Exploiting Degrees of Freedom Analysis for Interactive Constraint-based Design

Yung T. Tsai, Terrence Fernando and Peter M. Dew
School of Computer Studies
University of Leeds
Leeds LS2 9JT, U.K.
tsai,ltpf,dew@scs.leeds.ac.uk

Abstract

This paper presents an interactive constraint solver which exploits the degrees of freedom (DOFs) of under-constrained models to satisfy constraints and to support direct manipulation. The evolving constraints are maintained in a directed graph referred to as the Relationship Graph. A DOF technique referred to as allowable motion has been integrated with graph based algorithms to enable the constraint satisfaction, **incrementally**. Four basic constraint satisfaction methods (*basic allowable motion*, *locus analysis*, *inverse operation* and *hybrid method*) have been implemented which traverse the relationship graph to exploit the DOF of the under-constrained models to satisfy a given constraint. The direct manipulation of the under-constrained models is supported by combining the constraint satisfaction methods and graph re-writing techniques. These techniques handle **constraint loops** efficiently without resorting to time consuming numerical approaches.

Keywords: Degrees of Freedom Analysis, Interactive Constraint-based Geometric Design

1 Introduction

In recent years, there has been a considerable amount of interest in the development of constraint-based solid modelling systems. Several approaches have been used to implement these systems, such as the *numerical* approach[8, 11], the *algebraic* approach[7, 14, 12, 4], the *geometric reasoning* approach[3, 1, 9] and the *transformation operational* approach[13, 2, 6]. The important issues which need to be considered when developing a constraint-based solid modelling system are constraint representation, constraint satisfaction and user interaction.

Recently, degrees of freedom (DOF) analysis has been used as a means to satisfy constraints [2, 9, 10, 6] and to support direct interaction [6, 5] with constraint-based models. These approaches have combined specific geometric knowledge and DOF analysis to achieve higher computational advantages over conventional constraint solving approaches for supporting interactive constraint-based modelling. The DOF analysis can also support the realistic manipulation of under-constrained models by exploiting their DOFs and supports the kinematic simulation of assemblies.

However, several limitations are apparent within the current constraint solvers which use DOF analysis. For example, the DOF analysis reported in [9] does not support the direct manipulation of under-constrained models. The user needs to make sure the model is fully-constrained to support kinematic simulation. Whenever a new constraint is imposed, a new constraint satisfaction plan needs to be derived and the entire constraint set needs to be satisfied from scratch, hindering the interactive performance. Although the DOF mechanisms reported in [6] support direct manipulation, the user has to pre-plan the design sequence (order of constraint satisfaction and constraint dependencies) in advance. This imposes an additional burden on the designer when he or she is in the process of exploring the design alternatives. This work also does not address the mechanisms for handling constraint loops, restricting the domain of constraint models. The approach reported in [2] does not support direct manipulation of constrained models. In this approach, when a constraint loop is detected during propagation, all the previous operations are abandoned and constraints are transformed into a set of equations to be solved using iterative numerical techniques. This process can be extremely time consuming for complex constraint networks and is therefore not suitable for supporting interactive design. Further research work needs to be carried out to develop more efficient techniques for exploiting the full potential of DOFs analysis to support interactive constraint-based design.

This paper presents an incremental constraint solver referred to as FLEXI, based on *incremental DOF techniques*. These techniques support efficient constraint satisfaction and direct manipulation with constraint-based geometric models. A DOF technique referred to as allowable motion [6] has been integrated with graph based algorithms to support constraint satisfaction, incrementally. Four basic constraint satisfaction methods (*basic allowable motion*, *locus analysis*, *inverse operation* and *hybrid method*) have been implemented to traverse the current constraint graph incrementally, to exploit the DOF of the under-constrained models, to satisfy a given constraint. These methods can satisfy constraints within loops without resorting to time consuming numerical approaches. In this approach, the evolving constraints are maintained in a directed graph called a Relationship Graph. Constraint loops are detected during the constraint satisfaction and maintained within the relationship graph as macro nodes to support efficient constraint propagation. The direct manipulation of the under-constrained models is supported through the constraint satisfaction methods and graph re-writing techniques. The graph re-writing techniques dynamically change the constraint dependencies to *absorb* the user's manipulation according to the DOF of the models. This avoids the need to pre-plan the constraint dependencies for supporting realistic manipulation of models as in [6]. The techniques discussed in this paper build upon the existing research on DOFs to provide a framework for developing an highly interactive and efficient constraint solver for constraint-based design.

2 Terminology

This section identifies the terms used in this paper.

