COMPUTER-AIDED DESIGN & APPLICATIONS, 2017
VOL. 14, NO. 1, 117–126
http://dx.doi.org/10.1080/16864360.2016.1199764

Constraint representation of 2-dimensional models with respect to avoiding cycles

Manfred Rosendahl

University of Koblenz-Landau, Germany

ABSTRACT
A geometric constraint system which models 2-dimensional geometries in a form that often no multidimensional equation systems are necessary, when solving a given constraint system, is described. This is achieved because not only constraints between points are used, but also circles and lines are introduced as objects.

KEYWORDS
Modeling geometric constraints in 2D; graph-oriented declarative modeling

1. Introduction

There are many approaches to geometric constraint solving. For an overview of modeling geometric constraints, see [2]. The technologies for the solution of these geometric constraint systems can be divided into three classes:

- Equation-based methods.

Here the constraints are represented through non-linear equations involving the variables. A newer system using this approach is **solvespace** (solvespace.com/index.pl). A description of its internal working can be found in [8]

- Rule-based methods.

In this approach a geometric reasoning mechanism in which the dimensions and geometric relationships are defined as either facts or rules is used [7]. Also a geometric constraint system can be solved by a decomposition plan as in [4, 5]. Here the problem is fractionized and then solved bottom up.

- Graph-based methods.

In the graph-based technologies the constraints are modeled as hyper-edges between the points or other variables. An alternative to hyper-graphs are bipartite graphs, in which variables and constraints are both initially connected by non-oriented edges.

The nodes are divided into two disjoint sets V and C. V represents the geometric objects (here scalars, points, circles and lines). C represents the constraints. The edges in the graph link the V and the C nodes. In most approaches including [1], the points are the only geometric objects. Here it is shown that by using circles and lines also as geometric objects, a solution without cycles often exists and therefore a sequential computation is possible. For an overview of modeling geometric constraints, see [2].

Depending on the number of constraints, from a given subset of the variables the remaining variables can be calculated. The specific type of modeling makes it possible to use constructive operations when evaluating the graph from any sufficient set of given variables. Therefore, after the creation of the model it is possible to distinguish which variables are free and which are calculated. A point can be calculated as the intersection of two lines. In addition, if the intersection point is moved, the lines move appropriately. To construct a calculation sequence from the model and the fixed variables, an analysis of the degrees of freedom (DOF) is used. The goal is to produce an oriented graph from the non-oriented constraint graph. If a sequential calculation is possible, an acyclic graph should result. In many cases where cyclic dependencies exist, however, only a graph with one or more cycles can be produced. If too many variables are fixed, the system is over-constrained, and then no orientation is possible.

This paper extends that approach by Berling [1]. By using not only scalars and points but also circles and lines as variables, many models can be solved sequentially. Thus, systems of non-linear equations can be avoided. Graph-based methods have an advantage when dealing

CONTACT Manfred Rosendahl ros@uni-koblenz.de

with models that are not fully constrained. Here it is shown that by using circles and lines also as geometric objects, a solution without cycles often exists and therefore a sequential computation is possible.

2. Constraint system

When using only points and scalar values, we could have the following constraints:

1. DP (P1, P2, d): Distance between points P1 and P2 is d
2. DPX(P1, P2, d): x-distance between points P1 and P2 is d
3. DPY(P1, P2, d): y-distance between points P1 and P2 is d
4. A3(P1, P2, P3, alpha): Angle between the lines (P1,P2) and (P1,P3) is alpha
5. RW(P1, P2, P3):The lines (P1,P2) and (P1,P3) are perpendicular
6. Dl (P, Pa, Pe, d): the distance between the point P and the line from Pa to Pe is d.
7. Vertical (P1, P2): P1 and P2 have the same x coordinate.
8. Horizontal (P1, P2): P1 and P2 have the same y coordinate.
9. Equ (< expression >): The value of the expression is 0. It must be possible to compute one variable if the others are known.

All these constraints consume at least one degree of freedom. The points have 2 degrees of freedom; the scalar values have 1. But points and values can be fixed. If only one dimension of a point is fixed, it still has 1 degree of freedom. Otherwise, the point's degree of freedom is 0. The constraint system itself is represented by an undirected graph. To find a solution of the system, the graph must be directed and must fulfill the following conditions.

1. Each node of V has no more incoming edges than the node's degree of freedom.
2. Each node of C has as many outgoing edges as the constraint consumes degrees of freedom. All other edges of each node of C are incoming.

Each incoming edge of a point determines one locus for the position of the point. E.g., if a point must have a certain distance to another point the locus is the circle around the other point with the radius of that distance. If the point is the origin of a right angle, the locus is the Thales circle between the corner points. For a corner point, the locus is the line perpendicular to the line between the origin and the other corner through the origin.

For a point, we have the following loci: x coordinate, y coordinate, on line, on circle. For each combination of up to 2 loci, a procedure is defined to compute the point. As an example we consider in Figure 1 the line tangent to two circles.

