Spatial Reasoning and Constraint Satisfaction: The Other Way Round

Hans W. Guesgen and **Ute Lörch**
Computer Science Department, University of Auckland
Private Bag 92019, Auckland, New Zealand
phone: +64-9-373-7599, fax: +64-9-373-7453
email: {hans, ute}@cs.auckland.ac.nz

Abstract

Spatial (and temporal) reasoning and constraint satisfaction are closely related in that many approaches use a constraint satisfaction system for reasoning about networks of spatial or temporal information. Unlike these approaches, this paper describes how spatial reasoning can be used to enhance constraint satisfaction systems. In particular, it applies fuzzy spatial reasoning to constraint graphs to extract the relevant information from these graphs and to convert it into a format that can be used as input to a constraint satisfaction system.

Introduction and Motivation

Over the recent years, constraint satisfaction has become a popular problem solving technique in the area of spatial and temporal reasoning (Allen 1983; Anger & Rodriguez 1995; Guesgen & Hertzberg 1993). Information about time and space is often encoded in a network of temporal and spatial constraints, and constraint satisfaction algorithms are used to infer new information from the information given by the network.

Since constraint satisfaction is a general technique with applications beyond spatial and temporal reasoning, various authors have made attempts to implement constraint satisfaction systems that are independent of particular domains. They usually offer a variety of constraint satisfaction methods that can be configured to suit a particular application. Most commercial constraint satisfaction systems also provide a reasonable user interface to input constraints and constraint networks into the system, but non-commercial systems like CSP (Manchak & van Beek 1994) or CONSAT (Guesgen 1989) often lack this feature. When using a system without a user interface that specifically supports constraint and constraint networks, editing constraint is likely to become an erratic task.

One of the easiest ways for humans to denote constraint networks is by drawing constraint graphs. Such a representation has several advantages:

- Information is presented in two different ways. The connections between variables and constraints, i.e., which constraints involve which variables, are represented by graphical elements, whereas the names of the variables and the constraints, the domains of the variables, and the relations of the constraints are represented by text.
- Constraints are represented according to their spatial relationship. Constraints that are in the same neighborhood, i.e., share a variable with each other, are represented closely together, whereas constraints that are connected only indirectly via other constraints are represented further apart.

In this paper, we will discuss how constraint graphs can be converted automatically into a format that is readable by a constraint satisfaction system. The idea is to apply fuzzy spatial reasoning to the constraint graphs to extract information about variable domains, constraint relations, connections between variables and constraint, etc. from the graphs. This task appears to be trivial, but in fact it is not. If the graphs are produced by a human, it is very likely that there are imperfections in the graphs (Chok & Marriott 1995), which means that a simple parsing procedure would fail. To solve this problem, we introduced heuristics that can deal with the fuzziness in user-generated constraint graphs.

For the rest of this paper, we will use XFIG as the graphic editor for drawing constraint graphs and CONSAT and CSP as the constraint satisfaction systems.[1] We will introduce the program LIGECS which, given a constraint network designed with the graphic editor XFIG, converts the network into two different formats:

[1] The main reasons for choosing XFIG as the graphic editor were that it runs on many different platforms and that it is a suitable editor for drawing constraint networks. The reasons for choosing CONSAT and CSP as the constraint satisfaction systems were their availability in the public domain.

