The University of Auckland

Applying the Principle of Literate Programming to Constraint Satisfaction

Hans W. Guesgen **Ute Lörch**

Computer Science Report No. 138
December 1996

Applying the Principle of Literate Programming to Constraint Satisfaction

Hans W. Guesgen **Ute Lörch**
Computer Science Department, University of Auckland
Private Bag 92019, Auckland, New Zealand
{hans, ute}@cs.auckland.ac.nz

Abstract

When more than two decades ago David Waltz presented his now well-known filtering algorithm for labeling three-dimensional line-diagrams, one could hardly expect that the basic principle he introduced, namely the technique of constraint satisfaction, would become the basis of an emerging research area which has already outgrown the field of artificial intelligence and started to influence many other disciplines.

In the last few years many AI problems have been formulated as constraint satisfaction problems and solved by dedicated constraint satisfaction systems. Usually such a system offers a proper syntax for specifying constraint satisfaction problems. However, the syntax sometimes may be complex or awkward, and may require some effort to get familiar with.

In this paper, we argue that it is more appropriate to specify a constraint satisfaction problem by using a graphic editor and to translate the output of the editor into a notation that can be used as input for the constraint satisfaction system. Moreover, we are aiming at using the same output for documentation purposes. In other words, we are applying Knuth's idea of literate programming to constraint satisfaction.

1 Introduction and Motivation

Over the recent years, constraint satisfaction has become more and more popular as a general problem solving technique [1, 3, 4, 5, 6, 7, 9, 11, 12], and with that constraint satisfaction systems. Most commercial constraint satisfaction systems offer a reasonable user interface to input constraints and constraint networks into the system, but non-commercial systems like CSP [10] or CONSAT [8] are lacking this feature. It is often difficult to input constraints and constraint networks into such systems—and to edit them later—without making mistakes.

In most systems, constraints are specified by an explicitly given relation, i.e., an enumeration of value combinations. Each value combination denotes an assignment of values to the variables of the constraint such that the constraint is satisfied. If the constraint is a binary constraint[1] as in CSP, the allowed value combinations can be defined by a two-dimensional Boolean matrix, called constraint matrix, in which the rows indicate the values for the first variable and the columns the values for the second variable. The entries in the matrix indicate whether a combination of values is allowed or not.

A constraint network can then be represented by a matrix called the network matrix, in which the rows and columns indicate the variables of the network. Each entry of the network

[1] A binary constraint is a constraint with two variables.

matrix is a constraint matrix. If there are n variables in the network and k values in each of their domains, then we have to deal with an $n \times n$ matrix in which each entry is a $k \times k$ matrix.

Let us illustrate this representation by an example. Assume that there are five variables, $V_1, \ldots V_5$, each of which can assume a value from the set of natural numbers between 1 and 5, and seven constraints in the network:

Constraint	Variables	Relation
C_1	V_1, V_2	$\{(1,2),(1,1),(2,2)\}$
C_2	V_1, V_5	$\{(1,3),(2,4),(1,4),(1,2),(1,5)\}$
C_3	V_1, V_3	$\{(2,2),(2,1),(1,1)\}$
C_4	V_2, V_3	$\{(1,2),(1,3),(2,3)\}$
C_5	V_3, V_4	$\{(1,3),(2,3),(1,5),(3,5),(2,5)\}$
C_6	V_5, V_4	$\{(3,3),(5,5)\}$
C_7	V_5, V_3	$\{(2,2),(3,3)\}$

Representing these constraints in CSP would yield a five by five network matrix. Each entry of this matrix would be similar to the following constraint matrix, which is the constraint matrix for the constraint C_5:

$$C_5 = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Unlike CSP, CONSAT offers a Lisp-like language for defining constraints. Taking this language, the constraint C_5, for example, would result in the following Lisp expressions:

```
(DEFCONSTRAINT
        (:NAME C7)
        (:TYPE PRIMITIVE)
        (:INTERFACE L R)
        (:RELATION (:TUPLE (1 3))
                   (:TUPLE (2 3))
                   (:TUPLE (1 5))
                   (:TUPLE (3 5))
                   (:TUPLE (2 5))))
```

Although such an expression might be easier to read than an matrix, it is still not the ideal representation for large constraints networks. With hundreds of variables and constraints, one can easily confuse constraints and make mistakes when defining their relations.

