On Qualitative Route Descriptions

Representation, Agent Models, and Computational Complexity

Matthias Westphal (westpham@informatik.uni-freiburg.de),
Stefan Wölfl (woelfl@informatik.uni-freiburg.de) and
Bernhard Nebel (nebel@informatik.uni-freiburg.de)
Department of Computer Science
University of Freiburg, Germany

Jochen Renz (jochen.renz@anu.edu.au)
Research School of Computer Science
The Australian National University
Canberra, Australia

Abstract. The generation of route descriptions is a fundamental task of navigation systems. A particular problem in this context is to identify routes that can easily be described and processed by users. In this work, we present a framework for representing route networks with the qualitative information necessary to evaluate and optimize route descriptions with regard to ambiguities in them. We identify different agent models that differ in how agents are assumed to process route descriptions while navigating through route networks and discuss which agent models can be translated into PDL programs. Further, we analyze the computational complexity of matching route descriptions and paths in route networks in dependency of the agent model. Finally, we empirically evaluate the influence of the agent model on the optimization and the processing of route instructions.[1]

Keywords: Qualitative reasoning, spatial reasoning, route planning, route descriptions

1. Introduction

One of the most widely used applications of spatial data in everyday life is navigation in street networks. Since GPS receivers have been integrated into many devices, spatial data about routes or spatial traces of events have become widely available. At the same time, there is a growing interest in such data for applications ranging from map services (e.g., OpenStreetMap[1]) to query answering on GPS traces of biking or hiking tours (e.g., EveryTrail[2]). While many of these appli-

[1] This article is a significantly extended version of the following conference paper: Matthias Westphal, Stefan Wölfl, Bernhard Nebel, and Jochen Renz. On Qualitative Route Descriptions: Representation and Computational Complexity. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pp. 1120-1125. AAAI Press 2011.

[1] http://www.openstreetmap.org/

[2] http://www.everytrail.com/

cations require metric data to be present at runtime, the evaluation and generation of route descriptions can be performed on a qualitative representation of preprocessed metric data.

In the literature several approaches to representing spatial route information have been discussed. For example, Kuipers' Spatial Semantic Hierarchy (Kuipers, 2000) provides a model for the representation of spatial information that integrates qualitative and quantitative levels of description where the qualitative representation is in particular useful for processing large-scale maps. Based on ideas of the Spatial Semantic Hierarchy, Krieg-Brückner et al. (2004) introduce a concept of route graph that is tailored to the semantic representation of route instructions in the context of human-robot interaction.

Contrary to such integrated representations of spatial route information the literature on route descriptions is often based on rather simple graph-theoretical notions. The focus is usually on criteria for evaluating the quality of route descriptions, such as route length, simplicity, description length, and reliability.

The starting point in this research direction is the work by Mark (1986). He proposes to utilize A^*-search in the generation of route descriptions with a cost function that balances metric path length and path complexity. The complexity of a path is estimated by a sum of penalty values assigned to the intersections on a path according to a "frame and slot" representation. Following this idea, Duckham and Kulik (2003) minimize the complexity of a route description by a shortest path algorithm, where the cost function accounts for the amount of information required to negotiate each street intersection. Several different (cognitive) cost functions can be used in this context.

In many situations primitive instructions in a route description can be ambiguous. While following a route description, humans often face situations in which several options are consistent with the description and it is unclear to them how to proceed. Thus, it is natural to consider cost functions that take into account such ambiguities (see, e.g., Haque et al., 2006). Further aspects discussed in the literature include the role of landmarks in route descriptions (e.g., Duckham et al., 2010) as well as strategies to generate compact route descriptions by spatial chunking (Richter and Duckham, 2008).

While most work on route descriptions is motivated by cognitive aspects of human way-finding, the focus of our work is different: when route descriptions are generated automatically or by humans, the description usually presumes some particular behavior of the agent following it. The basic idea of our work is to model behaviors as execution strategies within a simple graph-like representation of the route network, which takes into account ambiguities in route descriptions. We will consider several basic agent models to address the following questions: what is the role of the agent model for both the generation of a route description and the evaluation of its quality? Is it always easy to check whether a route instruction leads to the desired destination? Where do ambiguous route instructions lead to? How hard is it to produce short route descriptions? And, how successful can an agent be when it follows a route instruction that was generated for agents with a differing execution strategy?

The contribution of this article is twofold. Firstly, we discuss how the information contained in map-like data can be used to extract an agent-centric qualitative representation of the graph. More precisely, we introduce a concept of transition system that is tailored towards a compact representation of just as much map-like information as needed for evaluating and optimizing qualitative route descriptions. These transition systems are adapted from a more general notion of route graph presented in (Renz and Wölfl, 2010). Secondly, we focus on ambiguities in route descriptions which leave several alternatives to the agents processing them. To this end, we discuss properties of agents processing route descriptions that influence the potential paths taken in a route graph. We distinguish different types of agents, describe how they can be formalized using propositional dynamic logic, and analyze the computational complexity for generating and for processing route descriptions with respect to these agent types. Finally, we present the result of an empirical study that demonstrates practical differences between the agent types.

The outline of the article is as follows: in the following section we discuss how a qualitative representation of an agent's decisions can be extracted from the layout of a route graph. Then, in section 3 we present different agent models for interpreting route descriptions. Based on that, we discuss in section 4 how these agent models can be translated into propositional dynamic logic. In section 5 the computational complexity of evaluating and optimizing route descriptions is discussed in dependency of the agent model. We report on empirical results on the effects of the agent model in section 6. Section 7 summarizes our results.

2. From Route Graphs to Route Descriptions

In this section we introduce a qualitative representation of route networks that takes into account the egocentric perspective of an agent traversing the network. This representation builds on input given by geodata, which we assume to be annotated directed graphs, as well as instruction primitives and their interpretation. Finally, we define route descriptions based on this representation.

2.1. Route Graphs

A reasonable assumption on available geodata is that it comes in the form of directed graphs annotated with metadata, such as GPS coordinates for each vertex, arcs annotated with the type of road they represent, prohibited turns at street intersections according to traffic regulations, etc. This is for example the case when data are retrieved from OpenStreetMap or when automatically recorded GPS traces are transformed into more refined graph structures (see, e.g., Edelkamp and Schrödl, 2003). The qualitative representation we introduce is based on such annotated directed graphs, and we assume that all the information relevant for our purposes, such as the metric information about the layout of the network and the information about allowed paths, is provided in the graph.

We make these annotated graphs explicit in the next definition and consider a two-dimensional setting by assuming that each street network is embedded into the Euclidean plane. Note that this does not require the graphs to be planar or strongly connected.

Definition 2.1. A *route graph* is given by an ordered quadruple $\mathcal{N} = \left\langle V, A, F, \cdot^{\mathcal{N}} \right\rangle$, where $\langle V, A\rangle$ is a directed graph, F is a set of *forbidden paths* in $\langle V, A\rangle$, and $\cdot^{\mathcal{N}} \colon V \to \mathbb{R}^2$ is a function that assigns to each vertex v a point $v^{\mathcal{N}}$ in the Euclidean plane. We say that a path π in $\langle V, A\rangle$ is *admissible* in $\mathcal{N}$ if none of its subpaths is forbidden, i.e., included in the set F.

The role of the forbidden paths in this definition is to model those parts of a street network that are irrelevant for the generation and the processing of route descriptions in a given application context. For example, for car navigation only those parts are relevant that are accessible by cars due to width or height limitations or traffic regulations. Forbidden turn actions are a further example: for instance, a path of the form uvu can be used to specify that at v no U-turn is permitted. Although the definition mentions forbidden paths, we are of course primarily interested in admissible paths. (Notice that paths that are not admissible are only required to have at least one forbidden subpath, i.e., they need not be explicitly listed in the forbidden path set F.)

Figure 1 gives an example of a route graph without metadata. Note that the representation of the street network in the route graph is dense enough in order to describe the physical shape of routes. This is important for the interpretation of instruction primitives, but apart from

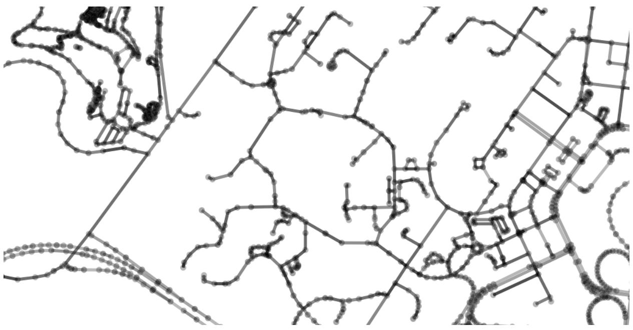


Figure 1. An example of a route graph (without metadata) given by geodata obtained from OpenStreetMap; the graph represents a very small part of Canberra, AUS.

this the fine-grained level of quantitative information is not required for queries on route descriptions. Before we turn to instruction primitives, we consider selecting specific vertices that are important for navigating a network. In our context important vertices in the route graph are typically those vertices which an agent might want to reach or those at which a decision has to be made by the agent while moving within the network. Given such a set of *points of interest* $P \subseteq V$, we consider specific paths between them — so-called P-paths.

Definition 2.2 (Diestel, 2010)**.** Let $\langle V, A\rangle$ be a directed graph. For a set of vertices $P \subseteq V$, a *P-path* is a path $\pi = v_0 \dots v_n$ in $\langle V, A\rangle$ with $v_0, v_n \in P$ and $v_i \notin P$ for each $0 < i < n$.

2.2. Turn Instructions

Route descriptions specify routes in a street network by describing step by step the actions to be performed in order to reach from some initial position in the network some desired goal position. Therefore, route descriptions use a variety of expressions to specify turn actions. In this work we are primarily interested in expressions that fit the egocentric perspective of an agent moving within the network.

