

Spatial Inference – Learning vs. Constraint Solving

Carsten Gips, Petra Hofstedt, Fritz Wysotzki
{cagi, ph, wysotzki}@cs.tu-berlin.de

Berlin University of Technology

Abstract. We present a comparison of two new approaches for solving constraints occurring in spatial inference. In contrast to qualitative spatial reasoning we use a metric description, where relations between pairs of objects are represented by parameterized homogenous transformation matrices with numerical (nonlinear) constraints. We employ interval arithmetics based constraint solving and methods of machine learning in combination with a new algorithm for generating depictions for spatial inference.

1 Introduction

Understanding and interpretation of textual descriptions of real world scenes are important for many fields, e.g. navigation and route descriptions in robotics [13, 15], in CAD or in graphical user interfaces (e.g. “The xterm is right of the emacs.”).

In contrast to qualitative approaches to spatial reasoning [6, 8], in [3] we presented a new metric approach to spatial inference based on mental models ([3, 12]). Starting from sentences like “The lamp is left of the fridge.” we try to create a mental model which represents the described spatial situation. This approach uses a directed graph, where the nodes represent the objects and the edges represent the given relation, e.g. `left(fridge, lamp)`, between two objects. From this model it is possible to infer relations which were not initially given in the text or to generate depictions compatible with the description.

The semantics of the relations is given by homogenous transformation matrices with constraints on the variables. As shown in [16], inference of a relation between two objects is done by searching a path between the objects and multiplying the matrices on this path. Thereby constraints containing inequalities and trigonometric functions must be propagated and verified. Only in some rare cases we can solve these constraints analytically. Furthermore in [16] a simple algorithm for generating depictions is proposed. It is restricted to default positions of objects and to rotations of multiples of $\pi/2$. Moreover, this approach requires to keep lists with possible positions for every object.

Our aim is now to find a method to solve this kind of constraints and to generate depictions without the restrictions mentioned above. In this paper we sketch two approaches to spatial reasoning: First, we use cooperative constraint

solving methods, in particular interval arithmetics, where inference is supported by further solving methods. Similarly, in [5] the constraint solver Parcon has been integrated into a 2D real-time animation environment. While [5] aims at one particular solution for the placement of objects, our interest is on the whole solution space. Moreover, we will see that our approach is more flexible because our system allows the integration of new solvers, like Parcon itself, in a simple way. This is advantageous if new requirements to the problem formulation appear. Our second approach on spatial reasoning applies machine learning in combination with a new algorithm for depiction generation.

This work is structured as follows: We start with an introduction into the description of spatial relations by means of examples in Sect. 2. In Sect. 3 we demonstrate an approach of directly solving spatial constraints with cooperating constraint solving methods. An alternative approach is to use machine learning as described in Sect. 4. In Sect. 5 we compare the different approaches wrt. their advantages and disadvantages and show perspectives for future work.

2 Expressing Spatial Relations

Our aim is to describe scenes with the help of spatial relations and, given such spatial descriptions, to generate appropriate scenes or to find out that no actual scene according to the current description exists, resp. For simplification, we consider only 2D scenes and represent objects by appropriate geometric figures.

Example 1. Given the spatial relations `right(cupboard, lamp)` and `left(fridge, lamp)` a possibly intended scene is given in Fig. 1.

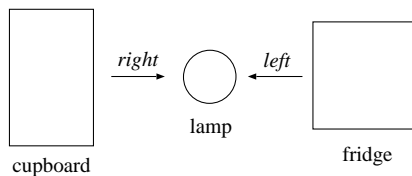


Fig. 1. Example scene 1: `right(cupboard, lamp)` and `left(fridge, lamp)`

However, this is not the only possible scene. The question, whether a scene is an actual representation of the given set of relations, depends of course on the intended meaning of the relations. For example, the relation `right(cupboard, lamp)` does not necessarily describe only a situation, where the lamp is straight right from the cupboard. The lamp could be situated for example downright or upright from the cupboard as well.

For our purposes we investigated scene descriptions based on the relations `left/2` and `right/2`, which describe the placement of an object left resp. right from another one, the relations `front/2` and `behind/2` which place objects in

front of or behind other objects and the relation `at_wall/2` for describing the placement of an object parallel to a wall with a fixed maximum distance. Further relations provide that an object is situated in a given room (`in_room/1`) and they ensure, that objects do not overlap (`not_overlap/n`).

As mentioned above, in contrast to qualitative techniques [6, 8] for spatial reasoning we use a metric approach [3, 4] known from the area of robotics ([2]). At this, we associate with every object a coordinate system, its form and size. Relations between pairs of objects are represented by their transformation matrices. Thus, the current coordinates of an object depend on its relation, i.e. orientation and distance, to its relatum, which may be different in different constraints. That means, changing the relatum of an object we need to transform its coordinates using the corresponding matrix.

