Intelligent CAD Systems III, Practical Experience and Evaluation,
P.J.W.ten Hagen P.J.Veerkamp (eds.)
Springer-Verlag, Berlin, 1991. pp.187-204.

13

Constraint Satisfaction Techniques for Spatial Planning

Can A. Baykan and Mark S. Fox

Center for Integrated Manufacturing Decision Systems
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA. 15213, USA

Abstract: WRIGHT is a CAD system for designing two dimensional layouts consisting of rectangles. This problem arises in space planning, i.e. the design of floorplans, arrangement of equipment in rooms, and site planning. Space planning is a search process characterized by very large search spaces. Constraint directed search provides a basic problem solving methodology for intelligent CAD by providing a formal method for representing domain knowledge uniformly as constraints, and by using constraints for efficient search. Each configuration is represented as a Constraint Satisfaction Problem (CSP), and constraint propagation restricts the domains of variables as information becomes available during search. Constraints reduce search complexity by choosing the most constrained decision. The measures that identify opportunistic decisions are variable tightness, constraint reliance and constraint tightness. Least commitment representations (value ranges) delay decisions until enough information becomes available. Abstractions reduce complexity by allowing abstract constraints to prune away entire design subsets.
Keywords: Space Planning, Search, Constraint Satisfaction, Least-commitment, Abstraction

1. Introduction

Intelligent CAD requires a fundamental problem solving methodology that can incorporate arbitrary amounts of knowledge in a principled manner. Constraint-directed search provides a formal method for representing expertise uniformly as constraints. From constraints an understanding of the structure of the problem (search) space, that leads to more efficient search, can be derived [1, 7, 8, 10]. Thus constraint-directed search addresses the needs of intelligent CAD by enabling the representation of knowledge from diverse sources and enabling the selection of efficient search strategies based on an understanding of search space structure.[1]

[1] This differs from encapsulating expertise in the form of rules in that rules do not provide for an understanding of problem space structure, but simply identify situations of applicability without any guarantee that the search being performed is efficient.

This basic problem solving methodology provides the framework within which the following issues can be addressed:

- knowledge representation,
- acquisition and maintenance of design expertise,
- user interface for graphical specification of constraints,
- user interface for interactive design.

WRIGHT is an intelligent interactive *space* planning system for generating two dimensional layouts consisting of rectangular shapes using constraint-directed opportunistic search. Space planning deals with the design of two dimensional layouts, such as floor plans, the arrangement of equipment in rooms and site planning. In space planning, topological relations and shape, dimension, distance and other functions of spatial arrangement are a principal concern. Almost all aspects of design have spatial implications, and influence space planning decisions.

2. Problem

WRIGHT deals with the generation of two-dimensional layouts consisting of configurations of rectangles. Inputs for generating a layout are:

- An existing layout which may be an empty space, and dimensions of which may be specified as ranges, as seen in Fig. 1.

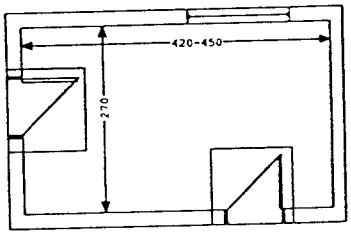

Fig. 1. Plan of kitchen showing existing layout

- Design units to be located and/or dimensioned, as seen in Fig. 2.

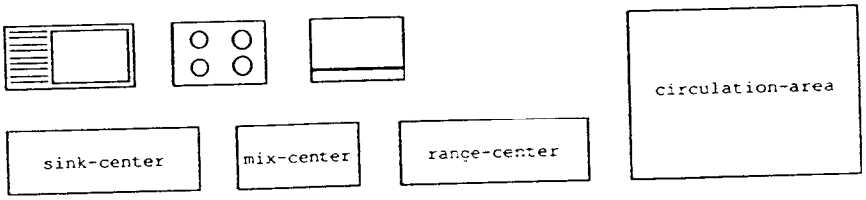

Fig. 2. Design units to be dimensioned and located in kitchen

- Knowledge about the design domain in the form of a class hierarchy of prototype design units and constraints on them, as seen in Fig. 3.

The output of WRIGHT is a set of optimal layouts that are significantly different from each other, as seen in Fig. 4.

