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
http://www.cg.its.tudelft.nl
A.Noort/R.Bidarra/W.F.Bronsvoort@its.tudelft.nl

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 searches a model in which such constraints are satisfied, by sampling the parameter space for the model. A Monte Carlo technique is applied to reduce the expected number of samples needed to find such a model.

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

1. INTRODUCTION

In feature modelling, features are used to model a product to be developed [Shah and Mäntylä, 1995]. Constraints play a major role in feature modelling, because they offer the possibility to store design intent in a model. Constraints can be specified on an individual feature in 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 in 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 only shapes with certain dimensions can 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. Interactions between features frequently represent conflicts with the design intent that is represented by the features, which should be avoided. Interaction constraints can be used to detect and avoid such interactions. In this paper, only the important class of interactions involving spatial overlap between the shapes of features is 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 is absorbed by the through slot.

This paper describes a 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]. SPIFF 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 determine whether a model satisfied an interaction constraint [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. It searches a model in which all interaction constraints are satisfied, by sampling the space that represents all variant parameters in the model. To reduce the expected number of samples needed to find such a model, a Monte Carlo technique is applied.

Section 2 describes the constraint model used here. Section 3 gives an overview of the interaction constraints supported by the algorithm. Section 4 presents different approaches to automatically satisfy interaction constraints. Section 5 describes the implemented approach. Section 6 gives 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 feature dependency graph, and at a lower level by a constraint model. In the feature dependency graph, the features are related by dependency relations. Dependency relations arise, for example, when feature elements of existing features are used to attach or position another feature; see Figure 1. In this case, the new feature depends on the existing features. Also all constraints between features, and their dependencies on these features, are represented in the dependency graph [Bidarra and Bronsvoort, 2000].

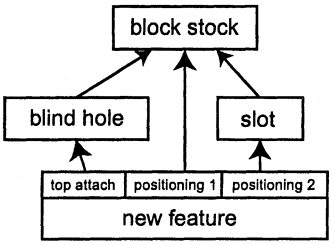

Figure 1. A new feature depends on existing features, because elements of the latter are used to attach and position the new feature

A constraint model is used to check whether the feature model satisfies the 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 *parallel*, i.e. the feature faces that are represented by the variables should be parallel, and *distance-face-face*, i.e. the feature faces that are represented by the variables should be at a given distance.

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

Constraints are satisfied if the relation that they specify holds, and are unsatisfied otherwise. Unsatisfied constraints can become 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 are not yet 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 parallel constraint between two feature faces involves determining whether the two faces have the same orientation.

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. In many situations, the influence is caused by a spatial overlap between the shapes of the features. For example, from a design point of view, two overlapping ribs interact if they were intended as cooling fins, because overlap between them reduces their surface and thus influences their cooling function. However, sometimes the influence may even occur between non-overlapping features. For example, from a manufacturing point of view, two parallel slots that do not overlap can still interact if the wall between them would be very thin and therefore be damaged when the second slot would be manufactured [Regli and Pratt, 1996].

This work deals with the important class of feature interactions involving spatial overlap between shapes of features, and each such overlap is said to be a feature interaction. 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 classes 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.

Topologic interaction corresponds to the violation of a boundary constraint on a feature face, and is always disallowed. Two types of boundary constraints exists; a constraint that requires a feature face, or at least part of it, to be on the boundary of the model, and a constraint that requires a feature face, or at least part of it, not to be on the boundary of the model.

So far, interaction constraints, i.e. constraints that disallow a particular class of interactions, could only be 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

interaction class	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 classes of interactions that are dealt with; from [Bidarra and Bronsvoort, 2000]

feature model, i.e. has material on one side and no material on the other side, or not [Bidarra et al., 1998].

The information in the cellular model can be used to check, among other things, 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 subtractive feature has a volume clearance interaction with an additive feature whenever part 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.

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).

A description of the checking algorithms for all interaction constraints is given in [Bidarra, 1999].

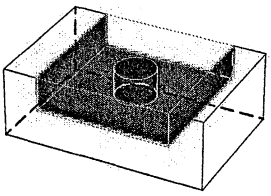

Figure 2. A volume clearance interaction on a through slot feature caused by a cylindrical protrusion

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

Algorithm 1: Volume clearance interaction constraint checking algorithm

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

4. SATISFYING INTERACTION CONSTRAINTS

A new type of constraints can be incorporated into a constraint model and satisfied in two ways: by developing a constraint conversion algorithm that is able to convert constraints of the new type into a set of constraints that together have the same meaning, consisting of types of constraints for which a constraint satisfaction algorithm is already available, or by developing an own constraint satisfaction algorithm for the new type of constraints. In the case of interaction constraints, an algorithm could be developed to convert them into geometric constraints, for which a constraint satisfaction algorithm is available, or an own constraint satisfaction algorithm could be developed to 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 center lines 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.