Let us consider the relation `right/2` in detail. The relation `right(cupboard, lamp)` places the lamp, which is the referent, right wrt. the cupboard, its relatum. The cupboard is the origin of the relation. The lamp as referent can be placed within the bisectors of the right angles of the cupboard. Figure 2 illustrates this situation: The lamp (represented by the circle) is placed `right` of the cupboard (represented by the rectangle).

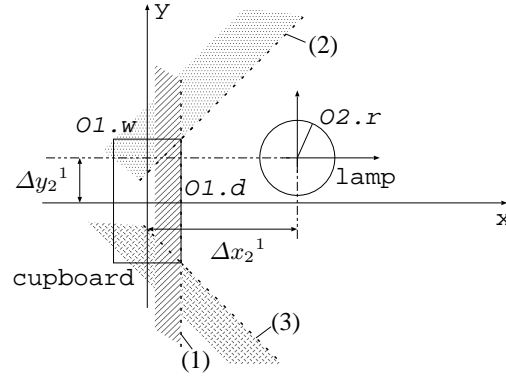


Fig. 2. The relation `right(cupboard, lamp)` in detail

Mathematically we can describe the relation `right(O1, O2)` ($O1$ and $O2$ stand for the cupboard and the lamp, resp.) by the following inequalities:

$$\Delta x_2^1 \geq O1.w + O2.r \quad (1)$$

$$\Delta x_2^1 \geq \Delta y_2^1 + O1.w - O1.d + \sqrt{2}O2.r \quad (2)$$

$$\Delta x_2^1 \geq -\Delta y_2^1 + O1.w - O1.d + \sqrt{2}O2.r \quad (3)$$

At this, $O1.w$ and $O1.d$ represent the width and the depth of the rectangle, i.e. the cupboard, and $O2.r$ stands for the radius of the lamp. The distances of the object $O2$ in the x - and y -directions from the relatum $O1$ are denoted by

Δx_2^1 and Δy_2^1 , resp. At this, the lower index is associated to the referent and the upper index to the relatum.

Note, that for the relation `right/2`, like for every spatial relation, in general the formulae differ depending on the form of the relata and referents.

In the remainder of this paper we will use the following example to demonstrate our work.

Example 2. We extend Example 1 giving further constraints which describe the sizes of our objects and a room. We require the objects to be inside the room and not to overlap each other. A corresponding scene is given in Fig. 3.

```

room.w ∈ [4.0, 4.5], room.d ∈ [4.0, 4.5], lamp.r = 0.3,
fridge.w ∈ [0.4, 0.5], fridge.d ∈ [0.4, 0.5]
cupboard.w = 0.4, cupboard.d ∈ [1.0, 1.2],
left(fridge, lamp), right(cupboard, lamp),
in_room(fridge), in_room(lamp), in_room(cupboard),
not_overlap(fridge, lamp, cupboard)

```

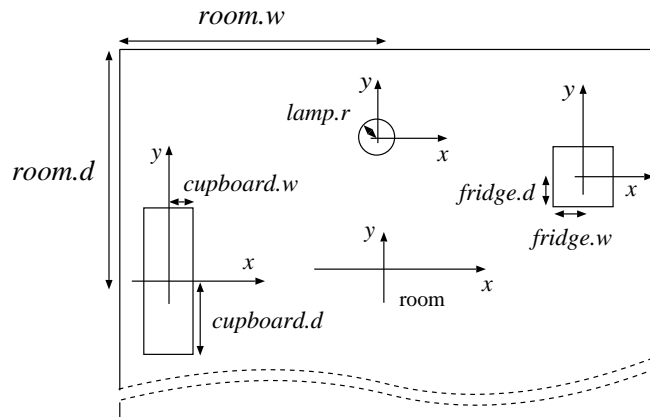


Fig. 3. Example scene

3 Constraint Solving

An approach to reason about such kind of spatial knowledge is to directly use constraint solvers appropriate for this kind of problem.

Representing spatial situations requires interval constraints, because the sizes of the objects and the room are often not given in detail, and we need further

arithmetic constraints. Thus, for this application it is appropriate to use an interval arithmetic solver, like the Brandeis Constraint Solver [1, 9], which handles basic operations, like addition and multiplication on rational intervals, as well as trigonometric and logarithmic functions.¹ Note that this solver is incomplete, which means, that it does not detect every unsatisfiable constraint. Thus, it could be useful to employ a further solver for a part of the constraints. Since we are interested in getting particular scenes, i.e. generating depictions, we, moreover, would like to use a constraint solver for guessing the placement of objects. Thus, the cooperation of different constraint solvers is desirable here.

