A Linear Constraint Technology for Interactive Graphic Systems

Richard Helm, Tien Huynh, Catherine Lassez, Kim Marriott

I.B.M. Thomas J. Watson Research Center,
P.O. Box 704, Yorktown Heights, NY 10598

Abstract

Constraints provide a natural formalism for user-interface design and graphical layout. Recent results and algorithms from symbolic computation and geometry provide new techniques to manipulate linear arithmetic constraints. We show how these results can be applied to interactive graphical user-interfaces and how they extend the capabilities of previous interactive constraint-based user interface systems. We propose an architecture for such systems based on these techniques.

Keywords: Linear Arithmetic Constraints, Interactive Techniques

1 Introduction

Constraints are a natural formalism for specifying user-interface design and graphical layout. They have shown their utility in interactive systems [1, 2, 3, 11, 12, 13, 14, 15], in graphic specification languages [5, 17] and recently in visual language parsing [4].

The requirements of adequate response time and graphical feedback for interactive systems demand certain capabilities of constraint solvers. In particular, previous work, for example ThingLab and ThingLab II [2, 11] suggests that constraint solvers for interactive graphics must provide:

- Incremental addition and deletion of constraints.
- Fast generation of plans of execution when the object that is the focus of manipulation changes.
- Adequate feedback bandwidth when manipulating a graphic object.

Other capabilities which constraints can provide and which an interactive constraint-based system should support are:

- Determining and presenting the range of values that a variable can take. For example, if a point is going to be dragged around on the screen, the system should be able to present graphically to the user whereabouts this point can be moved.
- Support the definition and compilation of compound objects. Although, this is straightforward in systems without constraints, the addition of constraints raises new issues. In particular, efficiently representing the constraints in the compound object, and determining which variables of a compound graphic object define all objects it contains.

In addition the constraint solver must support other, more usual, operations associated with constraints. These include:

- Detecting that a system of constraints is unsatisfiable, and identifying which constraints must be removed to restore satisfiability.
- Detecting an under-constrained system and identifying which variables must be further constrained.

Both the constraints causing unsatisfiability, and variables that need to be further constrained must be indicated to the user.

Recent results and algorithms from symbolic computation [6, 8, 9] provide powerful techniques to manipulate sets of linear arithmetic constraints containing both equalities and inequalities. We show that when applied to interactive constraint-based user-interface systems, these techniques give new capabilities, and enhance interaction through improved user-feedback. In particular, they provide:

1. A *canonical form* for a set of linear arithmetic constraints. The canonical form is concise, does not contain redundant constraints, identifies the degrees of freedom in the constraints, and makes explicit the equalities implied by the constraints. There is an incremental algorithm to compute the canonical form. This algorithm also determines whether or not the constraints are satisfiable. The canonical form is a key for both efficient representation and manipulation of constraints.

2. A *parametric solved form* for the solutions of a set of constraints in terms of distinguished parametric variables. The solved form, which corresponds to the *plans* of Thinglab II, allows the values of variables to be computed rapidly from the parameters. This permits the rapid re-satisfaction of the constraints when objects are being manipulated by the user.
3. An efficient projection algorithm to compute the range of values of distinguished variables. This allows the user to get feedback about the region within which selected objects on the screen can be moved while still satisfying the constraints. Projection also plays a role in computing the manipulable interface of compound objects.
4. Techniques to deal with unsolvable sets of constraints by identifying minimal unsolvable subsets. These provide feedback as to which constraints need to be relaxed or removed to restore solvability.
5. Techniques to determine if a set of constraints is underconstrained, that is if some variables cannot be uniquely determined in terms of certain distinguished parameters, and consequently which variables must be further constrained.

This work extends previous constraint technology for interactive constraint-based systems in two ways.

First, we present new feedback mechanisms and interaction styles for constraint-based systems. These are based on new techniques for extracting information about causes of over- and under- constrainedness in sets of constraints, and determining relationships on specific variables implied by the constraints. These techniques rely on manipulating constraints symbolically. To our knowledge these features have not appeared in previous systems.

Secondly, we extend previous work that uses symbolic techniques to solve constraints by allowing simultaneous linear equations and inequalities. This extension is important because such constraints arise naturally when specifying graphical layout. For efficiency reasons, previous systems have dealt mainly with systems of acyclic linear equations [3, 11, 15], and solved these constraints using local propagation techniques. Recent research has addressed the efficient and incremental recompilation of these types of constraints in response to user interaction [11]. For the more general constraints considered here, however, local propagation is not sufficient; global techniques must be used. We note that Witkin [16] and Nelson [12] deal with more powerful constraints, but use numerical techniques for constraint satisfaction. It is not clear how the feedback mechanisms we present can be provided using these numerical techniques.

The rest of the paper is organized as follows. In the next section, we present an example of a typical interactive session which illustrates these new techniques. In Section 3, we discuss two key elements of the underlying constraints technology: a canonical representation for constraint, and a new projection algorithm particularly suited to this application. In Section 4, we propose an architecture for a constraint manipulation sub-system to be used within an interactive user interface system. In Section 5 we present some empirical results concerning the performance of this constraint technology.

2 Example

To illustrate some of the capabilities of the constraint technology, we present a simple hypothetical session with an interactive constraint-based editor.

Consider the diagram illustrated in Figure 1 which consists of two pieces of text $T1$ and $T2$, two rectangles $R1$ and $R2$ and a surrounding box B. Suppose that the user wishes to satisfy the following requirements.

