisfy the relation that it specifies. For example, checking a coincident constraint between two geometric variables involves determining whether the two variables have the same position.

Satisfying a constraint involves adjusting the current values of its variables in such a way that the relation it specifies holds, while making sure that other, already satisfied, constraints do not become unsatisfied, and that other unsatisfied constraints can also be satisfied.

3. INTERACTION CONSTRAINTS

The notion of interaction among features refers to the influence they may exert on each other's functional meaning. Sometimes this influence may occur remotely, i.e. between non-overlapping features. For example, from a manufacturing point of view, two parallel slots might be so close that the wall between them would be damaged when the second slot is manufactured (Regli and Pratt 1996).

This work deals with the important type of feature interactions involving spatial overlap between shapes of features. Such interactions can have a wide range of effects on the features involved and, therefore, providing the possibility to disallow them is very important (Bidarra and Teixeira 1993). This is the role of interaction constraints.

Table 1 gives an overview of the types of interactions that can be disallowed by the interaction constraints available in the SPIFF modelling system (Bidarra and Bronsvoort 2000). For example, a splitting interaction constraint specifies that splitting interaction may not occur for some feature.

So far, interaction constraints have only been automatically checked. The interaction constraint checking mechanism of the SPIFF modelling system uses a non-manifold representation of the feature model geometry called cellular model, which integrates the contributions from all features.

The cellular model represents the model geometry as a set of quasi-disjoint cells of arbitrary shape, in such a way that each cell is either completely inside the shape of a feature or completely outside it. Intersections between features introduce additional cells. Each cell contains information on the features whose volume overlaps with the volume of the cell, and each cell face contains information on the feature faces that overlap with it. In addition, a cell also contains information on the fact whether its volume represents material, i.e. the cell has additive nature, or not, i.e. the cell has subtractive nature; a cell face also contains information on the fact whether it represents boundary of the feature model, i.e. has material on one side and no material on the other side, or not (Bidarra et al. 1998).

interaction type	description
splitting	splits the boundary of a feature into two (or more) subsets
disconnection	causes the volume of an additive feature (or part of it) to become disconnected from the model
boundary clearance	causes (partial) obstruction of a closure face of a subtractive feature
volume clearance	causes partial obstruction of the volume of a subtractive feature
closure	causes some subtractive feature volume(s) to become a closed void inside the model
absorption	causes a feature to cease completely its contribution to the model shape
geometric	causes a mismatch between a nominal parameter value and the actual feature geometry
transmutation	causes a feature instance to exhibit the shape imprint characteristic of another feature class
topologic	corresponds to the violation of a boundary constraint in a given feature

Table 1: The types of interactions that are dealt with; from Bidarra and Bronsvoort (2000).

The information in the cellular model can be used to check whether the shapes of two features overlap, and therefore to check interaction constraints. As an example, the checking algorithm for a volume clearance interaction constraint is given here. A description of the checking algorithms for all interaction constraints is given in (Bidarra 1999).

A subtractive feature has a volume clearance interaction with an additive feature, whenever a portion of its volume is obstructed by the additive feature. An example of a volume clearance interaction on a through slot feature, by a cylindrical protrusion, is given in Figure 2.

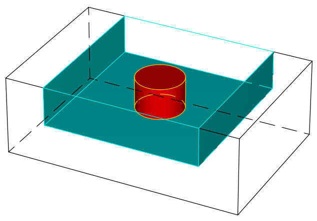

Figure 2: A volume clearance interaction on a through slot feature caused by a cylindrical protrusion.

The checking algorithm for this interaction constraint type determines whether all cells that overlap with the shape of the subtractive feature have subtractive nature. If so, the constraint is found to be satisfied, otherwise the constraint is found to be unsatisfied (see Algorithm 1).