In [10] a general scheme for the cooperation of different constraint solvers is proposed, in [11] an according implementation is shortly sketched. The problem of handling spatial knowledge is a typical application for this system.

The cooperating system allows to integrate arbitrary black box solvers providing a typical solver interface. A meta mechanism coordinates the work of the individual solvers and the information exchange between them. The system can be configured by the user wrt. strategies and for evaluation of experiments. It is possible to define a wide range of different cooperation strategies according to the current requirements. Using this cooperating system, it is possible to deal with hybrid constraints over different constraint domains, and thus to describe and to solve problems which none of the single solvers can handle alone.

An input file for the implementation (see [11] for details) describing the problem of Example 2 is given in Fig. 4. In the first line in the `[solver]` part we specify the constraint solver to be used: the Brandeis Interval Arithmetic Solver `ISolver`.²

In the file, `01` stands for the cupboard, and `02` and `03` represent the lamp and the fridge, resp. Instead of Δx_2 ¹ we write `dx21`. In the `[constraints]` part we give constraints describing the sizes of the room and our objects (lines (1)-(4)) and the constraints which (quantitatively) describe the relations `right(cupboard, lamp)` (lines (7)-(9)) and `left(fridge, lamp)` (lines (12)-(14)). Further constraints (lines (5)-(6) and (10)-(11)) express a transformation of object coordinates because of changes of the relata and/or referents. Constraints for placing the objects in the room and for ensuring that the objects do not overlap are left out here.

Our system computes for this input file the following solution space:

$$\begin{array}{ll}
 4.0 \leq \text{room.w} \leq 4.5 & 4.0 \leq \text{room.d} \leq 4.5 \\
 -4.1 \leq \text{dx10} \leq 2.7 & -3.5 \leq \text{dy10} \leq 3.5 \\
 -3.4 \leq \text{dx20} \leq 3.4 & -4.2 \leq \text{dy20} \leq 4.2 \\
 -2.7 \leq \text{dx30} \leq 4.1 & -4.1 \leq \text{dy30} \leq 4.1
 \end{array}$$

¹ For reasons of simplicity, we do not handle trigonometric nor logarithmic functions in this paper. Nevertheless they can be used, for example, to describe rotations of objects in our approach.

² Configurations of the used solvers and the solving strategy are allowed to be given in the input file by the user. Here they are left out.

```

[solver]
  ISolver = solver.brandeis.Brandeis
[constraints]
  room.w, room.d in [4, 4.5];          # width and depth of the room      (1)
  01.w = 0.4; 01.d in [1.0, 1.2];     # size of the cupboard          (2)
  03.w, 03.d in [0.4, 0.5];          # size of the fridge            (3)
  02.r = 0.3;                          # radius of the lamp            (4)
                                     # the coordinates of the lamp wrt. the cupboard
  dx21 = dx20 - dx10;                  (5)
  dy21 = dy20 - dy10;                  (6)
                                     # the relation right(cupboard, lamp):
  dx21 >= 01.w + 02.r;                  (7)
  dx21 >= dy21 + 01.w - 01.d + (2 ^ 0.5) * 02.r; (8)
  dx21 >= - dy21 + 01.w - 01.d + (2 ^ 0.5) * 02.r; (9)
                                     # the coordinates of the lamp wrt. the fridge
  dx23 = dx20 - dx30;                  (10)
  dy23 = dy20 - dy30;                  (11)
                                     # the relation left(fridge, lamp):
  dx23 <= - 03.w - 02.r;                 (12)
  dx23 <= dy23 - 03.w + 03.d - (2 ^ 0.5) * 02.r; (13)
  dx23 <= - dy23 - 03.w + 03.d - (2 ^ 0.5) * 02.r; (14)

```

Fig. 4. Constraint file describing Example 2

At this, dx_{10} , dx_{20} , and dx_{30} represent the distance of the objects 01, 02, and 03 resp. from the coordinate origin of the room as relatum. The computed solution space restricts the space for placing the objects. Figure 5 shows the area (shaded) for a possible placement of the cupboard (i.e. its center (dx_{10} , dy_{10})) in the room. The depth and width of the room ($room.d$ and $room.w$) are between 4 and 4.5 length units. The cupboard can be placed on the left wall ($dx_{10} = -4.1$) of the room (of width 4.5). Because of the relations `right(cupboard, lamp)` and `left(fridge, lamp)`, it cannot be situated on the right wall, there must be some distance for placing the other objects, thus $dx_{10} \leq 2.7$ holds. For dy_{10} holds $-3.5 \leq dy_{10} \leq 3.5$, this is due to the depth $01.d$ of the cupboard.