1. $R1$ and $R2$ have a fixed aspect ratio.
2. $R1$ and $R2$ have the same size and cannot overlap.
3. Ti is centered in Ri and Ri is large enough to contain Ti.
4. B contains both $R1$ and $R2$, and they are "nicely" placed inside B, that is the rectangles are equidistant from the borders of B and from each other.

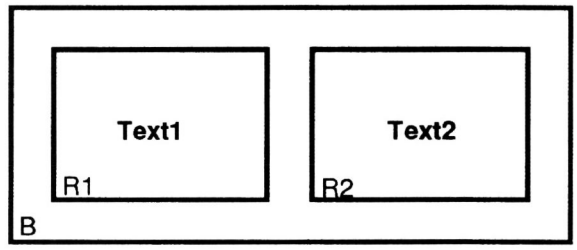

Figure 1: Layout of the diagram

To create this diagram, the user adds and deletes graphic objects and constraints and modifies the parameters (or attributes) of the objects. The resulting layout is defined by a set of constraints. These fall into the following categories: *local* constraints, which express the relationships between the modifiable parameters of system-defined objects; *global* constraints, provided by the user, which express the relations between objects; and *anchor* constraints which express that certain points or attributes are fixed.

Typically, local constraints are pre-defined for each object, and are highly redundant to allow multiple ways to define an object. In our example, each rectangle may be defined by its opposite vertices or its center and a vertex. Thus each rectangle has as parameters, its four vertices - bl, br, ul, ur for bottom-left, bottom right, etc...; its center point c, and its extent e in x and y dimensions. The

local constraints for an object O over these parameters are

$$\begin{array}{ll} O_{cx} = 0.5 * O_{blx} + 0.5 * O_{brx} & O_{bry} = O_{bly} \\ O_{cy} = 0.5 * O_{bly} + 0.5 * O_{uly} & O_{ulx} = O_{blx} \\ O_{brx} = O_{blx} + O_{ex} & O_{urx} = O_{brx} \\ O_{uly} = O_{bly} + O_{ey} & O_{ury} = O_{uly} \\ O_{ex} \geq 0 & O_{ey} \geq 0. \end{array}$$

Global constraints are provided by the user and are defined over the parameters of the relevant objects. For example, the requirement that $T1$ is contained in $R1$ is expressed by

$$\begin{array}{l} T1_{blx} \geq R1_{blx} \\ T1_{bly} \geq R1_{bly} \\ T1_{urx} \leq R1_{urx} \\ T1_{ury} \leq R1_{ury}. \end{array}$$

The entire set of local and global constraints which capture the previous 4 requirements for Figure 1 is shown in Figure 5. The large number of constraints generated by this relatively simple example is typical – systems such as ThingLabII generate similar numbers of constraints for similarly sized examples. The large numbers of constraints means that efficient representation and manipulation of constraints is important.

Anchor constraints are equalities which fix the values of variables of objects. For instance, the constraint

$$B_{blx} = B_{bly} = 0$$

expresses that the lower left corner of B is anchored at the origin of the screen. Anchor constraints are added by the user when the attributes of objects are not to be modified.

When creating this diagram, the user adds and deletes graphic objects and constraints, and modifies the parameters (or *attributes*) of the objects. As constraints and objects are added, the solvability of the entire set must be checked. If it becomes unsatisfiable, information as to which constraints need to be modified to restore solvability is fed back to the user by highlighting a graphical representation of the offending constraints.

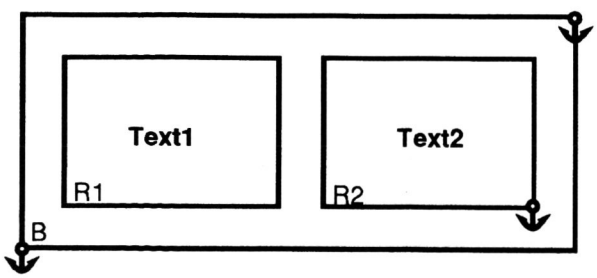

Figure 2: Three anchor points were selected

During a session, as graphic objects are moved around and their attributes modified, the system gives feedback about the current constraints and presents this in a suitable graphical format. For instance, suppose the user wants to move the upper-right corner of $R1$. Initially the vertices of B, one vertex of $R2$ and the text size are "anchored" so that the system will not change their present values (see Figure 2). At this point the set of constraints is over-constrained. When the user selects the upper-right corner of $R1$, the system indicates that $R1$ cannot be moved. This is because the constraints imply that the coordinates of $R1$ are fixed. Note that this information is not explicit in the original constraints.

The system can then indicate which anchor constraints need to be removed to restore some degrees of freedom in the system. If the user now removes the anchor-constraint on $R2$, it is now possible to automatically infer, from the constraints, the possible values for $R1$. The possible values are given by the constraints

$$\begin{array}{l} R1_{urx} - \frac{4}{3} R1_{ury} = 0 \\ 110 \leq R1_{ury} \leq 150 \end{array}$$

defining the line segment as depicted in Figure 3. The system displays this line and the cursor is constrained to remain on it. Whenever the cursor is moved, the display is updated to reflect the new configuration (Figure 4).