So the question is: What is an adequate representation for constraint networks. When looking at the literature on constraint satisfaction, one will immediately notice that most researchers represent constraint networks by graphs. A graphical 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 drawings of constraint networks can be converted automatically into a format that is readable by a constraint satisfaction system. We will introduce a computer program that links a graphic editor with the constraint systems CSP and CONSAT. In addition to that, we want to use the same representation of the network for documentation purposes, like using it in technical reports, conference papers, journal article, etc. To put it in other words, we want to apply Knuth's idea of literate programming to constraint networks.

In particular, we developed the program LIGECS which, given a constraint network designed with the graphic editor XFIG, converts the network into two different formats:[2]

1. Program code that can be used as input to a constraint satisfaction system.

2. LaTeX code that can be used for documentation purposes.

If the graphics generated by a user were always perfect, the development of LIGECS would have been a trivial task, as the problem would have boiled down to a simple parsing problem. However, most graphics contain imperfections—as has been noted by others before [2]—which often cause a simple parsing procedure to fail. We solved this problem by incorporating into LIGECS heuristics that can deal with the fuzziness in user-generated graphics.

The rest of this paper is organized as follows. The next section shows how constraint network can be represented by graphs (intuitively a trivial task, but tricky with regards to details), discussing the problems of automatically translating a graphical representation into CSP or CONSAT code. We will then discuss algorithms and heuristics to overcome these problems. Finally, we will present some technical details of LIGECS and will summarize the paper.

2 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.[3]

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. 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.[4] 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.

[2]The main reasons for choosing XFIG as editor were that it runs on many different platforms and that it is

Figure 1: Coloring the map of New Zealand.

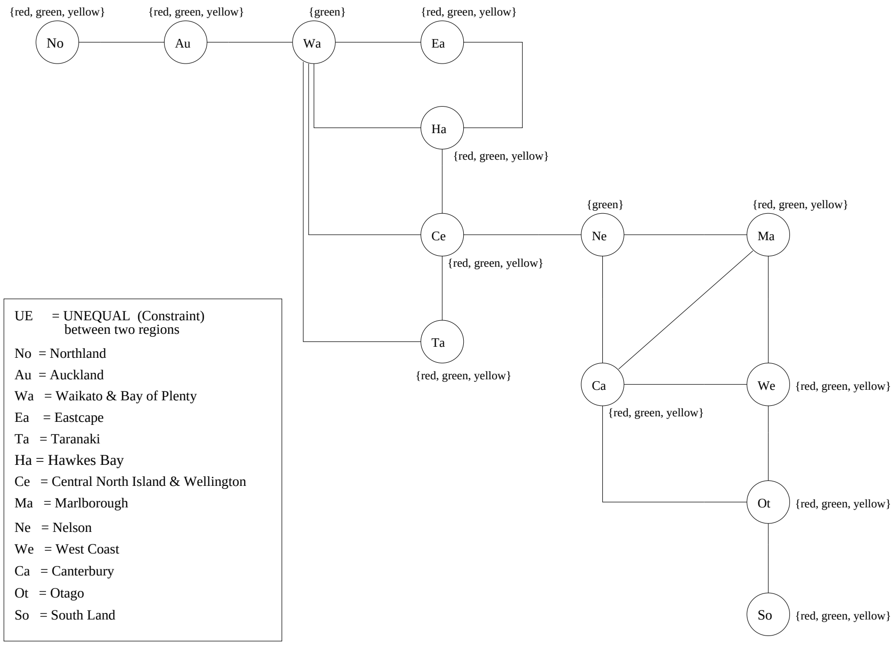


Figure 2: Constraint network for the New Zealand map coloring problem with constraints represented as edges.