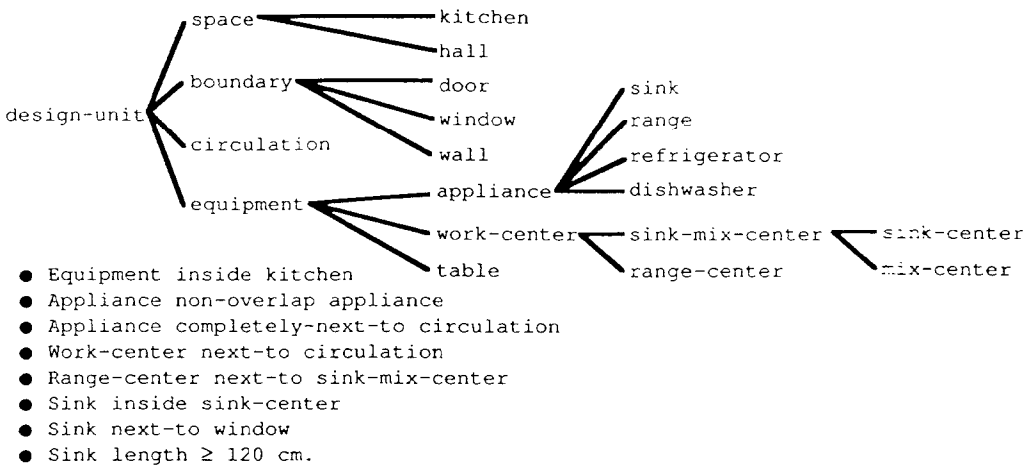

Fig. 3. Class hierarchy of kitchen design units and some kitchen constraints

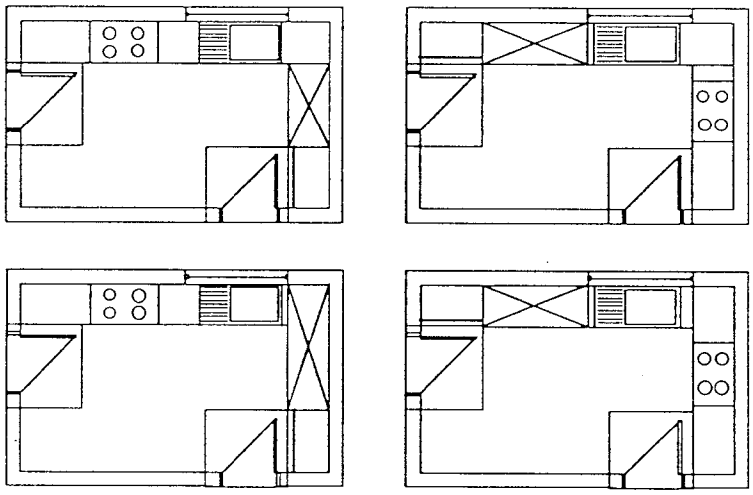

Fig. 4. Solutions to the kitchen design problem defined above

3. Background

Based on their underlying representations, previous approaches to spatial layout can be classified as *grid based, drawing based* and *relational*. Grid based representations partition objects to be located into subparts of equal area and divide the site into a grid of cells where each cell is equal in area to one subpart. Drawing based representations use polygons of fixed size and shape to represent objects. A polygon is represented as a set of sides, and a side as a set of points. Relational representations use adjacency or incidence between points, between lines and regions, or between regions to model layouts.

In space planning, search operates by selecting a spatial element(s) and an operator, and generates a new configuration by applying the operator to the element(s) in some state. Structuring the elements of search gives rise to different strategies. Starting search with an empty initial configuration results in a *build-up strategy*. There are two basic variations in a

build-up strategy: *organize by element* and *priority* solution methods [3]. In an organize by element strategy, the next object to enter the layout is selected, placed at alternative locations and tested. All relevant attributes of the object are determined at the time it enters the configuration, and all applicable tests are carried out to select satisfactory locations. Search continues by selecting a new object to enter the design. A priority strategy orders search operators as in ABSTRIPS [16] and other hierarchical planning systems. Operators with high priority are applied first, creating macro objects or configurations in unbounded space by determining the important attributes first. In an *improvement strategy,* search starts from a configuration which contains all the elements. Changes are made in response to failing constraints or in order to improve the score of an objective function.

Quadratic assignment formulation [12] uses a grid based representation. This representation can not deal with variable sizes, and makes it very hard to deal with issues of shape and alignment. Both build up and improvement strategies are used with this approach.

DPS [15] and GSP [3] use drawing based representations. GSP objects must be rectangles and DPS objects can be arbitrary polygons. In both systems dimensions of objects must be fixed. GSP uses an organize by element strategy, whereas DPS can also employ a priority strategy. In systems using drawing based representations, locations tried for placing an object depends on the existing layout, as seen in Fig. 5. As a result of this, configurations generated depend on the order in which objects enter the layout. Since GSP and DPS try only one ordering, they may miss possible solutions. Their correctness is not guaranteed. Locations in Fig. 5, from ([3]; p.57), are defined by lines projected by the edges of the space and the objects that are in place. Placing an object at every location above, in four possible orientations, results in 96 new configurations.)

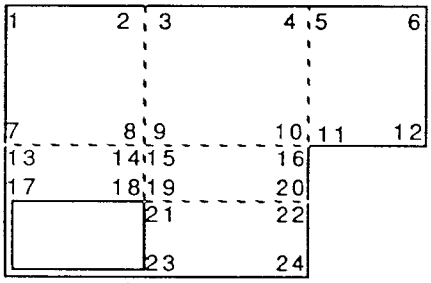

Fig. 5. Locations considered by GSP for placing the next design unit

GRAMPA [9], DIS [4] and LOOS [6] are space planners that use a relational representation. GRAMPA uses adjacencies between regions, and DIS and LOOS use the adjacencies between regions and lines. The relations used in LOOS are north-of and east-of, as seen in Fig. 6. Using these relations it is not possible to specify that rectangle 1 is next-to rectangle 2 – whether they are adjacent or not depends on the dimensions. The same is true of specifying alignment. Such relational considerations are treated as dimensional issues.

Relational layout programs systematically generate all distinct configurations defined by their representation. They use a two step process that deals with relational and dimensional aspects separately, thus are not opportunistic in the use of constraints. The relations that are

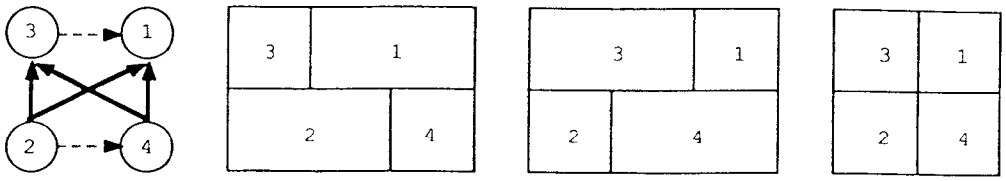

Fig. 6. An orthogonal structure and possible configurations represented by it

used for generating solutions are not based on the requirements of the problem, but on a restricted set of relations defined in each system.

WRIGHT uses relations between lines for expressing topology. It is possible to create new spatial relations by defining them in terms of relations between lines. The strategy employed in WRIGHT is a priority strategy, where search operators determine only the attributes specified by the selected constraint. The topology and dimensions of a configuration can be decided in any order due to the CSP formulation used.

4. Insights and Approach

Design is the process of constructing a description of an artifact that satisfies a functional specification, meets explicit or implicit performance criteria, is realizable and satisfies restrictions on the design process itself [13]. The artifact is initially defined by its desired properties, and it is natural to express them in terms of constraints. In WRIGHT, any knowledge that defines or restricts the domain is expressed as a constraint.