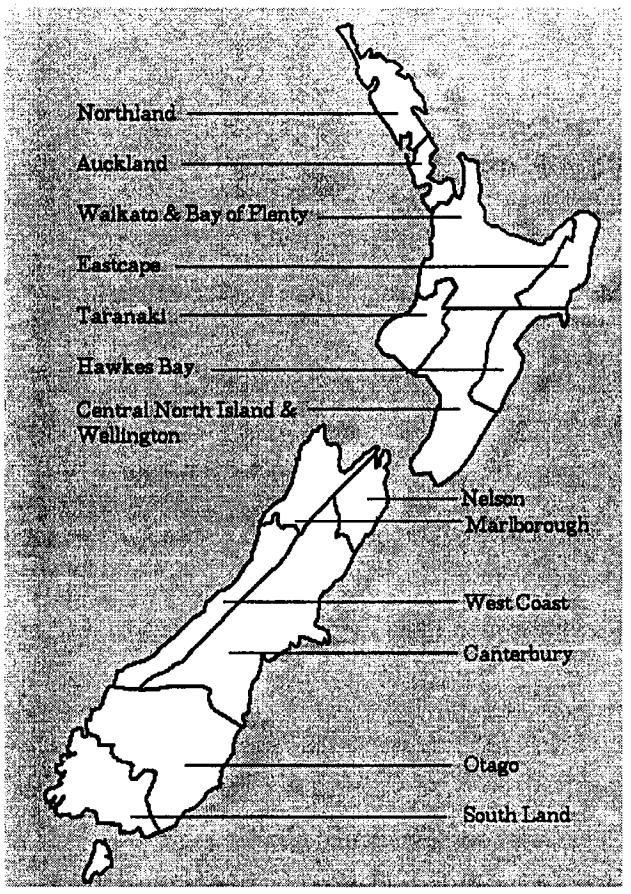


Figure 1: Coloring a map of New Zealand.

1. Lisp code that can be used as input to the constraint satisfaction system CONSAT.
2. C code that can be used as input to the constraint satisfaction system CSP.

From XFIG to CSP/CONSAT

Let us consider the following constraint satisfaction problem. A map of New Zealand, like the one shown in Figure 1, is to be colored with the colors red, yellow, and green in such a way that adjacent region have different colors and that the Waikato and Nelson region are colored green.[2]

There are at least two ways of representing the corresponding constraint network as a graph. Since all constraints in this problem are binary constraints, one can represent the constraints as edges in the graphs and the variables as nodes connected by these edges

[2]This problem seems quite trivial and its solution requires only little thought, but it serves the purpose of demonstrating how LIGECS works.

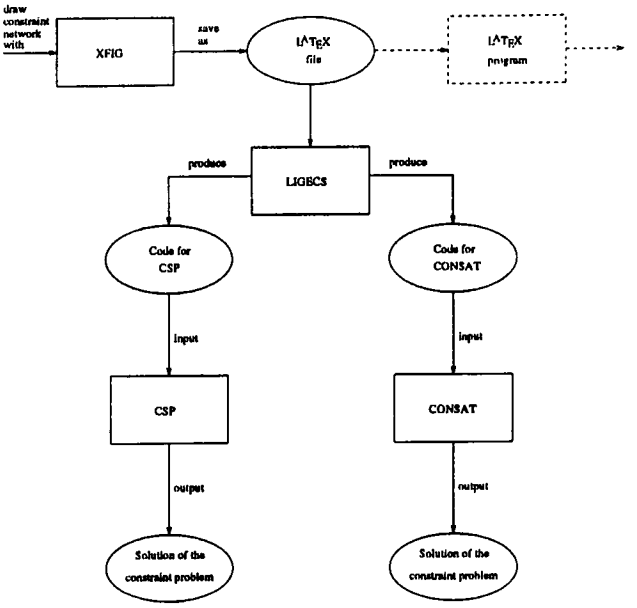


Figure 4: Converting a drawing of a constraint network into CSP, respectively CONSAT input.

(Mackworth 1977). Two nodes are connected by an edge if there is a constraint between the variables represented by the nodes. Figure 2 shows the resulting constraint network for the New Zealand map coloring problem.

Another way of representing a constraint network is as a bipartite graph (Guesgen 1989).[3] In this case, the variables are represented by one type of node (usually circles) and the constraints by the other type of node (usually boxes). Nodes of different types are connected if the variable represented by the circle is a one of the variables of the constraint represented by the box. Figure 3 shows the constraint network for the New Zealand map coloring problem as a bipartite graph.

Regardless of which representation is chosen, XFIG can be used to draw constraint networks like the above and to save them as LaTeX code with eepic macros.[4] The LaTeX code can then be included in documents like technical reports or conference papers. Beyond that, the code can be used as input for the LIGECS system, which analyzes the code, extracts all relevant information from it, and converts it into CSP or CONSAT code (see Figure 4).

LIGECS searches the LaTeX code for certain keywords like 'put', 'path', 'makebox', and 'ellipse'. It

[3]This representation is useful especially if the arity of the constraints in the network is greater than two.

[4]The eepic macro package extends the LaTeX picture environment by additional objects like ellipses, splines, etc.