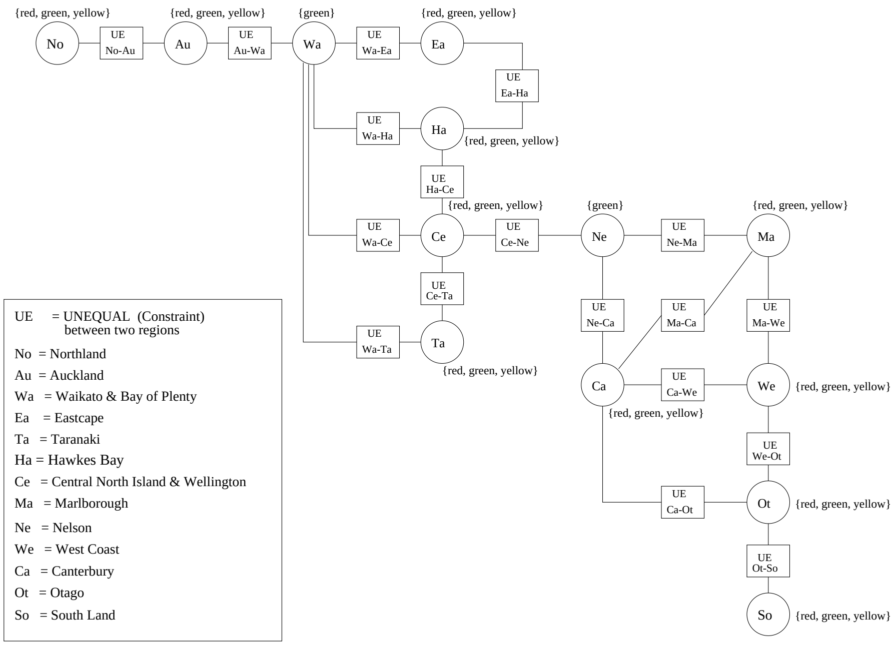


Figure 3: Constraint network for the New Zealand map coloring problem with constraints represented as boxes.

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.[5] 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).

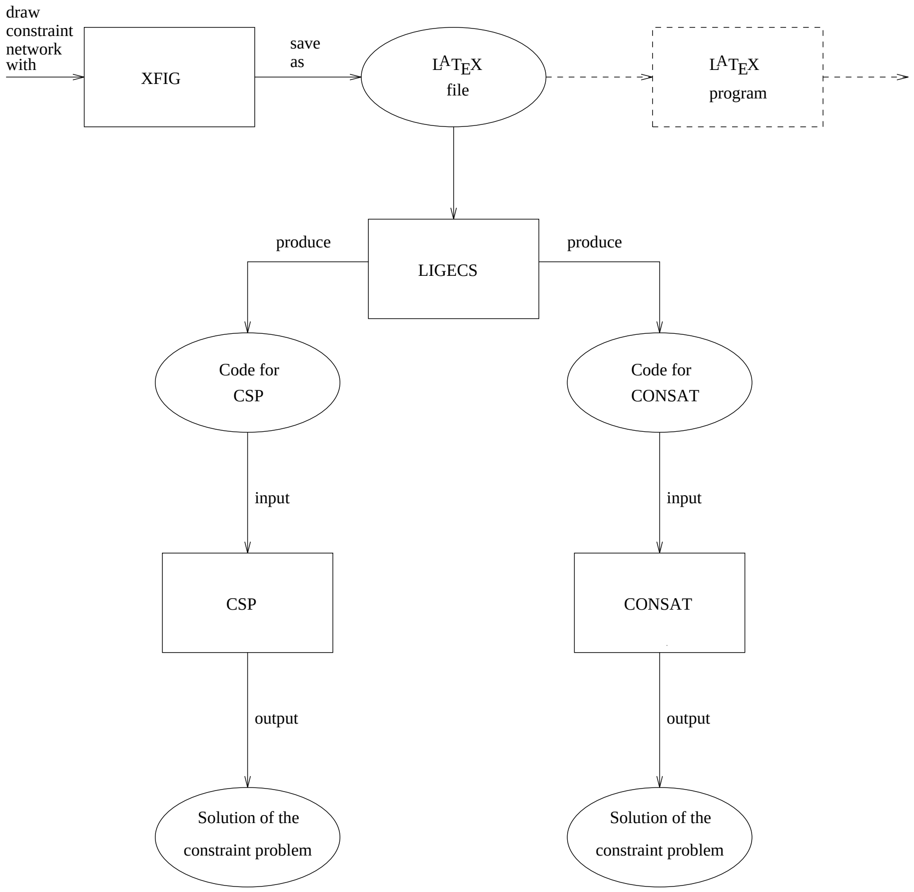


Figure 4: Converting a drawing of a constraint network into CSP, respectively CONSAT input.

