Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1998), *12*, 13–27. Printed in the USA.
Copyright © 1998 Cambridge University Press 0890-0604/98 $12.50

Constraint solving and preference activation for interactive design

CLAUDIO LOTTAZ,[1] RUTH STALKER,[2] AND IAN SMITH[2]

[1]AI Lab (LIA), Computer Science Department and
[2]Institute of Structural Engineering and Mechanics (ISS-IMAC), Department of Civil Engineering, Swiss Federal Institute of Technology (EPFL), CH-1015 Lausanne, Switzerland

(RECEIVED April 28, 1997; REVISED October 31, 1997; ACCEPTED November 3, 1997)

Abstract

This article contains a description of a constraint solver that determines complete solution spaces. These spaces are defined by sets of constraints in continuous variables. Interactive design is supported through improvements to existing algorithms that have increased system performance. In addition, heuristics for activating the best set of preferred constraints for a design task are presented, and two ways are proposed for interactively exploring design alternatives. IDIOM is an application framework that has successfully tested these algorithms for interactive apartment layout design.

Keywords: Constraint Solving; Preferences; Interactive Design

1. INTRODUCTION

Computer aided design (CAD) systems have the potential to support designers, particularly during revision of drawings. Although iterative design using these systems is more convenient than manual modification, preserving consistency remains a difficult task. Indeed, traditional CAD systems do not provide sufficient support for maintaining design consistency.

The knowledge involved in a design task originates from sources such as structural requirements, laws, guidelines, and personal preferences. Constraints on geometrical parameters of designs can be used to formalize parts of this knowledge. This is an approach that has already been applied to spatial configuration (Mitchell et al., 1976; Baykan & Fox, 1992; Flemming et al., 1992; Tommelein et al., 1992; Bakhtari et al., 1993; Medjdoub & Yannou, 1996), case-based design (Maher & Zhang, 1991; Bakhtari et al., 1993; Giretti et al., 1994; Purvis & Pu, 1995), and image understanding (Hummel & Zucker, 1983). However, not all knowledge can be transformed into mathematical forms. For example, social and political issues often depend on many contextual aspects and therefore, complete models are not possible. As a result, research into providing facilities for adaptation, combination, and exploration of design solutions have been undertaken (Navinchandra, 1991; Logan & Smithers, 1993; Dave et al., 1994). Design tools need to provide facilities to explore feasible design alternatives because the designer must ultimately decide which solution is the best for a specific task.

A CAD tool that uses constraints to represent knowledge and allows for the interactive exploration of solutions requires:

- globally consistent and complete solution spaces;
- fast algorithms for interactive use; and
- facilities for interactive adaptation of solutions.

To satisfy the first two requirements, we focus upon linear constraints and we have developed techniques for approximating nonlinear relationships. For linear systems of constraints, linear programming algorithms such as simplex (Schrijver, 1986) have been developed and applied to spatial layout. However, they require linear objective functions. Such objective functions are rarely able to model domain knowledge completely and furthermore, they cannot support interactive adaptation. Systems using nonlinear programming methods provide a more powerful means to specify objectives but still cannot support interactive adaptation (Mitchell et al., 1976; Liggett, 1985). Other approaches using local consistency algorithms (Maher & Zhang, 1991; Bay-

Reprint requests to: Claudio Lottaz, AI Laboratory, Computer Science Department (LIA-DI), Swiss Federal Institute of Technology at Lausanne (EPFL), IN (Ecublens), CH-1015 Lausanne, Switzerland. Phone: +41 (0)21 693 66 82; Fax: +41 (0)21 693 52 25; E-mail: lottaz@lia.di.epfl.ch.

https://doi.org/10.1017/S0890060498121133 Published online by Cambridge University Press

kan & Fox, 1992) are unreliable because they may supply globally inconsistent solutions (Sam-Haroud, 1995).

Moreover, most of the currently proposed systems for spatial layout cannot accommodate continuous variables. Instead, they are limited to discretized parameters (Giretti et al., 1994; Voss, 1994; Medjdoub & Yannou, 1996). This is partially understandable because until recently, designers used standard components having standard sizes to save costs. However, today this is no longer adequate because modern computer-integrated manufacturing techniques allow the production of custom-sized parts at no extra cost and therefore, standard components no longer provide an economic advantage. This has lead to a trend away from grid-based approaches such as those proposed in Gross (1990), Bakhtari et al. (1993), and Giretti et al. (1994). Recent testing with practicing designers has confirmed this trend (Smith et al., 1996).

Although hard constraints are appropriate for structural requirements and laws, they are not useful for representing knowledge such as guidelines that should be followed but may be disregarded if the design is overconstrained. In such instances we use preferences, that is, constraints that are deactivated when they are in conflict with other requirements. Although preferences can be deactivated, they are not lost; they are kept in memory and reactivated when possible. Similar approaches have been proposed that use assumption-based truth maintenance (Logan et al., 1991) for discrete variables. Borning (1992) used hierarchies of constraint sets (equalities only) to resolve contradictions in an interactive drawing system. Preferences have also been used for complex Pareto optimality problems (D'Ambrosio & Birmingham, 1995). In WRIGHT, Baykan and Fox (1992) allow for constraint weakening in overconstrained situations and Fox (1987) discussed relaxation of constraints in scheduling. PRIDE is another system that incorporates relaxable constraints but its authors recognize the difficulty to automatically determine which constraints should be relaxed (Mittal et al., 1986). IDIOM is a case-based application framework for interactive spatial configuration of rectangular spaces, using constraint solving, preference activation, and domain models to provide active design support (Smith et al., 1996). Its name is an acronym for Interactive Design using Intelligent Objects and Models. The system uses linear constraints and piecewise linear, convex approximations of nonlinear constraints. Disjunctive constraints can be handled through a collection of preferences.

Apartment design is used to test and validate the system. Using IDIOM, a designer incrementally chooses objects such as rooms from a case library and interactively composes them within a new site. Constraints and preferences on the new apartment are thereby activated according to the new design context and designer preferences. Moreover, a designer may change values of continuous parameters as well as priorities on preferences. Throughout the design process, IDIOM maintains design consistency.

This article focuses on the algorithmic aspects of IDIOM. Section 2 contains a description of the algorithms used for constraint solving. The algorithm that was implemented to determine the set of preferences to be activated is presented in Section 3 and the exploration of design alternatives is shown in Section 4. Detailed technical reports about IDIOM are available in Stalker (1996) and Lottaz (1996).

2. FINDING SOLUTION SPACES

IDIOM is based on two algebraic methods for solving imposed constraint sets. Dimensionality reduction (Hua, 1994) solves systems of equalities and prepares inequalities for Fourier–Motzkin elimination (Fourier, 1970; Schrijver, 1986), which then provides the solution spaces. Although these algorithms find sound solutions, Fourier–Motzkin elimination is often not fast enough for interactive applications. Through reducing redundancy of generated constraints (Section 2.2.2) and through an efficient strategy to eliminate the variables (Section 2.2.3) performance is greatly improved.

2.1. Dimensionality reduction

Equalities in the constraint set reduce the degrees of freedom of design spaces. This approach has been used in statistics (Krishnaiah & Kanal, 1982) as well as in image recognition (Saund, 1989) and was proposed for case-based design in Faltings (1990). Further research demonstrated that equalities can be used to reduce the number of variables occurring in inequalities (Hua, 1994; Hua et al., 1996).

IDIOM builds on this work through using Gauss–Jordan elimination to perform dimensionality reduction and to identify dependent and independent variables. Dependent variables are substituted by independent variables as illustrated in the following example:

Figure 1 shows a small example of an arrangement of two rectangular rooms. The following constraints are present, all dimensions are given in cm:

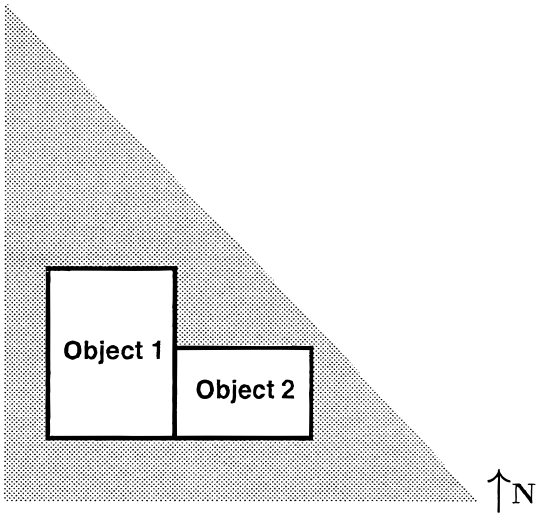


Fig. 1. Simple example for a design, two neighboring objects within a triangular site.

https://doi.org/10.1017/S0890060498121133 Published online by Cambridge University Press

- both rooms have a minimum width and length of 200 cm;
- the width of the second room is fixed to 250 cm;
- a neighborhood relationship defines how the rooms are joined, aligns the south walls, and maintains the north wall of object 1 more to the north than the north wall of object 2; and
- the rooms must stay within a triangular site with corners at (300, 500), (1300, 500) and (300, 1500).