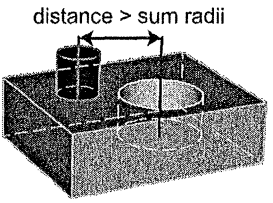

Figure 3. A boundary clearance interaction can be represented by a geometric distance constraint between the centerlines of the blind hole and the cylinder protrusion

However, a problem with this approach in general is that a single interaction constraint may need to be converted into a large number of geometric constraints, which causes serious problems during constraint satisfaction. For example, if in a model similar to that of Figure 3, but with n cylinder protrusions on top of the base block, a blind hole is attached to the top of the base block, then a boundary clearance interaction constraint on the top entrance-face of the blind hole needs to be converted into n geometric distance constraints.

Another problem is that additional geometric constraints may 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 on the blind hole.

Yet another problem is that it will be very difficult to develop an algorithm that generates the correct set of geometric constraints for an interaction constraint in any 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 influence the number and the types of geometric constraints to be generated.

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

The problem with the large number of geometric constraints and the creation of additional constraints could be solved by an incremental approach for

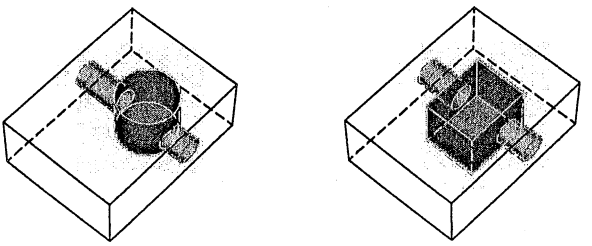

Figure 4. Avoiding splitting interaction on a through hole caused by a blind hole (a) requires other geometric constraints than avoiding the same interaction caused by a pocket (b)

the conversion of the interaction constraints into geometric constraints. Such an approach consists of repeatedly checking the model, creating geometric constraints for unsatisfied interaction constraints only, until all interaction constraints are satisfied. This results in no geometric constraints in case no interaction occurs, but still in a large number of geometric constraints in case several interactions occur in successive steps. However, an algorithm to create the correct set of geometric constraints for any model is infeasible.

4.2. A constraint satisfaction technique based on sampling

To avoid the problems with conversion of interaction constraints into geometric constraints, a new constraint satisfaction algorithm for interaction constraints has been developed. It is based on sampling the parameter space for the involved feature 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 for 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 is not critical to the user. Each point in the space represents 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 in which all constraints are satisfied, whereas other regions represent models in which some constraints are not satisfied.

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 cylinder protrusion should be completely attached with its bottom face. The height parameter of the rib and one of the positioning parameters of the cylinder protrusion are variant (see Figure 5(a)). The interaction 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)). In this example, the parameter space is two-dimensional (see Figure 5(c)). The region of the parameter space that represents models in which all constraints are satisfied, is shaded.

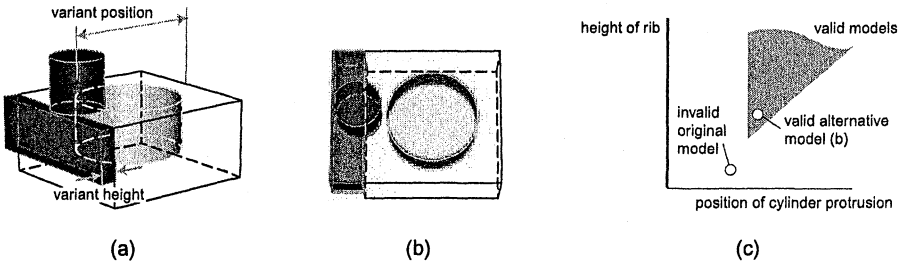

Figure 5. The invalid original feature model (a), a valid alternative model (b), and the parameter space (c)

In the new constraint satisfaction technique, the parameter space for the feature model is sampled to search a model that satisfies all constraints, including the interaction constraints. For each sample, the geometric and algebraic constraints are solved using the available geometric constraint solver [Kramer, 1992] and algebraic constraint solver [Sannella, 1993], and the interaction constraints are checked as described in Section 3.