Although an interaction constraint is specified on the shape of the constrained feature, it relates this shape to the shapes of several other features in the model. The actual number of features involved depends on the model structure, e.g. how the features in the model are positioned relative to each other, and it may even change after the constraint has been

```
foreach cell c in cellular model m do
    if subtractive feature s overlaps with c then
        if c has nature additive then
            return TRUE
return FALSE
```

Algorithm 1: Volume clearance interaction constraint checking algorithm.

created. This is one of the main difficulties of creating an interaction constraint satisfaction algorithm.

4. SATISFYING INTERACTION CONSTRAINTS

New types of constraints can be incorporated into a constraint model and satisfied in two ways: by extending the constraint satisfaction capabilities of the solver to support the new constraints, or by creating a conversion algorithm that is able to convert the new constraints into a set of constraints of an existing type that has the same meaning and can be satisfied. In the case of interaction constraints, an algorithm could be created to convert them into geometric constraints, for which an automatic constraint satisfaction algorithm is available, or a new constraint satisfaction algorithm could be created to directly satisfy the interaction constraints. Both approaches will be discussed here.

4.1 Conversion into geometric constraints

The interaction constraints dealt with here involve interactions between features that result from spatial overlap between their shapes. This involves the geometry of the model, and therefore using geometric constraints to represent constraints on such overlap seems to be a logical approach.

For example, in case of a boundary clearance interaction constraint on the top entrance-face of the blind hole in Figure 3, the position of the cylinder protrusion has to be constrained in such a way that its bottom face cannot overlap with the top entrance-face of the blind hole, i.e. the distance between the centerlines of the two features should be greater than the sum of their radii. In this example, a single geometric constraint is sufficient to represent this.

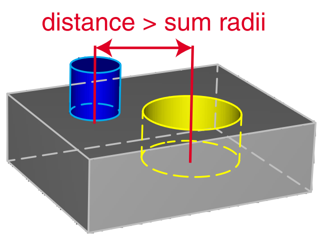

Figure 3: A boundary clearance interaction is represented by a geometric distance constraint between the centerlines of the blind hole and the cylinder protrusion.

However, a problem with this approach is that a single interaction constraint may need to be converted into an enormous number of constraints, which causes serious problems during constraint satisfaction. For example, if a through hole is attached to a surfaces on which *n* cylinder protrusions have been placed, then the boundary clearance interaction on the top entrance-face of the through hole needs to be converted into *n* geometric distance constraints.

Another problem is that additional geometric constraints need to be created for already converted interaction constraints when a new feature is added to the model. For example, when an additional cylinder protrusion is added to the model of Figure 3, an additional geometric constraint has to be added to the model to represent the previously specified interaction constraint.

Yet another problem is that it will be very difficult to create an algorithm that generates the correct set of geometric constraints for an interaction constraint in an arbitrary model. For example, the geometric constraints that have to be generated to avoid splitting interaction on a through hole feature depends on the fact whether the possibly interacting feature is a blind hole (Figure 4(a)) or a rectangular pocket (Figure 4(b)). In addition to the types of the involved features, also their relative position and orientation have influence on the geometric constraints to be generated.

In general, the number of geometric constraints that has to be created for an interaction constraint on a given feature, strongly depends on the number and types of features that could interact with it.

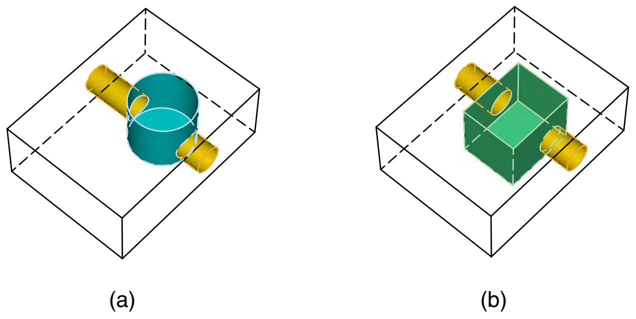

Figure 4: Avoiding splitting interaction on a through hole caused by a blind hole (a) requires other geometric constraints then when it is caused by a pocket (b).