The constraint system contains eight variables. (x_i^{pos}, y_i^{pos}) is the position of the ith room's south-west corner, x_i^{size} and y_i^{size} give the size of room i. This leads to the following equations:

$$y_2^{size} = 250 \tag{1}$$

$$x_1^{pos} + x_1^{size} = x_2^{pos}. \tag{2}$$

$$y_1^{pos} = y_2^{pos}. \tag{3}$$

Equation (1) comes from the constraint that fixes the width of object 2, while Eqs. (2) and (3) are introduced by the neighborhood relationship. The following inequalities are generated in this example:

$$\begin{gathered} x_1^{size} \geq 200 \\ y_1^{size} \geq 200 \\ x_2^{size} \geq 200 \\ y_2^{size} \geq 200. \end{gathered} \tag{4}$$

$$\begin{gathered} x_1^{pos} \geq 300 \\ y_1^{pos} \geq 500 \\ x_2^{pos} \geq 300 \\ y_2^{pos} \geq 500. \end{gathered} \tag{5}$$

$$x_1^{pos} + y_1^{pos} + x_1^{size} + y_1^{size} = 1760$$

$$x_2^{pos} + y_2^{pos} + x_2^{size} + y_2^{size} = 1760$$

$$y_1^{pos} + y_1^{size} \geq y_2^{pos} + y_2^{size}. \tag{6}$$

The inequalities (4) are the constraints requiring that the two rooms have a minimal length and width. Inequalities (5) express that the rooms remain on the site and inequality (6) is due to the neighborhood relationship that maintains the north wall of object 1 to be further north than the north wall of object 2.

The result of the Gauss–Jordan elimination for the equalities in the example is given below:

$$\begin{array}{cccccc} x_2^{pos} & -x_1^{pos} & -x_1^{size} & = & 0 \\ y_2^{pos} & -y_1^{pos} & & = & 0 \\ y_2^{size} & & & = & 250. \end{array} \tag{7}$$

The second step of the dimensionality reduction then substitutes variables x_2^{pos}, y_2^{pos}, and y_2^{size} into all inequalities. The result of this substitution is expressed in terms of inequalities (8).

$$\begin{array}{ccccccc} -x_2^{size} & & & & & \leq & -200 \\ x_2^{size} & +x_1^{pos} & +y_1^{pos} & +x_1^{size} & & \leq & 1510 \\ & -x_1^{pos} & & & & \leq & -300 \\ & -x_1^{pos} & & -x_1^{size} & & \leq & -300 \\ & x_1^{pos} & +y_1^{pos} & +x_1^{size} & +y_1^{size} & \leq & 1760 \\ & & -y_1^{pos} & & & \leq & -500 \\ & & -y_1^{pos} & & & \leq & -500 \\ & & & -x_1^{size} & & \leq & -200 \\ & & & & -y_1^{size} & \leq & -200 \\ & & & & -y_1^{size} & \leq & -250. \end{array} \tag{8}$$

Inequalities (8) no longer involve the eight variables given in (4), (5), and (6); only five parameters are present. Furthermore, one constraint is transformed to $0 \leq 50$ and then removed from the system since it is always satisfied.

Gauss-Jordan elimination has been proved to be a polynomial time method for exact calculus (Schrijver, 1986), while for floating-point arithmetic its complexity is $O(n^3)$. In IDIOM, the algorithm is implemented using sparse matrices, thus improving efficiency significantly because more than 90% of the coefficients in typical problems for IDIOM are zero.

2.2. Treatment of inequalities

After dimensionality reduction has been performed, a system of inequalities remains to be solved. In order to find valid values for parameters, Recursive transformation (RT) of all unsatisfied inequalities into equalities is proposed in a previous system (Hua, 1994). This method may omit correct solutions as shown in Smith et al. (1996). IDIOM avoids this by using the Fourier-Motzkin elimination method which is an algebraic algorithm for solving inequality-systems (Motzkin, 1936; Fourier, 1970; Schrijver, 1986).

The method suggested by Fourier eliminates variables one by one, keeping all information in the newly generated inequality set, so that each solution of the new set can be extended to a solution of the original set. Geometrically, this is equivalent to projecting the original set along the axis of the variable being eliminated. Elimination of all but one variable leads to a simple set of inequalities that contains only one variable and can be used to determine the whole interval of values for the remaining variable that can be extended to a solution for the whole system.

https://doi.org/10.1017/S0890060498121133 Published online by Cambridge University Press

Linear inequality systems can be written in the form $Ax \leq b$, where x is the vector of variables involved, A is the matrix of coefficients and b is a vector of constants in $\mathcal{R}$. The system has n_v variables and n_i inequalities. To eliminate x_1 from the original system, the inequalities are normalized through dividing each inequality with a nonzero coefficient c for x_1 by $|c|$. After reordering this yields the following set of inequalities:

$$x_1 + \sum_{i=2}^{n_v} a_{i,j} x_i \leq b_j \qquad (j = 1 \ldots n_i'). \tag{9}$$

$$-x_1 + \sum_{i=2}^{n_v} a_{i,j} x_i \leq b_j \qquad (j = n_i' + 1 \ldots n_i''). \tag{10}$$

$$\sum_{i=2}^{n_v} a_{i,j} x_i \leq b_j \qquad (j = n_i'' + 1 \ldots n_i). \tag{11}$$

Using Eqs. (9) and (10) the following interval of valid values can be determined for x_1.

$$\max_{j=n_i'+1 \ldots n_i''} \left(\sum_{i=2}^{n_v} a_{i,j} x_i - b_j \right) \leq x_1 \leq \min_{j=1 \ldots n_i'} (b_j - \sum_{i=2}^{n_v} a_{i,j} x_i). \tag{12}$$

To ensure that a set of values for $x_2 \ldots x_{n_v}$ allows a consistent value for x_1, the interval described in Eq. (12) must not be empty. This is true if the following constraints on $x_2 \ldots x_{n_v}$ hold:

$$\sum_{i=2}^{n_v} a_{i,k} x_i - b_k \leq b_j - \sum_{i=2}^{n_v} a_{i,j} x_i$$

$$(j = 1 \ldots n_i', k = n_i' + 1 \ldots n_i'').$$

Combining this with the inequalities set mentioned at (11) a new inequality set containing $n_v - 1$ variables and n_i^{new} inequalities is:

$$n_i^{new} = n_i'(n_i'' - n_i') + n_i - n_i''. \tag{13}$$

Any solution of this new set of inequalities can be extended to a solution of the original set by choosing x_1 within the interval given by Eq. (12). After this method has been performed for all but one variables, an interval for x_i, which only depends on the values of the variables $x_{i+1} \ldots x_{n_v}$ can be determined similar to Eq. (12). For x_{n_v} the method provides an interval in $\mathcal{R}$ given by constants.

To eliminate x_2^{size} from (8), the first two lines of the set have to be combined in the described manner and replaced by this combination. Equation 14 shows the new set of inequalities, where the first inequality is the only one generated by this elimination step:

$$\begin{array}{rrrrcr} x_1^{pos} & +y_1^{pos} & +x_1^{size} & & \leq & 1310 \\ x_1^{pos} & +y_1^{pos} & +x_1^{size} & +y_1^{size} & \leq & 1760 \\ -x_{1\,pos} & & & & \leq & -300 \\ -x_1^{pos} & & -x_1^{size} & & \leq & -300 \\ & -y_1^{pos} & & & \leq & -500 \\ & -y_1^{pos} & & & \leq & -500 \\ & & -x_1^{size} & & \leq & -200 \\ & & & -y_1^{size} & \leq & -200 \\ & & & -y_1^{size} & \leq & -250. \end{array} \tag{14}$$

Table 1 contains the inequalities that are calculated by Fourier–Motzkin elimination in this example. The second column contains the constraints generated to determine the minimum value for the variable mentioned in its first col-

Table 1. *All constraints generated by the classical Fourier–Motzkin elimination*

Var.	Constraints for Minimum	Constraints for Maximum
x_2^{size}	$x_2^{size} + x_1^{pos} + y_1^{pos} + x_1^{size} \leq 1510$	$-x_2^{size} \leq -200$
x_1^{pos}	$x_1^{pos} + y_1^{pos} + x_1^{size} + y_1^{size} \leq 1760$	$-x_1^{pos} \leq 300$
	$x_1^{pos} + y_1^{pos} + x_1^{size} \leq 1310$	$-x_1^{pos} - x_1^{size} \leq 300$
y_1^{pos}	$y_1^{pos} + x_1^{size} + y_1^{size} \leq 1460$	
	$y_1^{pos} + y_1^{size} \leq 1460$	$-y_1^{pos} \leq 500$
	$y_1^{pos} + x_1^{size} \leq 1010$	$-y_1^{pos} \leq 500$
	$y_1^{pos} \leq 1010$	
x_1^{size}	$x_1^{size} + y_1^{size} \leq 960$	
	$x_1^{size} + y_1^{size} \leq 960$	$-x_1^{size} \leq -200$
	$x_1^{size} \leq 510$	
	$x_1^{size} \leq 510$	
y_1^{size}	$y_1^{size} \leq 960$	
	$y_1^{size} \leq 960$	$-y_1^{size} \leq -200$
	$y_1^{size} \leq 760$	$-y_1^{size} \leq -250$
	$y_1^{size} \leq 760$	

https://doi.org/10.1017/S0890060498121133 Published online by Cambridge University Press

umn, the third column contains the constraints used to find its maximum value.