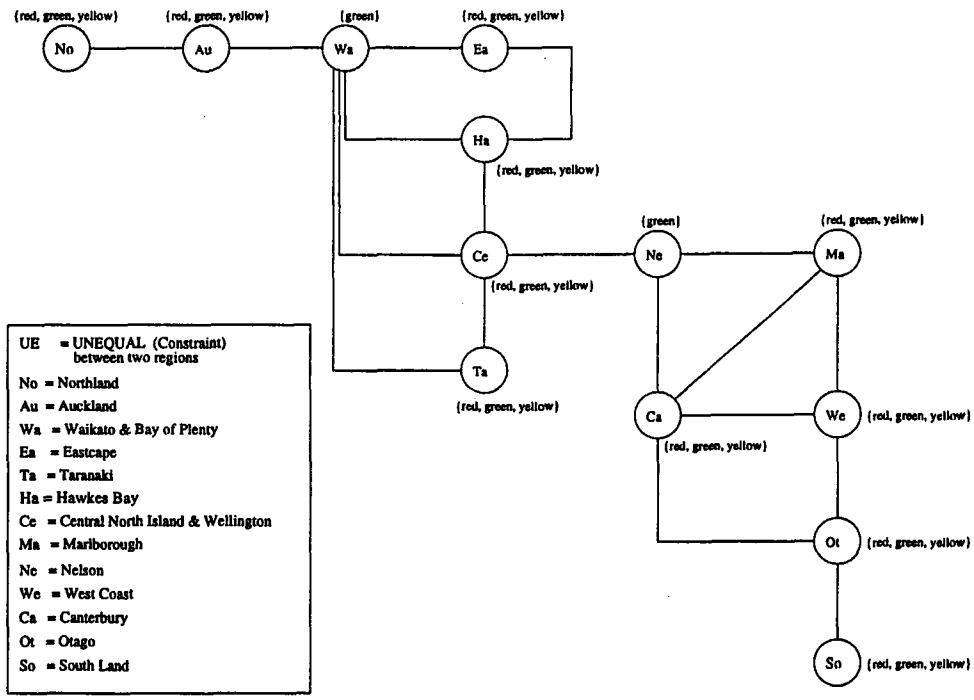


Figure 2: Constraint network for the New Zealand map coloring problem with constraints represented as edges.

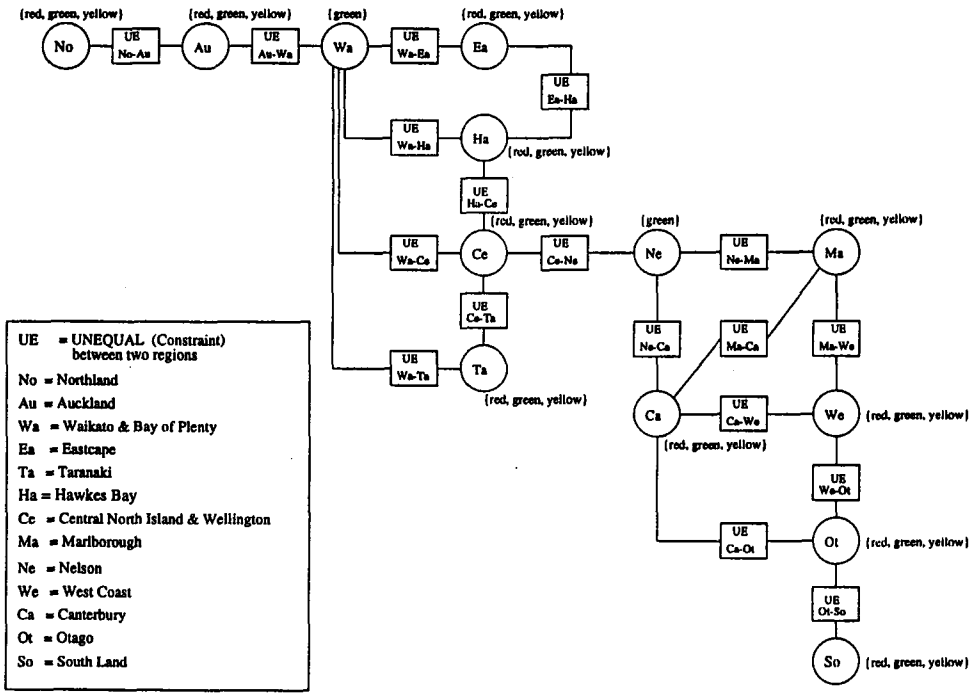
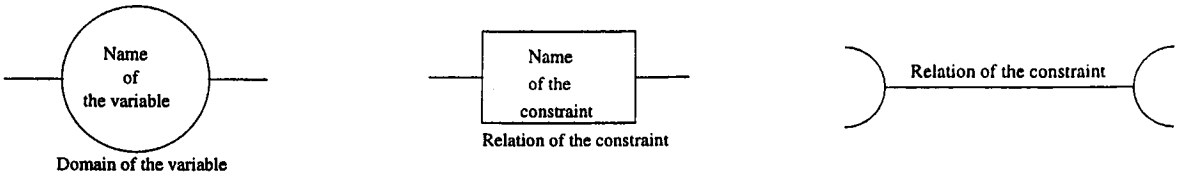