Typical examples are instructions to negotiate intersections. Such instructions can be given by using a rather restricted vocabulary of *turn expressions* such as “go straight”, “turn right”, etc. In the situations depicted in Figure 2, for instance, we may describe the turn actions at the intersection by “go straight” and “turn sharp left”, respectively.

The interpretation of egocentric turn expressions is based on how an agent perceives the possible options at an intersection, which means that an agent has to map a turn expression in the route description to one of the possible outgoing routes at the intersection. In contrast

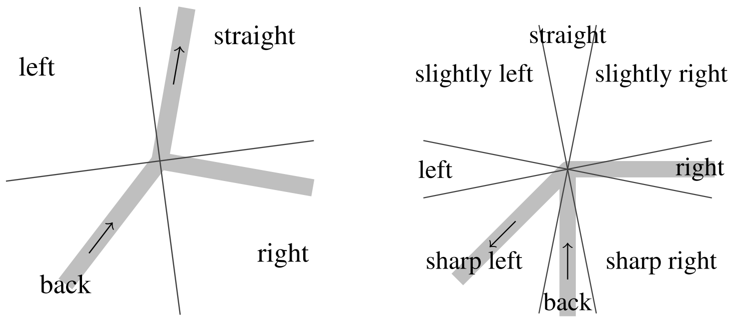


Figure 2. Two different qualitative schemata used to define turn actions at a street intersection.

to the simple examples depicted in Figure 2, in more complex situations this task can become difficult and may actually rely on different situation-dependent factors. Examples of such factors include the visual saliency of features that demarcate outgoing routes as well as the concrete shapes of the routes. For example, in cases where a route branches off in one direction, but immediately afterwards leads into a completely different one, it is crucial whether the local shape of the route is perceptually dominated by more distant features, or not.

Natural languages are rich in expressions describing turn actions. For example, such descriptions may refer to the shape of an outgoing path ("the one that slightly turns right"), relate orientation to visible landmarks ("right at the supermarket"), or include numerical information ("third route on the left"). In what follows we will merely assume that we can operate with a fixed set of turn expressions, where each expression has a fixed semantical interpretation.

Definition 2.3. Let $\mathcal{N} = \left\langle V, A, F, \cdot^{\mathcal{N}} \right\rangle$ be a route graph, $P \subseteq V$ a selection of points of interest. An *orientation* (of an agent) at vertex $v \in P$ is an admissible incoming arc uv in A (thus representing the orientation of the vector $u^{\mathcal{N}} v^{\mathcal{N}}$). A *turn instruction* on $\mathcal{N}$ is a function that assigns to each orientation uv a (possibly empty) set of admissible P-paths that start in v. An *instruction set* on $\mathcal{N}$ is a finite set $\mathcal{L}$ of turn instructions on $\mathcal{N}$ such that the following condition is satisfied:

($*$) for each orientation uv and each admissible P-path π starting in v, there exists some turn instruction $d \in \mathcal{L}$ with $\pi \in d(uv)$.

We will often specify an instruction set simply by listing a vocabulary of turn expressions $\mathcal{L} = \{d_1, \ldots, d_n\}$, when the interpretation of turn expressions (as turn instructions in the sense of the definition) is understood from the context. Moreover, where needed, we will assume that $\mathcal{L}$ contains a distinguished turn instruction "straight".

Given a fixed vocabulary of turn expressions $\mathcal{L}$, we can label all outgoing paths for any given orientation. This allows us to model the egocentric aspect in navigation tasks and to qualitatively represent the different ways in which an agent might proceed at any given vertex. It is always possible to simply assume that $\mathcal{L}$ contains turn expressions that uniquely identify each outgoing path for any orientation, e.g., by reference to street names or addresses. In this way, a description provider could always generate unambiguous route descriptions. However, complex turn expressions may be hard to understand and memorize. Furthermore, complex turn expressions may require that agents following a route description are able to perceive all the (visual) features mentioned in the expression. For example, how should an agent process an instruction such as “turn right at Schloßweg”, when no signage indicates that she has reached an intersection of “Schloßweg”. Finally, it is rather unrealistic to assume that humans do always interpret turn expressions with the interpretation intended by the description provider. Thus, ambiguities seem to be unavoidable both in the generation of route descriptions as well as in their execution.

As running examples we consider two simple instruction sets derived from the qualitative relations between the orientation of an agent approaching an intersection and its outgoing paths. These spatial relations specify directional information relative to the heading of the agent (when moving) or his line of sight. We consider so-called *sector models*, which serve as templates for partitioning the Euclidean plane into different non-overlapping sectors around the origin. The idea is that the agent, positioned at the origin and facing true North, partitions its environment into circular sectors (such as “front sector”, “left sector”, etc.).

Good sector models should be close to how humans discretize space. Klippel and Montello (2007) present a sector model that is based on an empirical study how humans cluster directional relations. This model does not exhibit equally sized sectors, nor is it front-back symmetric or left-right symmetric. In our empirical evaluation we will use simplified approximations of this model.

Sector models can be used to define instruction sets. Consider an orientation uv (of an agent approaching the intersection at $v^{\mathcal{N}}$). The orientation uniquely defines the half-line from $u^{\mathcal{N}}$ through $v^{\mathcal{N}}$ (thus representing the agent’s heading). We translate the origin to the point $v^{\mathcal{N}}$ and then rotate the sector model around $v^{\mathcal{N}}$ such that the half-line corresponds to the direction true North. The sectors can now be labeled with spatial terms, e.g., “straight”, “back”, “right”, etc. To define turn instructions in the sense of Definition 2.3, we map each path starting in

v to excactly one of the sectors. A simple approach for that is to select from each such path a vertex within some pre-fixed distance from $v^{\mathcal{N}}$ and assign to the path the sector of the selected vertex.

Sector models can easily be obtained from the family of $\mathcal{STAR}^r$ calculi (Renz and Mitra, 2004). These representation formalisms allow for defining directional relations at arbitrary granularity levels. In the following discussions and in the evaluation presented in section 6 we will apply these $\mathcal{STAR}^r$ calculi, namely egocentric variants of $\mathcal{STAR}_2^r$ with four equally sized sectors (left in Figure 2) and $\mathcal{STAR}_4^r$ with eight sectors (right in Figure 2). These sector models (with appropriate sector labelings and applied to the orientation of the agent) are depicted in Figure 2.

2.3. A Representation of Route Networks

The key idea for modeling the egocentric aspect in navigation is to explicitly represent the different orientations by which an agent might approach an intersection while traversing the route network. For each orientation the agent may have different options how to continue in the network by choosing one of the admissible outgoing paths, which will lead the agent to its goal or to the next situation that requires some decision. This idea is made explicit in the notion of transition system, in which turn instructions reappear as transition labels.

Definition 2.4. A *labeled transition system* with label symbols in $\mathcal{L}$ is a tuple $\mathcal{S} = \langle S, \rightarrow \rangle$ where S is non-empty set of *states* and $\rightarrow = \{\xrightarrow{d}\}_{d \in \mathcal{L}}$ is a family of binary relations on S.

Given a state $s \in S$ and a turn label $d \in \mathcal{L}$, let $d[s]$ denote the set of states s' such that $s \xrightarrow{d} s'$. A *path* in $\mathcal{S}$ is a sequence of states $\sigma = s_0 \dots s_n$ in S such that $s_{i+1} \in d_i[s_i]$ for some turn label $d_i \in \mathcal{L}$ for each $0 \leq i \leq n-1$. We write $\|\rightarrow\| := \max_{s \in S} \sum_{d \in \mathcal{L}} |d[s]|$ for the maximal number of transitions in any state in S.

We show that any given route graph and instruction set with a reasonably selected set of vertices induces a labeled transition system. Let $\mathcal{N} = \left\langle V, A, F, \cdot^{\mathcal{N}} \right\rangle$ be a route graph and $P \subseteq V$ be a non-empty set of vertices such that:

(a) for each orientation uv in $\mathcal{N}$ with at least two distinct admissible paths $uvx\dots$, $uvy\dots$ it holds $v \in P$, and

(b) for each orientation uv in $\mathcal{N}$ where $v \in P$ it holds that each outgoing arc vx where uvx is admissible is contained in some admissible P-path.

The first condition ensures that all situations in which an agent must make a decision are included, and the second one that all choices will lead to an orientation at some vertex in P. The set of states S of the induced transition system is the set of all orientations at vertices in P. This allows us to define the transition relations $\xrightarrow{d}$ on S as follows: $uv \xrightarrow{d} xy$ if and only if there exists a P-path $\pi = vw \dots xy$ such that $u\pi$ is an admissible path and π is in the interpretation of d at uv. Notice that the path π is not uniquely determined: it could be the case that there is another P-path $\pi' = vw' \dots xy$ that provides a transition from uv to xy and is included in the interpretation of the same turn instruction as π at state uv. But since on both paths no intermediary vertices are in P we can abstract from the differences between π and π'. Effectively, π and π' may be considered parallel lanes from state uv to state xy, leaving no choice to an agent, and hence the two paths can be considered identical.

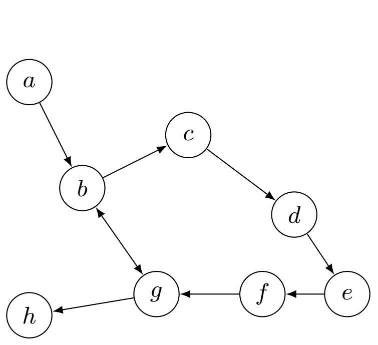
Any path in $\mathcal{S}$ can be represented by some alternating sequence of states and labels $s_0\, d_1\, s_1 \dots d_n\, s_n$ where $s_i \in d_i[s_{i-1}]$ for each $1 \leq i \leq n$. Thus, we can say that the path is described by the sequence of labels $d_1 \dots d_n$. This motivates the following notion of a route description.

