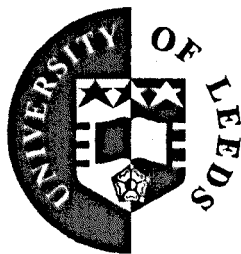


**An Incremental Constraint-based  
Approach to Support Engineering Design**

by

*Edgard Afonso Lamounier Junior*

Submitted in accordance with the requirements  
for the degree of Doctor of Philosophy.



**The University of Leeds  
School of Computer Studies**

**September 1996**

**The candidate confirms that the work submitted is his own and the  
appropriate credit has been given where reference has been made to the work  
of others.**

## Abstract

Constraint-based systems are increasingly being used to support the design of products. Several commercial design systems based on constraints allow the geometry of a product to be specified and modified in a more natural and efficient way. However, it is now widely recognised the needs to have a close coupling of *geometric constraints* (i.e. parallel, tangent, etc) and *engineering constraints* (i.e. performance, costs, weight, etc) to effectively support the preliminary design stages. This is an active research topic which is the subject of this thesis.

As the design evolves, the size of the equation set which captures the constraints can get very large depending on the complexity of the product being designed. These constraints are expected to be solved efficiently to guarantee immediate feedback to the designer. Such requirement is also necessary to support constraint-based design within Virtual Environments, where it is necessary to have interactive speed. However, the majority of constraint-based design systems re-satisfy all constraints from scratch after the insertion of a new design constraint. This process is time consuming and therefore hinders interactive design performance when dealing with large constraint sets.

This thesis reports research into the investigation of techniques to support interactive constraint-based design. The main focus of this work is on the development of **incremental graph-based algorithms** for satisfying a coupled set of engineering and geometric constraints. In this research, the design constraints, represented as simultaneous sets of linear and non-linear equations, are stored in a directed graph called *Equation Graph*. When a new constraint is imposed, **local constraint propagation** techniques are used to satisfy the constraint and update the current graph solution, incrementally. Constraint cycles are locally identified and satisfied within the Equation Graph. Therefore, these algorithms efficiently solve large constraint sets to support interactive design. Techniques to support under-constrained geometry are also considered in this research. The concept of **soft** constraints is introduced to represent the degrees of freedom of the geometric entities. This is used to allow the incremental satisfaction of newly imposed constraints by exploiting under-constrained space. These soft constraints are also used to support direct manipulation of under-constrained geometric entities, enabling the designers to test the kinematic behaviour of the current assembly. A prototype constraint-based design system has been developed to demonstrate the feasibility of these algorithms to support preliminary design.

*To*

*my mother, Anália*

*my wife, Eugênia*

*and my brothers Amadeu, Carmem and Ana.*

## Acknowledgements

I would like to express my sincere gratitude to my supervisors Prof. Peter M. Dew and Dr. Terrence Fernando for their invaluable advice and help throughout my research and the preparation of this thesis. This work would never have been finished without their patience and support.

The financial support from the Brazilian Agency for the Training of Higher Education Personnel (CAPES) is gratefully acknowledged.

I am especially thankful to Dr. Mingxian Fa for his initial ideas inspired from regular discussions. Thanks are also due to Dr. Mudarmeen Munlin, Yung-Teng Tsai, Steve Carden and Martin Thompson for many interesting debates and suggestions for improvement.

I would like to thank the following people for all their support and friendship throughout my studies at Leeds University: the staff in the General Office, Dr. Petros Mashwama, Elyas Nurgat, Nuha El-Khalili, Adriano Lopes, the CIS football members and specially to David Harkess for letting me play for the school football time.

My wife and I would like to thank Bryan and Pamela Marshall for making us feel at home and have such a pleasant time.

I had the opportunity to make a lot of Brazilian friends. It has been a great pleasure to meet such wonderful people and to have spent an enjoyable time with them. I am specially thankful to Alcimar and Tânia for their friendship despite their distance from Edinburgh.

I also would like to thank my colleagues Alexandre Cardoso, Ernane Coelho, Darizon Andrade and the DEENE staff, from the Federal University of Uberlândia, for their continuous support.

I believe in God and therefore I would like to thank and praise Him for numerous blessings during my stay in Leeds.

Finally, I am very grateful to my wife, Eugênia, for her help and understanding through difficult times. I am deeply thankful for her patience and love over the last months and years.

The publication list from this work is:

1. Lamounier, E., Fernando, T. and Dew, P.M., An Incremental Constraint Equation Solver for Variational Design, In *Proceedings of the Fourth International Conference on Computational Graphics and Visualization Techniques - COMPUGRAPHICS'95*, Santo, H.P. (Ed.), pp. 81-90, Alvor, Portugal, December 1995.
2. Lamounier, E., Incremental Constraint Equation Satisfaction, In *Proceedings of the Third Groningen International Information Technology Conference for Students - GRONICS'96*, Polak, I. (Ed.), pp. 33-37, Groningen, The Netherlands, February, 1996.
3. Lamounier, E., Fernando, T. and Dew, P.M., An Incremental Constraint Engine for Analysing Engineering Design, submitted to *Computer-Aided Design* in July 1996.
4. Lamounier, E., Fernando, T. and Dew, P.M., Incremental Constraint Satisfaction for Variational Design Systems, Technical Report 95.31, School of Computer Studies, University of Leeds, November 1995.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Objective and Research Goals . . . . .	2
1.3	The Engineering Design Process . . . . .	3
1.4	Design Constraints . . . . .	6
1.5	Constraint-based Design . . . . .	7
1.6	Thesis Organisation . . . . .	9
<b>2</b>	<b>Characterisation of Constraint-based Approaches</b>	<b>11</b>
2.1	Introduction . . . . .	11
2.2	Equation-based Approach . . . . .	12
2.2.1	Numeric Approach . . . . .	12
2.2.2	Symbolic Approach . . . . .	14
2.2.3	Graph-based Approach . . . . .	16
2.2.4	Graph-based Equation Solvers . . . . .	18
2.3	Geometric Constructive Approach . . . . .	28
2.3.1	Rule-based Approach . . . . .	29

2.3.2	Graph-based Approach . . . . .	31
2.4	Summary and Conclusions . . . . .	40
<b>3</b>	<b>Incremental Engineering Constraint Satisfaction</b>	<b>42</b>
3.1	Introduction . . . . .	42
3.2	Engineering Constraint Representation . . . . .	43
3.2.1	Basic Techniques . . . . .	43
3.2.2	Engineering Constraint Networks . . . . .	45
3.3	Incremental Equation Solver . . . . .	46
3.3.1	Inserting Engineering Constraints . . . . .	46
3.3.2	Engineering Constraint Cycles . . . . .	51
3.4	Summary . . . . .	55
<b>4</b>	<b>Engineering and Geometric Constraint Coupling</b>	<b>56</b>
4.1	Introduction . . . . .	56
4.2	Representation of Geometric Entities . . . . .	57
4.2.1	Line Segments . . . . .	57
4.2.2	Circles . . . . .	58
4.2.3	Arcs . . . . .	59
4.3	Geometric Constraint Representation . . . . .	60
4.4	Coupling of Design Constraints . . . . .	64
4.4.1	Coupled Constraint Network . . . . .	64
4.4.2	Composite Objects . . . . .	66
4.4.3	Inserting Geometric Constraints . . . . .	67

4.5	Direct Manipulations of Under-constrained Models . . . . .	68
4.5.1	Geometric Relationship Graph . . . . .	69
4.5.2	Direct Manipulations within a DAG . . . . .	70
4.5.3	Direct Manipulations within a DCG . . . . .	71
4.6	Summary . . . . .	72
<b>5</b>	<b>Implementation</b>	<b>73</b>
5.1	Introduction . . . . .	73
5.2	Overview of the Constraint Engine . . . . .	73
5.3	The Constraint Manager . . . . .	75
5.3.1	Implementation of the Equation Graph . . . . .	75
5.3.2	Constructing the Equation Graph . . . . .	76
5.3.3	Implementation of the Geometric Relationship Graph . . . . .	79
5.4	The Graphical User Interface . . . . .	81
5.4.1	Iris-Inventor Toolkit . . . . .	81
5.5	Interface Between the Constraint Manager and the GUI . . . . .	84
5.5.1	Creating a Geometric Entity . . . . .	84
5.5.2	Creating Variables and Engineering Constraints . . . . .	85
5.5.3	Imposing Geometric Constraints . . . . .	86
5.5.4	Direct Manipulation and Degrees of Freedom Inference . . . . .	87
5.6	Summary . . . . .	88
<b>6</b>	<b>Results, Discussions and Improvements</b>	<b>89</b>
6.1	Introduction . . . . .	89



6.2	Main Features of the Constraint Engine . . . . .	89
6.2.1	Case Study . . . . .	90
6.2.2	Incremental Nature of the Constraint Engine . . . . .	90
6.2.3	Ability to Handle Under-Constrained Geometry . . . . .	91
6.2.4	Coupling of Engineering and Geometric Constraints . . . . .	93
6.2.5	Direct Manipulations . . . . .	96
6.3	Comparison with Previous Work . . . . .	97
6.3.1	Efficiency in Solving Design Constraints . . . . .	98
6.3.2	Handling of Under-constrained Geometry . . . . .	98
6.4	Limitations and Improvements . . . . .	99
6.5	Summary . . . . .	102
<b>7</b>	<b>Conclusions and Future Work</b>	<b>103</b>
7.1	Introduction . . . . .	103
7.2	Conclusions . . . . .	103
7.3	Further Research . . . . .	105
7.3.1	Extension to 3D and VR Environments . . . . .	105
7.3.2	Feature-based Modeling . . . . .	106
7.3.3	Debugging Tools . . . . .	106
7.3.4	Inequality Constraints . . . . .	106
7.4	Concluding Remarks . . . . .	107

# List of Figures

1.1	The Product Design Process . . . . .	5
1.2	Geometric and Engineering Constraints . . . . .	6
2.1	Approaches for Constraint Solving in Design . . . . .	11
2.2	Geometric Construction for the Newton-Raphson Method . . . . .	13
2.3	Geometric Constraints for a Triangle Model . . . . .	15
2.4	Solution for the Triangle Model . . . . .	16
2.5	Undirected and Directed Graphs . . . . .	17
2.6	A Bipartite Graph and Two Maximum Matchings . . . . .	18
2.7	Undirected Equation Graph . . . . .	19
2.8	Maximum Matching and Directed Graph . . . . .	20
2.9	Tree-like Representation and a SC . . . . .	21
2.10	Topological Sorting in a Directed Acyclic Graph with SCs . . . . .	22
2.11	Two Triangular Systems . . . . .	23
2.12	A Constraint Cycle . . . . .	24
2.13	Multiple Output Methods . . . . .	24
2.14	Walkabout Strengths of Output Variables . . . . .	25
2.15	Constraint Insertion in DeltaBlue . . . . .	26

2.16	A Method Graph Partitioned into Constraint Cells . . . . .	28
2.17	Geometric Solution for a Symbolic Rule-based Problem . . . . .	30
2.18	A Geometric Constraint Problem and its Graph Representation . . . . .	31
2.19	Solution for the Constraint Problem of Figure 2.18 . . . . .	33
2.20	HyperGEM's Architecture for Constraint-driven Design . . . . .	34
2.21	Sketch Example and Corresponding Undirected Constraint Graph . . . . .	35
2.22	Cluster Formation from the Undirected Graph . . . . .	36
2.23	Clusters <b>V</b> and <b>W</b> of Figure 2.21 . . . . .	36
2.24	A Simple Dimensioned Drawing with Its Constraint Graph . . . . .	37
2.25	Constraint Graph Decomposition in DCM . . . . .	38
2.26	Integrating Geometric Constraints and Direct Manipulations in ICBSM . . . . .	40
3.1	Constraint Representation and Constraint Satisfaction . . . . .	43
3.2	Satisfaction Method Graphs for $c1 : a = b + c$ . . . . .	44
3.3	Number Attachment to Represent the Satisfaction Sequence . . . . .	44
3.4	Four-bar Linkage and a Solution Graph . . . . .	45
3.5	The Cantilever Beam Problem . . . . .	47
3.6	Inserted Free-variable Constraint and its Updated Equation Graph . . . . .	48
3.7	An Ancestor Subgraph . . . . .	49
3.8	Local Propagation During Solution Graph Construction . . . . .	50
3.9	Updated Solution Graph After $C7$ Insertion . . . . .	50
3.10	Current Equation Graph Before Fixing the Value of $K$ . . . . .	51
3.11	Constraint Cycle Updating . . . . .	52
3.12	A Maximum Matching Inside the MacroConstraint . . . . .	53

3.13	Current Solution After Fixing the Value of $K$ . . . . .	53
3.14	Downstream Constraint Propagation from a MacroConstraint . . . . .	54
4.1	Equation Representation for a Line Segment . . . . .	58
4.2	Equation Representation for an Arc . . . . .	59
4.3	Basic Geometric Entities and Their Characteristic Points . . . . .	60
4.4	Distance from Point to Line Segment . . . . .	61
4.5	Angle $\alpha$ between Line Segments . . . . .	62
4.6	Geometric Constraints Supported by the Constraint Engine . . . . .	63
4.7	Internal-Combustion Engine (Slider-Crank Mechanism) . . . . .	64
4.8	Coupling of Engineering Constraints and Geometric Constraints . . . . .	65
4.9	A Rectangular Shape as a Composite Object . . . . .	66
4.10	Partial Constraint Network for a Rectangular Shape . . . . .	67
4.11	Inserting a Coincident Constraint . . . . .	68
4.12	The Relationship Graph and the INCES's Underneath Constraint Network	69
4.13	Standard Form of a Directed Acyclic Graph . . . . .	70
4.14	Standard Form of a Directed Cyclic Graph . . . . .	71
4.15	Geometric Cycle Detection . . . . .	72
5.1	Constraint Engine Architecture . . . . .	74
5.2	Data Structure for a Constraint Equation Node . . . . .	75
5.3	Data Structure for a Variable Node . . . . .	76
5.4	Dynamic Updating of the Equation Graph - Constraint $C1$ . . . . .	77
5.5	Dynamic Updating of the Equation Graph - Constraint $C2$ . . . . .	77

5.6	Dynamic Updating of the Equation Graph - Constraint $C3$ . . . . .	78
5.7	Dynamic Updating of the Equation Graph - Constraint $C4$ . . . . .	79
5.8	Data Structure for a Geometric Constraint Edge . . . . .	80
5.9	Data Structure for a Node in the Relationship Graph . . . . .	80
5.10	A Graphical Scene Data Base . . . . .	82
5.11	A Snapshot of the Graphical User Interface . . . . .	83
5.12	Implementation Steps to Create Geometric Entities . . . . .	84
5.13	Implementation Steps to Create Variables and Engineering Constraints . . . . .	85
5.14	Implementation Steps to Impose Geometric Constraints . . . . .	87
6.1	Copy of Figure 4.7 . . . . .	90
6.2	(b) Plate After Imposing a Coincident Constraint - Crank $a$ and Connecting Rod $b$ . . . . .	92
6.3	Updating of the Equation Graph . . . . .	93
6.4	(b) Plate After the Satisfaction of Engineering Constraints . . . . .	94
6.5	Updating of Geometry through Engineering Constraint Satisfaction . . . . .	95
6.6	Frame Showing the Simulation of the Internal-combustion Engine . . . . .	96
6.7	Another Frame of Simulation of the Internal-combustion Engine . . . . .	96
6.8	A Geometric Constraint Problem . . . . .	100
6.9	Choosing a Specific <i>Soft</i> Constraint . . . . .	101
7.1	Constraint Conflict with Inequality . . . . .	107

# List of Tables

4.1	Design Constraints for the Internal-Combustion Engine . . . . .	65
6.1	Visited Percentage and Timing During Constraint Insertion . . . . .	91
6.2	Capabilities of Known Constraint-based Systems . . . . .	97

# Chapter 1

## Introduction

---

### 1.1 Motivation

In the early design phases of a product the designer needs freedom to explore design alternatives. A large proportion of the cost of a product is decided at the early design stages [84] and therefore considerable saving can be made if the design decisions are sound as possible [38]. In addition, increasing world-wide competition is imposing pressures on the designer to reduce both the costs and the time it takes to market. Therefore, it is important to provide design analysis tools that support timely and accurate information upon which designers may enhance their ability to make quickly quality decisions. However, traditional CAD tools are inadequate for supporting the early design stages, since they only supply tools for developing detailed geometry [67]. Non-geometric information, such as the weight, forces, stress and cost of a product, which are essential considerations during the initial design stages [115], are not well supported.

In recent years, constraint-based design has emerged as a design methodology for providing the decision support required by engineers [22]. Today, this approach is present in most commercial systems such as PTC's Pro-Engineer [24], D-Cubed's DCM [74] and SDRC's I-DEAS [92]. It is noted that there has been an increasing use of these commercial systems in industry [49, 78]. Early attempts in geometric modeling required an object to be defined in terms of absolute positions and dimension of its constituent geometric elements [18]. In this approach, considerable algebraic manipulation is required to determine the values

and positions of these dimensions, restricting the designer's creative methods. In contrast, the definition in terms of constraints allows the geometry of a product to be specified and modified in a more natural and efficient way [87]. Such facilities are very important in the preliminary design stages where the engineer iteratively analyses different design alternatives until design specifications are satisfied. Therefore, the constraint-based paradigm is seen as a strong candidate to support early design stages since it provides flexibility for creativity.

In a typical preliminary design scenario, an engineer specifies design considerations in terms of both geometric constraints (such as *parallel*, *tangent*, etc.) and engineering constraints (expressed in terms of engineering equations representing performance, costs, weight, moment of inertia, etc.). As Chung and Schussel pointed out in [23], design tools that allow the coupling of both geometric constraints and engineering constraints are necessary for supporting the analysis of preliminary design. Equally, at this stage, the designer is interested in exploring different *what if* scenarios by changing design parameters and constraints and also interested in testing the kinematic behaviour of the current assembly and to analyse the geometric profiles (e.g. the cutting envelope described by a backhoe).

As the design evolves, the size of the equation set which defines the design can get very large depending on the complexity of the product being designed. In addition, constraint cycles are constantly emerging as a result of newly imposed constraints by the user [75]. These constraints must be solved efficiently in order to guarantee immediate feedback to the designer. Although a number of constraint-based systems have been discussed in the literature, the majority of these systems re-satisfy all constraints from scratch due to the insertion of a new design constraint [14, 18, 43, 60, 61, 74, 93]. This process is time consuming and therefore hinders an interactive design performance when dealing with large constraint sets [117, 33]. In addition, these systems require the designer to make sure the design is fully constrained. This could obstruct the designer's creativity in testing different design alternatives by exploiting under-constrained space.

## 1.2 Objective and Research Goals

The objective of this research is to study **efficient algorithms for interactive constraint-based design that combines both engineering and geometric constrains.**

The following research goals have been identified in order to accomplish the above objective:



1. to develop a taxonomy to classify current constraint-based approaches in order to identify advantages and disadvantages of each approach;
2. to derive incremental constraint-based algorithms that can speed up the constraint satisfaction and propagation process;
3. to investigate techniques that can support engineers to explore under-constrained models;
4. to demonstrate the feasibility of these algorithms through a prototype system and case studies.

### 1.3 The Engineering Design Process

The process of designing a product is constraint-oriented since most of this process involves the recognition, formulation and satisfaction of constraints arising from diverse engineering and management areas [33, 67, 92]. Although it can vary from one terminology to another [93], according to French [35], Pahl and Beitz [76] and Thornton [109], most engineering design processes can be decomposed into the following major stages: *clarification of the task*, *conceptual design*, *embodiment design* and *detail design*.

#### Clarification of the Task

At this stage, design specifications for the product are identified according to the client's needs. The main result of this process is a requirement list describing the general properties and conditions (i.e. constraints) that the product should satisfy. These include: marketability, costs, performance, safety, assembly, maintenance etc. Some of those constraints could also be expressed as mathematical equations such as: *product unit*  $\leq x$  *pounds*. Although the requirement list can change throughout the design process, evolving designs are constantly checked against this list in order to avoid expensive errors in later stages of the design [109].

## Conceptual Design

Conceptual design starts with a problem statement in the form of the requirement list and outcomes a set of preliminary solutions in the form of concepts. First, a relationship between the inputs and outputs of the product is derived. This relationship is called *overall function*. Depending upon the complexity of the design, the overall function can be broken down into simpler sub-functions (**function structures**). This facilitates the search for engineering principles as solutions to fulfill each sub-function. Next, these solution principles are combined to fulfill the overall function. Suitable combinations are then selected and firmed up into *concept variants* which, in turn are evaluated against the demands of the requirement list. On the basis of this evaluation, the *best solution concepts* are selected for further refinements. During this stage, *abstract geometry* (incomplete geometric information that is subject to change at later stages) is often used.

## Embodiment Design

During this stage, the designer gives geometric forms for the design concepts. It is often necessary to produce and evaluate several layout drawings until a *definitive layout* is determined in order to develop the product according to technical and economic criteria. Preliminary engineering functions such as tolerance analysis and design optimisation are often performed as corrective steps. According to Pahl and Beitz [76], “the definitive layout must be developed to the point where a clear *check* of function, durability, production, assembly, operation and cost can be carried out”.

## Detail Design

At the *detail design* stage, individual geometric elements of the definitive layout are refined with exact dimensions and tolerances and complete details of the geometric shape of the design object are specified for production. The main purpose of this stage is to assure the designer that the design object can be manufactured and that it will work.

## Summary of Design Stages

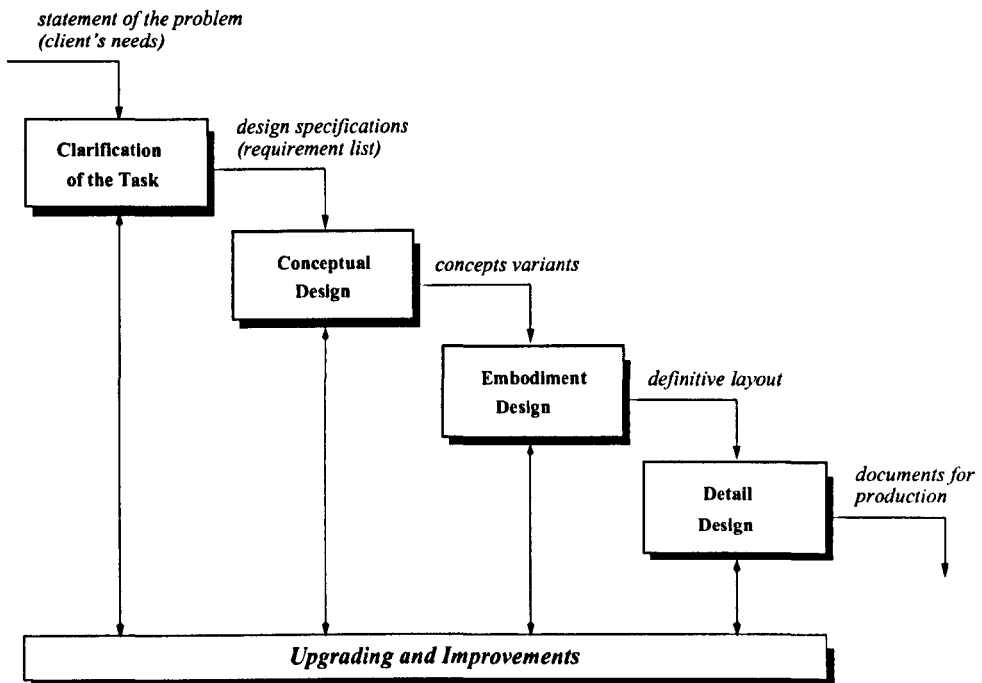


Figure 1.1: The Product Design Process

Engineering design is an iterative process. According to the inserted constraint, different stages interact with each other for upgrading and improvement. Figure 1.1 shows the flow of work during the design process. At every stage, decisions must be made to identify whether the next stage should be carried out or previous stages must first be repeated. The figure shows the main inputs and outputs for each design stage.

In addition, design constraints are often numerous, complex and conflicting since they are continuously being inserted, deleted and modified throughout the design process. This fact reflects the need to evaluate different design alternatives in order to produce the best possible solution. Therefore, constraint management tools should be provided to enable engineers to manipulate and control this “up and down” constraint flow during the iterative design cycle. Such tools are being developed as essential elements to build concurrent engineering environments that support the concurrent interaction among all stages of design [52, 69, 84]. The next section therefore describes current constraint-based techniques that can be used as a computer-based aid tools for the design process.

## 1.4 Design Constraints

As mentioned in Section 1.1, during preliminary design, a design is expressed in terms of geometric constraints and engineering constraints.

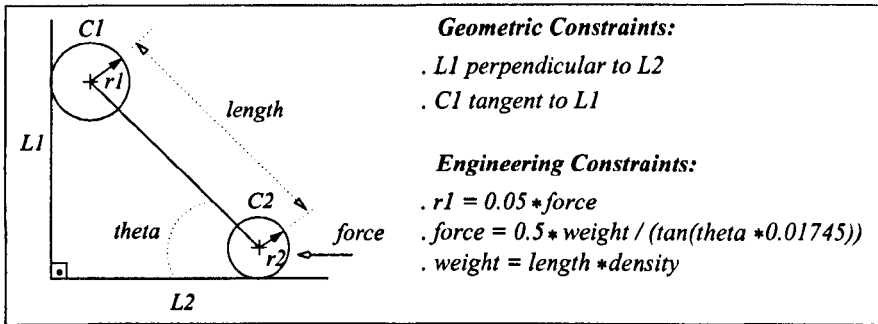


Figure 1.2: Geometric and Engineering Constraints

Geometric constraints are geometric relationships imposed on distinct geometric entities (e.g. a *tangency* constraint between a line and a circle). Engineering constraints are equations used to represent basic engineering principles such as the stress in a part, the length of a shaft, the performance of an electric motor etc. The ladder design of Figure 1.2 illustrates the concept of geometric constraints and engineering constraints used throughout this thesis.

Both engineering and geometric constraints can be seen as a unique set of equations [23, 61]. Thus, given such a mathematical platform, the distinction between engineering and geometric constraints cannot be determined by the computer. This distinction is, in fact, determined by the designer when deciding to impose either a geometric or an engineering constraint during the design process. This distinction is useful since different constraint satisfaction techniques which are more suitable for a particular type of constraint can be used to achieve efficiency.

Inequalities are also considered as engineering constraints since they are of fundamental importance for design. However, inequality constraints are not currently investigated in the context of this thesis.

## 1.5 Constraint-based Design

Traditional CAD systems can only support detailed geometry [67] since they require an object to be defined in terms of specific dimensions and locations of its constituent parts. In the field of solid modeling, these systems fall into two main categories : Constructive Solid Geometry (CSG) and boundary representation (B-rep) [82]. CSG modelers are binary trees constructed from regularised set operations on primitives that are parameterised by their sizes and locations. Examples of CSG modelers are PADL [83] and NONAME [114]. B-rep modelers store object's boundary in term of vertices, edges, faces and their topological relationships. Typical examples are ACIS [102] and PARASOLID [99].

Such representations require the designer to perform considerable algebraic manipulation to determine the values of the absolute positions and dimensions of the object models. Besides, any change in the definition of the object will require the recalculation of the affected values [18]. In contrast, definition of models in terms of geometric constraints such as tangency, concentricity, between the constituent geometric elements of objects, is seen as an intuitive way to capture the engineer's design intent [23]. This reflects the ability to change the data and to propagate it to downstream applications which naturally is a decisive and competitive factor for the current design market. For this reason, researchers have pointed out that traditional CAD and solid modeling tools are being rapidly replaced by constraint-based design systems [94]. However, in spite of being more attractive from the designer's point of view [27, 52, 84], considerable difficulties arise when implementing constraint-based systems [18, 33]. These difficulties are most related to the automation of algebraic manipulations (solution of constraint sets) required by the designer.

In recent years, two main approaches have emerged to support constraint-based design: the *equation-based approach* and the *geometric constructive approach*. The equation-based approach translates all sets of constraints into a system of simultaneous equations. The design model is updated by solving this system of equations. These systems rely on numeric techniques such as the Newton-Raphson iterative method to solve the equations [48, 61, 62, 43, 101, 105]. The main disadvantages of iterative methods is that they present convergence problems and are computationally expensive and therefore they are not suitable for interactive design [33]. Graph-based techniques have been introduced to represent and decompose the equations into smaller sets in order to achieve a more efficient equation solving [93].

The geometric constructive approach provides a high level representation to deal with geometric constraints. In this approach, geometric entities and geometric constraints are not treated as equations directly. Instead, a set of constructive steps is provided which place geometric elements relative to each other through operational transformations, according to their degrees of freedom [14, 37, 60, 74, 112, 116]. Degrees of freedom are the number of independent coordinates which are necessary and sufficient to define the position of a rigid body [44, 45]. The constructive technique has been recognised as more intuitive and superior to iterative equation-based approach for solving geometric constraints, since it avoids numeric instability [89]. However, the technique is only suitable to solve geometric constraint problems, since it works nominally with specific geometric knowledge. It does not efficiently support engineering constraints [5]. Thus, this approach is inadequate for supporting the early design stages, since it only supplies tools for developing detailed geometry [67].

Furthermore, both approaches present common limitations:

- The majority of these systems work on the following basis: ‘find a configuration for a set of geometric objects which satisfies a set of given constraints among them’. This means that the user has to specify a consistent and sufficient set of dimension and constraints and explicitly requests the system to satisfy them. It has been noted that such constraint assignment is tedious and error-prone [87]. Besides, the whole set of constraints must be evaluated from scratch due to the insertion of a new constraint. This can hinder immediate feedback for designers [70];
- The designer has to make sure that the specified constraints compose a fully-constrained set, since efficient and reliable mechanisms to handle under and over-constrained situations are not often available. This obstructs the designer’s task in exploring different design alternatives [33];
- There is also a limited use of graphical interaction techniques in such systems [40, 87, 100]. Therefore, the user cannot directly manipulate geometric entities and thus explore under-constrained spaces. This results on a non-intuitive way of specifying a complete set of constraints [85].

Therefore, this research has been directed towards the development of algorithms that can speed up the constraint satisfaction process of a constraint-based design system. These algorithms avoid satisfying the constraint set from scratch each time a constraint is inserted

or deleted. This work also includes the investigation of interactive techniques to provide more friendly design environments. This allows the designer to explore the degrees of freedom of under-constrained models through direct manipulation techniques. This should offer the potential to develop highly interactive constraint-based design systems as well as to improve the efficiency of such systems during the process of design.

## 1.6 Thesis Organisation

This thesis is divided in seven chapters including this introduction. Chapter 2 presents an overview of the state-of-the-art in constraint-based design. The research carried out in this thesis is built upon these advances.

Chapter 3 presents a set of algorithms to support the incremental satisfaction of engineering constraints. These algorithms maintain an evolving solution of the constraint set and allow engineering constraint cycles to be locally identified and solved. The cantilever beam problem is used throughout the chapter to demonstrate the feasibility of the algorithms. Chapter 4 shows how these incremental algorithms are extended to support the coupling of engineering constraints and geometric constraints. The concept of *soft* constraints is also introduced to allow direct manipulations of under-constrained geometric models.

Chapter 5 is concerned with the implementation of a prototype constraint engine developed in C++ on a SGI XS24/4000 Indigo. The constraint engine has two main modules: the Graphical User Interface (GUI) and the Constraint Manager. The Constraint Manager is composed of four sub-modules: the Equation Graph, the Incremental Equation Solver (INCES), the Geometric Relationship Graph and the Geometric Cycle Solver. The GUI has been developed using an object-oriented graphics toolkit called Iris Inventor [104]. The Equation Graph is a data structure which supports the coupling of geometric constraints and engineering constraints. The INCES algorithm incrementally solves the Equation Graph when linear and nonlinear equations are inserted. The Geometric Relationship Graph is a high level representation of the geometric entities used to handle direct manipulations of under-constrained models. The Relationship Graph is also used to identify geometric constraint cycles which are solved by the Geometric Cycle Solver. The chapter covers a brief description of Iris Inventor and details the implementation of the constraint engine.

Chapter 6 presents the results and discusses the advantages and limitations of the constraint

engine. The design of the internal-combustion engine is used to demonstrate the capabilities of the constraint engine. The benefits of the proposed techniques are then discussed in comparison to previous work. In addition, several limitation are identified and suggestions for improvements are presented.

Finally, Chapter 7 presents the conclusions for this work and recommendations for future research.



# Chapter 2

## Characterisation of Constraint-based Approaches

---

### 2.1 Introduction

There have been several attempts to develop constraint-based algorithms to support engineering design. The purpose of this chapter is to present a characterisation of the current constraint-based design approaches. The main features of these approaches are discussed with references to some key research in the area.

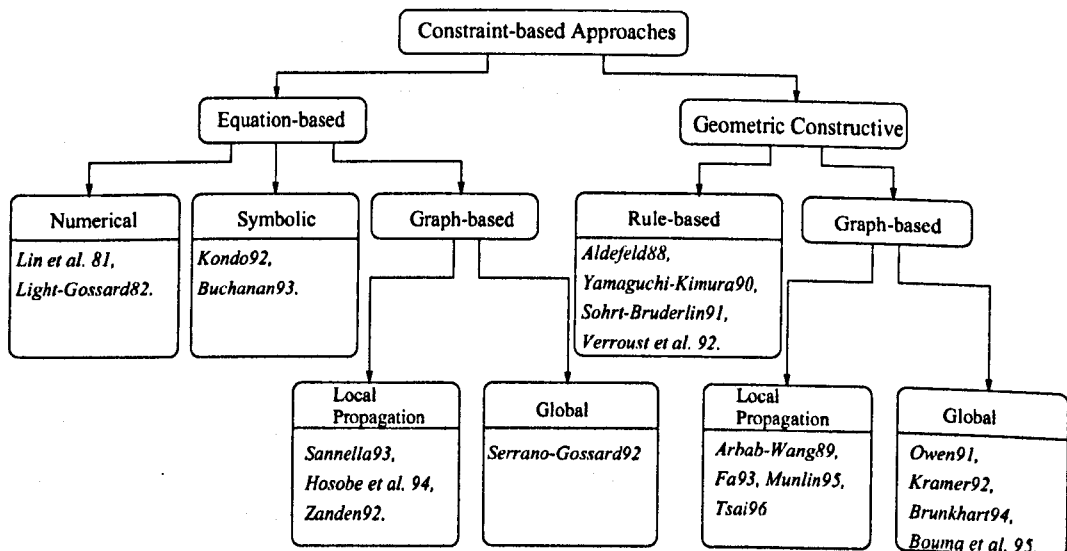


Figure 2.1: Approaches for Constraint Solving in Design

As shown in Figure 2.1, the current constraint-based approaches can be divided into two main categories: Equation-based approach and Geometric Constructive approach.