Space planing is a search process [2] characterized by very large search spaces. Constraints play a major role in reducing search complexity, by opportunistically choosing the most constrained decision to make at each step [7]. Constraint propagation restricts the set of alternatives by eliminating values that are inconsistent with the current decision. WRIGHT uses constraints to select an efficient search path, and uses constraint propagation to ensure that values of all variables are consistent.

Design representations should tolerate ambiguity and incomplete specification. They are different from representations of existing situations. Least commitment [16] representations delay decisions about uncertain aspects, while making it possible to reason about the certain aspects of designs. Least commitment is achieved by using value ranges, abstract constraints, and generating only the attributes specified in constraints.

Abstractions can further reduce complexity by allowing abstract constraints to prune entire design subsets [14]. Abstract constraints bound the solution space, and objects at different levels of aggregation simplify search in WRIGHT. Decisions are not forced by the representation used. When a commitment is made, it is always for satisfying a constraint. Attributes of design not specified by the selected constraint are deferred.

The goal of this research is to identify and elaborate the semantics of constraints to express knowledge of the problem domain, to identify the knowledge that enables efficient search decisions, and to find the limitations, applicability and performance of this approach with respect to type and structure of constraints.

5. Representation

The elements of WRIGHT are design units (objects), spatial relations and constraints. Knowledge is represented by a class hierarchy of prototype design units, and constraints specifying desired relations between design units or restrictions on their attributes such as length or orientation. *Abstraction by aggregation* combines design units into larger objects, such as combining a set of rooms into a house; or a sink, dishwasher, and counter-top area into a sink-center. WRIGHT can handle problems involving design-units at different levels of aggregation.

5.1. Objects

There are two types of objects used in WRIGHT for representing layouts: *numerical variables* and *structured objects*. Structured objects contain other structured objects or numerical variables. Numerical variables are *interval* or *discrete*. Design units and rectangles are structured objects. Lines, dimensions, areas, and distances are interval variables. Orientations are discrete variables.

Design unit is the building block of configurations. A design unit is a structured object made of two components: a rectangle and an orientation, as seen in Fig. 7. Design units have fronts, backs and sides. Orientation of a design unit specifies which way its front is facing. The counterclockwise angle between the front of the design unit and the y-axis is its orientation. A rectangle is a structured object consisting of four lines, length, width, and area.

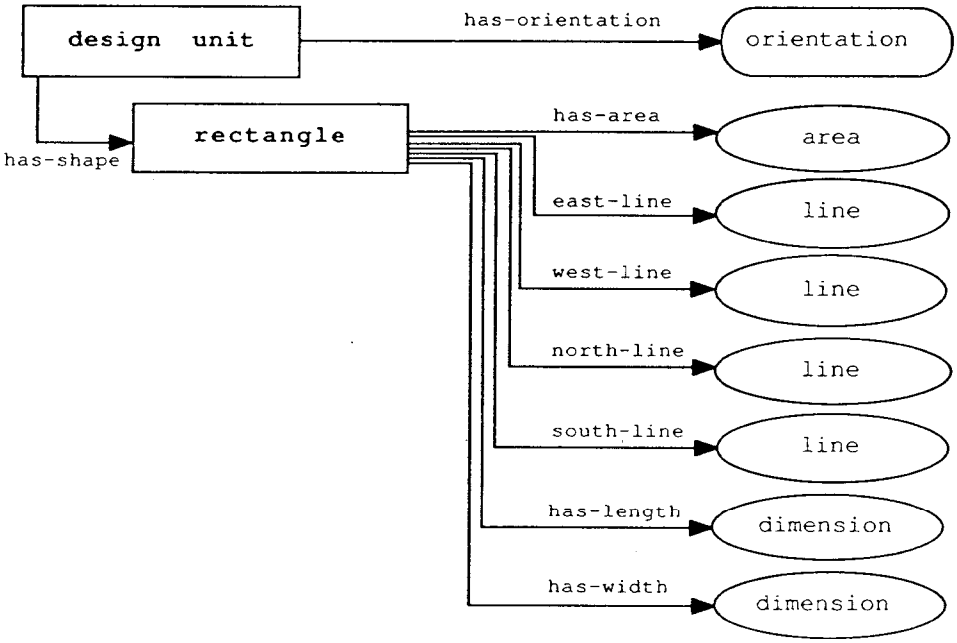

Fig. 7. Structured objects and variables for representing rectangular design units

Discrete variables take a single value selected from a set of allowable values. Orientation is discrete variable, and the set of allowable values for it are: { 0, 90, 180, 270 }. Interval variables have a range of values defined by a minimum and a maximum. For vertical lines, minimum and

maximum values bound the x-coordinate of the line. For horizontal lines, they bound the y-coordinate.

5.2. Relations

Spatial relations express the types of configurations that are of interest in space planning problems. Spatial overlap and adjacency relations are topological relations where orientations or relative locations with respect to global or object centered coordinates are not considered.

- **Spatial overlap:** inside, contains, overlaps, non-overlapping, one-dimensional-overlap.
- **Adjacency:** next-to, completely-next-to, covers.

Location and alignment relations listed below are defined with respect to the global directions: north, south, east, and west. Orientations of the design units are not considered. For example, align-north specifies that the north sides of two design units are to be aligned, regardless of their orientations.

- **Location:** north-of, south-of, east-of, west-of.
- **Alignment:** align-north, align-south, align-east, align-west.

The following spatial relations are defined with respect to object centered coordinates, and depend on the orientation of one or both design units.

- **Orientation**: parallel-to, perpendicular-to, opposite.
- **Relative location**: in-front, at-back, at-left, at-right.
- **Relative alignment**: align-front, align-back, align-left, align-right.
- **Relative distance**: front-distance, back-distance, left-distance, right-distance.

Orientation relations specify relative orientations of two design units. If the orientations are equal, the design units are parallel. If the orientations differ by 90°, the objects are perpendicular. If the orientations differ by 180°, the objects are opposite. Relative location relations specify the location of design unit 2 with respect to design unit 1. Relative alignment relations specify that sides of two design units are collinear. Relative distance relations specify the distance from the specified side of design unit 1 to the closest side of design unit 2.