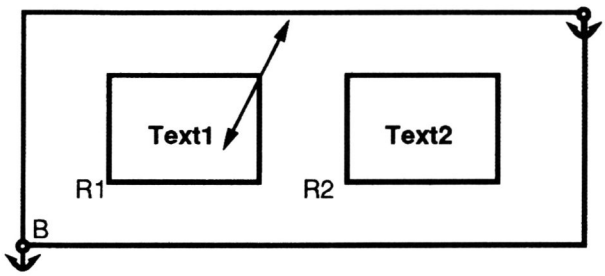

Figure 3: Range of motion for point $R1_{ur}$

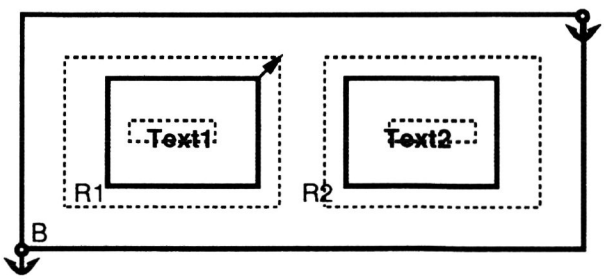

Figure 4: Display reflecting motion of point $R1_{ur}$

3 The Linear Constraint Technology

From the previous example, we can see that an underlying constraint technology for interactive constraint-based systems should provide efficient:

- Incremental addition and deletion of constraints.
- Detection of unsatisfiability and identification of which constraints or anchor constraints are causing it.
- Rapid re-satisfaction of existing constraints when a small number of parameters, such as the location of a vertex, are changed.
- Detection of under-constrained system and identification of which parameters can be fixed to constrain it.
- Recognition of an over-constrained constraint system and identification of the responsible anchor constraints.
- Computation of the range of values that a parameter can take while leaving the system satisfiable.

Recent results from symbolic computation provide a technology for linear arithmetic constraints with these capabilities. The key to an efficient implementation is based on a new representation for sets of linear constraints called the *canonical form* [10], and a new algorithm for variable projection [9]. We now discuss the canonical form, how it supports the compilation of plans of execution or parametric solved forms, and projection.

The Canonical Form

The sets of constraints that arise in interactive systems often contain redundancy. Local constraints defining attributes of objects often contain redundancy to allow flexible definition of objects. Constraints on objects can become redundant as a user adds further constraints. Because of the potentially large number of constraints that can be generated in interactive systems, it is important to have non-redundant representations. In the technology we present, this representation is given by the canonical form.

The *canonical form*[1] of a set of linear arithmetic constraints consists of:

1. A set of equations that defines the affine hull of the solution set of the constraints. This affine hull is the space having the smallest dimension that contains the set of solutions to the constraints.
2. A set of inequalities that define the full dimensional solution set.

The canonical form has the following important properties: it contains no redundant constraints; it identifies the degrees of freedom in the constraints; and it makes explicit the equalities implied by the constraints.

[1]Note: the full definition of the canonical form also includes negative constraints which are not discussed here. The interested reader is referred to the reference for a complete treatment.

Eliminating redundancy is important because typically the system of constraints may contain many redundant constraints mainly due to the local constraints. For some systems of constraints, the corresponding canonical form has an order of magnitude fewer constraints. Making equalities explicit is important because they can greatly simplify the set of constraints. The degrees of freedom of the set of constraints is given by the number of variables less the number of equalities in the canonical form. This means it is possible to determine if the system is under-constrained and which variables need to be further constrained.

Transforming an arbitrary set of constraints into its canonical form is a complex three-stage process using a quasi-dual formulation of optimization techniques (such as the simplex method) from Linear Programming [7, 10]. The three stages are identification of implicit equalities, simplification, and elimination of redundancy. The first stage performs a test for satisfiability. If the system of constraints is unsatisfiable, then a minimal subset of constraints causing unsatisfiability is identified.

Computing the canonical form from scratch is expensive. However, we use an incremental algorithm that efficiently recomputes the canonical form when constraints are added. This is an important consideration when interactively constructing systems of constraints.

The system given in Figure 5 has the canonical form given in Figure 6. The original system has 54 equalities, and 28 inequalities involving 32 variables. The canonical form has 51 equalities, and 10 inequalities involving 9 variables. Note the substantially reduced number of variables in the inequalities.

The Parametric Solved Form

One of the most important operations in an interactive graphics system is the manipulation of objects on the display. To do this efficiently requires computing a plan of execution in terms of the object being manipulated, and then continually re-executing this plan.

In our technology, a plan of execution is called a *parametric solved form with respect to a set of parametric variables*. The parametric solved form is a set of constraints such that all dependent variables are expressed in terms of the parameters, and only the parametric variables occur in the inequalities. The parametric variables are those that correspond to the object being manipulated. Thus the parametric solved form has the property that if the constraints contain the variables $x_1, .., x_n$, and the parameters are $x_1, ..., x_i$, then for $j = i+1, ..., n$, there exists an f_j such that $x_j = f_j(x_1, ..., x_i)$.

The system of constraints corresponding to Figure 4 when parameterized by variable Rl_{ury} has the parametric solved form given in Figure 7. This solved form consists of 59

$T1_{cx} = 0.5 * T1_{blx} + 0.5 * T1_{brx}$
$T1_{cy} = 0.5 * T1_{bly} + 0.5 * T1_{uly}$
$T1_{brx} = T1_{blx} + T1_{ex}$
$T1_{bry} = T1_{bly}$
$T1_{urx} = T1_{brx}$
$T1_{blx} \geq R1_{blx}$
$T1_{urx} \leq R1_{urx}$
$T1_{cx} = R1_{cx}$
$T1_{ex} \geq 0$
$R1_{cx} = 0.5 * R1_{blx} + 0.5 * R1_{brx}$
$R1_{cy} = 0.5 * R1_{bly} + 0.5 * R1_{uly}$
$R1_{brx} = R1_{blx} + R1_{ex}$
$R1_{bry} = R1_{bly}$
$R1_{urx} = R1_{brx}$
$R1_{blx} \geq B_{blx}$
$R1_{urx} \leq B_{urx}$
$R1_{ex} = 2 * R1_{ey}$
$R1_{ex} \geq 0$
$B_{brx} = B_{blx} + B_{ex}$
$B_{ulx} = B_{blx}$
$M_x = R1_{blx} - B_{blx}$
$M_x = B_{brx} - R2_{brx}$
$3 * M_x + R1_{ex} + R2_{ex} - B_{ex} = 0$