## 2.2 Equation-based Approach

This approach deals with constraints as equations. In this approach, geometric and/or engineering constraints are all converted into a system of equations and solved by using a variety of equation manipulation techniques. These can be characterised as: numeric, symbolic and graph-based.

### 2.2.1 Numeric Approach

Numerical solvers translate a set of design constraints into a system of nonlinear equations. Instances of a geometric model are derived by solving the system of equations using iterative numeric techniques such as Newton-Raphson method [48, 62, 61, 72, 101].

#### *The Newton-Raphson Method*

The Newton-Raphson method is a well-known technique for finding the roots of a function in one variable which can easily be generalised to find roots in multiple variables [16, 68, 79]. This section presents a brief introduction to Newton's method in one dimension and shows its extension to multiple dimensions.

To illustrate how Newton's method works, consider Figure 2.2 where  $p$  is the root for the function  $y = f(x)$ . The method starts by considering  $p_0$  as a initial guess for the root. Then, the point  $p_1$  is calculated by intersecting the line tangent to the curve at point  $(p_0, f(p_0))$  and the  $x$ -axis.

Note in Figure 2.2 that  $p_1$  will be closer to  $p$  than  $p_0$ . The slope  $m$  of the line through  $(p_1, 0)$  and  $(p_0, f(p_0))$  is given by:

$$m = \frac{0 - f(p_0)}{p_1 - p_0} = f'(p_0). \quad (2.1)$$

Equation 2.1 can be rearranged to find the value for  $p_1$  as in Equation 2.2:

$$p_1 = p_0 - \frac{f(p_0)}{f'(p_0)} \quad (2.2)$$

If the initial guess is sufficiently close to the root  $p$ , the process above can be repeated to obtain a sequence  $\{p_k\}$  that converges to  $p$ . This sequence can be obtained as:

$$p_k = p_{k-1} - \frac{f(p_{k-1})}{f'(p_{k-1})}, \quad k = 1, 2, \dots \quad (2.3)$$

or

$$\Delta p = -[f'(p_{k-1})]^{-1} f(p_{k-1}) \quad (2.4)$$

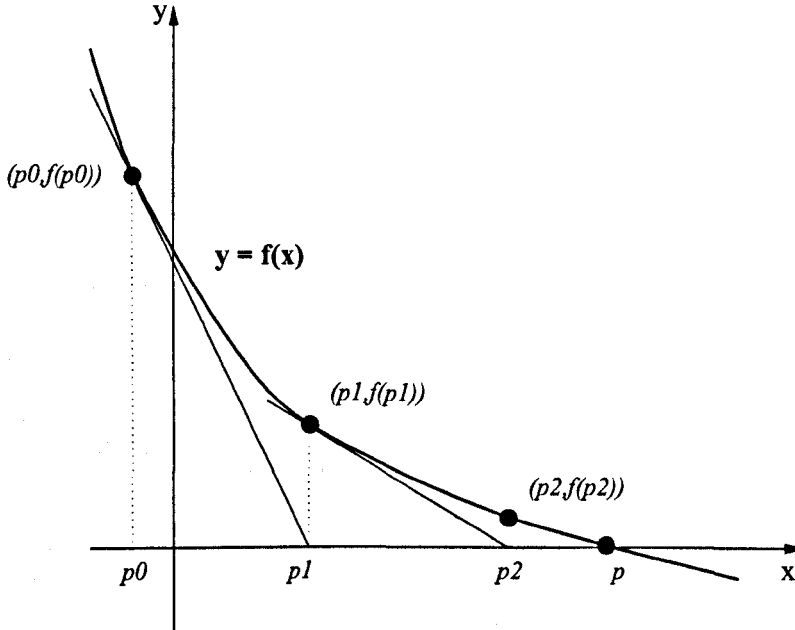


Figure 2.2: Geometric Construction for the Newton-Raphson Method

This means that for a given continuous function with continuous first derivative, there exists an initial guess  $p_0$  for the root  $p$ , such that  $|f(p_0)| \geq \epsilon$ . Providing that  $p_0$  is sufficiently close and using  $p_0$  as an initial guess, Newton's method will converge to some  $p_n$  such that  $|p - p_n| \leq |p - \epsilon|$ , for an arbitrary  $\epsilon > 0$ .

Newton's method can be modified to solve constraint problems involving systems of equations with multiple variables. For example, given the following system with  $n$  variables and  $k$  constraints,

$$\begin{aligned} f_1(x_1, x_2, \dots, x_n) &= 0 \\ &\vdots \\ f_k(x_1, x_2, \dots, x_n) &= 0 \end{aligned}$$

Equation 2.4 can be seen as:

$$\Delta \mathbf{P} = -\mathbf{J}^{-1}(p_{k-1})\mathbf{F}(p_{k-1}) \quad (2.5)$$

where  $\mathbf{P}$  represents the vector  $[x_1, x_2, \dots, x_n]$ ,  $\mathbf{F}$  represents the vector  $[f_1(\mathbf{P}), f_2(\mathbf{P}), \dots, f_k(\mathbf{P})]$  and  $\mathbf{J}$  represents the *Jacobian matrix*. The Jacobian matrix is the generalisation of the “derivative” for systems of functions of several variables [68]

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & & \\ \frac{\partial f_k}{\partial x_1} & \frac{\partial f_k}{\partial x_2} & \cdots & \frac{\partial f_k}{\partial x_n} \end{bmatrix}$$

Note that the Jacobian matrix must be nonsingular (i.e.  $\mathbf{J}^{-1}$  must exist). Otherwise Equation 2.5 can not be solved. This implies that the Jacobian must be a square matrix and therefore  $n = k$ . In this case, researchers refer to the system of equations as a *fully constrained* system. This approach has been used in the design systems reported in [61, 62, 101].

Although the numeric approach has the potential to solve configurations that may be non-solvable using other approaches, it performs in lengthy solution times and therefore is inappropriate for interactive design [14, 33]. Furthermore, as the system of equations may have multiple solutions, convergence to the required solution cannot be guaranteed. Although the user can control the solution by choosing a set of starting guesses, this is not a robust approach to determine all or particular solutions [18].

### 2.2.2 Symbolic Approach

In the symbolic approach, the constraints are translated into a system of algebraic equations and solved through symbolic algebraic methods, such as Gröbner basis [19].

#### *Gröbner Basis*

Given a set of nonlinear equations, its Gröbner basis is a canonical form where the resulting equations are ordered to produce a solution [9, 47]. Therefore, the Gröbner basis is a set of equations which is mathematically equivalent to the original set. It is obtained by eliminating variables between equations, in much the same way that Gaussian elimination

is applied to a system of linear equations. For example, given the system of equations:

$$\begin{aligned} x + y &= 5 \\ x^3 + y^2 &= 11 \end{aligned}$$

the two following Gröbner basis can be derived:

$$\begin{aligned} x + y - 5 &= 0 & x + y - 5 &= 0 \\ y^3 - 16y^2 + 75y - 114 &= 0 & x^3 + x^2 - 10x + 14 &= 0 \end{aligned}$$

Note that in both cases, the second equation contains only one unknown. This equation can be solved numerically and the roots may be substituted into the first (this process is called *back-substitution*) to obtain the value of the other unknown and thus to solve the system.

As an application of Gröbner basis, consider the triangle with vertices  $P_1$ ,  $P_2$  and  $P_3$  with sides defined by the constraints in Figure 2.3.

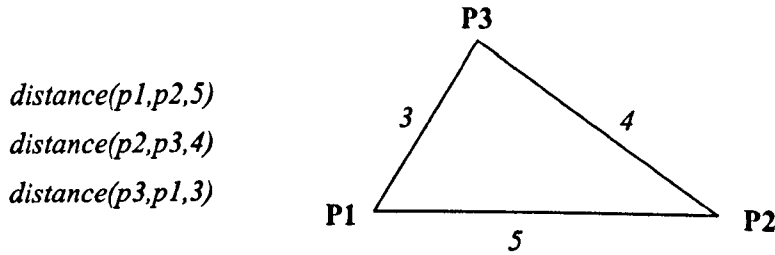


Figure 2.3: Geometric Constraints for a Triangle Model

Fixing  $p_1 = (1, 1)$  and  $p_{2x} = 4$ , produces the following system of equations:

$$\begin{aligned} 9 + (1 - p_{2y})^2 - 25 &= 0 \\ (4 - p_{3x})^2 + (p_{2y} - p_{3y})^2 - 16 &= 0 \\ (p_{3x} - 1)^2 + (p_{3y} - 1)^2 - 9 &= 0 \end{aligned}$$

Note that this system of equations is not triangularised and therefore requires a numeric solution. However, one Gröbner basis for this system is:

$$\begin{aligned} 16p_{3y} + 3p_{2y}p_{3x} - 12p_{2y} - 3p_{3x} - 4 &= 0 \\ p_{2y}^2 - 2p_{2y} - 15 &= 0 \\ 25p_{3x}^2 - 104p_{3x} + 16 &= 0 \end{aligned}$$

Note that this system is now triangularised and can be solved using back-substitution from the last equation. The following solution is then produced: ( $p_{3x} = 4$  or  $p_{3x} = 0.08$ ),

( $p_{2y} = 5$  or  $p_{2y} = -3$ ) and  $p_{3y} = 1$ . This leads to the problem of multiple solutions. However, the back-substitution procedure can be performed in a rigorous manner that unwanted roots can be eliminated at each step [17]. Thus, if the user selects  $p_{3x} = 4$  and ( $p_{2y} = 5$  the final solution for this problem will be the one presented in Figure 2.4.

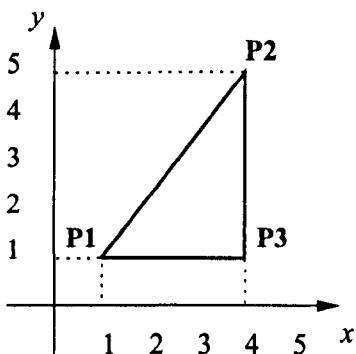


Figure 2.4: Solution for the Triangle Model

Symbolic solvers are more robust than standard numeric techniques when solving general nonlinear constraint problems. Constraint design systems such as those reported in [18, 56] have used Gröbner basis to solve their systems of constraint equations. However, this approach is still computationally expensive [14, 17].

### 2.2.3 Graph-based Approach

#### Graph Theory Terminology

A brief introduction to the graph terminology used in this section follows. Graphs are generally chosen to represent constraint networks because they have the following advantages [92]:

1. they are a very general domain-independent representation;
2. they allow both qualitative and quantitative operations;
3. there are a large number of applicable algorithms from the graph theory literature.

*Graph Definition*

A **graph**  $G = (V, E)$  consists of two sets: a finite set  $V$  of elements called *vertices* (or nodes) and a finite set  $E$  of elements called *edges* (or arcs) [1, 107]. Each edge is identified with a pair of vertices. An edge defines a binary relation between its constituent vertices.

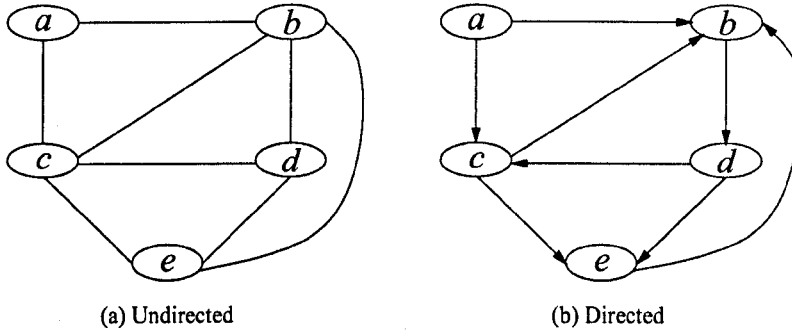


Figure 2.5: Undirected and Directed Graphs

If the edges of a graph  $G$  are identified with ordered pairs of vertices, then  $G$  is called a *directed graph* (or digraph). Otherwise,  $G$  is called an *undirected graph*. Figure 2.5(a) and Figure 2.5(b) illustrate the concept of undirected and directed graphs, respectively.

A *path* in a directed graph is a list of vertices  $(v_1, v_2, \dots, v_k)$  such that there is an arc from each node to the next, i.e.  $v_i \rightarrow v_{i+1}$  for  $i = 1, 2, \dots, k$ . For example,  $(a, b, d, e)$  and  $(a, c, e)$  are paths in the directed graph of Figure 2.5(b). A node  $a$  is called an *ancestor* of node  $d$  if there is a path from  $a$  to  $d$ . In turn,  $d$  is called a *descendant* of  $a$ . In Figure 2.5(b), node  $b$  is an ancestor of node  $e$ .

A *cycle* in a directed graph is a path that begins and ends at the same node. For example, in the graph shown in Figure 2.5(b) the paths  $(b, d, c, b)$  and  $(b, d, e, b)$  define two distinct cycles. If a graph has one or more cycles, it is referred to as a *cyclic graph*. If there are no cycles, the graph is said to be *acyclic*. The concept of *ancestor* and *descendant* nodes does not apply to the nodes inside a cycle.

*Bipartite Graphs*

A graph  $G = (V, E)$  is called a *bipartite graph* if its vertex set  $V$  can be partitioned into two disjoint subsets  $V_1$  and  $V_2$  (i.e.  $V_1 \cap V_2 = \emptyset$ ) such that each edge of  $E$  has one vertex in  $V_1$  and another in  $V_2$ . Figure 2.6(a) shows an example of a bipartite graph.

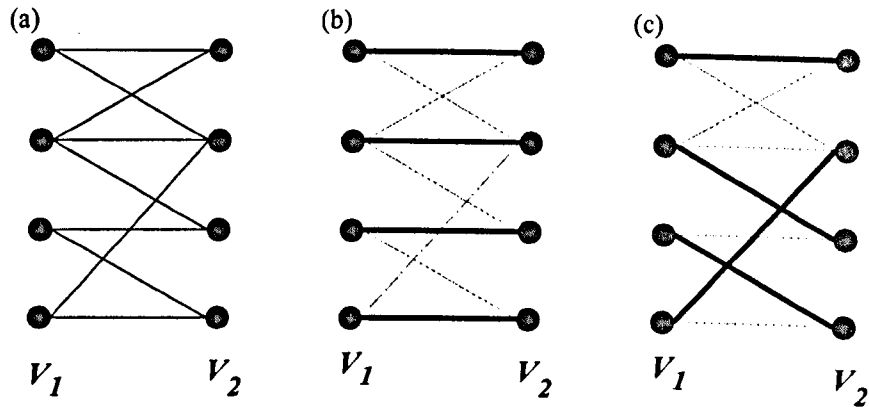


Figure 2.6: A Bipartite Graph and Two Maximum Matchings

Given an undirected bipartite graph  $G = (V_1, V_2, E)$ , a **matching** is a subset of edges  $M \subseteq E$  such that for all vertices  $v \in V_1 \cup V_2$ , at most one edge  $M$  is incident on  $v$ . A **maximum matching** is a matching of maximum cardinality, that is, a matching  $M$  such that for any other matching  $M'$ ,  $|M| \geq |M'|$  [25]. In other words, a maximum matching is a largest subset of  $E$  with the property that no two pairs share a common element in the same  $V_1$  or  $V_2$ . A bipartite graph can have more than one maximum matching between the elements of  $V_1$  and  $V_2$ . Figure 2.6(b) and Figure 2.6(c) show two different maximum matching for the bipartite graph of Figure 2.6(a).

## 2.2.4 Graph-based Equation Solvers

Graph-based equation solvers [50, 90, 93] first translate the set of constraints into an undirected graph. This graph is composed of two distinct sets: constraints (equations) and variables. Each variable is connected to the constraint it belongs to, through an edge. Then, different techniques are used to direct the graph edges in order to produce a sequence of constraint satisfaction. This approach can be further divided into two other distinct ones: *global* and *local propagation* approaches.

### Global Approach

In this approach, the entire set of constraints is evaluated from scratch on the insertion of a new constraint. Thus, a new undirected graph is generated and a new constraint dependency inside the graph is derived globally to produce the new constraint satisfaction



sequence.

Serrano presents a system to support Conceptual Design (called Concept Modeler), which is based on the global approach [93]. In order to illustrate Serrano's approach consider the following set of equations:

$$C1 : \sigma - \frac{MY}{I} = 0$$

$$C2 : M - FL = 0$$

$$C3 : I - \frac{WH^3}{12} = 0$$

$$C4 : Y - \frac{H}{2} = 0$$

$$C5 : K - \frac{3EI}{L^3} = 0$$

$$C6 : \phi - \frac{FL^2}{EI} = 0$$

The problem is to identify the values of the variables  $L$ ,  $H$  and  $W$ , given  $\phi$ ,  $M$ ,  $E$ ,  $K$  and  $\sigma$ .

The main steps of the algorithm are as follows:

1. *Represent the equations in an undirected graph:* Figure 2.7 shows the undirected graph representing equations  $c_1 - c_6$ . Again, in this graph nodes represent variables and arcs represent constraints. Two variables are connected through an arc if and only if they belong to the same equation.

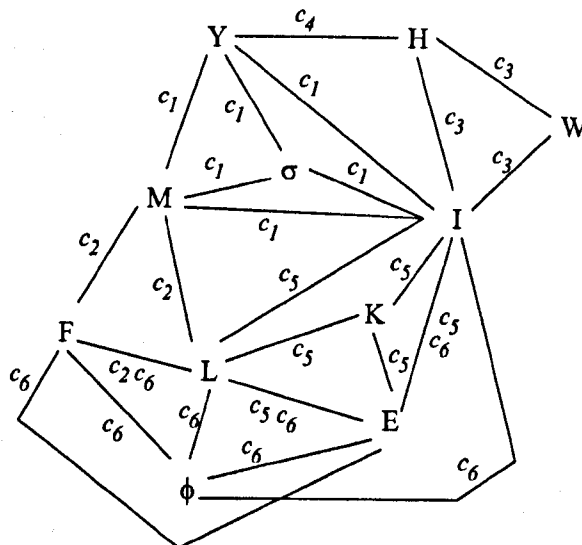


Figure 2.7: Undirected Equation Graph

2. *Derive the constraint dependency:* the user is required to specify the known and un-

known variables in the equations. The system then automatically derives dependency information through an assignment of every unknown variable to a constraint equation. First, a bipartite graph  $G = (\{U, C\}, A)$  is constructed where  $U$  represents the unknown variables and  $C$  represents the constraints.  $A$  is a set of arcs that link every unknown variable to the constraint equations it belongs to (Figure 2.8(a)). Note that this is a new graph and not the one used to represent the original constraint network. Next, a **maximum matching** is found between the elements of  $U$  and  $C$ . Figure 2.8(a) presents a maximum matching in the bipartite graph composed of the unknown variables and constraints. The dashed lines indicate the assignments that were not used. Each variable in this matching is set as the output of its matched constraint in the original constraint network.

Once the assignments are made, a directed graph is generated (Figure 2.8(b)). In this graph, the known variables are represented as squares whereas the unknown ones are represented as circles. The unmatched constraint-variable arcs are removed from the previous undirected graph. The arc direction defines the constraint dependency inside the network. For example,  $H \xrightarrow{c_4} Y$  in the graph means that  $H$  “depends on”  $Y$ , i.e.  $H$  is going to be calculated using equation  $c_4$ , after the value of  $Y$  is identified.

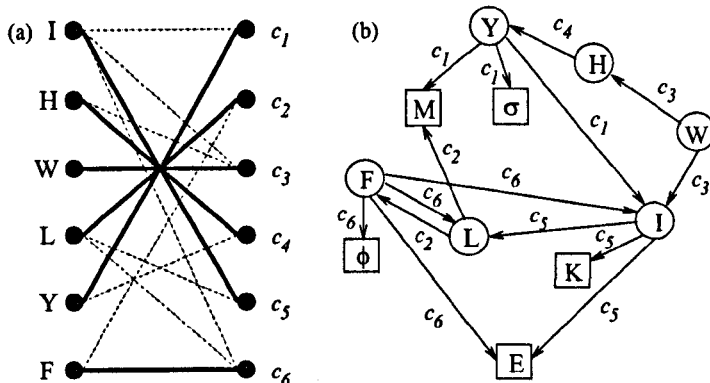


Figure 2.8: Maximum Matching and Directed Graph

3. *Identify strongly connected components:* The next step is to check if the constraint network contains special loops called strongly connected components. Given a directed graph, a strongly connected component (SC) is a maximal set of nodes inside a subgraph such that every node in the subgraph is reachable from every other node in the same subgraph [1].

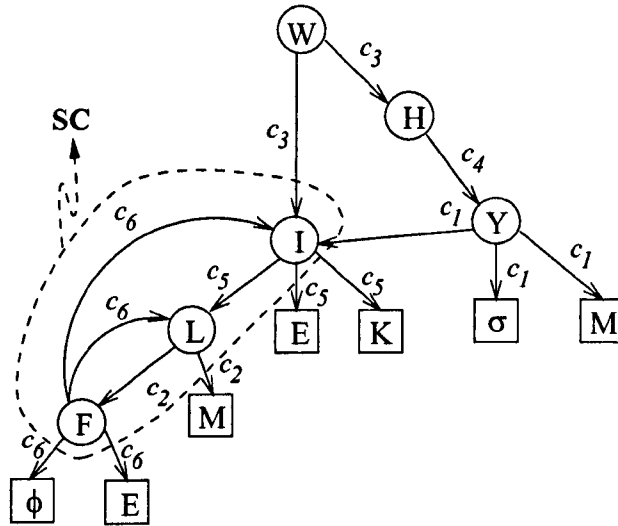


Figure 2.9: Tree-like Representation and a SC

The dashed curve in Figure 2.9 shows a strongly connected component. Note that the loop between  $F$  and  $L$  is inside the loop composed by  $F$ ,  $L$  and  $I$ . Thus, the latter is considered as the SC because it is maximal. Serrano’s approach treats a SC as a cycle of constraints that must be solved simultaneously through a numeric solver. SCs are located by search-based algorithms which traverse the network labelling the visited nodes and storing the identified cycles [7] (the graph of Figure 2.8(b) has been written as a tree-like representation in Figure 2.9 to help the reader to identify the constraint cycle composed by the variables  $F$ ,  $L$ , and  $I$  inside the dashed ellipse).

4. *Derive the sequence of constraint satisfaction:* With the formation of strongly connected components, the constraint network is now seen as a directed acyclic graph (DAG) as shown in Figure 2.10. A fundamental operation on DAGs is to process all parent nodes before any of their children. This operation is called topological sorting and is shown as a dashed line in the DAG of Figure 2.10. Serrano’s algorithm first performs a topological sort on the remaining DAG after SCs are identified. Then, the algorithm reverses this topological order to produce the sequence of satisfaction. Thus, according to Figure 2.10, the sequence of satisfaction after the reverse topological sort is : **SC-Y-H-W**. In other words :

- calculate the values of  $F$ ,  $L$ , and  $I$  within the SC node, by solving the system of equations composed by  $c_2$ ,  $c_5$  and  $c_6$ , using a numeric solver;

- calculate  $Y$  using  $c_1$ ;
- calculate  $H$  using  $c_4$ ;
- calculate  $W$  using  $c_3$ .

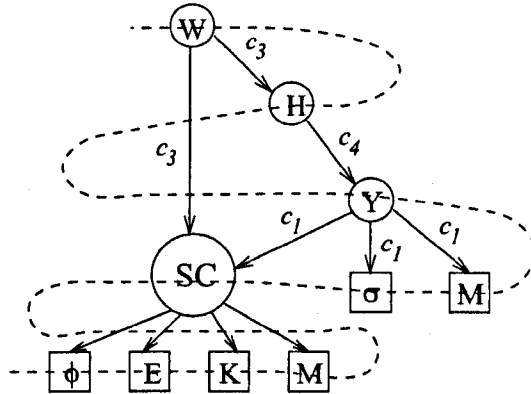


Figure 2.10: Topological Sorting in a Directed Acyclic Graph with SCs

Note that all the necessary information to calculate a variable is provided either by known variables (such as  $\sigma$  and  $M$  when calculating  $Y$  through  $c_1$ ) or previously satisfied equations (such as  $Y$  when calculating  $H$  through  $c_4$ ).

Serrano handles under- and over-constrained situations by counting the number of constraints ( $c$ ) and the number of unknown variables ( $u$ ). If  $c > u$  the system is over-constrained. The system is under-constrained if  $c < u$ . When the system is over-constrained, there will be unmatched constraints in the bipartite graph that may prove to be either redundant or conflicting. In this case, the subset included in the matching is evaluated. Next, the unmatched constraints are used to verify the results of the evaluation. If an unmatched constraint can be satisfied using the results obtained in the bipartite matching, it is considered as redundant. On the contrary, if an unmatched constraint cannot be satisfied it is considered as conflicting. When the constraint network is under-constrained, Serrano asks the user to either fix variables or insert more constraints.

Serrano claims that at a feasible solution there are often as many inequality constraints satisfied by the equality condition ( $=$ ) of the inequality ( $\leq$ ), as there are design variables. In addition, at a point in the space which produces a feasible solution, not all inequalities are active while other behave as equalities [92]. Therefore, Serrano uses the maximum matching in the bipartite graph as a starting point to determine a possible constraint set

which defines a solution. Then, inactive constraints are used to check on the values of the computed parameters and may either prove to be redundant or conflicting.

The global graph-based approach has proven to be more efficient than numeric and symbolic techniques [92]. However, this approach could still be expensive for a large set of equations since it derives the sequence of satisfaction from scratch whenever a new constraint is inserted.

### Local Propagation Approach

In this approach, the directed graph is locally updated to absorb the newly imposed constraint. Thus, only part of the current graph is affected. Algebraically, this approach works by propagating the value of a variable to all constraints in the constraint set that depend on that variable. When the value for an unknown variable is defined, the local propagation technique checks its associated constraint (or constraints) to see if only one unknown variable is remaining in the equation. If so, the constraint is solved by making the unknown variable the output variable. The new value of the output variable is then propagated to the downstream nodes of the network by resatisfying them.

However, local propagation is only effective for systems of constraints that can be ordered into lower triangular form [16, 29]. Figure 2.11 shows two triangular systems.

$x - 3y + 2z + 5w = 4$
$y + 5z + 3w = 2$
$3z + 2w = 0$
$w = 3$
$x^2 + 4y^3 - 2z^2 + 3w = 2$
$2y^2 + w = 0$
$3z^3 = 8$
$w^2 = 5$

Figure 2.11: Two Triangular Systems

A necessary and sufficient condition to solve a set of constraints through local propagation

is that of having a constraint with only one remaining variable, whenever known values are propagated to the constraint set. If such a remaining (or unknown) variable cannot be identified, it is deduced that the constraint network has a set of constraints which must be solved simultaneously (i.e. a constraint cycle) and a numerical solution is required. Figure 2.12 shows a system of constraints which can not be triangularised and therefore forms a constraint cycle.

$x - 3y + 5w = 4$
$x + 2y + 3t = 0$
$3z + 2w - 5t = 2$
$2z - 3w + 2t = 3$
$5z + 4w - 3t = 5$

Figure 2.12: A Constraint Cycle

The local propagation approach identifies output variables for each constraint through a set of *methods*. Each method is a procedure which is invoked to satisfy the constraint, i.e. it defines how to calculate the value of the output variable given the value of the input variables. For example, the constraint  $a = b + c$  would present the following methods:  $a \leftarrow b + c$ ,  $b \leftarrow a - c$  and  $c \leftarrow a - b$ , where  $\leftarrow$  denotes assignment, in contrast to  $=$ , which denotes the equality constraint. Some methods allow multiple variables to be considered as output at the same time (*multi-output methods*). For example, suppose that  $x$  and  $y$  are variables that represent the Cartesian coordinates of a point. Similarly, the variables  $\rho$  and  $\theta$  represent the polar coordinates of the same point. To keep this representation consistent the following two methods could be defined:  $c : (x, y) \leftarrow (\rho \cos \theta, \rho \sin \theta)$  and  $c : (\rho, \theta) \leftarrow (\sqrt{x^2 + y^2}, \arctan(x, y))$ . These method graphs are respectively shown in Figure 2.13 where the constraint  $c$  is represented as a square and the variables as circles.

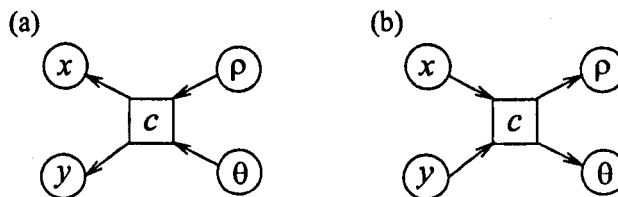


Figure 2.13: Multiple Output Methods

Researchers in this subject also refer to this approach as an *incremental* technique. Such techniques have been recognised as suitable to achieve interactive performance [117, 32]. This section presents current systems which are based on the local propagation approach.

DeltaBlue is an incremental constraint solver based on constraint hierarchies [36]. The constraint hierarchy theory (how strongly a constraint should be satisfied [13]) is introduced to define the sequence of constraint satisfaction and to help the solver in handling under- and over-constrained networks.

When using DeltaBlue, the user is required to specify strength attributes to all the constraints involved. A strength value for each variable is then derived by DeltaBlue to predict the effect of inserting a new constraint. This information is called the *walkabout strength* of the variable that is defined as follows:

When variable  $v$  is determined by method  $m$  of constraint  $c$  (i.e.  $v \in c$ ) the walkabout strength of variable  $v$  is the minimum of  $c$ 's strength and the walkabout strengths of  $m$ 's input.

If a variable is not determined by any constraint then its walkabout strength is defined as the weakest value. To illustrate the concept of walkabout strength consider Figure 2.14. This figure shows part of a directed graph which is used by DeltaBlue to represent the current solution for a set of constraints.

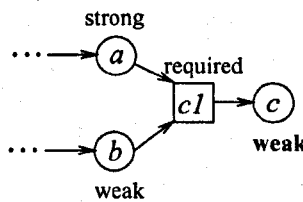


Figure 2.14: Walkabout Strengths of Output Variables

In this figure, the method  $c \leftarrow a - b$  is chosen to satisfy the **required** constraint  $c1 : a = b + c$ . In an increasing order, the strengths used are **weak**, **strong** and **required**. Thus, the walkabout strengths of variable  $c$  (output variable) is **weak**, because it is the minimum strength of the constraint  $c1$  (**required**) and the input variables  $a$  and  $b$  for the chosen method (**strong** and **weak**).

When a new constraint is inserted, DeltaBlue *incrementally* accommodates this constraint

into the current solution. This is achieved by selecting the weakest variable associated with the constraint as the potential output. If the constraint is weaker than the weakest associated variable then the constraint cannot be satisfied.

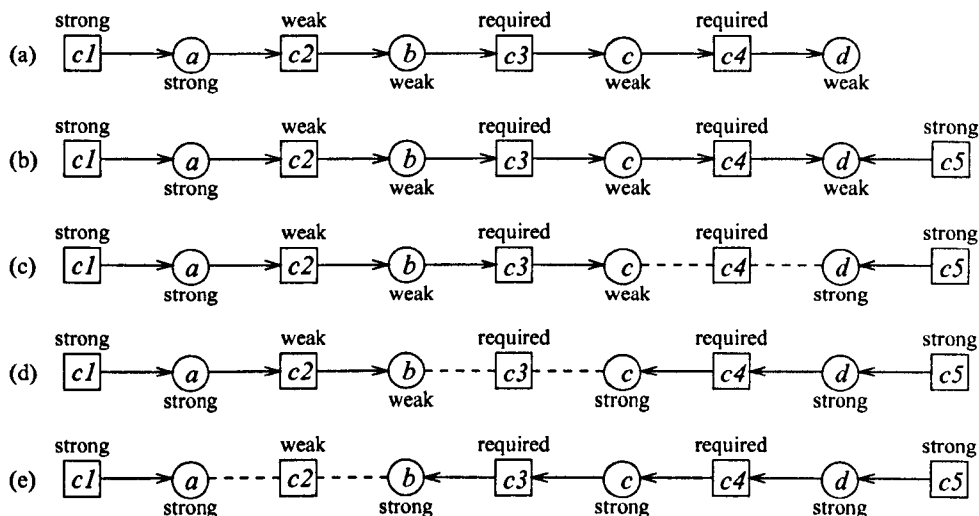


Figure 2.15: Constraint Insertion in DeltaBlue

Figure 2.15 shows the process of inserting a constraint. Figure 2.15(a) shows the initial constraint network. In Figure 2.15(b), a **strong** constraint on  $d$  ( $c5$ ) has been inserted. Since the walkabout strength of variable  $d$  is weaker than that of constraint  $c5$ , the algorithm sets  $d$  as the output of  $c5$ . As a result, variable  $d$  becomes the output of constraints  $c4$  and  $c5$ . This situation is called *constraint conflict* and must be avoided since it does not make sense to have the value of one variable being calculated by two different constraints.

To avoid such a conflict, the algorithm revokes constraint  $c4$  which previously flowed into  $d$  and caused its strength to be set as **weak**. This leads to the situation shown in Figure 2.15(c). Note that now the walkabout strength of variable  $d$  is set as **strong** because of constraint  $c5$ . Since constraint  $c4$  is stronger than the walkabout strength of its variables  $c$  and  $d$ , the algorithm knows that constraint  $c4$  can be satisfied. Thus,  $c$  is chosen as the output variable since it has the weaker walkabout strength and  $c4$  is re-satisfied in the new direction, as shown in Figure 2.15(d). This causes the constraint flowing into variable  $c$  to be revoked and this constraint propagation process continues until the algorithm finds a constraint that can not be satisfied. This is the case with constraint  $c2$  in Figure 2.15(e) where  $c2$  is weaker than the walkabout strength of its variables. In such cases, the constraint is left unsatisfied which terminates the algorithm. The dotted lines for constraint  $c2$  mean



that its methods are temporarily unused. If constraint  $c_5$  was weaker than the walkabout strength of variable  $d$  (e.g. **very weak**), it would have remained as unsatisfied.

DeltaBlue has been used as an equation solver for the WAYT *Why-Are-You-There?* system presented by Mäntylä [67]. This system has been designed to support preliminary design by allowing the user to manipulate conceptual geometric entities and to specify their interdependencies as a set of equations that are maintained by DeltaBlue. However, DeltaBlue is limited to a simple domain (only linear equations), despite its significance for interactive applications. As pointed out by Mäntylä, a more elaborate constraint satisfaction and propagation mechanism is required.

SkyBlue [90] is a successor of the DeltaBlue algorithm by allowing the presence of cycles and multi-output methods. When a cycle is encountered, several external *cycle solvers* are called that try to find values for the cycle output variables. For example, if all constraints are linear equations, a cycle solver incorporating a simultaneous linear equation solver could produce values that satisfy the constraints inside the cycle. However, the algorithm can also detect constraint cycles which can not be satisfied by the available cycle solvers. In this case, the variables inside the cycle are left invalid and the downstream method graphs that depend on that cycle are not executed.