However, we are not only interested in restricting the solution space of the variables, but we also want to derive particular depictions. Thus, we introduce constraints which propose possible positions of the objects and a solver to handle such constraints.

Besides the interval arithmetic solver `ISolver` we use now a finite domain constraint solver `FDSolver` for the computation of depictions. A finite domain solver is able to handle constraints over finite domains, in particular constraints over finite sets. In our case, this is used to handle (finite) sets which describe possible object locations. The corresponding constraints of the `FDSolver` are given in Fig. 6. While the constraints of the lines (1)-(14) as given in Fig. 4 are constraints of the interval solver `ISolver`, the set constraints of lines (15) and

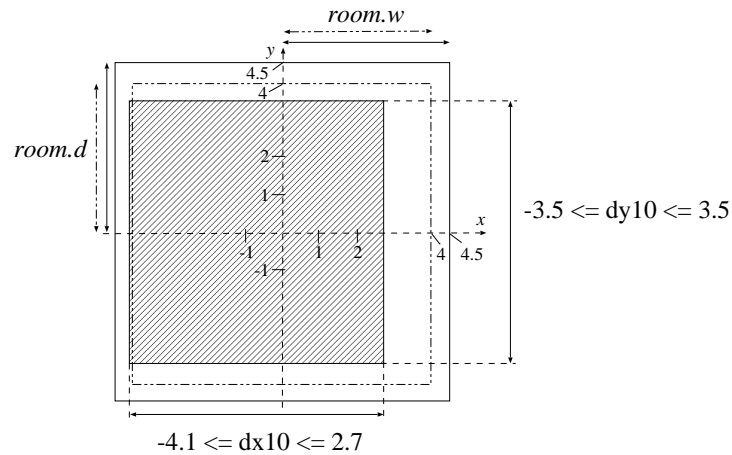


Fig. 5. Possible positions for a placement of the cupboard 01 in the room

```

[solver]
  FDSolver = solver.csplib.CSPlib
  ISolver = solver.brandeis.Brandeis
[constraints]
...
dx10, dx20, dx30 inn {-3.0, 0.0, 3.0}; (15)
dy10, dy20, dy30 inn {-1.5, 1.5}; (16)

```

Fig. 6. Extension of the constraint file for generating depictions

(16) in Fig. 6 are constraints of the finite domain solver `FDSolver`. Both solvers understand equality constraints between variables and ground values like the constraint `01.w = 0.4` of line (2) and the constraint of line (4) which are, thus, assigned to both solvers.

Computing solutions, the cooperating system delays the generation of depictions by the finite domain constraint solver to avoid the splitting of the solution space as long as possible. This principle, known from other systems, e.g. [7], avoids unnecessary computations caused by nondeterministic search.

Now, the computation yields the 4 solutions represented in Fig. 7. Looking at these depictions as solutions, we see that the objects are always placed in the sequence cupboard – lamp – fridge. This is due to the constraints `right(cupboard, lamp)` and `left(fridge, lamp)`. For the fridge and the lamp the `dy`-values are always the same while the `dy`-value of the cupboard may differ from that. The reason is that the lamp can be placed within the bisectors of the left angles of the fridge only and a depiction with different `dy`-values for the lamp and the fridge is, thus, no solution. The cupboard has a larger depth and thus

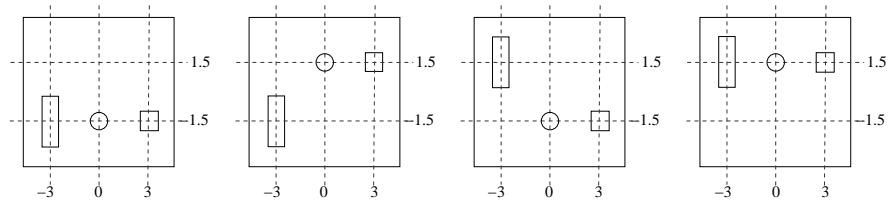


Fig. 7. Generated depictions

the bisectors of its right angles include the placement of the lamp at another dy -value as well.

Even if we can describe our problem in this way, and we are able to compute depictions, specifying the problem using the pure mathematical description is not very comfortable and causes the user to write a large number of constraints. To get nearer to a human description, it is possible to integrate a new solver which simply performs macro rewriting for particular constraints, for example it replaces a constraint `right(cupboard, lamp)` by the corresponding mathematical constraints (lines (5)-(9) in Fig. 4). In this way the number of constraints which the user must provide can be decreased and the readability is improved.