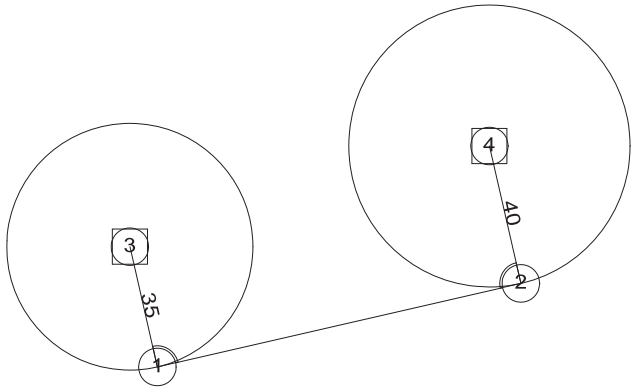

Figure 1. Connection from point 1 to point 2 forms a line tangent to two fixed circles.

The tangent line in Figure 1 must fulfill the following conditions.

- The distance between the center and the tangent point must be equal to the radius of the circle.
- The angle with the origin at one tangent point and the corners at the other tangent point and the center of the circle must be a right angle.

The corresponding constraint graph and the possible orientation (when the center and the radii of the circles are fixed) are shown in Figure 2 below.

Note: Oval or rounded nodes represent variables (scalars, points, circles and lines). If points are rounded, they are fixed in the coordinates annotated (here X and Y). The rectangular nodes represent the constraints. The first number in the notation is the number of the node in the underlying implementation. The second symbol in the notation is either the type of the constraint node or the value or notation of the variable node. For totally or partially fixed points, the third symbol in the notation identifies what is fixed.

To compute one tangent point, the other tangent point must be known. Therefore, it is not possible to compute the solution sequentially. But if the two circles are known, it is possible to compute the tangent line. If we introduce both circles and lines as geometric objects with corresponding constraints, a sequential solution is possible.

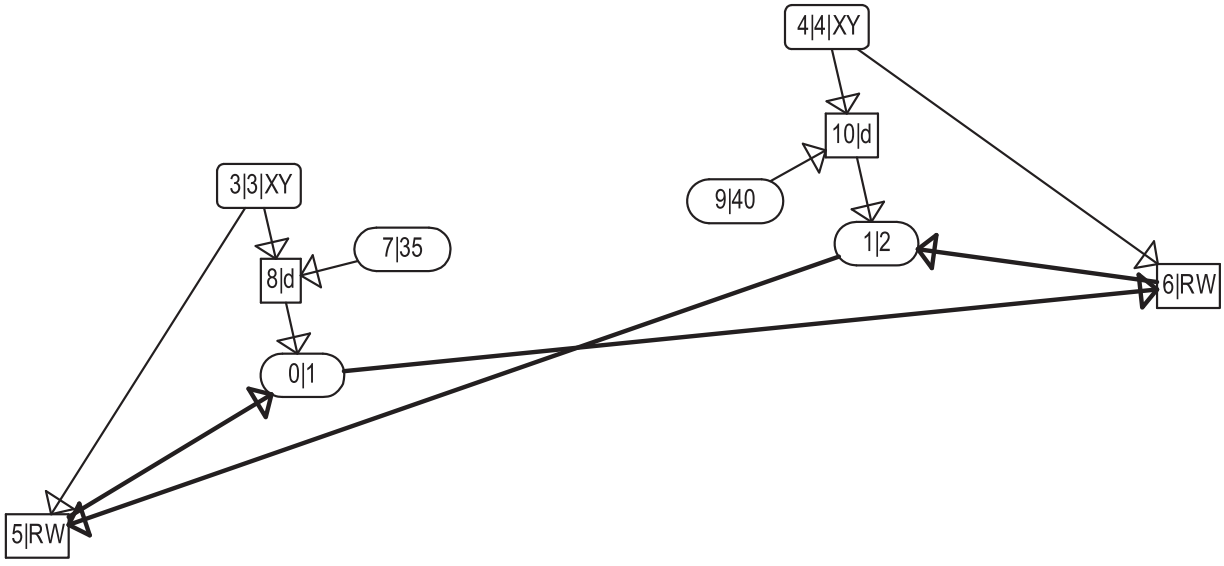

Figure 2. Oriented constraint graph of the model in Fig. 1.

The class circle is derived from the class point (its center) and has 3 degrees of freedom (the X, Y coordinates of the center and the radius). A line has 2 degrees of freedom. As parameters for the line, we take the length and the direction of the perpendicular line from the origin to the line.

As new constraints we introduce:

- 2 circles are tangential
- Line and circle are tangential
- Point on line or circle (incidence)
- 2 lines parallel or perpendicular
- Distance from a point to a line
- Direction of a line
- Radius of a circle

Am intersection point is defined by the incidence with 2 lines or circles

The tangent line in Figure 3 is defined by the following constraints:

- Line tangential to circle 1
- Point 3 on Circle 1
- Point 4 in circle 2
- Point 3 on line
- Point 4 on line
- Line tangential to circle 2

For the line and the 2 tangent points, we have 6 degrees of freedom. These degrees are consumed by the 6 constraints. So if the two circles are fixed, the model is fully defined. Starting with the fixed circles, the constraint graph in Figure 4 can be propagated.

