Strings and holes: an exercise on spatial reasoning

Pedro Cabalar[1] and Paulo Santos[2]

[1] Department of Computer Science
Corunna University, Galicia, Spain
cabalar@udc.es

[2] Department of Electrical Engineering
Centro Universitário da FEI, São Paulo, Brazil
psantos@fei.edu.br

Abstract. This paper investigates the challenging problem of encoding the knowledge and reasoning processes involved in the common sense manipulation of physical objects. In particular we provide a formalisation of a domain involving rigid objects, holes and a string within a reasoning about actions and change framework. Therefore, this work investigates the formalisation and reasoning about flexible objects and void space (holes) in a single domain. Preliminary results of automated reasoning within this domain are also presented.

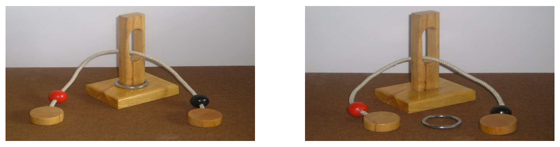
1 Introduction

The field of qualitative spatial reasoning (QSR) [10] attempts the formalisation of spatial knowledge based on primitive relations defined over elementary spatial entities. One of the best known QSR theories, for instance, is the Region Connection Calculus [6], which is a first order axiomatisation of space based on regions and the connectivity relation. Other representations of spatial knowledge include theories about shape [4], distance [3], position [1] amongst others as surveyed in [2]. However, the use of qualitative spatial knowledge within a planning system remains largely neglected.

One possible reason for the lack of problem solving methods handling qualitative spatial knowledge may be connected to the fact that research on QSR has being conducted independently from research on reasoning about actions and change (RAC) and AI planning (apart from exceptions such as [9] and [8]). One of the motivations for the present work is to approximate RAC to reasoning about spatial knowledge by investigating the formalisation and automatic solution of a challenging spatial puzzle.

This paper assumes the puzzle called *The Fisherman's Folly* (Figure 1) that involves spatial entities such as strings, posts, rings, spheres and holes (through the last ones some (but not all) domain objects can pass). The Fisherman's Folly puzzle consists in going from the configuration shown in Figure 1(a) to the configuration in Figure 1(b) by moving the objects positions respecting some domain restrictions. In this sense, the assumed puzzle is similar to the classic 8-puzzle; however, in the present work the domain objects have non-trivial spatial characteristics (such as flexibility and permeability through holes).

The elements of the puzzle are a holed post fixed to a wooden base, a string, a ring, a pair of spheres and a pair of disks. The disks and spheres are attached to the string, along which the latter can move whereby the former is fixed to the string endpoints.



(a) The initial state. (b) The goal state.

Fig. 1: A spatial puzzle: the Fisherman's Folly.

In the initial state (shown in Figure 1(a)) the post is in the middle of the ring, which is supported on the post's base. On the other hand, the string passes through the post's hole in a way that one sphere and one disk remain on each side of the post. It is worth pointing out that the spheres are larger than the post's hole, therefore the string cannot be separated from the post without cutting either the post, or the string, or destroying one of the spheres. The disks and the ring, in contrast, can pass through the post's hole. The goal of this puzzle (depicted in Figure 1(b)) is to find a sequence of transformations of the spatial configuration of the puzzle's objects such that the ring is freed from the system *post-base-string*, maintaining the physical integrity of the domain objects. In fact, the goal state is not fixed to the one shown in Figure 1(b). In order to be considered a solution, it is sufficient to move the ring completely out of the rest of the system, regardless the final configuration of the remaining domain objects.

The complexity imposed by the distinct states of the string allied to the existence of holes in the domain objects makes the formalisation and reasoning about this domain a challenging problem. In order to provide a formal account of the spatial relations involved in the Fisherman's Folly we need to consider in our formalisation (and reasoning processes) the holes in objects, such as the post's hole and the space limited by the ring. This calls for assuming holes as real objects, therefore having the same ontological status as spheres and disks. Reasoning about holes and holed objects has been discussed in detail in [11] from a topological standpoint. However, to the best of our knowledge, the present paper presents the first approach that investigates the problem of how these entities could be engaged in actions.