Spatial relations between design units are defined in terms of and/or combinations of algebraic relations between their lines. The algebraic relations are: $=$, $>$, $\geq$, $+$, and $\times$. The statement vertical-line1 > vertical-line2 means that vertical-line1 is to the east of vertical-line2, since the value assigned to a vertical line specifies its x-coordinate.

There is a grammar for defining spatial relations, which maps the relation into disjunctive and conjunctive combinations of algebraic relations between numerical variables or constants. As a result of observing the problem solving behaviour of WRIGHT on kitchens, we have identified new spatial relations such as: completely-overlapping, one-dimensional-overlap, at-side in addition to the ones we have started with and included them in the system.

5.3. Representing layouts

A layout is represented by a set of variables, their values, and algebraic relations that should always be satisfied. This defines a Constraint Satisfaction Problem (CSP). A CSP has a set of variables, each with an associated domain, and a set of constraining relations, each involving a

subset of the variables. The variables in the CSP representation of a layout are the lines, dimensions, areas and orientations of the design unit instances, plus other variables that indicate distances or amounts of overlap. Types of relations are: =, >, ≥, +, and ×. The first three relations are either unary relations between a variable and a constant or binary relations between two variables. The last two are ternary relations between three variables.

The constraining relations in a CSP are defined by rectangles on their component variables, and by topological and geometrical relations between design units. A rectangle defines constraints between its lines, dimensions and area, as seen in Fig. 8.

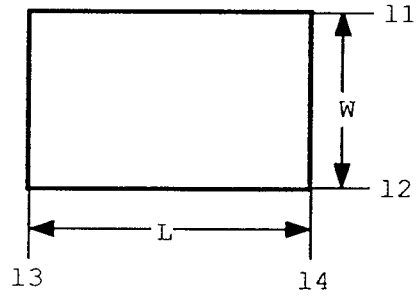

Fig. 8. Constraining relations defined by a rectangle

Setting up relations between design units results in algebraic relations between their components. In Fig. 9, the sink defined by lines ln1, ln2, ln3, ln4 is placed next-to and south of the window defined by lines ln5, ln6, ln7, ln8. This configuration should satisfy the constraint that sink is next-to the window for ≥ 50 cm. The algebraic constraints seen in Fig. 9 are added to the CSP as a result. Variables v1 and v2 are created for expressing the adjacent distance between sink and window.

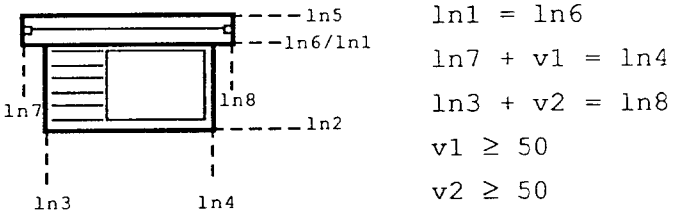

Fig. 9. Constraining relations defined by a configuration

5.4. Constraints

Any knowledge that defines or restricts the domain is a constraint. Constraints indicate restrictions or desired relations. In WRIGHT constraints are of the form:

- <object> <relation> <object>, or
- <variable> <algebraic relation> <number>.

Constraints expressing domain knowledge are posted to prototype design units, as seen in Fig. 3. These are called *domain constraints*. In addition to specifying a relation or a restriction on some value, a constraint expresses knowledge about how it is to be used during search. This

knowledge is expressed by *similarity* and *relaxation* relations between constraints and *combining-instances*, *importance*, and *utility* values.

When it is posted, each constraint is assigned an *importance* value specifying the importance of satisfying that constraint in a solution. All constraints are binary, so they are either satisfied or contradicted. Solutions are rated by subtracting the importance values of failing constraints. *Relaxations* of constraints are specified by other constraint expressions that specify alternative relations, alternative design units, or looser bounds on numerical variables. Relaxations are tried when a constraint can not be satisfied in its original form. States where a constraint is relaxed are assigned lower ratings as specified by the *utility* of the relaxation.

Design knowledge is expressed in terms of required spatial relations in WRIGHT. Consider the relationship of the sink to windows: "The average housekeeper spends nearly 1 and 1/4 hours at the sink each day so there is a good case for putting the sink at a window for good light and view." ([11]; p.72) One way of satisfying the requirements is placing the back of the sink completely next to the window, which is expressed by the following constraints:

- *Sink completely-next-to window*
- *Sink at-back window*

When it is not possible to put the sink completely next to the window, placing it in front of and perpendicular to the window will allow direct light and a view of outside. The sink must also be close enough to the window. The following constraints express this case:

- *Sink distance window, max=120 cm.*
- *Sink one-dimensional-overlap window, min=30 cm.*
- *Sink perpendicular-to window.*

Distance is measured between closest points. One dimensional overlap means overlap in either the vertical direction or horizontal direction. The second set of constraints are a relaxation of the first set, and have lower utility values.

When there is more than one window, the constraints between sink and window should be satisfied for one window only. Whether the constraint should apply to all windows or just one window is indicated by *combining-instances* value attached to the constraint or to the relation specified in the constraint. There are defaults for relations, i.e. for non-overlap the default value for combining-instances is *and* whereas for most other relations the default is *or*.

Similarity is a relation between two constraints which restricts the way the constraints apply in case there is more than one instance. In the example above, sink should be completely next to and have at its back the same window. This is expressed by posting a *similarity* condition between the two constraints.

5.5. Constraint graph

The constraint graph is an and/or network that refines design knowledge represented by constraints on prototype design units into a design specification represented as combinations of algebraic constraints on the components of the design unit instances.

A constraint graph consists of nodes and links as seen in Fig. 10. (The constraint *sink next-to window* is mapped to constraints on the components of sink1 and window1.) It is an

abbreviated graph, where a constraint in the CSP and the variables it connects, such as: line1=line2, are represented by a single node. Nodes are of two types: and-nodes, or-nodes. And-nodes are expressed by connecting the links leaving that node by an arc. The links in the constraint graph indicate *reliance* between constraints.