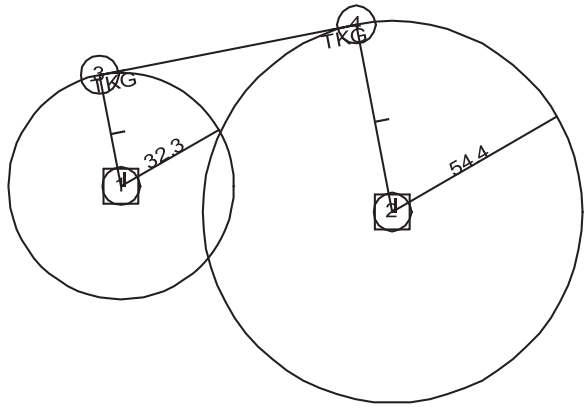

Figure 3. Model with a line tangential to 2 given circles. Points 3 and 4 on line and circle.

For each type of node, variable and constraint, a computation method is defined. For constraints, the method sets the loci to the geometric objects which are reached by the outgoing edges. For geometric objects, the method computes the parameters of the object with respect to the given loci. If the constraint graph has no cycles, the computation can be done sequentially in topological order.

A geometric object can have at most as many incoming edges as its degree of freedom. But not all combinations are possible. E.g., a line cannot be determined twice by a constraint which fixes the line's direction. For a circle only 1 constraint can fix the radius, and mostly 2 constraints can fix the center point. Points may have only one constraint which can be satisfied only by either the X coordinate or the Y coordinate. Each constraint has as

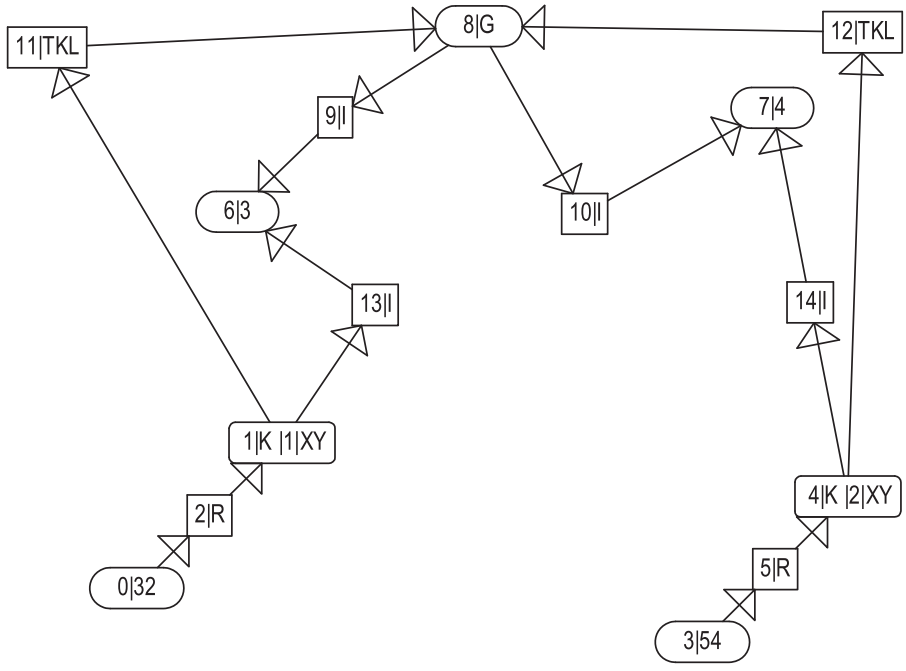

Figure 4. Oriented constraint graph of model from Figure 3.

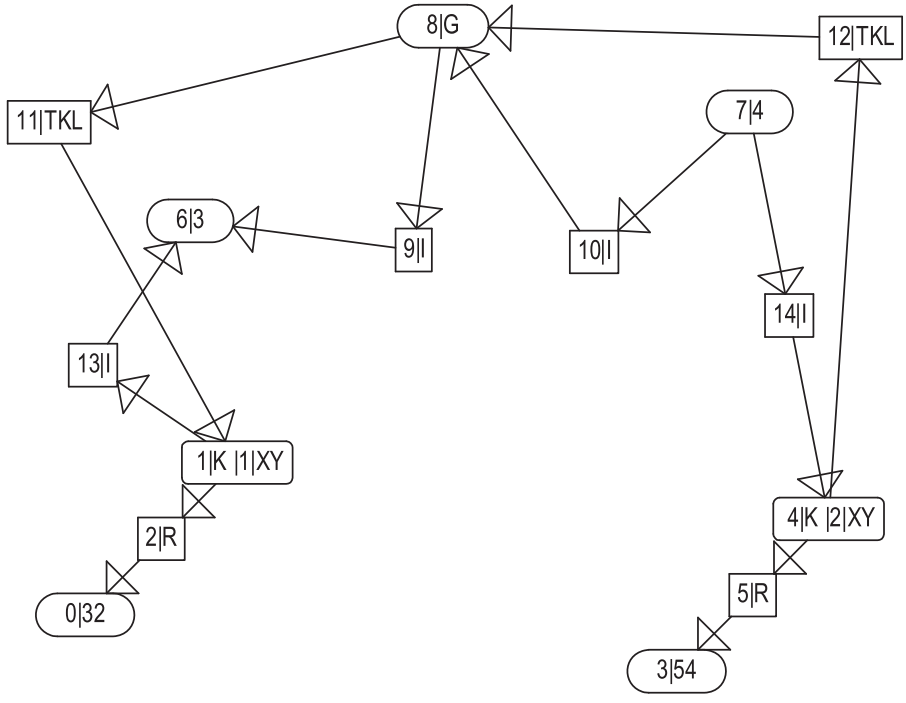

Figure 5. Oriented constraint graph in which the tangential point 4 and the centers of the circles are fixed.

its property the coordinates, which are influenced by the constraint.