In general, this algorithm generates an exponential number of inequalities; for example, see Schrijver (1986). However, it can be shown that for binary constraints, that is, if each inequality involves only two variables, the Fourier–Motzkin elimination has a complexity of $O(mn(^{2}\log n+3)\log n)$. Unfortunately, most inequalities in IDIOM cannot be restricted in this way. Nevertheless, the use of sparse matrices, redundancy reduction, and variable ordering, as shown in Sections 2.2.2 and 2.2.3, improve this method's performance significantly for problems treated by IDIOM.

2.2.1. Refine variable domains

The redundancy reduction described in the next section uses bounds on variables and depends upon the precision of these bounds; the better they estimate the real interval of feasible values the better the detection of redundancy works. Therefore, these bounds are refined during the process of elimination whenever possible. They are initialized according to the site where the design is to be placed, and after each elimination step all constraints are used to refine the domains of the variables.

Figure 2 shows the algorithm as it is used in IDIOM to refine bounds according to one constraint C, for example, $\sum_{i=1}^{n} a_i x_i \leq b$. To refine a bound for variable x_j the algorithm uses the following fact: If a_j is positive, C can be written as:

$$x_j \leq \frac{-\sum_{\substack{i=1\ldots n \\ i\neq j}} a_i x_i + b}{a_j}.$$

Thus, using the appropriate bounds as shown in Figure 2, a better upper bound for x_j might be found. Lower bounds can be improved for variables with a negative coefficient, C is then written as

$$\frac{\sum_{\substack{i=1\ldots n \\ i\neq j}} a_i x_i - b}{x_j} \leq x_j.$$

Consider the constraint $x_1^{pos} + y_1^{pos} + x_1^{size} + y_1^{size} \leq 1760$ taken from Table 1. The bounds for x_1^{pos} were initialized with [300, 1300], those for y_1^{pos} to [500, 1500] and those for the sizes to [200, 1000] due to the size of the construction site and the minimum size constraints on both rooms. The constraint above can be rewritten as

$$x_1^{pos} \leq 1760 - y_1^{pos} - x_1^{size} - y_2^{size}.$$

and the upper bound of x_1^{pos} can be calculated using the lower bounds of y_1^{pos}, x_1^{size} and y_1^{size}. The new upper bound for x_1^{pos} is 860. In a similar way the domains of all variables involved here can be refined to [300, 860] for x_1^{pos}, [500, 1060] for y_1^{pos}, and [200, 760] for sizes.

```
proc refine_intervals(C) ≡
  while bounds change do
    for each variable x_i with a_i ≠ 0 do
      limit = 0.0;
      for each variable x_j with a_j ≠ 0 ∧ i ≠ j do
        if a_j > 0
          then limit = limit − lower_bound(x_j)(a_j/a_i);
        else limit = limit − upper_bound(x_j)(a_j/a_i); fi
      od
      limit = limit − b/a_i;

      if a_i > 0
        then if limit < upper_bound(x_i)
                then upper_bound(x_i) = limit; fi
      else if limit > lower_bound(x_i)
              then lower_bound(x_i) = limit; fi
      fi
    od
  od
```

Fig. 2. Algorithm to refine bounds, C represents the constant $\sum a_i x_i \leq b$.

2.2.2. Redundancy reduction

The standard method for Fourier–Motzkin elimination is known to generate an exponential number of inequalities in the general case (Schrijver, 1986). However, it is also known that this elimination method produces many redundant constraints. Lassez et al. (1993) propose techniques to eliminate redundancy from large systems of linear constraints. However, these methods involve solving linear programming problems and therefore are inappropriate for use during the elimination process. Techniques presented in Klein and Holm (1983) helped inspire the methods used in IDIOM. Using bounds on variables, as described here, the redundancy in the systems of inequalities that are generated by the elimination process can be reduced with little cost resulting in considerable gains in speed. Two types of redundancy are detected:

- Type 1: A constraint is implied by the bounds on variable values.
- Type 2: A constraint is implied by bounds and another constraint.

https://doi.org/10.1017/S0890060498121133 Published online by Cambridge University Press

Any constraint is considered redundant if it is implied by the bounds on its variables or if it is implied by another constraint and the bounds on variables. After each elimination step, redundancy in the remaining system of inequalities is reduced by removing every constraint that is found to be redundant.

Detecting constraints that are implied by bounds is accomplished by checking if a constraint remains satisfied when variables are substituted by the worst-case values. When $\sum_i a_i x_i \leq b$ is the constraint to be checked, worst-case values are lower bounds for all x_i with $a_i < 0$ and upper bounds for all x_i with $a_i > 0$. Figure 3 shows an example for such a case in two dimensions. The grey rectangular area shows the valid values defined by bounds on the involved variables and the grey bar along the constraint C shows which half-plane contains the valid solutions according to C. In the example shown in Figure 3, this half-plane entirely contains the area defined by the bounds of the involved variables, and therefore C is implied by these bounds.

Refinement of bounds on variable values and elimination of type 1 redundancy are performed before each elimination step. Therefore, the inequality system in Eq. (8) is reduced to the following before the elimination process begins:

$$\begin{array}{llllll} x_2^{size} & +x_1^{pos} & +y_1^{pos} & +x_1^{size} & \leq & 1510 \\ & x_1^{pos} & +y_1^{pos} & +x_1^{size} \quad +y_1^{size} & \leq & 1760. \end{array}$$

All constraints that contain only one variable are used to refine bounds and are then eliminated. The constraint $-x_1^{pos} - x_1^{size} \leq 0$ is eliminated because x_1^{pos} and x_1^{size} have only positive domains and finally inequalities that do not involve variables are removed.

To detect whether one of two constraints C_1 and C_2 is redundant, the difference $D = C_1 - C_2$ is analyzed. When D is implied by the bounds of the involved variables, C_1 is implied by C_2 and these bounds because $C_1 = D + C_2$. An illustration for this in two dimensions is given in Figure 4.

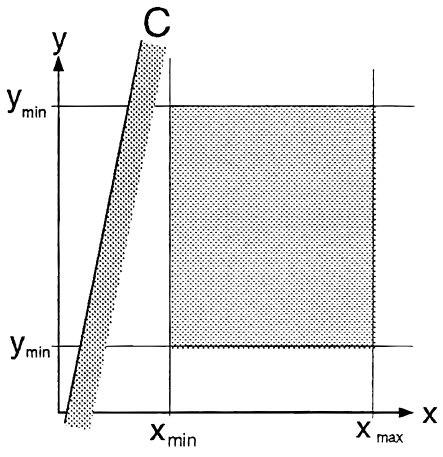


Fig. 3. Example of Type 1 redundancy, C is implied by bounds on x and y.

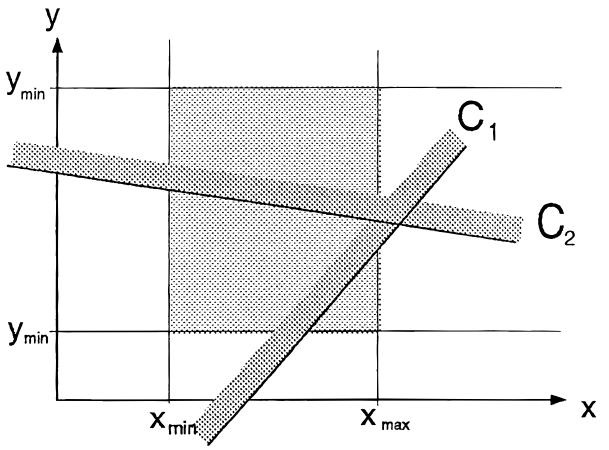


Fig. 4. Example of Type 2 redundancy, C_1 is implied by C_2 and bounds on x and y.