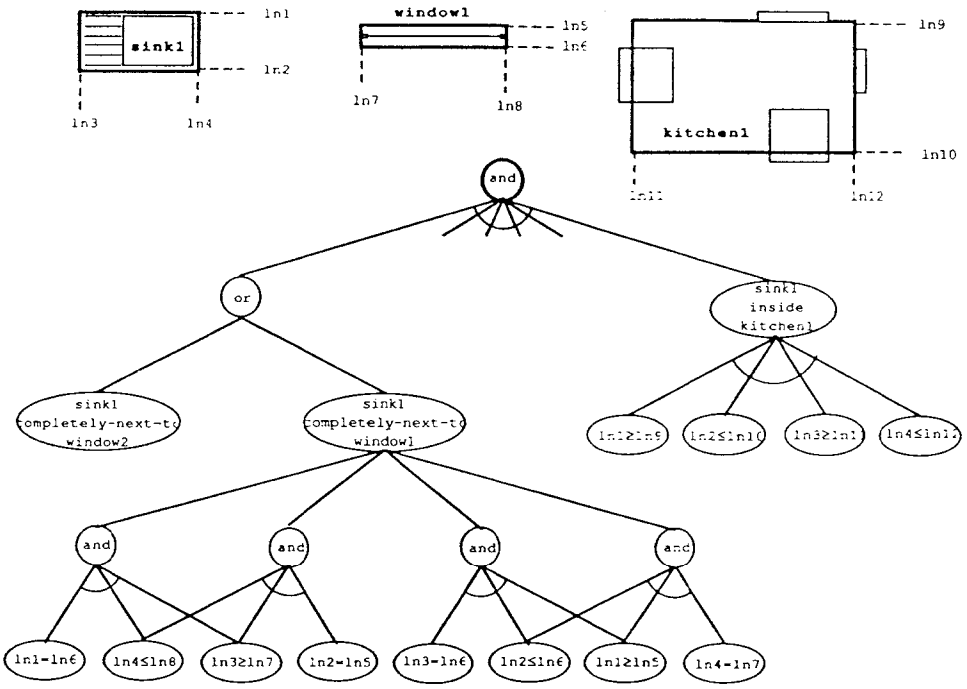

Fig. 10. Partial constraint graph

The constraint graph specifies alternative ways of satisfying a constraint. Prototype design units that have more than one instance and spatial relations that can be satisfied in different ways introduce disjuncts to the constraint graph. The top level of the graph is in *conjoint normal form*.

When there are conditions which hold true in all the alternatives, they can be used to bound solutions without committing to a specific alternative. These are called abstract constraints. Abstract constraints exist for adjacency and distance relations, and for dimensional constraints. Fig. 11 shows the abstract constraints for the constraint graph seen in Fig. 10. The configuration in the figure shows the location defined by the abstract constraints for the sink when the location of the window is fixed.

6. Search Architecture

The search architecture used in WRIGHT operates by selecting a node connected to the top level of the constraint graph and satisfying it in all possible ways. Operators are associated with constraints as in means-ends analysis. The variables indicated by the constraint are assigned new values, constraints are propagated, and any constraints that have only one way of being satisfied are also satisfied. Dealing with variables that are not affected is deferred until later. This is a priority strategy where *texture measures* select the constraint to be satisfied. The

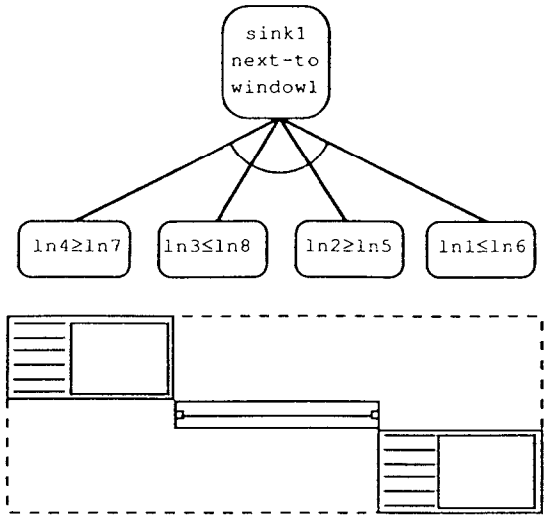

Fig. 11. Abstract constraints for *sink1 next-to window1* and resulting bounds on location of sink

constraint graph provides a mesh over the problem space and is used for generation and testing. Texture measures provide an approximation of problem space topology that allows search to be focused.

6.1. Focus of attention algorithm

Given an initial configuration and design units to locate and dimension, search starts by compiling the constraint graph. Every search state contains a CSP representation of the layout, and a constraint graph with values based on the CSP. New layouts are generated by selecting some constraints from the constraint graph, and adding them to the CSP. Search is controlled by the focus of attention algorithm, which operates as follows:

1. The state to continue search from is selected using a *best first* strategy.
2. In a selected state, one of the constraint nodes connected to the root of the constraint graph is selected, based on the problem space textures described below.
3. A new state is generated for satisfying each disjunct of the selected constraint. In each new state, constraints that contain no disjuncts are satisfied. Satisfying a constraint can determine values of disjuncts in other constraints, therefore this step loops until quiescence.
4. When the status of all constraints in the conjoint normal top level of the constraint graph are determined, that state is a solution. States are rated by subtracting importances of violated constraints at top level of constraint graph.

6.2. Texture measures

WRIGHT uses three texture measures for selecting constraints. The first measure is from a variable perspective, and the last two are from a constraint perspective. The texture measures are:

- **Variable Tightness:** is approximated by the number of remaining conjunctive constraints on each design unit instance.
- **Constraint Reliance:** is approximated by *1/number of disjuncts*. Constraints with fewer disjunctive cases remaining are selected. If the number of disjuncts is 1, the constraint is satisfied without needing to generate a new state for each disjunct.
- **Constraint Tightness:** is approximated by the reduction in the domains of continuous variables involved, as a result of satisfying the constraint. Constraints that result in large reductions are favoured. Types of algebraic relations that will be added to the CSP due to each of the competing constraints are also taken into account.