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

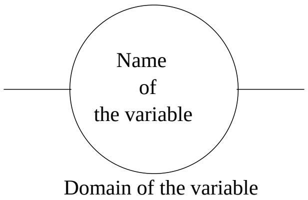
a suitable editor for drawing constraint networks.

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

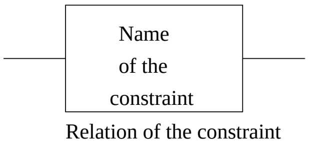
[4]This representation is useful especially if the arity of the constraints in the network is greater than two.

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

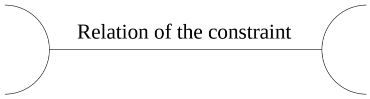
that is given by the circle. Or if a text object is inside a circle, then it is assumed that the text object specifies the name of the variable represented by the circle (see Figure 5).



(a)



(b)



(c)

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

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 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.

3 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.

3.1 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 pointer vector of the center (m_x, m_y) of each circle and the pointer vector (x, y) of each text object:

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

```
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 realtion 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.

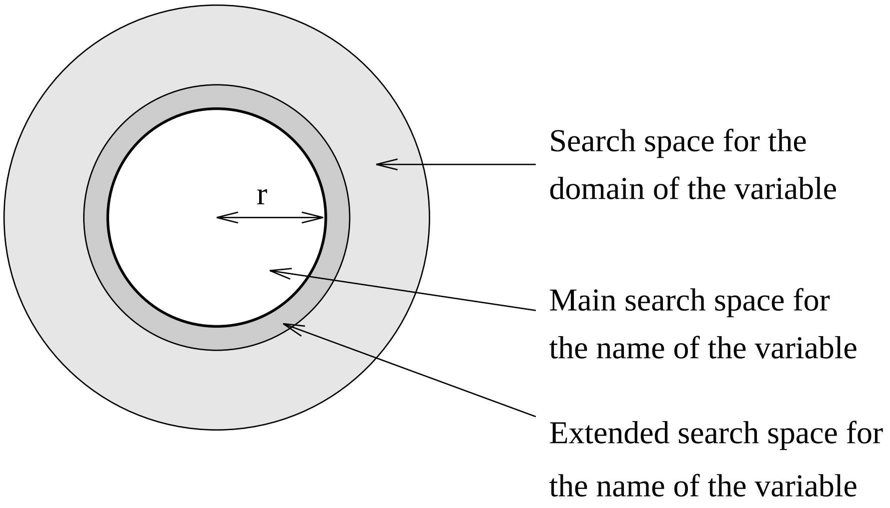


Figure 7: Search spaces for the name and domain of a variable.

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.[6]

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).

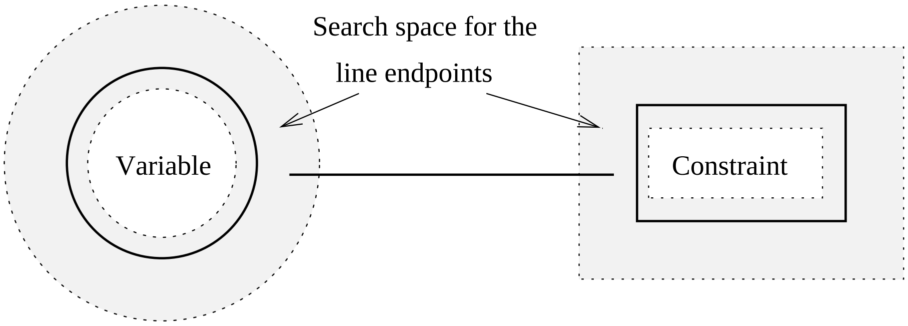


Figure 8: Lines connecting circles and boxes.

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.

[6]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.

3.2 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).

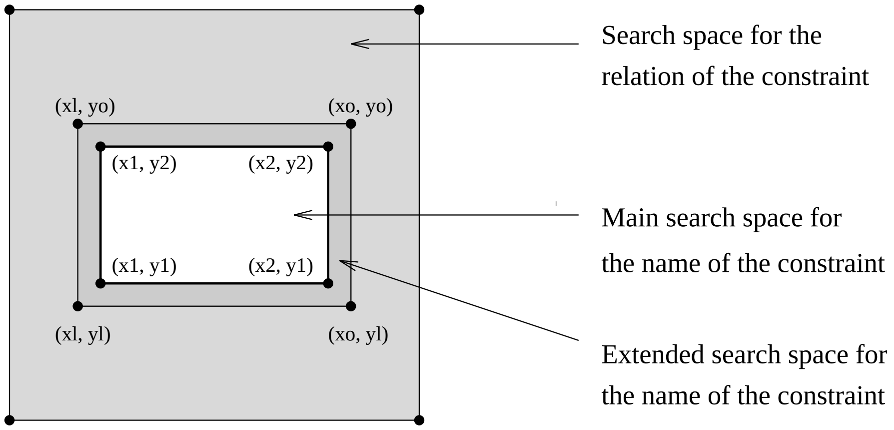


Figure 9: Search spaces for the name and relation of a constraint.

Let (x, y) be the coordinates of the pointer 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:

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

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.

3.3 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

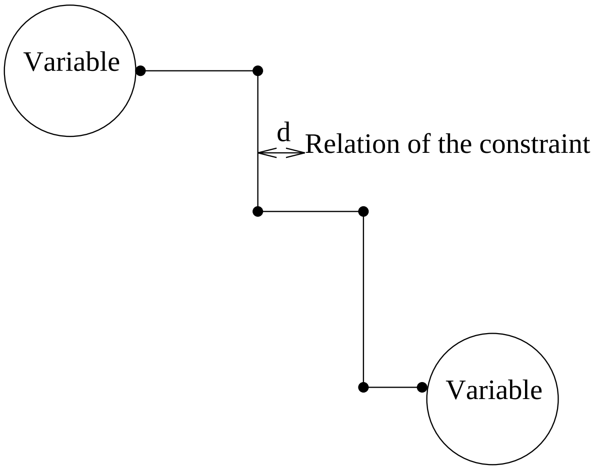


Figure 10: Binary constraint represented as a sequence of lines connecting two variables