When trying to satisfy a multi-output constraint using one of its outputs, SkyBlue uses backtracking when faced with constraint conflicts at a downstream node. In this case, the algorithm backtracks to reach the original constraint in order to try another output. However, there is no guarantee that this new attempt will not find another constraint conflict. This process could take exponential time [117]. In addition, according to the chosen methods the algorithm might report a cycle when in fact it is possible to find an acyclic solution. SkyBlue has been used as an equation manipulation tool in the Pika simulation system [3]. Pika constructs simulations in domains such as electronics or thermodynamics by collecting algebraic and differential equations representing relationships among objects attributes.

Hosobe et al. have presented a similar incremental algorithm based on constraint hierarchies for satisfying linear constraints [50]. The algorithm decomposes the method graph into subgraphs called *constraint cells*. A constraint cell is a set of variables and constraints where each variable and each constraint can belong to only one constraint cell in the solution graph. Hosobe's algorithm associates the concept of walkabout strength with each constraint cell

and not with variables as in DeltaBlue. In fact, given a constraint cell  $p$ , the walkabout strength of  $p$  is the weakest among  $p$ 's internal strength and walkabout strengths of cells with variables adjacents to  $p$ . When a constraint is inserted, the algorithm identifies 'victimised' constraint cells with weaker strength that will absorb this insertion. Thus, the constraint dependency is updated and a new constraint cell partitioning is produced. Figures 2.16(a) and 2.16(b) shows how Hosobe's algorithm uses the partitioning of cells (boxes with round corners) to solve the constraint problem illustrated in Figures 2.15(a) and 2.15(b)

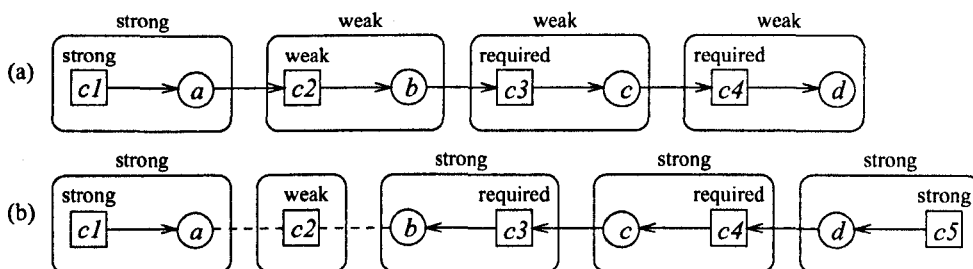


Figure 2.16: A Method Graph Partitioned into Constraint Cells

Incremental constraint satisfaction algorithms based on constraint hierarchies force the user to specify a *strength value* for each and every constraint. Therefore, this could obstruct the designer's train of thought since they need some understanding about how the algorithm works. Thus, less restrictive approaches are required to support the designers.

Vander Zanden has proposed an incremental algorithm for solving systems of linear constraints [117], which does not depend upon constraint hierarchies. In this approach, a directed graph is also used to represent the dependency between constraints and variables and to define the order of constraint satisfaction. When a new constraint is inserted, the constraint-variable dependency is locally updated. All the constraints are equally satisfied. However, nonlinear constraints and constraint cycles are not allowed.

### 2.3 Geometric Constructive Approach

In this approach, constraints are not translated into a system of equations as in the Equation-based approach. Instead, a set of constructive steps is provided which place geometric elements relative to each other. This is achieved through rigid body transformations according to the degrees of freedom of the geometric elements. The techniques

developed under this approach can be divided into two main categories: **rule-based** and **graph-based**.

### 2.3.1 Rule-based Approach

Rule-based constructive solvers apply rewrite rules to discover and execute the construction steps. Brüderlin presents a Prolog-based system for building 3D geometric objects using geometric constraints [15]. First, a symbolic solution plan is determined by applying geometric rules on a set of geometric Prolog facts. Then, the solution is numerically determined by procedures written in Modula-2. For example, consider the following geometric problem written as a set of Prolog predicates:

```

c(P1, [100,100]).    /* the coordinates of point P1 are (100,100) */
c(P2, [200,100]).    /* the coordinates of point P2 are (200,100) */
d(P1, P3, 75).       /* the distance from P1 to P3 is 75 */
d(P2, P3, 80).       /* the distance from P2 to P3 is 80 */
d(P3, P4, 50).       /* the distance from P3 to P4 is 50 */
s(P2, P4, 90).       /* the slope between points P2 and P4 is 90 */

```

The first two predicates fix points P1 and P2 (known geometry) by specifying their Cartesian coordinates. The next three predicates declare geometric constraints specifying the distance between two points. Finally, the last predicate specifies that point P2 and P4 have the same  $x$  coordinate (i.e. the slope of a line passing through both points is  $90^\circ$ ).

Given this database of Prolog predicates, a set of geometric rules are applied to derive a symbolic solution. These rules are of the form *if L then R* (or  $L \Rightarrow R$ ) and express geometric knowledge based on a ruler-and-compass construction basis. For this particular problem, the derived solution is as follows:

```

c(P1, [100,100]).
c(P2, [200,100]).
c(P3, intersection(circle(c(P1), 75), circle(c(P2), 80))).
c(P4, intersection(circle(c(P3), 50), line(c(P2), 90))).

```

This constraint satisfaction sequence is interpreted as:

1. fix the positions of points P1 and P2;
2. calculate the position of point P3 by intersecting the circle centered in P1 with radius 75 and the circle centered in P2 with radius 80;
3. calculate the position of point P4 by intersecting the circle centered in P3 with radius 50 and the line that passes through P2 with slope  $90^\circ$ .

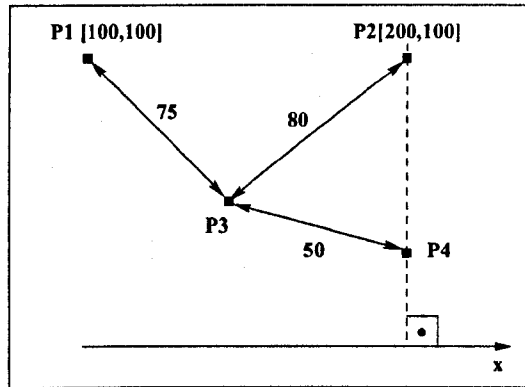


Figure 2.17: Geometric Solution for a Symbolic Rule-based Problem

Note that the sequence of constraint satisfaction is constructed from known to unknown geometric entities. Note also that evaluating some of these operations algebraically may result in more than one solution as for example when calculating point P3 by the intersection of two circles. In this case, Brüderlin uses implicit order information of points given by the sketch. When the intersection of the two circles with centers P1 and P2 is performed, it is possible to decide which of the two intersection points is on the right and which is on the left side of the line going from P1 to P2. The default solution for Brüderlin is “the one that corresponds to the sketch measured by this criterion, and therefore looks most similar to the sketch”.

Recently, Brüderlin has extended his work to support automatic definitions of constraints through direct manipulations [100]. Other similar rule-based solvers are reported in [2, 106, 112, 116]. Although this approach is faster than pure numeric or symbolic approaches, excessive time is still required to find a constraint satisfaction sequence through the searching and matching of rules.

### 2.3.2 Graph-based Approach

In the **graph-based approach**, a graph is used to represent a set of constraints. In this graph, nodes represent geometric entities and edges (arcs) represent geometric constraints. Current graph-based techniques falls into two categories: **global** and **local propagation**.

#### Global Approach

In the **global approach**, the geometric constraints are first represented in an undirected graph. This undirected graph is then analysed and a plan of construction steps is derived. Each construction step is then executed by satisfying constraints algebraically.

Kramer has proposed a new technique called *Degrees of Freedom Analysis* that is able to automatically derive a sequence of actions to satisfy a set of given geometric constraints [60]. To satisfy a geometric constraint, the degrees of freedom (DOF's) of a geometric entity are consumed by moving the geometric entity through operational transformations. Two fundamental operations are used to satisfy binary constraints: *Action Analysis* and *Locus Analysis*. Action Analysis searches for those constraints where one of the geometric entities is “fixed-enough” so that the other geometric entity can be moved to satisfy the constraint. If the “fixed-enough” concept can not be applied, the algorithm tries to satisfy the constraint through Locus Analysis. Locus Analysis identifies the intersection of the locally determined loci of two partially constrained geometric entities to satisfy a specified geometric constraint between them. To illustrate Kramer’s approach, consider the following example.

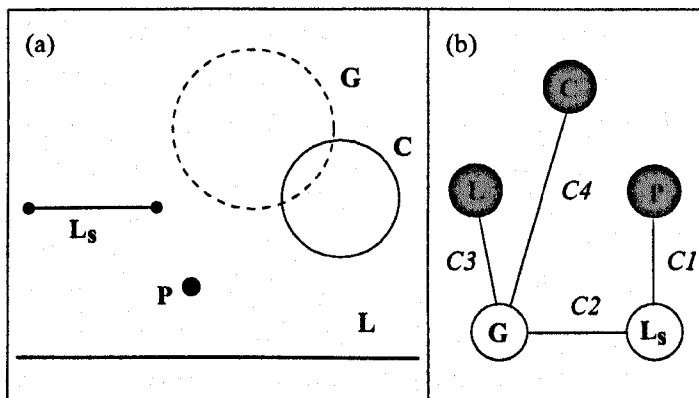


Figure 2.18: A Geometric Constraint Problem and its Graph Representation

Figure 2.18(a) presents a set of geometric entities with the following configuration:

- circle  $C$  with fixed position, orientation and radius;
- infinite line  $L$  with fixed position and orientation;
- a fixed point  $P$ ;
- line segment  $L_s$  with fixed length and free to rotate and translate in the same plane as  $C$  and  $L$ ;
- circle  $G$  which is free to translate and change its radius.

Additional geometric constraints to be solved are:

- $C1$  : *dist:point-point(end-1( $L_s$ ),  $P$ , 0)*;  
/\* the distance from the first end point of  $L_s$  to point  $P$  is 0 \*/
- $C2$  : *dist:point-point(end-2( $L_s$ ), center( $G$ ), 0)*;  
/\* the distance from the second end point of  $L_s$  to the center of circle  $G$  is 0 \*/
- $C3$  : *dist:line-circle( $L$ ,  $G$ , 0)*; /\* the distance from line  $L$  to circle  $G$  is 0 \*/
- $C4$  : *dist:circle-circle( $C$ ,  $G$ , 0)*. /\* the distance from circle  $C$  to circle  $G$  is 0 \*/

Figure 2.18(b) shows the graph representation for this constraint problem. Darkened nodes represent grounded (fixed) geometries. Geometric constraints are shown as arcs in the graph. Kramer's approach first traverses the graph searching for those constraints that can be solved using Action Analysis. Thus, constraints  $C_1$ ,  $C_3$  and  $C_4$  can be satisfied in this order since  $P$ ,  $L$  and  $C$  are respectively grounded which allows the other entity to be moved to satisfy the constraint. Constraint  $C_2$  cannot be satisfied by Action Analysis because  $L_s$  and  $G$  are partially constrained and therefore both are not "fixed\_enough" in relation to each other. In this case, Locus Analysis takes place.

According to its degrees of freedom, the locus for the line segment  $L_s$  is a circle shown as  $L_c$  in Figure 2.19. Similarly, the locus for the circle  $G$  is a parabola shown as  $L_p$ . The intersection of these two loci is then performed. Since multiple intersections are possible, the user is required to choose one of them. Then, Action Analysis is again performed to move  $L_s$  and  $G$  towards the chosen solution, which grounds them both. The final

solution for this geometric problem is shown in Figure 2.19. To implement Action and Locus Analysis, Kramer has proposed a set of specialised routines called *plan fragments*. These routines specify how the configuration of a set of rigid bodies must be changed to satisfy a set of constraints. Clearly, one drawback of this approach is the difficulty to write plan fragments for each specific geometric problem. Recently, Kramer has presented a methodology that uses a specific knowledge base of rules to automatically synthesize plan fragments [10]. Although the rules are difficult to write Kramer argues that the total effort to write and debug these rules is less than writing every piece of code. He intends to extend this methodology to more complex constraints and geometries.

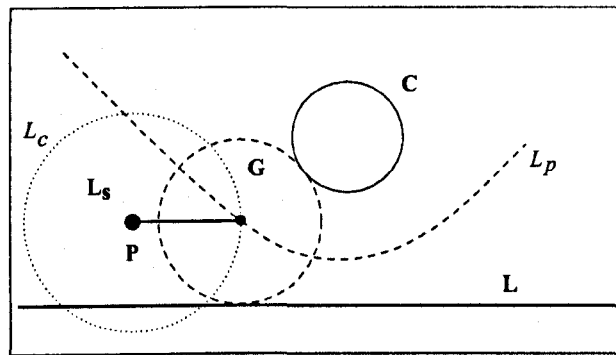


Figure 2.19: Solution for the Constraint Problem of Figure 2.18

An acyclic geometric constraint graph can easily be solved by Kramer's approach by iterative applications of action and locus analysis. In contrast, geometric loop problems are more complex. In this case, Kramer's algorithm uses a technique known as *contraction* [16]. This technique works by traversing the constraint graph and contracting sets of three nodes into a single node which is treated as a rigid body. The contraction process involves two action analyses and one locus analysis. For example, in Figure 2.18(b) consider the grounded nodes  $P$ ,  $L$  and  $C$  as a unique node referred to as  $RB$  (for Rigid Body). Thus, the graph now contains three nodes  $RB$ ,  $G$  and  $L_s$ . To contract these three nodes, the algorithm applies action analysis to fix  $G$  and  $L_s$  with respect to  $RB$  and then applies locus analysis to fix  $G$  and  $L_s$  with respect to each other. Given a geometric loop problem, contraction is iteratively applied until the entire graph is contracted into a single node [59]. However, Kramer's algorithm is unable to solve 2D geometric problems that involve the contraction of more than three nodes since it is necessary to use a numeric solution technique, such as the one suggested by [16]. Furthermore, the system still requires re-satisfaction from scratch when a new constraint is inserted into the constraint graph.

Pabon has presented a system referred to as HyperGEM [75] to support preliminary design. HyperGEM is also based on the Degrees of Freedom Analysis technique to satisfy geometric constraints for a specific set of configurations. The system allows the definition of engineering constraints which are solved through numeric techniques. After satisfying the engineering constraints, the values of variables specifying dimensional attributes are propagated to the set of geometric constraints. Figure 2.20 shows the basic architecture of the HyperGEM system. If a new geometric constraint is inserted, the entire set of geometric constraints is satisfied from scratch. The same happens with respect to the set of engineering constraints due to the insertion of a new engineering constraint.

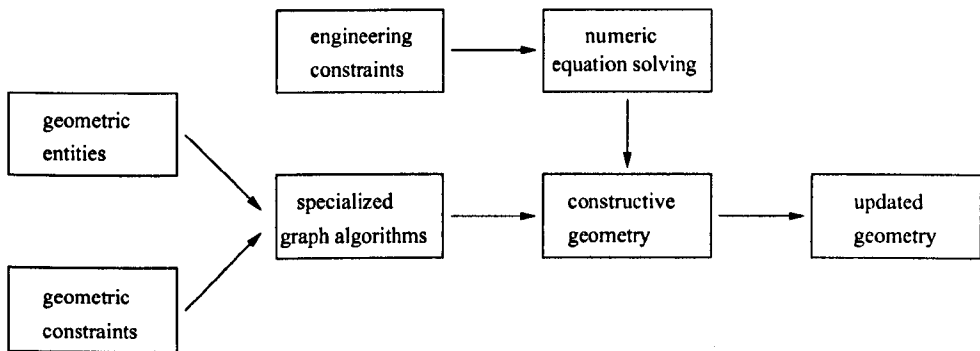


Figure 2.20: HyperGEM's Architecture for Constraint-driven Design

Bouma presents an algorithm that solves geometric constraint problems by creating and combining sets of geometries and constraints to form rigid bodies called *clusters* [14]. Given the undirected graph representing the geometric constraint problem, the algorithm first places any two geometric elements (graph nodes) with respect to each other, by means of construction steps.

The construction steps such as placing a point on a line, placing two lines at a given angle etc, correspond to solving standardised small systems of algebraic equations. The algorithm stops when no further cluster formation is possible. Once a cluster is formed, the algorithm starts constructing another cluster in the same way. All generated clusters are then recursively grouped by placing one with respect to the other, using a rigid body transformation.

To illustrate Bouma's approach, consider the user sketch of Figure 2.21(a), annotated with constraints, which is translated into the undirected graph of Figure 2.21(b). In this graph,  $d$  is a distance constraint,  $a$  an angle constraint and  $p$  represents the perpendicularity. A



*tangent* constraint is expressed as a distance constraint between the centre of the circle and the line tangent to the circle. All the other graph edges represent incidence. Line segments are defined by specifying that the distance from the endpoints to the underlying line is zero. For instance, in the sketch of Figure 2.21(a), line segment  $a$  is given by zero distances from points  $A$  and  $B$  to its underlying line. Similarly, arcs are defined by distances from its endpoints to the centre of the underlying circle with fixed radius. For this reason, there is no node in the graph corresponding to arc  $c$ , but only distances to the underlying circle's centre  $X$ .

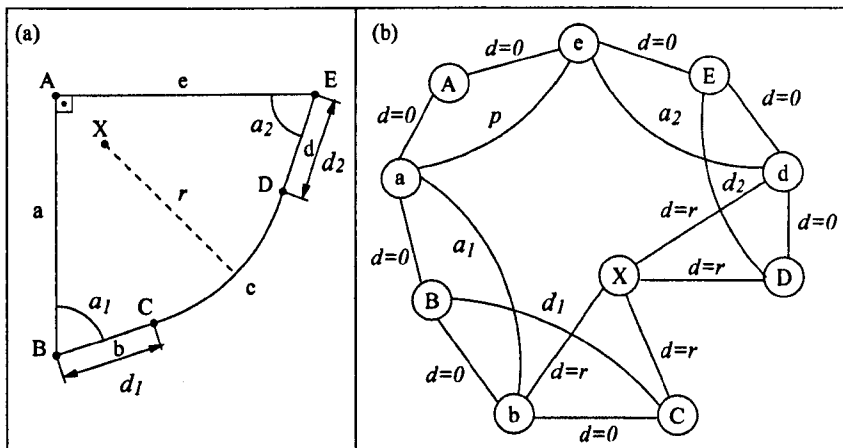


Figure 2.21: Sketch Example and Corresponding Undirected Constraint Graph

To start forming a cluster, the algorithm first picks any two geometric elements that are related by a constraint and place them with respect to each other. For example, the algorithm could start with nodes  $a$  and  $B$  (dotted curve 1 in Figure 2.22(a)). These two elements are now *known* and all other geometries are *unknown*. Next, the algorithm looks for any unknown geometric element with two constraints relating to the known geometric elements. Note that node  $b$  has two constraints related to the current known elements (distance and angle constraint). Thus, it is placed with respect to the known ones by a construction step (curve 2). Now, the placed geometric element  $b$  is also known. This process is repeated and nodes  $C$  (curve 3) and  $X$  (curve 4) are also inserted into the known set. Since no unknown geometric element with two constraints related to the known set is found, the algorithm stops and the first cluster  $U$  of Figure 2.22(b) is complete.

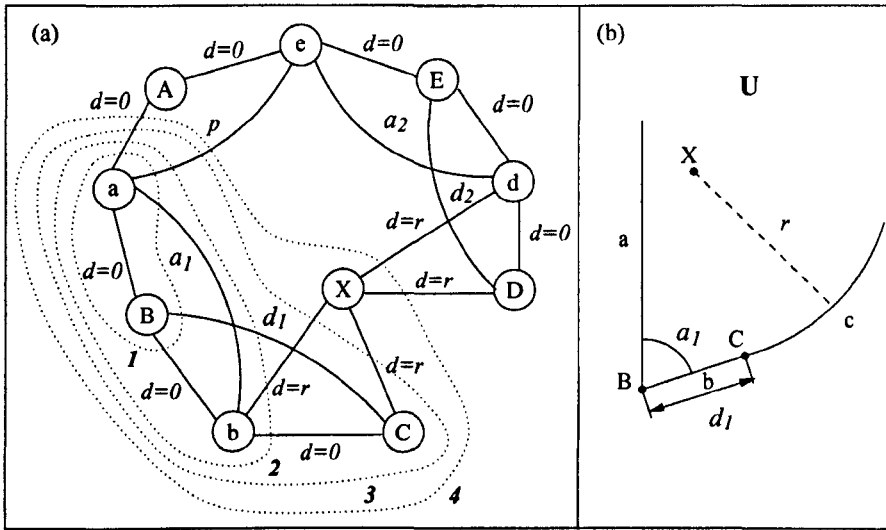


Figure 2.22: Cluster Formation from the Undirected Graph

Note that, at this point, the algorithm does not know where point *A* is situated and how far the arc *c* extends. Thus, the algorithm tries to form new clusters. In this example, two other clusters are determined. One consists of the circle center *X*, the points *D* and *E* and the line segments *d* and *e* (cluster *V*). The other consists of point *A* and the line segments *a* and *e* (cluster *W*).

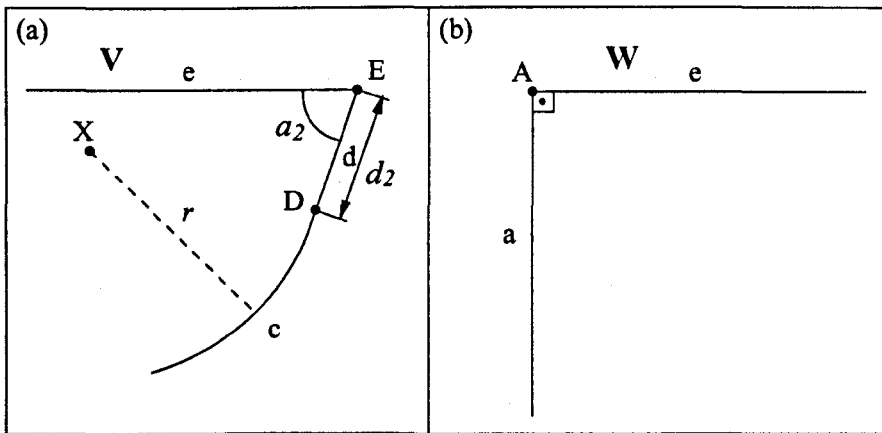


Figure 2.23: Clusters *V* and *W* of Figure 2.21

Note that for subsequent cluster formation, one of the two initial geometric elements may already belong to an existing cluster. Figure 2.23 shows these two remaining clusters. After all clusters are formed, the algorithm starts recursively placing one cluster with respect

to the other, using rigid body transformations. For example, the algorithm could start by fixing cluster **U**. Then, since clusters **U** and **W** shares the line segment  $a$ , cluster **W** is positioned with respect to cluster **U** through this line segment. Finally, by positioning cluster **V** with respect to cluster **W** (through line segment  $e$ ) the sketch of Figure 2.21 is generated.

Owen presents an algorithm that consist of breaking the graph into small subgraphs of constraints that can be solved quadratically and in turn combine these subgraphs to get the final solution [74]. The set of geometric constraints are first translated into an undirected connected graph. An undirected graph is said to be connected if every pair of nodes is connected by a path [25]. An **articulation pair** in an undirected connected graph  $G$  is a pair of nodes whose removal disconnects  $G$ .

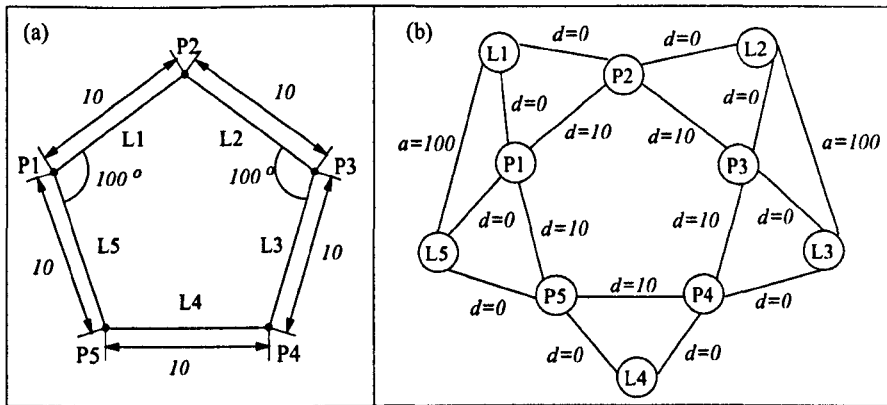


Figure 2.24: A Simple Dimensioned Drawing with Its Constraint Graph

Owen's algorithm has been used to develop the Dimensional Constraint Manager which is a constraint-based design system commercially available by D-Cubed Ltd [26]. Figure 2.24(a) shows a simple drawing with dimensional constraints and its respective undirected constraint graph in Figure 2.24(b).

Similarly to Bouma's approach, Owen's algorithm decomposes the constraint graph into smaller subgraphs that can easily be solved quadratically. However, the decomposition process is different. First, DCM looks for one articulation pair in the constraint graph and splits the graph into two subgraphs. Note that in Figure 2.25 the nodes  $P2$  and  $P5$  define an articulation pair which is, in turn, used to break the original constraint graph into *Subgraph I* and *Subgraph II*. Next, the splitting process is repeated for each subgraph. For instance, in *Subgraph I* the nodes  $L1$  and  $P1$  form another articulation pair. Thus,

this subgraph is also decomposed through the articulation pair deriving the *Triangle 1* subgraph. The remaining subgraph can also be broken since the nodes *L5* and *P1* form another articulation pair. Thus, *Triangle 2* and *Triangle 3* subgraphs are also determined. The process is equally repeated for *Subgraph II*. The algorithm stops when no further decomposition is allowed (no more articulation pairs are found). Finally, Owen's algorithm combines the triangles in a similar way to that of Bouma's when grouping clusters. At the end of the splitting process, only triangle subgraphs and/or single edges are expected. Remaining subgraphs with more than three nodes are solved numerically.

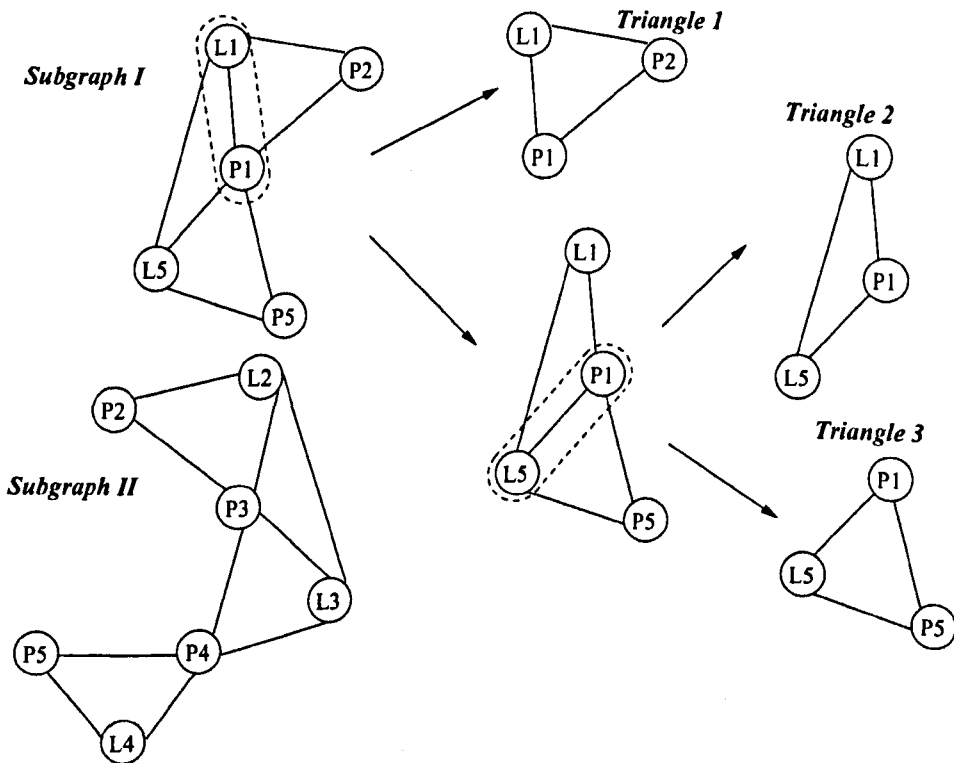


Figure 2.25: Constraint Graph Decomposition in DCM

Although the global approach has been recognised as suitable for interactive response [89], its constraint re-satisfaction process may hinder immediate feedback for engineers, when dealing with very large set of geometric constraints. Besides, this approach does not provide tools to support the representation and satisfaction of engineering constraints [5], which restricts its application to preliminary design.

## Local Propagation Approach

In the **local propagation approach**, constraints are satisfied one at a time following the user's design sequence. Every geometric constraint inserted is locally accommodated into the current solution graph. Constraints are always specified from the known to the unknown geometries and are solved incrementally.

Based on this approach, Rossignac proposes a technique whereby CSG representations are specified through graphic input, in terms of constraints between boundary elements [86, 87]. Geometric constraints imposed on CSG solid models are independently evaluated through operational transformations in a user-specified-order. Although this technique avoids the problem of converting the constraints into a large set of equations, it poses two problems:

1. the user is responsible for deriving the sequence of operations as well as dealing with conflicting constraints;
2. the technique is not able to solve problems where many constraints must be satisfied simultaneously (constraint cycles).

At Leeds University, Fa has extended Rossignac's approach to support assembly modeling within virtual environments. He has presented an Interactive Constraint-based Solid Modeler (ICBSM) which avoids the re-satisfaction of constraints during constraint insertion [32]. In this approach, a directed graph is used to represent a geometric constraint problem. Geometric constraints are automatically recognised through direct manipulations within a virtual environment. A technique referred to as Allowable Motion is used to support constraint satisfaction and direct manipulations of under-constrained models, according to their DOFs. When a geometric entity is being manipulated, it propagates rigid body transformations to its children nodes, according to the dependency established by the arc direction. Figure 2.26 illustrates the principles of ICBSM. In Figure 2.26(a) the user can grab and manipulate block B since it is free to translate and rotate in space. The user can specify an *Against* constraint between the bottom face of block B and the top face of block A (dark faces) by moving block B towards block A. An *Against* constraint is recognised when the two surfaces are within a predefined tolerance. This constraint is represented in the directed graph as shown in Figure 2.26(b). At this moment, the degrees of freedom of block B are re-calculated and its current allowable motion is captured and maintained in the graph. For example, in this case, the allowable motions for block B are translation on the plane (top

face of A) and rotation about any axis normal to the plane of block A's top face. If the user translates block A, this motion is propagated to block B. Next, by moving block B to the right, ICBSM similarly recognises that a *Coincident* constraint can be imposed between the right faces of both blocks. Thus, the final geometry and direct graph are again updated as shown in Figure 2.26(c).

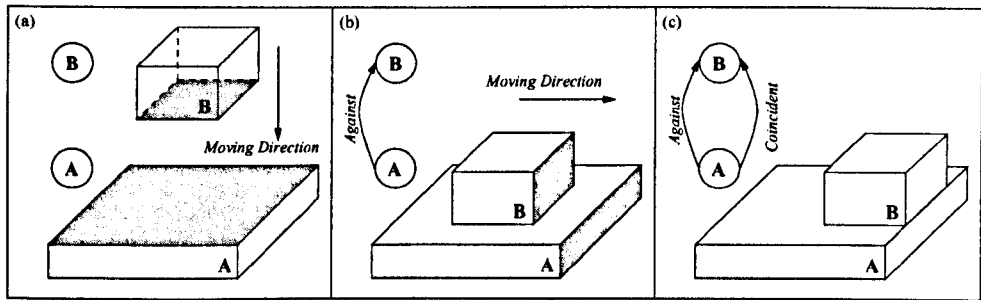


Figure 2.26: Integrating Geometric Constraints and Direct Manipulations in ICBSM

By introducing Allowable Motion and rigid body transformations, Fa avoids the updating of geometry through the use of numeric/algebraic techniques. This provides faster geometric constraint solution. Munlin [70] has extended Fa's work by supporting more complex assembly geometric constraints such as *screw fit*, *gear contact* etc.

However, Fa's ICBSM is unable to solve geometric constraint loops when simulating closed-loop mechanisms. Recently, Tsai has solved this problem by introducing an incremental geometric constraint solver [110]. Nevertheless, as mentioned before, these approaches are unable to represent engineering constraints and are therefore not adequate to be used in the early stages of design. Similar techniques have been reported in [6, 30, 31, 57].

## 2.4 Summary and Conclusions

This chapter has presented a survey of recent progress in constraint-based design. Two main approaches have emerged to support engineering design defined in terms of constraints: the **Equation-based** approach and the **Geometric Constructive** approach. The Equation-based approach translates every constraint into an equation and thus allows the coupling of geometric and engineering constraints. For this reason, researchers argue that this approach provides a better mathematical platform to support preliminary engineering functions such as Tolerance Analysis and Sensitivity Analysis [23, 27]. Although the numeric [62, 61, 101]

and symbolic [18, 56] approaches are general enough to solve these equations, they are time consuming and not robust. The global graph based approach [93] provides a better performance but still could be expensive for large sets of constraints, since it satisfies the constraint network from scratch, whenever a new constraint is inserted. Algorithms based on the local propagation approach provide the best performance for equation-based systems [50, 91, 117]. Zanden's approach seems to be a better algorithm for supporting design since it doesn't force the designer to specify strength values for each constraint as in SkyBlue. However, this algorithm doesn't handle nonlinear equations and constraint cycles in its current form.

The Geometric Constructive approach deals mainly with geometric constraints. The techniques developed under this approach mainly use geometric knowledge and degrees of freedom analysis to provide efficient constraint satisfaction algorithms. The rule-based approach [2, 15, 100, 116, 112] has proven to be still computationally expensive due to its rule unification process. The global graph-based approach [14, 16, 60, 74], although more efficient than pure numeric solvers, still perform the constraint satisfaction from scratch whenever a new constraint is inserted. Better computer performances have been achieved with the algorithms based on the local propagation approach [6, 32, 110]. However, this approach satisfies the constraints one at a time, in response to the designer's sequence.

Inasmuch as the Geometric Constructive Approach provides constructive steps to satisfy a set of geometric constraints, it provides a faster constraint satisfaction process than those approaches that rely on numeric techniques [89]. Furthermore, since it provides a high level representation of the geometric entities this approach has the potential to support direct manipulation techniques which allow the development of highly interactive constraint-based systems [100, 33]. Nevertheless, the Geometric Constructive approach as a whole doesn't provide a unique representation for integrating engineering and geometric constraints.

A major conclusion from this survey is that most current constraint-based design systems re-satisfy the set of design constraints from scratch, to satisfy a newly imposed constraint. This can hinder interactive performance when dealing with design models defined with very large sets of constraint equations. However, the local propagation approach is the best way to support highly interactive constraint-based design systems. Furthermore, the survey also concludes that the Equation-based approach is more appropriate for supporting preliminary design.