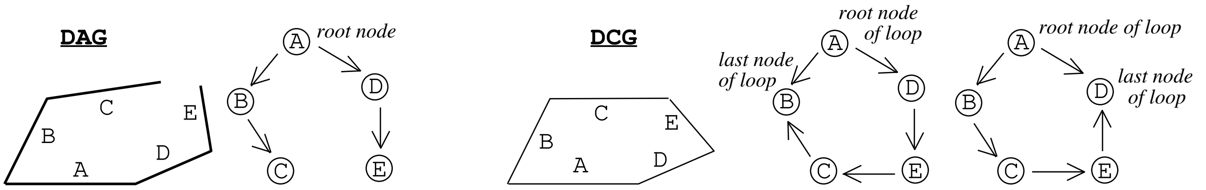
- Geometric Entity (GE): refers to a point, line or circle in 2D space.
- Geometric Constraint: refers to the geometric relationships between geometric entities. (At present, five kinds of geometric constraints are supported in the current FLEXI modeller, they are *point-point coincident*, *point-line coincident*, *circle-circle tangent*, *line-circle tangent* and *line-line angle* constraints.)
- Constraint Dependency: this means that one geometric entity is dependent on another geometric entity in terms of a constraint. The former is called the target geometric entity (Tar_GE) and the latter is called the reference geometric entity (Ref_GE). The node which represents the Ref_GE within the relationship graph is called the parent node and the node which represents the Tar_GE is called the child node.

3 Constraint Representation within FLEXI

The FLEXI solver uses a directed constraint graph referred to as the *Relationship Graph* (RG) to represent the constraints between geometric entities. Each node of the RG represents a geometric entity and each arc of the RG represents a constraint between two geometric entities. When a constraint is imposed between two geometric entities (i.e. Ref_GE and Tar_GE), an arc is created in the RG from the Ref_GE's node to the Tar_GE's node.

The relationship graph can take the form of a *directed acyclic graph* (DAG) or a *directed cyclic graph* (DCG). The relationship graph is considered to be a DAG if it contains no loops and is considered to be a DCG otherwise. These graph types are maintained in a *standard* format within the FLEXI solver to support loop detection, constraint satisfaction and direct manipulation. (This is explained in Sections 4 and 5).

The standard form of the directed acyclic graph is as shown in Figure 1(a). It takes the form of a tree, where the root node has no parents and all the other nodes have only one parent. The arcs always start from the parent node to the child node and it is possible to have more than one arc (constraint) between a child and its parent node. The standard form of the directed cyclic graph is as shown in Figure 1(b). It is possible to have more than one cyclic graph in a DCG. For each cyclic graph, there is one root node and one last node and the direction of arcs always starts from the root node and ends at the last node of the loop.



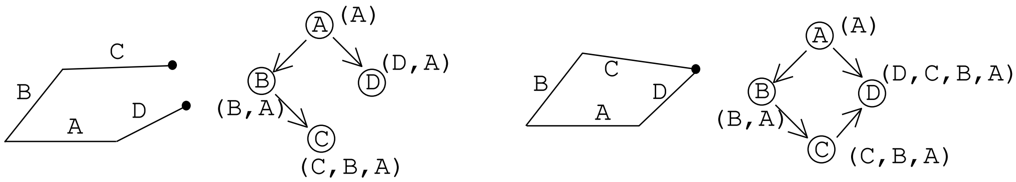
(a) standard form of a DAG

(b) standard form of a DCG

Figure 1: The Direct Acyclic and Directed Cyclic Graph of five Bar Model

3.1 Loop Detection and Macro Nodes

One of the merits of the FLEXI solver, as compared with other constraint solving systems, is its ability to handle constraint loops efficiently. Loops are detected by maintaining a dependency hierarchy list for each node in the RG. This list maintains the parent nodes that a node is dependent on. When a new constraint is imposed between two geometric entities, an intersection operation is performed between their dependency hierarchy lists to check for any loops. If the result is not empty, this implies that the two geometric entities have at least one common parent node in the RG and therefore form a loop. The first node in the resulting intersection list is considered as the root node of the loop. At this stage, FLEXI satisfies the new constraint by using the constraint satisfaction methods discussed later in Section 4.2. Finally the loop is transformed into the standard form as in Figure 1(b) and re-written into a macro-node. Multi-loops are supported in FLEXI by maintaining macro nodes within macro nodes (see Figure 13(b)). By re- writing the loops into macro nodes, FLEXI simplifies the RG into a DAG to achieve efficient direct manipulation support (This is explained in Section 5).



(a) initial dependencies

(b) loop detection

Figure 2: The constraint dependency and loop detection

The dependency hierarchy list and loop detection can be explained with an example. Figure 2(a) shows the dependency hierarchies for each node before a constraint is imposed between Bar C and Bar D. After the constraint is inserted, the intersection of C's and D's dependency lists is A. Therefore a new loop is formed and A is considered as the root node of the loop graph (Figure 2(b)). Another similar example is the seven bars model (see Figure 13(a),(b)). Once the constraints between the first four bars $(1, 2, 3, 4)$ are inserted into the system, FLEXI re-writes the first loop graph into a macro node M_1. Similarly once the constraints between the second set of nodes ($M_1, 5, 6, 7$) are inserted into the system, FLEXI re-writes the second loop graph into another macro node M_2.