The problem with the large number of geometric constraints and the creation of additional constraints could be solved by an incremental approach for the conversion of the interaction constraints into geometric constraints. This approach consists of repeatedly checking the model, creating geometric constraints for unsatisfied interaction constraints only, until no interaction constraints are unsatisfied. This results in no geometric constraints in case no interaction occures, but still in an enormous number of geometric constraints in case a lot of interactions occur in successive steps. However, an algorithm to create the correct set of geometric constraints for an arbitrary model is unfeasible.

4.2 A direct constraint satisfaction technique

To avoid the problems with conversion of interaction constraints into geometric constraints, a new direct constraint satisfaction algorithm for interaction constraints has been developed. It is based on sampling of the parameter space of the model.

The problem of finding a model in which certain features do not spatially overlap can be compared to spatial planning problems. A characteristic example of such a problem is to find room for another suitcase in the trunk of a car. The configuration space approach is a well-known approach to solve this class of problems (Lozano-Pérez 1983, Bowyer et al. 2000). The approach described here to satisfy interaction constraints is inspired by the configuration space approach, using the parameter space of the feature model instead of a configuration space.

This parameter space is a multi-dimensional space with one dimension for each variant feature parameter, i.e. parameter of a feature that may be changed by the modelling system because the actual value of the parameter does not matter to the user. Each point in the space represents an instance of a model that is derived from the original model, i.e. the model specified by the user, by changing one or more of its variant parameters. Some regions of the parameter space represent models that satisfy all constraints, other regions represent models that invalidate some of the constraints.

An example of the parameter space for a feature model will be given here. The feature model consists of a base block with a blind hole, a rib, and a cylinder protrusion. Due to interaction constraints, the entrance face of the blind hole should not be obstructed by any other feature, and the bottom faces of the rib and the cylinder protrusion should be completely attached to the faces they are currently attached to. In addition, the height parameter of the rib and one of the positioning parameters of the cylinder protrusion are variant (see Figure 5(a)). The constraints in this model can only be satisfied if the cylinder protrusion is positioned left of the blind hole, partially on the base block and partially on the rib (see Figure 5(b)). The parameter space for this model is two-dimensional (see Figure 5(c)). The region of the parameter space that represents models that satisfy all constraints is shaded.

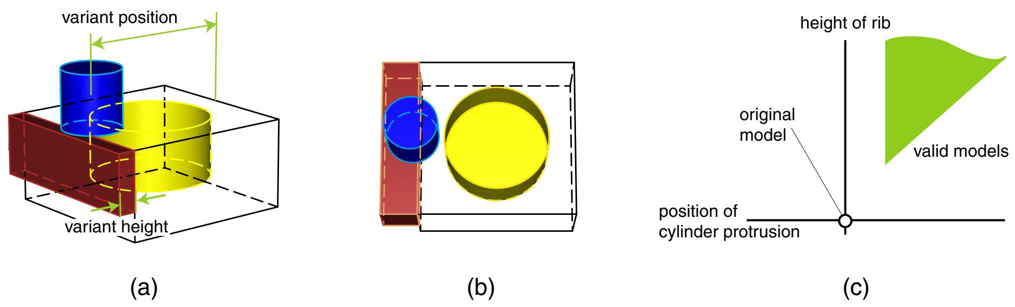

Figure 5: A feature model (a,b) and its parameter space (c).

In the direct constraint satisfaction technique, the parameter space of the feature model is sampled in order to find a model that satisfies all constraints, including the interaction constraints. For each sample, the geometric and algebraic constraints are solved using the available geometric (Kramer 1992) and algebraic (Sannella 1993) constraint solvers, and the interaction constraints are checked using the interaction constraint checking algorithm described in Section 3.