The detection of redundancy that occurs when a constraint is implied by another within the bounds of the variables is illustrated in the following example:

$$x - z \leq -120$$

$$x + y - z \leq -120,$$

where bounds are [0, 545] for x, [200, 745] for y, and [400, 865] for z. Subtracting the second constraint from the first gives $-y \leq 0$, which is always true because the lower bound on y is 200. Therefore, $x - z \leq -120$ is redundant.

This method of reducing redundancy removes many constraints. However, it is important to note that not to lose information, IDIOM must also consider the bounds imposed on each variable to be constraints. Nevertheless, this type of reduction improves the performance of Fourier–Motzkin elimination. Table 2 shows the decrease in generated constraints when using redundancy reduction for a few examples. The rows "Generated w/o rr" and "Generated with rr" give the number of constraints that are generated during Fourier–Motzkin elimination, the rows "Stored w/o rr" and "Stored with rr" list how many constraints that are kept in memory. Without redundancy reduction, only those constraints that do not involve variables are eliminated, whereas with redundancy reduction other redundant constraints are also discarded. The execution times given refer to a Silicon Graphics workstation using an R4600 processor.

2.2.3. Variable elimination order

When solutions to discrete constraint satisfaction problems are enumerated through backtracking, the order in which variables are instantiated is an important issue. Selecting the most difficult variables first for instantiation is likely to avoid

https://doi.org/10.1017/S0890060498121133 Published online by Cambridge University Press

Table 2. *Comparison of performance of Fourier–Motzkin elimination with and without redundancy reduction (rr)*

Example	1	2	3
Configuration			
Variables	8	20	73
Equalities	3	7	39
Inequalities	11	41	174
Generated w/o rr	36	72,157	>500,000
Generated with rr	17	119	255
Stored w/o rr	23	7357	>50,000
Stored with rr	6	42	58
Time used w/o rr [s]	0.01	27.99	>5000
Time used with rr [s]	0.01	0.02	0.08

building partial solutions that cannot be completed later and therefore often improves performance. Heuristics to find these variables are suggested and evaluated in several contexts (Dechter & Meiri, 1989; Sadeh & Fox, 1996). IDIOM improves the performance of Fourier–Motzkin elimination in a similar manner.

Variable ordering schemes proposed for discrete constraint satisfaction problems usually first select variables that are named in several constraints for instantiation. In contrast, IDIOM reduces combinatorial effects in Fourier–Motzkin elimination through first eliminating variables that are named in few constraints. For example, if we eliminate x_2^{size} from Eq. (8), two constraints are replaced by one, so the number of constraints can decrease when eliminating variables. While the variables that do not cause the combinatorial problems are eliminated, the system collects information about bounds on variables and thus improves the detection of redundancy. Often, combinatorial effects can be avoided altogether.

At each step during Fourier–Motzkin elimination, the variable that generates the smallest number of constraints is treated. The number $n_i^{new}(x)$ of new constraints generated by elimination of some variable x can be calculated according to Eq. (13) as follows:

$$n_i^{new}(x) = n_i^{pos}(x) * n_i^{neg}(x),$$

where $n_i^{pos}(x)$ is the number of constraints in which variable x has a positive coefficient and $n_i^{neg}(x)$ denotes the number of constraints in which x has a negative coefficient. At each elimination step, IDIOM eliminates a variable with minimal $n_i^{new}(x)$.

In Eq. (14) this method would eliminate y_1^{size} instead of x_1^{pos} because this elimination only generates two instead of four constraints and results in the inequality system (15).

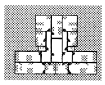
$$\begin{array}{rrrcr} x_1^{pos} & +y_1^{pos} & +x_1^{size} & \leq & 1310 \\ x_1^{pos} & +y_1^{pos} & +x_1^{size} & \leq & 1560 \\ x_1^{pos} & +y_1^{pos} & +x_1^{size} & \leq & 1510 \\ -x_1^{pos} & & & \leq & -300 \\ -x_1^{pos} & & -x_1^{size} & \leq & 0 \\ & -y_1^{pos} & & \leq & -500 \\ & -y_1^{pos} & & \leq & -500 \\ & & -x_1^{size} & \leq & -200. \end{array} \quad (15)$$

The benefit of choosing the variables to be eliminated in this way can only be illustrated together with redundancy reduction in larger examples; otherwise, the combinatorial explosion is just delayed, not avoided. In the next elimination step in (15), at least six new constraints would be generated and the number of inequalities generated in subsequent steps would rapidly increase. Table 3 illustrates the effects of reordering the variables in Fourier–Motzkin elimination for three more complex examples, which among other constraints contain approximations of minimum area requirements on most of the rooms. For all numbers given in Table 3, redundancy reduction was used. The second example in Table 3 shows that the algorithm becomes more predictable when variables are reordered. Although Example 1 is a subset of Example 2, the algorithm without variable reordering solves Example 2 faster than Example 1. This effect disappears when variables are ordered.

2.3. Choose a solution with minimal change

The methods described in the previous sections define a solution space for a constraint system. This process is performed whenever the set of constraints is manipulated or the number of parameters is changed by adding or remov-

Table 3. *Comparison of performance of Fourier–Motzkin elimination with and without variable reordering (vr)*

Example	1	2	3
Configuration			
Variables	61	73	180
Equalities	29	39	92
Inequalities	141	183	403
Generated w/o vr	692	486	29,593
Generated with vr	326	356	736
Stored w/o vr	198	135	3616
Stored with vr	88	106	193
Time used w/o vr [s]	0.21	0.20	157.05
Time used with vr [s]	0.09	0.16	1.30

https://doi.org/10.1017/S0890060498121133 Published online by Cambridge University Press

ing objects. Prior to interactive adaptation, the system proposes an initial solution. Because objects are based on cases of good solutions, this new solution should involve the least change with respect to the original case. In addition, any changes the designer has introduced before need to be maintained where possible.

Similar to Eq. (12), the Fourier–Motzkin elimination provides inequalities for every x_i as follows:

$$\max_{j=n_i'+1\ldots n_i''}\left(\sum_{k=i+1}^{n_v} a_{k,j}x_k - b_j\right) \le x_i$$

$$\le \min_{j=1\ldots n_i'}\left(b_j - \sum_{k=i+1}^{n_v} a_{k,j}x_k\right). \qquad (16)$$

The inequalities in Eq. (16) allow the solver to calculate an interval of possible values for variable x_i, the bounds of which depend only on $x_{i+1}\ldots x_{n_v}$, where the interval for x_{n_v} is given by constants. To find a solution for the inequalities, the solver starts by choosing a value for x_{n_v}. If this value is chosen within the interval for x_{n_v} the Fourier–Motzkin elimination guarantees that, for x_{n_v-1}, a nonempty interval of possible values can also be found. Therefore, the solver can recursively determine values for all variables.

Using intervals of possible values, it is easy to find a solution that is as near to the current solution as possible. The solver chooses a value for a variable by checking its interval of possible values. If the current value of the variable is within the interval, the solver will use this value. If the value is outside, it will be set to the nearest interval boundary. When all values for free variables are determined in this way, the results of the Gauss–Jordan elimination are used to find values for the dependent variables.

2.4. Nonlinear constraints

Important constraints are often nonlinear. A limitation to linear constraints is therefore too restrictive. To make our solver powerful enough for real-world applications, some nonlinear constraints are approximated through formulation of a set of linear constraints.

Consider the constraint, $xy \ge A_{\min}$, where x and y are the length and width of an object and $A_{\min}$ is the minimum area imposed by the constraint (illustrated in Fig. 5). Using the minimum sizes that all variables must have, together with the above constraint, a maximum value for x is determined as follows:

$$x_{\max} = \frac{A_{\min}}{y_{\min}}.$$

If values of x are larger than $x_{\max}$, then the constraint defining a minimum on y implies that there are always acceptable values for xy. Thus, it is sufficient to approximate $xy \ge A_{\min}$ in the interval $[x_{\min}, x_{\max}]$.

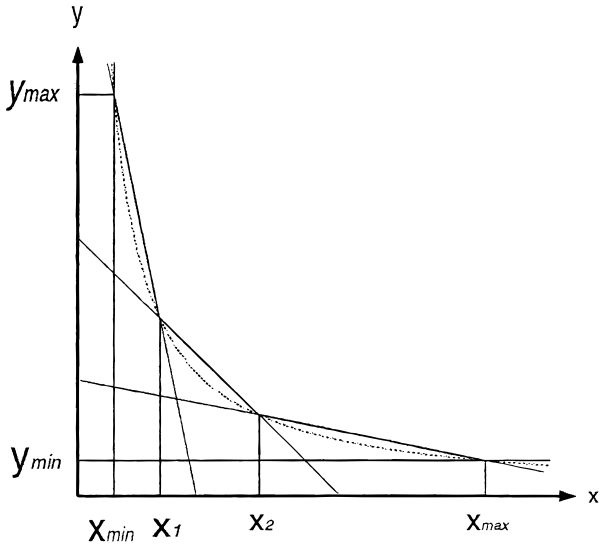


Fig. 5. Linear approximation of $xy > A_{\min}$.