4 Degrees of Freedom Techniques for Constraint Satisfaction

Within FLEXI, the satisfaction of a constraint is treated as the issue of planning a number of transformation operations (TOPs) and applying them to the geometric entities which are *affected* by this constraint. It uses a technique referred to as *allowable motion* to derive the degrees of freedom of the under-constrained models, from the existing constraints. Four constraint satisfaction methods (basic allowable motion method, locus analysis method[9], inverse operation method and hybrid method) have been implemented within FLEXI to satisfy constraints by exploiting the DOFs of under-constrained models.

The allowable motion technique is based on the work previously carried out at Leeds (see [6] for more detail). A brief description of how the allowable motions are derived for the constraint types supported within FLEXI is first described in Section 4.1. Section 4.2 then explains how allowable motion is exploited by the constraint satisfaction methods for performing constraint satisfaction.

4.1 Allowable Motion

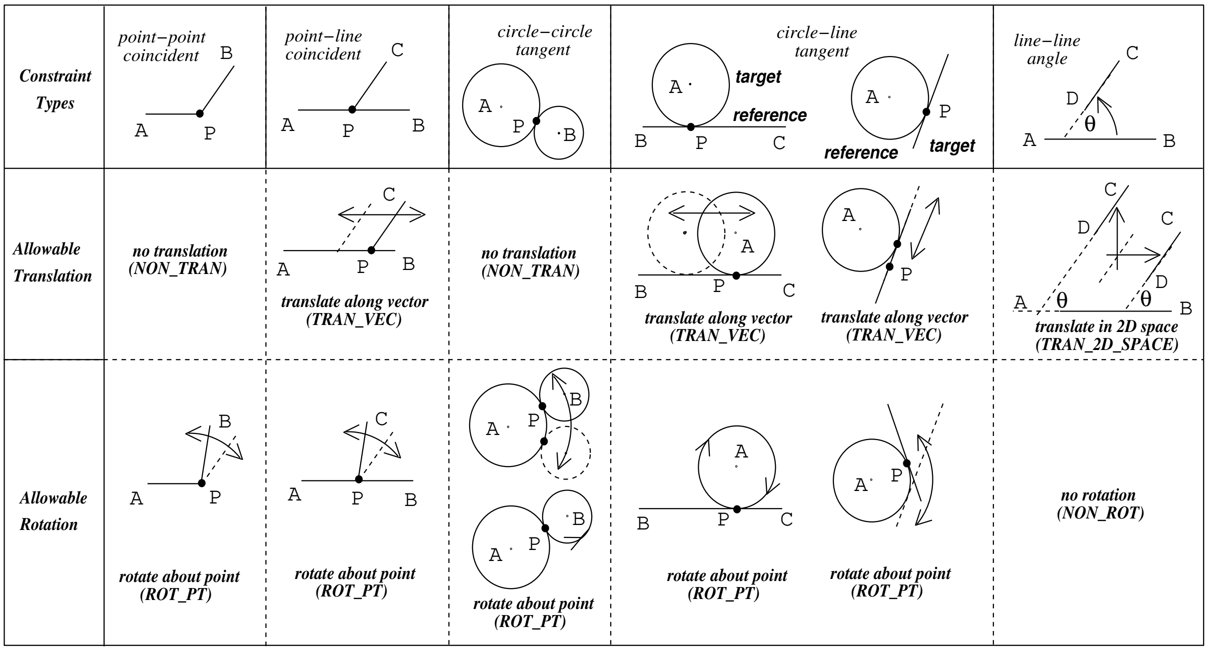


Figure 3: The allowable motion for single geometric constraint

Allowable Motion (AM) refers to the *Degree Of Freedom* (DOF) of a geometric entity. Allowable motion of a geometric entity is defined by two elements: (i) *motion type* which refers to either allowable translations or rotations; and (ii) *motion geometric element*, which defines the translation vector or the point of rotation. Figure 3 shows the allowable motions for the constraints supported by FLEXI. The abbreviations NON_TRAN, TRAN_VEC, TRAN_2D_SPACE and NON_ROT, ROT_PNT are used to represent the allowable translation and the allowable rotation for a given constraint (See Figure 3). The resulting allowable motion for several constraints is derived by intersecting the allowable translations and rotations for each constraint separately. Table 1 and Table 2 show the allowable motions as a result of intersecting two allowable rotations and two allowable translations. (In these tables, the abbreviation FIXED is used to represent fully-constrained GEs which have no allowable translation or rotation).

Resulting Allowable Translation	1st Allowable Translation			
2nd Allowable Translation	FIXED	NON_TRAN	TRAN_VEC	TRAN_2D_SPACE
FIXED	*FIXED*	*FIXED*	*FIXED*	*FIXED*
NON_TRAN	*FIXED*	*NON_TRAN*	*NON_TRAN*	*NON_TRAN*
TRAN_VEC	*FIXED*	*NON_TRAN*	See *Note*†	*TRAN_VEC*
TRAN_2D_SPACE	*FIXED*	*NON_TRAN*	*TRAN_VEC*	*TRAN_2D_SPACE*

†if the direction of the two translation vectors are parallel to each other, the result is *TRAN_VEC* otherwise the intersection is *NON_TRAN*.