Using the described system of cooperating solvers allows to specify spatial problems in an explicitly quantitative way and as well by introducing translation functions in a more natural way. It is possible, on the one hand, to restrict the solution spaces and, on the other hand, to compute depictions, i.e. example scenes which satisfy a given description, by introducing a finite domain constraint solver and appropriate constraints. Times for computing solutions are acceptably of a fraction of a second. The system allows to introduce new black box solvers according to the current problem. If the kind of problem changes or there are new requirements to the problem formulation, the integration of new constraints and new appropriate solvers is possible in a simple and comfortable way.

However, the approach has two disadvantages: At first, the available solvers are often incomplete. That means, that a constraint solver is not able to detect every unsatisfiable constraint. Incompleteness is due to the underlying constraint domain and the associated solving algorithms. Even if, due to information exchange between the solvers, the cooperating system is able to detect inconsistencies of a given constraint conjunction which the individual solvers are not able to detect, in general, the incompleteness of the individual solvers is taken over by the overall system. The integration of further constraint solvers, for example, a solver for linear arithmetics based on the simplex method, could allow to stronger restrict the solution space, however it does not actually solve the incompleteness problem in general. The other problem is, that with increasing complexity of the spatial scene and with an increasing number of objects and, thus, of constraints and variables the computations become more time consuming.

Thus, in the following Sect. 4 we consider another but similar approach for describing and solving spatial scenes which is as well promising. In Sect. 5 we discuss their relations and a possible interaction.

4 Machine Learning

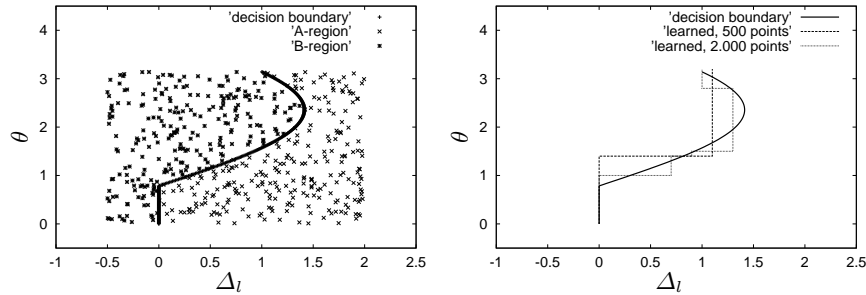
Instead of solving the constraints directly we try to learn the decision function $A(x_1, \dots, x_n)$ which decides whether a vector $x = (x_1, \dots, x_n)$ of the configuration space belongs to a region where the predicate A is true, i.e. the corresponding constraints are satisfied. In our problem domain (objects located in a room), the constraints consist of equations and inequalities containing trigonometric functions which lead to computational difficulties well known from robotics [2].

Before employing machine learning algorithms, we have to construct a training set by exploiting the given constraint description or by using results of psychological studies. These datasets consist of preclassified feature vectors where each variable of the constraints represents an attribute, i.e. a dimension in the feature space. In the following, we will use “class A ” for the regions where a constraint A is satisfied (and “class B ” otherwise). By means of the training sets algorithms of classification learning (e.g. decision tree learning like CAL5 or neuronal nets like Dipol, see [14] for both) construct classifiers now. These decide the class membership of an arbitrarily chosen point x (not necessarily contained in the training set) by inductive generalization. This decision is very fast in comparison to using the system of equations and inequalities for a direct computation of the class membership and therefore it is especially suited for on-line tasks. In addition, the decision rules make the regions of the configuration space, where the constraints are satisfied, (approximately) explicit.

Generating training sets in order to get an acceptable approximation of the decision boundary is also known as “learning by exploration” or “active learning” in literature and is subject of current research ([17]).

4.1 Learning Spatial Relations with CAL5

Decision tree learners approximate the class boundaries piecewise linearly by axis-parallel hyperplanes. Usually, there is a generalization error due to the unavoidable approximation of the boundaries between the A -regions and the B -regions. This error can be measured using a test set of classified example vectors different from the training set. By increasing the number of training data (and simultaneously shrinking a certain parameter of CAL5) the generalization error can be reduced (i.e. the accuracy of the class boundary approximation can be made arbitrarily high), and in the limit of an infinite training data set the error becomes zero. This is shown in the following example, which is taken from [4]. There we learned a constraint $\sin \theta = \Delta_l/2 \pm \sqrt{1/2 - \Delta_l^2/4}$, which expresses the relation “A bar is right or behind of an object O ”. The point S is the origin of the coordinate system of object O . Thus neither O nor S occur in the inequality. Δ_l is the difference of the displacement of a particular point of the bar in x -direction and y -direction of S divided by the length of the bar.