The string brings a further source of complexity which comes from the related infinity of distinct configurations due to its flexibility. The problem of incorporating knowledge about strings and string manipulation has been tackled in [5] where a robotic system capable of learning to tie a knot from visual observation is proposed, this system is called *the Knot Planning from Observation* (KPO) paradigm. In KPO each state of a string is represented by a matrix encoding the string segments, which are defined by the portion of the string that lies in between its endpoints and points where it crosses over itself. Actions on flexible objects in this context were defined as an extension of the Reidemeister moves in knot theory [7]. This representation is suitable for the identification of string states from a computer vision system; however, it falls short in the context of problem solving, which is the main purpose of the present paper. In this work, we propose a representation for string states that takes into account other objects (including holes) that may be related to the string in the domain. In contrast to the work proposed in [5], we do not take into account knots. Incorporating some of the ideas of the KPO paradigm in our work shall be investigated in the future.

In summary, the purpose of this paper is to investigate the formalisation and autonomous solution of a spatial domain involving holes and a flexible string, contributing with a novel benchmark problem for common sense knowledge representation.

This paper is organised as follows: the next section introduces the formal representation we use for the domain objects; Section 3 discusses the basic actions that operate on the puzzle; some details on a simple Prolog implementation are presented in Section 4; and, finally, Section 5 concludes this paper.

2 Domain objects

A straightforward classification of the objects in the puzzle would lead to six sorts: the spheres, the disks, the string, the ring, the holed post and the post base. Although the post and its base form a same object, they are clearly distinguishable. In fact, something similar happens with the disks and the string: they form a same "tandem," but a disk and a string clearly have different properties. For the spheres there is no doubt: although limited by the string and the disks in the domain, they are topologically independent objects.

Although the commonsense knowledge about all these sorts can be very rich (for instance, we know that a sphere can roll, a string can form knots, etc) we must focus however on their *relevant* properties for obtaining a satisfactory solution to the puzzle, applying somehow an Occam's razor criterion. Besides, it is also important to fix the very same idea of *satisfactory* solution. Our criterion in this sense is trying to obtain a qualitative description of the movements to be performed, in similar terms to those one could find in a textual description of the solution written in natural language. A related observation is that the same puzzle problem could actually be built with objects of different nature to those commented before. For instance, the spheres and the disks could also be boxes of different sizes, the post base could be a large disk, etc. These small changes are not really essential. However, when a human describes the puzzle solution, she immediately talks about passing objects through *holes*. The puzzle, in fact, deals with four holes: the post hole (ph), the ring hole (r), and the two sphere holes (s_1, s_2). Furthermore, in the last three cases, the human will usually identify the hole with its host object so that, for instance, she would simply talk about "passing a sphere through the ring" and not through the "ring hole." This apparently subtle distinction may help to drastically simplify the problem representation. The spatial possibilities of objects with multiple holes and their possible interactions by passing through other objects may be interesting, but are not relevant for the problem. Thus, a first important simplification we make is to identify each *hole* with a *single-holed object*. The post hole seems to be an exception in this sense, as there is some difference between the ring being in the post down the hole or not. As we will see later, this will be handled by "partitioning" the post into two imaginary pieces: the hole itself (ph) and a connected long object (the post body p). Let's describe now our hole representation in more detail.

In this work we assume that each hole has two poles representing the two sides of the hole. These poles subdivide the space local to the hole into two parts, named the *hole subspaces*. As shown in Figure 2.

Figure 2 represent the hole poles and their relative subspaces. In this figure the hole is the shaded region, the poles and subspaces are represented by a '+' and a '-'

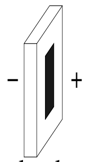


Fig. 2: Poles and subspaces of a hole.