Table 1. Intersection of 2D allowable translations

Resulting Allowable Rotation	1st Allowable Rotation		
2nd Allowable Rotation	FIXED	NON_ROT	ROT_PNT
FIXED	*FIXED*	*FIXED*	*FIXED*
NON_ROT	*FIXED*	*NON_ROT*	*NON_ROT*
ROT_PNT	*FIXED*	*NON_ROT*	See *Note*‡

‡ if the rotational points (motion geometric elements) of the two rotations are the same point, the result is *ROT_PNT*, otherwise the intersection is *NON_ROT*.

Table 2. Intersection of 2D allowable rotations

4.2 Constraint Satisfaction Methods

This section explains four constraint satisfaction methods which have been implemented within FLEXI, namely basic allowable motion method, locus analysis method, inverse operation method and hybrid method. The main strength of all of these methods is that they incrementally satisfy the new constraint by exploiting the allowable motions of the geometric entities in the relationship graph. These methods are capable of handling constraint loops without the help of a numeric solver.

4.2.1 Basic Allowable Motion Method

In this method, when a constraint is imposed between two geometric entities, a transformation operation (TOP) is derived by using the current allowable motion of the Tar_GE and the allowable motion of the new constraint. Once the constraint is satisfied, the new allowable motion of the Tar_GE is re-computed as explained in section 4.1. If the Tar_GE has dependent children, they are transformed together with the Tar_GE. Therefore the existing constraints are maintained without re-satisfying them from scratch.

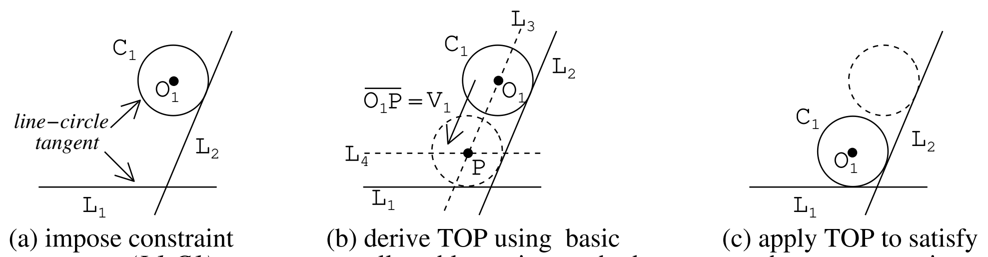


Figure 4: The basic allowable motion method for satisfying a constraint

For example, in Figure 4(a), C_1 is already constrained by the constraint *tangent*(line L_2, circle C_1). Therefore C_1's allowable translation is TRAN_VEC (translate along L_2) and allowable rotation is ROT_PNT (rotate about O_1). For the allowable translation, the *locus* of the centre point O_1 of circle C_1 is the line L3 which is parallel to L_2. Similarly, for the new constraint *tangent*(line L_1, circle C_1), the locus of O_1 is line L_4. The new location of O_1 which satisfies both constraints is computed by intersecting the two loci, L_3 and L_4. The TOP for this new location is derived using the translation vector between the initial and the new position of O_1. This TOP is then applied on the circle C_1 to find its new location (Figure 4(c)). Finally, the resulting allowable motion of C_1 is re-computed as explained in Section 4.1. In this example, the resulting allowable motion (see Table 1 and 2) is NON_TRAN and ROT_PNT which only allows C_1 to rotate about its centre point O_1. Since C_1 has no dependent children, no propagation is needed.

4.2.2 Locus Analysis Method

The *locus analysis method* (adopted from [9]) satisfies a constraint by deriving TOPs which are based on the existing allowable motions of both the Ref_GE and Tar_GE. Three steps are used in the locus analysis method to satisfy a constraint:

- derive loci for the Tar_GE and the Ref_GE according to their allowable motions;
- intersect the derived loci to find the intersection points and derive the transformation operations for both Ref_GE and Tar_GE from one of the intersection points. (the intersection point is selected to minimise the relative movement of geometric entities);
- apply the transformation operations on the Ref_GE and Tar_GE to move them to the intersected point to satisfy the constraint.

Once the constraint is satisfied and the allowable motion of the Tar_GE is updated, the Ref_GE and Tar_GE propagate the changes to their down-stream children to maintain all the other constraints in the system.