(a) *A*-region and *computed* boundary (b) Boundaries for 500 and 2.000 points

Fig. 8. Learned vs. computed boundaries

Figure 8(a) shows the *A*-region with computed boundary vs. the *B*-region. In Fig. 8(b) the solid real (computed) boundary of the *A*-region is compared with the dashed learned boundaries for a training set of 500 and 2.000 points, resp. It can be seen that the generalization error of the obtained decision tree shrinks with an increasing number of points for learning. However, in practice we reach the manageable limit at 200.000 training examples. The constraints of our relations (like `right/2` for `circles` and `rectangles`, see Sect. 2) affect up to seven parameters, thus, we obtain a configuration space of up to seven dimensions. Thereby we get approximately ten data points per dimension³ in average. Because of this sparsely populated configuration space, both the training and generalization errors are rather high. This is shown in Tab. 1, where we used 200.000 uniformly distributed data points for learning and 5.000 points for testing.

Relation ⁴	Number of class <i>A</i> leaves	Points		Test error		
		in <i>A</i>	in <i>B</i>	for <i>A</i> only	for <i>B</i> only	overall
<code>atwall(r, r)</code>	143	15.909	184.091	10 %	1 %	2 %
<code>front(c, r)</code>	1.984	84.285	115.715	22 %	14 %	17 %
<code>right(c, r)</code>	1.702	84.557	115.443	20 %	13 %	16 %
<code>right(r, c)</code>	2.079	87.550	112.450	26 %	14 %	19 %

Table 1. Results of the learning process for some spatial relations

³ Supposed we obtain the same number of data on each dimension (like a grid), the 7th root of 200.000 yields approximately six data points on each dimension. Note, this is not a correct calculation but a simple estimation.

Furthermore we have to take care for a sufficient large subspace of configuration space. The result of a too small scope of training data vs. the area of the example room is shown in Fig. 9. The resulting classifier does not cover the intersection of the room area and the `right/2` sector.

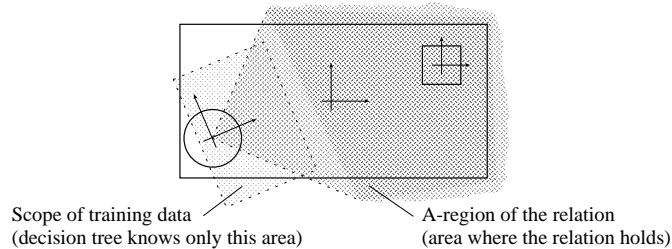


Fig. 9. Scope of training data not adapted to the size of the room

The benefits of our learning approach are to get a new, easier representation of the decision boundary (i.e. the constraints). The new representation contains the solution of the constraints (i.e. the A-regions), and the accuracy of the approximation can be arbitrary high. The drawback, however, is the problem of generating suitable data sets.

4.2 Generating Depictions

In the previous section we transformed the constraints of the spatial relations into a new representation (i.e. the learned classifiers). As it was our aim to solve the spatial constraints, we present now an algorithm (see Fig. 10) for generating depictions, which uses the learned CAL5 decision trees.

As mentioned above, for every needed relation and for every pair of object types, corresponding (sub-)relations must be learned. Since we are interested in those regions in the parameter space where the relation holds, we do some preprocessing. This means, for each leaf node of the decision tree in the tree with class *A* the admissible intervals for every parameter are being extracted. Note, that each depiction is a point in the configuration space. These points are represented by vectors, which contain all variables of the problem. Recall that our relations are binary. Thus, we get three cases: both objects are ‘unknown’, one object is already placed or both objects are placed. In the first case, we place one object randomly in the room (lines 09 and 10 in Fig. 10). This leads us to the second case. There we pick a class *A* leaf of the tree and compute the size, relative position and relative orientation of the other object by assigning values to the remaining variables of this object within the intervals of the chosen leaf (lines 11 and 12). If the collision check in line 13 fails (for both, case one and two), we repeat the procedure up to *k* times (block between lines 08 and 14). If

⁴ ‘r’ denotes a `rectangle`, and ‘c’ a `circle`

we do not have admissible values after the k th trial (line 15), we suppose that the current relation cannot hold in combination with the others and we reject the depiction generated so far. However, there may exist solutions. Actually we cannot distinguish between the case “no solution” and “disadvantageous values”. So if we reject the vector, we have to start again with the first relation. For practical reasons we work instead on a number of object constellations in parallel. This means, we are starting with v empty vectors and apply the algorithm to each vector. Thus, we obtain in each step at most v depictions (valid for the relations processed so far). If we have to drop a vector, we still have $v_n - 1$ other scenes in step n . It is not critical to discard “disadvantageous values” because usually the scenes are underconstrained. In the last case both objects are already placed and we have to check, whether the values of the objects range in the intervals represented by at least one leaf (lines 05 and 06). If not, the relations do not hold, at least for the calculated values.