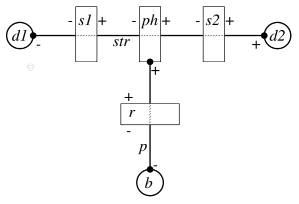
sign. Therefore, the holes function as local (bi-dimensional) reference frames, as we can localise objects that are just in “front” or just “behind” any particular hole (but not sidewise objects). It is worth pointing out that, in this work, the holes have no dimension, i.e. the space *inside* the hole is null. This simplifying assumption can be easily dropped by considering a third symbol (‘0’ for instance) representing the spatial region that lies in between the two hole poles. In this paper, however, for each hole h we represent its corresponding poles as h^- and h^+. Furthermore, if a is a hole pole, then $-a$ represents the opposite one, so that $-h^- = h^+$ and $-h^+ = h^-$.

Apart from the hole sort, there are two objects in the puzzle that also share some common features: the string (str) and the post (p). If we momentarily forget the hole in the post, both objects are “long” in the sense that, in principle, they could be simultaneously crossing several holes. Another common feature is that we can recognise two tips in each of these objects: for instance, we would probably talk about the “right end of the string,” or the “top end of the post,” to put a pair of examples. Thus, we will consider a second sort called *long object*, in which we include p and str. For each long object l we define two tips l^- and l^+, following an analogous notation to that used wrt hole poles. Each tip of a long object can be *linked* to something else. For instance, the string tips are connected to the disks, whereas the bottom of the post is linked to the post base. Although encoded in the same sort, the string and the post have an obvious difference: the flexibility inherent in the former, which is not a characteristic of the latter. As we shall see, in this work the string’s flexibility is reflected in the constraints imposed on the movements of the domain objects connected to it.

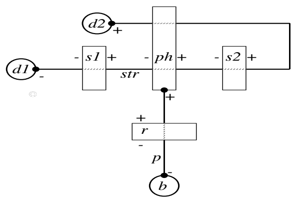
The remainder objects of the puzzle, that are the disks (d_1, d_2) and the post base (b), will just be classified as *regular* objects, without showing any particular feature, except perhaps that, due to their size, they can or cannot pass through each existing hole.

We illustrate the formalisation of the puzzle domain using diagrams. In these diagrams a box represents a *hole*, a circle a *regular object*, a thick line stands for a *long object* and a small black circle represents a link or connection. An example of this graphical representation is shown in Figure 3. Note how the post has been divided into a post hole (ph) linked to the top part of the post body (p). It may be reasonably objected that this division is not so natural, but it is also true that it would be possible to build an equivalent puzzle where, for instance, ph was a ring and p a second string.

Since each long object X can be crossing several holes, we will represent this using a list of crossings, called $chain(X)$. This list should collect the sequence of all hole crossings made by object X starting, for instance, from its negative tip and moving towards its positive one, whereas the same hole may occur several times in the list. Furthermore, the *direction* in which the string crosses the hole is also relevant. To see why, assume we represent the situation for Figure 4(a) simply as $chain(str) = [ph, s_1, ph, s_2, ph, ph]$. Then, we could not distinguish Figure 4(a) from Figure 4(b).

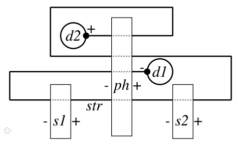


(a) S_0 (initial state)

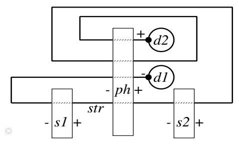


(b) S_1 (d_2 passes left)

Fig. 3: A pair of puzzle states.



(a)



(b)

Fig. 4: Two different states that could not be distinguished without crossing directions.

Figure 4(b) clearly represents a substantially different situation wrt Figure 4(a): the disk d_2 is now to the *right* (or positive side) of the post hole ph. Instead, a more suitable representation of Figure 4(a) would be: $chain(str) = [ph^-, s_1^+, ph^+, s_2^+, ph^-, ph^-]$.

Using the formalisation of the puzzle in terms of the list *chain*, presented in this section, we are able to define the basic actions on domain objects, as introduced below.

3 Acting on objects

In this section we define the two actions that implement the basic movements on the puzzle's objects: the action $pass\_o$ (passing an object through a hole) and the action $pass\_h$ (passing a hole through another hole).