Given two circles, we can construct not only a common tangent line. For instance, we can also construct the second tangent point if the centers of the circles and the first tangent point are given. The variation is done over the radii of the circles. The directed graph is shown in Figure 5 below.

With this modeling all other variations can be solved sequentially, too.

For a circle we have, beside the loci for the center as with points the following loci: tangential to line or circle, point incident with circle, and radius given. For up to three combinations of these loci, a procedure is defined to compute the circle. In many cases two or more solutions are possible. Then the solution nearest to the

actual situation is chosen. For lines we have up to two of the following loci: direction, tangential to circle, point co-incident with the line.

3. Orientation of the constraint graph

The idea of orienting the edges in a constraint graph to determine the sequence of the computations is well known. When an orientation without cycles is possible, this orientation can be achieved by propagating the constraints and the degree of freedom. With cycles, the maximal constraint matching is adapted from the Edmonds method for bipartite graphs [3, 6].

With the constraints used here, we have to consider some special semantics. When orienting the constraint graph, each variable node may have at most as many incoming edges as the variable's degree of freedom. That is 2 for points and lines, and 3 for circles. However, this condition is necessary but not sufficient. Some constraints as horizontal or vertical alignment of point can only be satisfied by the x or y coordinate, respectively. For lines, constraints like horizontal or vertical can only be satisfied by the direction. A circle can have only 1 incoming constraint involving only the radius. To orient the undirected constraint graph of the model, we need therefore a special function for each variable class. This function determines whether for an adjacent constraint the variable still has a degree of freedom to orient an edge as incoming to the variable.

For all classes of variables, a function `sdf(afix: tfix):integer` is implemented. `afix` can have the following values.

- lsx: The constraint can be satisfied only by the x coordinate of a point respectively the direction of a line.
- lsy: The constraint can be satisfied only by the y coordinate of a point
- lsxy: The constraint can be satisfied by the x or the y coordinates of a point.
- lsr: The constraint can be satisfied only by the radius of a circle respectively the distance of a line from the origin.
- lsxr: The constraint can be satisfied by the x coordinate of the center of a circle or the radius of a circle respectively the direction or the distance from the origin of a line.
- lsyr: The constraint can be satisfied by the y coordinate of the center of a circle or the radius of a circle.
- lsxyr: The constraint can be satisfied by the x or the y coordinates of the center of a circle or the radius of a circle.

So the parameter `afix` identifies the parameters of the variable with which the constraint can be satisfied. The function `sdf` examines all already incoming edges, and determines what is already fixed.

For scalars this is simple. When an edge is already incoming, `sdf` is 0.

For points, the following transition matrix in Figure 6 must be used.

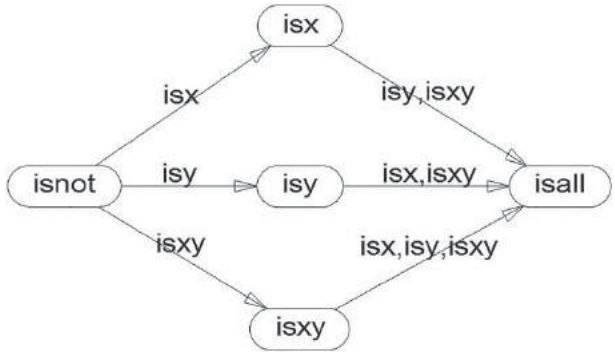

Figure 6. Transition matrix for points.

A point has initially 2 degrees of freedom, which corresponds to the state `isnot`. If the x-coordinate is fixed by a constraint, the state changes into `isx`. Now it can only fulfill yet another constraint, which changes the y-coordinate too. This can be either a constraint only affecting the y-coordinate (`isy`), or a constraint which specifies a locus (`isxy`).

If no further transition with `afix` is possible, `sdf` is 0. Using the transition matrix when both constraints can only be satisfied by the x coordinate, 2 edges can not be incoming oriented. For a circle, we have the same requirements for the center point. But an additional requirement is that there can be at most one incoming edge, which can be satisfied only by the radius. For a line, we can have at most one incoming constraint which can be satisfied only by the direction of the line. That is the case with the two constraints parallel and perpendicular, and generally for a constraint which determines the direction of a line.

The orientation can be demonstrated with the example of Figure 5. The point [7|4] and the centers of the circles [1|1] and [4|2] are fixed. Because [7|4] has no degree of freedom, all edges are outgoing. This determines the orientation of the constraints [10|I] and [14|I]. Now circle [4|2] is totally fixed, which determines the orientation of the constraints [5|R] and [12|TKL]. This fixes line [8|G], which determines the orientation of the constraints [11|TKL] and [9|I]. This, on the other hand, fixes the circle [1|1] totally. Therefore the other edges of [1|1] must be outgoing. That orients the constraints [13|I] and [1|R], which in turn fixes radius [0|32] and point [6|3].