## Chapter 3

# Incremental Engineering Constraint Satisfaction

---

### 3.1 Introduction

This chapter presents a set of techniques for supporting the incremental satisfaction of engineering constraints. A graph structure is used to represent a set of engineering constraints which are expressed as equalities. The underlying constraint satisfaction algorithms are based on the local propagation approach because it is computationally efficient. These algorithms maintain an evolving solution of the engineering constraint set.

The remainder of this chapter is organised as follows. Section 3.2 shows how engineering constraints are represented through a graph structure. Section 3.3 describes an incremental equation solver to speed up the constraint satisfaction process. Finally, a summary of this chapter is given in Section 3.4.



## 3.2 Engineering Constraint Representation

### 3.2.1 Basic Techniques

This section presents the basic techniques used to represent and solve design constraints. In [103], the term **constraint** is defined as *a relation stating what should be true about one or more objects*. In the **Equation Graph**, objects are variables and a constraint equation defines a relation among its variables. Thus, the Equation Graph which is also referred to as a **constraint network** is a declarative structure which expresses relations among equations and variables.

The Equation Graph is a bipartite graph that maintains the constraints. The two disjoint sets of this bipartite graph are the class of nodes representing the constraint equations ( $C$ ) and the class of nodes representing the variables ( $V$ ). Thus, the set  $E$  of edges (arcs) is defined as:  $E = \{(v, c) \in V \times C \mid v \text{ is constrained by } c \text{ (i.e. } v \text{ belongs to } c)\}$ .

Figure 3.1(a) shows how the graph maintains the constraint  $c1 : a = b + c$ . In this figure, variables are represented as ellipses and constraints as hexagons. Any time a constraint is inserted, it is solved for one of its variables (according to the algorithm presented in Section 3.3), establishing a constraint-variable dependency in the graph. For example, the fact that variable  $a$  is chosen to solve for  $c1$  is represented by the arc direction given in Figure 3.1(b). In this case,  $a$  is called the *output variable* of  $c1$  while  $b$  and  $c$  are called the *input variables*. This means that once given the values of the input variables  $b$  and  $c$ , constraint  $c1$  can be satisfied. The value of a variable  $v$  can be fixed through an equation such as  $c_i : v = k, k \in \mathbb{R}$ .

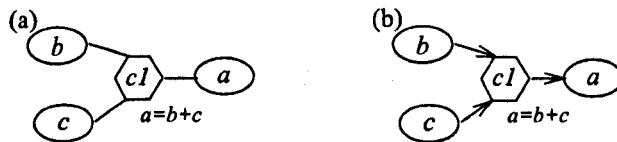


Figure 3.1: Constraint Representation and Constraint Satisfaction

#### *Method Graphs*

A constraint is said to be satisfied when the value of its output variable is calculated. Different constraint satisfaction techniques are provided, according to a constraint's linearity. Each linear constraint is associated with methods, called *method graphs*, which are used

to satisfy the constraint. These methods are procedures that read the values of the input variables and calculate the value of the output variable [117, 90, 50]. For example, consider constraint  $c1 : a = b + c$ , in Figure 3.1. This constraint can be satisfied by any of the following methods :  $a := b + c$ ,  $b := a - c$  or  $c := a - b$  (see Figure 3.2).

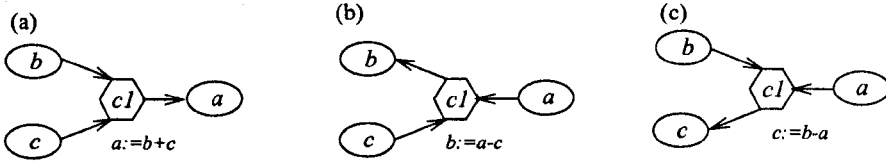


Figure 3.2: Satisfaction Method Graphs for  $c1 : a = b + c$

This means that a constraint can be satisfied by the execution of any of its methods. The selected method only depends on the output variable. In addition, in Figure 3.2(a), the constraint  $c1$  could be maintained by executing the method  $a := b + c$ , even if the value of  $b$  or  $c$  (or both) were changed. This property is very important for design circumstances where the designer wants to try different configurations by only changing the value of variables.

Given a set of constraints with their selected output variables, the constraint network is viewed as a directed bipartite graph. For example, Figure 3.3 shows an Equation Graph for the constraints  $c1 : a = b + c$  and  $c2 : e = a + d$ . In this graph, real numbers attached to each constraint node (i.e. 100, 150) indicate the order of satisfaction with lower numbers being satisfied before higher numbers. Thus, according to Figure 3.3, the order for satisfying the constraints is  $c1, c2$ .

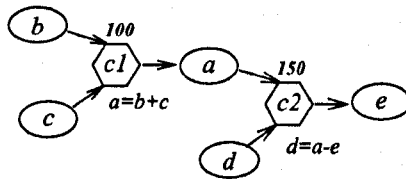


Figure 3.3: Number Attachment to Represent the Satisfaction Sequence

*Nonlinear Constraints*

If the engineering constraint is a nonlinear constraint, then it is symbolically processed according to its output variable [28, 21, 81]. The NAG-Fortran [71] library is used to solve nonlinear constraints. An arbitrary solution is chosen from the multiple solutions presented in a nonlinear constraint.

A constraint's linearity is automatically determined when the constraint is created and set as a flag for the constraint node. However, when updating the constraint dependency within the Equation Graph, it is not known if a constraint is linear or nonlinear. The identification of the constraint's linearity takes place only during constraint satisfaction, where linear constraints are satisfied through method graphs and nonlinear constraints through symbolic processing.

### 3.2.2 Engineering Constraint Networks

Given a set of engineering constraints with their selected output variables, the engineering constraint network is viewed as a directed bipartite graph. Consider, for example, the four-bar mechanism in Figure 3.4(a) where *Bar1* is fixed. This figure represents a simple design problem (adapted from [45]), where the length of each bar is given by  $d$ ,  $a$ ,  $b$  and  $c$ . From Figure 3.4(a) the Freudenstein's equation, which relates the input angle  $\theta$  to output angle  $\psi$  as a function of the sizes of the linkages, is expressed as:

$$k_1 \cos \theta + k_2 \cos \psi - k_3 = \cos(\theta - \psi) \tag{3.1}$$

where,

$$c1 : k_1 = d/c; \tag{3.2}$$

$$c2 : k_2 = d/a; \tag{3.3}$$

$$c3 : k_3 = (a^2 - b^2 + c^2 + d^2)/2ac. \tag{3.4}$$

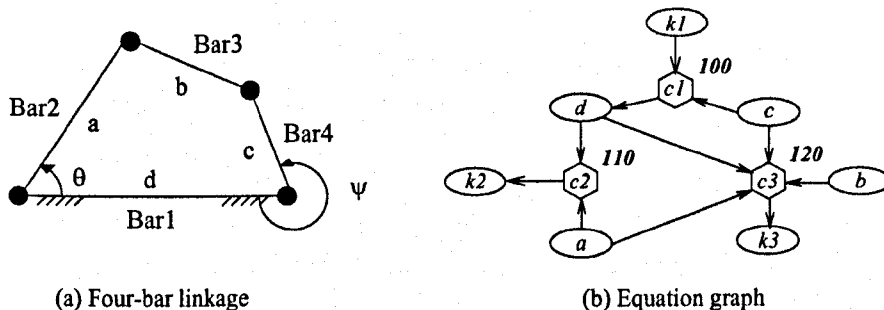


Figure 3.4: Four-bar Linkage and a Solution Graph

An Equation Graph, which consists of the engineering constraints  $c1$ ,  $c2$  and  $c3$ , is shown in Figure 3.4(b). In this graph, numbers attached to each constraint node (i.e. 100, 110

and 120) indicate the increasing order of constraint satisfaction. Thus, according to Figure 3.4(b), the order for satisfying the constraints is:  $c_1$ ,  $c_2$  and  $c_3$ . Note that it makes sense to satisfy constraint  $c_1$  before constraint  $c_2$ , since variable  $d$  is an input variable for  $c_2$  and an output for  $c_1$ .

Note that for the current graph of Figure 3.4(b) the output variables are  $d$ ,  $k_2$  and  $k_3$ . If the user is satisfied with this configuration and is looking for the calculated values for these output variables, he needs to specify the values of the input variables ( $k_1$ ,  $a$ ,  $b$ ,  $c$ ) according to the current Equation Graph.

### 3.3 Incremental Equation Solver

This section presents an Incremental Equation Solver, referred to as **INCES**. Based on local propagation techniques, this solver is designed to speed up the constraint satisfaction process. INCES locally accommodates a newly inserted constraint within the Equation Graph. The INCES algorithm has extended Vander Zanden's approach in order to support preliminary design. As pointed out in Chapter 2, Vander Zanden's approach does not support either nonlinear constraints or constraint cycles. This poses a significant drawback for design applications since simple geometric relationships, such as distance and orientation, as well as some design specifications, give rise to nonlinear constraints [41]. In addition, constraint cycles as well as under- and over-constrained situations are constantly emerging according to the designer's input and must be handled robustly [16, 75]. Therefore, INCES has made the following improvements to Vander Zanden's approach:

- Nonlinear constraints are allowed in the constraint network. These constraints are symbolically processed according to their output variable and are solved by a numerical equation solver (Section 3.2.1);
- Constraint cycles are locally identified and solved. These cycles are evaluated and handled according to their constraint states, i.e. fully, under- or over-constrained;

#### 3.3.1 Inserting Engineering Constraints

This section discusses how INCES allows the incremental insertion and satisfaction of engineering constraints and also how constraint cycles are locally handled. The following case

study (adapted from [93]) is used to illustrate the algorithm. The case study presents the problem of designing a cantilever beam. The relevant engineering constraints are presented below and their respective geometries are shown in Figure 3.5.

$$C1 : \sigma - \frac{MY}{I} = 0 \quad (3.5)$$

$$C2 : M - FL = 0 \quad (3.6)$$

$$C3 : I - \frac{WH^3}{12} = 0 \quad (3.7)$$

$$C4 : Y - \frac{H}{2} = 0 \quad (3.8)$$

$$C5 : K - \frac{3EI}{L^3} = 0 \quad (3.9)$$

$$C6 : \phi - \frac{FL^2}{EI} = 0 \quad (3.10)$$

$\sigma$  is the bending stress in the beam,  $I$  is the cross sectional area moment of inertia,  $M$  is the maximum bending moment due to the load  $F$ ,  $Y$  is the distance from the neutral axis to a fiber on the surface of the beam,  $K$  is the stiffness of the beam and  $E$  is the Young's modulus of elasticity.  $L$ ,  $H$  and  $W$  are the length, height and width of the beam, respectively.  $\phi$  is the slope of the beam. The problem is to identify the values of the dimensions  $L$ ,  $H$  and  $W$ , given  $\phi$ ,  $M$ ,  $E$ ,  $K$  and  $\sigma$ .

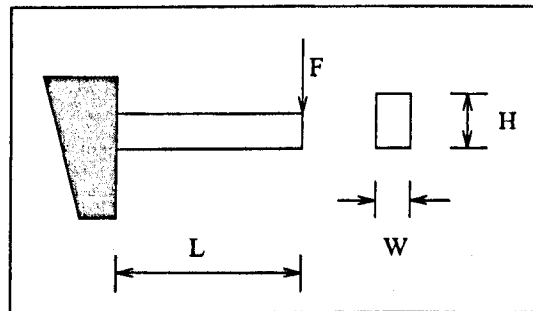


Figure 3.5: The Cantilever Beam Problem

In building up this problem, each time a constraint is inserted, the INCES algorithm carries out the following steps to update the Equation Graph.

- **STEP 1:** Search the Equation Graph to find a free variable to absorb the new constraint. A free variable is a variable that belongs to only one constraint. During

this step, the variables of the new constraint are checked for a free variable. If this fails, then the algorithm searches for a free variable in the ancestor subgraph of the variables of the new constraint.

- **STEP 2:** *Update the Equation Graph locally.* This step locally derives a constraint satisfaction plan. If there is a free variable in the new constraint, this variable is set as the output of the new constraint. This new constraint is then inserted into the constraint satisfaction plan after the maximum numbered constraint of its input variables. In situations where a free variable is only available in the ancestor subgraph, the constraint satisfaction plan is locally updated from the free variable down to the new constraint.
- **STEP 3:** *Satisfy the Equation Graph locally.* This step performs the constraint satisfaction process for the affected area of the Equation Graph.

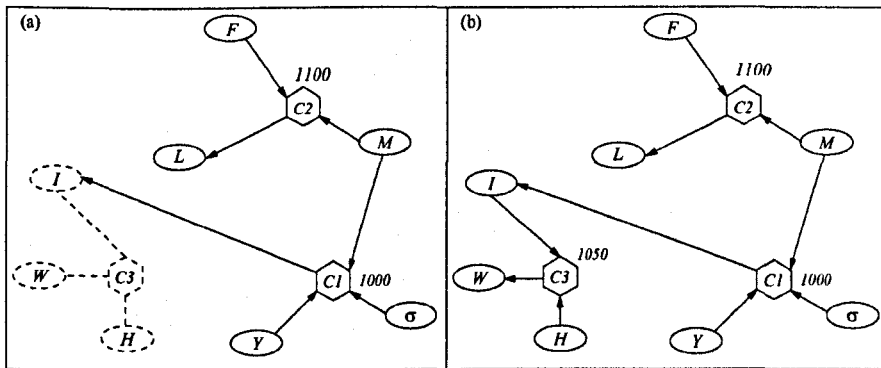


Figure 3.6: Inset Free-variable Constraint and its Updated Equation Graph

For example, consider the case in Figure 3.6(a) where the user intends to insert constraint  $C3$  into the Equation Graph which currently consists of constraints  $C1$  and  $C2$ . First, the algorithm searches for free variables in the inserted constraint and identifies that variables  $H$  and  $W$  are free variables for constraint  $C3$  (STEP 1). When updating the constraint dependency it is important to have only one output variable for each constraint. This corresponds to the triangularisation process for local propagation techniques as explained in Section 2.2.3. Therefore, any of these two variables can be selected as the output variable. The algorithm chooses, for example, variable  $W$ . The next step is to insert the constraint in the constraint satisfaction plan (STEP 2). Note that a constraint can only be satisfied if the values of its input variables are provided. Thus, this new constraint must be satisfied

after the constraints which calculate the value of its input variables. Therefore,  $C3$  must be satisfied after  $C1$  since variable  $I$  is the output variable for  $C1$  and input variable for  $C3$ . Thus, the new constraint satisfaction plan is  $C1, C3$  and  $C2$ . The numbers on the constraint nodes have been updated to reflect the new constraint satisfaction plan (Figure 3.6(b)). Constraint  $C3$  can now be locally satisfied (STEP 3).

Under certain situations, the inserted constraint may not have a free variable. This is the case in Figure 3.7 when one tries to fix the value of variable  $\phi$  by inserting constraint  $C7: \phi = 3$ . Note that variable  $\phi$  is not a free variable, since it belongs to constraints  $C6$  and  $C7$ . In this case, the INCES algorithm searches the *ancestor subgraph* of the variables of the new constraint looking for a free variable (STEP 1). A node  $v$  is called an *ancestor* of node  $w$  if there is a directed path from  $v$  to  $w$ . The ancestor subgraph of variable  $\phi$ , due to the insertion of constraint  $C7$ , is shown in Figure 3.7.

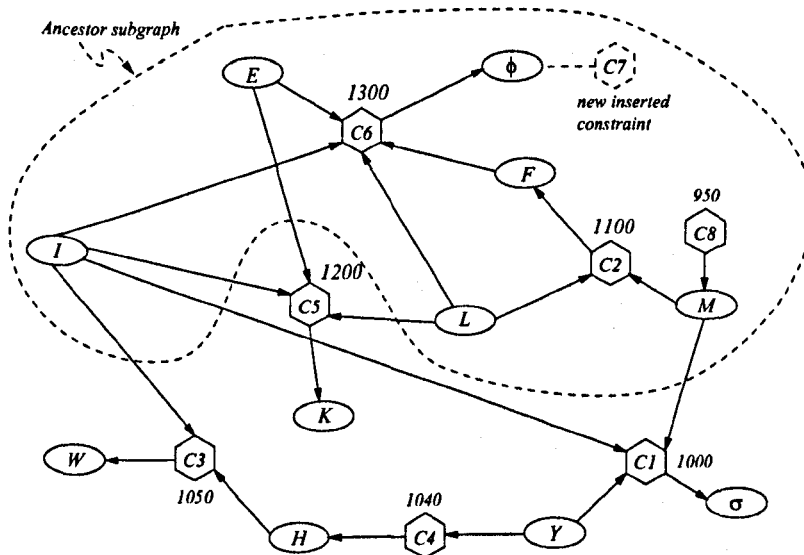


Figure 3.7: An Ancestor Subgraph

The ancestor subgraph corresponds to the set of constraints that can be triangularised (Section 2.2.3) with the inserted constraint  $C7$ . The new constraint can only be accommodated and solved using local propagation techniques if a triangularised constraint set can be found within the ancestor graph (i.e. if there is a free variable within the ancestor graph). However, sometimes only part of the ancestor graph needs to be processed to accommodate the new constraint. This is explained below with an example.

The new constraint  $C7$  is the first to be visited in the ancestor subgraph. A constraint node is always visited with its associated variables. Therefore, variable  $\phi$  is considered visited with  $C7$ . Every variable of a visited constraint is checked for the status of free variable in order to “absorb” the new constraint. Since variable  $\phi$  is not a free variable, the search continues by visiting its ancestor node  $C6$ .

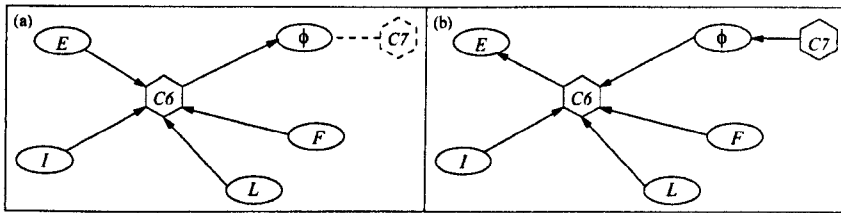


Figure 3.8: Local Propagation During Solution Graph Construction

During this process, only the input variables are potential candidates to become free variables. For example, as the construction reaches  $C6$ , the algorithm detects that  $E$ ,  $L$  and  $I$  are free variables, since they belong to only  $C6$  in the ancestor subgraph. Variable  $E$  is selected as a free variable, in this example. Then, the algorithm starts defining the new constraint satisfaction plan by re-writing the constraint dependency in the Equation Graph from variable  $E$  (STEP 2). Variable  $E$  is then set as the output for constraint  $C6$  and its constraint dependency is updated as shown in Figure 3.8(b). The algorithm takes care to insert  $C6$  before  $C5$  in the satisfaction sequence, since  $E$  is now one of  $C5$ 's input variable.

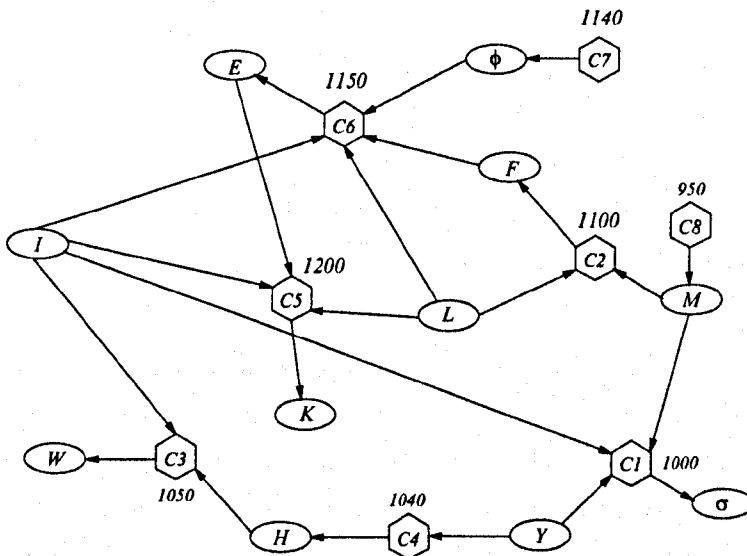


Figure 3.9: Updated Solution Graph After  $C7$  Insertion



Local propagation then continues by setting  $\phi$  as the output for  $C7$ . Again, the algorithm takes care to insert  $C7$  before  $C6$  in topological order because  $\phi$  is now one of  $C6$ 's input variables. Therefore, the constraint dependency inside the Equation Graph is re-written without the need to visit other ancestor variables such as those belonging to  $C2$  and  $C8$ . Figure 3.9 shows the updated Equation Graph (STEP 2). Finally, only the affected area of the the graph (constraints  $C7$ ,  $C6$  and  $C5$ ) are locally re-satisfied through constraint propagation (STEP 3).

### 3.3.2 Engineering Constraint Cycles

If a free variable cannot be found in the ancestor subgraph, as explained in the previous section, the inserted constraint cannot be eliminated through local propagation. This means that the constraint network contains cycles, as explained in Section 2.2.3. In such situations, the INCES algorithm locally solves this set of simultaneous constraints.

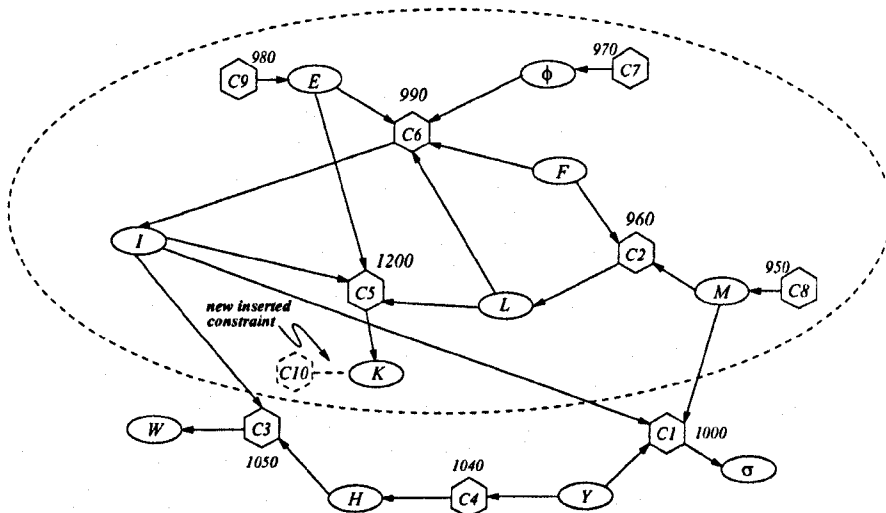


Figure 3.10: Current Equation Graph Before Fixing the Value of  $K$

Figure 3.10 shows the current equation graph when one tries to fix the value of variable  $K$  (constraint  $C10$ ), after inserting all the six constraints and fixing the values of  $\phi$ ,  $M$  and  $E$ , through constraints  $C7$ ,  $C8$  and  $C9$ , respectively. As can be noted, the ancestor subgraph (dashed ellipse) will not have a free variable in this case. In INCES, such constraint sets are solved simultaneously, using numerical techniques (Newton-Raphson method). However, in order to provide a more efficient solution, the algorithm tries to reduce the size of the

Jacobian matrix which is decomposed during each iteration of Newton-Raphson [53, 79], by minimising the constraint set to be solved simultaneously.

First, the algorithm traverses the ancestor subgraph and eliminates those constraints which fix the value of a variable (*FixValue* constraint,  $C_i : v = \alpha, \alpha \in \mathbb{R}$ ) and keeps their attached numbers. These constraints are represented as smaller dashed ellipses in Figure 3.11. The number attached to the remaining constraints  $C_2, C_5$  and  $C_6$  are removed since these constraints will be solved at the same time.

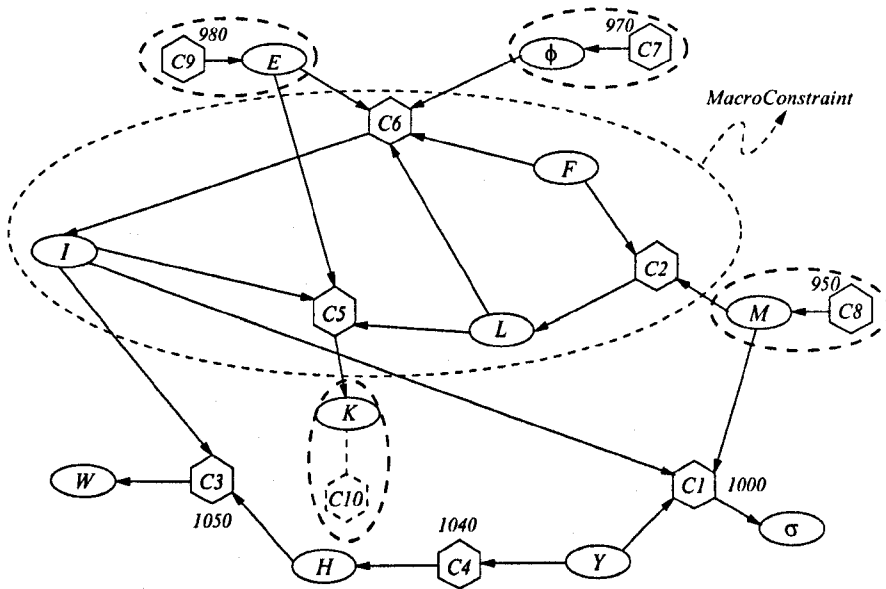


Figure 3.11: Constraint Cycle Updating

In order to represent the constraint cycle, the algorithm creates another constraint called a *MacroConstraint*. This constraint is a node in the Equation Graph which contains a set of constraints to be solved simultaneously through numerical techniques (larger dashed ellipse in Figure 3.11). At this point, if the newly inserted constraint is of the type *FixValue*, it is inserted in the constraint satisfaction plan after the highest-numbered *FixValue* constraint inside the ancestor subgraph. The *MacroConstraint* is, in turn, inserted after the inserted constraint. If the new constraint is not of the type *FixValue* then it is included into the *MacroConstraint*.

Precautions should be made to avoid taking the variables inside the *MacroConstraint* as the output variables for the subsequently inserted constraints. Therefore, the constraint dependency is rearranged in such a way that each variable inside the *MacroConstraint* is an

output for exactly one constraint. For example, in the graph of Figure 3.11,  $F$  is an input variable for constraints  $C2$  and  $C6$ . It must be rearranged so that it becomes an output variable to one constraint inside the cycle.

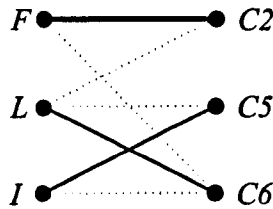


Figure 3.12: A Maximum Matching Inside the MacroConstraint

This interdependency updating is achieved by finding a maximal matching in the bipartite graph composed of the variables and constraints inside the MacroConstraint (Section 2.2.3). Figure 3.12 presents a maximum matching in the bipartite graph of the MacroConstraint of Figure 3.11. The dashed lines indicate the assignments which were not used. Each variable in this matching is set as the output of its matched constraint.

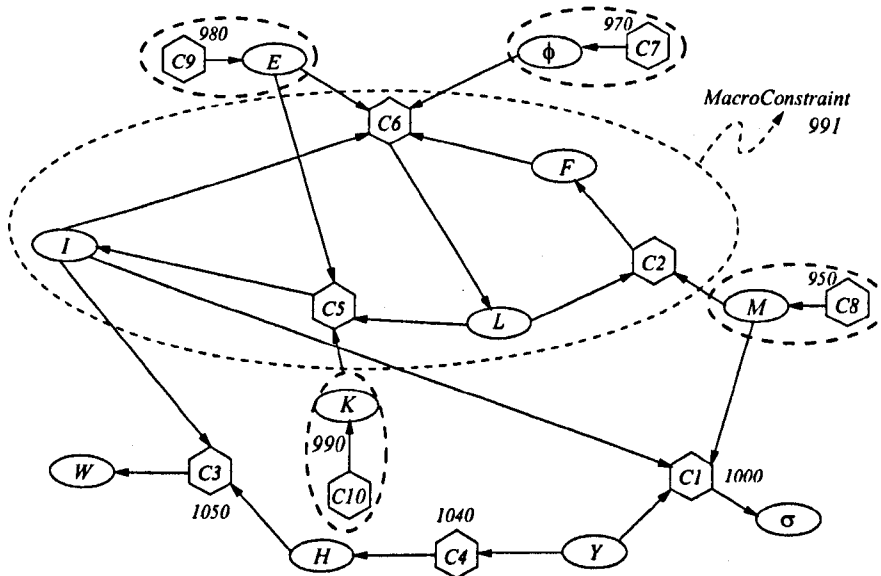


Figure 3.13: Current Solution After Fixing the Value of  $K$

The constraint dependency inside the MacroConstraint is updated accordingly as in Figure 3.13. Now, during the constraint satisfaction process in topological order, the algorithm relies on numerical techniques when it reaches a constraint cycle and applies the methods associated to the other constraints.

In addition, when another constraint is inserted, the algorithm avoids visiting the internal constraints of a MacroConstraint when searching the ancestor subgraph for a free variable. The solution of the constraints inside a MacroConstraint is propagated downstream to the remaining set of constraints in the graph. Therefore, the MacroConstraint does not have any ancestors other than the *FixValue* constraints and thus the algorithm saves time by not visiting them. Besides, the constraints inside a MacroConstraint will remain non-triangular after the insertion of another constraint. Figure 3.14 shows how a MacroConstraint behaves inside the Equation Graph. After being calculated through numerical techniques, the values of the variables inside a MacroConstraint are just propagated downstream to the descendant nodes during constraint propagation.

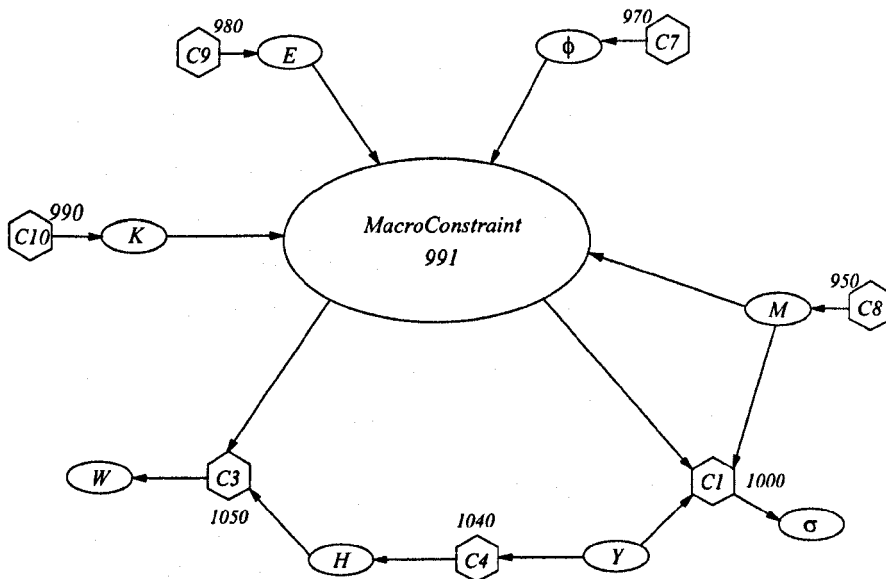


Figure 3.14: Downstream Constraint Propagation from a MacroConstraint

INCES also handles under- and over-constrained cycles. A constraint cycle is said to be under-constrained if it has more variables than equations and it is over-constrained if it has more equations than variables. A fully constrained cycle has an equal number of constraints and variables. Identifying and correctly solving such cases is very important in real-world design. When the constraint cycle is under-constrained, INCES asks the user to fix the value of some variables to make the cycle fully constrained. For example, if the cycle has four variables and three constraints, the algorithm asks the user to fix one of the variables. For over-constrained cases, excessive constraints are deleted through user intervention. The cycle is then solved as shown above. Consistency of over-constrained cycles [55] is not

evaluated in the current version of the algorithm.

Since INCES locally identifies a subset of constraints which is under- or over-constrained, it is able to provide efficient debugging tools to help the user analyse and handle such situations. This approach is more appropriate for interactive design than previous systems which identify under- and over-constrained networks by considering the entire constraint graph [61, 93].

### 3.4 Summary

This chapter presented a set of techniques to support the incremental satisfaction of engineering constraints. A directed graph, referred to as an **Equation Graph** is used to represent engineering constraints which are expressed through linear and nonlinear equations. The chapter also described an incremental equation solver (INCES). The main contribution of this solver is that it allows highly nonlinear equations to be incrementally inserted and satisfied in the Equation Graph. Engineering constraint cycles are also locally identified and solved. Only a small part of the Equation Graph is visited due to a constraint insertion.

The next chapter discusses how these incremental techniques are extended to support the incremental satisfaction of a coupled set of engineering constraints and geometric constraints.

## Chapter 4

# Engineering and Geometric Constraint Coupling

---

### 4.1 Introduction

Chapter 3 has presented a set of algorithms for incrementally satisfying engineering constraints. This chapter shows how these algorithms are extended to support geometric constraints. As already pointed out, the coupling of geometric and engineering constraints is very important to support early design stages [23]. In fact, the geometric attributes of the constituent of a part are in general defined by the satisfaction of engineering constraints [23, 75]. This chapter also discusses how the degrees of freedom of geometric entities are explored to support direct manipulations of under-constrained geometry.

The remainder of this chapter is organised as follows. Section 4.2 discusses how geometric entities with their respective degrees of freedom are represented in the Equation Graph. Section 4.3 presents the equations which are derived from a set of geometric constraints. Section 4.4 discusses how the equations derived from geometric entities and geometric constraints are coupled with engineering constraints in the Equation Graph. Section 4.5.1 describes a high level graph, referred to as *Relationship Graph*, which is used to represent geometric entities. The techniques used to support direct manipulations of under-constrained geometry are explained in Section 4.5. Finally, a summary of this chapter is given in Section 4.6.

## 4.2 Representation of Geometric Entities

This section explains how geometric entities are represented as equations within the Equation Graph. The values used to represent a geometric entity are referred to as *characteristic elements*. The degrees of freedom of a geometric entity are associated with these characteristic elements. The basic geometric entities supported are lines, circles and arcs. In the following, the thesis explains how these geometries are represented.

### 4.2.1 Line Segments

The representation of a geometric entity in the Equation Graph is achieved through the concept of *soft constraints*. *Soft constraints* are constraints imposed on the initial (default) values of the characteristic elements of a geometric entity to express its degrees of freedom. This means that there is a one-to-one mapping between degrees of freedom and soft constraints.