In order to increase the probability of finding a point that represents a model that satisfies all constraints, a Monte Carlo technique is used. Monte Carlo techniques, in general, reduce the number of samples needed to approximate a certain property with a certain accuracy, or increase the accuracy of the approximation with the same number of samples (Fishman 1996). The technique is used here to increase the probability of finding a model that satisfies all constraints, if such a model exists, within a certain number of samples. Samples are created in parameter space in the following way.

To prevent unnecessary changes to the model as much as possible, the parameter space is subdivided into sub-spaces, based on the extent of the changes to the model, with respect to the original model. The sub-spaces are ordered based on this extent, and on the probability that at least some of the represented models in the sub-spaces satisfy all constraints. An example of a parameter space of a model with its sub-spaces is given in Figure 6. Samples are generated randomly in subsequent sub-spaces, starting in the first sub-space and continuing in the other sub-spaces according to their order.

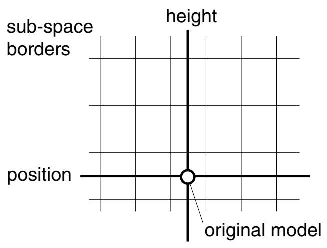

Figure 6: The parameter space of the feature model of Figure 5 divided into subspaces.

The probability that a sub-space represents at least some models that satisfy all constraints depends on the influence of the changed parameters on the unsatisfied constraint, which in turn depends on the meaning of the changed parameters. For example, the interaction of two feature shapes is more likely to be removed by changing the value of a parameter that specifies the relative position of the two features, than by changing the value of a parameter that has effect on a totally different region of the model. In Figure 6, it is more likely to remove the interaction by changing the position parameter than by changing the height parameter.

The sample process generates a specified number of random samples per sub-space, and after that continues with the next sub-space; see also Section 5.

5. INTERACTION CONSTRAINT SOLVER

This section describes the structure of the interaction constraint solver, and its implementation. The implementation will be described as far as it involves parameter space sampling.

5.1 Structure of the interaction constraint solver

The interaction constraint solver encapsulates the existing model validation process used when the user modifies the model, which consists of solving all algebraic and geometric constraints, evaluating the cellular model, and checking the interaction constraints (Bidarra and Bronsvoort 2000). In the new solver, if the model is found to be invalid and the user has not stopped the solver, a sample in the parameter space is generated and the model that is represented by this sample is validated, otherwise the interaction constraint solver stops (see Figure 7).

5.2 Implementation of the interaction constraint solver

The parameter space is implemented by a list of intervals for each dimension. In a preprocessing step, the parameter space is subdivided into sub-spaces by dividing each dimension into intervals with a fixed length for each dimension.

The order of the sub-spaces is determined by first ordering them to their distance to the point in parameter space that represents the original model, and subsequently ordering

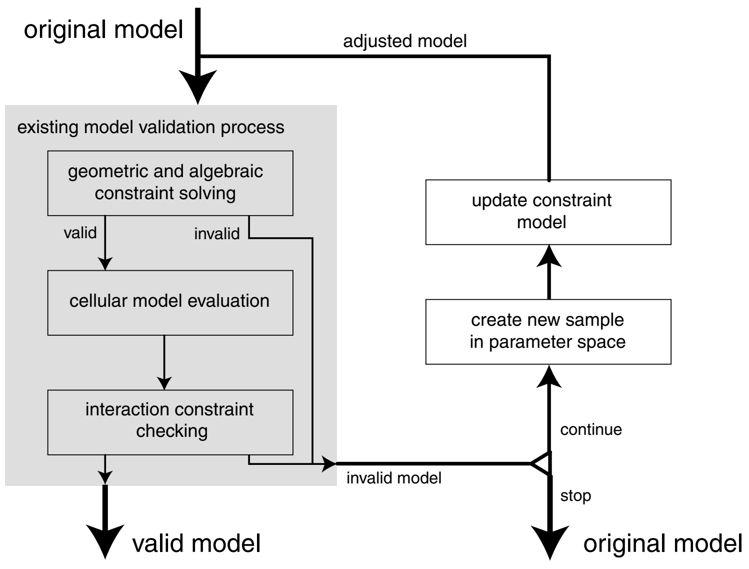
original model
adjusted model
existing model validation process
geometric and algebraic constraint solving
valid
invalid
cellular model evaluation
interaction constraint checking
update constraint model
create new sample in parameter space
continue
invalid model
stop
valid model
original model