Definition 2.5. Let $\mathcal{L}$ be an instruction set. A *route description* (or *route instruction*) is a sequence $\delta = d_1 \dots d_n$ with $d_i \in \mathcal{L}$ for each $1 \leq i \leq n$.

To summarize, given a route graph and a reasonably selected set of vertices in that graph, there exists a labeled transition system that reflects the possible choices of an agent traversing the network. In what follows, the transition system defined above is referred to as the *transition system induced by $\mathcal{N}$, $\mathcal{L}$, and P*. Figure 3 provides a simple example that illustrates the concept of an induced transition system.

2.4. Remarks

Instead of labeled transition systems as introduced here one can also consider directed multigraphs in which each arc is annotated with a decision. From the discussion above it should be obvious that these multigraphs are formally equivalent to so-called decision frames as introduced by Westphal et al. (2011). Technically, transition systems in the sense of Definition 2.4 are decision frames modulo an equivalence relation of instruction equivalence on arc-inducing paths. Moreover, all these notions provide an egocentric variant of the qualitative route graphs presented by Renz and Wölfl (2010). The multigraph approach may be seen as a simple generalization of the notion of *line graph* considered in graph theory (Diestel, 2010), which can be found in previous



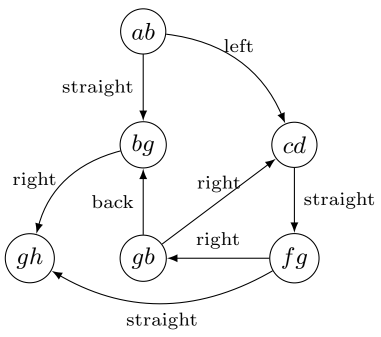


Figure 3. A small route graph and its induced transition system with selected vertices $P = \{b, d, g, h\}$. U-turns are permitted at b but not at g. As labels we use relations from $\mathcal{STAR}_2^r$.

work on route descriptions, e.g., the concept of evaluation mapping discussed by Duckham and Kulik (2003).

Transition systems are flexible enough for our purposes and can be easily extended. For example it may be desirable to consider probabilistic extensions that more closely model the human understanding of instruction primitives. Also, one could introduce transitions that can be made without any given label (so-called ϵ-transitions) to model states where one could argue no conscious decision is made by an agent e.g., if for a given orientation there is only one admissible outgoing path.

Given some route description δ, the interesting problems are whether the instruction can be realized in the transition system and whether the instruction leads from some initial location to the desired goal destination. A crucial problem here is that route instructions can be interpreted differently by agents. To that aim, we will discuss *agent models* in the following section.

3. Agent Models

In many approaches the quality of a route description is evaluated in terms of a cost function that aggregates the difficulty of following its primitive turn instructions at intersections. The difficulty of instruction primitives in turn is evaluated dependent on spatial features of the location where a decision has to be made, but is otherwise independent of the overall graph. Obviously this has benefits for the computational cost of generating and evaluating descriptions. For such measures an optimal route can be found by a shortest path algorithm with appropriate cost function. Haque et al. (2006) introduce a further criterion, namely the reliability of a route description, which approximates the expected success of an agent following it. In addition to these previous works, however, we want to stress the role of the underlying agent model that is used when route descriptions are generated or evaluated. We argue that it is not only route- and route description-related factors that influence the usefulness of a route instruction, but also the assumptions about the agent that follows the instruction. As we will see, the agent model is in fact crucial for both the generation of route descriptions and the success of the agent following it. The problem is quite natural from a practical point of view. When you are asked by a person to give a route description to a specific place, your answer will usually depend on your assumptions on what the person might know and how she (presumably) behaves in navigation situations, but also on what you think might be problematic situations and how instruction primitives could be interpreted by her.

In what follows we discuss a short list of rather simplistic agent models, each of which makes a distinguished execution model for route instructions explicit. From first glance, it is clear that there is wide variety of agent models that appear reasonable: examples include agents that use landmark information, agents that learn the route network they traverse, agents with bounded memory being able to execute descriptions of a certain length only, etc. For the purposes of this article, however, we will restrict consideration to agent models that share the following general assumptions: (i) the agent can follow a description regardless of its length, i.e., it does not forget or confuse parts of the instruction; (ii) the agent's interpretation of turn expressions coincides with their intended meaning; in case of ambiguities the agent does not prefer any of the options associated with the same turn instruction; (iii) the environment is unknown to the agent and hence wrong turns cannot be recognized; (iv) the agent processes a route description step-by-step from the beginning, that is, the agent tries to follow the first unprocessed label in the description (as long as there is one left) at the current state in the transition system. If no instruction is left, the agent halts.

In order to model different types of simple agents, we consider the following features. We say that an agent (a) *strictly* processes a route description if it halts at the first state in which the next unprocessed instruction is not executable (and thus skips the rest of the instruction); (b) processes a route description with *default straight actions* if in case the current instruction cannot be executed, it tries to continue on some path in straight direction, deferring the execution of the current instruction (if no path continues straight, the agent halts and skips the rest of the instruction); (c) *recognizes* the goal if it halts once a desired destination has been visited; (d) *learns* the visited states if it halts once a state is revisited (and thus skips the rest of the instruction); in a sense the agent performs loop checking.

Condition (b) expresses that a turn instruction is interpreted as "turn at the next opportunity", (c) expresses that an agent has some knowledge about the destination that allows it to stop processing a route description in goal states, and (d) expresses that an agent accumulates some knowledge about the traversed network (book keeping of visited states) such that it will assume to be lost if it arrives at some already visited place with the same orientation.

Although the features we consider here are simplifying, the assumptions they reflect are quite natural and fundamental. The strict processing model is the most basic one one can think of. From there, the extension to default actions is plausible if an agent simply assumes that the description provider forgot about intermediary wrong turn options and meant "at the next opportunity". Goal recognition is quite natural if the desired destination is already known to the agent. Finally, learning is reasonable because it is natural for an agent to assume that a mistake has been made if following the route instruction has lead to traversing a cycle. In the literature similar assumptions have been made. Haque et al. (2006) evaluated their generation algorithm based on a simulated navigator that makes assumptions on the execution and interpretation of descriptions that are effectively the same as in the strict processing model without any of the additional features we consider.

More formally, these agent models can be defined as follows.

Definition 3.1. Let $\mathcal{S}$ be a labeled transition system with label symbols in $\mathcal{L}$.

(a) A *configuration* in $\mathcal{S}$ is a sequence $\sigma\delta$ consisting of a path $\sigma = s_0 \dots s_i$ in $\mathcal{S}$ and a route instruction $\delta = d_j \dots d_m$ in $\mathcal{L}$ (here we also allow for the empty instruction ϵ).

(b) An *execution relation* on $\mathcal{S}$ is any binary relation $\vdash$ on the set of configurations of $\mathcal{S}$.

An agent model maps to each labeled transition system $\mathcal{S}$ an execution relation on S. Thus it is sufficient to define the agent models mentioned above in terms of an execution relation. The execution relation of the *strict agent model* (referred to by a^{s}) is defined by:

$$s_0 \dots s_i \, d_j \dots d_m \vdash^{\mathrm{s}} s_0 \dots s_i s_{i+1} \, d_{j+1} \dots d_m$$

if and only if in $\mathcal{S}$ it holds $s_i \xrightarrow{d_j} s_{i+1}$. For the *default straight agent model* (symb., a^{d}), we define the execution relation as follows:

$$s_0 \ldots s_i\, d_j \ldots d_m \vdash^{\mathrm{d}}$$
$$\begin{cases} s_0 \ldots s_i s\, d_{j+1} \ldots d_m & \text{if } s_i \text{ has } d_j\text{-successors and } s_i \xrightarrow{d_j} s \\ s_0 \ldots s_i s\, d_j \ldots d_m & \text{if } s_i \text{ has no } d_j\text{-, but ``straight''-successors, and } s_i \xrightarrow{\text{straight}} s \end{cases}$$

Both these relations can be further qualified if we add one or both of the following conditions: for *agents with goal recognition* (symb., a^{g}) we require that $s_i \notin S_\star$ where $S_\star$ is a fixed set of goal states in the transition system and for *agents with learning* (symb., a^{l}) we add the condition $s_i \notin \{s_0, \ldots, s_{i-1}\}$.

Thus, we obtain in total eight different execution relations (and accordingly as many different agent models) $\vdash^{\mathrm{s}}$, $\vdash^{\mathrm{sg}}$, $\vdash^{\mathrm{sl}}$, $\vdash^{\mathrm{sgl}}$, $\vdash^{\mathrm{d}}$, $\vdash^{\mathrm{dg}}$, $\vdash^{\mathrm{dl}}$, and $\vdash^{\mathrm{dgl}}$. We write $a^\dagger$ for the agent model with execution relation $\vdash^\dagger$ ($\dagger \in \{\mathrm{s}, \mathrm{sg}, \ldots, \mathrm{dgl}\}$). In what follows, let $\vdash$ be any of these execution relations, and $\vdash^\star$ its reflexive-transitive closure.