4. Handling under-constraint systems

With this graph-oriented approach we can also handle under-constrained systems in an adequate manner.

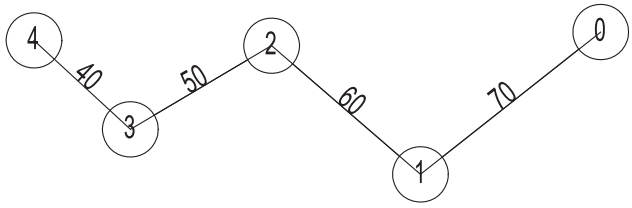

Figure 7. Snake.

If we move in the snake in Figure 7 the point 0, the resulting constraint graph is shown below in Figure 8.

The move of point 0 is propagated through all points. So the other points will follow the moving of point 0. To achieve this, a breadth-first search starting with point 0 is done. Then the edges are oriented in the order of the sorting (when possible). If we move point 0 further we get the result in Figure 9.

If point 4 is fixed and point 0 is veered away from point 4, the snake in Figure 7 is stretched as far as possible. If the point is moved further away no move is done, because we no longer have a solution. The point 3 is determined by the distance to point 4 and by the propagated move of point 1 and 2 due to the move of point 0. The constraint graph of this model is shown in Figure 10 below.

5. Models with inherent cycles

Even with circles and lines, not all models can be solved sequentially. Some models have inherited cycles. E.g., we put a box into a given contour. Figure 11a shows the complete model. The fixed contour is given by the points 1-4-3-2. The distance between A and B (50) is the width of the Box. The right angle at B is the edge of the box. The box is placed on the connection between 4 and 1, the connection between 4 and 3 and on point 2. So position of the box is fully determined. But it should be possible to change the width of the box or the position of the points of the contour.

To simplify the constraint graph for the demonstration, we use a simplified model of Figure 11b. The task here is to compute the points A and B. No orientation without a cycle is possible and therefore A and B can

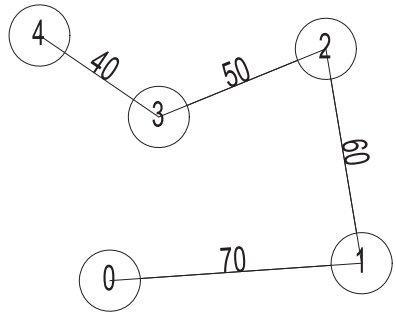

Figure 9. Result after moving point 0 further.

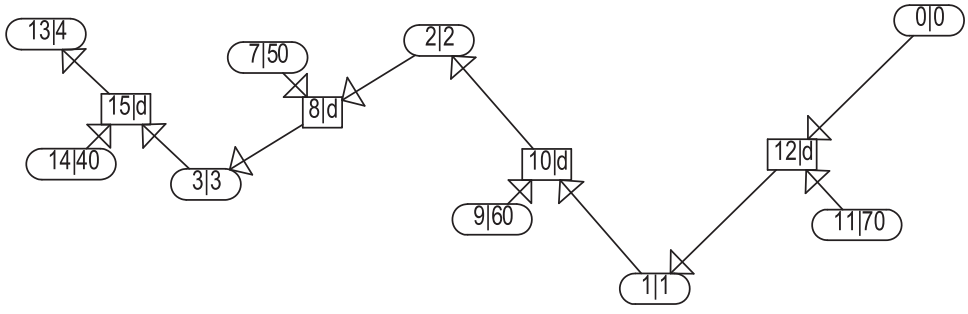

Figure 8. Constraint graph after moving point 0.

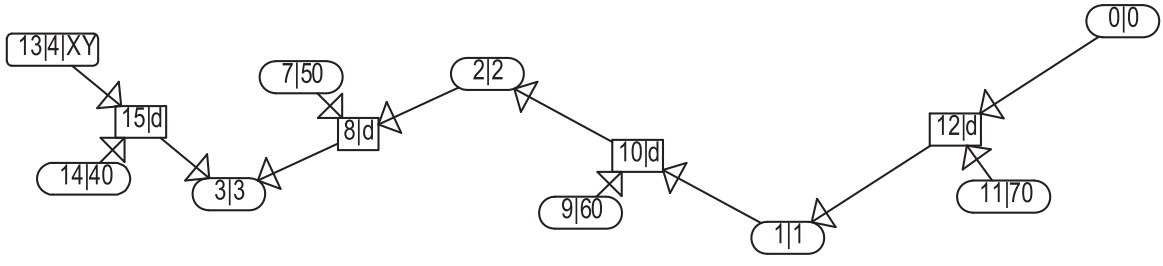

Figure 10. Constraint graph of moving point 0 with point 4 fixed.