Figure 7: The structure of the interaction constraint solver.

them based on the probability that at least some models exists in the sub-space that satisfy the constraints.

The distance of a sub-space to the point in parameter space that represents the original model is determined by grouping the sub-spaces into shells. The first shell consists of the sub-space in which the point that represents the original model lies, the second shell consists of the sub-spaces that enclose the first shell; see Figure 8. The distance of a sub-space to the point in parameter space that represents the original model is equal to the number of the shell in which the sub-space is.

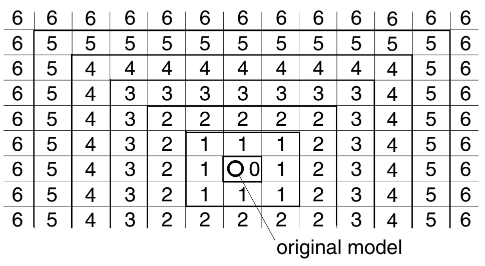

Figure 8: Shells of sub-spaces in a two-dimensional parameter space.

The probability that a sub-space represents at least some models that satisfy all constraints, depends on the meaning of the parameters that have changed with respect to the point that represents the original model. This probability is higher if the number of changed parameters that are parameters of features that position or dimension the interacting features is larger. The features that position or dimension the interacting features can be found in the feature dependency graph. For example, the features that position the blind hole feature and the protrusion feature of Figure 9 are the slot and the block stock feature.

A sample is created in a sub-space by generating a sample value in the related interval for each dimension. The combination of these sample values represents a sample point in the parameter space.

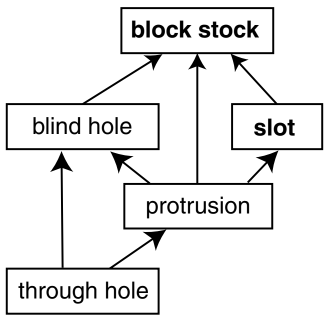

Figure 9: The features that position the protrusion and the blind hole features are denoted by their name in bold.

The Monte Carlo technique has been implemented by randomly generating the samples in a sub-space, based on a uniform distribution of the samples. According to the Monte Carlo concepts, this increases the probability that a valid model is found in a sub-space when such a model exists in that sub-space.

6. RESULTS AND CONCLUSIONS

An approach to automatically solve interaction constraints has been presented. This approach extends the current approaches for automatically checking interaction constraints, with the capability of automatically satisfying such a constraint, i.e. adjusting the model in such a way that it satisfies the constraint.

The proposed approach, based on sampling the parameter space of the model, is more suitable than the approach of converting the interaction constraints into geometric constraints. The main reasons for this are that conversion into geometric constraints generally results in an enormous number of geometric constraints, and that an algorithm that creates the correct geometric constraints for an arbitrary model is unfeasible.

An example of the use of the interaction constraint solver will be given here. Some remarks on the probability that the solver finds a solution for the interaction constraints, if one exists, and some directions for future research are also given.

The example of the use of the interaction constraint solver that is given here, is based on the product model of Figure 10, which has been taken from the “NIST Design, Planning and Assembly Repository” (Regli and Caines 1996). In this product, the length of the base-block feature, the depth of the through slot features, and the position of the stepped through-slot features are taken to be variant.