Definition 3.2. Let $\mathcal{S}$ be a labeled transition system, $\delta = d_1 \ldots d_m$ a route description, s_0 an initial state, and $S_\star$ a set of goal states in $\mathcal{S}$. An *execution trace* of δ is a path $s_0 \ldots s_i$ in $\mathcal{S}$ such that $s_0\, d_1 \ldots d_m \vdash^\star s_0 \ldots s_i\, d_j \ldots d_m$. In this situation s_i is called a *halting state* if there exists no configuration $\sigma'\delta'$ such that $s_0 \ldots s_i\, d_j \ldots d_m \vdash \sigma'\delta'$.

Let $\mathrm{tr}^\dagger(s_0, \delta)$ denote the set of execution traces with execution relation $\vdash^\dagger$ starting in s_0 given the route description δ. The *halting set* of a route description δ in s_0, $\mathrm{halt}^\dagger(s_0, \delta)$, is the set of halting states, for an agent with model $a^\dagger$ starting in s_0 and executing δ. If the execution relation is clear from the context, we drop the superscript.

The following lemma is immediately clear from the definition of the execution relations.

Lemma 3.3. *Let $\mathcal{S} = \langle S, \rightarrow \rangle$ be a labeled transition system. For a given (finite) path σ in $\mathcal{S}$, route description δ, and initial state s_0, the decision problem whether σ is contained in $tr(s_0, \delta)$ can be decided in time $\mathcal{O}(|\sigma| \cdot \|{\rightarrow}\|)$ for each of the agent models a^s, a^{sg}, a^{sl}, a^{sgl}, a^d, a^{dg}, a^{dl}, and a^{dgl}.* □

Notice that in the case of agent models with learning each state in the transition system can be visited at most once, i.e., the bound can be trivially tightened to $\mathcal{O}(\min(|\sigma|, |\mathcal{S}|) \cdot \|{\rightarrow}\|)$.

4. Representing Agent Models in PDL

Can we relate the agent models presented in the previous section to logics that allow for expressing dynamic behavior? If so, we might be able to borrow results about complexity and algorithms of reasoning in these logics. Obvious candidates of such logics include propositional dynamic logic (PDL; Pratt, 1980, Fischer and Ladner, 1979) and the modal μ-calculus (see, e.g., Bradfield and Stirling, 2006).

In what follows we will confine ourselves to PDL which is a modal logic that allows (in its basic setting) for representing the execution traces of so-called *regular programs* by means of modal operators. On the syntactic level it employs two sets of symbols: namely atomic propositional formulae to express state properties and atomic program terms to describe one-step transitions between states due to some program execution. Complex formulae are then formed according to the following rules: First the set of formulae is closed under Boolean connectives. Moreover, if α is any program term and φ is a formula, then $\langle\alpha\rangle\,\varphi$ and $[\alpha]\,\varphi$ are formulae, expressing that at least one (each, resp.) terminating execution of α in the present state results in a state in which φ is true. Moreover, complex program terms can be formed from atomic program terms by applying the following constructors: $(\alpha;\beta)$ (do α followed by β), $(\alpha\cup\beta)$ (do α or β, non-deterministically), $\alpha*$ (repeat α for a finite, but non-deterministic number of times), $\varphi?$ (proceed if φ is true, else fail). In PDL one can express control flows of regular programs such as "**if** φ **is true, do** α**, else do** β" and "**while** φ **is true, do** α": the first control can be expressed by $((\varphi?;\alpha)\cup(\neg\varphi?;\beta))$, the second as $((\varphi?;\alpha)*;\neg\varphi?)$. As further syntactic sugar, one can e.g. introduce shortcuts for programs such as **skip** (defined as $\top?$) or **fail** (defined as $\bot?$).

Semantically, PDL formulae are interpreted over Kripke structures, i.e., labeled transition system $\mathcal{S}$ augmented with a truth assignment to each propositional variable in each state. Each atomic program term a is associated with a binary relation $a^{\mathcal{S}}$ on the set of states of the structure. The evaluation of complex program terms $|\alpha|^{\mathcal{S}}$ is then recursively defined as follows: $|\alpha;\beta|^{\mathcal{S}} = |\alpha|^{\mathcal{S}}\circ|\beta|^{\mathcal{S}}$ (where $\circ$ denotes the composition of binary relation), $|\alpha\cup\beta|^{\mathcal{S}} = |\alpha|^{\mathcal{S}}\cup|\beta|^{\mathcal{S}}$, $|\alpha*|^{\mathcal{S}} = (|\alpha|^{\mathcal{S}})^*$ (where R^* denotes the reflexive-transitive closure of relations R). Finally $|\varphi?|^{\mathcal{S}}$ is defined as the relation $\{\,(s,s)\colon \mathcal{S},s\models\varphi\,\}$. Here $\mathcal{S},s\models\varphi$ means that formula φ holds true in state s. As usual in modal logics, truth in a state s is defined in a recursive manner as well; the only interesting conditions here are:

$$\mathcal{S}, s \models \langle\alpha\rangle\,\varphi \iff \mathcal{S}, t \models \varphi,\ \text{for some } t \text{ with } (s,t) \in |\alpha|^{\mathcal{S}}$$
$$\mathcal{S}, s \models [\alpha]\,\varphi \iff \mathcal{S}, t \models \varphi,\ \text{for each } t \text{ with } (s,t) \in |\alpha|^{\mathcal{S}}$$

Obviously, in PDL we can express different agent models for route instructions. For the discussion we assume a fixed labeled transition system as defined in Definition 2.4, which we conceive of as a Kripke structure over which programs are interpreted. The atomic program terms will then be the turn instructions of the chosen instruction set (**str** will refer to the distinguished straight direction). In what follows we define translations τ of route instructions into PDL program terms.

A rather trivial agent, for example, is one that just "reads" the description, which may be expressed as: $\tau(\delta) := \mathbf{skip}$. Another agent is one that simply skips each non-executable turn instruction:

$$\begin{aligned} \tau(\epsilon) &:= \mathbf{skip} \\ \tau(d_1 \ldots d_n) &:= \big((d_1?; d_1) \cup \neg d_1?\big)\ ;\ \tau(d_2 \ldots d_n) \end{aligned}$$

– we use here $d?$ and $\neg d?$ as shortcuts for $\langle d\rangle\top\,?$ and $\neg\,\langle d\rangle\top\,?$, respectively.

Let us now discuss the agent models presented in the last section. For the strictly processing model a^{s} it suffices to define the translation as follows:

$$\begin{aligned} \tau^{\mathrm{s}}(\epsilon) &:= \mathbf{skip} \\ \tau^{\mathrm{s}}(d_1 \ldots d_n) &:= \big((d_1?; d_1; \tau^{\mathrm{s}}(d_2 \ldots d_n)\big) \cup \neg d_1? \end{aligned}$$

For the agent model with default straight actions (a^{d}) we define:

$$\begin{aligned} \tau^{\mathrm{d}}(\epsilon) &:= \mathbf{skip} \\ \tau^{\mathrm{d}}(d_1 \ldots d_n) &:= (\neg d_1?; \mathbf{str}?; \mathbf{str})*\,; \\ &\qquad\qquad \big((d_1?; d_1; \tau^{\mathrm{d}}(d_2 \ldots d_n)) \cup (\neg d_1?; \neg\mathbf{str}?)\big) \end{aligned}$$

This program expresses that the agent is performing straight actions as long as this is possible and there is no possibility to perform an action in direction d_1. After the agent could execute an action in direction d_1, the rest of the instruction is executed.

Agents with goal recognition can be expressed quite easily. For $S_\star$ a set of goal states in $\mathcal{S}$, consider a new propositional variable **goal** and extend the Kripke structure such that $\mathcal{S}, s \models \mathbf{goal}$ if and only if $s \in S_\star$.

We need to adapt the translations τ^{s} and τ^{d} only slightly:

$$\begin{aligned}\tau^{\mathrm{sg}}(d_1 \dots d_n) &:= \big((\neg\mathbf{goal}?; d_1?; d_1; \tau^{\mathrm{sg}}(d_2 \dots d_n)\big) \cup \\ &\quad (\neg\mathbf{goal}?; \neg d_1?) \cup \mathbf{goal}? \\ \tau^{\mathrm{dg}}(d_1 \dots d_n) &:= (\neg\mathbf{goal}?; \neg d_1?; \mathbf{str}?; \mathbf{str})* ;\ \big((\neg\mathbf{goal}?; d_1?; d_1; \\ &\quad \tau^{\mathrm{dg}}(d_2 \dots d_n)) \cup (\neg\mathbf{goal}?; \neg d_1?; \neg\mathbf{str}?) \cup \mathbf{goal}?\big)\end{aligned}$$

For agents with learning, there seems to be no way to express that an agent has reached a state in which she has already been before, even if we allow nominals for the states in the transition system. The issue is that one cannot mark in a dynamic way states as visited. An approximation of this agent model, however, can be defined if we consider an extension of PDL by the so-called **repeat**-construct (Streett, 1982).

Given a program term α, $\mathbf{repeat}(\alpha)$ is true in state s if there exists an infinite execution of α starting in s. We can use this construct in our context as follows: if the agent executing $\delta = d_1 \dots d_m$ revisits a state it is clear that some suffix $d_j \dots d_m$ of δ has an execution trace that could be repeated infinitely often. Before the agent executes the next action, it can check for each of these suffixes, whether $\mathbf{repeat}(\tau(d_j \dots d_m))$ holds true in the visited state. But the converse is not true: if one of these repeat-checks is true, one can in general not conclude that the state has been visited before.

Finally it is worth mentioning that the translations of the agent models a^{s}, a^{d}, a^{sg}, and a^{dg} are all polynomial in the length of the route instruction. As we will later see, it appears to be very unlikely that such a translation of the learning agent model to PDL is possible.