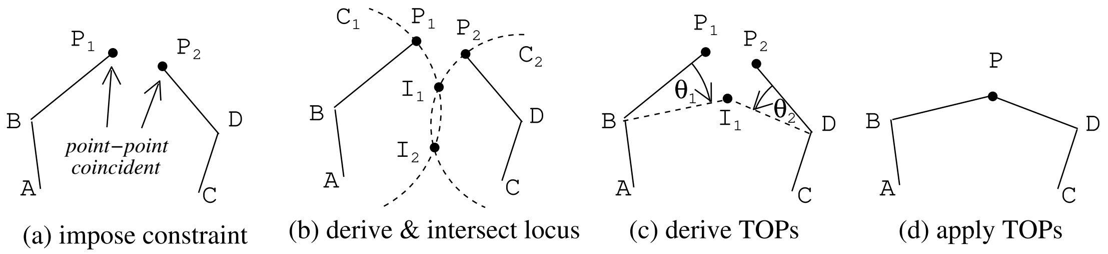


Figure 5: The locus analysis method for satisfying a constraint

For example, in Figure 5(a), when a constraint *coincident*(point P_1, point P_2) is imposed between the points P_1 and P_2, the locus analysis method is used to satisfy the constraint in three steps. Firstly, two loci circles C_1 and C_2 are derived from the allowable motions of line BP_1 and DP_2. Secondly, two intersection points I_1 and I_2 are found from C_1 and C_2, and I_1 is chosen since it is closer to the initial positions of P_1 and P_2 (Figure 5(b)). Finally, two transformation operations are derived which force the line BP_1 to rotate by θ_1 and the line DP_2 to rotate by θ_2 to satisfy the imposed constraint (Figure 5(c),(d)).

4.2.3 Inverse Operation Method

The *inverse operation method* satisfies a constraint by deriving the transformation operation using the allowable motions of the Tar_GE's parents and the allowable motion of the new constraint.

For example, in Figure 6(a), C_2 is chosen as the Tar_GE to satisfy the constraint *tangent*(circle C_1,circle C_2). Since C_2's and C_1's allowable motions only allow them to translate along L_2 and L_1, the basic allowable motion method and locus analysis method fail to satisfy the new constraint. Therefore FLEXI exploits the allowable motion of the Tar_GE's parent L_2

to satisfy the constraint, as follows. The current allowable motion of L_2 is TRAN_2D_SPACE, and hence the line O_2O_1 represents one of the possible motions for satisfying the new constraint. For the new constraint *tangent*(circle C_1,circle C_2), the allowable motion of C_2 is ROT_PNT about the centre point O_1 of circle C_1. For this allowable motion, the *locus* of the centre point O_2 of circle C_2 is circle C_3. Therefore, the new location of O_2 (point P) which satisfies both constraints is computed by intersecting the two loci, circle C_3 and line O_2O_1. The TOP for this new location is derived using the translation vector between O_2 and P (Figure 6(b)). This TOP is then applied on L_2 and C_2 to satisfy the new constraint (Figure 6(c)).

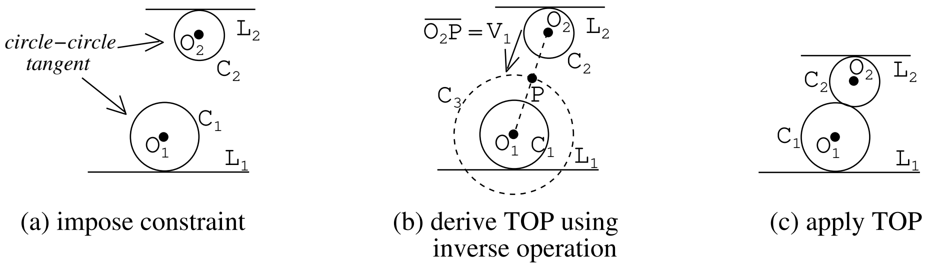


Figure 6: The inverse operation method for satisfying a constraint

4.2.4 Hybrid Methods

In certain cases, the above three methods have to be combined to satisfy a constraint. This can be explained with the example shown in Figure 7. In this case, line C is selected as the Tar_GE and line D as the Ref_GE to satisfy a *point-point coincident* constraint. This constraint can not be satisfied by any of the above constraint satisfaction methods. Therefore the following steps are taken:

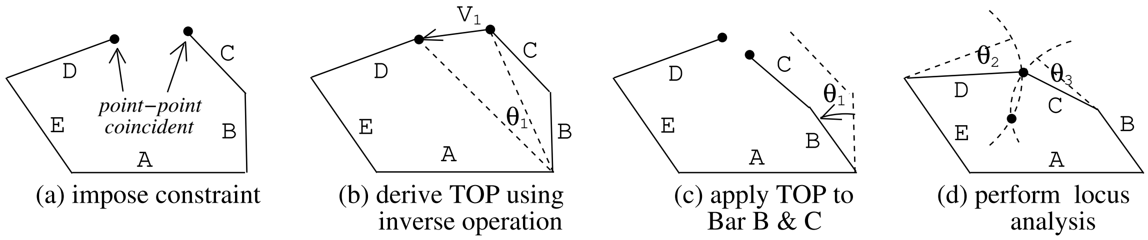


Figure 7: The hybrid method for satisfying a constraint

- The inverse operation method is applied on the Tar_GE's parent B to move the two points closer (Figure 7(b)).
- According to the translate vector V_1 and line B's allowable motion, a transformation operation is derived which forces line B and C to rotate by an angle θ_1 (Figure 7(b),(c)).
- The locus analysis method is invoked to satisfy the constraint. Two transformation operations are derived which force line C and line D to rotate by angles θ_2 and θ_3 to satisfy the constraint (Figure 7(d)).