Texture measures are applied lexicographically. Active constraints are assigned ratings with respect to a metric, and constraints with lower values are eliminated from contention. If there is a single constraint with the best measure, it is used. If more than one constraint remains, the next texture measure is applied, or a constraint is selected randomly.

6.3. Search operations

Each state contains a different layout and a different constraint graph, as seen in Fig. 12. When generating a state, the layout is changed by operators that satisfy a constraint. The operators carry out one or both of these actions:

- add new relations between objects,
- assign minimum or maximum values to line locations, dimensions, areas, and distances.

As a result of adding new relations or values, constraints are propagated to ensure that the values are globally consistent. Constraint propagation removes inconsistent values at continuous variables by increasing the minimums and decreasing the maximums. The constraint graph is changed by marking the nodes as *satisfied* or *contradicted*. These labels propagate up through the constraint graph according to rules of propagation defined for and-nodes and or-nodes.

Properties of the search architecture and representation of WRIGHT are:

- Search is *monotonic*. States are generated by adding new relations to the CSP, Therefore a requirement that is satisfied can not be violated later.
- Disjuncts specified in the constraint graph are *mutually exclusive*. Therefore, it is not possible to get duplicate solutions.
- Search efficiency depends on the order constraints are satisfied. Satisfying a set of constraints at the leaves of the goal tree in any order leads to the same solution.

Each solution satisfies a different subset of the constraints at the leaves of the constraint graph. Fig. 4 shows the set of solutions for the kitchen seen in Fig. 1. Mix-center is the rectangle with diagonals. In the two solutions on the left, the order of design units clockwise starting from top left is: range-center, sink-center, mix-center and refrigerator. The configuration of sink-center and mix-center at the top left corner of the kitchen is different between the solutions on the top left bottom left in Fig. 4. In the two solutions on right, the order of design units clockwise starting from top left is: refrigerator, mix-center, sink-center and range-center. The difference between the top and bottom solutions is again the configuration at the corner.

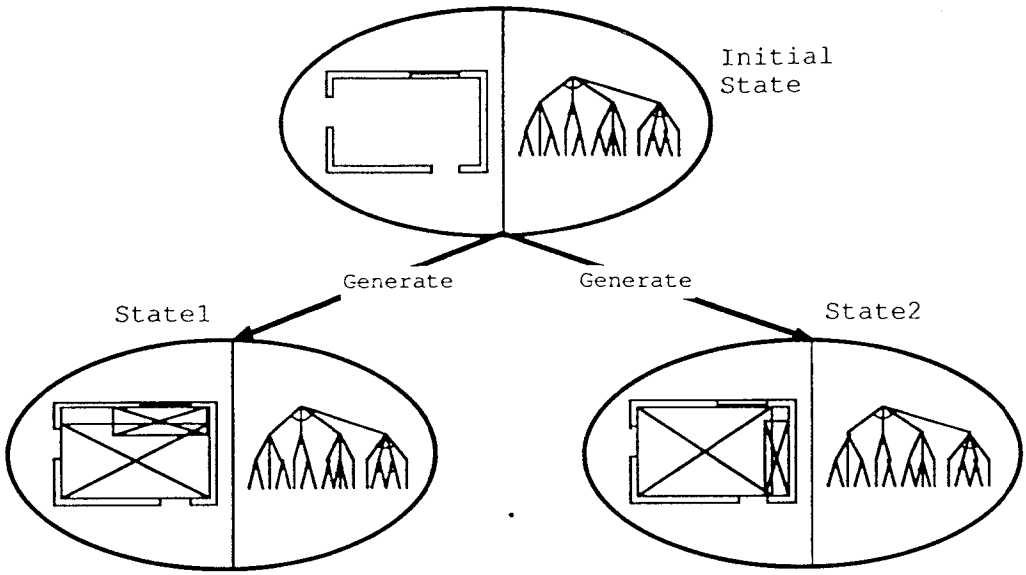

Fig. 12. Search states

WRIGHT uses constraints to define significant differences between alternatives. In this case significant differences are defined by different adjacencies and by different ways of placing the objects adjacent to each other at corners.

The attributes of the layout that are not constrained are treated as unimportant. This permits solutions at a higher level of abstraction than in other space planning systems, while enabling exact determination of relevant aspects. Unless the constraints force the assignment of a unique value, a variable has a range of values even in a solution. It is possible to refine a solution further by assigning unique values to all variables.

6.4. Performance

Texture measures reduce search. As a result, WRIGHT looks at a smaller number of states. Fig. 13 shows the number of search states required for finding all solutions to five kitchen layout problems, under different combinations of texture measures. The combinations tested are:

- *method 0:* select a constraint at random,
- *method 1:* variable tightness,
- *method 2:* constraint reliance,
- *method 3:* variable tightness and constraint reliance,
- *method 4:* variable tightness, constraint reliance and constraint tightness.

When a combination of measures is used, they are applied in the order: variable tightness, constrain reliance, constraint tightness. Each measure eliminates some constraints from consideration. If more than one constraint remains after applying the texture measure(s), specified by the method, a constraint is selected at random. The number of states given for each problem-method combination is the average of three runs. In the second problem, method 4 reduces search by more than 80% compared to method 0, and in the third problem by 35%.