$T2_{brx} = T2_{blx} + T2_{ex}$
$T1_{ulx} = T1_{blx}$
$T1_{ury} = T1_{uly}$
$T1_{bly} \geq R1_{bly}$
$T1_{ury} \leq R1_{ury}$
$T1_{cy} = R1_{cy}$
$T1_{ey} \geq 0$

$R2_{brx} = R2_{blx} + R2_{ex}$
$R1_{ulx} = R1_{blx}$
$R1_{ury} = R1_{uly}$
$R1_{bly} \geq B_{bly}$
$R1_{ury} \leq B_{ury}$
$R2_{ex} = 2 * R2_{ey}$
$R1_{ey} \geq 0$
$B_{bry} = B_{bly}$
$B_{uly} = B_{bly} + B_{ey}$
$M_x = R2_{blx} - R1_{brx}$
$M_y = R2_{bly} - B_{bly}$

$T2_{cx} = 0.5 * T2_{blx} + 0.5 * T2_{brx}$
$T2_{cy} = 0.5 * T2_{bly} + 0.5 * T2_{uly}$
$T1_{uly} = T1_{bly} + T1_{ey}$
$T2_{bry} = T2_{bly}$
$T2_{urx} = T2_{brx}$
$T2_{blx} \geq R2_{blx}$
$T2_{urx} \leq R2_{urx}$
$T2_{cx} = R2_{cx}$
$T2_{ex} \geq 0$
$R2_{cx} = 0.5 * R2_{blx} + 0.5 * R2_{brx}$
$R2_{cy} = 0.5 * R2_{bly} + 0.5 * R2_{uly}$
$R1_{uly} = R1_{bly} + R1_{ey}$
$R2_{bry} = R2_{bly}$
$R2_{urx} = R2_{brx}$
$R2_{blx} \geq B_{blx}$
$R2_{urx} \leq B_{urx}$

$R2_{ex} \geq 0$
$B_{urx} = B_{brx}$
$B_{ex} \geq 0$
$M_y = R1_{bly} - B_{bly}$
$M_y = B_{uly} - R2_{uly}$
$2 * M_y + R1_{ey} - B_{ey} = 0$

$T2_{uly} = T2_{bly} + T2_{ey}$
$T2_{ulx} = T2_{blx}$
$T2_{ury} = T2_{uly}$
$T2_{bly} \geq R2_{bly}$
$T2_{ury} \leq R2_{ury}$
$T2_{cy} = R2_{cy}$
$T2_{ey} \geq 0$

$R2_{uly} = R2_{bly} + R2_{ey}$
$R2_{ulx} = R2_{blx}$
$R2_{ury} = R2_{uly}$
$R2_{bly} \geq B_{bly}$
$R2_{ury} \leq B_{ury}$

$R2_{ey} \geq 0$
$B_{ury} = B_{uly}$
$B_{ey} \geq 0$
$M_y = B_{uly} - R1_{uly}$
$M_x \geq 0, M_y \geq 0$
$R1_{ey} - R2_{ey} = 0$

Figure 5: Original Set of 82 Constraints

equations and the two inequalities

$$110 \leq R1_{ury} \qquad R1_{ury} \leq 150.$$

One can readily see how all variables are either assigned to constants or are expressed solely in terms of $R1_{ury}$. Thus, only one degree of freedom remains in the solved form.

The parametric solved form is efficiently derived from the canonical form (see Section 5). It can then be either interpreted directly or used to compile efficient code to compute the values of dependent variables as the values of parameters are changed.

Projection

Projection provides a technique to examine the relationships between particular variables that are implied by the constraints. Projecting onto a single variable gives the range of values it can take while still satisfying the constraints. Projecting a system of constraints onto a set of variables shows how these variables are inter-related in any solution to the constraints.

In the example in Section 2, we saw how projecting the constraints on to variables R_{urx}, R_{ury} indicated that they were constrained to be on the line $R1_{urx} - \frac{4}{3}R1_{ury} = 0$ where $110 \leq R1_{ury} \leq 150$.

Unfortunately, the doubly-exponential complexity of general algorithms for projection has prevented its use in many application domains. Recently, however, a new algorithm has been developed [9] that is very efficient when the number of variables in the projection space is small. This is exactly the case we are interested in, as we typically project onto a small number of variables (typically one or two) corresponding to the object currently being manipulated.

The new algorithm computes a projection by successive approximations using an on-line algorithm for convex hull construction in the projection space. It provides an exact solution when the size of the output is small, and an approximation (upper or lower) when the size of the output is unmanageable. Previous methods usually failed to produce any output, even in small cases, because of the enormous amount of intermediate computation. Initial testing has shown extremely good performance, especially for small projection spaces.

4 Proposed Architecture

In this section we propose an architecture for a constraint manipulation sub-system within an interactive constraint-based user-interface system. The architecture exploits the technology described in the previous section. This architecture must address issues of incrementality, interaction latency and feedback bandwidth [11]. These issues are most critical when anchor constraints and the values of the parameters of objects are changed during manipulation.