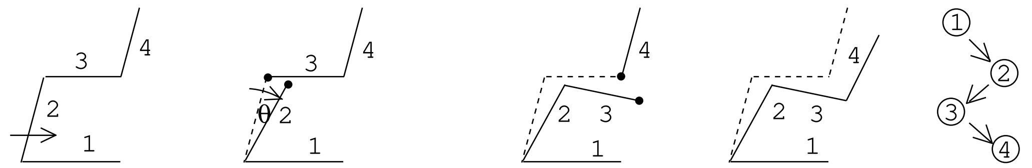
5 Supporting Direct Manipulation

The FLEXI solver supports direct manipulation techniques through the constraint satisfaction methods and graph re-writing techniques, enabling the user to exploit the remaining degrees of freedom of the under-constrained models. When the user manipulates a geometric entity, FLEXI first derives a TOP to satisfy the user's intention. This TOP is then applied (propagated) to the children nodes to maintain the existing constraints. The constraint propagation is dependent on the type of graph (i.e. DAG or DCG) within which the manipulated node is residing. This is described in the following sections.

5.1 Direct Manipulation Within a DAG

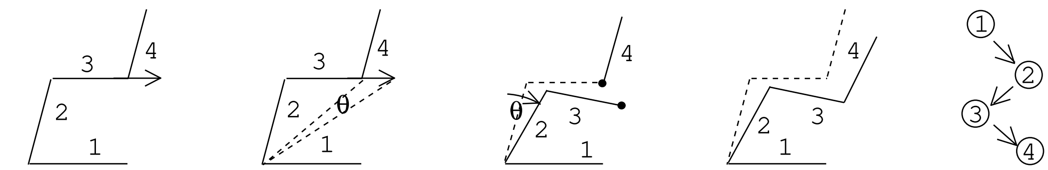
This is the case when the manipulated node is not inside a loop. In this case, the basic allowable motion method or the inverse operation method is used to support the direct manipulation, depending on the allowable motion of the manipulated node. This is explained with the following examples.

In Figure 8(a), when the user is manipulating Bar 2, the basic allowable motion method is used since the user's manipulation can be supported by the allowable motion of Bar 2. A TOP is derived which forces Bar 2 to rotate with angle θ to achieve the user's manipulation intention (Figure 8(b)). The change is then propagated to the children node by executing the same TOP as the manipulated node Bar 2 (Figure 8(c),(d)). If a child node is a loop macro node it is treated as a single node and the same TOP is applied to all the geometric entities inside the macro node. However, if the user applies the same direct manipulation to Bar 3 (Figure 9(a)), the basic allowable motion method fails since the allowable motion of Bar 3 prohibits it to move forward. In this case, the inverse operation method is invoked to support the manipulation by using the allowable motion of the parent Bar 2. A TOP is derived by using the allowable motion of Bar 2 which forces Bar 2 and Bar 3 to rotate to achieve the user's intention (Figure 9(b),(c)). Then a propagation is initiated from node 3 to its down-stream child node 4 as explained before (Figure 9(d)). It should be noted that if the user keep manipulating Bar 3 horizontally, Bar 3 will become horizontal by using the basic allowable motion method.



(a) manipulate Bar 2 (b) derive & apply Bar 2's TOP using basic allowable motion (c) apply Bar 2's TOP to Bar 3 (d) apply Bar 2's TOP to Bar 4 (e) DAG of constrained model

Figure 8: direct manipulation within a DAG – *basic allowable motion* method



(a) manipulate Bar 3 (b) derive TOPs using inverse op. (c) apply TOPs to Bar 2 & 3 (d) apply Bar 3's TOP to Bar 4 (e) DAG of model

Figure 9: direct manipulation within a DAG – *inverse operation* method

5.2 Direct Manipulation Within a Loop

This section explains how the direct manipulation is handled when the manipulated node is inside a loop. As in DAGs, first a TOP is derived to support the user's manipulation and then a constraint propagation sequence is derived to maintain the existing constraints. The main advantage of the techniques presented in this section is that they handle the under-constrained loops incrementally, without resorting to numerical approaches. Graph re-writing techniques are employed to support the constraint propagation within loops.

The method of supporting direct manipulation within a loop depends on the allowable motion of the manipulated node. The following three methods are supported within FLEXI to handle direct manipulation within a loop for all possible situations :

Method I: The basic allowable motion method is used if the direct manipulation can be supported purely by the allowable motion of the manipulated node. In this case, the TOP for the manipulation is derived using the allowable motion of the manipulated node which is then propagated to the children nodes.