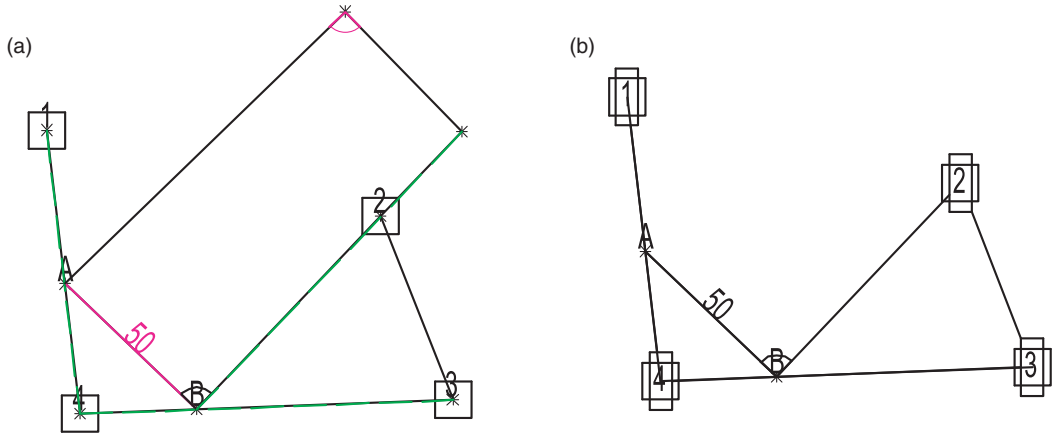

Figure 11. (a) Model which can not be computed sequentially. (b) Simplified model of Figure 11a.

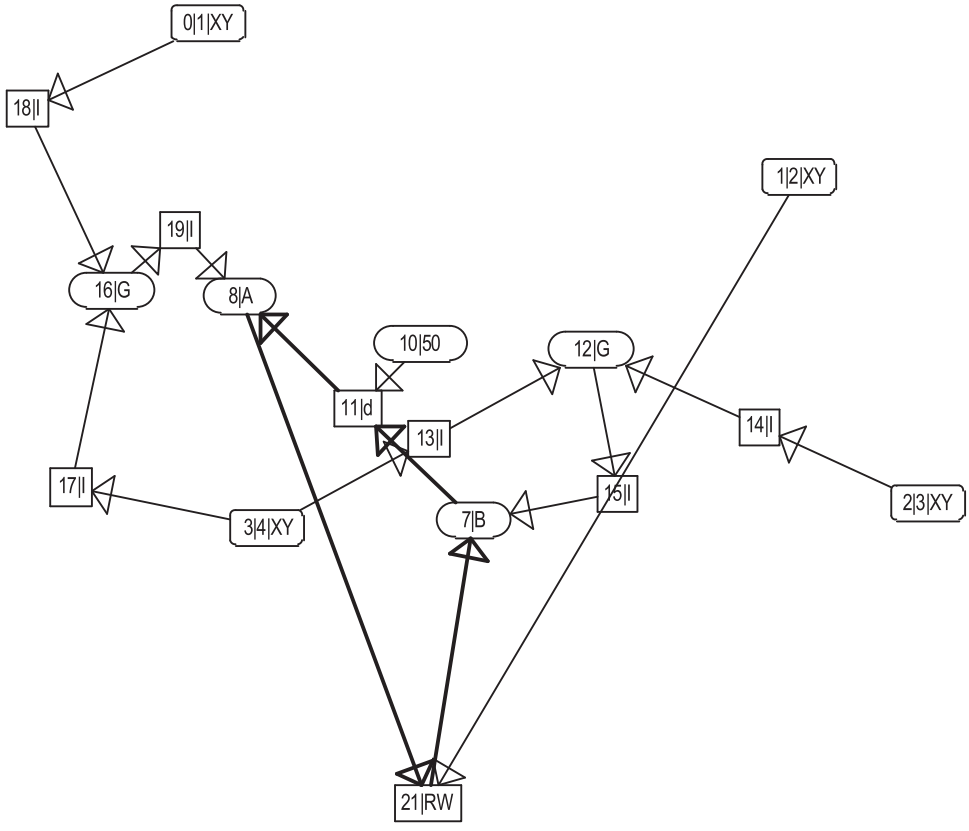

Figure 12. Constraint graph of the model of Figure 11b.

not be solved sequentially. The orientation of model in Figure 11b is shown in Figure 12.

The oriented graph in Figure 12 contains a cycle, which includes the points A and B and the constraints 11 and 21. Therefore the points A and B must be solved simultaneously. Four values (namely, the X and Y coordinates of the two points) are computed using the following four constraints: 2 times point on line (15, 19), the distance (11), and the right angle (21). To compute the cycle not only the constraints in the cycle must be included but also the constraints, which have an outgoing edge to a value in the cycle, in this case the constraints 19 and 15. Each constraint defines an equation between the values. This non-linear equation system is solved by iteration. Note also that in this case a compass and straight-edge construction is not possible.

6. Application

The approach was used in building a diagram editor. The nodes (for instance, a square) adapt their sizes according to the contents of the node. Then there are constraints

which adjust the width or the height of two nodes with respect to the larger one. Furthermore there are four special constraints:

- Horizontal chain
- Vertical chain
- Horizontal alignment
- Vertical alignment

If two nodes build a chain, the distance in the x resp. y direction is half of the sum of the width resp. height of the two nodes. So the nodes are positioned back-to-back. With alignment, the sizes of the nodes are aligned to the maximum width resp. height.

With these constraints it is possible to build tables. When the content of the nodes is changed, these tables adjust automatically.

An example for a table consisting of 4 nodes is shown in Figure 13. The constraint graph of the model in Figure 13 is shown in Figure 14. If the upper left box of the table in Figure 13 is changed to hold more rows and columns than the other boxes in that table, those other boxes are aligned automatically as shown in Figure 15.