IDIOM uses the following relationship to determine the points x_i, where piecewise linear approximation intersects the curve to be approximated. When approximating $xy \ge A_{\min}$ using n linear constraints, x_i is calculated as follows:

$$x_i = \sqrt[n]{x_{\min}^{n-i}x_{\max}^{i}} \qquad (i = 1\ldots n-1).$$

The approximation shown in Figure 5 never underestimates xy, such that the constraint is always fulfilled. Figure 6 shows the accuracy of this approximation through a worst-case overestimation with respect to the given $x_{\min}$ for an approximation using three, four, and five constraints. It is assumed that $y_{\min} = x_{\min}$ and $A_{\min} = 14m^2$ for this figure.

Constraints of the form $xy \ge A_{\min}$ occur in IDIOM when a room must have a minimum area. Figure 6 shows that such a

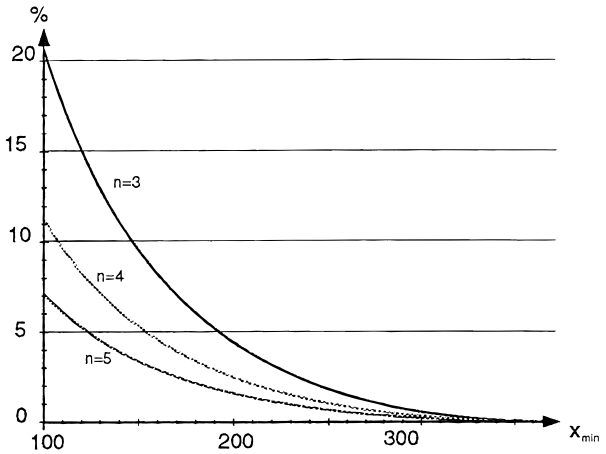


Fig. 6. Worst-case overestimation by the approximation of $xy \ge 14m^2$ using n linear constraints.

https://doi.org/10.1017/S0890060498121133 Published online by Cambridge University Press

minimum area constraint can be approximated with small errors when high minimum constraints are given on width and length of the room. In the example given in the figure the error is below 5% using only three constraints if the minimum width and length are at least 200 cm. This value decreases when a smaller $A_{\min}$ is needed. In practical applications a minimum width and length of 200 cm is reasonable to ensure the usefulness of most rooms. Therefore, it can be expected that this approximation is sufficiently accurate for architectural tasks when three linear approximations are used.

3. ACTIVATING PREFERENCES

Constraints in IDIOM may be fixed or preferred, hereafter referred to as fixed constraints and preferences, respectively. Preferences have a priority and may be deactivated if they are in conflict with other preferences or fixed constraints. Each preference has a priority and several preferences may have equal priority.

3.1. Activation behavior

The following rules summarize how IDIOM activates preferences:

- A preference that conflicts with fixed constraints is deactivated.
- If two preferences with different priorities conflict, the higher priority preference is activated.
- If two preferences with the same priority conflict, IDIOM activates the preference that conflicts with fewer lower priority preferences.
- IDIOM reactivates preferences whenever possible.

3.1.1. Conflicts between a preference and a fixed constraint

When a preference conflicts with fixed constraints, it will be deactivated by IDIOM and remain deactivated as long as the conflicting fixed constraints remain in the system. However, IDIOM reactivates preferences whenever possible. For example, if a designer prefers a hall width of 140 cm, whereas the site and minimum areas imposed on other rooms make such a wide hall impossible, the preference will remain deactivated until the designer removes all conflicting fixed constraints.

3.1.2. Conflicts between preferences having different priorities

An example of conflicting preferences having different priorities is shown in Figure 7. A designer has introduced high-priority preferences asking for a minimum area of $16m^2$ on single rooms and a low-priority preference for aligning north walls of the single rooms. Not all preferences can be fulfilled due to other constraints. IDIOM activates the more important minimum area preferences and then discovers that the aligning preference cannot be activated.

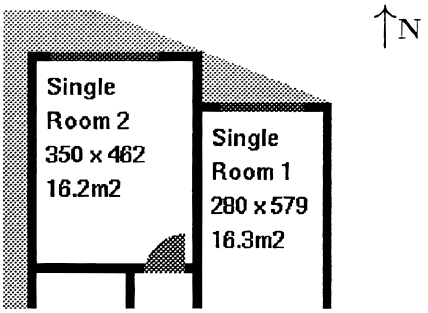


Fig. 7. The preference to align northern walls of single rooms remains deactivated due to other constraints restrictions.

3.1.3. Conflicts between preferences with same priority

The most complex situation occurs when two conflicting preferences have the same priority. IDIOM evaluates both possibilities until the preference that allows the activation of most other preferences is found. In the example illustrated in Figure 7, if only one of the two minimal area preferences can be active, IDIOM chooses which one to activate. Now let us assume that on the right room there is a less important minimal x^{size} preference that is in conflict with the minimal area preference on the left room. IDIOM activates the minimal area preference on the right room in order to have the minimal x^{size} preference active as well.

3.1.4. Reactivation of preferences

Reactivation of preferences is carried out in several ways. After changes in the set of constraints and preferences, IDIOM starts the algorithm described below to find the best combination of preferences that can be active at the same time. Therefore, each time that a constraint or preference is removed, added, or given another priority, IDIOM may change the set of active preferences.

3.2. Preference activation algorithm

When IDIOM searches for the best set of preferences to activate, it treats preferences in groups having the same priority. It proceeds in decreasing order of priority, evaluating opportunities to activate as many preferences as possible. All possibilities to activate the highest number of preferences are considered in the next activation step to find the best set of nonconflicting preferences. Activating a preference in this context means to add a preference to a set of constraints and preferences, and is only possible if this action does not cause any conflicts.

The method `activate_preferences` (see Fig. 8) controls the activation of preferences. It takes the set of fixed constraints with no preferences active `cs`, and a list of preferences for priority level (`prefs[]`) as inputs and generates a list of active preferences according to the guidelines described in Section 3.1. As a temporary structure, a list of

https://doi.org/10.1017/S0890060498121133 Published online by Cambridge University Press

```
funct activate_preferences(cs, prefs[]) ≡
  new_bests := {cs}; old_bests := ∅;
  for priority := 0 to nb_priority_levels do
    old_bests := new_bests;
    for each preference_set in old_bests do
      call activate_level(prefs[priority], preference_set, new_bests);
    od
  od
  return first(new_bests);
```

Fig. 8. Pseudo-Code for activation of preferences.

sets of nonconflicting preferences `new_bests` is used to store all found sets of preferences with best quality according to Section 3.1.

The procedure `activate_level` adds recursively as many preferences as possible from the list it receives to the constraint set. It writes the output into `new_bests`. The sets of preferences stored in `new_bests` are replaced when better ones are found (see Section 3.1). If `activate_level` only finds sets that are of equal quality (see Section 3.1) as those that are already in `new_bests` the procedure appends the new sets. If no better sets of preferences are found, `new_bests` remains unchanged. Figure 9 shows an initial implementation of this algorithm.

The procedure activates the first preference *p* in the list of preferences it receives as a parameter. If *p* is not in conflict with any fixed constraints and preferences in *current_set*, the algorithm checks if better sets of active preferences can be found with *p* active. In all cases it searches for better sets of active preferences with *p* inactive. During this process it maintains *new_bests*, which always contains the best combinations encountered during the search.

```
proc activate_level(prefs, current_set, new_bests) ≡
  if prefs = ∅ then exit fi;
  p := first(prefs); add p to current_set; old_bests := new_bests;
  if activation of p was successful
    then
    if |current_set| > |first(new_bests)| then new_bests := ∅; fi
    if |current_set| ≥ |first(new_bests)|
      then add current_set to new_bests fi
    call activate_level(prefs - p, current_set, new_bests); fi
  call activate_level(prefs - p, current_set - p, old_bests);
```

Fig. 9. Pseudo-Code for activation of preferences on one priority level, without pruning.

```
proc activate_level(prefs, current_set, new_bests) ≡
  if prefs = ∅ then exit fi;
  if |prefs| + active(current_set) < active(new_bests) then exit fi;
  p := first(prefs); add p to current_set; old_bests := new_bests;
  if ∃x|x ⊆ current_set ∧ x ∈ known_contradictions then exit fi;
  if activation of p failed
    then add current_set to known_contradictions;
  else
    if |current_set| > |first(new_bests)| then new_bests := ∅; fi
    if |current_set| ≥ |first(new_bests)|
      then add current_set to new_bests fi
    call activate_level([]prefs - p, current_set, new_bests); fi
    call activate_level(prefs - p, current_set - p, old_bests);
```

Fig. 10. Pseudo-Code for activation of preferences on one priority level, with pruning.