In order to increase the probability of finding a point in the parameter space 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.

In a pre-processing phase, the parameter space is subdivided into sub-spaces, in order to prevent unnecessary changes to the original model as much as possible. The sub-spaces are ordered based on the extent of the changes to the model, and on the probability that at least some of the models represented in

the sub-spaces satisfy all constraints. The parameter space for the model of Figure 5(a) with its sub-spaces is given in Figure 6.

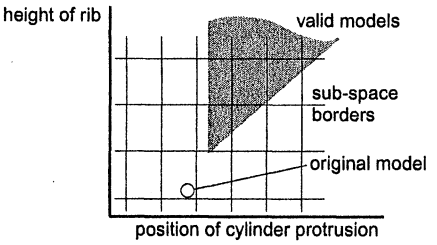

Figure 6. The parameter space of Figure 5(c) divided into subspaces

The probability that a sub-space represents at least some models that satisfy all constraints, depends on the relation between the parameters that have been changed from the original model and the unsatisfied constraint, and the extent in which these parameters have been changed. For example, the interaction of two feature shapes is more likely to be removed by changing the value of a parameter that directly specifies the relative position of the two features, than by changing the value of a parameter of another feature. In the model of Figure 5(a), it is more likely to remove the existing interaction by changing the position parameter than by changing the height parameter, although a change of the height parameter is also needed to obtain a valid model.

In the sampling phase, a specified number of 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, until a valid model has been found; see also Section 5.

5. INTERACTION CONSTRAINT SOLVER

This section describes the interaction constraint solving process and its implementation, the latter as far as it involves parameter space sampling.

5.1. The interaction constraint solving process

The interaction constraint solving process encapsulates the existing model validation process that is performed to validate a model after the user has modified it, and 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).

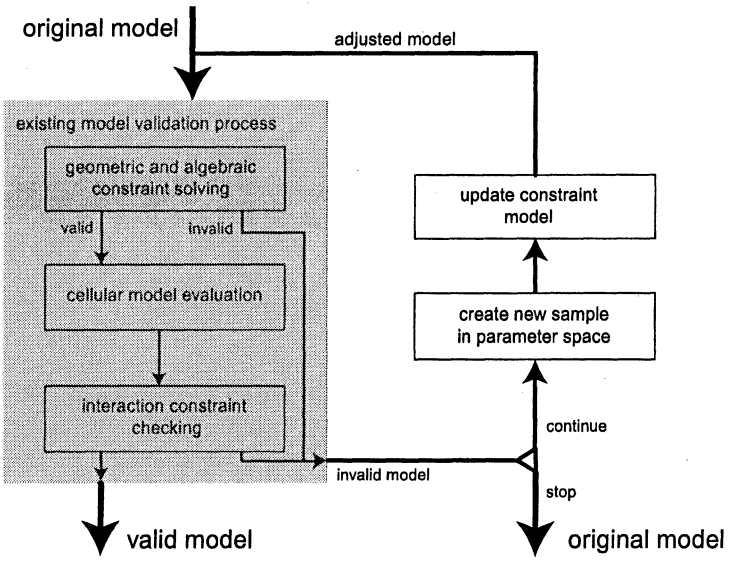

Figure 7. The interaction constraint solving process

5.2. Implementation of the interaction constraint solver

The parameter space is implemented by a list of intervals for each dimension. In the pre-processing phase, the parameter space is subdivided into sub-spaces by subdividing each dimension into intervals with a fixed length for each dimension.

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

As has already been described in the previous section, the probability that a sub-space represents at least some models that satisfy all constraints, depends on the changes of the parameters in that sub-space with respect to the original model. This probability is higher if the changes to the parameters of features that position or dimension the interacting features are larger. The relative position and the dimensions of the interacting features are determined by their own

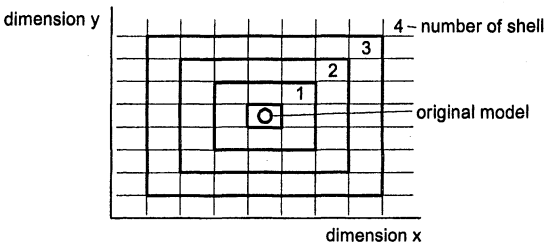

Figure 8. Shells of sub-spaces in a two-dimensional parameter space

parameters and by the parameters of the features on which they depend. The features on which a feature depends can be found in the feature dependency graph (see Section 2). For example, the features on which the blind hole feature and the protrusion feature of Figure 9 depend are the slot and the block stock feature.