3.1 Moving object endings: action $pass\_o$

The action $pass\_o(A, B)$ represents passing a long object tip A through some hole towards the hole pole B. For example, the execution $pass\_o(str^+, ph^-)$ in the initial state S_0 leads to S_1 (both depicted in Figure 3) and corresponds to moving the positive ending of str (currently linked to disk d_2) to the left of the post hole. It is clear that the execution of $pass(X^+, B)$ (resp. $pass(X^-, B)$) may equally mean that we are adding or removing the hole from $chain(X)$ depending on the context. For instance, the movement described above, $pass\_o(str^+, ph^-)$ should add ph^- to $chain(str)$ leading to the list $[s_1^+, ph^+, s_2^+, ph^-]$ in S_1, whereas performing $pass\_o(str^+, ph^+)$ in that state should return us to the initial situation S_0, removing ph^- from the list.

The possible effects of $pass\_o$ are depicted in Figure 5.

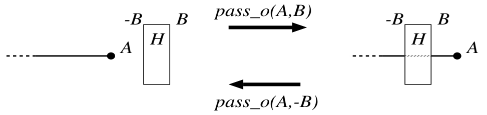


Fig. 5: Possible effects of $pass\_o$.

Looking from the right to the left execution of $pass\_o$, we can conclude that, when we are performing $pass\_o(X^+, B)$ on the $chain(X) = [\dots, -B]$, we must remove $-B$ from the tail of this chain. The analogous case would be $pass\_o(X^-, B)$ with $chain(X) = [-B, \dots]$ where we would remove $-B$ from the head of $chain(X)$. If none of the two previous cases occur, then $pass\_o(A, B)$ is actually *inserting* a new crossing in $chain(X)$. Thus, $pass\_o(X^+, B)$ adds crossing B in the tail of $chain(X)$ whereas $pass\_o(X^-, B)$ adds crossing $-B$ to the chain head.

3.2 Passing holes through holes: action $pass\_h$

The previous action is not enough for describing the solution of the problem, since it does not take into account the movement of passing an (object containing a single) hole through another hole. To understand why, let us assume that, given the initial situation depicted in Figure 3 we tried to move the ring up. This is equivalent to move the post hole ph down the ring, that is, to pass ph through r^-. So we would need an action such as $pass\_h(A, B)$ where A is now a hole and B a hole pole. Back to the example, we would execute the action $pass\_h(ph, r^-)$ on the initial situation leading to the resulting state depicted in Figure 6.

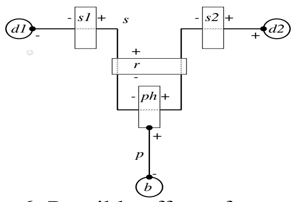
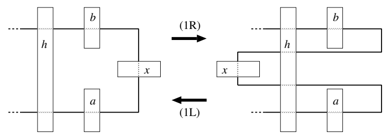


Fig. 6: Possible effect of $pass\_h$.

The most relevant effect of this action is that the string chain, which was previously unrelated to the ring hole, has gained two new crossings as an effect of $pass\_h$. In other words, the list: $chain(X) = [s_1^+, \underline{ph^+}, s_2^+]$ has to be updated to: $chain(X) = [s_1^+, \underline{r^-, ph^+, r^+}, s_2^+]$.

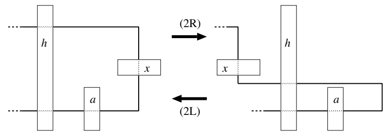
3.3 Possible movements

This section presents some possible movements that can be operated applying the two rules defined above. In the diagrams, we assume that the uppermost and the rightmost tips of long objects are positive.