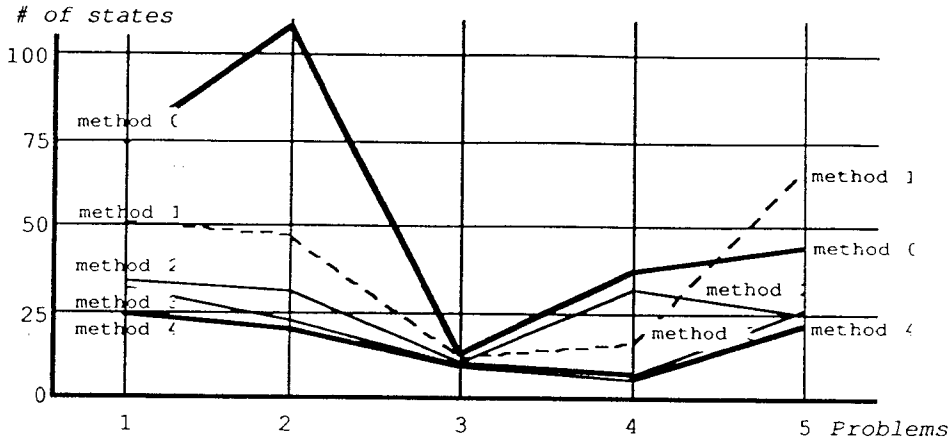

Fig. 13. Effectiveness of texture measures in reducing search

In order to compare the search plus filtering approach with generate and test, WRIGHT is compared with two space planning programs: DPS [15] which uses a drawing based representation, and LOOS [5] which uses a relational representation.[2] The problem used in the comparison is arranging six fixed size blocks in a box such that no blocks overlap. Due to the simplicity of the problem, exactly the same set of constraints can be used by all three programs. The programs are compared in terms of the number of states and search plies[3] generated when finding the first solution and when finding all 24 solutions, seen in Table 1.

	First Solution		All 24 Solutions	
WRIGHT	5 plies	14 states	5-6 plies	119 states
LOOS	6 plies	68 states	6 plies	232 states
DPS	6 plies	72 states	*(not available)*	

Table 1. Comparison of WRIGHT, DPS and LOOS in terms of search efficiency

In DPS and LOOS, the number of search plies is always equal to the number of objects to be located, as a result of the organize by design unit strategy. WRIGHT's performance in terms of number of search plies and number of search states depends on number and strength of available constraints and their interactions. Although the constraints in this problem are not as varied as in kitchen layout, WRIGHT performs better than DPS and LOOS. WRIGHT looks at a smaller number of search states by selecting decisions with fewer alternatives, and by eliminating inferior alternatives earlier.

Another area of comparison is how search behaviour changes when problems are under or over constrained. In an underconstrained problem, DPS and LOOS find the first solution faster, but there will be a large number of solutions. WRIGHT also finds the first solution faster, and will avoid generating a large number of solutions by having solutions at a higher level of

[2] See §3 **Background** for a discussion of these programs.

[3] The number of search plies is the number of intermediate states on a path from the initial state to a solution state.

abstraction. In an overconstrained problem, DPS will not be able find any solutions because it rejects a solution that fails any constraint. For LOOS, overconstrained problems pose the same difficulty as underconstrained ones: too many states with equivalent scores. Finding the first solution will take much longer too. Overconstrained problems will cause WRIGHT to search longer before finding the first solution. When all constraints can be satisfied, solutions are defined by alternative ways of satisfying all constraints. When all constraints can not be satisfied, combinations of constraints that result in equal ratings need to be tried. By defining explicit relaxations for some domain constraints in its knowledge base, WRIGHT avoids searching a large number of constraint combinations.

7. User Interface

WRIGHT's model of the design process and the division of labour between system and user is that the user inputs the problem, identifies, modifies or relaxes constraints, and the system finds solutions. The user may also make search control decisions.

The user interface provides a menu-driven, graphical means for designing, posting constraints and carrying out search. The user interface is designed to facilitate:

- examining partial solutions,
- adding missing constraints,
- relaxing overly restrictive constraints,
- selecting which partial solution to pursue next,
- observing search decisions and reasons for it.

A change such as adding or removing a constraint and bounding the location of a design unit takes place in the current search state and only affects states that are generated from it.

Fig. 15 shows three windows: for graphics, for the search tree, and for text. There is a command menu in the top left corner. When the user selects a command by clicking on it with the mouse, menus pop up for selecting the parameters of the command. A pop-up menu for selecting design unit instances is seen in Fig. 15, superimposed on the plan.

The **select-state** command brings the search tree window in the bottom right corner of the screen to the top, and enables the user to select a state by pointing and clicking with the mouse. The configuration in the selected state is displayed in the graphics window, and following operations take place in that state. The next two commands, **interactive-search** and **exhaustive-search**, are for generating a set of new states from the current state and for carrying out exhaustive search.

There are commands to **create**, **size**, **locate**, and **orient** design units. The designer can define a rectangle by clicking at its top left and bottom right corners in the graphics window using the mouse. Rectangles are used to input minimum size, maximum size and location of objects. It is possible to think of a rectangle as a constraint, because it indicates bounds. During interactive sizing and locating operations WRIGHT will not allow the user to violate existing bounds on a design unit. For relaxing bounds, one needs to move up in the search tree to a state where those variables have looser bounds. Constraints specifying relations between design units at any level of the class hierarchy, including particular instances of design units, are posted by