For example, in Figure 10(a), when the user is manipulating Bar B, since the AM of Bar B is able to absorb the user's manipulation intention, the basic allowable motion method is invoked to derive a TOP which forces Bar B to rotate with angle θ to absorb the user's manipulation intention (Figure 10(b)). Then a sequence of propagations is initiated from the manipulated node (Bar B) to the children nodes to maintain the existing constraints. When a child node with a single parent receives a propagation, it maintains the propagated constraint by executing the same TOP as its parent. For example, since Bar C has only one direct parent (i.e. Bar B), it maintains the propagated constraint by executing the same TOP as Bar B. This change is then propagated to Bar D. When the propagation reaches the last node of the loop, which has more than one direct parent, the propagated constraint is broken and re-satisfied by using the constraint satisfaction methods discussed in Section 4.2. For example, since Bar D is the last node of loop in the above example, the constraint between Bar C and D are broken and re-satisfied using the locus analysis method (Figure 10(d)).

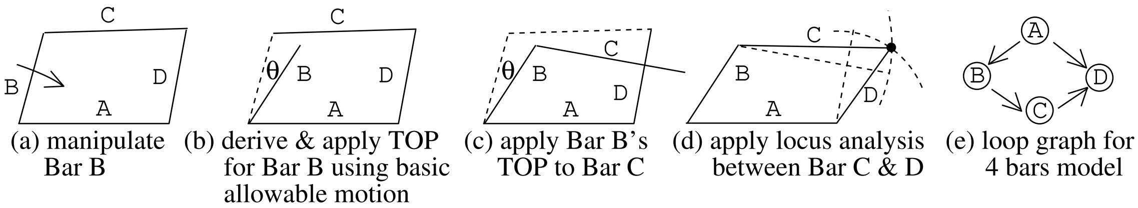


Figure 10: direct manipulation within a loop – *basic allowable motion* method

Method II: If the direct manipulation can not be handled by the allowable motion of the manipulated node, then the inverse operation method is used. In this case, the TOP for the manipulation is derived using the allowable motion of the parents of the manipulated node.

For example, in Figure 11(a), when the user is manipulating Bar C, the basic allowable motion method can not support the users manipulation intention. This is because, during the constraint propagation, when we attempt to re-satisfy the constraint between Bar C and Bar D using the locus analysis method, the two bars will always be located in their initial positions. Therefore the inverse operation method is invoked which uses the allowable motion of the parent node (i.e. Bar B) of the manipulated node. This derives a TOP which forces Bar B and Bar C to rotate with angle θ to achieve the direct manipulation (Figure 11(b),(c)). The TOP is then propagated to Bar C's child Bar D. Since Bar D is the last node of loop the propagated constraint is broken and re-satisfied using the locus analysis method (Figure 11(d)).

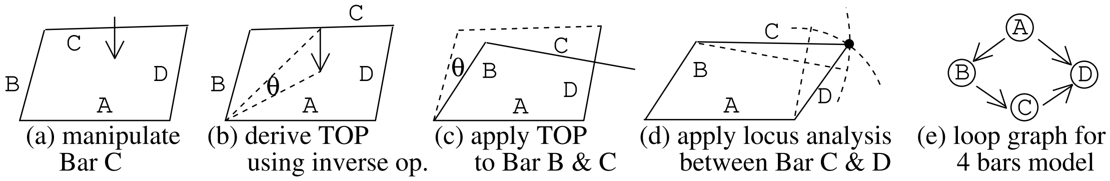


Figure 11: direct manipulation within a loop – *inverse operation* method

Method III: If the manipulated node is fixed and it is the last node of the loop, the constraint dependencies of the loop are reversed to gain allowable motion for the last node of the loop. The direct manipulation is then handled as in Method 1.

For example, in Figure 12(a), Bar D is the last node of the loop and it has no allowable motion to support direct manipulation. Therefore when the user is manipulating Bar D, the dependencies of the loop are reversed to gain allowable motion for the Bar D. This action releases the last node from being fully-constrained to under-constrained and hence gives allowable motion to Bar D to support direct manipulation (Figure 12(c)). The basic allowable motion method is then invoked to derive the TOP for the manipulated node. The propagation of constraints is then performed as in Method I.

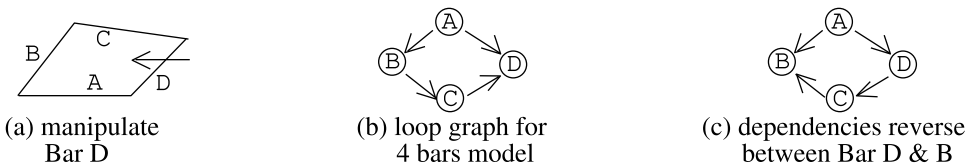


Figure 12: direct manipulation within a loop – *loop nodes dependency reverse* method

5.3 Direct Manipulation Within Multi-Loop Graphs

FLEXI solver also supports the manipulation of nodes within a multi-loop graph. This is explained with the example shown in Figure 13. When the user is manipulating Bar 2, FLEXI solver first handles the constraint propagation inside the macro node (M_1) where the manipulated node is residing. This is handled as explained in Section 5.2. The manipulation is then propagated to the outer macro nodes(i.e. M_2), through nodes M_1,5,6 and 7. This method is repeated until the outer most macro node is reached.