To address these issues, we use a two level architecture which maintains sets of constraints in canonical form. The first, the *free canonical form*(FCF), is a canonical form of the local and global constraints. The second, the *anchored canonical form*(ACF), is a canonical form for the entire set of local, global and anchor constraints. In addition, local constraints associated with primitive and compound objects are kept in canonical form.

The FCF does not change often, only when local or global constraints are added or deleted. However, whenever an

$$
\begin{aligned}
&T1_{cx} = B_{ulx} + B_{ey} + M_x - 2M_y \\
&T1_{blx} = -T1_{urx} + 2B_{ulx} + 2B_{ey} + 2M_x - 4M_y \\
&T1_{brx} = T1_{urx} \\
&T1_{ex} = 2T1_{urx} - 2B_{ulx} - 2B_{ey} - 2M_x + 4M_y \\
&T1_{ulx} = -T1_{urx} + 2B_{ulx} + 2B_{ey} + 2M_x - 4M_y \\
&T2_{cx} = B_{ulx} + 3B_{ey} + 2M_x - 6M_y \\
&T2_{blx} = -T2_{urx} + 2B_{ulx} + 6B_{ey} + 4M_x - 12M_y \\
&T2_{brx} = T2_{urx} \\
&T2_{ex} = 2T2_{urx} - 2B_{ulx} - 6B_{ey} - 4M_x + 12M_y \\
&T2_{ulx} = -T2_{urx} + 2B_{ulx} + 6B_{ey} + 4M_x - 12M_y \\
&R1_{blx} = B_{ulx} + M_x \\
&R1_{cx} = B_{ulx} + B_{ey} + M_x - 2M_y \\
&R1_{brx} = B_{ulx} + 2B_{ey} + M_x - 4M_y \\
&R1_{urx} = B_{ulx} + 2B_{ey} + M_x - 4M_y \\
&R1_{ex} = 2B_{ey} - 4M_y \\
&R2_{bly} = B_{uly} - B_{ey} + M_y \\
&R2_{cy} = B_{uly} - 0.5B_{ey} \\
&R2_{uly} = B_{uly} - M_y \\
&R2_{ury} = B_{uly} - M_y \\
&B_{blx} = B_{ulx} \\
&B_{brx} = B_{ulx} + 4B_{ey} + 3M_x - 8M_y \\
&B_{urx} = B_{ulx} + 4B_{ey} + 3M_x - 8M_y \\
&R1_{ulx} = B_{ulx} + M_x \\
&R2_{ulx} = B_{ulx} + 2B_{ey} + 2M_x - 4M_y \\
&R2_{ey} = B_{ey} - 2M_y \\
&B_{ex} = 4B_{ey} + 3M_x - 8M_y
\end{aligned}
$$

$$
\begin{aligned}
&T1_{cy} = B_{uly} - 0.5B_{ey} \\
&T1_{bly} = -T1_{ury} + 2B_{uly} - B_{ey} \\
&T1_{uly} = T1_{ury} \\
&T1_{bry} = -T1_{ury} + 2B_{uly} - B_{ey} \\
&T1_{ey} = 2T1_{ury} - 2B_{uly} + B_{ey} \\
&T2_{cy} = B_{uly} - 0.5B_{ey} \\
&T2_{bly} = -T2_{ury} + 2B_{uly} - B_{ey} \\
&T2_{uly} = T2_{ury} \\
&T2_{bry} = -T2_{ury} + 2B_{uly} - B_{ey} \\
&T2_{ey} = 2T2_{ury} - 2B_{uly} + B_{ey} \\
&R1_{bly} = B_{uly} - B_{ey} + M_y \\
&R1_{cy} = B_{uly} - 0.5B_{ey} \\
&R1_{uly} = B_{uly} - M_y \\
&R1_{ury} = B_{uly} - M_y \\
&R2_{blx} = B_{ulx} + 2B_{ey} + 2M_x - 4M_y \\
&R2_{cx} = B_{ulx} + 3B_{ey} + 2M_x - 6M_y \\
&R2_{brx} = B_{ulx} + 4B_{ey} + 2M_x - 8M_y \\
&R2_{urx} = B_{ulx} + 4B_{ey} + 2M_x - 8M_y \\
&R2_{ex} = 2B_{ey} - 4M_y \\
&B_{bly} = B_{uly} - B_{ey} \\
&B_{bry} = B_{uly} - B_{ey} \\
&B_{ury} = B_{uly} \\
&R1_{ey} = B_{ey} - 2M_y \\
&R1_{bry} = B_{uly} - B_{ey} + M_y \\
&R2_{bry} = B_{uly} - B_{ey} + M_y
\end{aligned}
$$

$$
\begin{aligned}
&M_y + T1_{ury} - B_{uly} \leq 0 \\
&-B_{ey} - 2T1_{ury} + 2B_{uly} \leq 0 \\
&M_y - B_{uly} + T2_{ury} \leq 0 \\
&-B_{ey} + 2B_{uly} - 2T2_{ury} \leq 0 \\
&-M_x \leq 0
\end{aligned}
$$

$$
\begin{aligned}
&-2T1_{urx} + 2B_{ulx} + 2B_{ey} + 2M_x - 4M_y \leq 0 \\
&-B_{ulx} - 4B_{ey} - 2M_x + 8M_y + T2_{urx} \leq 0 \\
&2B_{ulx} + 6B_{ey} + 4M_x - 12M_y - 2T2_{urx} \leq 0 \\
&T1_{urx} - B_{ulx} - 2B_{ey} - M_x + 4M_y \leq 0 \\
&-M_y \leq 0
\end{aligned}
$$

Figure 6: Canonical Form of Original Set – 61 constraints

anchor constraint is changed, or the picture is manipulated via a new object, a new ACF is incrementally computed from the current FCF. Obviously – it is better not to redo the work already done in finding redundancy and implicit equalities. The role of the ACF is to be a parametric solved form with respect to the variables in the object currently being manipulated. This allows efficient updating of the display when the parameters are changed.