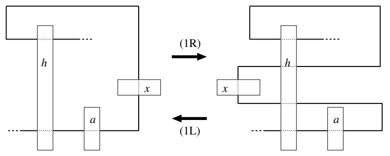
$[\dots, h^+, a^+, \underline{x^+}, b^-, \dots]$ $[\dots, h^+, a^+, \underline{h^-, x^+, h^+}, b^-, \dots]$

In Move (1R) we have that x is not contiguous to h in the chain. Therefore, by executing $pass\_h(x, h^-)$ we replace x^+ by the triple h^-, x^+, h^+. Movement (1L) starts in a state where x is preceded and succeeded by h in the chain but with alternate signs. A second possible movement would be:



$[\ldots, h^+, a^+, \underline{x^+, h^-}, \ldots]$ $\qquad$ $[\ldots, h^+, a^+, \underline{h^-, x^+}, \ldots]$

The problem of Move 2R is that it cannot be applied when x is followed by h^+ instead of h^-, as shown in the instance of movement 1, as follows:



$[\ldots, h^+, a^+, \underline{x^+}, h^+, \ldots]$ $\qquad$ $[\ldots, h^+, a^+, \underline{h^-, x^+, h^+}, h^+, \ldots]$

In general, for any hole e, assuming we want to execute $pass\_h(x, e)$ and x is crossed by some string, then for any string Y crossing x, and any occurrence of x in $chain(Y)$ we have the following list of possible movements:

(1R) $chain(Y) = [\ldots, a, \underline{x^z}, b, \ldots] \Longrightarrow [\ldots, a, \underline{e, x^z, -e}, b, \ldots]$ with $a, b \notin \{e, -e\}$ or $a = e, b = -e$.

(1L) $chain(Y) = [\ldots, \underline{-e, x^z, e}, \ldots] \Longrightarrow [\ldots, \underline{x^z}, \ldots]$

(2R) $chain(Y) = [\ldots, a, \underline{x^z, e}, \ldots] \Longrightarrow [\ldots, a, \underline{e, x^z}, \ldots]$ with $a \neq -e$

(2L) $chain(Y) = [\ldots, \underline{-e, x^z}, a, \ldots] \Longrightarrow [\ldots, \underline{x^z, -e}, a, \ldots]$ with $a \neq e\}$

Note that the above movements are complete, in the sense that if x occurs in $chain(Y)$ as follows $[\ldots, a, x^z, b, \ldots]$ both a and b could be equal to e, equal to $-e$ or none of the two. As a result, we would have $3 \times 3 = 9$ possibilities which, for reasons of space we do not depict here, but can be seen to be all covered by the movements above. The cases in which x is at head or tail end of the chain are also covered by assuming that the ends themselves are elements different from all the rest in the list.

Another important observation is that, while all the represented elements in each movement would be involved in the distinction of the movement type, only the underlined parts constitute the movement effect. This means, for instance, that in movement (2R), a is only used in the predecessor state, to establish that we have a (2R) movement and not a (1L), whereas in the successor state, it could be the case that a results moved to the left or even removed by the effect of another movement (remember that x may occur several times in the chain). An example of this *accumulated* movement would be, for instance, the execution of $pass\_h(x, h^+)$ on the list $[h^-, x^+, h^+, x^-, h^+]$ where we would perform (1L) on the first x and (2R) in the second leading to $[x^+, h^+, x^-]$.

With the representation developed above we can now formally express one solution to the Fisherman's puzzle and the sequence of states involved in its execution. Figure 7

state	$chain(p)$	$chain(str)$	next action(s)	movements
S_0	$[r^+]$	$[s_1^+, ph^+, s_2^+]$	$pass\_o(str^+, ph^-)$	
S_1	$[r^+]$	$[s_1^+, ph^+, s_2^+, ph^-]$	$pass\_o(p^+, r^-)$ $\&\ pass\_h(ph, r^-)$	(1R) $\times$ 2
S_2	$[\,]$	$[s_1^+, r^-, ph^+, r^+, s_2^+, r^-, ph^-, r^+]$	$pass\_h(s_2, r^-)$	(1L)
S_3	$[\,]$	$[s_1^+, r^-, ph^+, s_2^+, ph^-, r^+]$	$pass\_h(r, ph^+)$	(2R)+(2L)
S_4	$[\,]$	$[s_1^+, ph^+, r^-, s_2^+, r^+, ph^-]$	$pass\_h(s_2, r^+)$	(1L)
S_5	$[\,]$	$[s_1^+, ph^+, s_2^+, ph^-]$		