Figure 3: Constraint network for the New Zealand map coloring problem with constraints represented as boxes.



(a) (b) (c)

Figure 5: Graphical representation of a variable (a), a constraint as a box (b), and a constraint as a line (c).

analyzes the data associated with these keywords and generates objects of type 'circle', 'box', 'line', and 'text'. It then puts these objects into different databases and uses the position of the objects to determine the names of the variables and constraints, their domains and relations, the connections between variables and constraints, and so on. If, for example, the coordinates of the starting point of a line are on the circumference of a circle (defined by XFIG as ellipse), then it is assumed that the constraint represented by the line restricts the variable that is given by the circle. Or if a text object is inside a circle, then it is assumed that the text object specifies the name of the variable represented by the circle (see Figure 5).

Although this task is straightforward in principle, the praxis often looks different, as most drawings created by humans are fuzzy:

- Edges don't always end exactly at the boundaries of circles and boxes.
- Text indicating the name of a variable or constraint isn't always completely within a circle, respectively a box.
- Text denoting the domain of a variable or the relation of a constraint isn't always in the close vicinity of a circle, respectively a box.

To resolve this problem, we implemented a variety of algorithms and heuristics, which will be described in the next section.

Resolving Fuzziness

The basis of the parsing algorithm of LIGECS is an iterative algorithm that searches the LaTeX code of a constraint network for various elements like circles, boxes, lines, etc. and the connections between them. Figure 6 shows an outline of this algorithm. The algorithm starts with searching for the names and domains of the variables. For each variable, it then searches for the adjacent constraints, their names, and their relations. The two search algorithms are called variable filtering and constraint filtering, and will be described in the following subsections.

Variable Filtering

The first step in the execution of LIGECS is the filtering of the names and domains of the variables. For each circle representing a variable, the parser searches for the text placed in the interior of the circle. Beyond that, the parser also looks in the proximity of the circle, to allow for fuzziness in the drawings. In Figure 7, the search spaces for the name and the domain of a variable are shown.

LIGECS finds the names and domains of a variable by calculating recursively the distance d of the point vector of the center (m_x, m_y) of each circle and the point vector (x, y) of each text object:

$$d = \sqrt{(m_x - x)^2 + (m_y - y)^2}$$

If $d \leq r$, i.e., the text is inside the circle, or if $d \leq (r + \epsilon r)$, i.e., the text is inside a circle with a slightly larger radius, then the text is interpreted as the name of the variable. If $(r + \epsilon r) < d \leq (r + 11\epsilon r)$, then the text is interpreted as the name of the variable. ϵ is a fuzzy factor that can be changed anytime when running the program.[5]

After filtering out the domain and the name of a variable, LIGECS searches for the lines adjacent to the circle. Again, the inaccuracy of the drawing will be considered by using the fuzzy factor ϵ, which in this case means that the lines don't have to start exactly at the circumference of the circle (see Figure 8).

For the further progression of the program it is essential to distinguish between drawings in which constraints are represented as boxes and those in which constraints are represented as lines. We will discuss both cases in the following subsections.