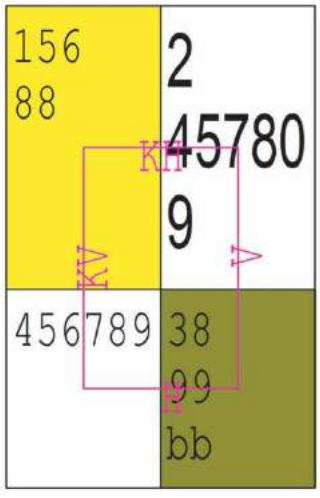

Figure 13. Tables with chain and alignment.

These boxes can be used in diagrams.

There are two special connections between the boxes. On the left in Figure 16 is a Manhattan-edge, which is parted in a vertical part and a horizontal part. On the right in Figure 16 are Z-edges. These Z-edges consist of either vertical-horizontal-vertical parts (if the vertical distance is greater than the horizontal distance) or of horizontal-vertical-horizontal parts (if the horizontal distance is greater than the vertical distance).

By simply moving the boxes, a new layout can be obtained automatically. The edges are assigned according

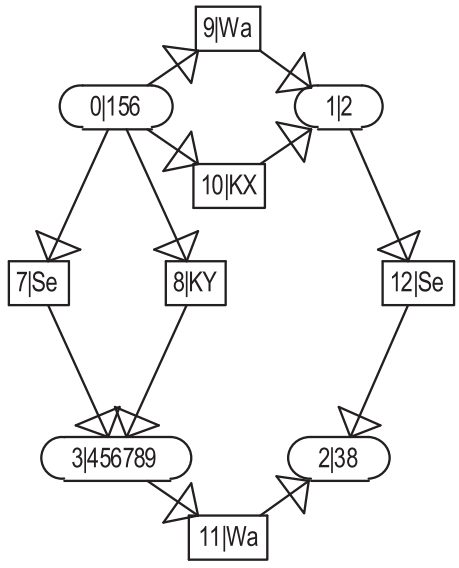

Figure 14. Constraint graph of the table.

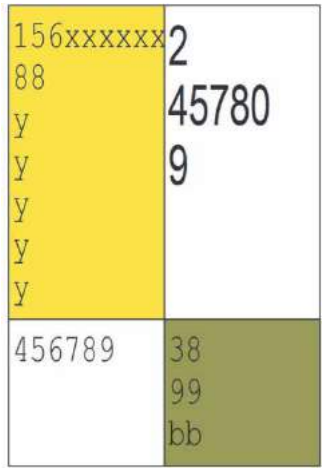

Figure 15. Table with new alignments.

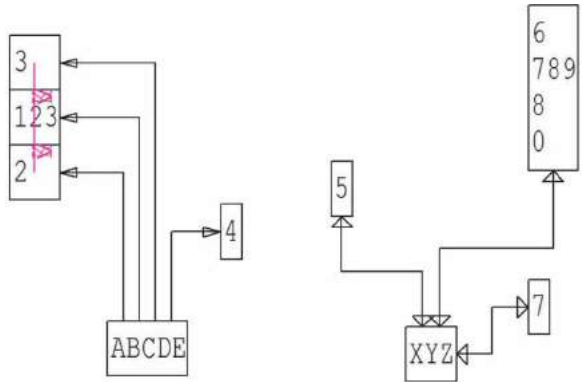

Figure 16. Diagram, original layout.

to the new position to the appropriate edge of the box and the order of the connection is sorted so, that no overlapping arises. In this manner, one can start from

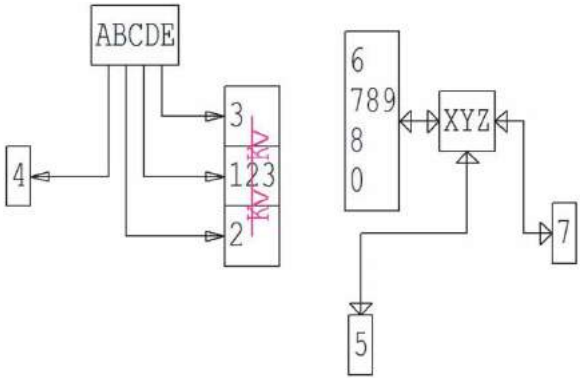

Figure 17. Diagram with new layout.

the diagram in Figure 16 above and reach the diagram in Figure 17 below, automatically by just moving the boxes.

Application with segments

To model more complex parts it is often not necessary to model each line. Instead a segment is defined, which can be included according to 1 to 3 reference points.

A segment can have:

- 1 reference point: position
- 2 reference points: position and direction and size
- 3 reference points: position, orientation and size in 2 dimensions

An example with segments is shown in Figure 18 below. The pistons are segments with 3 reference points. One for the position and the other two define the high and the radius of the piston and the orientation of the piston. The crank and the rods are similar defined. For the constraint model only the reference points are relevant.

If the angle of the crank is changed, the motor will rotate. Changing the angle of the crank W = 45 from 45 to 60 makes it a 60° degree V-motor instead of a 90°.