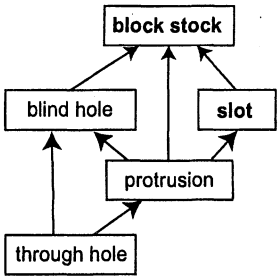

Figure 9. The features on which the protrusion and the blind hole features depend are denoted by their name in bold

An example of the use of the two criteria mentioned above to order the sub-spaces of the parameter space of Figure 6, belonging to the models of Figure 5, is given here. The dependency graph of this model, which is used to determine whether a variant parameter influences the relative position or one of the dimensions of the interacting features, is given in Figure 10(a). The variant height parameter of the rib does not influence the relative position or the dimensions of the interacting features, i.e. the cylinder protrusion and the blind hole, because these features do not depend on the rib. The variant position of the cylinder protrusion, on the other hand, does influence the relative position of the interacting features, because it is a parameter to position one of them. Based on this, the sub-spaces of the parameter space are ordered, starting with the shells close to the sub-space in which the point that represents the original model lies, and starting with the sub-spaces within these shells that represent a larger change to the variant position of the cylinder protrusion and a smaller

change to the variant height of the rib (see Figure 10(b)). This scheme is easily extendible to sub-spaces of parameter spaces with more than two dimensions.

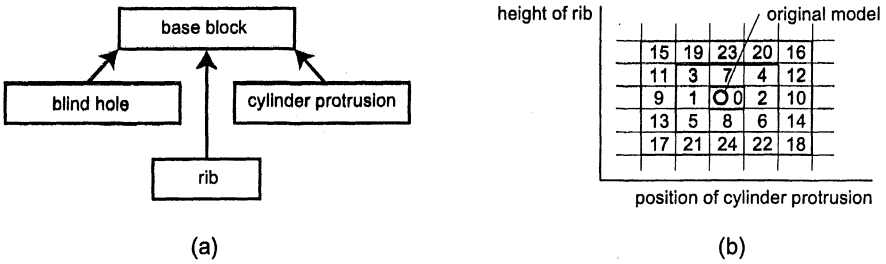

Figure 10. The dependency graph of the models of Figure 5, and ordering of the sub-spaces of the parameter space for this model

In each sub-space, a specified number of samples is created. A Monte Carlo technique has been implemented to randomly generate these samples, based on a uniform distribution. According to the Monte Carlo theory, this increases the probability that a valid model is found in the sub-space, if such a model exists in that sub-space.

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

6. RESULTS AND CONCLUSIONS

An approach to automatically solve interaction constraints in a feature model has been presented. It extends an approach 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.

An example of the use of the interaction constraint solver will now be given. It is based on the product model of Figure 11, which has been taken from the "NIST Design, Planning and Assembly Repository" [Regli and Gaines, 1997]. In this product, the length of the base-block feature, the depth of the through slot features, and the position of the stepped through hole features are taken to be variant.

Now, a rounded passage feature is added to the model between the two through holes, and is positioned according to the scheme of Figure 12(a). There is, however, not enough space between the two through hole features to accommodate the passage, and an interaction constraint on the rear through hole, requiring its side face to be completely on the boundary of the product, is now unsatisfied. However, because the rear through hole is positioned with respect to the back of the base block and the front through hole is positioned with re-