[5]By testing the program with different examples, we found that $\epsilon = 0.1$ is a reasonable setting for us. However, a different user may have a different preference.

```
begin
  variable list ← nil;
  for each circle do
    for each text do
      if coordinates of text are inside search space
      of variable name then
        variable name ← text
      end if
      if coordinates of text are inside search space
      of variable domain then
        variable domain ← text;
      end if
    end for
    add variable name and domain to variable list;
    for each line do
      if one endpoint of line is inside search
      space for line endpoints of circle then
        if no box exists then
          for each text do
          calculate distance d to the line;
            if d ≤ 1.5ε then
              generate constraint name;
              constraint name ← generated name;
              constraint relation ← text;
              add constraint name and relation to
              constraint list of variable in variable list;
            end if
          end for
        else
          for each box do
            if other endpoint of line is inside
            search space for line endpoints of box then
              for each text do
                if coordinates of text are inside
                search space for constraint name then
                  constraint name ← text;
                end if
                if coordinates of text are outside
                search space for constraint name and
                inside search space for
                constraint relation then
                  constraint relation ← text;
                end if
              end for
            end if
          end for
        end if
      end if
    end for
  end for
end
```

Figure 6: Outline of the LIGECS algorithm for general constraint networks.

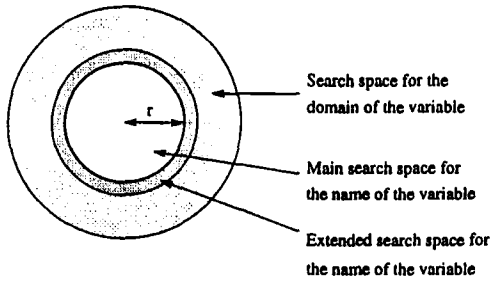


Figure 7: Search spaces for the name and domain of a variable.

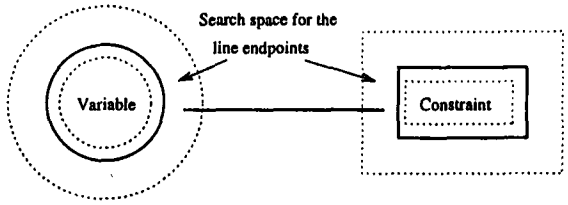


Figure 8: Lines connecting circles and boxes.

Filtering Constraints Represented as Boxes

The processing of constraints represented as boxes is very similar to how variables are processed. Text found inside a box or in the close proximity of the box is interpreted as the name of the constraint, whereas text further outside is interpreted as the relation of the constraint (see Figure 9).

Let (x, y) be the coordinates of the point vector of the text. For each text found in the drawing, LIGECS first checks if $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$, i.e., the text is inside the box. If this fails, then LIGECS checks if $x_l \leq x \leq x_u$ and $y_l \leq y \leq y_u$, i.e., the text is in close proximity of the box, where x_l, x_u, y_l, and y_u are defined as follows:

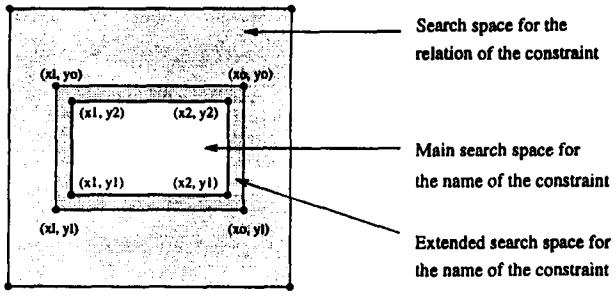


Figure 9: Search spaces for the name and relation of a constraint.

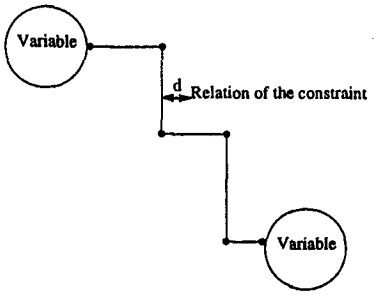


Figure 10: Binary constraint represented as a sequence of lines connecting two variables

- $x_l = (x_1 - \epsilon(x_2 - x_1))$
- $x_u = (x_2 + \epsilon(x_2 - x_1))$
- $y_l = (y_1 - \epsilon(y_2 - y_1))$
- $y_u = (y_2 + \epsilon(y_2 - y_1))$

If one of the checks is successful, the text is interpreted as the name of the constraint.