5. Queries on Transition Systems

We are interested in queries on labeled transition systems that relate route descriptions and paths in the system. To this end, we investigate the computational complexity of answering these queries in dependency of the agent model. When we talk about agent models, we always refer to one of the agent models presented at the end of section 3.

For polynomial-time reductions we construct a labeled transition system $\mathcal{S}$ directly, because for any such transition system we can construct a suitable route graph setup that induces $\mathcal{S}$. We make this idea explicit in the following proposition.

Proposition 5.1. *Let $\mathcal{S} = \langle S, \rightarrow \rangle$ be a labeled transition system with label symbols in $\mathcal{L}$. There exists a route graph $\mathcal{N}$, an interpretation of $\mathcal{L}$, and a selection of points of interest P that induce $\mathcal{S}$.*

Proof. We first construct a directed graph $\langle V, A\rangle$. For each state $s \in S$, we add two fresh vertices u_s, v_s to V and add each edge (u_s, v_s) to A. The points of interest are defined as $P = \{\, v_s \colon s \in \mathcal{S} \,\}$. For each possible transition $s \xrightarrow{d} s'$ we construct a path $v_s \dots u_{s'}$ with fresh intermediary vertices.

Each path for a transition $s \xrightarrow{d} s'$ needs to be included in the interpretation of d at $u_s v_s$ only. We simply define the interpretation of the labels $\mathcal{L}$ to match the intended label on outgoing paths. For turn labels $\mathcal{L}$ from a sector model, we can do better. We have argued in section 2.2 that the interpretation is based on the immediate shape of the route and hence paths with a fixed number of vertices suffice to describe a shape that matches the intended interpretation. (Notice that the route graph is not required to be planar.) In this case we simply assign positions to the vertices such that the interpretation given by the sector model matches the intended interpretation given by $\rightarrow$. The set of forbidden paths F remains empty and we have constructed the entire route graph $\mathcal{N} = \left\langle V, A, F, \cdot^{\mathcal{N}} \right\rangle$. □

In the following proofs, we often assume that there are four distinct labels in $\mathcal{L}$. This is a reasonable assumption as most works on route descriptions consider at least the four distinct labels as in the $\mathcal{STAR}_2^r$ schema ("left", "straight", "right", "back"). In particular for these label symbols it is easy to construct the route graphs mentioned in the proof of Proposition 5.1.

5.1. Relating Route Descriptions to Paths

It is clear that given a starting point in a labeled transition system and a route descriptions several paths in the system are possible under the given description. As a first computational problem, we consider the complexity of determining whether a given state is a possible halting state.

Theorem 5.2. *Let $\mathcal{S} = \langle S, \rightarrow\rangle$ be a labeled transition system with label symbols in $\mathcal{L}$, where $|\mathcal{L}| \geq 3$.*

The decision problem whether for given states $s_0, s_\star \in S$ and route description δ, $s_\star$ is contained in $\mathrm{halt}(s_0, \delta)$ *can be decided in time $\mathcal{O}(|S| \cdot |\delta| \cdot \|{\rightarrow}\|)$ for the agent models without learning (i.e., a^s, a^{sg}, a^d, a^{dg}). For each agent model with learning (a^{sl}, a^{sgl}, a^{dl}, a^{dgl}) the problem is* NP*-complete.*

Proof. For agents without learning, one can apply a dynamic programming approach based on a table with $|S| \cdot |\delta|$ entries. Dynamic programming builds the result of $s_\star \in \mathrm{halt}(s_0, \delta)$ based on the configurations $s_0 \dots s_i\, d_j \dots d_m$ that appear during recursive invocation where $d_j \dots d_m$ are suffixes of δ. Each such configuration corresponds to the subproblem $s_\star \in \mathrm{halt}(s_i, d_j \dots d_m)$ and the table stores the computed yes/no-results. Thus we avoid recomputing subproblems that can appear multiple times during the recursive evaluation. For agents without learning it suffices to store configurations of the form $s_i\, d_j \dots d_m$, because their execution relation does not depend on the prefix $s_0 \dots s_{i-1}$. Hence is easy to see that at most $|S| \cdot |\delta|$ distinct subproblems need to be solved.

Our agent models have a natural recursive definition and we can apply this idea in a top-down evaluation of $s_\star \in \mathrm{halt}(s_0, \delta)$ by starting with the configuration $s_0\delta$. Assume we have some strict agent. For any given configuration $s_i\, d_j \dots d_m$ that appears in the recursion it is either a halting state where we can immediately answer the subproblem or we have to recursively evaluate it. To recursively evaluate it, we pick a state s such that $s_0 \dots s_i\, d_j \dots d_m \vdash s_0 \dots s_i s\, d_{j+1} \dots d_m$. If the result $s_\star \in \mathrm{halt}(s, d_{j+1} \dots d_m)$ is already known we reuse it, otherwise we advance to the next state with $s_0 \dots s_i s\, d_{j+1} \dots d_m$. For each entry at most $\mathcal{O}(\|{\rightarrow}\|)$ next states in $\mathcal{S}$ need to be considered. This justified the overall bound $\mathcal{O}(|S| \cdot |\delta| \cdot \|{\rightarrow}\|)$.

For agents with default straight actions, possible deferred actions need to be evaluated as well. The only issue here are possible cycles as the prefix length does not decrease in case a default action is executed. However, the above idea can be easily extended to detect and avoid cycles. The bound estimation remains unaffected as each configuration is analyzed once.

For agents with learning, the proof is as follows: NP membership follows immediately from the fact that the length of the path is bounded by the number of states in $\mathcal{S}$ and Lemma 3.3: we can guess a path and check in polynomial time that it is consistent with the route description. NP-hardness can be obtained by a polynomial time reduction from the Hamiltonian path problem, which is known to be NP-complete even for planar graphs (Garey et al., 1976). We construct for a given undirected graph $G = \langle V, E \rangle$ a suitable transition system $\mathcal{S}$ as follows: we set $S := V \,\dot\cup\, \{s_0, s_\star\}$ and choose distinct turn labels d_1 and d_2 from $\mathcal{L}$ that are different from “straight” (in order to avoid interference with the default straight agent model). Define $\xrightarrow{d_1} := \{\, (s,t) \in S^2 \colon s,t \in V, \{s,t\} \in E \,\} \cup \{\, (s_0, t) \colon t \in V \,\}$ and $\xrightarrow{d_2} := \{\, (s, s_\star) \colon s \in V \,\}$. The transition relations for all other turn labels are defined to be empty. It is then easy to prove that G has a Hamiltonian path if and only if the route instruction $\delta = d_1 \dots d_1 d_2$ with as many

d_1-instructions as vertices in G has an execution trace that starting in s_0 halts in $s_\star$. If π is a path in G of length $|V| - 1$ that visits all vertices of G exactly once, π defines an execution trace in $\mathcal{S}$ (the first d_1-instruction is consumed by jumping from s_0 to the start vertex of π) that visits first all states distinct from $s_\star$, and finally by the d_2-instruction terminates in $s_\star$. Conversely, if σ is an execution trace of δ, the agent must visit all states s of $\mathcal{S}$. Thus, dropping the first and the last state of σ defines a Hamiltonian path in G. □

Theorem 5.2 makes a statement about the complexity of testing whether a given route instruction leads from a given initial state to a given goal state. By using the translations for the agent models $a = a^{\mathrm{s}}, a^{\mathrm{sg}}, a^{\mathrm{d}}, a^{\mathrm{dg}}$ presented in section 4, we can cast this problem as a model checking problem on the considered labeled transition graph $\mathcal{S}$. When **goal** is true only in state $s_\star$, then we can simply consider the problem, whether it holds

$$\mathcal{S}, s_0 \models \langle \tau^a(\delta) \rangle \, \mathbf{goal} \; .$$

The general model checking problem for PDL, i.e., the problem of checking whether an arbitrary PDL-formula is true in state s is PTIME-complete (Lange, 2006). Lange also presents an algorithm that calculates for a given transition system and PDL formula φ, the set of all states in which the formula is true in time $\mathcal{O}(|\varphi|^2 \cdot n^5)$ (where n is the number of states in the structure). Of course, in the setting of the theorem presented above, no nested modalities need to be taken into account.

A natural generalization of the claim of Theorem 5.2 is to consider sets of goal states. As an immediate corollary of the theorem we obtain the following result.

Corollary 5.3. *Let $\mathcal{S} = \langle S, \rightarrow \rangle$ be a labeled transition system with label symbols in $\mathcal{L}$, where $|\mathcal{L}| \geq 3$.*

The decision problem whether, given some state s_0 in $\mathcal{S}$, a route instruction δ, and a set of states $Q \subseteq S$, the condition $Q \subseteq \mathrm{halt}(s_0, \delta)$ holds is solvable in polynomial time for the agent models without learning and NP-complete for the agent models with learning. □

To further strengthen the result we consider the decision problem whether a given set of states is equal to the halting set. For learning agents this adds the problem to check whether halting states are omitted in the given set. It turns out that this causes the problem to be D^P-complete (Papadimitriou and Yannakakis, 1984). This complexity class is defined by $D^P = \{ L_1 \cap L_2 \colon L_1 \in \mathsf{NP}, L_2 \in \mathsf{coNP} \}$ and is also known as BH_2 (Cai et al., 1988) as it is a part of the Boolean hierarchy.

The perhaps easiest example of a D^P-complete decision problem is the SAT–UNSAT problem: Given a pair of propositional formulae (φ_1, φ_2), the task is to show that φ_1 is satisfiable and φ_2 is unsatisfiable.

Theorem 5.4. *Let $\mathcal{S} = \langle S, \rightarrow\rangle$ be a labeled transition system with label symbols in $\mathcal{L}$, where $|\mathcal{L}| \geq 3$.*

The decision problem whether, given a state s_0, a route instruction δ, and a set of states Q in S, it holds $Q = \mathrm{halt}(s_0, \delta)$ can be solved in polynomial time for the agent models without learning, but is D^P-complete for the models with learning.