When objects or global constraints are added to the system new canonical forms are computed from the old one. Firstly, an FCF for the new local and global constraints is computed incrementally from the old FCF. Then, the ACF for the entire system of constraints is computed incrementally from the new FCF by adding the anchor constraints. This is quite efficient because the anchor constraints are always equations. Computation of these canonical forms will reveal if the system is unsatisfiable, and if so which constraints are at fault.

Deletion of objects or global constraints is more problematic. However, because local constraints associated with each object are kept in canonical form, then a new FCF and subsequently a new ACF can be computed relatively quickly. Thus the architecture provides:

- Incremental addition and deletion of constraints.
- Detection of unsatisfiability and identification of which constraints are the cause.

When an object is selected for manipulation, the ACF is examined to see if it is in parametric solved form with respect to the variables in the object. There are three cases to consider: the ACF is under-constrained, over-constrained, or it is in parametric solved form.

Examination of the ACF will reveal the system is over-constrained if there are no degrees of freedom, that is all variables are uniquely determined. For instance, consider the anchor constraints in Figure 2. These constraints, as follows,

$$
\begin{array}{lll}
B_{llx} = B_{lly} = 0 & B_{urx} = 400 & B_{ury} = 200 \\
T1_{ex} = 30 & T1_{ey} = 10 & T2_{ex} = 40 \\
T2_{ey} = 20 & R2_{lrx} = 300 & R2_{lry} = 87.5
\end{array}
$$

will fix the width and height of the boxes containing text. Before adding these anchor constraints, the FCF is that shown in Figure 6. When the anchor constraints are added, we derive from the FCF, the following very simple ACF with 60 equations that assign each variable to a constant:

$$T1_{cy} = 100, \quad T1_{blx} = 110, \quad T1_{bly} = 95, \ldots$$

As every variable is uniquely determined, there are no remaining degrees of freedom. Thus, with these anchor

$$
\begin{array}{ll}
T1_{cx} = 200 - \frac{2}{3}R1_{ury} & T1_{cy} = 100 \\
T1_{blx} = 185 - \frac{2}{3}R1_{ury} & T1_{bly} = 95 \\
T1_{brx} = 215 - \frac{2}{3}R1_{ury} & T1_{ury} = 105 \\
T1_{ulx} = 185 - \frac{2}{3}R1_{ury} & T1_{bry} = 95 \\
T1_{ex} = 30 & T1_{ey} = 10 \\
T2_{cx} = 200 + \frac{2}{3}R1_{ury} & T2_{cy} = 100 \\
T2_{blx} = 180 + \frac{2}{3}R1_{ury} & T2_{bly} = 90 \\
T2_{brx} = 220 + \frac{2}{3}R1_{ury} & T2_{uly} = 110 \\
T2_{ulx} = 180 + \frac{2}{3}R1_{ury} & T2_{bry} = 90 \\
T2_{ex} = 40 & T2_{ey} = 20 \\
R1_{cx} = 200 - \frac{2}{3}R1_{ury} & R1_{cy} = 100 \\
R1_{blx} = 400 - \frac{8}{3}R1_{ury} & R1_{bly} = 200 - R1_{ury} \\
R1_{brx} = \frac{4}{3}R1_{ury} & R1_{bry} = 200 - R1_{ury} \\
R1_{ulx} = 400 - \frac{8}{3}R1_{ury} & R1_{uly} = -400 + 4R1_{ury} \\
R1_{ex} = -400 + 4R1_{ury} & R1_{ey} = -200 + 2R1_{ury} \\
R2_{blx} = 400 - \frac{4}{3}R1_{ury} & R2_{bly} = 200 - R1_{ury} \\
R2_{cx} = 200 - \frac{2}{3}R1_{ury} & R2_{cy} = 100 \\
R2_{brx} = \frac{8}{3}R1_{ury} & R2_{bry} = 200 - R1_{ury} \\
R2_{urx} = \frac{8}{3}R1_{ury} & R2_{ury} = R1_{ury} \\
R2_{ex} = -400 + 4R1_{ury} & R2_{ey} = 200 - R1_{ury} \\
B_{blx} = 0 & B_{bly} = 0 \\
B_{brx} = 400 & B_{bry} = 0 \\
B_{urx} = 400 & B_{ury} = 200 \\
B_{ulx} = 0 & B_{uly} = 200 \\
B_{ex} = 400 & B_{ey} = 200 \\
T1_{urx} = 400 - \frac{8}{3}R1_{ury} & T1_{ury} = 105 \\
T2_{urx} = 220 - \frac{2}{3}R1_{ury} & T2_{ury} = 110 \\
M_x = 400 - \frac{8}{3}R1_{ury} & M_y = 200 - R1_{ury} \\
R1_{urx} = \frac{4}{3}R1_{ury} & \\
110 \leq R1_{ury} & R1_{ury} \leq 150
\end{array}
$$

Figure 7: Paremetric Solved form in R_{ury} – 61 constraints

constraints the system is over-constrained – it only has one solution. Note that this was certainly not obvious in the original set of constraints in non-canonical form.