Now, a rounded through-pocket feature is added to the model between the two through holes, and is positioned according to the scheme of Figure 11(a). There is, however, not enough space between the two through hole features, and an interaction constraint on the rear through hole, requiring its side face to be completely on the boundary of the product, is invalidated (see Figure 11(a)).

The modelling system tries to automatically satisfy the invalid constraint by creating a parameter space for the variant parameters in the model, and subsequently sampling the parameter space until a model has been found that satisfies the constraints. One of the samples that is generated, before a valid model is found, results in a model that also invalidates interaction constraints on the front through hole feature; see Figure 11(b). The valid model that is finally found is shown in Figure 11(c).

In this example, only the variant length of the base block is adjusted in order to try to satisfy the invalid constraint. The variant depth of the through slots and the variant position

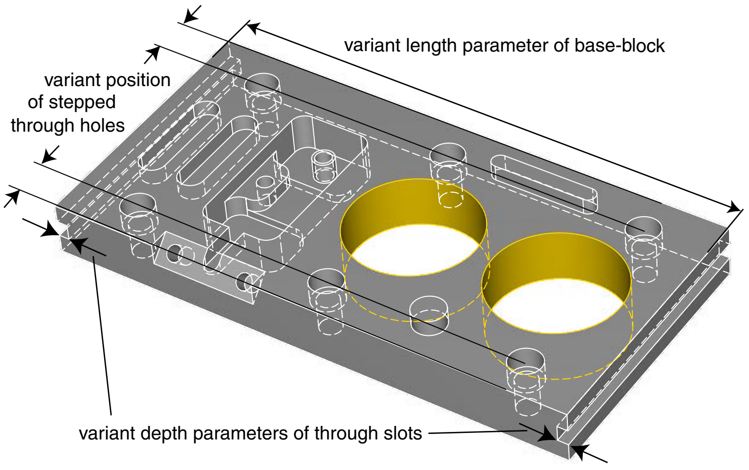
variant length parameter of base-block
variant position of stepped through holes
variant depth parameters of through slots

Figure 10: An example product.

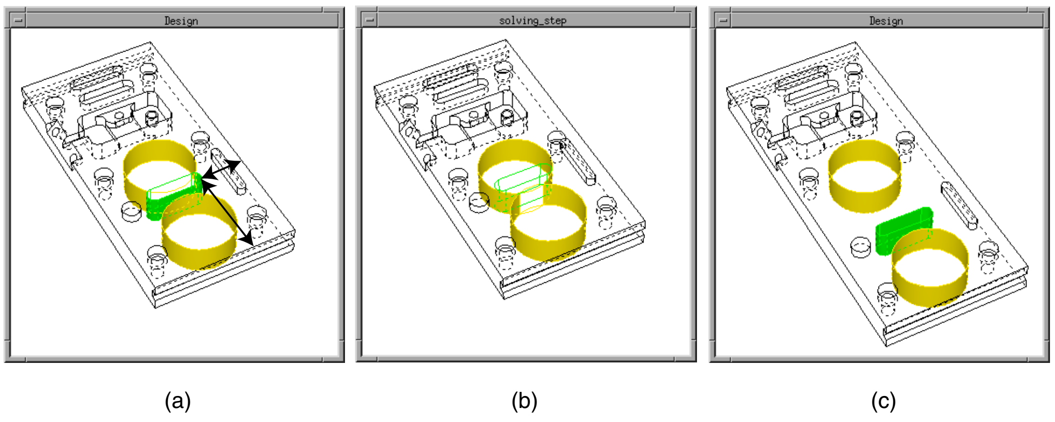

Figure 11: An invalid interaction constraint that results from adding a new feature (a) is automatically satisfied by the system by first trying, among others, the model of (b), and finally finding the model of (c).

of the stepped through holes do not influence the problem, and are therefore not changed by the interaction constraint solver.