Proof. The claim regarding non-learning agents follows immediately from the proof of Theorem 5.2, since the halting set itself can be computed using the dynamic programming approach used there.

For the agent models with learning we observe first that, by Corollary 5.3, the problem $Q \subseteq \mathrm{halt}(s_0, \delta)$? is in NP. Moreover, the problem $\mathrm{halt}(s_0, \delta) \nsubseteq Q$? is in NP, since one can guess an execution trace of δ and check in polynomial time whether it terminates in a state not contained in Q. Thus the problem $Q = \mathrm{halt}(s_0, \delta)$?, conceived of as the pair of problems $(Q \subseteq \mathrm{halt}(s_0, \delta)?, \mathrm{halt}(s_0, \delta) \subseteq Q?)$, is in D^P.

We use the following decision problem to construct a polynomial reduction: Given a pair of undirected graphs (G_1, G_2), decide whether the first has a Hamiltonian path and the second has none.

As in the proof of Theorem 5.2 we first construct for each of the graphs a labeled transition system $\mathcal{S}_i$ (with start state $s_{0,i}$ and goal state $s_{\star,i}$) and a route description δ_i such that G_i has a Hamiltonian path if and only if δ_i has some execution trace from $s_{0,i}$ to $s_{\star,i}$. Further, for each state in $S_i \setminus \{s_{\star,i}\}$ we add a self-loop labeled by d_1. Then it holds for $\mathcal{S}_i$: $\mathrm{halt}(s_{0,i}, \delta_i) = S_i$ if there is a Hamiltonian path in G_i and $\mathrm{halt}(s_{0,i}, \delta_i) = S_i \setminus \{s_{\star,i}\}$ otherwise.

Now let $\mathcal{S}_3$ be a transition system that is simply a path $p_1 \dots p_n$ of length $|V_1|$ (where V_1 is the set of vertices of G_1) and where each link is labeled by d_1. We attach $\mathcal{S}_2$ to $\mathcal{S}_3$ to form a transition system $\mathcal{S}_2'$ by linking p_n to $s_{0,2}$ with label d_2. Trivially we have $\mathrm{halt}(s_{0,2}, \delta_2)$ in $\mathcal{S}_2$ is the same set as $\mathrm{halt}(p_1, \delta_1\delta_2)$ in $\mathcal{S}_2'$.

Finally, we combine $\mathcal{S}_1$ and $\mathcal{S}_2'$ to form $\mathcal{S}$ by adding a new state s_0 with links to $s_{0,1}$ and p_1 using label d_1. Now consider $Q := S \setminus \{s_0, p_1, \dots, p_n, s_{\star,2}\}$ and $\delta := d_1\delta_1\delta_2$ in $\mathcal{S}$. By construction of $\mathcal{S}$ and δ we know

$$S \setminus \{s_0, p_1, \dots, p_n, s_{\star,1}, s_{\star,2}\} \subseteq \mathrm{halt}(s_0, \delta) \subseteq S \setminus \{s_0, p_1, \dots, p_n\}$$

and $s_{\star,i}$ appears in the halting set if and only if G_i has a Hamiltonian path. Thus $Q = \mathrm{halt}(s_0, \delta)$ if and only if G_1 has a Hamiltonian path, but G_2 has none. □

The crucial task considered in Theorem 5.4 is to check whether each execution trace $s_0 s_1 \ldots s_n$ of a route description δ that starts in s_0 terminates in some state $s_n \in S_\star$. It is tempting to translate the problem in PDL as the question whether

$$\mathcal{S}, s_0 \models [\tau^a(\delta)]\,\mathbf{goal}$$

holds true — but this is incorrect, as the formula just asks whether each *terminating* execution of a route description δ, starting in s_0, halts in some state of $S_\star$. Of course, all agent models based on the model a^{s} are guaranteed to terminate. But in general this is not the case for the models based on a^{d}. In basic PDL there is no way to check whether a regular program has infinite executions. To express this one needs a dedicated new PDL construct (Pratt, 1980; Harel et al., 2000) or the **repeat**-construct mentioned in section 4.

5.2. Relating Paths to Descriptions

We are further interested in finding a description of given paths in $\mathcal{S}$.

Lemma 5.5. *Let $\mathcal{S}$ be a labeled transition system with label symbols in $\mathcal{L}$.*

The decision problem whether for a given path $\sigma = s_0 \ldots s_n$ in $\mathcal{S}$ and bound $k \in \mathbb{N}$, there exists a description δ of length at most k such that σ is contained in $tr(s_0, \delta)$ can be solved in polynomial time for all agent models.

Proof. For learning agents we first check whether $s_i = s_j$ for some $0 \leq i \neq j < n$. If this is the case then σ has no description δ with $\sigma \in \mathrm{tr}(s_0, \delta)$ as the agent would halt before the end of the given trace.

Clearly for all models based on a^{s} all descriptions must be of length at least n; they cannot be shorter as all states of the path must be visited. Given that σ is a path, there always is a description of length exactly n.

For an agent with default straight actions a suitable description may be of length less than n. To answer the decision problem we cast the sequence $s_0 \ldots s_n$ as a digraph: The states are the vertices and we add the (immediate) arcs (s_i, s_{i+1}) for $0 \leq i < n$. In this graph we explicitly add default straight actions. For each $0 < i < j \leq n$ we add an additional arc (s_i, s_j) if there is a turn expression $d \in \mathcal{L}$ such that

$$s_i \xrightarrow{straight} s_{i+1} \xrightarrow{straight} \cdots \xrightarrow{straight} s_{j-1} \xrightarrow{d} s_j$$

and for each $i \leq l < j - 1$, there is no transition $s_l \xrightarrow{d} s'$ for some s' in $\mathcal{S}$. Further, for each $0 \leq i < n$ we add an additional arc (s_i, s_n) if there is a turn expression $d \in \mathcal{L}$ such that

$$s_i \xrightarrow{straight} s_{i+1} \xrightarrow{straight} \cdots \xrightarrow{straight} s_n$$

and for $i \leq l < n-1$, there is no transition $s_l \xrightarrow{d} s'$ for some s' in $\mathcal{S}$. A shortest path search from s_0 to s_n in the constructed graph then yields the desired result. $\square$

As our last formal result, we show that optimizing description length for a given start and destination state is a non-trivial task for agents featuring learning combined with default straight actions.

Theorem 5.6. *Let $\mathcal{S} = \langle S, \rightarrow \rangle$ be a labeled transition system with label symbols in $\mathcal{L}$, where $|\mathcal{L}| \geq 4$.*

The decision problem whether for given states $s_0, s_\star \in S$, and bound $k \in \mathbb{N}$, there exists a route description δ of length at most k such that $s_\star$ is contained in $\mathrm{halt}(s_0, \delta)$ is (a) NP-complete for the agent models a^{dl} and a^{dgl}, and (b) solvable in polynomial time for all other considered agent models.

Proof. (b) For all agent models based on a^{s}, the claim follows easily from the proof of Lemma 5.5, since it suffices to perform a shortest path search and check the description of any shortest path whether the length bound is satisfied. Notice that in the models based on a^{s} shortest paths never contain any state multiple times.

For the agent models a^{d} and a^{dg}, we can also apply the same idea used in the proof of Lemma 5.5. But notice that in case of default straight actions, the shortest description generated in this way may allow only for paths in the original transition system that revisit states (for an example of such a situation see Westphal and Renz, 2011, Figure 4). Thus an agent with default straight actions and learning will not be able to reach the goal state, when it executes this shortest description.

(a) The problem is in NP, as we can guess and describe a suitable path satisfying the length bound (cf. Lemma 5.5). We will prove NP-hardness by a reduction from the MinimumSetCover problem (Karp, 1972).

Assume we are given the domain $B = \{b_1, \ldots, b_n\}$, a family of subsets $B_j \subseteq B$, and the decision problem whether there exists a selection of at most l subsets that covers B. We construct a suitable transition system that has a route description of length $2 \cdot n + 1 + l$ if and only if there is a selection of l subsets that covers B. The learning property is used to guarantee the cover, while default actions are required to achieve the proper length of the description.

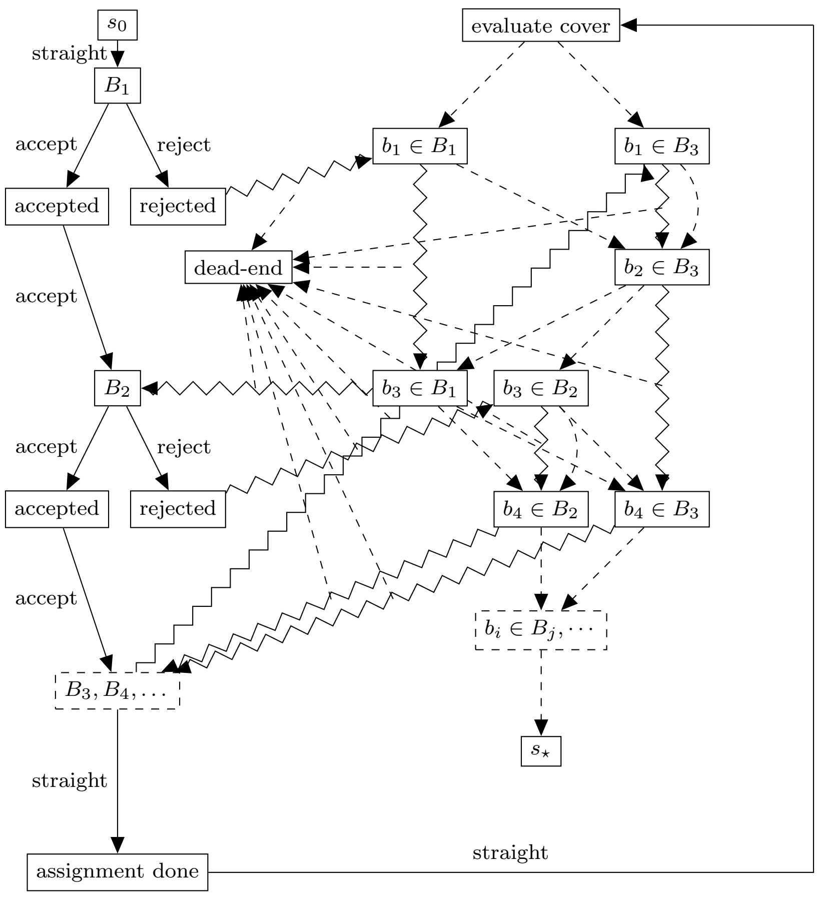


Figure 4. Schema of the transition system construction for a MinimumSetCover instance. Dashed arcs are labeled with "evaluate", zigzag arcs represent dummy chains.