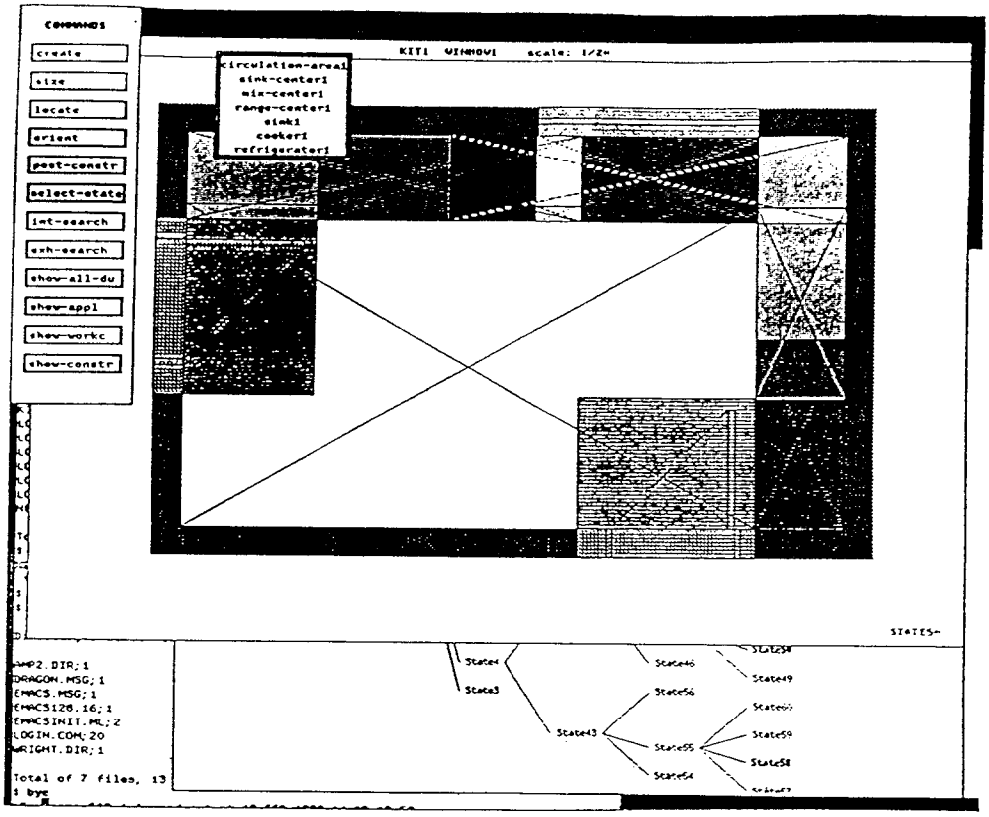

Fig. 15. WRIGHT's user interface

first selecting **post-constraint** from the command-menu, and then selecting a design unit, a relation and another design unit from pop-up menus.

The rest of the commands are for displaying information: displaying all design units – as seen in Fig. 15, displaying only the appliances, and displaying only the work centers. **Show-constraints** command displays the constraints that have not been satisfied yet, in the text window at the bottom left hand corner of the screen. It is also possible to inspect the database and call functions, as the text window is the lisp listener.

8. Conclusion

WRIGHT has been tested on kitchen design, house layout and blocks problems. The representation scheme used in WRIGHT is more flexible than other space planning systems in two respects:

- It allows design units at different levels of aggregation.
- It enables declaratively defining new relations.

If, in some domain of application, we identify new relations that express the conditions we are interested in, they can be defined as combinations of =, >, ≥, +, and × relations between lines and can be used in constraints.

The opportunistic constraint directed search approach of WRIGHT leads to significant increases in performance relative to other spatial planning techniques. This is due to two factors. First, WRIGHT uses knowledge of constraints to develop a mesh over the problem (search) space. Propagation of restrictions occurs within the mesh resulting in a reduction in size of the problem space. Second, WRIGHT opportunistically selects variables to constrain once propagation ends. Knowledge of problem space 'texture' is used to identify the appropriate variable. Texture measures used are: *variable tightness*, which chooses a design unit having a large number of constraints; *constraint reliance* which selects a constraint having fewer disjunctive alternatives, and *constraint tightness* which selects a constraint that reduces the domains of its variables most [8].

In summary, the philosophy behind this approach is to use constraints to understand the structure of the search space to make search efficient. WRIGHT generates fewer alternatives as a result of selecting decisions opportunistically, avoiding premature commitment, and eliminating inferior alternatives earlier. Constraints guide generation of significantly different alternatives. Insignificant aspects of the design do not cause alternatives to be generated while relevant differences are explored in all possible ways.

References

1. R. Dechter and J. Pearl, "Network-Based Heuristics for Constraint-Satisfaction Problems," *AI*, vol. 34, no. 1, pp. 1-38, 1988.
2. C.M. Eastman, "On the Analysis of Intuitive Design Processes," in *Emerging Methods in Environmental Design and Planning*, ed. G.T. Moore, MIT Press, Cambridge, MA, 1970.
3. C.M. Eastman, "Automated Space Planning," *AI*, vol. 4, pp. 41-64, 1973.
4. U. Flemming, "Wall Representations of Rectangular Dissections and their Use in Automated Space Allocation," *Environment and Planning B 5*, pp. 215-232, 1978.
5. U. Flemming, "On the Representation and Generation of Loosely Packed Arrangements of Rectangles," Tech. Rept. DRC-48-05-85, Carnegie-Mellon University Design research Center, 1985.
6. U. Flemming, M.D. Rychener, R.F. Coyne, and T. Glavin, "A Generative Expert System for the Design of Building Layouts," Center for Art and Technology, CMU, Pittsburgh PA, 1986.
7. M.S. Fox, "Observations on the Role of Constraints in Problem Solving," *Proceedings Sixth Canadian Conference on Artificial Intelligence*, pp. 172-187, 1986.
8. M.S. Fox, N. Sadeh, and C. Baykan, "Constrained Heuristic Search," *Proceedings of IJCAI-11*, pp. 309-315, 1989.
9. J. Grason, "Methods for the Computer-Implemented Solution of a Class of Floor Plan Design Problems," Ph.D. Th., Carnegie-Mellon University, 1970.
10. R.M. Haralick and G.L. Elliott, "Increasing Tree Search Efficiency for Constraint Satisfaction Problems," *AI*, vol. 14, pp. 263-313, 1980.
11. Architects Journal, "Domestic Kitchen Design: Convential Planning," *Architecs Journal*, pp. 71-78, 1984.

12. J.C. Koopmans and M.J. Beckmann, "Assignment Problems and the Location of Economic Activities," *Econometrica*, vol. 25, pp. 53-76, 1957.

13. J. Mostow, "Towards Better Models of the design Process," *AI Magazine*, vol. 6, no. 1, pp. 44-57, 1985.

14. A. Newell, J.C. Shaw, and H.A. Simon, "The Processes of Creative Thinking," in *Contemporary Approaches to Creative Thinking*, ed. H.E. Gruber, G. Terrel, and J. Wertheimer, Atherton, 1962.

15. C. Pfeffercorn, "Computer Design of Equipment Layouts Using the Design Problem Solver," Ph.D. Th., Carnegie-Mellon University, 1971.

16. E.D. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," *AI*, vol. 5, pp. 115-135, 1974.