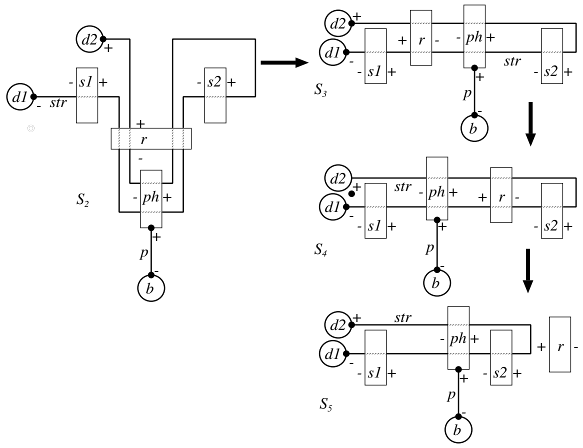


Fig. 7: A formal solution for the Fisherman's puzzle.

shows this solution step by step and depicts the corresponding spatial configurations of states S_2 through S_5 (S_2 and S_3 were already shown in Figure 3).

Clearly, S_5 is a solution, since the ring r is not passed through any long object. Note that the action performed in state S_1 is actually a combined one. This is because moving the ending p^+ to r^- implies passing also the post hole, as p^+ and ph are linked.

The next section discusses an implementation of the puzzle into an action language.

4 A simple Prolog implementation

As an actions domain, our abstraction of the Fisherman's folly is quite simple in the sense that most complex features of actions reasoning are not required for the problem. We deal with two actions, $pass\_o$ and $pass\_h$ whose execution causes a direct effect on the fluents $chain(X)$, for each long object X. Rather than providing a precondition per each action, we have found more convenient to specify general constraints in which the actions are not executable. We have used a Prolog predicate *nonexecutable(S,A)* to represent when an action *A* cannot be performed in a state *S*, including the rules:

$$\begin{array}{l} nonexecutable(\_, pass\_o([X, \_], [H, \_])) :- \\ \qquad\qquad cannot\_pass(X, H), !. \\ nonexecutable(S, pass\_o(P, [H, \_])) :- \\ \qquad\qquad member(linked\_to(P) = X, S), cannot\_pass(X, H), !. \\ nonexecutable(\_, pass\_h(X, [H, \_])) :- cannot\_pass(X, H), !. \end{array}$$

$nonexecutable(S, pass\_h(X, \_)) :-$
$\qquad member(linked\_to(\_) = X, S), !.$

The pairs *[X,Y]* are used to represent tips, so that for instance, *[p,+]* would stand for p^+. The fourth non-executability condition is used to force that, when an object tip is connected to a hole, the planner tries first to pass the object's tip and later the hole in a same transition. In this way we avoid irrelevant solutions where we can try to do it in the opposite ordering, obtaining exactly the same effects.

Of course, the main difficulty of this scenario from the standpoint of planning representation languages (STRIPS, ADL, PDDL) or even formalisms for action reasoning is the need for dealing with lists and pattern matching. In fact, this has motivated the choice of Prolog in order to build this preliminary prototype. Our implementation includes a Prolog predicate *process_chain(X,HP,L1,L2)* to describe the effect of performing $pass\_h(X, \mathit{HP})$ on the chain list *L1* leading to the list *L2*. An example showing the implementation of movement (1R) is shown below.

$process\_chain(X, HP, [A, [X, S], B|Cs], Ds) :-$
$\qquad opposite(HP, HP1), A \backslash = HP1, B \backslash = HP, !,$
$\qquad process\_chain(X, HP, [B|Cs], Ds0),$
$\qquad Ds = [A, HP, [X, S], HP1|Ds0].$

Note how the right neighbour of *[X,S]*, the crossed tip *B*, is used to keep processing the rest of the chain in the recursive call, and how the result of this recursive call *Ds0* may not contain *B* any more – it could be deleted by an accumulated movement (1L).

From the planning algorithm point of view, we have just implemented a blind search, relying on depth-first forward chaining with an iterative deepening strategy. Since the plan is really short, the Prolog program[3] just takes 10.30 seconds to find a first solution, despite of the inefficient planning strategy.