The construction builds on a pair of start-goal states, three types of (sub-) transition systems, and four distinct turn labels, namely "straight" (our default action), "accept", "reject", and "evaluate". For each B_j we have a subsystem that can be traversed in two ways: either accept the subset for the cover or reject it. This type of subsystems builds the *selection.* The second type of subsystems evaluates the cover condition, our *evaluation matrix.* A selection not covering B blocks all short route descriptions to the goal state. Finally, the third type is a chain of dummy states of length $2 \cdot n$ linked by "straight" instructions, with the "evaluate" instruction immediately leading to a dead-end state. Such a chain is entered and exited by a "straight" instruction and thus can be completely skipped by either "accept" or "reject" instructions, but requires more than $2 \cdot n$ instructions to explicitly traverse it stepwise using “straight” and further cannot be skipped or traversed by “evaluate” instructions. The reader may refer to Figure 4 as an example of the overall construction described in the following.

For every B_j we have a state with two leaves: one accept, the other reject. Accepting a subset B_j requires two instructions, while rejecting it only requires one. Each reject leaf connects to a chain where we can traverse all “b_i in B_j”-states – for which b_i is in the current B_j – using the default action. Inbetween such states we place dummy chains. By construction these do not interfere with traversing all these states via the default action, but require more than $2 \cdot n$ explicit “straight” instructions, and cannot be traversed via “evaluate” ones. Note, the “b_i in B_j”-states are visited (i.e., *blocked* for future traversal) if and only if the reject leaf is traversed. These B_j-trees are sequentially connected and we add at the end an evaluation matrix over all the “b_i in B_j”-states that can be traversed via “evaluate” along the elements b_i. Note, the non-determinism used to connect over all possible “b_i in B_j”. There is a description using only “evaluate” instructions if and only if for each b_i there is some non-blocked “$b_i \in B_j$”-state. This description is of length $n + 1$.

For correctness, we argue as follows. Assume we choose not to traverse all B_j-states. This requires at least a “reject” and $2 \cdot n$ “straight” instructions to arrive in a “$b_i \in B_j$”-state. Further, this requires at least $n+1$ instructions to get to one reject-state plus reaching the goal state via “evaluate” instructions. In total such a description has at least $3 \cdot n + 1$ instructions – which is no better than simply selecting all B_j subsets.

Observe that the shortest instruction sequence through the evaluation matrix is $n + 1$ “evaluation” instructions. Stepping through the “straight” arcs of dummy chains requires more instructions, and “accept” or “reject” lead back to B_j-states. Further, if the chosen B_j do not cover B there is no path through the evaluation matrix using only “evaluate” instructions, and thus at least one dummy chain with $2 \cdot n$ states has to be traversed. Conversely, if there is a selection of l subsets that covers B, then there is a description of length $2 \cdot n + l + 1$ that leads from start to goal state, accepting the corresponding B_j subsets (length $n + l$) and traversing the evaluation matrix with $n + 1$ “evaluate” instructions. □

6. Evaluation

We report on an empirical evaluation of the agent models based on a route graph of Canberra, AUS, extracted from OpenStreetMap and covering approximately 20 km^2. The initially route graph contains 14 837 vertices and 7289 arcs. As point of interest we mainly used intersections in that area, and thus got a selection of 659 vertices in the route graph. The induced transition systems represent all relevant information about streets and intersections in the map. It is worth mentioning that each of the labeled transition systems has 1719 states with 4934 transitions. These figures already indicate the benefit of qualitative abstractions provided by labeled transition systems: to generate and evaluate route descriptions we can work with significantly smaller graphs.

The aim of the study was to measure the influence of both the chosen agent model and the used sector model (see section 2.2), when route descriptions are generated and evaluated on real world data.

In the study we used the $\mathcal{STAR}_2^r$ and $\mathcal{STAR}_4^r$ representation schemes as presented in section 2.2. For both representations the generated labeled transition systems differ only in the used instruction sets. As agent models we considered the execution models a^{s} (strictly executing agent), a^{dg} (default straight actions and goal recognition), and a^{dgl} (default straight actions, goal recognition, and learning). Based on a fixed randomly chosen set of 1000 distinct start-destination pairs, we generated shortest route descriptions for each of these agent models, that is, from all potential paths from the start to the destination location we selected one with a minimal description length assuming an agent behavior according to one of the agent models. Note that the optimization criterion is description length, not path length. Moreover, notice that for the different agent models the generated route descriptions may be based on different paths in the route network. Despite the (theoretically) computational complexity of these tasks for some of the models, it turned out to be computationally feasible in all cases — most likely due to the low average degree of real-world route networks. In the case of a^{dgl} we used a common branch-and-bound search for shortest descriptions. The search took less than half a second for any pair.

Our results in Table I show that with regard to description length, there are only small differences between the two considered qualitative sector models and also between routes optimized for the models a^{dg} and a^{dgl}. There is however a significant reduction in the length when generating route descriptions with the default action assumption. On average the length is almost halved. Considering the two qualitative sector models, there is little reduction in the length of descriptions. Choosing $\mathcal{STAR}_4^r$ over $\mathcal{STAR}_2^r$ allows to take more advantage of the default actions as we can express "at the next opportunity *slightly* left" which is more selective than "left" in $\mathcal{STAR}_4^r$. In the case of model a^{s}, the shortest path is of course unaffected by the qualitative

Table I. Average and standard deviation of the length of route descriptions optimized for different agent models in different qualitative representations.

	$\mathcal{STAR}_2^r$	$\mathcal{STAR}_4^r$
a^{s}	17.83 (7.14)	17.83 (7.14)
a^{dg}	10.06 (4.04)	9.96 (4.19)
a^{dgl}	10.05 (4.04)	9.96 (4.19)

Table II. Geometric mean (and standard deviation) of the success probability of route descriptions. The tables lists the success probabilities, when an optimized route description is generated for model a, but executed by an agent of model a' (rows optimized for, column evaluated on).

$\mathcal{STAR}_2^r$ Opt.\Eval.	a^{s}	a^{dg}	a^{dgl}
a^{s}	0.41 (0.38)	0.41 (0.38)	0.41 (0.38)
a^{dg}	0.01 (0.11)	0.25 (0.33)	0.24 (0.33)
a^{dgl}	0.02 (0.12)	0.29 (0.35)	0.29 (0.35)

$\mathcal{STAR}_4^r$ Opt.\Eval.	a^{s}	a^{dg}	a^{dgl}
a^{s}	0.62 (0.36)	0.62 (0.36)	0.62 (0.36)
a^{dg}	0.01 (0.11)	0.56 (0.39)	0.56 (0.39)
a^{dgl}	0.01 (0.11)	0.61 (0.38)	0.61 (0.38)

representation as the length of the shortest description is the same as the length of the shortest path (shortest path here refers to the number of intersections passed on the route).

In a second step, we compared the probability of reaching the destination when an agent (with agent model a) follows a description that has been optimized for an agent model a'. Note that the agent model imposes a uniform distribution on decision choices, which induces a probability distribution over paths. The descriptions are the same as the ones previously used, i.e., they are optimized for length. Table II lists the geometric mean of the success probability of reaching the desired destination. Overall, the numbers may seem low, but readers should keep in mind that one ambiguous turn description often already reduces the success probability from 1.0 to 0.5. The results demonstrate that assuming default straight actions for the generation of route descriptions usually reduces the probability of reaching the destination. This is plausible because descriptions can rely on many (potentially ambiguous) *implicit* straight actions. However, it can also be seen that the more refined qualitative sector model $\mathcal{STAR}_4^r$ with the smaller "straight" sector compared to $\mathcal{STAR}_2^r$ mitigates this effect.

We further observed that the average number of distinct paths to the destination under the same given instruction for a^{dgl} is about 1.14-1.38 for $\mathcal{STAR}_2^r$ and 1.04-1.064 for $\mathcal{STAR}_4^r$. For a^{dg}, infinitely many paths to the destination might exist in cases where loops in the network may be traversed arbitrarily often. We also found that descriptions optimized for a^{dg} can also be reasonably used by a^{dgl}-agents. For only 6.9% of the descriptions in $\mathcal{STAR}_2^r$ (1.9% for $\mathcal{STAR}_4^r$) states had to be revisited.