At this point we need to determine which anchor constraints need to be removed. To do this, a set of inequations are added corresponding to the variables in the object that we wish to manipulate. This of course results in an unsatisfiable system of constraints, but, in computing the new FCF, the unsatisfiable system can be analyzed to find other constraints, other than those we just added, which cause the unsatisfiability. These can be then removed by the user.

If we consider only two anchor constraints instead of three, as in Figure 3, the ACF, as shown in Figure 7, is in parametric solved form having parameter $R1_{ury}$. This solved form consists of 59 equations and 2 inequalities,

$$110 \leq R1_{ury} \qquad R1_{ury} \leq 150 \qquad (1)$$

and clearly has only one degree of freedom. This solved form can now be compiled into code to update the dependent variables whenever this parameter is changed.

If the ACF is not in solved form, then the constraint system is under-constrained. From the ACF it is straightforward to determine possible choices for additional anchor constraints. These are variables which are not uniquely determined by the desired parametric variables of the ACF.

Thus the architecture provides:

- Rapid re-satisfaction of existing constraints when a small number of parameters, such as location of a vertex, are changed.
- Detection of under-constrained system and identification of which parameters can be fixed to constrain it.
- Recognition of an over-constrained system and identification of anchor constraints responsible for the unsatisfiability.

When an object is selected for manipulation, the ACF is projected onto the object's variables giving the ranges of values that they may take while still satisfying the constraints. As the constraints are linear, the range will always be a convex polygon allowing it to be easily displayed. In the example, we saw that the upper right corner of $R1$ was constrained to move only on a diagonal line. In other situations it might be constrained to move inside a small region. In both cases it is possible to present these regions graphically to the user.

Thus the architecture provides:

- Computation of the range of values that a parameter can take and still leave the system satisfiable.

Interactive editors must support the definition and compilation of compound objects. Although this is straightforward in systems without constraints, the addition of constraints raises new issues. In particular, efficiently representing the constraints in the compound object, and determining which variables of a compound graphic object define the objects it contains.

An efficient representation for the constraints of a compound object can be obtained by computing the canonical form for all the constraints inside it.

Defining a compound object is more problematic. Intuitively, the user should be able to select a group of objects on the display, and then the editor should abstract the desired definition from this instance of the definition. Essentially this can be done as follows. Once the user has selected a set of objects to form the compound object, selected anchor constraints must then be removed to permit the compound object to be located freely, and finally a set of *definitional* variables which ensure that all its internal objects are well-defined – none of their variables are left unspecified – must be selected. The last step can be done by ensuring that the constraints in the compound object are parametric solved form for these definitional variables.

Projection can be used to simplify the constraints associated with the definition, acting as a type of compilation or partial evaluation. The idea is to project the constraints onto the definitional variables of the new object: this will remove local or internal variables from the constraints, and so simplify them.

For example suppose we were constructing a square compound object from the rectangle definition given earlier. The square object has the additional constraints that the x and y extents are equal. If we allow the square to be defined in terms of a diagonal and a point, then the square's local constraints are simpler than those of the rectangle.

Thus the architecture supports:

- The definition and compilation of compound objects.

5 Empirical results

We now present empirical results concerning the performance of this constraint technology. We consider the following operations which are typical of those which occur during an interactive session:

1. Testing the solvability of a set of constraints and computing their canonical form.
2. Adding new constraints or anchor points, and computing a new canonical form.
3. Generating a parametric solved form in terms of a set of specific variables that correspond to a point which is being dragged.

All these operations are handled by a new incremental constraint solving system we are developing. This system integrates algorithms for: testing solvability; computing the canonical form; performing Gauss-Jordan reduction to obtain parametric solved forms; and performing projections. This system is implemented in C++ on an IBM Risc System/6000, model 530 running AIX. Run times are measured in virtual CPU seconds.

In this evaluation, we use three sets of constraints as test data and time the operations described above. The results are shown in Figure 8.

The first test corresponds to the well known example of recursively nested quadrilaterals (see Figure 9). The initial set of 76 constraints consist of constraints that ensure that the endpoints of lines in the quadrilaterals touch, that the midpoints of the quadrilaterals form a parallelogram, and that the vertices of the embedded quadrilaterals are at the midpoints of the edges of the enclosing quadrilateral. It takes 0.06 seconds to transform this system into its canonical form which contains 64 constraints. (Remark: the twelve constraints that are eliminated are exactly those that constrain the midpoints

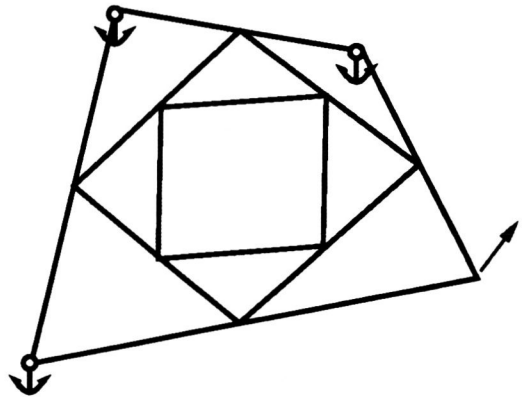

Figure 9: Recursively Nested Quadrilaterals

of each quadrilateral's edges to form a parallelogram. In retrospect, this was to be expected. The geometry theorem that this example illustrates states this fact!) To this system, we then add the three anchor points, consisting of 6 equality constraints, shown in Fig. 9. It takes 0.11 seconds to derive a new canonical form. Lastly, the time taken to obtain the parametric solved form in terms of the unanchored vertex of the quadrilateral is below the resolution of our timing system (1/100th of a second).