For example, a line segment  $a$  is represented by its starting point  $P1(x_{1a}, y_{1a})$ , slope  $\theta$  and its length  $a$ , as in Figure 4.1(a). Thus, when a line segment is created, soft constraints are imposed on these variables, to represent translational, rotational and dimensional DOFs. A line segment in the plane has four degrees of freedom which are represented by the soft constraints on the characteristic elements. I.e.

2 Translational DOFs	→	soft constraint on $(x_{1a}, y_{1a})$
1 Rotational DOF	→	soft constraint on $\theta$
1 Dimensional DOF	→	soft constraint on $a$

In addition, constraints  $C1$  and  $C2$  below are used to calculate the coordinates of the end point  $P2(x_{2a}, y_{2a})$  to display the line segment

$$C1 : x_{2a} = x_{1a} + a \times \cos \theta \quad (4.1)$$

$$C2 : y_{2a} = y_{1a} + a \times \sin \theta \quad (4.2)$$

The constraint network (Equation Graph) for a line segment is shown in Figure 4.1(b).

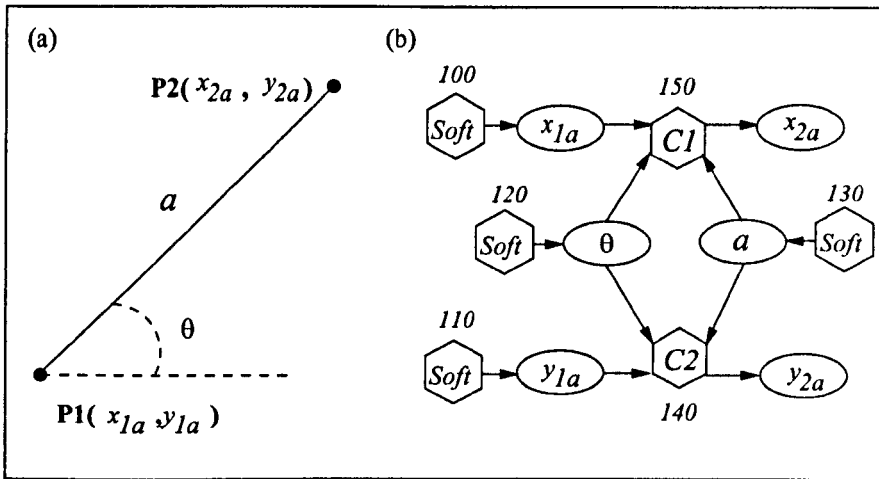


Figure 4.1: Equation Representation for a Line Segment

The direct manipulation of geometric entities is supported by changing the values of the soft constraints. For example, when the user wants to rotate a line segment, the values of  $\theta$  are updated and propagated to the constraint network, according to the constraint dependency.

### 4.2.2 Circles

A circle is represented by its center and radius. Note that in 2D, a circle has only three degrees of freedom: two translational and one dimensional. The translational degrees of freedom are represented by imposing soft constraints on the circle center and the dimensional degree of freedom by imposing a soft constraint on the radius. Therefore, given the circle  $C(C_x, C_y, r)$ , the following soft constraints are created:

$$C1 : C_x = k_1; \tag{4.3}$$

$$C2 : C_y = k_2; \tag{4.4}$$

$$C3 : r = k_3; \tag{4.5}$$

where  $k_1, k_2, k_3 \in \mathbb{R}$ .



4.2.3 Arcs

Arcs are represented by the two end points  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$  and the radius  $r$  of the underlying circle. Given these characteristic elements, the center  $C(x_c, y_c)$  of the underlying circle can be calculated and hence the arc can be derived. Two equations are used to calculate the center coordinates:

$$C1 : \|C - P_1\| = r \Rightarrow (x_c - x_1)^2 + (y_c - y_1)^2 = r^2 \tag{4.6}$$

$$C2 : \|C - P_2\| = r \Rightarrow (x_c - x_2)^2 + (y_c - y_2)^2 = r^2 \tag{4.7}$$

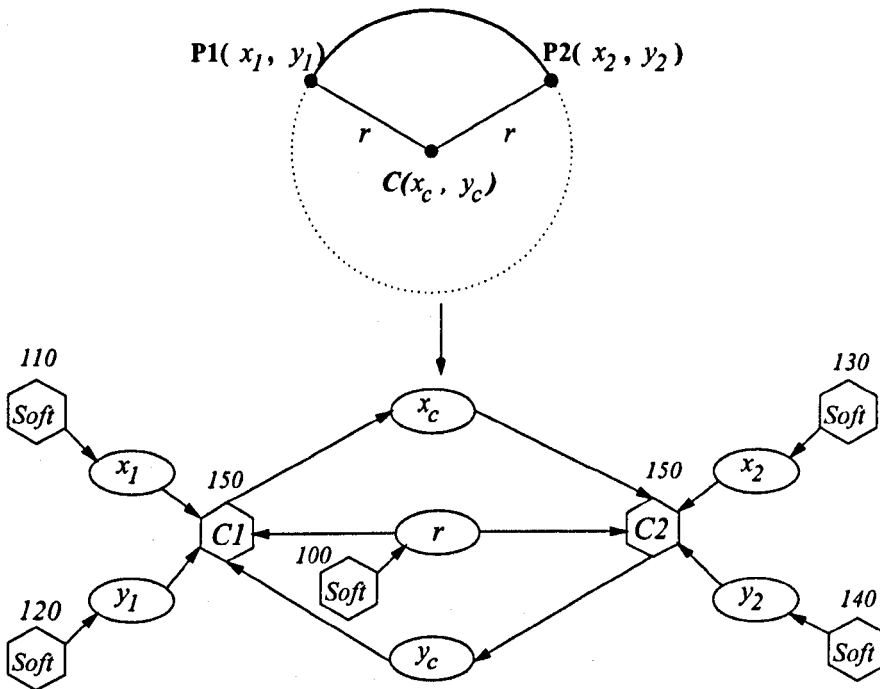


Figure 4.2: Equation Representation for an Arc

Figure 4.2 presents the constraint network that is created to represent an arc. Note that in this figure constraints  $C1$  and  $C2$  have the same number (150). This means that these constraints form a cycle (as shown by the arc flow) and are solved simultaneously. This procedure is explained in Section 3.3.2. When the cycle is solved, two solutions are found for the position of the centre  $C(x_c, y_c)$ . In turn, for each centre two other solutions can be derived for the arc: the solid arc and the dotted arc in Figure 4.2. This produces a total of 4 solutions. However, in the current implementation, one solution for the arc

centre is arbitrarily chosen and the arc is drawn in an anticlockwise direction from point  $P_2$  (right-hand side point) to point  $P_1$ .

**Summary**

Figure 4.3 summarises and illustrates the characteristic elements used for the basic geometric entities.

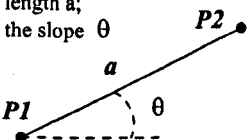
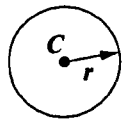

Line segment	Circle	Arc
<ul style="list-style-type: none"> <li>. starting point <math>P(x_{1a}, y_{1a})</math>;</li> <li>. length <math>a</math>;</li> <li>. the slope <math>\theta</math></li> </ul> 	<ul style="list-style-type: none"> <li>. center <math>C(Cx, Cy)</math>;</li> <li>. radius <math>r</math>.</li> </ul> 	<ul style="list-style-type: none"> <li>. radius <math>r</math>.</li> <li>. end points <math>P1</math> and <math>P2</math>;</li> </ul> 

Figure 4.3: Basic Geometric Entities and Their Characteristic Points

### 4.3 Geometric Constraint Representation

Constraints between geometric entities are represented in terms of a set of generic constraints which are specified between points and line segments. These generic constraints are represented as equations in the Equation Graph. The generic constraints are:

1. Distance  $d$  from point  $P_1(x_1, y_1)$  to point  $P_2(x_2, y_2)$

$$\|P_1 - P_2\| = d \Rightarrow (x_1 - x_2)^2 + (y_1 - y_2)^2 = d^2 \tag{4.8}$$

2. Point  $P_1(x_1, y_1)$  coincident with point  $P_2(x_2, y_2)$

$$x_1 = x_2 \tag{4.9}$$

$$y_1 = y_2 \tag{4.10}$$

Note that this property is a particular case of the distance between two points ( $d = 0$ ) above. However, this representation is chosen because it involves only linear equations and thus these constraints can be solved more efficiently due to the methods graphs attached to them (Section 3.2.1).

3. Distance  $d$  from point  $P_1(x_1, y_1)$  to line segment  $l$

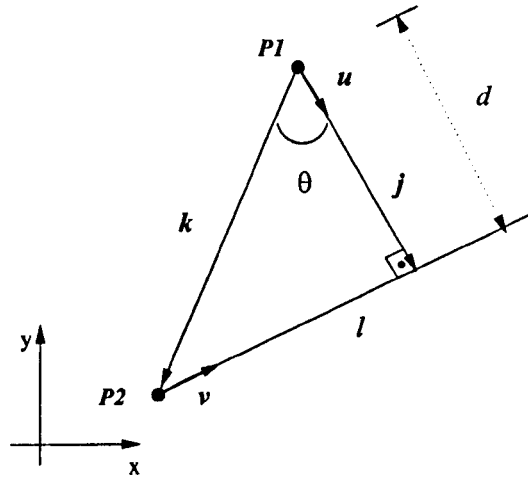


Figure 4.4: Distance from Point to Line Segment

Consider Figure 4.4 where  $P_2(x_2, y_2)$  is the starting point for the line segment  $l$ . In the figure, the unit vector  $\vec{v}$  is used to represent  $(\cos \theta, \sin \theta)$ , where

$$\vec{k} \cdot \vec{j} = \|\vec{k}\| \|\vec{j}\| \cos \theta \tag{4.11}$$

$$\cos \theta = \frac{d}{\|\vec{k}\|} \tag{4.12}$$

This implies that

$$\vec{k} \cdot \vec{j} = \|\vec{j}\| d \tag{4.13}$$

and hence

$$\vec{k} \cdot \frac{\vec{j}}{\|\vec{j}\|} = d \Rightarrow \vec{k} \cdot \vec{u} = d, \tag{4.14}$$

where  $\vec{u}$  is the unit vector of  $\vec{j}$  and  $\vec{u} \perp \vec{v}$ .

Thus, given the point  $P_1(x_1, y_1)$  and the starting point  $P_2(x_2, y_2)$  of line segment  $l$ , the distance  $d$  from  $P_1$  to  $l$  is given by:

$$(x_2 - x_1) \sin \theta - (y_2 - y_1) \cos \theta = d \tag{4.15}$$

4. Angle  $\alpha$  between a pair of line segments  $(l_1, l_2)$

$$\cos \theta_2 = \cos \theta_1 \cos \alpha - \sin \theta_1 \sin \alpha \tag{4.16}$$

$$\sin \theta_2 = \sin \theta_1 \cos \alpha + \cos \theta_1 \sin \alpha \tag{4.17}$$

The line segments  $l_1$  and  $l_2$  do not necessarily have to have a common end point. To demonstrate the use of these equations, consider Figure 4.5. In this figure, an angle  $\alpha$  is given between two vectors  $\vec{u}(u_x, u_y)$  and  $\vec{v}(v_x, v_y)$ . The length of both vectors is  $L$ .

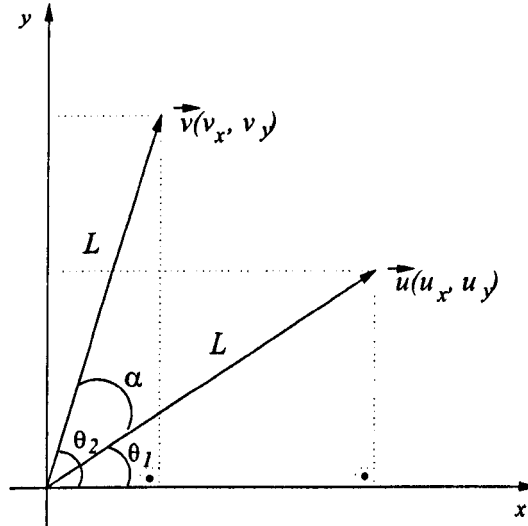


Figure 4.5: Angle  $\alpha$  between Line Segments

From the figure it can be seen that

$$\cos \theta_1 = \frac{u_x}{L} \Rightarrow u_x = L \cos \theta_1 \quad (4.18)$$

$$\sin \theta_1 = \frac{u_y}{L} \Rightarrow u_y = L \sin \theta_1 \quad (4.19)$$

Similarly,

$$\cos \theta_2 = \cos(\theta_1 + \alpha) = \frac{v_x}{L} \quad (4.20)$$

$$\sin \theta_2 = \sin(\theta_1 + \alpha) = \frac{v_y}{L} \quad (4.21)$$

Using the trigonometry formulae for  $\cos(\theta_1 + \alpha)$  in Equation 4.20 it follows that:

$$\frac{v_x}{L} = \cos \theta_1 \cos \alpha - \sin \theta_1 \sin \alpha \Rightarrow \quad (4.22)$$

$$v_x = L \cos \theta_1 \cos \alpha - L \sin \theta_1 \sin \alpha \quad (4.23)$$

From Equations 4.18 and 4.19, Equation 4.23 can be seen as:

$$v_x = u_x \cos \alpha - u_y \sin \alpha \tag{4.24}$$

Now, considering vectors  $\vec{u}$  and  $\vec{v}$  as the unit vectors of the underlying line segments to impose an angle  $\alpha$  and also that  $\vec{u}$  and  $\vec{v}$  can be respectively represented by  $(\cos \theta_1, \sin \theta_1)$  and  $(\cos \theta_2, \sin \theta_2)$ , Equation 4.24 can be written as Equation 4.16, i.e.

$$\cos \theta_2 = \cos \theta_1 \cos \alpha - \sin \theta_1 \sin \alpha \tag{4.25}$$

Similarly, Equation 4.21 can be used to derive Equation 4.17.

**Summary**

Figure 4.6 presents a summary of the generic constraints and corresponding equations for various geometric constraints. In the figure, the  $(\cos \theta, \sin \theta)$  components used in the mathematical representation of a line segment are represented by the unit vector  $U(U_x, U_y)$  for simplicity.

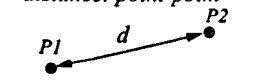

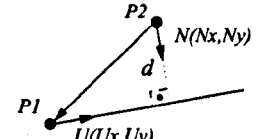
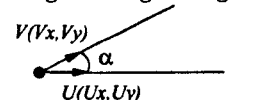
Generic Constraints	Equations	Geometric Constraints
 <p>distance: point-point</p>	$\  P1 - P2 \  = d$	tangent: circle-circle
 <p>coincident: point-point</p>	$x1 = x2$ $y1 = y2$	coinc.: lineSeg-lineSeg (end pts.) coinc: lineSeg-arc (end pts.) concentric: circle-circle
 <p>distance: point-lineSeg</p>	$(P1 - P2) \cdot N = d$	tangent: circle-lineSeg
 <p>angle: lineSeg-lineSeg</p>	$V_x = U_x \cos \alpha - U_y \sin \alpha$ $V_y = U_x \sin \alpha + U_y \cos \alpha$	parallel: lineSeg-lineSeg perpendicular: lineSeg-lineSeg

Figure 4.6: Geometric Constraints Supported by the Constraint Engine

### 4.4 Coupling of Design Constraints

As mentioned earlier, the coupling of geometric constraints and engineering constraints is very important to support preliminary design as well as to capture the engineer’s design intent more effectively [23]. This section shows how the Equation Graph maintains the set of engineering and geometric constraints and also presents how such coupling is used to support the concept of composite objects.

#### 4.4.1 Coupled Constraint Network

To illustrate how the Equation Graph maintains the coupling of engineering constraints and geometric constraints, consider Figure 4.7 which shows the design of the internal-combustion engine as an application of the slider-crank mechanism [64, 95]. Table 4.1 presents some of the constraints related to the design of this mechanism.

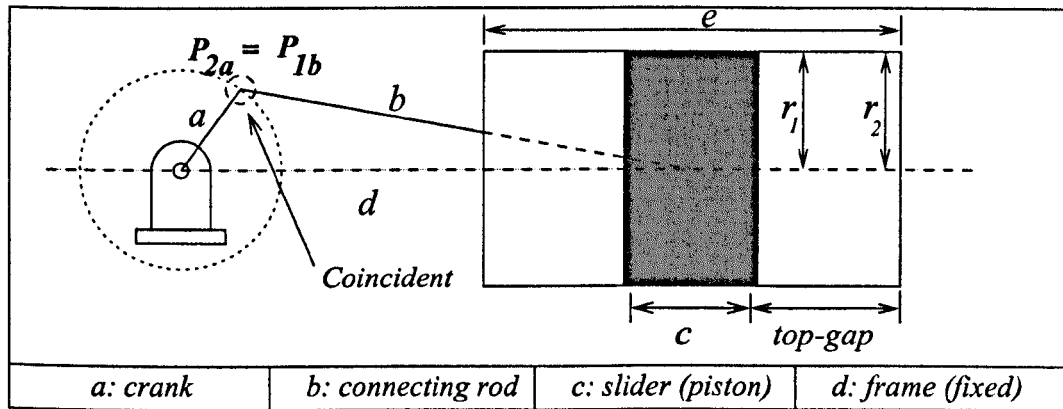


Figure 4.7: Internal-Combustion Engine (Slider-Crank Mechanism)

Constraints  $C1 - C3$  in Table 4.1 are engineering constraints where  $\alpha$  is a constant used in the computation of the engine power; *displacement* is the volume of mixed air and fuel consumed per engine cycle; *compression\_ratio* is the approximate ratio between the maximal and minimal pressures during the compression part of the cycle;  $a$  is the length of the rotary part of the crankshaft; *top\_gap* is the length of the minimal distance between the top of the cylinder and the piston top during the cycle;  $n$  is the number of engine cylinders and  $r$  is the radius of the engine’s cylinders.

Equations	Constraint Type
$C1 : power = \alpha \times displacement \times compression\_ratio$ $C2 : compression\_ratio = \frac{2 \times a + top\_gap}{top\_gap}$ $C3 : displacement = 2 \times a \times \pi \times r^2 \times n$	Engineering constraints
$C4 : x_{2a} = x_{1a} + a * \cos \theta_a$ $C5 : y_{2a} = y_{1a} + a * \sin \theta_a$ $C6 : x_{2b} = x_{1b} + b * \cos \theta_b$ $C7 : y_{2b} = y_{1b} + b * \sin \theta_b$ $C8 : x_{2a} = x_{1b}$ $C9 : y_{2a} = y_{1b}$	Geometric constraints

Table 4.1: Design Constraints for the Internal-Combustion Engine

Constraints  $C4 - C5$  and  $C6 - C7$  are used to create the line segments which represent the crank  $a$  and the connecting rod  $b$ , respectively. In addition, constraints  $C8$  and  $C9$  are created to satisfy the revolute joint (*coincident constraint*) between the crank and connecting rod at points  $P_{2a}$  and  $P_{1b}$ .

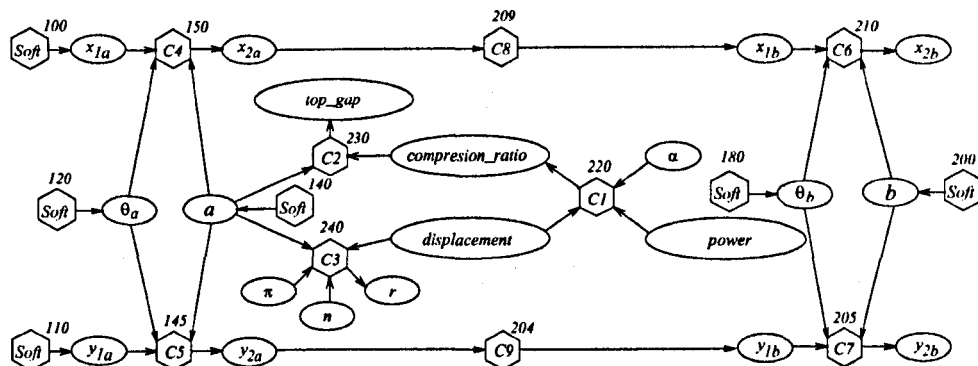


Figure 4.8: Coupling of Engineering Constraints and Geometric Constraints

Figure 4.8 shows an Equation Graph for these constraints. Note, for example, that the user can try different values for the length of the crank  $a$  or connecting rod  $b$ , by simply changing the value for the soft constraints 140 and 200, respectively. Thus, easy geometry updating derived from engineering constraints is achieved.

## 4.4.2 Composite Objects

The Equation Graph also allows the user to build composite objects by combining a set of basic geometric entities. All geometric and dimensional constraints related to the composite objects are then treated as a unique *constraint set*. For example, the piston *c* of the internal-combustion engine in Figure 4.7 is treated as a rectangular shape created from four line segments  $l_1$ ,  $l_2$ ,  $h_1$  and  $h_2$  as shown in Figure 4.9. Constraints such as *right angle* constraints ( $\square$ ) and *coincident end points* constraints ( $\bullet$ ) can be imposed among the line segments to define the composite objects as in Figure 4.9.

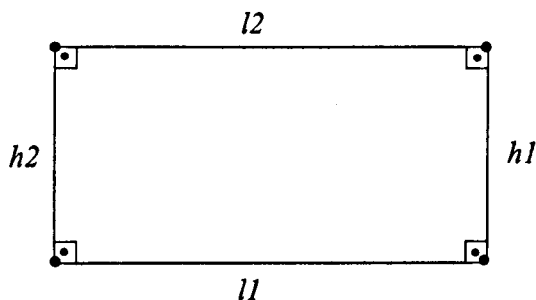


Figure 4.9: A Rectangular Shape as a Composite Object

Finally, the following engineering constraints can be introduced in any order to control the dimensional attributes (length and height) of the rectangular shape.

$$C1 : l_2 = l_1$$

$$C2 : h_2 = h_1$$

Figure 4.10 shows the constraint network derived from the definition of the rectangular shape. In this figure,  $y$  coordinates and numbers used to show the sequence of constraint satisfaction are omitted for simplicity.  $x$  coordinates are written in the form  $x_{i_i}$  or  $x_{i_{h_i}}$ ,  $i = 1, 2$ . The engineering constraints  $C_1$  and  $C_2$  are shown coupled with the geometric constraints. Constraints with the symbol  $=$  represent the equations derived from the *coincident end points* constraints. Constraints  $C_3 - C_6$  represent the constraints for the line segments end points as explained in Section 4.2.1. Soft constraints are imposed on the main length attributes to support parameterised models. Thus, the user can try different rectangular shapes by changing the values of  $l_1$  and  $h_1$ . In addition, the user can translate and rotate such rectangular shape as illustrated by the *soft* constraints on  $x_{1_{l_1}}$  and  $\theta_{1_1}$ , respectively.



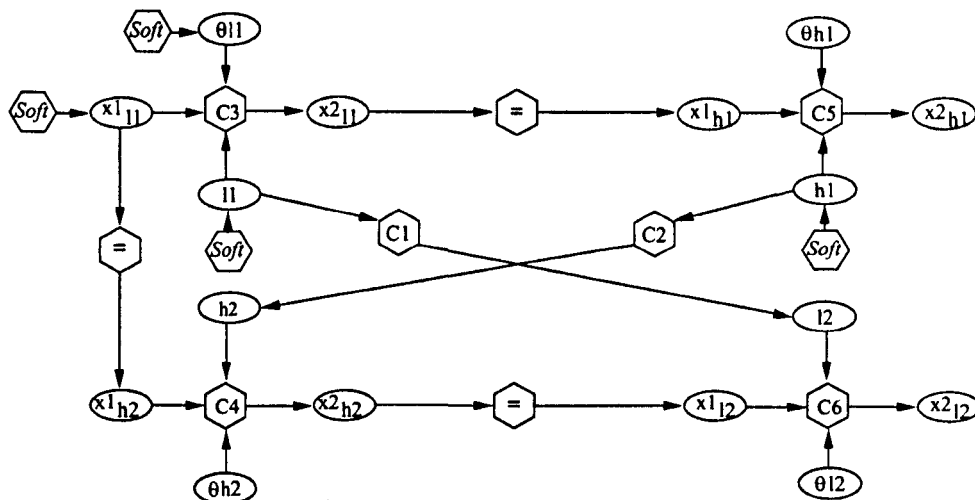


Figure 4.10: Partial Constraint Network for a Rectangular Shape

The construction of this rectangular shape illustrates that the system allows the user to create more complex objects based on the existing set of geometries and constraints. Furthermore, an excessive work is saved since the user does not need to worry about the complexity of the Equation Graph, such as the one in Figure 4.10.

#### 4.4.3 Inserting Geometric Constraints

When a geometric constraint is imposed between two geometric entities, one geometric entity is chosen to be the reference and the other is chosen as the target. Then the constraint dependency is established from the reference (parent) to the target object (child). The target object moves towards the reference to satisfy the imposed constraint according to its degrees of freedom (DOFs). When both objects have the same DOFs, the first one to be picked by the user is selected as the reference.

For example, consider the scenario in Figure 4.11(a), where the user wants to impose a coincident constraint between line segments. In this case the constraints ( $x_{1d} = x_{1c}$ ) and ( $y_{1d} = y_{1c}$ ) are established in the Equation Graph as in Figure 4.11(b) (the figure only shows the constraints between the  $x$  coordinates).

In order to insert this constraint, the algorithm searches the ancestor graph of the variable on which the constraint is imposed ( $x_{1c}$  in the example), looking for a free variable. Note that in this case the algorithm is already establishing the arc direction for the inserted

equation (constraint dependency from the  $x$  coordinate of the reference object to the  $x$  coordinate of the target object). Variables with soft constraints are treated as free variables and they are used to absorb the new constraint. In the above example, the variable  $x_{1c}$  is chosen to be the output variable to absorb the new constraint, as explained in Section 3.3.1. The soft constraint attached to the variable is then deleted as in Figure 4.11. The updated geometry and the constraint network after the insertion and satisfaction of the coincident constraint are shown in Figure 4.12(a) and (b).

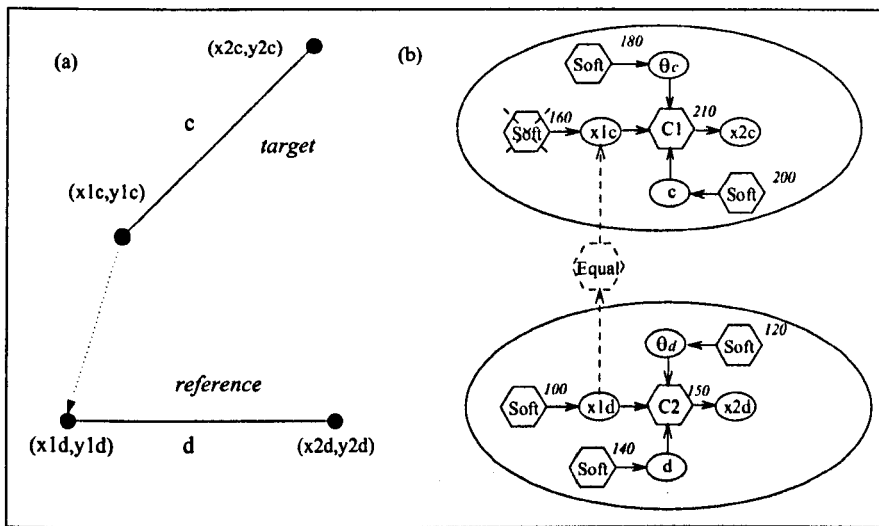


Figure 4.11: Inserting a Coincident Constraint

### 4.5 Direct Manipulations of Under-constrained Models

The ability to directly manipulate constrained models provides an easy way for the designer to explore under-constrained spaces [70]. Direct manipulation techniques have been recognised as an intuitive approach for many CAD systems [8, 11, 12, 20, 54, 73, 80, 96]. Direct manipulations of under-constrained geometries is supported by exploiting their respective degrees of freedom. This is achieved through both the soft constraints in the Equation Graph and a high-level abstraction representing the constraint sub-graphs corresponding to the geometric entities and composite objects. This representation is referred to as *Relationship Graph (RG)*.

### 4.5.1 Geometric Relationship Graph

In the Relationship Graph, nodes represent geometric entities and arcs represent geometric constraints. The nodes maintain information about the type of the geometric entities (lines, circles, etc) and point to their corresponding constraints and variables in the Equation Graph. This is used to pick geometric objects and to support direct manipulation as well as kinematic simulation of assemblies, efficiently. For example, Figure 4.12 shows the links between the nodes of the Relationship Graph and the Equation Graph. The figure shows that a *coincident* (Equal) constraint has been imposed between line segment *c* and line segment *d* as discussed in Section 4.3 (only *x* coordinates are shown for simplicity). The degrees of freedom for each geometric entity can be found by searching for its variables with soft constraints. The knowledge about the degrees of freedom of individual geometric entities is used to support the direct manipulation and the kinematic simulation of assemblies. This is explained in detail in the following section.

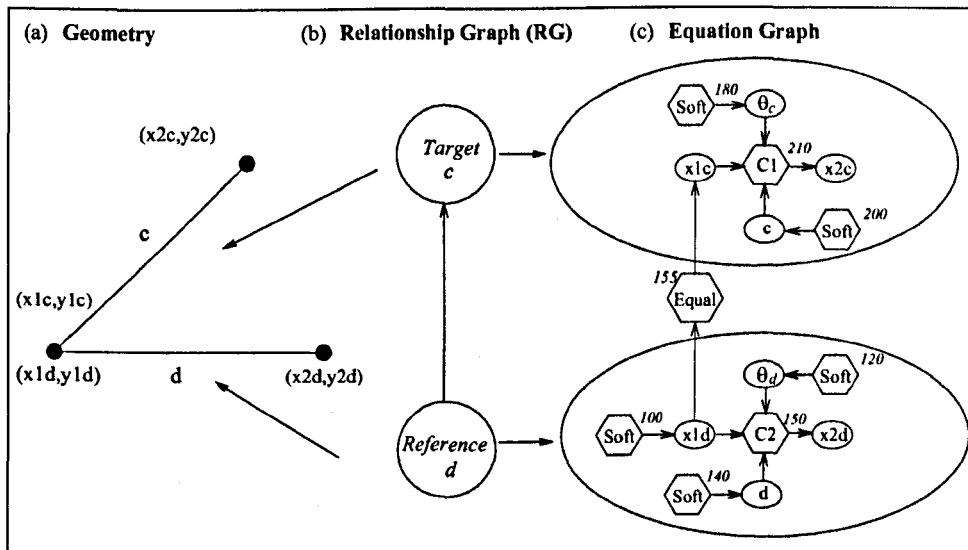


Figure 4.12: The Relationship Graph and the INCES's Underneath Constraint Network

A geometric object selected by the user (through the GUI) is associated with a node in the Relationship Graph. As mentioned above, this node maintains the information about the characteristic elements of the geometric object it represents. Thus, the degrees of freedom for each geometric entity can be determined by searching for the variables of its characteristic elements that have soft constraints imposed on them. The Relationship Graph can take the

form of a *directed acyclic graph* (DAG) or a *directed cyclic graph* (DCG). The identification of one of these graph types and the knowledge about the degrees of freedom of individual geometric entities is used to support direct manipulations as well as the kinematic simulation of mechanisms.

### 4.5.2 Direct Manipulations within a DAG

The Relationship Graph is said to be a DAG if it contains no cycles. A DAG is maintained in a consistent format to support direct manipulations as well as cycle detection. The format used for a DAG is shown on the right side of Figure 4.13 where it represents an open linkage of five line segments. The DAG takes the form of a tree, where the root node has no parent and all other nodes have only one parent. The arcs always go from the parent node to the child node. All geometric entities are considered as a root node when they are first created. They lose this root status when geometric constraints are imposed on them i.e, when they become a child node.

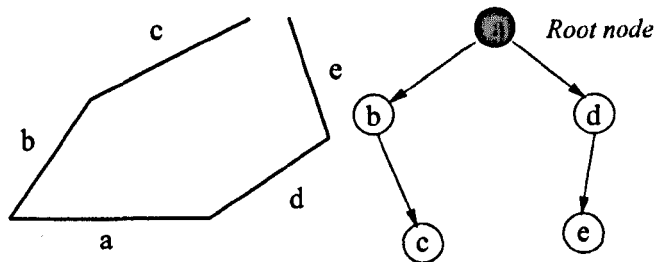


Figure 4.13: Standard Form of a Directed Acyclic Graph

Two operations are used to support direct manipulations: Translation and Rotation. The user is supposed to choose one of them when he selects a geometric entity. When the user manipulates a geometric entity, the DOF of the selected entity are checked to determine the validity of the manipulations. If the manipulations are allowed then the values of the DOF variables (soft constraints) of its characteristic elements are updated. Next, the updated values are propagated to the descendant nodes in the Equation Graph to locally update the geometry. If the DOFs of the geometric entity do not allow the user's manipulations, then the DOFs of its parent node are checked to see if it can support the manipulations.

For example, suppose the user wants to translate line segment *d* in Figure 4.12(a). This line has a translational DOF, according to its soft constraints in the Equation Graph in

Figure 4.12(c). The translation vector, determined by the GUI, is applied to the  $x_{1d}$  and  $y_{1d}$  coordinates of line  $d$ . Then, the values of the descendant nodes are updated through constraint propagation. The effect of these values are also propagated to line  $c$ . Thus, both lines move together. If the user wants to rotate line  $d$ , the GUI determines the angle of rotation and thus the line is rotated about the point  $(x_{1d}, y_{1d})$ .

Now, suppose the user wants to translate line  $c$ . In this case, it is identified that such an operation is not allowed since line  $c$  has no translational DOF. In this case, the parent node of line  $c$  (line  $d$ ) in the Relationship Graph is checked to absorb this translation. Since line  $d$  has a translational DOF, the operation is performed on this line as explained above.

### 4.5.3 Direct Manipulations within a DCG

The form of a direct cyclic graph (DCG) is as shown in Figure 4.14. In this form, there is one root node and one last node. The direction of arcs always starts from the root node and ends at the last node of the cycle.

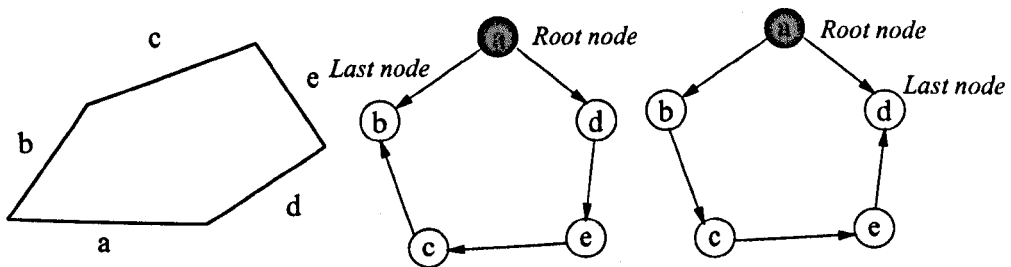


Figure 4.14: Standard Form of a Directed Cyclic Graph

A geometric constraint cycle is detected by maintaining a dependency hierarchy list for each node in the Relationship Graph. When a new constraint is imposed between two geometric entities, an intersection is performed between their dependency hierarchy list to check for any cycle. If this intersection is not empty, then the geometric nodes have at least a common parent and therefore a cycle is detected.