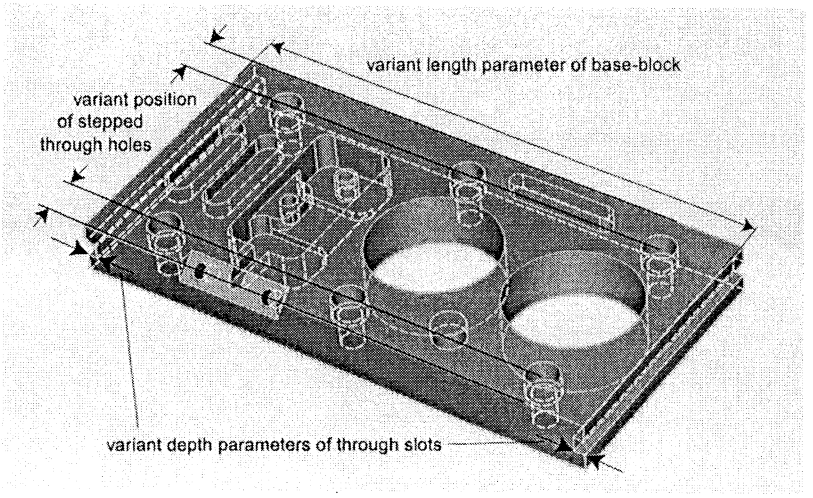

Figure 11. The example product

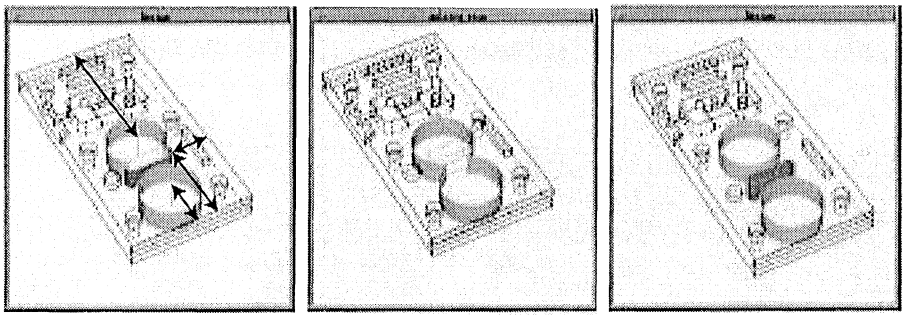

Figure 12. An unsatisfied interaction constraint that results from adding a new feature (a) is automatically satisfied by the system by first trying, among others, the model of (b), and eventually finding the model of (c)

spect to the front of the base block, the distance between the two through holes can be changed by changing the variant length of the base block.

The modelling system tries to automatically satisfy the unsatisfied 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 in which all constraints are satisfied. One of the samples that is generated, before a valid model is found, results in a model with an additional unsatisfied interaction constraint on the front through hole feature; see Figure 12(b). The valid model that is eventually found is shown in Figure 12(c).

Due to the subdivision of the parameter space into sub-spaces and the ordering of the sub-spaces, the changes to the variant parameter that do not influence

the relative position of the interacting through hole and passage, i.e. the variant depth of the through slots and the variant position of the stepped through holes, are limited. This limited disturbance property ensures that the resulting valid model meets the expectations of the designer.

A number of extensions to the interaction constraint solving approach are possible.

Sometimes, the geometry of the model that results from solving interaction constraints does not meet the expectations of the designer. This may happen if some requirements on the model are not yet represented by constraints. For these situations, the system should be extended to allow the designer to add more constraints to the model, thus further restricting the model, which then hopefully results in a model that does meet the expectations of the designer.

Another way to improve the chance that the geometry of the model that results from solving interaction constraints meets the expectations of the designer, is to include an objective function that is optimized by the interaction constraint solver. The solver would then not only find a solution that satisfies all interaction constraints, but even the solution that, according to the objective function, is closest to the original model.

Up to now, the interaction constraint solver does not take into account any constraint-type 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 $\langle 0, 20 \rangle$ for that parameter, because these will definitely result in an invalid model. Reduction of the regions would reduce the required number of samples.

It can be concluded that the proposed approach to solve interaction constraints, based on sampling the parameter space for a model, is better 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 a large number of geometric constraints, and that an algorithm that creates the correct geometric constraints for any model is infeasible.

The probability that the interaction constraint solver is able to solve interaction constraints if a solution exists, depends on the extent to which the interaction constraints restrict the possible shapes of the model. Obviously, the more tight 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.

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

Acknowledgment

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*, 32(3):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*, 30(4):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., editors, *Proceedings Second Symposium on Solid Modeling and CAD/CAM Applications*, pages 483–485, Montreal, Canada. ACM Press.

Bowyer, A., Eisenthal, D., Pidcock, D., and Wise, K. (2000). Configurations, constraints, and CSG. In *Proceedings First Korea-UK Workshop on Geometric Modelling and Computer Graphics*, pages 27–33, Seoul, Korea.

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., editors, *Geometric Modeling: Theory and Practice - The State of the Art*, pages 159–174. Springer-Verlag.

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*, C-32(2):108–120.

Regli, W. C. and Gaines, D. M. (1997). A repository for design, process planning and assembly. *Computer-Aided Design*, 29(12):895–905.

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.