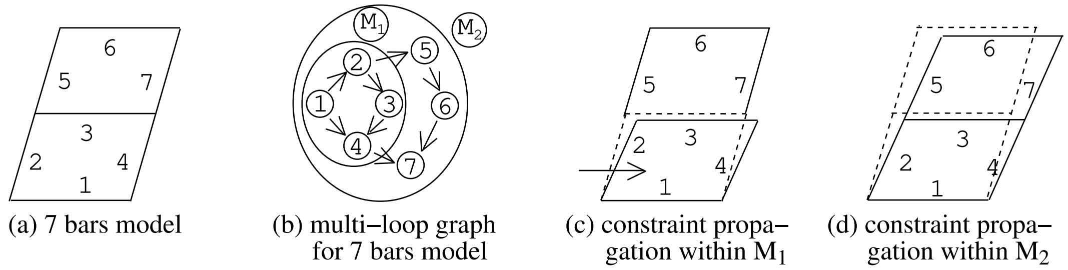


Figure 13: direct manipulation within multi-loops

6 Conclusion and Future Work

This paper presents a geometric constraint solver called FLEXI which exploits the DOFs of the under-constrained models to support an interactive constraint-based design environment. Four incremental constraint satisfaction methods have been implemented. By combining the constraint satisfaction techniques with graph re-writing techniques, FLEXI supports direct manipulation of the under-constrained models. These techniques handle constraint loops efficiently without resorting to time consuming numerical approaches.

In comparison with Kramer's GCE[9], FLEXI presents its advantages in the incremental nature and the ability to handle the under-constrained model and direct manipulation. Compared with Fa's work[6], FLEXI solver is more intuitive in that it frees the user from pre-defining the design procedure and supports a wider domain of constrained models (i.e. models with loops). FLEXI solver is a step forward from Arbab's OTP[2] approach in many aspects such as powerful constraint satisfaction methods, efficient constraint propagation strategy, efficient loop handling techniques and the ability to support the user's direct manipulation.

This paper has discussed the current progress with multi-loop graphs. The propagation of constraints within multi-loops can be complex depending on the complexity of the graph. Future work will focus on analysing these techniques further to support complex multi-loop graphs. In addition, debugging tools need to be developed to provide feedback to the user for handling contradicting or redundant constraints. The techniques discussed in this paper can support 2D geometric models which can be constructed by the geometric entities and constraints discussed in Section 4. In the future, these techniques will be incorporated within the existing 3D interactive constraint-based modeller [5] which is being developed in the University of Leeds.

7 Acknowledgements

The authors would like to thank Dr. M. Fa, E. Lamounier, A. Munlin, S. Carden and F. Gao for the regular discussion which led to inspiring ideas.

References

[1] B. Aldefeld. Variation of geometries based on a geometric-reasoning method. *CAD*, 20(3):117–126, April 1988.

[2] F. Arbab and B. Wang. A constraint-based design system based on operational transformation planning. In *Proc. of 4th International Conference on Applications of Articial Intelligence*, pages 405–428, Springer-Verlag, 1989.

[3] B. Bruderlin. Constructing three-dimensional geometric objects defined by constraints. In F. Crow and S. M. Pizer, editors, *1986 Workshop on Interactive 3D Graphics*, pages 111–130, Chapel Hill, N.C., October 1986. ACM Press.

[4] D-Cubed. The 2 dimensional dcm technical overview. Technical report, D-Cubed Ltd., July 1995.

[5] M. Fa. *Interactive Constraint-based Solid Modelling.* PhD thesis, The University of Leeds, September 1993.

[6] M. Fa, T. Fernando, and P. M. Dew. Interactive Constraint-based Solid Modelling using Allowable Motion. *Proc. of ACM/SIGGRAPH Symposium on Solid Modelling and Applications*, pages 243–252, May 1993.

[7] B. N. Freeman-Benson, J. Maloney, and A Borning. An incremental constraint solver. *Communication of the ACM*, 33(1):54–93, January 1990.

[8] R. C. Hillyard and I. C. Braid. Analysis of dimensions and tolerances in computer-aided mechanical design. *CAD*, 10(6):161–166, June 1978.

[9] G. A. Kramer. A geometric constraint engine. *Artificial Intelligence*, 58(1-3):327–360, 1992.

[10] G. A. Kramer. *Solving Geometric Constraint System – A Case Study in Kinematics.* The MIT Press, 1992.

[11] R. A. Light. Symbolic dimensioning in computer aided design. Master's thesis, Massachusetts Institute of Technology, 1980.

[12] J. C. Owen. Algebraic solution for geometry from dimensional constraints. In *Symposium on Solid Modelling Foundations and CAD/CAM Applications*, pages 397–407, Austin, Texas, June 1991. ACM SIGGRAPH.

[13] J. R. Rossignac. Constraints in constructive solid geometry. In F. Crow and S. M. Pizer, editors, *1986 Workshop on Interactive 3D Graphics*, pages 93–110, Chapel Hill, N.C., October 1986. ACM Press.

[14] D. Serrano and D. Gossard. Tools and techniques for conceptual design. *Artificial Intelligence in Engineering Design*, 1:71–116, 1992.