It is interesting to observe that the program actually finds several solutions in five steps. For instance, apart from the obvious symmetric solution where we begin working with d_1 instead of d_2 making $pass(s^-, ph^-)$, we also get a variant of the depicted solution in Figure 7 where to reach state S_3 we execute instead the sequence $pass\_o(str^+, ph^-)$, $pass\_h(s2, r^-)$ and $pass\_o(p^+, r^-)$ & $pass\_h(ph, r^-)$. This solution is not valid for the original puzzle since, although both the sphere and the post can pass through the ring, they cannot do so *simultaneously*. For immediate future work, we plan extending our representation so that the predicate *cannot_pass* describes when a group of objects cannot be altogether simultaneously passing through a given hole.

We have also made some small variations of the original puzzle by changing some of the premises. For instance, by allowing spheres to pass through the post hole we directly get a shorter solution: $pass\_o(str^+, ph^-)$, then $pass\_h(s2, ph^-)$, that gets the string-disks-spheres tandem out of the post and, finally, $pass\_o(p^+, r^-)$ & $pass\_h(ph, r^-)$ to get the ring free.

[3] We have used SWI-Prolog 5.2.11 interpreter running on Linux Mandrake 10 on a Pentium IV 1.5 GHz with a RAM of 512 MB.

5 Concluding remarks

In this work we investigated a challenging problem for spatial reasoning and common sense knowledge formalisation, namely, the problem of reasoning about spatial domains that contain non-trivial objects such as holes and strings. We propose a representation whereby holes identify sub-spaces in which objects could be located. The string in this paper is formalised as a long object that restricts the movement of the objects linked to it. In fact, the flexibility of this object is not fully explored in this work, as we abstracted away the possibility of tying knots. This issue shall be investigated in future work.

The formalisation of spatial knowledge is not the only challenge in the domain assumed in this work. Solving the puzzle also involves interesting issues that are beyond search (or planning) through a state space. For instance, when changing the spatial configuration of the puzzle, a person has a *selective observation* of domain objects, whereby only a portion of the space is observed. Actions are, thus, only applied within this limited view of the scene. A second issue is the minimisation of the *spatial configuration complexity*; the string allows for the application of a sequence of actions rolling the string around the post, or through it, many times. Minimisation of the puzzle's configuration complexity could be used as an heuristic in an automated problem solver. However, how this complexity should be measured is still an open problem. Finally, when trying to solve the puzzle for the first time, a human agent may not know every constraint or every possible movement of the domain objects. Searching for an automated way in which such spatial knowledge can be assimilated is also a very challenging issue for further investigations.

References

1. E. Clementini, P. di Felice, and D. Hernandez. Qualitative representation of positional information. *Artificial Intelligence*, 1997.
2. A. G. Cohn and S. M. Hazarika. Qualitative spatial representation and reasoning: An overview. *Fundamenta Informaticae*, 46(1-2):1–29, 2001.
3. D. Hernandez, E. Clementini, and P. di Felice. Qualitative distances. In W. Kuhn and A. Frank, editors, *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1995.
4. R. C. Meathrel and A. P. Galton. A hierarchy of boundary-based shape descriptors. In *Proc. of IJCAI*, pages 1359 – 1369, 2001.
5. T. Morita, J. Takamatsu, K. Ogawara, H. Kimura, and K. Ikeuchi. Knot planning from observation. In *Proc. of the IEEE ICRA*, pages 3887 – 3892, 2003.
6. D. Randell, Z. Cui, and A. Cohn. A spatial logic based on regions and connection. In *Proc. of KR*, pages 165 –176, 1992.
7. K. Reidemeister. *Knot Theory*. BCS Associates, 1983.
8. P. Santos and M. Shanahan. Hypothesising object relations from image transitions. In *Proc. of ECAI-02)*, pages 292 – 296. 2002.
9. M. Shanahan. Default reasoning about spatial occupancy. *Artificial Intelligence*, 74(1):147 – 163, 1995.
10. O. Stock, editor. *Spatial and Temporal Reasoning*. Kluwer Academic, 1997.
11. A. C. Varzi. Reasoning about space: The hole story. *Logic and Logical Philosophy*, 4:3 – 39, 1996.