The probability that the interaction constraint solver is able to solve the interaction constraints if such a solution exists, depends on the extent to which the interaction constraints restrict the possible shapes of the model. The more strict the interaction constraints specify the geometry of the model, the lower the probability that the constraint solver will be able to find a model that satisfies all interaction constraints.

A number of possible future extensions to the interaction constraint solver are described here.

Up to now, the interaction constraint solver does not take into account any constraint-specific information, such as the way the model should be adjusted in order to satisfy an unsatisfied interaction constraint of a given type, which makes it generic but also rather inefficient. Extending the interaction constraint solver to take into account constraint-specific information, could make it more efficient.

In addition, information from non-interaction constraints could be used to reduce the regions in parameter space in which samples are created. If a variant parameter of a feature has been constrained to be between 0 and 20, then it makes no sense to create samples outside the range $< 0, 20 >$ for that parameter, because these will surely result in a invalid model. The reduction of the regions would reduce the required number of samples.

Finally, the algorithm to determine the parameters that have the highest probability to result in a valid model when they are changed, can be enhanced. Up to now, all parameters of the features that relate the interacting features are assumed to have equal probability. However, some of the parameters of these features may not relate the position or dimension of the interacting features at all, and changing these parameters would never result in a valid model. The algorithm should be enhanced to determine exactly which parameters of the feature have high probability.

Altogether, it can be concluded that the parameter-space based approach to satisfy interaction constraints can be very useful in feature modelling.

ACKNOWLEDGEMENT

We thank Adrian Bowyer for showing us the way to parameter space.

REFERENCES

Bidarra, R. (1999). *Validity Maintenance in Semantic Feature Modelling*, PhD Thesis, Delft University of Technology.

Bidarra, R. and Bronsvoort, W. F. (2000). Semantic feature modelling, *Computer-Aided Design*, Vol. 32, No. 3, pp. 201–225.

Bidarra, R., de Kraker, K. J. and Bronsvoort, W. F. (1998). Representation and management of feature information in a cellular model, *Computer-Aided Design*, Vol. 30, No. 4, pp. 301–313.

Bidarra, R. and Teixeira, J. C. (1993). Intelligent form feature interaction management in a cellular modeling scheme, In Rossignac, J. R., Turner, J. and Allen, G. (eds): *Proceedings Second Symposium on Solid Modeling and CAD/CAM Applications*, ACM Press, pp. 483–485.

Bowyer, A., Eisenthal, D., Pidcock, D. and Wise, K. (2000). Configurations, constraints, and CSG, In *Proc. 1st Korea-UK Workshop on Geometric modelling & Computer Graphics*, pp. 27–34.

Bronsvoort, W. F., Bidarra, R., Dohmen, M., van Holland, W. and de Kraker, K. J. (1997). Multiple-view feature modelling and conversion, In Strasser, W., Klein, R. and Rau, R. (eds): *Geometric Modeling: Theory and Practice - The State of the Art*, Springer-Verlag, pp. 159–174.

Fishman, G. S. (1996). *Monte Carlo; Concepts, Algorithms and Applications*, Springer-Verlag.

Kramer, G. A. (1992). *Solving Geometric Constraint Systems: a Case Study in Kinematics*, The MIT Press.

Lozano-Pérez, T. (1983). Spatial planning: A configuration space approach, *IEEE Transactions on Computers*, Vol. C-32, No. 2, pp. 108–120.

Regli, W. C. and Caines, D. M. (1996). Catalog of the NIST design, planning, and assembly repository, Technical report, National Institute of Standards and Technology, Gaithersburg, MD 20899. http://www.parts.nist.gov/.

Regli, W. C. and Pratt, M. J. (1996). What are feature interactions?, In *CD-ROM Proceedings of the 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, ASME.

Sannella, M. (1993). The SkyBlue constraint solver and its applications, In *First workshop on principles and practice of constraint programming*.

Shah, J. J. and Mäntylä, M. (1995). *Parametric and Feature-based CAD/CAM*, John Wiley & Sons, Inc.