The algorithm shown in Figure 9 uses backtracking to search through all sets of active preferences to find the best set for the current priority level. However, large portions of this search are futile and are pruned by IDIOM's preference activation algorithm. The algorithm shown in Figure 10 improves the initial implementation of the preference activation algorithm through the following:

- Recalculation of any combination of preferences that contains a known contradiction is avoided.
- As soon as IDIOM detects that no solution with at least equal quality as those in *new_bests* can be found, even if all remaining preferences are activated, it stops backtracking.

3.3. Efficiency

Because this algorithm uses binary backtracking, performance may not be adequate in all cases. However, pruning large parts of the search space helps to keep the task feasible. Pruning performs well when only a few or almost all preferences are active. Nevertheless, combinatorial explosion cannot be avoided when many preferences have the same priority. Users of IDIOM are not expected to create

https://doi.org/10.1017/S0890060498121133 Published online by Cambridge University Press

such situations because this complicates a rational exploration of design alternatives.

4. EXPLORING DESIGN ALTERNATIVES

One of the most important aspects of IDIOM is its interactivity. IDIOM provides two facilities for supporting designers during exploration of design alternatives: interactive adaptation of parameters and strengthening and weakening of preferences.

4.1. Interactive adaptation of parameters

Because IDIOM manipulates values for continuous variables, there is rarely only one solution; often there is an infinite number. Designers need to explore spaces of solutions to let them introduce knowledge that cannot be modeled in IDIOM.

In IDIOM designers can modify a solution proposed by the system through interactively adapting parameters. Adaptable parameters are positions of walls and elements. Adaptation consists of the following steps:

1. The designer chooses a parameter to adapt by clicking on a wall or an element.
2. IDIOM calculates the current range of valid values for the parameter and shows this range to the designer, if requested.
3. The designer adapts the parameter by moving the mouse while IDIOM keeps track of all changes in the design and continuously shows the adapted solution.

Figure 11 shows an example of an adaptation. In the left figure the user clicks on the left wall, which is shared by a double room and the living/dining room. Arrows appear to indicate the range of feasible adaptations. The user then moves the mouse to the right to obtain the configuration by the right-hand figure. The apartment contains minimum area constraints on the double room and the living/dining room to the lower left, and this is why parts of the apartment move toward the north during adaptation.

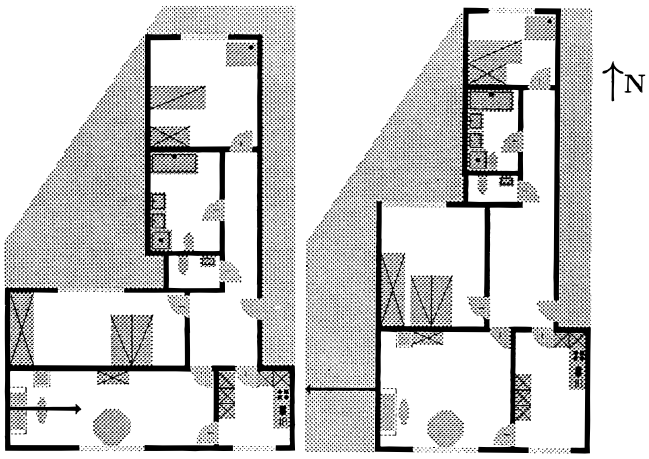


Fig. 11. Adaptation of a wall position.

4.1.1. Determine range of valid values

When the designer chooses to adapt a particular parameter p by clicking on a wall or an element, IDIOM calculates a range of possible values for p. Because all parameters available for adaptation in IDIOM have a geometrical meaning, the system shows this range with an arrow as illustrated in Figure 11.

The range of feasible values is calculated using the Fourier–Motzkin elimination algorithm. As shown in Section 2.3, this algorithm provides an interval of feasible values for every variable x_i, which depends in variables $x_{i+1} \ldots x_{n_v}$ only [see Eq. (16)] and the interval for the last variable eliminated is given by constants. Prior to adaptation of parameter p, IDIOM performs the Fourier–Motzkin elimination, eliminating all variables except for p, thus calculating the largest interval of feasible values for p.

This method ensures that the interval given for a parameter includes all possible values. An earlier approach to interactive adaptation (Hua et al., 1996) is based on the premise that only *one* free variable at a time is changed. In IDIOM, the interval given includes all feasible values for p, even those that imply the adjustment of *several* free parameters. The designer can be sure that no value outside the given range can be reached as long as all current constraints remain active.

The method described above can only be used if the chosen parameter is free. If dimensionality reduction determines p to be a dependent variable, it is eliminated from the set of inequalities and does not occur in the Fourier–Motzkin elimination at all. In this situation, IDIOM searches a free variable q upon which p depends and makes q a dependent variable. This is done by dividing the equality that defined the value of p before by q's coefficient, swapping columns p and q, and by substituting q in all equalities and inequalities. Now q is a dependent variable while p is free and can be used for adaptation. After adaptation, these actions are reversed to come back to the original system of constraints. When IDIOM cannot find any free variable which p depends upon, the chosen parameter cannot be adapted because the constraints have bound it to a fixed value.

4.1.2. Choose a solution with minimal change during adaptation

During adaptation, IDIOM repeatedly calculates and shows solutions according to the parameter that was chosen to be adapted and the position of the mouse pointer. The choice of these solutions is performed as described in Sec-

https://doi.org/10.1017/S0890060498121133 Published online by Cambridge University Press

tion 2.3 except that the parameter *p* is set as close as possible to the value indicated by the mouse position.

All other variables are chosen to be as near as possible to the design prior to adaptation. As a result, IDIOM's behavior while adapting is analogous to tearing at an elastic model of the design. If the designer drags the chosen parameter to an extreme value and then comes back, the original situation is resumed.

Figure 11 shows an example of many parameters involved in adaptation, although the designer modifies the value of only one variable.

4.1.3. Adapting two parameters at a time

When a designer clicks on an element that is not attached to a wall, two variables, *x* and *y* positions, can be changed. For example, when moving a piece of furniture, a designer expects the system to allow movement of the element in both *x* and *y* directions. The same need arises when:

- moving corners of rooms;
- moving rooms as a whole; and
- moving elements attached to a wall, thus influencing the position of the wall to which they are attached.

Because the constraint system is linear, the range of possible values for the adaptation of two parameters p_1 and p_2 has the form of a convex polygon and thus cannot be shown using arrows. However, the calculation of the range of feasible values and the choice of a solution used in adaptation of one parameter can be extended to the adaptation of two parameters.

Instead of delaying elimination of only one parameter to the end, both parameters to be adapted have to be eliminated after all other variables. For the parameter eliminated last, for example, p_1, the interval of valid values is given by constants and found by the Fourier–Motzkin elimination algorithm. To approximate the polygon of feasible combinations of the two parameters, we use the fact that Fourier–Motzkin elimination gives us intervals for p_2, which only depend upon p_1. The resulting algorithm shown in Figure 12 therefore approximates the polygon of feasible combinations.

The choice of a solution with minimal change to the current solution is performed exactly in the same manner as described in Section 4.1.2, except that values of two parameters, p_1 and p_2, are influenced by the designer.

$$\begin{array}{l}\textbf{for } p_1 := min_1() \textbf{ to } max_1() \textbf{ step } \frac{max_1()-min_1()}{n} \textbf{ do}\\ \quad \text{add } (p_1, min_2(p_1) \text{ to } polygon\\ \quad \text{add } (p_1, max_2(p_1) \text{ to } polygon\\ \textbf{od}\end{array}$$

Fig. 12. Determine the range of feasible values for adaptation of two variables. $\min_1()$ and $\max_1()$ define the feasible interval of p_1. $\min_2(p_1)$ and $\max(p_1)$ define the feasible interval for p_2 with respect to p_1.

4.2. Strengthening and weakening preferences

IDIOM attributes a priority to each preference. Nevertheless, a designer can interactively change these priorities. The increasing of a preference's priority is called *strengthening* and the decreasing of a preference's priority *weakening*.

Each time strengthening or weakening is performed, the algorithm described in Section 3.2 is called to find the best set of preferences to be active. It resets the set of active preferences to empty every time it is launched and therefore preferences that were active before can become inactive and vice versa. This is *deactivation* and *activation*. A preference may be reactivated, that is, it is first active and after being deactivated becomes active again. This may happen when a conflict between two preferences deactivates one of them. If the now active preference is deactivated by some new fixed constraint or more important preferences, the formerly deactivated preference may become active again.