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.[7] 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.[8]

4 Conclusion

In this paper, we described the LIGECS system for generating specifications of constraint networks from drawings of these networks. The system is based on the Knuth's idea of literate programming, which in the context of constraint satisfaction can be summarized as follows: Draw a constraint network with a graphic editor and use the result for both computation and documentation purposes. 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.

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

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

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.

References

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

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

[3] R. Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24:347–410, 1984.

[4] J. de Kleer and B.C. Williams. Diagnosing multiple faults. In *Proc. AAAI-86*, pages 132–139, Philadelphia, Pennsylvania, 1986.

[5] M. Dincbas, H. Simonis, and P. van Hentenryck. Extending equation solving and constraint handling in logic programming. In *Proc. Colloquium on Resolution of Equations in Algebraic Structures*, Austin, Texas, 1987.

[6] M.S. Fox, B. Allen, and G. Strohm. Job-shop scheduling: An investigation in constraint-directed reasoning. In *Proc. AAAI-82*, pages 155–158, Pittsburgh, Pennsylvania, 1982.

[7] H. Geffner and J. Pearl. An improved constraint-propagation algorithm for diagnosis. In *Proc. IJCAI-87*, pages 1105–1111, Milan, Italy, 1987.

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

[9] J. Jaffar and J.L. Lassez. Constraint logic programming. In *Conference Record of the 14^{th} Annual ACM Symposium on Principles of Programming Languages*, pages 111–119, Munich, Germany, 1987.

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

[11] R.M. Stallman and G.J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9:135–196, 1977.

[12] M. Stefik. Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16:111–140, 1981.

[13] D.L. Waltz. Generating semantic descriptions from drawings of scenes with shadows. Technical Report AI-TR-271, MIT, Cambridge, Massachusetts, 1972.