This procedure is repeated for every relation with the remaining objects, which satisfy the relations processed in the former steps. Finally we obtain up to v depictions according to the given spatial description. In the case, that we have found no depiction, we have to assume that the constraints are unsatisfiable.

Up to now each decision tree represents only the constraints for the particular relations with the two objects of the specified forms. The background knowledge (e.g. the objects must not overlap) was not learned but checked after every step explicitly. In general, it is possible to learn the background knowledge constraints as well and to check them like the other relations.

The parameters v and k depend on the given relations and on the number of relations. They have to be chosen large enough to get a correct answer (“There is no solution.” or “We have found at least one.”) by some probability. At the same time, one should choose rather small v and k , because by increasing the parameters the calculation time increases, too. So they have to be chosen in relation to the problem to be solved.

As shown in Tab. 2, the more relations to solve, the higher v has to be. A value of 100 for k seems to be a good choice. The number of trials per valid solution increases exponentially in the number of relations to solve. Not shown, but critical is the processing sequence. Very restrictive relations like `atwall/2` should be solved at the beginning.

Example 3. Supposed we have two relations, `right(cupboard, lamp)` and `atwall(wall1, cupboard)`. Now we fulfill first the `right/2` relation. Therefore the cupboard may be placed somewhere in the room. After that we cannot satisfy `atwall/2`, because the cupboard should be placed near `wall1`, but actually it is already placed in the room. So we would have to increase v , but nevertheless the probability to get a solution is very small.

5 Discussion

In the previous sections we sketched two approaches to spatial reasoning. First, we have shown how to use a system of cooperating constraint solvers, where

Depiction generation algorithm

```
INPUT: number  $v$  of initial vectors and number  $k$  of trials
       relations  $r$  that have to hold
OUTPUT: up to  $v$  depictions, where all relations  $r$  hold
ALGORITHM:
01 foreach relation  $r$ :
02   identify objects and object types by object descriptions
03   load the corresponding (pruned) decision tree
04   foreach vector  $v$ :
05     if both objects were placed
06     then check whether relation  $r$  holds
07     else
08       repeat up to  $k$  times:
09         if both objects are new
10         then place first object randomly in room
11           pick area (random according to weight of leaf)
12           assign values to variables within intervals of leaf
13           check non-overlapping with other objects and walls
14         until check passes
15         if no success
16         then drop vector  $v$ 
17   show remaining vectors (depictions)
```

Fig. 10. Algorithm for generation depictions using decision trees

the inference of an interval arithmetic solver is supported by further solving methods. In the second approach we applied machine learning in combination with a new algorithm for depiction generation to this kind of problem.

The direct constraint solving approach may at first seem to be evident. Using our system of cooperating solvers the restriction of the solution space is possible as well as to compute particular depictions. The system is comfortable in use and can be extended by new solvers in a simple way. However, the disadvantages of the possible incompleteness of the solvers, and, thus, as well of the overall system, and an increasing time effort by increasing complexity of the spatial scene must be taken into consideration.

Using the learning approach yields decision trees, which are very well interpretable. The approximation of the decision boundaries may be (at least in principle) arbitrarily high. Generating suitable training sets, however, is not trivial. The depiction generation algorithm employs the decision rules for restricting the space of possible solutions. In the limit of generating an infinite number of depictions (i.e. exhaustive search) the algorithm finds every possible solution. Because the processing sequence of the relations is critical, we may find no solution, although there is one. However, the scene descriptions in this problem domain are usually underconstrained, and, thus, it is usually not a problem

Combination of relations	suitable v/k	number v per valid depiction (average)
single relation, e.g. <code>right(steffi, cupboard)</code>	100/10	2
two relations, e.g. <code>right(steffi, cupboard)</code> <code>front(steffi, fridge)</code>	1.000/100	10
three relations, e.g. <code>right(steffi, cupboard)</code> <code>left(fridge, lamp)</code> <code>right(cupboard, lamp)</code>	1.000/100	61
four relations, e.g. <code>right(steffi, cupboard)</code> <code>left(fridge, lamp)</code> <code>right(cupboard, lamp)</code> <code>front(steffi, fridge)</code>	1.000/100	375
five relations, similar to above	10.000/100	638

Table 2. Some test runs and typical results of our algorithm

to find an alternative solution by constructing another sequence (i.e. following another path in the problem space).