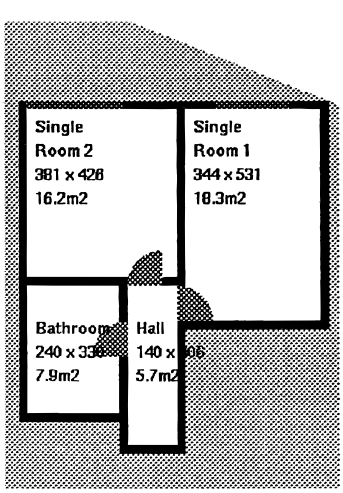
Deactivation and activation is illustrated using the example situation shown in Figure 13. Suppose a luxury apartment with simple facades and a large hall is desired. Two single bedrooms, a bathroom and a hall are already introduced into the site. Furthermore, a preferred minimum area is imposed on each single room with priority 4, an alignment preference for the northern walls of both single rooms has priority 8 and the designer has posted a preference of priority 9 on the minimum width of the hall. Because the site is large enough to meet all these requirements, IDIOM activates all preferences.

To satisfy plumbing requirements, the south-west corner of the bathroom has to be fixed where it appears in Figure 14. The preference activation algorithm is started when this new constraint is introduced and detects that there is no way to activate all preferences present. As it can still activate both preferences on priority-level 4, it does so but then deactivates the simple facade preference. Nevertheless, the constraint on the width of the hall remains active.

Suppose that the designer does not agree with the configuration of Figure 14 because it contains a narrow single room with a width of only 280 cm and therefore posts a preferred minimum width of 300 cm at priority 2. This results in several changes as shown in Figure 15. As the preference activation algorithm makes its way through the priority levels, the following actions occur:

- The high-priority preference on the width of single room 1 is activated.
- The minimum area preference on single room 1 is kept active but the algorithm detects that the minimum area on single room 2 is in conflict with the more important preference on the width of single room 1 and therefore must be deactivated.
- The simple facade preference is reactivated because single room 2 can shrink to align with single room 1, now that the preference on its area is inactive.
- Finally the preferred minimum width on the hall is in conflict with the minimum width constraints and pref-

https://doi.org/10.1017/S0890060498121133 Published online by Cambridge University Press



Pr.	Description	Act.
4	$area_{Single_Room_1} \geq 16m^2$	$\surd$
4	$area_{Single_Room_2} \geq 16m^2$	$\surd$
8	$Facade^{Single_Room_1}_{Single_Room_2}$	$\surd$
9	$x_size_{Hall} \geq 140\text{cm}$	$\surd$

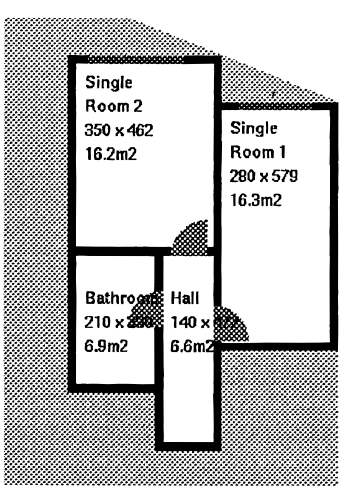
Fig. 13. Example for strengthening and weakening preferences (Pr. Means priority, Act. The condition of activity).

erences on the bathroom and the single room 1, and thus becomes inactive.

Supposing that the designer now wants the minimum width preference on the hall to become more important and therefore strengthens it to the same priority as the preferred minimum width on single room 1. These two preferences are in conflict and IDIOM chooses one of them. First, it follows both possibilities but on priority level 4 it detects that using the preferred minimum width on the hall allows activating both preferences on minimum areas while the preferred minimum on single room 1 allows only the preferred minimum area on single room 1 to be activated. Thus, IDIOM chooses the combination of active preferences shown in Figure 16.

5. CONCLUSION

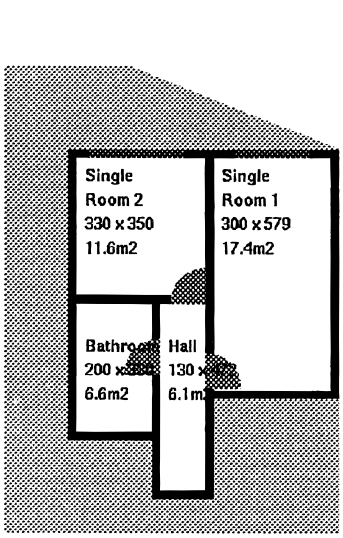
This research describes a novel approach for spatial constraint solving. The combination of Gauss–Jordan elimination with an improved Fourier–Motzkin elimination algorithm solves constraint sets that correspond to spatial configuration problems involving rectangular shapes and continuous parameters. Linear constraints and piecewise linear, convex approximations of nonlinear constraints are treated. The Fourier–Motzkin elimination method has been improved in such a way that the system can be used interactively. Moreover, these two algorithms provide a consistent description of the complete solution space of such constraint sets. This is useful for interactive adaptation and for investigating dependencies between parameters.



Pr.	Description	Act.
4	$area_{Single_Room_1} \geq 16m^2$	$\surd$
4	$area_{Single_Room_2} \geq 16m^2$	$\surd$
8	$Facade^{Single_Room_1}_{Single_Room_2}$	-
9	$x_size_{Hall} \geq 140\text{cm}$	$\surd$

Fig. 14. Simple facade preference deactivated.

https://doi.org/10.1017/S0890060498121133 Published online by Cambridge University Press



Pr.	Description	Act.
2	$x_size_{Single_Room_1} \geq 300\text{cm}$	$\surd$
4	$area_{Single_Room_1} \geq 16m^2$	$\surd$
4	$area_{Single_Room_2} \geq 16m^2$	-
8	$Facade^{Single_Room_1}_{Single_Room_2}$	$\surd$
9	$x_size_{Hall} \geq 140\text{cm}$	-

Fig. 15. Simple facade preference reactivated.

In addition, preferences (defined as constraints that may be deactivated and reactivated automatically) are included, as well as an algorithm for approximating the optimal set of active preferences. This algorithm accommodates preferences of different priorities and preferences having equal importance, while avoiding unnecessary backtracking. Finally, interactive adaptation of priorities related to preferences offers another means for identifying design opportunities and for exploration of design solutions.

ACKNOWLEDGMENTS

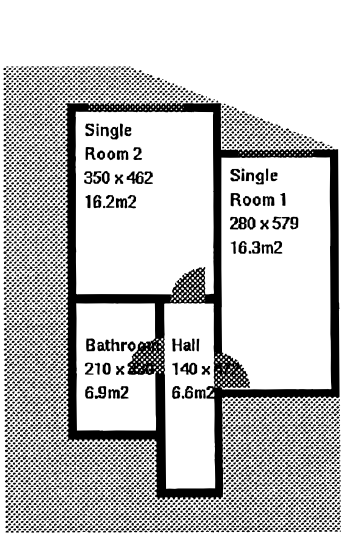
The funding for the project IDIOM was provided by the Swiss Priority Programme in Computer Science (SPP-IF). This project was performed in collaboration, with CAAD, Federal Institute of Technology, Zurich. The authors thank Nathanea Elte and David Kurmann for their collaboration, Boi Faltings and Gerhard Schmitt for discussions, and our practicing architects: Geninasca–Delfortrie Architects, Neuchatel; Atelier d'Architecture, Lausanne; and Archilab, Lausanne for providing comments and for helping with testing and validation.

REFERENCES

Bakhtari, S., Börner, K., Bartsch-Spörl, B., Coulon, C.-H., Janetzko, D., Knauff, M., Hovestadt, L., & Schlieder, C. EWCBR (1993). Contributions of FABEL. FABEL-Report No. 17. Gesellschaft für Mathematik und Datenverarbeitung mbH, Forschungsbereich Künstliche Intelligenz.

Baykan, C.A., & Fox, M.S. (1992). WRIGHT: A constraint based spatial layout system. *Artif. Intell. Eng. Design, 1*, 395–432.

Borning, A., Freeman-Benson, B.N., & Wilson, M. (1992). Constraint hierarchies. *Lisp Symbolic Computation, 5*, 223–270.