The second test corresponds to that given in section 3. The initial set of constraints (see Figure 5) consists of 82 constraints (54 equalities, 28 inequalities) over 60 variables. It takes 0.80 seconds to convert this to its canonical form (Fig. 6) containing 65 constraints. We then add 2 anchor points, consisting of 4 equalities. It takes 0.17 seconds to compute a new canonical form. Finally, the parametric solved form in terms of the point (Rl_{urx}, Rl_{ury}) takes 0.01 seconds to compute.

The third test, although from outside the graphics domain, has similar characteristics (sparsity, relative percentage of equalities and inequalities, etc..) as the previous examples. It is of interest because the initial constraints are all inequalities. The initial set of constraints consist of 1819 inequalities over 68 variables. This is simplified into 58 equalities plus 90 inequalities over 10 variables. Then, 10 new equalities are added and the set is updated accordingly. Finally, 10 variables are chosen to obtain a parameterized representation.

In these examples, the advantage of keeping constraints in canonical form is demonstrated by how little time it takes to add constraints, and to compute the parametric solved forms. The most time consuming operation is the detection of equalities implicitly defined by inequalities in the original system. This can be significant when there are large number of inequalities. However because the system is incremental, it will rarely, if ever, have to deal with the size and complexity of dataset 3 in an interactive application. Adding large numbers of inequalities at once

Total Constraints N (=,<)	Computing Canonical Form Constraints (=,<)	(sec)	Addition (sec)	Solved Form (sec)
Recursive Quads 76 (76,0) : 72 var.	64 (64,0)	0.06	0.11	<0.01
Layout: 82 (54,28) : 60 var.	65 (51,14)	0.80	0.17	0.01
Dataset 3: 1819 (0,1819) : 68 var.	148 (58,90)	17.96	0.28	0.01

Figure 8:

is unlikely.

6 Conclusion

We have shown how recent results and algorithms to manipulate linear arithmetic constraints provide a technology for interactive constraint-based user-interface systems. This extends previous constraint technology for user interfaces, by allowing simultaneous linear equations and inequalities, and by providing techniques which give improved user-feedback.

We have proposed an architecture based on this technology and are currently implementing a interactive constraint-based editor based upon it.

Acknowledgements

The authors thank Jean-Louis Lassez for his helpful comments.

References

[1] A. Borning, The programming language aspects of ThingLab – a constraint-oriented simulation laboratory, *ACM Trans. on Prog. Lang. and Systems* **3**, 1981, 343-387.

[2] A. Borning, R.A. Duisberg. Constraint based tools for building user interfaces. *ACM Transactions on Graphics*. **5**(4). 1986.

[3] D. Epstein and W.R. Lalonde, A Smalltalk window system based on constraints, in *Proc. of ACM Conference on Object Oriented Programming Systems, Languages and Applications (OOPSLA'88)*, pp. 83–94, ACM Press, 1988.

[4] R. Helm, K. Marriott and M. Odersky, Building visual language parsers, in *ACM CHI'91*, pp. 105–112, ACM Press, 1991.

[5] R. Helm and K. Marriott, Declarative specification of visual languages, in *1990 IEEE Workshop on Visual Languages*, pp. 98–103, IEEE Comp. Soc. Press, 1990.

[6] T. Huynh, C. Lassez and J-L. Lassez, *Fourier Algorithm Revisited*, 2nd International Conference on Algebraic and Logic Programming, Lecture Notes in Computer Sciences, Springer-Verlag 1990.

[7] T. Huynh, J-L. Lassez and K. McAloon, *Simplification and Elimination of Redundant Linear Arithmetic Constraints*, proc. NACLP 89, MIT Press.

[8] J-L. Lassez, *Querying Constraints*, Proceedings of the ACM conference on Principles of Database Systems Nashville 1990.

[9] C. Lassez and J-L. Lassez, *Quantifier Elimination for Conjunctions of Linear Constraints via a Convex hull Algorithm*, IBM Research Report. RC 16779. 1991.

[10] J-L. Lassez and K. McAloon, *A Canonical Form for Generalized Linear Constraints*, IBM Research Report RC 15004, IBM T.J. Watson Research Center, to appear Journal of Symbolic Computation.

[11] J.H. Maloney, A. Borning and B.N. Freeman-Benson, Constraint technology for user-interface construction in Thinglab II, *OOPSLA '89*, pp. 381–388, ACM Press, 1989.

[12] G. Nelson, Juno: a constraint-based graphics system, in *ACM SIGGRAPH' 85 Conf. Proc.*, pp. 235-243, ACM Press, 1985.

[13] D.R. Olson Jr., K. Allan, Creating Interactive Techniques by Symbolically Solving Geometric Constraints. *Proc. of Third Symp. on User Interface Software and Technology* Snowbird, Utah. October 1990. pp. 102-107.

[14] I.E. Sutherland, Sketchpad: A man-machine graphical communication system, in *Proc. of the Spring Joint Computer Conference*, pp. 329-345, 1963.

[15] P.A. Szekely and B.A. Myers, A user interface toolkit based on graphical objects and constraints, in *OOPSLA'88*, pp. 36–45, ACM Press, 1988.

[16] A. Witkin, M. Gleicher, W. Welch. Interactive Dynamics. in *Proc. of SIGGRAPH'90.* 1990

[17] C.J. Van Wyk, A high-level language for specifying pictures, *ACM Transactions on Graphics* **2**, 1982, 163-182.