Consider, for example, the four-bar mechanism of Figure 4.15(a). In this figure, the designer wants to impose a revolute joint (*coincident end-pts* constraint) between bar  $b$  and bar  $c$  (dashed bars). Figure 4.15(b) shows the dependency hierarchy for each node before the constraint is imposed. After the constraint is satisfied, the intersection of bar  $b$ 's and bar  $c$ 's dependency lists is bar  $d$ . Therefore, a cycle is formed and bar  $d$  is considered to be

the root node (Figure 4.15(c)). After a cycle is detected, the constraint dependency in the Relationship Graph is always re-written to maintain the consistent format as shown in Figure 4.14.

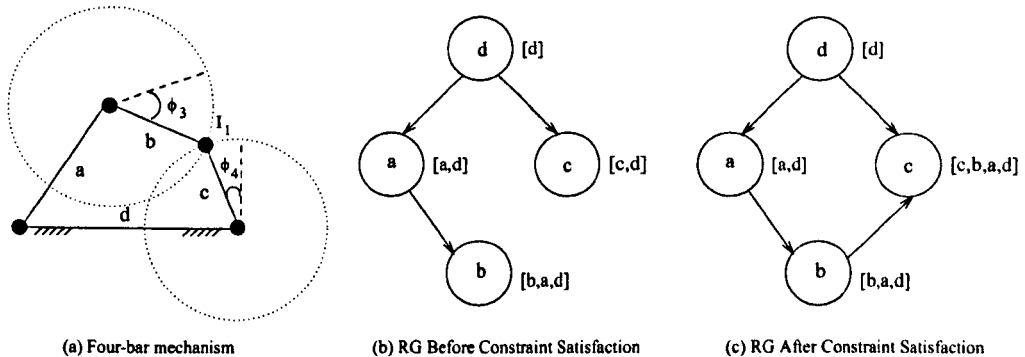


Figure 4.15: Geometric Cycle Detection

To support direct manipulations within a DCG, a set of algorithms developed by Tsai and Fernando [110, 111] is used. These algorithms are based on the Degrees of Freedom Analysis technique as proposed by Kramer [60]. The main advantage of these algorithms is that they handle under-constrained geometric cycles incrementally, without resorting to numerical methods.

## 4.6 Summary

This chapter discussed how the incremental algorithms used to satisfy engineering constraints are extended to support geometric constraints. Such extension allows the **coupling** of engineering and geometric constraints in the Equation Graph. The concept of *soft* constraints was also introduced. *Soft* constraints are used to identify the degrees of freedom of geometric entities. They are also used to satisfy geometric constraints by using local propagation techniques. Finally, a high level graph representation called a *Relationship Graph* was also presented. This representation supports the direct manipulations of under-constrained geometric models, according to the DOF's determined by their soft constraints.

The main contribution of these techniques is that incremental constraint satisfaction and the handling of under-constrained geometry is supported by the coupling of design constraints associated with the concept of *soft* constraints.

## Chapter 5

# Implementation

---

### 5.1 Introduction

This chapter describes the implementation of a prototype constraint-based engine to demonstrate the feasibility of the techniques and algorithms proposed in Chapter 3 and Chapter 4. The prototype system has been implemented in C++ on a SGI XS24/4000 Indigo. There are two main components involved in this implementation: the Graphical User Interface (GUI) and the Constraint Manager.

The rest of this chapter is organised as follows. Section 5.2 presents an overview of the prototype constraint engine. Section 5.3 details the implementation of the Constraint Manager. Section 5.4 briefly describes the implementation of the GUI. Section 5.5 shows the information flow between the GUI and the Constraint Manager through several design activities supported by the constraint engine. Finally, a summary of this chapter is given in Section 5.6.

### 5.2 Overview of the Constraint Engine

As shown in Figure 5.1, the constraint engine has two main modules: the Graphical User Interface (GUI) and the Constraint Manager. The Constraint Manager is composed of four sub-modules: the Equation Graph, the Incremental Equation Solver (INCES), the

Geometric Relationship Graph and the Geometric Cycle Solver. The GUI of the constraint engine provides the user with a set of geometric entities (i.e. lines, circles, arcs) and the ability to specify constraints between geometric entities through menus. These facilities enable the user to sketch the geometry of the design by picking geometric entities and applying constraints between them. Engineering constraints are inserted through menu interaction.

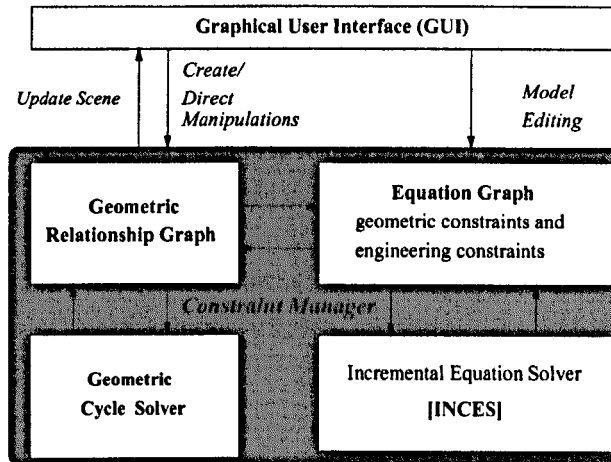


Figure 5.1: Constraint Engine Architecture

The geometric and engineering constraints specified by the user are translated into equations and maintained in the **Equation Graph**. The degrees of freedom of the under-constrained geometric entities are represented in the graph using *soft constraints* imposed on the corresponding variables. These soft constraints are maintained in the graph to support the direct manipulation of under-constrained models. The Incremental Equation Solver (**INCES**) solves the Equation Graph using efficient local propagation algorithms when linear or non-linear constraints are inserted. Constraint cycles are identified and solved locally.

The **Geometric Relationship Graph** maintains a high-level representation of the geometric entities and the constraints between them. In this graph, each node maintains information about the type of geometric entity and arcs maintain the constraint types (i.e. tangent, parallel etc). Each node in the Geometric Relationship Graph maintains pointers to its corresponding nodes in the Equation Graph. The purpose of this Geometric Relationship Graph is to provide a high-level abstraction for separating the geometric constraints from the engineering constraints, within the Equation Graph, to support efficient direct manipulation of geometric entities. This graph can also be used to identify geometric con-



straint cycles and apply efficient algorithms to solve them through the **Geometric Cycle Solver**.

### 5.3 The Constraint Manager

The Constraint Manager is composed by three main components as shown in Figure 5.1. They are the **Equation Graph**, the Incremental Equation Solver (**INCES**) and the **Relationship Graph**. This section shows how each of these components are implemented.

#### 5.3.1 Implementation of the Equation Graph

The Equation Graph supports the coupling of geometric constraints and engineering constraints. It is a bipartite graph that maintains the information about the solution of the current constraint set. The two distinct types of nodes in this bipartite graph are nodes representing the variables and nodes representing the constraint equations. Each constraint node maintains a list of its associated variables. In turn, each variable node maintains a list of constraints that it is input and output for.

#### *Representation of Constraint Equations*

The constraint equation node is implemented as a class in C++. Figure 5.2 shows the data structure used to define the main data members of the constraint node. In the figure, the **TYPE** field indicates the constraint type, either a linear or a nonlinear equation. The **POSITION** field identifies the constraint position in the sequence of constraint satisfaction (i.e. its number in the Equation Graph). The **OUTPUT** field maintains the current output variable of the constraint. Finally, **VAR\_LIST** maintains a pointer pointing to the list of variables belonging to the constraint equation.

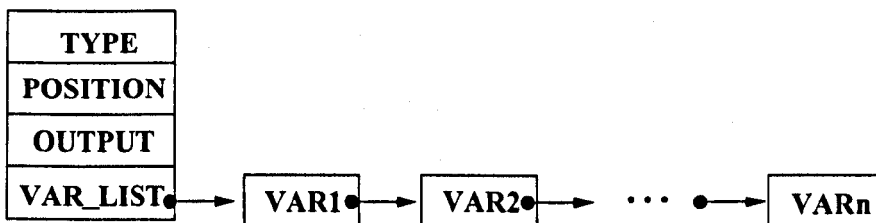


Figure 5.2: Data Structure for a Constraint Equation Node

### Representation of Variables

Variables are also implemented as a C++ class. Figure 5.3 shows the data structure used to define the main data members of the variable node. The **NAME** field is a string of characters that give the name of the variable. The variable name can either be created by the user or derived automatically as in the case of those representing the coordinates of geometric entities. The **VALUE** field holds the current value of the variable. The rest of the structure is composed by three main pointers.

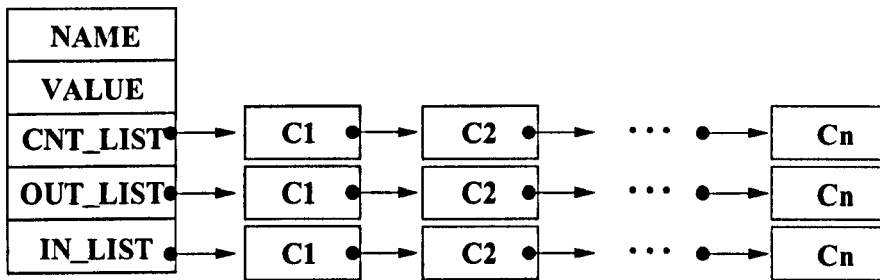


Figure 5.3: Data Structure for a Variable Node

**CNT\_LIST** points to a list of constraint equation nodes that the variable belongs to. This information is used to identify if the variable is a free variable during constraint insertion. **OUT\_LIST** points to a list of *out* constraint nodes, i.e. the set of constraints that the variable is an input for. Similarly, **IN\_LIST** points to a list of *in* constraint nodes, i.e. the set of constraints that the variable solves for (output variable). In fact, this list should always have only one element. If the system identifies that the number of elements in this list is greater than one, it triggers a flag detecting a constraint conflict. This is because the algorithms of the constraint engine are based on the fact that each variable will be the output to at most one constraint.

#### 5.3.2 Constructing the Equation Graph

The Equation Graph is implemented as a linked list of constraint nodes. This linked list is dynamically constructed through the INCES algorithm presented in Section 3.3. This allows the solution represented by the Equation Graph to be incrementally updated. This section illustrates how this incremental updating is implemented using an example.

Suppose that the user wants to use the constraint engine to satisfy the following set of

equations:

$$C1 : a = b + c$$

$$C2 : c = d$$

$$C3 : b = e - f$$

The user could start inserting constraint  $C1$  as shown in Figure 5.4(a). The engine then creates the node of the linked list as represented by the shaded rectangular box with rounded corners. Each node of the linked list representing the Equation Graph is a structure containing two main fields: a pointer to a constraint equation and a pointer to the next element in the list. Next, INCES seeks for free variables in the inserted constraint.

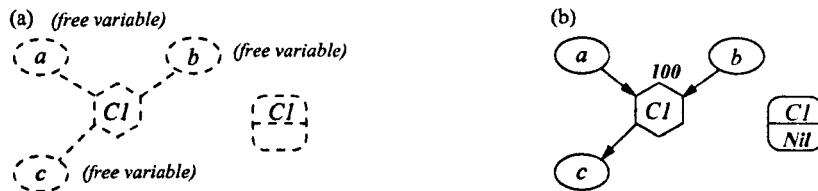


Figure 5.4: Dynamic Updating of the Equation Graph - Constraint  $C1$

Since  $C1$  is the first constraint to be inserted, all its variables are seen as free variables because they belong to only one equation ( $C1$ ). The algorithm chooses any of these variables to be the output variable,  $c$  for example, and a number (e.g. 100) is attached to constraint  $C1$  (Figure 5.4(b)). Since this is the first node of the linked list the field that maintains the pointer for the next constraint in the satisfaction sequence is *nil*.

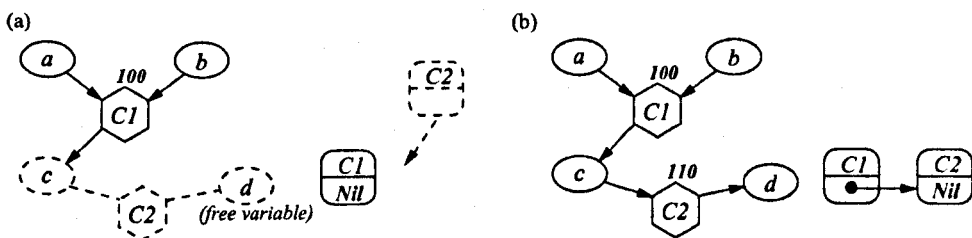


Figure 5.5: Dynamic Updating of the Equation Graph - Constraint  $C2$

Next, constraint  $C2$  is inserted and the algorithm again starts searching for free variables in this constraint. At this moment, the list pointed by the `CNT_LIST` pointer of variable  $c$

contains two elements (constraints  $C1$  and  $C2$ ) whereas the same list of variable  $d$  contains only one element (constraint  $C2$ ), as shown in Figure 5.5(a). Therefore, variable  $d$  is a free variable. Thus, variable  $d$  is chosen as the output for constraint  $C2$  and the constraint dependency is updated.

The next step is to insert constraint  $C2$  in the linked list representing the constraint satisfaction plan. This new constraint must be satisfied after the constraints which calculate the value of its input variables. Therefore,  $C2$  must be inserted after  $C1$  because variable  $c$  is the input variable for  $C2$  and  $c$  belongs to  $C1$ . Thus, the algorithm chooses a number greater than that of constraint  $C1$  (100) to be attached to  $C2$  (110) and now the node representing constraint  $C1$  in the linked list points to  $C2$  (Figure 5.5(b)).

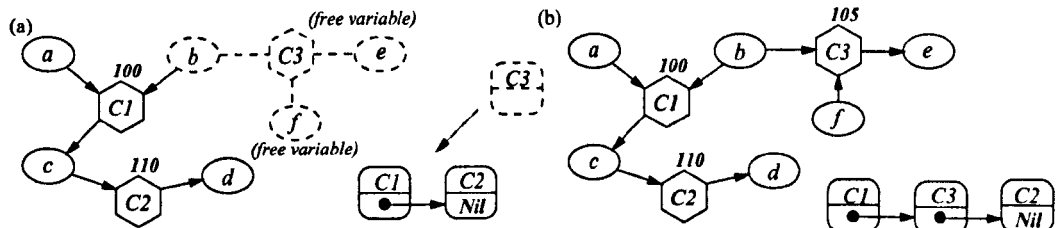


Figure 5.6: Dynamic Updating of the Equation Graph - Constraint  $C3$

Then, constraint  $C3$  is inserted and the algorithm identifies  $e$  and  $f$  to be free variables for this constraint (Figure 5.6(a)). Variable  $e$  is selected to be the output variable and the constraint dependency is again locally updated. Similarly, constraint  $C3$  must be inserted after constraint  $C1$ . Since constraint  $C2$  was previously inserted after constraint  $C1$ , the number to be attached to constraint  $C3$  (105) must be between the numbers attached to constraint  $C1$  (100) and constraint  $C2$  (110), respectively. When an integer number cannot be found between the number attached to the extreme constraints, the new allocated number can be calculated through the arithmetic median between the extreme numbers. The pointers in the linked list are also updated as shown in Figure 5.6(b)).

Finally, the user wants to fix the value of variable  $d$  through constraint  $C4$ . Since there is no free variable in this case, the algorithm visits the ancestor subgraph and finds variable  $a$  to be a free variable (as shown by the dashed path in Figure 5.7(a)). The algorithm sets  $a$  as the output for  $C1$  and starts changing the dependency. Next, variable  $c$  is chosen to be output for constraint  $C2$ . Thus, constraint  $C2$  must be inserted before  $C1$  in the linked list. To achieve this, the algorithm destroys  $C2$  from its current position and updates  $C3$ 's

pointer to *nil*. Then, constraint *C2* is inserted before *C1* and its number (90) is updated to be less than *C1*'s number. Similarly, constraint *C4* is inserted before *C2*, receiving 80 as its position number. Figure 5.7(b) shows the updated Equation Graph and its respective linked list.

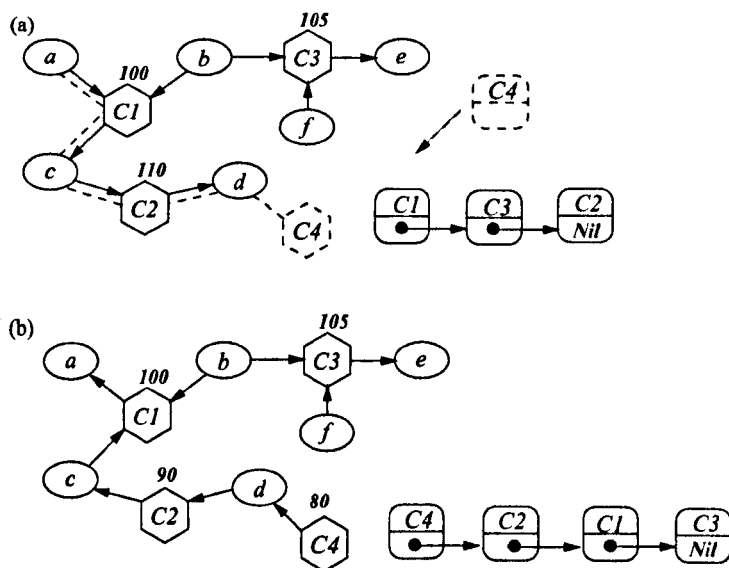


Figure 5.7: Dynamic Updating of the Equation Graph - Constraint *C4*

### 5.3.3 Implementation of the Geometric Relationship Graph

The Relationship Graph (RG) is a directed graph composed by a unique set of nodes representing the geometric entities and a set of arcs representing geometric constraints. Since the RG has only one type of nodes, it cannot be classified as a bipartite graph as the Equation Graph. The direction of the arcs denotes the direction in which geometric changes are propagated in the design model. The Relationship Graph is implemented as a linked list of geometric constraints.

#### *Representation of Geometric Constraints*

Geometric constraints are implemented as a C++ class. Figure 5.8 shows the data structure used to define the main data members of the geometric constraint class.

In the figure, the **TYPE** field indicates the constraint type, such as *distance*, *tangency*, *angle*,

etc. A geometric constraint is usually a binary relationship between two geometric entities. As explained in Section 4.4.3, these geometric entities are referred to as reference and target geometric entities, according to their DOFs. This information is respectively stored as the fields *RefGO* and *TarGO* as shown in Figure 5.8.

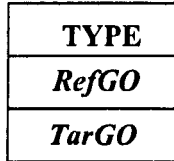


Figure 5.8: Data Structure for a Geometric Constraint Edge

**Representation of Geometric Entity Nodes**

Nodes in the Relationship Graph are also implemented as C++ classes. Figure 5.9 shows the data structure used to define the main data members of a node in the Relationship Graph.

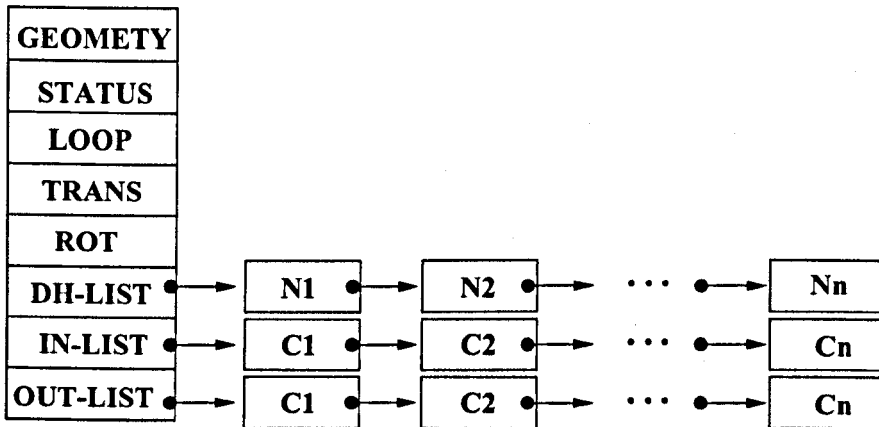


Figure 5.9: Data Structure for a Node in the Relationship Graph

The **GEOMETRY** field maintains a pointer to the geometric entity created by the Constraint Manager. This geometric entity is also implemented as a class in C++. In turn, this geometric entity class maintains the pointers to the variables of the characteristic elements that define it. For example, consider that the **GEOMETRY** field maintains a pointer to the class ‘Line-Segment’. This class will in turn contain, as its data members, pointers to the variables corresponding to the coordinates of the initial point, the length and the angle of the line

segment. Since these variables are represented in the Equation Graph, the link between the Relationship Graph and Equation Graph is established.

The **STATUS** field in the structure indicates whether the geometric entity is under-constrained, fully constrained or over-constrained. The **LOOP** field maintains a flag indicating whether the geometric entity belongs to a loop or not. This information is used to support direct manipulations within geometric loops as explained in Section 4.5.3. The fields **TRANS** and **ROT** indicate the current allowable translation and allowable rotation of the geometric entity, respectively. The **DH-LIST** field maintains a pointer to a list of ancestor nodes of the geometric entity. This information is used to detect a cycle of geometric constraints (Section 4.5.3). The **IN-LIST** field maintains a pointer pointing to a list of *in* constraints which are imposed on the geometric entity. Finally, the **OUT-LIST** field maintains a pointer pointing to a list of the *out* constraints which are the dependent geometric entities.

## 5.4 The Graphical User Interface

The GUI is based on Iris Inventor which is an object-oriented graphics toolkit [104]. The GUI handles the user's interaction events and provides an interface to the Constraint Manager. This is achieved through callback mechanisms provided by Iris Inventor.

### 5.4.1 Iris-Inventor Toolkit

Iris Inventor toolkit provides an object-oriented framework for describing scenes containing geometric objects and operations on them. It provides high level support for writing graphical applications that may make use of 2D or 3D interaction. This provides the ability to directly manipulate geometric objects. The toolkit library consists of three main components: *scene database*, *interaction* and *node kits*.

The *scene database* maintains a graphical representation of geometric objects in a graph. This graph maintains information about geometric objects such as their shapes, material and position as well as how the scene is to be viewed (lights, cameras, etc). Iris Inventor provides different types of nodes such as *shape nodes*, *property nodes* and *group nodes* to construct the scene database. Figure 5.10 shows a graph for a scene database representing the four line segments of the 4-bar mechanism in Figure 4.15(a).

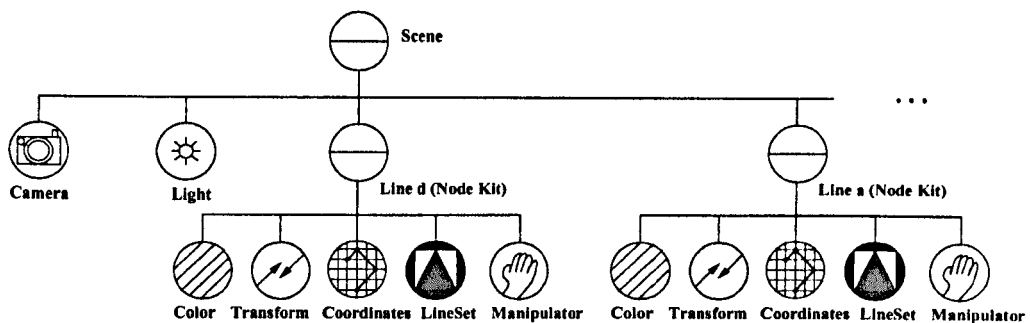


Figure 5.10: A Graphical Scene Data Base

First, a group node **Scene** is created to contain the scene and a **Camera** and a **Light** nodes are added to it so that the scene is visible. Then, each line segment is created through a node kit and added to the scene (Figure 5.10 shows only two of the created lines). *Node kits* facilitate the creation of structured, consistent graphical databases. Each node kit is a collection of database nodes with a specified arrangement. They are useful to create high-level objects that provide application-specific behaviour. Essentially, it combines some scene database subgraphs with attachment rules and other policies into a single class. A template associated with the node kit determines which nodes can be added when necessary and where they should be placed. For example, the **LineKit** of Figure 5.10 represents a line segment object. Its template allows room for **Colour**, Geometric Transformation (**Transform**) before the **LineSet** node in the node kit. This node is then followed by an interaction node called **Manipulator**.

The *interaction* section of Inventor introduces nodes that process events. An example of such node is the *selection* node which provides an easy way for applications to maintain a list of selected objects. The *manipulator* node is another type which responds to interaction events. Such nodes provides an easy way to allow the user to pick and drag a geometric entity. For example, when the **Line d** node receives an event from user's manipulations it passes the event to each child. The first three children rejects the event since they are property nodes. The event is then passed to the **LineSet** node. Again, this node rejects the event because it is a shape node. Finally, when the event is passed to the **manipulation** node it checks the event type and reacts to the event in some way such as moving the object. The **manipulation** node then returns to its parent node that the event was handled to update the geometry.

As mentioned earlier, the Graphical User Interface is built based on the facilities provided



by Iris Inventor. The Graphical User Interface essentially acts as a high-level application interface between the Iris Inventor toolkit and the Constraint Manager. Figure 5.11 shows a snapshot of the Graphical User Interface. The top line of the window shows the “pull-down” menu created through the library provided by Iris Inventor.

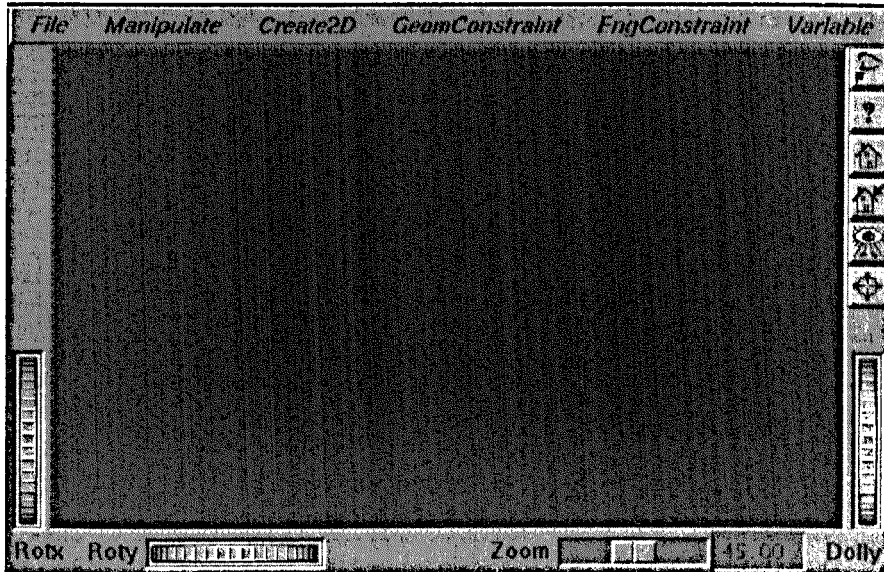


Figure 5.11: A Snapshot of the Graphical User Interface

Each option in this menu is used as follows.

- File: designed to allow the user to open and save a file with the information about its design model. These functions are not currently implemented. Another option allows the user to quit the system.
- Manipulate: allows the user either to translate or rotate a geometric entity.
- Create2D: allows the user to create geometric entities such as line segments and circles.
- GeomConstraint: presents the user with the set of geometric constraints. Current geometric constraints supported include *coincident end-pts*, circle-circle tangent and *angle* constraint.
- EngConstraint: allows the user to create engineering constraints. Simple linear equations such as  $x = y + z$  and  $x = y - z$  are supported. The system does not provide a general equation parser and therefore nonlinear constraints cannot be inserted through

the GUI. Thus, the GUI does not support the full functionality of the system in its current version.

- *Variable*: allows the user to create or destroy a variable in the Equation Graph.

## 5.5 Interface Between the Constraint Manager and the GUI

The purpose of this section is to describe the information flow between Iris Inventor and the Constraint Manager through the GUI. The section explains the steps carried out when the user performs operations to create geometric entities, variables, engineering and geometric constraints. The information flow during direct manipulation of geometric entities is also discussed.

### 5.5.1 Creating a Geometric Entity

Figure 5.12 shows the interface between Inventor and the Constraint Manager during the creation of a line segment  $d$ . The actions carried out by the user and the information flow between Inventor and the Constraint Manager are as follows.

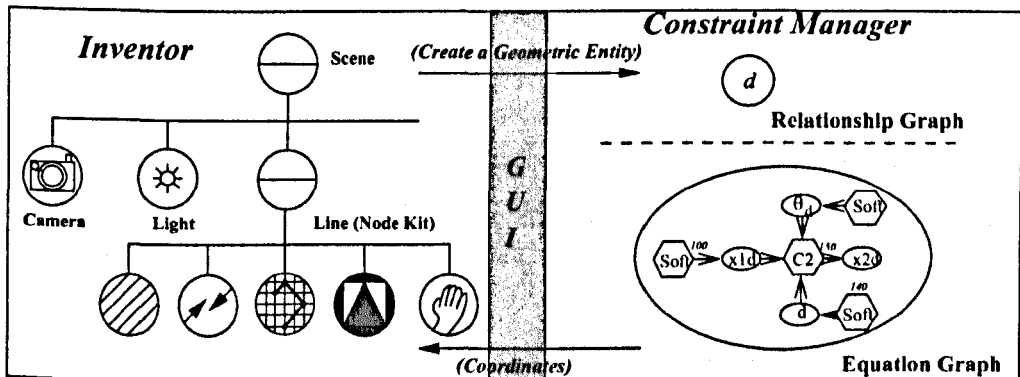


Figure 5.12: Implementation Steps to Create Geometric Entities

- The user selects to create a line segment from the *Create2D* option.
- Inventor creates a node kit for a line segment.
- Inventor sends a request to the Constraint Manager to create a line segment.

- The Constraint Manager constructs a RG node and an Equation subgraph to represent the node.
- The Constraint Manager passes the coordinates of the line segment to Inventor for displaying purposes.
- Inventor displays the line segment.

### 5.5.2 Creating Variables and Engineering Constraints

Some variables in the Equation Graph are automatically created during the creation of geometric entities. However, some of the variables must be created by the user. For example, variables  $k_1$ , and  $k_2$  must be created for the engineering constraints  $C4 : k1 = d/c$  and  $C5 : k2 = d/a$ , as shown in Figure 5.13. The information flow to create additional variables and engineering constraints are respectively as follows (Figure 5.14).

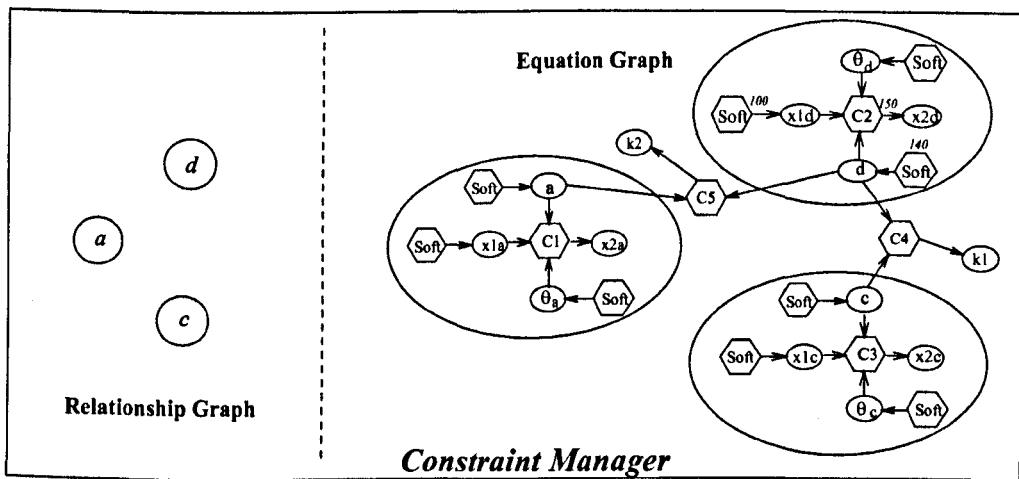


Figure 5.13: Implementation Steps to Create Variables and Engineering Constraints

- The user selects to create a variable from the *Variable* menu option.
- Inventor sends a request to the Constraint Manager to create an additional variable.
- The Constraint Manager creates a node in the Equation Graph to represent the variable. No information is sent back to Inventor since variables are not graphically represented in the current implementation.

- The user selects to create an engineering constraint from the *EngConstraint* menu option.
- Inventor sends a request to the Constraint Manager to create an engineering constraint.
- The Constraint Manager creates the engineering constraint as a node in the Equation Graph (for example, constraint *C5* in Figure 5.13). The INCES algorithm is then invoked to locally update the constraint dependency for the inserted constraint. Then, the **POSITION** and **OUTPUT** fields for the constraint equation node are updated. In addition, the **CNT\_LIST**, **OUT\_LIST** and **IN\_LIST** fields for each variable node of the constraint is updated.
- After updating the constraint dependency, the descendant nodes are locally satisfied through constraint propagation. The geometric information affected by this constraint propagation process is then sent to Inventor to be updated on the screen.

### 5.5.3 Imposing Geometric Constraints

This section explains how a *coincident end-pts* geometric constraint is imposed between two line segments. The steps carried out are as follows.

- The user selects the *GeomConstraint* menu option to impose a geometric constraint and then picks the two line segments he wants to impose the constraint on.
- Inventor requests the Constraint Manager to impose the requested geometric constraints. It also informs the Constraint Manager of the two selected geometric entities.
- The Constraint Manager creates the equations derived from the *coincident end-pts* constraint and inserts them in the Equation Graph. The constraint dependency is locally updated. The inserted constraints are then satisfied to update the line segments' coordinates. The constraint dependency in the Relationship Graph is also updated. First, the fields *RefGO* and *TarGO* of the created geometric constraint are updated. This geometric constraint is then seen as an edge in the Relationship Graph by updating the **OUT-LIST** and **IN-LIST** fields of the *RefGO* and *TarGO* nodes respectively (Figure 5.9). Finally, the Constraint Manager passes the new coordinates of the line segments to Inventor for display purposes.

- Inventor displays the line segments in their new position.