We observe that with regard to success probabilities the influence of the sector model seems stronger than the influence of the agent model. However, the results also demonstrate that taking advantage of default actions can significantly reduce the length of descriptions where the lower success probabilities can be mitigated with a more refined instruction set. Moreover, assuming learning agents does have an effect on both the generation and evaluation of descriptions. One can argue that testing for repeated states under route descriptions during generation can be ignored as most of the descriptions here do not lead into cycles. On the other hand, the simple branch-and-bound algorithm we have used rectifies cycles if they appear (6.9% of $\mathcal{STAR}_2^r$ descriptions) and is still efficient to compute. Further, the instructions generated for a^{dgl} are shorter than those for a^{s}, better than those for a^{dg}, and almost as reliable as those for a^{s}. Instructions generated for agents with default actions and learning are superior considering a description provider wants to balance length and reliability.

Finally, we conjecture that increasing the detail of the sector model $\mathcal{STAR}_n^r$ leads to better descriptions, but on the other hand it is questionable whether the resulting descriptions can still be processed by humans in a reasonable way. A natural generalization of the work presented here is to consider turn instructions that allow to reference landmarks (such as "turn right at the church"). We also refer the reader to other works that focus on optimizing reliability and success probabilities. For example, Haque et al. (2006) consider optimizing for approximations of success probabilities. And Westphal and Renz (2011) measure the quality of route instructions in terms of the likelihood that the destination is reached. For this they present a probabilistic extension of the concept of transition systems.

7. Conclusion and Future Work

In this article we have introduced a simple qualitative representation of street networks that showed to be useful for generating optimized qualitative route descriptions. Such descriptions are rarely unambiguous to humans and we propose to take ambiguities into account when descriptions are generated and evaluated. A model of how route descriptions are processed needs to be chosen, whenever their quality is evaluated (in the literature, such models are often described in an informal manner only). However, the used agent model may also have an effect on which descriptions are generated. To our knowledge, this aspect has not yet been systematically investigated in the literature.

The agent models discussed here are close to the simple model used by Haque et al. (2006) to evaluate route descriptions. Although from a practical (human) point of view, these models are rather simplifying (some of them can even be defined in propositional dynamic logic), these models already allow to make rather general claims about the computational difficulty to generate and execute route instructions as well as to answer queries on the descriptions. For elementary queries about relations between paths and descriptions as well as for optimizing description length, we have provided computational complexity bounds in dependency of the agent model. Some of these results imply that presumably there are no simple translations from the graph-theoretic formalization into standard propositional dynamic logic. Several of our results indicate that it is (theoretically) infeasible to find optimal descriptions for certain criteria. Especially the identification of possible end states of routes under a fixed description is already hard for some simple agent models. In particular, these results hold despite the simplicity of the basic models discussed here and thus are expected to apply to more sophisticated models as well.

In this context, it is interesting to see that not only the agent model, but also the spatial vocabulary used to describe turn actions (more precisely, the sector model) plays a crucial role on the success probability of route descriptions. Our empirical results show that already easy assumptions about the agent (such that some default action is executed, when the instruction cannot be performed), leads to a significant decrease of the success probability of the description. However, we have also seen that this effect can be mitigated by switching to a more refined sector model.

Taking ambiguities into account, enables us to identify the expected traversed paths for given route instructions. In turn this facilitates a number of interesting criteria that can be used to evaluate and optimize route descriptions: examples include the probability of reaching the destination, the expected distance to the destination, or the set of possible places where agents could end up. If future evaluations show that for specific agent models the generation of optimal descriptions is in fact not practically feasible (e.g., optimizing the chance of reaching the destination), it will be necessary to analyze possible approximations.

The results of our work indicate that it is important to balance a number of criteria when optimizing routes in order to guarantee robust descriptions for different underlying route networks and different agents processing them.

Acknowledgments

The authors thank the anonymous reviewers for helpful feedback and suggestions. Further, the authors are grateful to Julien Hué and Robert Mattmüller for fruitful discussions. This work was supported by DFG (*SFB/TR 8 Spatial Cognition*, project R4-[LogoSpace]), an ARC Future Fellowship (FT0990811), and a joint Go8/DAAD project (D/08/13855).

References

Bradfield, J. and C. Stirling: 2006, 'Modal mu-calculi'. In: P. Blackburn, J. van Benthem, and F. Wolter (eds.): *The Handbook of Modal Logic*. Elsevier, pp. 721–756.

Cai, J.-Y., T. Gundermann, J. Hartmanis, L. A. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung: 1988, 'The Boolean hierarchy I: Structural properties'. *SIAM Journal on Computing* **17**(6).

Diestel, R.: 2010, *Graph Theory*. Springer, 4th edition.

Duckham, M. and L. Kulik: 2003, '"Simplest" Paths: Automated Route Selection for Navigation'. In: W. Kuhn, M. F. Worboys, and S. Timpf (eds.): *Spatial Information Theory. Foundations of Geographic Information Science, International Conference, COSIT 2003, Proceedings*. pp. 169–185, Springer.

Duckham, M., S. Winter, and M. Robinson: 2010, 'Including landmarks in routing instructions'. *Journal of Location Based Services* **4**(1), 28–52.

Edelkamp, S. and S. Schrödl: 2003, 'Route Planning and Map Inference with Global Positioning Traces'. In: R. Klein, H.-W. Six, and L. M. Wegner (eds.): *Computer Science in Perspective, Essays Dedicated to Thomas Ottmann*. pp. 128–151, Springer.

Fischer, M. J. and R. E. Ladner: 1979, 'Propositional Dynamic Logic of Regular Programs'. *Journal of Compututer and System Sciences* **18**(2), 194–211.

Garey, M. R., D. S. Johnson, and L. J. Stockmeyer: 1976, 'Some simplified NP-complete graph problems'. *Theoretical Computer Science* **1**(3), 237–267.

Haque, S., L. Kulik, and A. Klippel: 2006, 'Algorithms for Reliable Navigation and Wayfinding'. In: T. Barkowsky, M. Knauff, G. Ligozat, and D. R. Montello (eds.): *Spatial Cognition*. pp. 308–326, Springer.

Harel, D., D. Kozen, and J. Tiuryn: 2000, *Dynamic Logic*. MIT Press.

Karp, R. M.: 1972, ‘Reducibility Among Combinatorial Problems’. In: R. E. Miller and J. W. Thatcher (eds.): *Proceedings of a symposium on the Complexity of Computer Computations*. pp. 85–103, Plenum Press, New York.

Klippel, A. and D. R. Montello: 2007, ‘Linguistic and Nonlinguistic Turn Direction Concepts’. In: S. Winter, M. Duckham, L. Kulik, and B. Kuipers (eds.): *Spatial Information Theory, 8th International Conference, COSIT 2007, Proceedings*. pp. 354–372, Springer.

Krieg-Brückner, B., U. Frese, K. Lüttich, C. Mandel, T. Mossakowski, and R. J. Ross: 2004, ‘Specification of an Ontology for Route Graphs’. In: C. Freksa, M. Knauff, B. Krieg-Brückner, B. Nebel, and T. Barkowsky (eds.): *Spatial Cognition IV: Reasoning, Action, Interaction, International Conference Spatial Cognition 2004, Revised Selected Papers*. pp. 390–412, Springer.

Kuipers, B.: 2000, ‘The Spatial Semantic Hierarchy’. *Artificial Intelligence* **119**(1-2), 191–233.

Lange, M.: 2006, ‘Model checking propositional dynamic logic with all extras’. *Journal of Applied Logic* **4**(1), 39–49.

Mark, D. M.: 1986, ‘Automated Route Selection For Navigation’. *Aerospace and Electronic Systems Magazine* **1**(9), 2–5.

Papadimitriou, C. H. and M. Yannakakis: 1984, ‘The Complexity of Facets (and Some Facets of Complexity)’. *Journal of Computer and System Sciences* **28**(2), 244–259.

Pratt, V. R.: 1980, ‘Application of Modal Logic to Programming’. *Studia Logica* **11**(2–3), 257–274.

Renz, J. and D. Mitra: 2004, ‘Qualitative Direction Calculi with Arbitrary Granularity’. In: C. Zhang, H. W. Guesgen, and W.-K. Yeap (eds.): *PRICAI 2004: Trends in Artificial Intelligence, 8th Pacific Rim International Conference on Artificial Intelligence, Proceedings*. pp. 65–74, Springer.

Renz, J. and S. Wölfl: 2010, ‘A Qualitative Representation of Route Networks’. In: *ECAI 2010 - 19th European Conference on Artificial Intelligence, Proceedings*. pp. 1091–1092, IOS Press.

Richter, K.-F. and M. Duckham: 2008, ‘Simplest Instructions: Finding Easy-to-Describe Routes for Navigation’. In: T. J. Cova, H. J. Miller, K. Beard, A. U. Frank, and M. F. Goodchild (eds.): *Geographic Information Science, 5th International Conference, GIScience 2008, Proceedings*. pp. 274–289, Springer.

Streett, R. S.: 1982, ‘Propositional Dynamic Logic of Looping and Converse Is Elementarily Decidable’. *Information and Control* **54**(1/2), 121–141.

Westphal, M. and J. Renz: 2011, ‘Evaluating and minimizing ambiguities in qualitative route instructions’. In: I. F. Cruz, D. Agrawal, C. S. Jensen, E. Ofek, and E. Tanin (eds.): *19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2011, Proceedings*. pp. 171–180, ACM.

Westphal, M., S. Wölfl, B. Nebel, and J. Renz: 2011, ‘On Qualitative Route Descriptions: Representation and Computational Complexity’. In: T. Walsh (ed.): *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, 2011*. pp. 1120–1125, IJCAI/AAAI.

Address for Offprints: Kluwer Prepress Department
P.O. Box 990
3300 AZ Dordrecht
The Netherlands