7. Conclusion

It was shown that, by using circles and lines as geometric objects in a model, cycles which are necessary when using points as the exclusive geometric objects can often be avoided. This allows the building of systems in which any point can be moved in real time by dragging the point with a mouse device. The orienting of the constraint graph is done once before dragging the points. During the drag, only the computing is necessary. When there are no cycles, this can be done in sequential order. When the model must be oriented with cycles, the algorithm finds a solution in which a minimum number of variables are involved.

The constraint-based techniques, which have been introduced here, can also improve the editing of diagrams significantly.

The implemented constraint system is based on the self implemented CAD system VarioCAD and is free for private or educational use and can be downloaded from the following websites:

http://userpages.uni-koblenz.de/ ~ ros/variocad.html
or: http://www.heise.de/software/download/variocad/50375

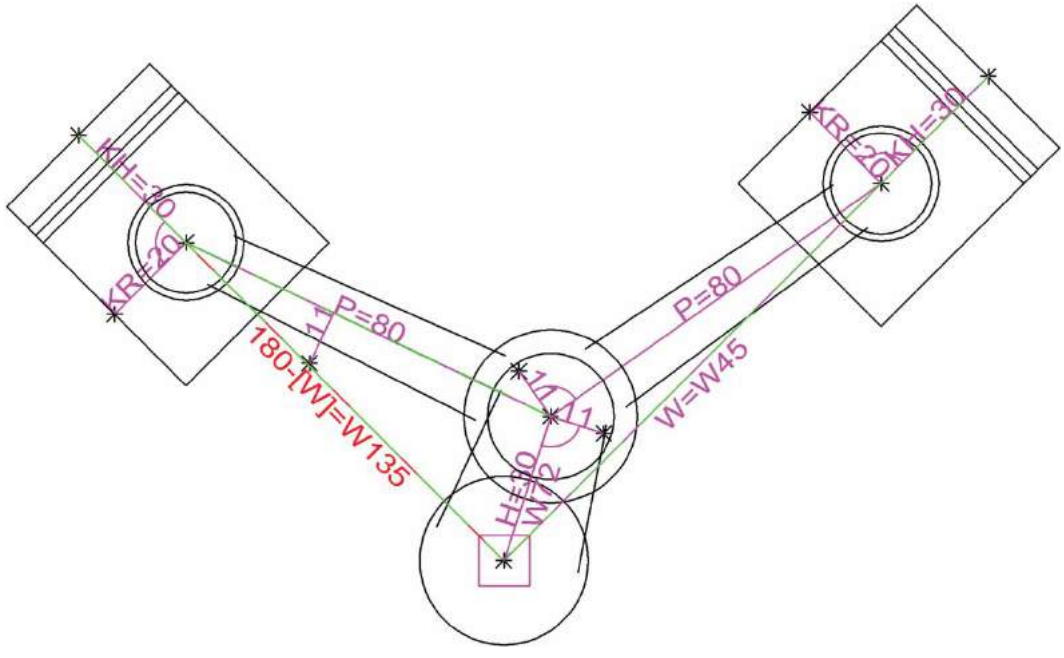

Figure 18. Motor with segments.

ORCiD

Manfred Rosendahl http://orcid.org/0000-0002-1476-4306

References

[1] Berling, R.; Rosendahl, M.: Modeling of Geometric Constraints in CAD-Applications, Part of: Geometric Constraint Solving and Applications, Brüderlin, B; Roller, D (Eds.), 151–169, Springer, Berlin, Heidelberg, 1998. http://dx.doi.org/10.1007/978-3-642-58898-3_8

[2] Bettig, B.; Hoffmann, C.M.: Geometric Constraint Solving in Parametric Computer-Aided Design, Journal of Computing and Information Science in Engineering, 11(2), 2011,021001-021001-9. http://dx.doi.org/10.1115/1.3593408, http://www.cs.purdue.edu/homes/cmh/distribution/PapersChron/ConstraintSurvey2010.pdf

[3] Edmonds, J.: Paths, trees, and flowers, Canadian Journal of Mathematics, 17, 1965, 449–467. http://dx.doi.org/10.4153/CJM-1965-045-4

[4] Hoffmann, C.M.; Lomonosov, A.; Sitharam, M.: Decomposition Plans for Geometric Constraint Systems, Part I: Performance Measurements for CAD, Journal of Symbolic Computation, 31(4), 2001, 367–408. http://dx.doi.org/10.1006/jsco.2000.0402

[5] Hoffmann, C.M.; Lomonosov, A; Sitharam, M: Decomposition Plans for Geometric Constraint Problems, Part II: New Algorithms, Journal of Symbolic Computation, 31(4), 2001, 409–427. http://dx.doi.org/10.1006/jsco.2000.0403.

[6] McHugh, J.: Algorithmic Graph Theory, Prentice-Hall, New Jersey, 1990. http://www.mathe2.uni-bayreuth.de/axel/papers/mchugh:algorithmic_graph_theory.pdf

[7] Verroust, A; Schonek, V;Roller, D.: Rule-oriented method for parameterized computer-aided design, Computer-Aided Design, 24 (10), 1992, 531–540. http://dx.doi.org/10.1016/0010-4485(92)90040-h

[8] Westhues, J.: SketchFlat, a Constraint-Based Drawing Tool, 2007. http://cq.cx/dl/sketchflat-internals.pdf