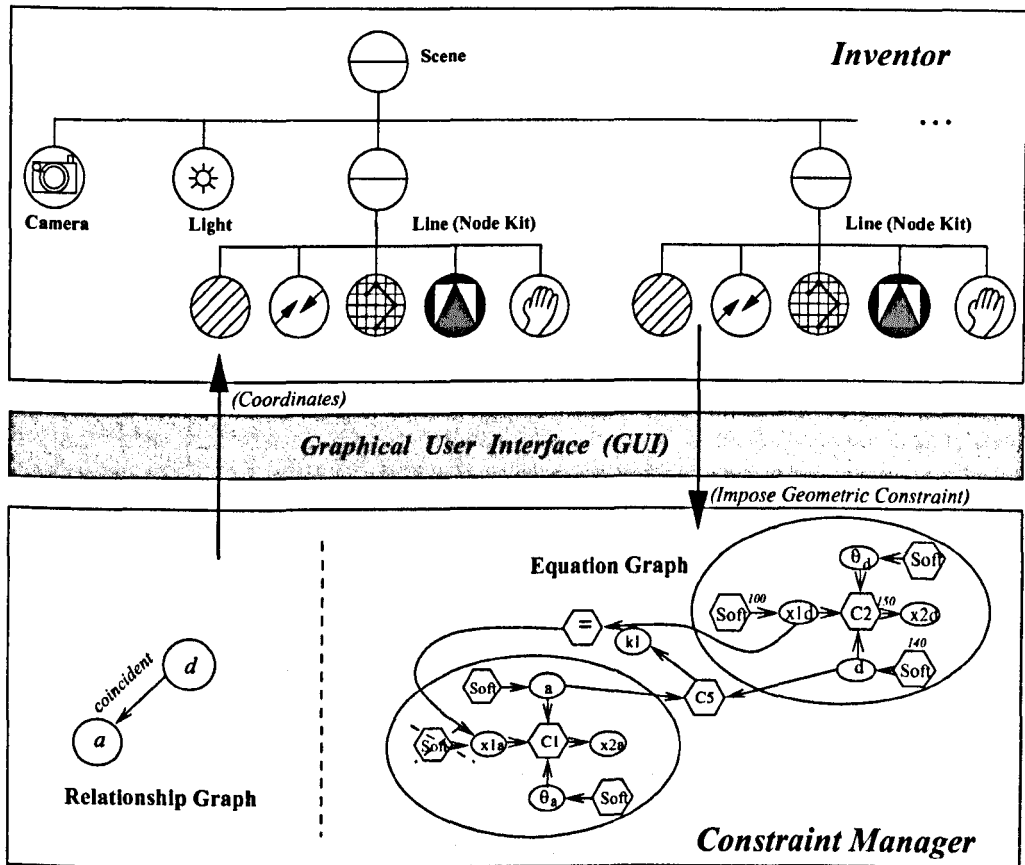


Figure 5.14: Implementation Steps to Impose Geometric Constraints

### 5.5.4 Direct Manipulation and Degrees of Freedom Inference

When a geometric entity is created, it is set to be free to translate and to rotate. This is identified by the *soft* constraints imposed on the characteristic elements of the geometric entity. When geometric constraints are imposed on the geometric entity, its TRANS and ROT fields are updated according to which *soft* constraints are consumed to satisfy the imposed geometric constraint. The information flow between Iris Inventor and the Constraint Manager when the user is manipulating a geometric entity is as follows.

- The user selects a translate or rotate operation from the *Manipulate* menu and picks a geometric entity.

- Inventor sends the picked object and the transformation matrix to the Constraint Manager, requesting the object to be directly manipulated.
- The Constraint Manager updates the *soft* constraints on the characteristic elements of the picked geometric entity according to the transformation matrix. For example, if a line segment is picked to be translated, the coordinates of the initial point are updated. Similarly, for a rotation the value of the angle  $\theta$  is updated. The other coordinates of the geometric entity are updated through constraint propagation as explained in Section 4.2. The Constraint Manager then passes the new coordinates of the geometric entity to Inventor.
- Inventor then displays the geometric entity in its new position.

## 5.6 Summary

The constraint engine allows the representation and satisfaction of coupled engineering and geometric constraints through the integration of the Equation-based approach and Geometric Constructive approach.

This chapter has described the implementation of the constraint engine proposed in this thesis. The engine is composed by two main components: the Graphical User Interface (GUI) and the Constraint Manager. The GUI is based on Iris Inventor which is an Object-Oriented Graphics Toolkit. The Constraint Manager has been built through graph-based techniques to maintain the geometric information necessary for Iris Inventor to update the design geometry. The chapter has also described the callback mechanisms used for establishing the interaction between these two main components. In addition, this chapter has discussed the implementation of the Equation Graph and its dynamic updating as well as the implementation of the Relationship Graph.

The next chapter presents the results and discusses advantages and limitations of the prototype constraint engine.

## Chapter 6

# Results, Discussions and Improvements

---

### 6.1 Introduction

The purpose of this chapter is to demonstrate the main features of the interactive constraint-based design engine discussed in the previous chapters. The internal-combustion engine has been used as a case study to demonstrate these features. Finally, the limitations and future improvements are discussed in this chapter.

### 6.2 Main Features of the Constraint Engine

The main features of the constraint engine discussed in this chapter include

- **The incremental nature:** this is evaluated according to the percentage of the constraint equation nodes in the Equation Graph that is visited due to a constraint insertion. A constraint node is said to be **visited** when it is first encountered by the INCES algorithm during the search for free variables in the Equation Graph.
- **The handling of under-constrained geometry:** this evaluates how a newly inserted geometric constraint can be satisfied without the need to fully specify the entire constraint set.

### 6.2.1 Case Study

The problem of designing the internal-combustion engine as an application of the slider-crank mechanism is used as a case study to illustrate the properties of the constraint engine. Figure 4.7 is replicated here as Figure 6.1 for the convenience of the reader. A rectangular shape (composite object) is used to represent the piston  $c$  and its casing box  $e$  as shown in Figure 4.10.

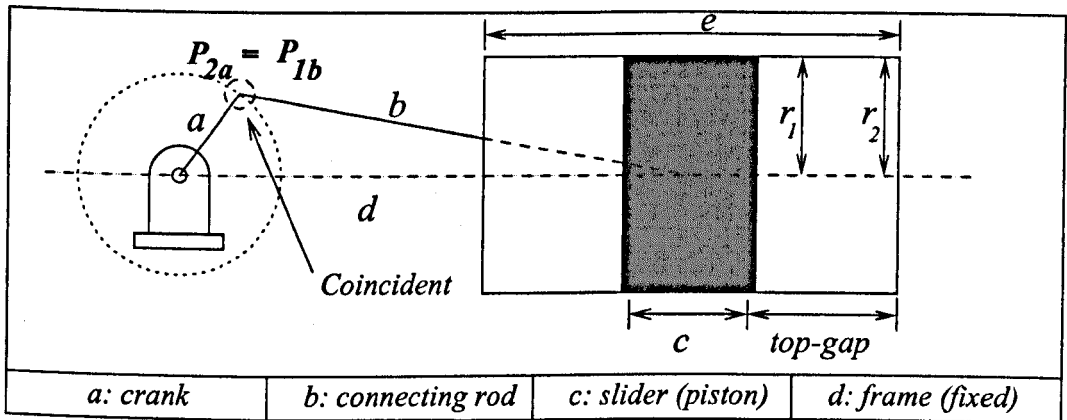


Figure 6.1: Copy of Figure 4.7

### 6.2.2 Incremental Nature of the Constraint Engine

The purpose of this section is to show the efficiency of the constraint engine in solving design constraints. This efficiency is measured by checking how much of the Equation Graph is affected when design constraints are inserted. The time taken for inserting a constraint is also presented in this section. Table 6.1 shows an approximate timing and percentage of the Equation Graph visited due to the insertion of the engineering and geometric constraints during the design of the internal-combustion engine. Each constraint was incrementally inserted in the Equation Graph in the order presented in Table 6.1. These constraints were inserted after the creation of the frame  $d$ , the piston  $c$  and its casing box  $e$ .

Constraints  $C1 - C7$  presented a free variable when they were first inserted. This is reflected in the decrease in percentage of nodes visited and also in the same time for inserting them. A greater percentage of the Equation Graph is visited due to the insertion of the geometric constraints  $C8$  and  $C9$  because such insertion requires the search for free variables



as explained in Section 4.4.3. This is the reason for an increased time to insert these constraints. In cases where a *soft* constraint or a free variable can not be found, the percentage of the Equation Graph visited is further increased. For example, the percentage of the Equation Graph visited in Figure 3.10 in Chapter 3 (where a constraint cycle is locally identified for the cantilever beam problem) is 70%.

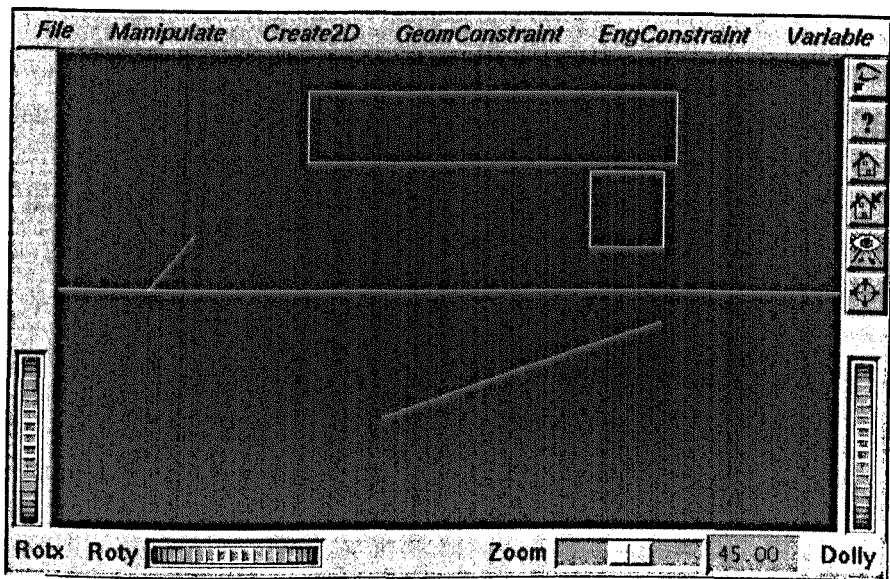
Equations	Percentage	Timing ( $\mu$ sec)
$C1 : power = \alpha \times displacement \times compression\_ratio$	2.00%	1.01327
$C2 : compression\_ratio = \frac{2 \times a + top\_gap}{top\_gap}$	2.00%	1.01327
$C3 : displacement = 2 \times a \times \pi \times r_1^2 \times n$	1.75%	1.01327
$C4 : x_{2a} = x_{1a} + a * \cos \theta_a$	1.75%	1.01327
$C5 : y_{2a} = y_{1a} + a * \sin \theta_a$	1.75%	1.01327
$C6 : x_{2b} = x_{1b} + b * \cos \theta_b$	1.70%	1.01327
$C7 : y_{2b} = y_{1b} + b * \sin \theta_b$	1.65%	1.01327
$C8 : x_{2a} = x_{1b}$	3.25%	2.02655
$C9 : y_{2a} = y_{1b}$	3.25%	2.02655

Table 6.1: Visited Percentage and Timing During Constraint Insertion

Although the case study presented in this section contains a reasonable number of equations (approximately 50 equations), the gain of efficiency in solving design constraints is equally expected for larger constraints networks, due to the incremental nature of the constraint engine. Other case studies have been presented in the literature [16, 74] where the constraint set can reach the order of over 400 equations.

### 6.2.3 Ability to Handle Under-Constrained Geometry

The purpose of this section is to demonstrate the ability of the constraint engine to handle under-constrained geometry. In Figure 6.2(a) the user has already imposed a *coincident* constraint between frame  $d$  and the crank  $a$ . Next, a *coincident end-pts* constraint is inserted between the crank  $a$  and the connecting rod  $b$ . Figure 6.3 shows the equation derived from this geometric constraint (dashed lines).



(a) Plate Before Imposing a Coincident Constraint - Crank  $a$  and Connecting Rod  $b$

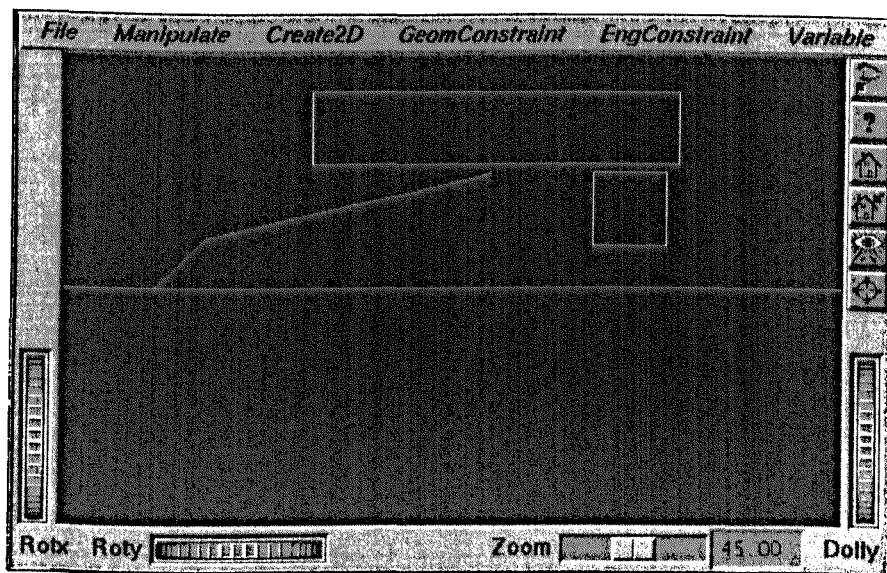


Figure 6.2: (b) Plate After Imposing a Coincident Constraint - Crank  $a$  and Connecting Rod  $b$

Note that before this constraint insertion, crank  $a$  and connecting rod  $b$  are under-constrained. Crank  $a$  has a rotational DOF and connecting rod  $b$  has a translational and rotational DOFs. These DOFs are maintained in the Equation Graph using *soft* constraints. Figure 6.2 shows that the *coincident end-pts* constraint has been satisfied even though the crank  $a$  and the connecting rod  $b$  are not fully constrained.

### Equation Graph

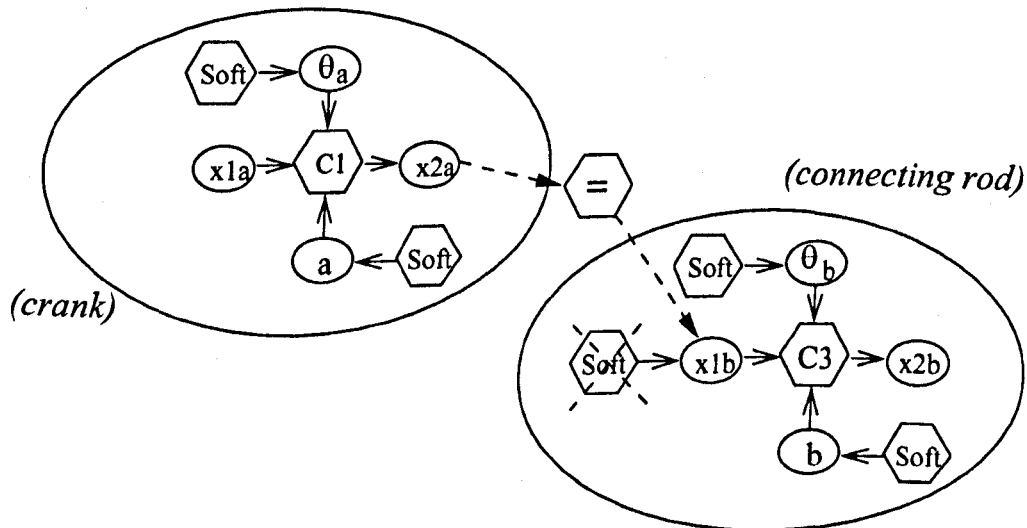
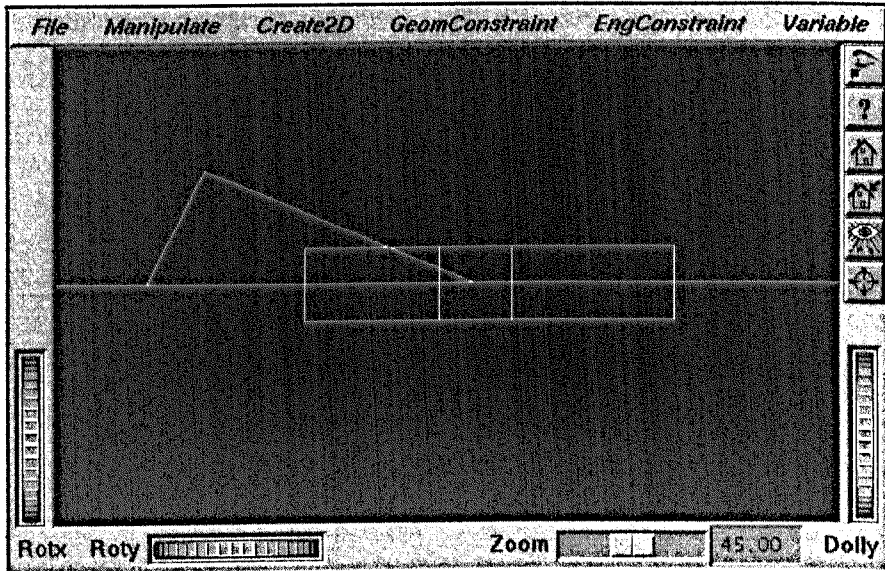


Figure 6.3: Updating of the Equation Graph

However, the soft constraint on  $x_{1b}$  (and  $y_{1b}$ ) have now been removed to reflect the new DOF of connecting rod  $b$  as seen in Figure 6.2. Approximately, 5% of the Equation Graph is visited to satisfy the equations derived from this constraint. This shows that the constraint engine is able to efficiently satisfy geometric constraints and handle under-constrained geometry.

#### 6.2.4 Coupling of Engineering and Geometric Constraints

The purpose of this section is to show the coupling of engineering and geometric constraints. Figure 6.4(a) and (b) respectively show the updating of the geometry after a coupled set of engineering and geometric constraints is satisfied. The updating of the geometry after the satisfaction of engineering constraints is explained as follows.



(a) Plate Before the Satisfaction of Engineering Constraints

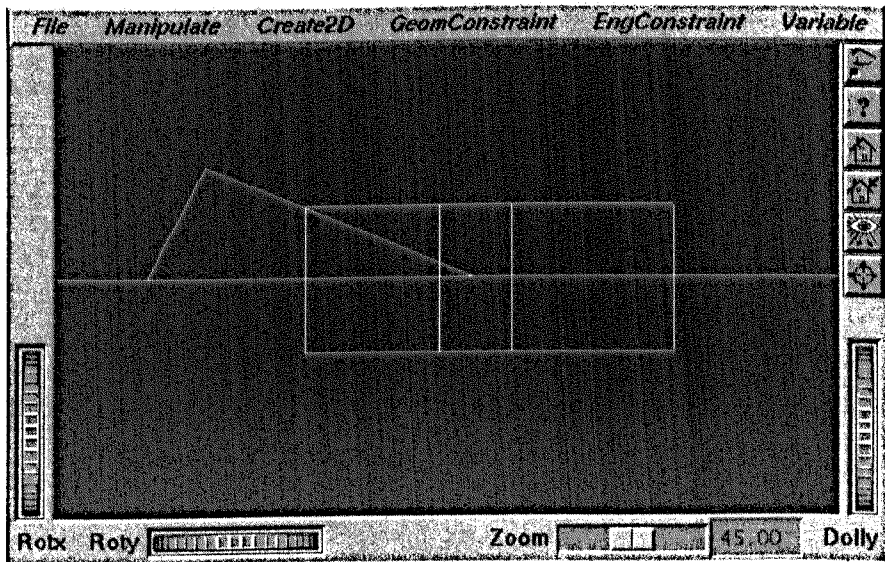


Figure 6.4: (b) Plate After the Satisfaction of Engineering Constraints

Figure 6.5(a) shows the current Equation Graph after the insertion of the engineering constraints. In this figure, the user wants to fix the value of variable  $n$  which represents the number of cylinders<sup>1</sup>. To fix variable  $n$ , the INCES algorithm searches for a free variable in the Equation Graph. Thus, the *soft* constraint on  $r_1$  is eliminated and the constraint dependency is updated as shown in Figure 6.5(b). After satisfying the engineering constraint  $C3$ , the radius  $r_1$  of the piston  $c$  is updated. The radius  $r_2$  of the casing rectangular box is also updated through the propagation to the engineering constraint  $r_2 = r_1$ . Constraint propagation continues downstream the Equation Graph until all remaining geometry coordinates are updated.

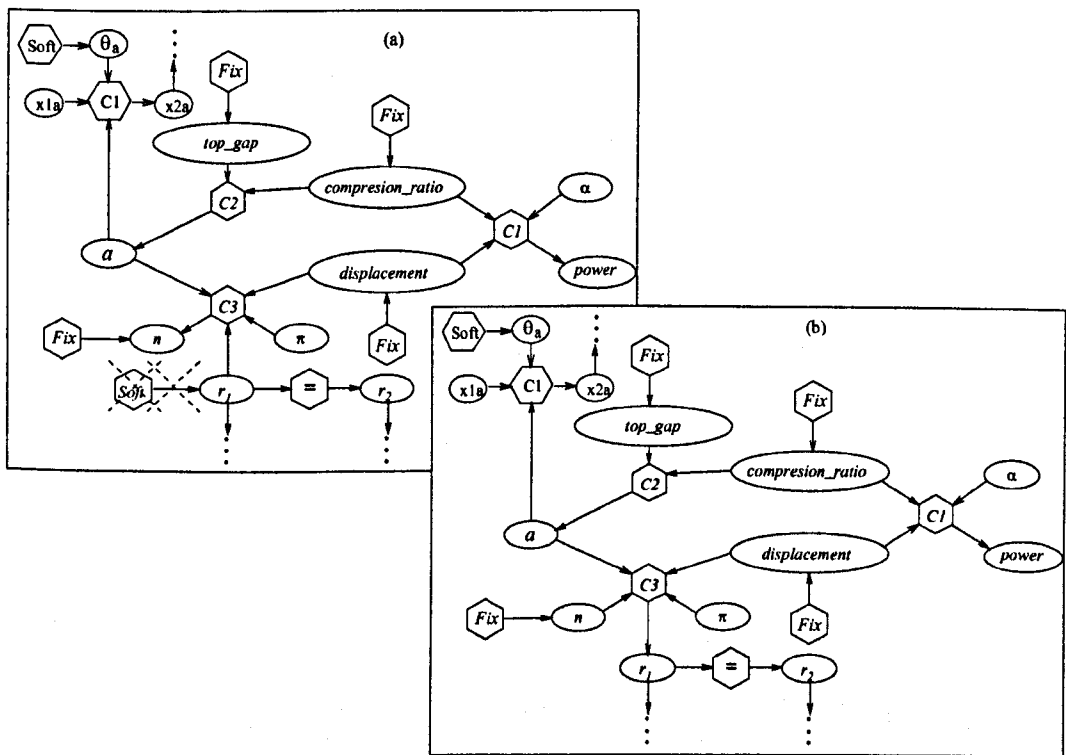


Figure 6.5: Updating of Geometry through Engineering Constraint Satisfaction

When fixing variable  $n$  only 6% of the Equation Graph is visited to locally insert this constraint. After satisfying the engineering constraint  $C3$ , approximately 32% of the Equation Graph is visited through constraint propagation to update the geometry.

<sup>1</sup>Since  $n$  is always supposed to have an integer value, constraint satisfaction techniques where a specific domain for variables is defined [97] is worth investigating.

### 6.2.5 Direct Manipulations

The purpose of this section is to show how the constraint engine supports direct manipulations of under-constrained models.

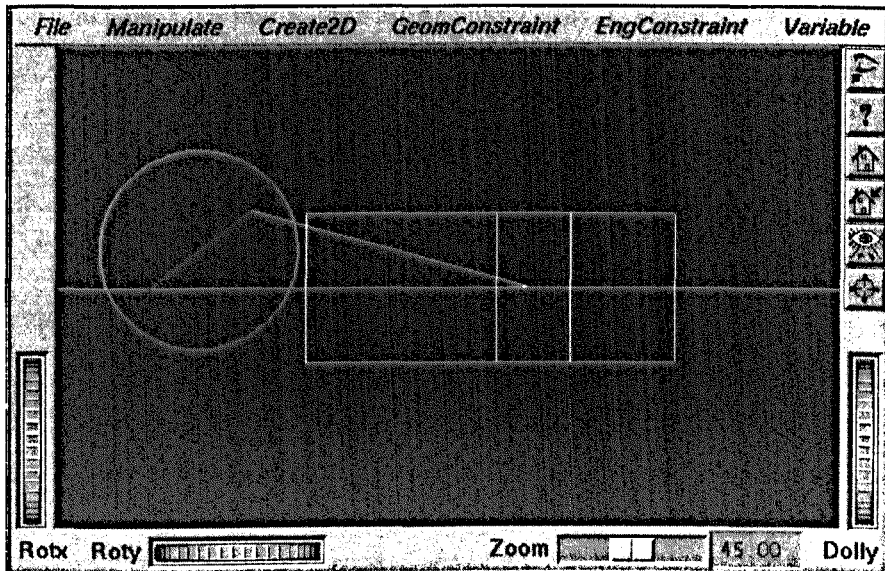


Figure 6.6: Frame Showing the Simulation of the Internal-combustion Engine

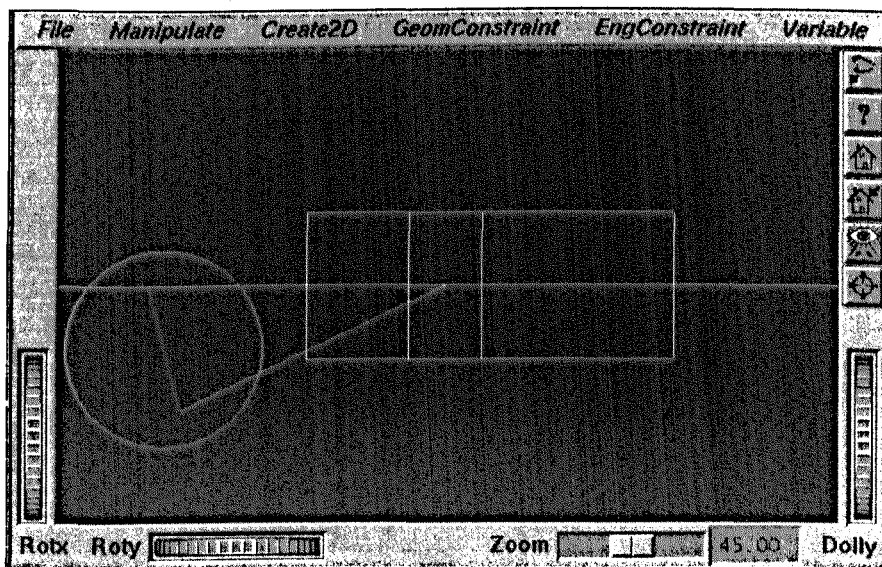


Figure 6.7: Another Frame of Simulation of the Internal-combustion Engine

Figure 6.6 shows that the user selected the crank  $a$ . This is illustrated by the circle created by Inventor when a geometric entity is selected for rotation. As shown in Figure 6.5, the crank  $a$  has a rotational degree of freedom (*soft* constraint on  $\theta_a$ ). Since the internal-combustion engine corresponds to a close-loop mechanism (slider-crank), the constraint engine uses the techniques discussed in Section 4.5.3 to support direct manipulation within a geometric constraint cycle. The constraint engine handles the geometric constraint cycles without resorting to numerical methods. This illustrates how the constraint engine can efficiently support direct manipulations of under-constrained geometry. Figure 6.7 shows another frame of the mechanism after the direct manipulation of the crank  $a$ .

### 6.3 Comparison with Previous Work

The purpose of this section is to discuss the advantages of this work in comparison with previous work in the literature. This discussion is based on the two case studies carried out in this thesis: the cantilever beam problem and the design of the internal-combustion engine. Table 6.2 below presents a general comparison of the capabilities of the constraint engine (referred in the table as to INCES) against some of the current constraint-based systems.

<i>System</i>	SkyBlue	CONSTRAINT	GCE	D-Cubed	Erep	DesignPack	INCES
Reference	[90]	[117]	[60]	[74]	[14]	[93]	thesis
Incremental	✓	✓	×	×	×	×	✓
Nonlinear Equations	×	×	✓	✓	✓	✓	✓
Constraint Cycles	×	×	✓	✓	✓	✓	✓
Constraint Coupling	✓	✓	×	×	×	✓	✓
Handling of Under-Constrained Nets.	✓	✓	×	×	×	×	✓

Table 6.2: Capabilities of Known Constraint-based Systems

### 6.3.1 Efficiency in Solving Design Constraints

Previous approaches re-satisfy the entire constraint set due to the insertion of a new constraint [14, 60, 93]. In contrast, the INCES algorithm avoids satisfying the constraint set from scratch by visiting only a small percentage of the Equation Graph. Therefore, an evolving solution is maintained by locally updating the previously satisfied constraints. This saves a significant amount of time when updating the geometry. Such incremental approach is required to allow the constraints to be solved quickly enough to provide the user with immediate feedback in interactive design.

Furthermore, in other approaches such as that proposed by Serrano [93], the user is involved in deriving the sequence of satisfaction since he is required to specify the set of known and unknown variables any time a new constraint is inserted. This procedure can be time-consuming and error-prone for large networks. On the contrary, the proposed constraint engine avoids such requirement since the constraint dependency is always automatically updated due to a constraint insertion.

It has been shown that for an acyclic constraint network composed only by linear constraints the time complexity of local propagation algorithms for inserting a design constraint is, in general,  $O(n)$  where  $n$  is the total number of constraints [65, 117]. Such behaviour is also expected by the INCES algorithm under the same conditions. However, in real design situations, nonlinear constraints and constraint cycles always emerge during the design process. In addition, the performance of the algorithm depends on how far a free variable is from the inserted constraint. Thus, as the size of a set of design constraints becomes larger and larger, it is expected that the performance of the INCES algorithm is exponential.

### 6.3.2 Handling of Under-constrained Geometry

As mentioned before, most of the current constraint-based design systems do not handle under-constrained situations [14, 16, 60]. These systems require the user to specify a fully constrained set of design constraints. For example, in Figure 6.2, these systems would not satisfy the *coincident end-pts* constraint between the crank  $a$  and the connecting rod  $b$  at that stage, since more geometric constraints would be required to fully constrain the entire constraint set defined by the internal-combustion engine. This can obstruct the designer's train of thought during preliminary design. The constraint engine supports such constraint



assignment by the consumption of *soft* constraints in the Equation Graph (see Figure 6.3). Furthermore, current systems do not support direct manipulations of under-constrained geometric entities. Instead, after deriving a sequence of satisfaction for a fully-constrained set, these systems allow the kinematic simulation of mechanism by iteratively changing the value of a variable [16, 74]. For example, one could simulate the internal-combustion engine as presented in Section 6.2.1 by iteratively changing the angle between the crank  $a$  and the frame  $d$ . However, if the user wants to manipulate another link, all constraint satisfaction sequence is derived from scratch in these systems, since this requires the deletion and insertion of new constraints. In contrast, the constraint engine supports under-constrained geometry through the concept of *soft* constraints. *Soft* constraints identify the degrees of freedom of picked geometric entities. This allows the constraint engine to update the values of the *soft* constraints according to the direct manipulation handled by the user. Fast geometry updating is also achieved through local constraint propagation in the Equation Graph (see Section 4.5).

## 6.4 Limitations and Improvements

The constraint engine reported in this thesis is only a prototype to demonstrate the feasibility of the proposed techniques. This section describes a number of limitations identified from initial experiments with the prototype engine and discusses possible solutions to them.

As mentioned in Section 3.3.2, a constraint cycle is solved through numeric iterative methods. Although nonlinear systems may have an exponential number of solutions, the Newton-Raphson iteration method will find only one. Thus, if the user obtains a solution that he did not expect, there is no standard way for debugging it. The user would have to experiment with different values for input variables until he gets the desired result. However, from a practical point of view, this might be tedious and time consuming.

Another limitation is related to the incremental insertion of geometric constraints. As explained in Section 4.4.3, when an algebraic equation derived from a geometric constraint is inserted in the Equation Graph, the INCES algorithm searches for a free variable in the ancestor subgraph. However, more than one *soft* constraint can be found at the same level during this searching process and therefore the constraint engine has to decide which *soft* constraint is to be used. This requires the use of procedures for each specific geometric con-

straint problem. The development of such specific geometric knowledge is seen as a general disadvantage as pointed out by researchers when referring to the Geometric Constructive Approach [16, 60]. To illustrate this problem, consider for example Figure 6.8. In this figure, the user wants to impose a *tangent circle-circle* constraint between circles  $C1$  and  $C2$ . Circle  $C1$  is fixed.

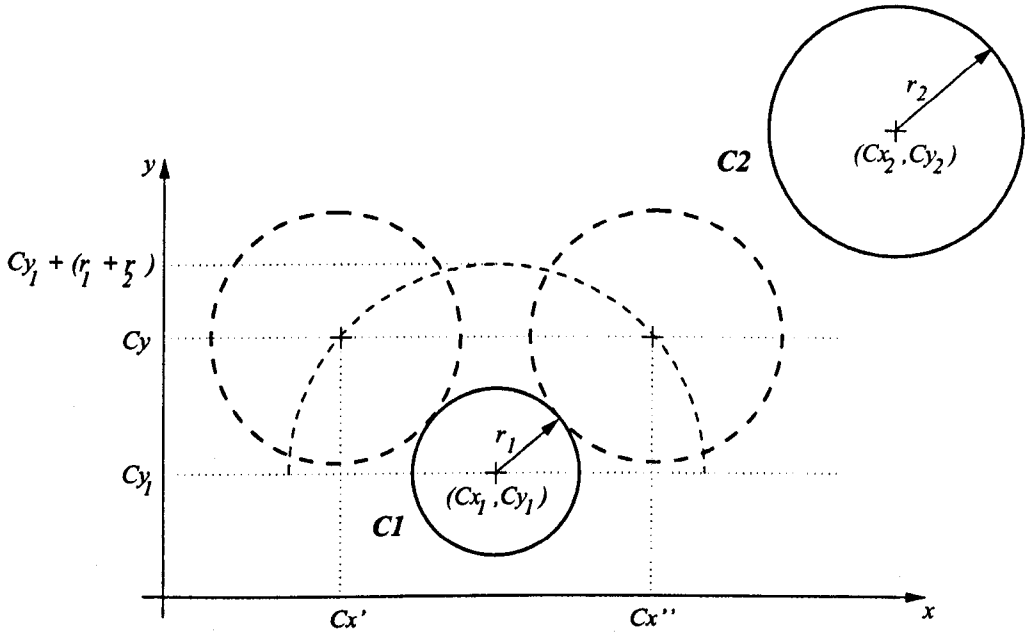


Figure 6.8: A Geometric Constraint Problem

The equation derived by the constraint engine to satisfy this constraint is:

$$C : (Cx_1 - Cx_2)^2 + (Cy_1 - Cy_2)^2 = (r_1 + r_2)^2 \quad (6.1)$$

Figure 6.9(a) shows the inserted constraint  $C$ . As it can be seen, circle  $C2$  has *soft* constraints imposed on  $Cx_2$ ,  $Cy_2$  and  $r_2$ , at the same level. This means that any one of these constraints could be destroyed to locally satisfy the inserted constraint (as represented by the ? symbol). Thus the circle  $C2$  could either be translated or have its radius increased to satisfy the *tangency* constraint. The current version of the constraint engine translates the circle to satisfy the constraint. This process is carried out in two steps. First, the constraint engine calculates a new value  $Cy$  for the *soft* constraint on variable  $Cy_2$  (this is shown by the darked constraint node in Figure 6.9(b)). This value is taken from the interval between  $Cy_1$  and  $Cy_1 + (r_1 + r_2)$  as shown by the geometry in Figure 6.8. Next, the soft constraint on  $Cx_2$  is destroyed and  $Cx_2$  is chosen as the output for the constraint

equation  $C$ . Equation 6.1 is then satisfied and circle  $C_2$  is translated to satisfy the *tangent* constraint. The two remaining *soft* constraints on  $r_2$  and  $Cy_2$  shows that circle  $C_2$  can still have its radius increased and also to translate around circle  $C_1$ .