If the text is in the outer area of the whole search space, it is interpreted as the relation of the constraint. The conditions for a text object to be in that area are the following:

- $(x_l - 5\epsilon(y_o - y_l)) \leq x < x_l$ or
  $(x_o + 5\epsilon(y_o - y_l)) \geq x > x_o$
- $(y_l - 3\epsilon(x_o - x_l)) \leq y < y_l$ or
  $(y_o + 3\epsilon(x_o - x_l)) \geq y > y_o$

Note that the width of the outer space depends on the height of the original box, and the height of the outer space on the width of the original box. This means that a tall box is extended further in x-direction than a smaller box with the same width. Although this seems to be counterintuitive, it gave us the best results when testing LIGECS with several constraint networks.

Filtering Constraints Represented as Lines

In this section we will discuss drawings of binary constraint networks in which the constraints are represented by lines or series of lines annotated with the relations of the constraints (see Figure 10). The extraction of constraints from these drawings is simpler than the one from drawings in which the constraints are represented as boxes, since we assume that the names of the constraints are defined by the names of the variables and are not explicitly given in the drawing.[6] As a result, the only information that has to be filtered out for each series of lines is the relation of the constraint.

To determine whether a text object denotes the relation of a constraint given by a series of lines, the distance d of the text object to the closest line segment of the series is calculated. If $d \leq 1.5\epsilon$ then the text is considered to be the relation of the constraint.[7]

Conclusion

In this paper, we described the LIGECS system, which is an attempt at providing an environment for visual programming with constraints. The system takes the XFIG eepic representation of a constraint network and converts it automatically into a format that is readable by a constraint satisfaction system. LIGECS has been implemented in Common Lisp as a set of library functions.

LIGECS was developed as part of a diploma thesis, the time restrictions of which made it impossible to consider all aspects of how constraint networks can be represented graphically. As probably has become obvious in this paper, there are quite a few restrictions imposed on the user. If the conventions mentioned in this paper are violated, LIGECS delivers very poor results which require a significant amount of editing before they can be used by CSP or CONSAT.

Another shortcoming of LIGECS is the restriction to certain constraint satisfaction programs, in this case CSP and CONSAT. If there were a standard format for specifying constraint networks, this format could have been used as the output format of LIGECS. However, to the best of our knowledge, such a standard format doesn't exist yet, so we had to make a decision of what output format to use. To choose a C-based system on the one hand and a Lisp-based system on the other seemed to be reasonable.

There is no principal problem to overcome the shortcomings mentioned above. Other shortcomings, however, are much harder (or even impossible) to deal with. For example, LIGECS can't always determine the right order of the variables with respect to the relation of the constraint. Only if the name of the constraint contains the names of the variables (like in the example shown in Figure 3), the variables can be associated with the constraint relation in the right way, assuming that the order in which the variables occur in the name of the constraint is intentional and reflects which component of the constraint relation restricts which variable. Otherwise, the variables are associated with the constraint relation randomly, which often editing output file generated by LIGECS.

[6] LIGECS constructs a name for each constraint by concatenating the names of its variables.

[7] Again, the value of ϵ is based on the result of our experiments with LIGECS; its initial value is 0.1 .

References

Allen, J. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26:832–843.

Anger, F., and Rodriguez, R. 1995. Reasoning with unsynchronized clocks. In *Proc. IJCAI-95 Workshop on Spatial and Temporal Reasoning*, 25–34.

Chok, S., and Marriott, K. 1995. Automatic construction of user inferfaces from constraint multiset grammars. In *Proc. 11th International IEEE Symposium on Visual Languages (VL'95)*, 242–249.

Guesgen, H., and Hertzberg, J. 1993. A constraint-based approach to spatiotemporal reasoning. *Applied Intelligence (Special Issue on Applications of Temporal Models)* 3:71–90.

Guesgen, H. 1989. *CONSAT: A System for Constraint Satisfaction.* Research Notes in Artificial Intelligence. San Mateo, California: Morgan Kaufmann.

Mackworth, A. 1977. Consistency in networks of relations. *Artificial Intelligence* 8:99–118.

Manchak, D., and van Beek, P. 1994. A c-library of constraint satisfaction techniques. Technical report, Available by anonymous ftp from ftp.cs.ualberta.ca:pub/ai/csp.