Pr.	Description	Act.
2	$x_size_{Single_Room_1} \geq 300\text{cm}$	-
2	$x_size_{Hall} \geq 140\text{cm}$	$\surd$
4	$area_{Single_Room_1} \geq 16m^2$	$\surd$
4	$area_{Single_Room_2} \geq 16m^2$	$\surd$
8	$Facade^{Single_Room_1}_{Single_Room_2}$	-

Fig. 16. Strengthening a preference.

https://doi.org/10.1017/S0890060498121133 Published online by Cambridge University Press

D'Ambrosio, J.G., & Birmingham, W.P. (1995). Preferences-directed design. *AIEDAM, 9*, 219–230.
Dave, B., Schmidt, G., Shih, S.-C., Bendel, L., Faltings, B., Smith, I., Hua, K., Bailey, S., Ducret, J.-M., & Jent, K. (1994). Case-based spatial design reasoning. *Eur. Workshop Case-Based Reasoning*, pp. 115–124.
Dechter, R., & Meiri, I. (1989). Experimental evaluation of preprocessing techniques in constraint satisfaction problems. *Proc. 11th Int. Joint Conf. Artif. Intell.*, pp. 271–277. Morgan Kaufmann, Detroit, Michigan.
Faltings, B. (1990). Case based representation of architectural design knowledge. Proc. Int. Symp. on Computational Intelligence. (Cercone, N. & Gardin, F., Eds.) pp. 273–281. North-Holland, Amsterdam.
Flemming, U., Baykan, C.A., Coyne, R.F., & Fox, M.S. (1992). Hierarchical generate and test *vs.* constraint-directed search. In *Artif. Intell. Design* (Gero, J.S., Ed.), pp. 817–838. Netherlands.
Fourier, J.B. (1970). (reprinted in) *Histoire de l'Academie Royale des Sciences de l'Institut de France*. pp. 325–328. Olms G, Hildersheim.
Fox, M.S. (1987). *Constraint-directed search: A case study of job-shop scheduling*. Research Notes in Artificial Intelligence. Morgan Kaufmann Publishers, Los Altos, California.
Giretti, A., Spalazzi, L., & Lemma, M. (1994). A.S.A. An interactive assistant to architectural design. In *AI in Design '94*, pp. 93–108. (Gero, J.S. and Sudweeks, F., Eds.) Kluwer, Boston.
Gross, M.D. (1990). Knowledge-based support for subsystem layout in architectural design. In *Application of Artificial Intelligence in Engineering: Design*, (Gero, J.S., Ed.), Computational Mechanics Publications, Southampton, Boston.
Hua, K. (1994). *Case based design of geometric structures*. PhD Thesis. Swiss Federal Institute of Technology in Lausanne, Switzerland.
Hua, K., Faltings, B., & Smith, I. (1996). CADRE: Case based geometric design. *Artif. Intell. Eng., 10*, 171–183.
Hummel, R.A., & Zucker, S.W. (1983). On the foundations of relaxation labeling processes. *IEEE Trans. Pattern Analysis Machine Intell., 5(3),* 267–287.
Klein, D., & Holm, S.J. (1983). Some reduction of linear programs using bounds on problem variables. In *Lecture notes in economics and mathematical systems*, Vol. 206, Redundancy in mathematical programming, pp. 80–86. Springer-Verlag, New York.
Krishnaiah, P., & Kanal, L. (1982). *Handbook on Statistics, Vol. 2*. North-Holland, Amsterdam.
Lassez, J.L., Huynh, T., & McAloon, K. (1993). Simplification and elimination of redundant linear arithmetic constraints. In *Constraint Logic Programming—Selected Research*, (Benhamou, F., and Colmerauer, A., Eds.), pp. 73–87. The MIT Press, Cambridge, Massachusetts.
Liggett, R.S. (1985). Optimal spatial arrangement as a quadratic assignment problem. In *Design Optimization*, (Gero, J.S., Ed.), pp. 1–40. Academic Press Inc., Orlando, Florida.
Logan, B., Corne, D.W., & Smithers, T. (1991). Enduring support: On defeasible reasoning in design support systems. In *AI in Design '91*, (Gero, J.S., Ed.), pp. 433–454. Butterworth-Heinemann, Oxford, UK.
Logan, B., & Smithers, T. (1993). Creativity and design as exploration. In *Modelling Creativity and Knowledge Based Design*, (Gero, J.S., and Maher, M.L., Eds.), pp. 139–175. Lawrence Erlbaum Mahwah, NJ.
Lottaz, C. (1996). *Constraint solving, preference activation and solution adaptation in IDIOM*. Technical Report No. 96/204. Artificial Intelligence Laboratory, Swiss Federal Institute of Technology in Lausanne, Switzerland.
Maher, M.L., & Zhang, D.M. (1991). Case-based reasoning in design. In *AI in Design '91*, (Gero, J.S., Ed.), pp. 137–150. Butterworth-Heinemann Oxford, UK.
Medjdoub, B., & Yannou, B. (1996). A functional approach in architectural cad: Archiplan. In *Information Processing in Civil and Structural Engineering Design*, (Kumar, B., Ed.), pp. 31–43. Civil Comp Press, Edinburgh.
Mitchell, W.J., Steadman, J.P., & Ligget, R.S. (1976). Synthesis and optimisation of small rectangular floor plans. *Environ. Plann. B, 3*, 37–70.
Mittal, S., Dym, C.L., & Morjaria, M. (1986). PRIDE: An expert system for the design of paper handling systems. *IEEE Computer, 19(7)*, 102–114.
Motzkin, T.S. (1936). *Beiträge zur Theorie der linearen Ungleichungen.* PhD Thesis. University Basel, Switzerland.
Navinchandra, D. (1991). Exploration and Innovation in Design: Towards a Computational Model. In *Symbolic Computation, Artificial Intelligence*, Springer-Verlag, New York.
Purvis, L., & Pu, P. (1995). Adaptation using constraint satisfaction techniques. In *CBResearch and Development, Lecture Notes in AI 1010*, pp. 289–300. Springer-Verlag, New York.
Sadeh, N., & Fox, M.S. (1996). Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence, 86*, 1–41.
Sam-Haroud, D. (1995). *Constraint consistency techniques for continuous domains.* PhD Thesis No. 1423. Swiss Federal Institute of Technology in Lausanne, Switzerland.
Saund, E. (1989). Dimensionality-reduction using connectionist networks. *IEEE Trans. Pattern Analysis Machine Intell., PAMI-11(3)*, 304–313.
Schrijver, A. (1986). Primal-dual, elimination and relaxation method. *Theory of Linear and Integer Programming* (Graham, R.L., & Lenstra, J.K., Eds.) pp. 155–157. John Wiley & Sons, Chichester.
Smith, I., Stalker, R., & Lottaz, C. (1996). Creating design objects from cases for interactive spatial composition. In *Artificial Intelligence in Design '96*, (Gero, J.S., & Sudweeks, F., Eds.), pp. 97–116. Kluwer Academic Publishers, Boston.
Stalker, R. (1996). *The design and development of IDIOM-GUI, a graphical user interface for interactive design using intelligent objects and models.* Technical Report No. 96/211, Artificial Intelligence Laboratory, Swiss Federal Institute of Technology in Lausanne, Switzerland.
Tommelein, I.D., Heyes-Roth, B., & Levitt, R.E. (1992). Altering the sightplan knowledge-based systems. *AIEDAM 6*, 19–37.
Voss, A. (1994). The need for knowledge acquisition in case-based reasoning—some experiences from an architectural domain. In *Proc. Eleventh Eur. Conf. Artif. Intell.*, (Cohn, A.G., Ed.), pp. 463–467. John Wiley & Sons, New York.

Claudio Lottaz is currently a Ph.D. student at the Artificial Intelligence Laboratory of EPFL. He graduated at the Institute for Computer Science and Applied Mathematics at the University of Bern in 1994. His fields of interest include applications of artificial intelligence techniques such as constraint satisfaction in design and concurrent engineering.

Ruth Stalker is a Ph.D. student at the Institute of Structural Engineering and Mechanics, EPFL. She received a B.Sc.(Hons) in Computer Studies from the Liverpool John Moores University and a PGDip. In Applied Artificial Intelligence from Aberdeen University. She spent 2 years at the Laboratory of Artificial Intelligence, EPFL prior to her current position. Her research interests include the application of human-computer interaction and artificial intelligence to structural monitoring.

Ian Smith is a professor in the Civil Engineering Department at EPFL. He graduated from the University of Waterloo in 1978 and received his Ph.D. from the University of Cambridge, U.K. in 1982. He has taught, consulted, and performed research in AI applications, fracture mechanics, structural design and repair, code creation, HCI, CSCW, intelligent CAD, global sustainability, infrastructure monitoring, and performance-based structural engineering. He is the current Chair of the European Group for Structural Engineering Applications of Artificial Intelligence (EG-SEA-AI) and Chair of the working commission on information technology of the International Association for Bridge and Structural Engineering (IABSE). He is also an Associate Editor of *AIEDAM*.

https://doi.org/10.1017/S0890060498121133 Published online by Cambridge University Press