Comparing the approaches of learning on the one hand and constraint solving on the other hand, at first they seem to be very different. However, they are not: The constraint solving approach takes the given constraints, checks their satisfiability, and tries to compute solutions. The learning approach generates rules for constraint decision using training data. This means, however, that these rules together with the search procedure build as well a constraint solver for the provided constraints; its behavior depends on the kind, amount and complexity of training data.

We think, that the direct solving approach can be useful for a more large-grain scene estimation, while the learning approach can allow (dependent on the provided training data) a more fine-grain consideration and as well looking at exceptions. This yields further research perspectives: Constraint solving could be used for an early exclusion of unsatisfiable scene descriptions. Because of the possible incompleteness of the solvers, it will not in general detect every inconsistency. For a more fine-grain elaboration of the remaining areas the learning approach could yield then better results. A second promising idea is to use constraints and constraint solving to prestructure the rule set for the learning method and to tune particular parameters afterwards using the learning mechanism with selected training data. This may allow to reduce the training cost and increase the speed of the depiction generation.

References

1. The Brandeis Interval Arithmetic Constraint Solver, March 2002. Available from <http://www.cs.brandeis.edu/~tim/>.
2. A. P. Ambler and R. J. Popplestone. Inferring the Positions of Bodies from Specified Spatial Relationships. *Artificial Intelligence*, 6:157–174, 1975.
3. B. Claus, K. Eyferth, C. Gips, R. Hörnig, U. Schmid, S. Wiebrock, and F. Wysotzki. Reference Frames for Spatial Inferences in Text Comprehension. In C. Freksa, C. Habel, and K. F. Wender, editors, *Spatial Cognition*, LNAI 1404. Springer, 1998.
4. P. Geibel, C. Gips, S. Wiebrock, and F. Wysotzki. Learning Spatial Relations with CAL5 and TRITOP. Technical Report 98-7, TU Berlin, 1998.
5. P. Griebel, G. Lehrenfeld, W. Mueller, C. Tahedl, and H. Uhr. Integrating a Constraint Solver into a Real-Time Animation Environment. In *Proc. of the 1996 IEEE Symposium on Visual Languages*, September 1996.
6. H. W. Guesgen. Spatial Reasoning Based on Allen's Temporal Logic. Technical Report TR-89-049, ICSI, Berkeley, Cal., 1989.
7. S. Haridi, S. Janson, J. Montelius, T. Franzen, P. Brand, K. Boortz, B. Danielsson, B. Carlson, T. Keisu, D. Sahlin, and T. Sjöland. Concurrent Constraint Programming at SICS with the Andorra Kernel Language. In P. Kanellakis, J.-L. Lassez, and V. Saraswat, editors, *First Workshop on Principles and Practice of Constraint Programming – PPCP'93*, 1993.
8. Daniel Hernández. *Qualitative Representation of Spatial Knowledge*. LNAI 804. Springer, 1994.
9. T.J. Hickey, M.H. van Emden, and H. Wu. A Unified Framework for Interval Constraints and Interval Arithmetic. In M. Maher and J.-F. Puget, editors, *Principles and Practice of Constraint Programming – CP'98*, LNCS 1520. Springer, 1998.
10. P. Hofstedt. Better Communication for Tighter Cooperation. In *First International Conference on Computational Logic*, LNCS 1861. Springer, 2000.
11. P. Hofstedt, E. Godehardt, and D. Seifert. A Framework for Cooperating Constraint Solvers - A Prototypic Implementation. In E. Monfroy and L. Granvilliers, editors, *Workshop on Cooperative Solvers in Constraint Programming - CoSolv*, 2001.
12. P. N. Johnson-Laird. *Mental Models: Towards a Cognitive Science of Language, Inference and Consciousness*. Cambridge University Press, Cambridge, 1983.
13. T. Jörding and I. Wachsmuth. An Antropomorphic Agent for the Use of Spatial Language. In *Proceedings of ECAI'96-Workshop on Representation and Processing of Spatial Expressions*, pages 41–53, 1996.
14. G. Nakhaeizadeh and C. C. Taylor, editors. *Machine Learning and Statistics - The Interface*. Wiley, 1997.
15. T. Röfer. Routemark-based Navigation of a Wheelchair. In *Third ECPD International Conference on Advanced Robotics, Intelligent Automation and Active Systems*, Bremen, 1997.
16. S. Wiebrock, L. Wittenburg, U. Schmid, and F. Wysotzki. Inference and Visualization of Spatial Relations. In C. Freksa, W. Brauer, C. Habel, and K. Wender, editors, *Spatial Cognition II*, LNAI 1849. Springer, 2000.
17. S. Wiebrock and F. Wysotzki. Lernen von räumlichen Relationen mit CAL5 und DIPOL. Technical Report 99-17, TU Berlin, 1999.