This also demonstrates that there can be multiple solutions when satisfying geometric constraints. Multiple solutions can also emerge when the system is dealing with non-linear engineering constraints. The system currently chooses an arbitrary solution which may not be the one expected by the user. Therefore, further investigation is required to provide means for presenting the user with multiple solutions derived from engineering and geometric constraints and also to allow him to choose one of the solution. For example, with respect to the problem of Figure 6.8, the constraint engine could present the two solutions to the user (two dashed circles in Figure 6.8) and ask him or her to select one of them. Another alternative would be to use a set of rules based on the concept of signed distance or signed angle to automatically calculate the natural solution [37].

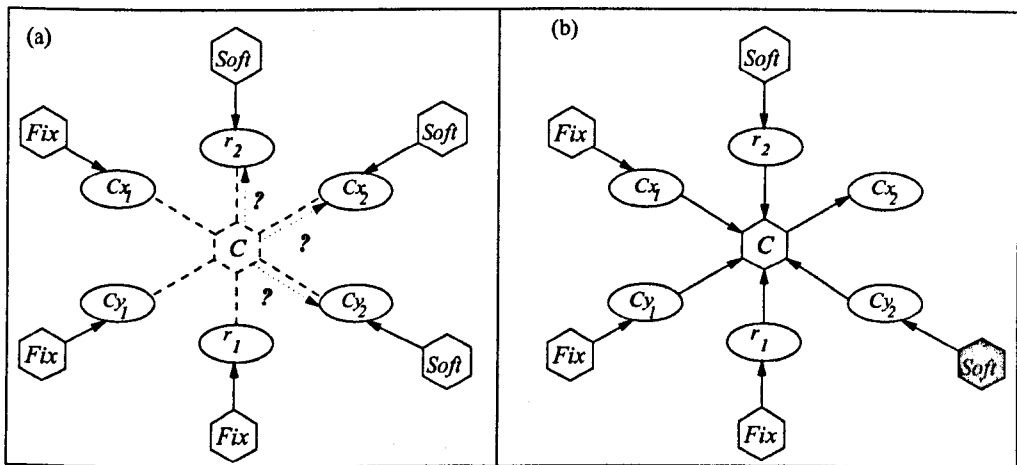


Figure 6.9: Choosing a Specific *Soft* Constraint

Other limitations identified during experiments with the constraint engine include.

#### Assembly Modeling

The prototype engine supports only simple 2D geometric constraints such as *tangency*, *coincident* and *angle* constraints. This is not enough for industrial applications [88, 98]. Other assembly relationships such as *against*, *cylindrical fit* and *spherical fit* need to be implemented. In addition, techniques to automatically recognise these assembly relationships might be beneficial for interactive design. Techniques such as *automatic constraint recogni-*

tion developed by Fa [32] and Munlin [70] can improve the interactive assembly modeling.

### Equation Editor

An editor for mathematical formulae is required. This will offer the engineers the possibility to express algebraic equations or logic formulae in a way they are used to. In addition, a parser will be necessary to convert the expressions explicitly typed by the engineer to the representation used in the Equation Graph. Representation of dimensional variables in the graphical window is also required.

## 6.5 Summary

This chapter presented the internal-combustion design problem to show how the proposed local propagation techniques can be used for supporting preliminary design. The feasibility of the GUI was demonstrated throughout the internal-combustion problem. The efficiency gain of the proposed incremental techniques were discussed and the advantages of this work in comparison with previous work were described. Limitations of the prototype constraint engine were also discussed and suggestion for further improvements were made.

The next chapter concludes this research work and consider long-term future research.

## Chapter 7

# Conclusions and Future Work

---

### 7.1 Introduction

This thesis reports on work to design and implement an interactive constraint-based engine to support preliminary design. A new approach has been developed that integrates the Equation-based Approach and the Geometric Constructive Approach. A prototype system has been implemented in C++ on a SGI XS24/4000 Indigo. The system has been evaluated using well known design problem.

This chapter draws some conclusions and discusses possible long-term research.

### 7.2 Conclusions

The primary contribution of this work is the study of new incremental algorithms for interactively supporting constraint-based design that combines both engineering and geometric constraints. A new taxonomy for current constraint-based approaches has been developed to identify advantages and disadvantages of each approach. This has resulted in the implementation of an experimental constraint-based design engine. This engine has been used to demonstrate several techniques towards supporting interactive preliminary design. These include:

- An Equation Graph is used to represent the current sequence of satisfaction of design constraints expressed as linear and nonlinear equations. This frees the user from the need to maintain constraint consistency.
- The INCES algorithm incrementally updates the Equation Graph through local propagation techniques. Constraint cycles are locally identified and solved. This allows interactive design since only a small percentage of the Equation Graph is affected due to a constraint insertion. In addition, the graph is locally updated to aid in efficiently propagating the changes to the relevant design constraints.
- The concept of *soft* constraints is used to represent the degrees of freedom of geometric entities. These *soft* constraints are consumed to locally accommodate a geometric constraint in the Equation Graph through the INCES algorithm.
- A Relationship Graph is used to represent geometric entities and geometric constraints among them. This representation supports the direct manipulations of under-constrained geometric models, according to the DOF's determined by their soft constraints in the Equation Graph. This allows the simulation of the kinematic behaviour of mechanisms in response to direct manipulations.

An experimental constraint engine has been built based on the above techniques. This constraint engine has demonstrated the capability to efficiently solve a coupled set of engineering and geometric constraints since a small percentage of the Equation Graph is visited due to a constraint insertion. In addition, the constraint engine has also demonstrated the ability to handle under-constrained geometry. Providing these features within a design scenario enables the user to interact directly with the design model and receive immediate feedback on the changes to the model.

This research has opened the way to further areas for research including the investigation of how to use the proposed techniques in 3D modeling, the handling of inequalities and feature-based modeling and the investigation of debugging tools to help engineers in analysing under- and over-constrained situations.

The next section considers how this work could be extended to conduct future research into these areas.

### 7.3 Further Research

It is the author's assertion that the constraint engine developed in this thesis will improve the productivity of designers by better supporting the early stages of design. Future work is needed to validate this assertion and this would involve experiments through a realistic engineering design scenario. The work in interactive constraint-based preliminary design can be developed in a number of ways as given below.

#### 7.3.1 Extension to 3D and VR Environments

A future research area is to extend the application domain of the constraint engine to support 3D geometry. This is important since currently the interactions between engineers and CAD workstations become more three-dimensional [75]. Extension of the proposed techniques presented in this thesis is worth investigating since a design model in 3D is also seen as a set of simultaneous nonlinear equations, where the variables are derived from the geometric elements and the equations are derived from the geometric constraints. In fact, 3D geometric elements could also be represented by *characteristic elements*. For example, a plane can be defined by its unit normal and its distance from the origin. The unit normal  $\vec{n}$  could in turn be represented by two angles  $\phi$  and  $\theta$  which are the inclination (angle from  $z$  axis to vector  $\vec{n}$ ) and the azimuth (angle from the  $x$  axis to the projection of  $\vec{n}$  on the  $xy$  plane) angles, respectively. In addition, spatial relationships between 3D geometric objects (e.g. *against, fit*) can be mathematically expressed as equalities and inequalities [4, 77]. Therefore, further research can be carried out to investigate how the concept of *soft* constraints and how the Equation Graph can be used to support such constraint representation and constraint satisfaction in 3D. An alternative way to support 3D information is to allow the user to derive a 2D profile using the constraint engine and then to use sweeping techniques to derive the 3D model. This alternative has been used in systems such as EREP from Purdue University [14] and Pro-Engineer [24].

Another area of investigation is to study the use in virtual environment applications. Currently, research is underway in the Virtual Working Environment Group at Leeds to support maintenance simulation within virtual environments. One of the research issues involved in this project is the investigation of techniques to simulate physical constraints such as gravity, friction and forces within a virtual environment [108]. Since these constraints are

expressed as equations and efficiency is required in solving them, the incremental techniques proposed in this thesis can contribute to such virtual simulation.

### 7.3.2 Feature-based Modeling

Features are meaningful collections of geometry and constraints that corresponds to shapes of engineering significance [67, 75]. Non-geometric information can be associated with features and thus the designer can better examine different and more complete design concepts. A great deal of research has been carried out towards the development of techniques to support the representation and the manipulation of high level features [46, 58, 66, 113]. The author therefore believes that the concept of **composite objects** as proposed in this thesis can be explored in order to contribute to the development of such feature-based modeling techniques.

### 7.3.3 Debugging Tools

It has been pointed out that engineers experience difficulty in constructing and understanding large set of design constraints [55, 91]. Therefore, another research area is to investigate extension mechanisms that provide engineers with debugging tools for displaying and analysing constraint networks. When evaluating different design alternatives, it is crucial to help engineers to understand the behaviour of constrained models. Essentially, such debugging tools should allow the user to visualise and explore the constraint dependency inside both the Relationship Graph and the Equation Graph.

### 7.3.4 Inequality Constraints

During the design process, many design specifications and performance measures are defined in terms of inequalities. Therefore, further research is required in order to allow the constraint engine to support inequality constraints. It is believed that new mechanisms to represent an inequality constraint may not be required since such constraint can also be represented as a hexagon node in the Equation Graph. Nevertheless, the constraint satisfaction process must be re-evaluated. Consider, for example, Figure 7.1. This figure presents a constraint conflict on variable  $x$  since it is set as the output for both constraints  $C2$  and  $C3$ . Note however that such constraint conflict causes no damage to the solution



set since both constraints are simultaneously satisfied. On the other hand, if constraint  $C1$  is replaced by  $C1 : y = 0.5$  such constraint conflict is unacceptable. Clearly, investigation to deal with such constraint conflicts is required. Debugging tools could be used here to guide the user to identify the value for variable  $y$  as the cause of the conflict and to suggest him to change its value. Additional symbolic processing of constraints may also be required due to inequality constraints since the goal of the constraint satisfaction process is to find a point in the search space that does not violate any constraints. Finally, the problem of how to solve a constraint cycle composed by inequality constraints must also be investigated.

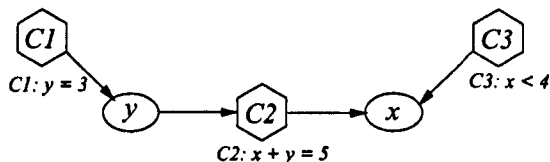


Figure 7.1: Constraint Conflict with Inequality

## 7.4 Concluding Remarks

This research has demonstrated an interactive constraint-based design engine based on local propagation techniques. This work makes contribution in the following areas:

- the development of a new taxonomy for current constraint-based approaches;
- the development of an incremental algorithm (INCES) that supports highly nonlinear engineering constraints and constraint cycles efficiently;
- the use of *soft* constraints to represent the DOF of geometric entities. These *soft* constraints are used to satisfy geometric constraints and to support direct manipulations of under-constrained geometry.

This work contributes to enhance the understanding of techniques for supporting interactive constraint-based preliminary design.

# Bibliography

- [1] Aho, A.V. and Ullman, J.D., *Foundations of Computer Science*, Computer Science Press, New York, 1992, ISBN 0-7167-8233-2.
- [2] Aldefeld, B., Variation of Geometries Based on a Geometric Reasoning Method, *Computer-Aided Design*, Vol. 20, No. 3, pp. 117-126, April 1988.
- [3] Amador, F.G., Finkelstein, A. and Weld, D.S., Real-Time Self-Explanatory Simulation, In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 562-567, AAAI Press/The MIT Press, July 1993.
- [4] Ambler, A.P. and Popplestone, R.J., Inferring the Positions of Bodies from Specified Spatial Relationships, *Artificial Intelligence*, Vol. 6, pp. 157-174, 1975.
- [5] Anderl, R. and Mendgen, R., Modeling with Constraints: Theoretical Foundation and Application, *Computer-Aided Design*, Vol. 28, No. 3, pp. 155-168, March 1996.
- [6] Arbab, F. and Wang, B., A Constraint-Based Design System Based on Operational Transformation Planning, In *Proceedings of the 4th International Conference on the Applications of Artificial Intelligence in Engineering*, pp. 405-426, Cambridge-UK, July 1989.
- [7] Baase, S., *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley, Reading, Mass., 1978, ISBN 0201003279.
- [8] Badler, N.I., Manoochehri, K.H. and Baraff, D., Multi-dimensional Input Techniques and Articulate Figure Positioning by Multiple Constraint, In *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, Crow, F. and Pizer, S. (Eds), ACM Press, pp. 151-169, October 23-24 1986.

- [9] Becher, T. and Weispfenning, V., *Gröbner Bases - A Computational Approach to Commutative Algebra*, Graduate Texts in Mathematics, Springer-Verlag, New York, 1993, ISBN 0-387-9791-9.
- [10] Bhansali, S., Kramer, G.A. and Hoar, T.J., A Principled Approach Towards Symbolic Geometric Constraint Satisfaction, *Journal of Artificial Intelligence Research*, Vol. 4, pp. 419-443, 1996.
- [11] Bier, E. A., Skitters and Jacks: Interactive 3D Positioning Tools, *Visual Computer*, pp. 183-196, October 1986.
- [12] Bier, E. A., Snap-dragging in Three Dimensions, *1990 Symposium on Interactive 3D Graphics*, pp. 193-204, 1990.
- [13] Borning, A., Freeman-Benson, B. and Wilson, M. Constraint Hierarchies, *Lisp and Symbolic Computation*, Vol. 5, No. 3, pp. 223-270, September 1992.
- [14] Bouma, W., Fudos, I., Hoffmann, C., Cai, J. and Paige, R., Geometric Constraint Solver, *Computer-Aided Design*, Vol. 27, No. 6, pp.487-501, June 1995.
- [15] Brüderlin, B., Constructing Three-Dimensional Geometric Objects Defined by Constraints, In *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, Crow, F. and Pizer, S. (Eds), ACM Press, pp. 111-129, October 23-24 1986.
- [16] Brunkhart, M.K., *Interactive Geometric Constraint Systems*, Master's thesis, Computer Science Division/University of California, Berkeley, 1994.
- [17] Buchanan, S.A., Some Theoretical Problems when Solving Systems to Polynomials Equations using Gröbner Bases, *ACM SIGSAM Bull.*, Vol. 25, No. 2 pp. 24-27, 1991.
- [18] Buchanan, S.A. and de Pennington, A., Constraint Definition System: a Computer-Algebra based Approach to Solving Geometric-Constraint Problems, *Computer-Aided Design*, Vol. 25, No. 12, pp. 741-750, December 1993.
- [19] Buchberger, B., A Theoretical Basis for the Reduction of Polynomials to Canonical Forms, *ACM SIGSAM Bull.*, Vol. 39, pp. 19-29, 1976.
- [20] Butterworth, A.D., Hench, S. and Marc Olano, T., 3DM: A Three Dimensional Modeller Using Head-Mounted Display, *1992 Symposium on Interactive 3D Graphics*, 1992.

- [21] Char, B.W., Geddes, K.O., Gonnet, G.H., Leong, B.L., Monahan, M.B. and Watt, S.M., *First Leaves: A Tutorial Introduction to Maple V*, Springer-Verlag, 1992.
- [22] Chen, X. and Hoffmann, C.M., Design Compilation of Feature-Based and Constraint-Based CAD, *ACM/SIGGRAPH Third Symposium on Solid Modeling and Applications*, pp. 13-19, Salt Lake City, Utah, May 1995.
- [23] Chung, J. and Schussel, M., Comparison of Variational and Parametric Design, In *Proceedings of Autofact '89*, pp. 5/27-5/44, 1989.
- [24] Clarke, C., Pro-Engineer, *CAD/CAM*, Vol. 12, No. 1, January 1993.
- [25] Cormen, T. H., Leiserson, C.E. and Rivest, R.L., *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts, USA, 1994.
- [26] D-Cubed Ltd., The 2 Dimensional DCM Technical Overview, Technical Report, November 1993.
- [27] Dohmen, M., A Survey of Constraint Satisfaction Techniques for Geometric Modeling, *Computers & Graphics*, Vol. 19, No. 6, pp. 831-845, 1995.
- [28] Drinkart, R.D. and Sulinski, N.K., *MACSYMA: A Program for Computer Algebra Manipulations*, Naval Underwater Systems Center, Newport, Rhode Island, NUSC Technical Document 6401, 1981.
- [29] Edelen, D.G.B and Kydoniefs, A.D., *An Introduction to Linear Algebra for Science and Engineering*, 2nd edition, American Elsevier Publishing Company, Inc., 1976, ISBN 0-444-0019-56.
- [30] van Emmerik, M.J.G.M., Creation and Modification of Parameterized Solid Models by Graphical Interaction, *Computers & Graphics*, Vol. 13, No. 1, pp. 71-76, 1989.
- [31] van Emmerik, M.J.G.M., Interactive Design of 3D Models with Geometric Constraints, *The Visual Computer*, Vol. 7, No. 5/6, pp. 309-325, 1991.
- [32] Fa, M., *Interactive Constraint-based Solid Modeling*, PhD thesis, Leeds University, School of Computer Studies, UK, September 1993.
- [33] Fa, M., Fernando, T. and Dew, P.M., Interactive Constraint-based Solid Modelling using Allowable Motion, *ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pp. 243-252, May 1993.

- [34] Fernando, T., Fa, M., Dew, P.M. and M.Munlin, "Constraint-based 3D Manipulation Techniques within Virtual Environments", *Virtual Reality Applications*, Academic Press, 1995, pp. 71-89, ISBN 0-12-227755-4.
- [35] French, M., *Conceptual Design for Engineers*, 2nd edition, Design Council, London, 1985.
- [36] Freeman-Benson, B., Maloney, J. and Borning, A., "An Incremental Constraint Solver", *Communications of the ACM*, Vol. 33, No. 1, pp. 54-63, January 1990.
- [37] Fudos, I., *Editable Representations for 2D Geometric Design*, Master thesis, Purdue University, USA, December 1993.
- [38] Geiger, T.S. and Dilts, D.M., "Automated Design-to-Cost: Integrating Costing into the Design Decision", *Computer-Aided Design*, Vol. 28, No. 6/7, pp. 423-438, Junho/Julho 1996.
- [39] Gifford, R., "Workstations Bring Benefits of Variational Geometry to Mechanical Engineers", *MicroCAD News*, pp. 56-58, November 1990.
- [40] Gleicher, M.L., "Integrating Constraint and Direct Manipulations", *1992 Symposium on Interactive 3D Graphics*, pp. 171-174, 1992.
- [41] Gleicher, M.L., "Practical Issues in Graphical Constraints", *First Principles and Practice of Constraint Programming Workshop (PPCP'93)*, Newport, RI, 1993, [ftp://wilma.cs.brown.edu/pub/ppcp93](http://wilma.cs.brown.edu/pub/ppcp93).
- [42] Gleicher, M.L., *A Differential Approach to Graphical Interaction*, PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, USA, November 1994.
- [43] Gossard, D.C., Zuffante, R.P. and Shakurai, H., "Representing Dimensions, Tolerances and Features in MCAE Systems", *IEEE Computer Graphics and Applications*, pp. 51-59, March 1988.
- [44] Green, W.G., *Theory of Machines*, 2nd ed., Blackie & Son Limited, 1962.
- [45] Grosjean, J., *Kinematics and Dynamics of Mechanisms*, McGraw-Hill Book Company Ltd, London, England, 1991.
- [46] Gupta, S.K., Regli, W.C and Nau, D.S., "Manufacturing Feature Instances: Which Ones to Recognize?", *ACM/SIGGRAPH Third Symposium on Solid Modeling and Applications*, pp. 141-152, Salt Lake City, Utah, May 1995.

- [47] Harper, D., Wooff, C. and Hodgkinson, D., *A Guide to Computer Algebra Systems*, John Wiley & Sons, 1991, ISBN 0-471-92910-7.
- [48] Hillyard, R.C. and Braid, I.C., Analysis of Dimensions and Tolerances in Computer-Aided Mechanical Design, *Computer-Aided Design*, Vol. 10, No. 6, pp. 161-166, June 1978.
- [49] Hoffmann, C.M. and Rossignac, J.R., A Road Map To Solid Modeling, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 2, No. 1, March 1996.
- [50] Hosobe, H., Miyashita, K., Takahashi, S., Matsuoka, S. and Yonezawa, A., Locally Simultaneous Constraint Satisfaction. *Second Principles and Practice of Constraint Programming Workshop (PPCP'94)*, Washington, USA, 1994, <ftp://ftp.cs.washington.edu/pub/constraints/ppcp94>.
- [51] Juster, N.P., *A Graph Based Approach to Tolerance Analysis*, PhD thesis, Department of Mechanical Engineering, The University of Leeds, UK, September 1988.
- [52] Juster, N.P., Modelling and Representation of Dimensions and Tolerances: a Survey, *Computer-Aided Design*, Vol. 24, No. 1, pp. 3-17, January 1992.
- [53] Kahaner, D., Moler, C., and Nash. S., *Numerical Methods and Software*, Prentice Hall, Englewood Cliffs, NJ, 1989.
- [54] Kaufman, A. and Yagel, R., Direct Interaction with a 3D Volumetric Environment, *1990 Symposium on Interactive 3D Graphics*, pp. 33-34, 1990.
- [55] Keirouz, W.T., Kramer, G.A. and Pabon, J., Exploiting Constraint Dependency Information for Debugging and Explanation, *First Principles and Practice of Constraint Programming Workshop (PPCP'93)*, Newport, RI, 1993, <ftp://wilma.cs.brown.edu/pub/ppcp93>.
- [56] Kondo, K., Algebraic Method for Manipulation of Dimensional Relationships in Geometric Models, *Computer-Aided Design*, Vol. 24, No. 3, pp. 141-147, March 1992.
- [57] de Kraker, K.J., Dohmen, M. and Bronsvort, W.F., High-level Constraints in Interactive CSG Modelling, In *Proceedings of CSG94: Set-theoretic Solid Modelling Techniques and Applications*, pp. 275-289, Ed. Information Geometers Ltd, Winchester, UK, 13-15 April 1994, ISBN 1-874728-05-4.

- [58] de Kraker, K.J., Dohmen, M. and Bronsvort, W.F., Multiple-way Feature Conversion to Support Concurrent Engineering, *ACM/SIGGRAPH Third Symposium on Solid Modeling and Applications*, pp. 105-114, Salt Lake City, Utah, May 1995.
- [59] Kramer, G.A., *Solving Geometric Constraint Systems: A Case Study in Kinematics*, MIT Press, Cambridge, Massachusetts, 1992.
- [60] Kramer, G.A., A Geometric Constraint Engine, *Artificial Intelligence*, Vol. 58, pp. 327-360, December 1992.
- [61] Light, R. and Gossard, D., Modification of Geometric Models through Variational Geometry, *Computer-Aided Design*, Vol. 14, No. 4, pp. 209-214, July 1982.
- [62] Lin, V.C., Gossard, D.C. and Light, R.A., Variational Geometry in Computer Aided Design, *ACM Computer Graphics (SIGGRAPH'81)*, Vol. 15, No. 3, pp. 171-175, August 1981.
- [63] Lucas, W. K, Kim Roddis, W.M. and Brown, F.M., Constraint-Based Reasoning for Structural Concrete Design and Detailing, *Constraint-95: Workshop on Constraint-Based Reasoning*, FLAIRS-95, April 1995.
- [64] Mabie, H.H. and Ocvirk, F.W., *Mechanisms and Dynamics of Machinery*, John Wiley and Sons, 1978.
- [65] Maloney, J., *Using Constraints for User Interface Construction*, PhD Thesis, Department of Computer Science and Engineering, University of Washington, August 1991.
- [66] Mandorli, F., Otto, H.E. and Kimura, F., A Reference Kernel Model for Feature-based CAD Systems Supported by Conditional Attributed Rewrite Systems, *ACM/SIGGRAPH Second Symposium on Solid Modeling and Applications*, pp. 343-354, Montreal, Canada, 1993.
- [67] Mäntylä, M., A Modeling System for Top-Down Design of Assembled Products, *IBM Journal of Research Development*, Vol. 34, No. 5, pp. 636-659, September 1990.
- [68] Mathews, J.H., *Numerical Methods for Mathematics, Science and Engineering*, Prentice-Hall International, Inc., 1992, ISBN 0-13-625047-5.
- [69] Maxfield, J., Fernando, T. and Dew, P.M., A Distributed Virtual Environment for Concurrent Engineering, in *Virtual Prototyping - Virtual Environments and the Product Development Process*, Chapman & Hall, July 1995.

- [70] Munlin, M., *Interactive Assembly Modeling within a Virtual Environment*, PhD thesis, Leeds University, School of Computer Studies, UK, September 1995.
- [71] Numerical Algorithms Group Limited, *The NAG Fortran Library Manual – Mark 15*, 1995.
- [72] Nelson, G., Juno, A Constraint-based Graphics System, *SIGGRAPH'85*, Vol. 19, No. 3, ACM Press, pp. 235-243, San Francisco, July 1985.
- [73] Nielson, G. M., Direct Manipulation Techniques for 3D Objects Using 2D Locator Devices, In *Proceedings of 1986 Workshop on Interactive 3D Graphics*, pp. 175-182, Crow, F. and Pizer, S.M. (Eds), October 1986.
- [74] Owen, J.C., Algebraic Solution for Geometry from Dimensional Constraints, *ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pp. 397-407, June 1991.
- [75] Pabon, J., Young, R. and Keirouz, W., Integrating Parametric Geometry, Features and Variational Modelling for Conceptual Design, *International Journal of System Automation: Research and Applications (SARA)*, Vol. 2, pp. 17-32, 1992.
- [76] Pahl, G. and Beitz, W., *Engineering Design*, Design Council, London, 1984.
- [77] Popplestone, R.J., Ambler, A.P. and Bellos, I.M., An Interpreter for a Language Describing Assemblies, *Artificial Intelligence*, Vol. 14, pp. 79-107, 1980.
- [78] Porter, S., Winds of Change: Are Users Ready to Switch CAD Systems to Get the Benefits of Solids?, *Computer Graphics World*, pp.34-40, February 1991.
- [79] Press, W.H. et al., *Numerical Recipes in C: the Art of Scientific Computing*, Cambridge University Press, 2nd edition, 1992.
- [80] Prime, M.J., Human Factors Assessment of Input Devices for EWS, Technical report, Rutherford Appleton Laboratory, April 1991.
- [81] Rayna, G., *REDUCE: Software for Algebraic Computation*, Springer-Verlag, 1987, ISBN 038796598X.
- [82] Requicha, A.A.G., Representations for Rigid Solids: Theory, Methods and Systems, *Computing Surveys*, Vol. 12, No. 4, December 1980.



- [83] Requicha, A.A.G., Representations of Tolerance in Solid Modeling: Issues and Alternative Approaches. In *Solid Modeling by Computers from Theory to Applications*, 1984.
- [84] Requicha, A.A.G. and Rossignac, J.R., Solid Modelling and Beyond, *IEEE Computer Graphics and Applications*, pp. 31-44, September 1992.
- [85] Roller, D., An Approach to Computer Aided Parametric Design, *Computer-Aided Design*, Vol. 23, No. 5, pp. 385-391, June 1991.
- [86] Rossignac, J.R., Constraints in Constructive Solid Geometry, In *Proceedings of 1986 Workshop on Interactive 3D Graphics*, pp. 93-110, Crow, F. and Pizer, S.M. (Eds), October 1986.
- [87] Rossignac, J.R., Through the Cracks of the Solid Modeling Milestone, *Eurographics'91 State of the Art Report on Solid Modeling*, pp. 23-109, 1991.
- [88] Roy, U. and Lin, C.R., Establishment of Functional Relationships Between Product Components in Assembly Database, *Computer-Aided Design*, Vol. 20, pp. 579-580, December 1988.
- [89] Salomons, O.W, van Slooten, F. and Kals, H.J.J., Conceptual Graphs in Constraint Based Re-design, *ACM/SIGGRAPH Third Symposium on Solid Modeling and Applications*, pp. 55-64, Salt Lake City, Utah, May 1995.
- [90] Sannella, M., The SkyBlue Constraint Solver and Its Applications, *First Principles and Practice of Constraint Programming Workshop (PPCP'93)*, Newport, RI, 1993, <ftp://wilma.cs.brown.edu/pub/ppcp93>.
- [91] Sannella, M., Analyzing and Debugging Hierarchies of Multi-Way Local Propagation Constraints, *Second Principles and Practice of Constraint Programming Workshop (PPCP'94)*, Washington, USA, 1994.
- [92] Serrano, D., *Constraint Management in Conceptual Design*, PhD Thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, October 1987.
- [93] Serrano, D. and Gossard, D., Tools and Techniques for Conceptual Design, In *Artificial Intelligence in Engineering Design*, Vol. I, C. Tong and D. Sriram (Eds), pp. 71-116, 1992.

- [94] Shapiro, V. and Vossler, D.L., What is a Parametric Family of Solids?, *ACM/SIGGRAPH Third Symposium on Solid Modeling and Applications*, pp. 43-54, Salt Lake City, Utah, May 1995.
- [95] Shigley, J.E. and Uicker Jr, J.J., *Theory of Machines and Mechanisms*, McGraw-Hill Book Company, 1980.
- [96] Sistare, S. Graphical Interaction Techniques in Constraint-based Geometric Modeling, *Graphics Interface'91*, pp. 85-93, 1991.
- [97] Smith, B.M and Grant, S.A., Where the Exceptionally Hard Problems Are, Technical Report No. 95.35, School of Computer Studies, Leeds Universtiy, 1995.
- [98] Subramaniam, S., Turner, J. and Sanderson, A., Establishing Part Positions in Assembly Modelling, In *Product Modelling for Computer-Aided Design and Manufacturing*, pp. 199-226, Elsevier Science Publishers B.V. (North-Holand), 1991.
- [99] Inc. SHAPEDATA, PARASOLID Programming Reference, 1992.
- [100] Sohrt, W. and Brüderlin, B., Interaction with Constraints in 3D Modeling, In *Proceedings Symposium on Solid Modeling Foundations and CAD/CAM Applications*, Rossignac, J. and Turner, J. (Eds), ACM Press, pp. 387-396, 1991.
- [101] Solano, L. and Brunet, P., A System for Constructive Constraint-based Modeling, In Falcidieno, B. and Kunii, T. (Eds), *Modeling in Computer Graphics*, Springer-Verlag, pp. 61-84, 1993.
- [102] Inc. Spatial Technology, ACIS Geometric Modeler, 1993.
- [103] Stephamopoulos, G., A Knowledge-Based Framework for Process Design and Control, *NSF-AAI Workshop on Artificial Intelligence in Process Engineering*, Columbia University, New York, N.Y, March 1987.
- [104] Strauss, P. S. and Carey, R., An Object-Oriented 3D Graphics Toolkit, *SIGGRAPH'92*, Vol. 26, No. 2, pp. 341-349, July 1992.
- [105] Sutherland, I., Sketchpad, A Man-Machine Graphical Communication System, In *Proceedings of Spring Joint Conference IFIP*, pp. 329-345, 1963.
- [106] Suzuki, H., Ando, H. and Kimura, F., Synthesizing Product Shapes with Geometric Design Constraints and Reasoning, In *Proc. of the IFIP TC 5/WG 5.2 Workshop on Intelligent CAD*, pp. 309-322, Cambridge, UK, September 1988.

- [107] Swamy, M.N. and Thulasiraman, K., *Graphs, Networks, and Algorithms*, John Wiley & Sons, Inc., 1981, ISBN 0-471-03503-3.
- [108] Thompson, M., First Year Report, Technical Report, School of Computer Studies, University of Leeds, September 1996.
- [109] Thornton, A.C., *Constraint Specification and Satisfaction in Embodiment Design*, PhD thesis, University of Cambridge, Department of Engineering, UK, July 1993.
- [110] Tsai, Y., Fernando, T. and Dew, P.M., Exploiting Degrees of Freedom Analysis for Interactive Constraint-based Design, In *Proceedings of The Fourth International Conference in Central Europe on Computer Graphics and Visualization (WSCG'96)*, Thalmann, N. and Skala, V. (Eds), pp. 377-387, Plzen, Czech Republic, February 1996.
- [111] Tsai, Y., *Incremental Geometric Constraint Satisfaction Algorithms*, PhD thesis, The University of Leeds, School of Computer Studies, UK, September 1996.
- [112] Verroust, A., Schonek, F. and Roller, D., Rule-oriented Method for Parameterized Computer-aided Design, *Computer-Aided Design*, Vol. 24, No. 3, pp. 531-540, October 1992.
- [113] Waco, D.L. and Kim, Y.S., Geometric Reasoning for Machining Features Using Convex Decomposition, *ACM/SIGGRAPH Second Symposium on Solid Modeling and Applications*, pp. 343-354, Montreal, Canada, 1993.
- [114] Wickens, L.P., NONAME Training Courses, Technical report, Department of Mechanical Engineering, Leeds University, UK, July 1982.
- [115] Wolter, J. and Chandrasekaran P., A Concept for a Constraint-based Representation of Functional and Geometric Design Knowledge, *ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pp. 409-418, 1991.
- [116] Yamaguchi, Y. and Kimura, F., A Constraint Modeling System for Variational Geometry, In Wozny, J., Turner, J. and Preiss, K. (Eds), *Geometric Modeling for Product Engineering*, North Holland, pp. 221-233, 1990.
- [117] Vander Zanden, B.T., A Domain-Independent Algorithm for Incrementally Satisfying Multi-Way Constraints, *Technical report CS-92-160*, University of Tennessee, Computer Science Department, 1992.