Affordance-based Algorithms for Categorization of Road Network Data

Technical Report v. 1

Simon Scheider and Jörg Possin

Westfälische Wilhelms-Universität Münster, Weseler Strasse 253 D-48151 Mnster, Germany,
simon.scheider@uni-muenster.de

Abstract. We propose algorithms that can be used to automatically check road network data for consistency, and to categorize parts of it as junctions. The main idea is to use locomotion affordances, i.e. restricted patterns of supported movement, in order to define the categories channel network and junction, and to motivate and derive tractable verification and classification strategies for them. Affordance based category definitions are part of a wider project of the authors to develop a grounded theory for geographic data, especially a theory of channel networks. The theory comes with a natural semantics in terms of reproducible observation procedures. It therefore provides a very general approach to map road network databases into it, and to check or categorize their features. The algorithms are applied using extracts from Open Street Map.

1 Introduction

Even though there are informal standards for road network labeling available (e.g. the GDF standard, ISO 14.825:2004), geometries and category labels in different commercial road network data are far from denoting equivalent things. Furthermore, the available categories often lack high-level features, such as junctions and roads, as well as labels for applications that go beyond navigation systems [1]. User-generated geodata, such as *Open Street Map (OSM)*, allows for flexible grassroot labeling, but the many quality issues[1] make semantic heterogeneity of such data an even greater challenge.

In [2], we have proposed a theory of *channel networks* which is grounded in observable locomotion affordances. Into this theory, road network data, data schemata and related ontologies map in an obvious way. We have given affordance-based definitions of a *channel network* and an *n-way junction*, and showed that the junction definition is satisfied by a collection of most common junction types, for example 4-way-intersections, diamond interchanges, jughandles and cloverleafs. Using this theory to semantically annotate, check and translate data about road networks requires to have tractable algorithms available. As we will demonstrate in this paper, the grounded semantics of the channel network theory not

[1] http://wiki.openstreetmap.org/wiki/Category:Quality_Assurance

only allows for transparent categorization across applications, but is also a powerful inspiration for tractable algorithmic solutions.

We will shortly summarize the basic ideas from [2] about channel networks in Section 2 and junction affordances in Section 3. In Section 2.2, we will show how channel networks should be interpreted in terms of OSM data. In Section 4, we develop a tractable algorithm for categorizing junctions, and discuss evaluation results in Section 5 using data extracts from OSM.

2 Grounding Road Networks in Affordances

In [2], we argued that observation primitives for individuating road networks and their features can be found in Gibson's *meaningful environment* [3], more correctly in *affordances*. Affordances are directly perceivable action potentials useful for defining categories: A 'chair', for example, is something humans can 'sit on'. In a more general paper about grounding concepts in geographic data [4], we used perceivable affordance relations, like simulated locomotion steps, to define *substances, surfaces and media (e.g. air media)* as *wholes* (equivalence classes, compare also [5], Section 5.3) with respect to these relations. We summarize in the next subsection, how road networks can be conceived as a special kind of a medium, without going into details of the formalism in [2].

2.1 Channel Networks and what they afford

We defined a *channel network* in [2] as a special medium that allows supported locomotion to reach each point on a support surface from each other point via channels. The observable fact that a pair of locations in the environment is connected by a continuous support surface, that it affords *supported locomotion*, is denoted by a primitive relation $SupportC$. *Channels* are parts of support-connected media that restrict supported movements to a regulated flow into only one direction, from an entry portal to a distinct exit portal (Figure 1). Channels do not overlap each other and are not self-connected at their portals. They exist in finite amounts and can be directly perceived: Channels are discrete parts of the environment in which supported movements are restricted *by convention*, for example by perceivable signs on the road surface. One can think of a channel as one side of a bidirectional road or a one way street between two intersections. If two channels are connected such that one can move from one to the other via appropriate portals, this can be expressed by the irreflexive affordance relation $LeadsTo(x, y)$ [2], whose domain and range consists of channels in the environment of the observer. The *transitive closure* of this relation, which was called $ReachableFrom$ in [2], denotes those paths of channels that afford locomotion in the environment.

A channel network can be defined as a *mereological sum of channels being a whole with respect to reachability*, so that every channel is reachable from every other one inside of it [2]. In consequence, channels must be reachable from themselves via others; it is always possible to return to the origin; the minimal

model of a channel network consists of two mutually reachable ones (because of irreflexivity); and channel networks never contain *graveyards and factories*, channels without the possibility to leave or enter. One practical consequence of this is that dead ends must allow for U-turns. If we consider a network's set of channels as a set of vertices V, and the relation $LeadsTo$ as the arc relation A, then it is described by a *directed graph* $D(V, A)$. It is then provable that a *channel digraph* D does not have loops and multiple arcs, and that it must be *strongly connected* in a graph theoretical sense [2].

2.2 Constructing and Testing Channel Networks with OSM Data

In order to interpret data schemas like OSM into the theory of channel networks, we have to tell what in the schema corresponds to channels, the $LeadsTo$ and the $SupportC$ relation.

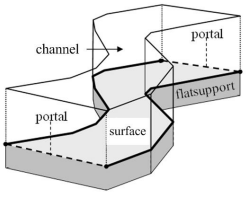

Fig. 1. Illustration of a channel

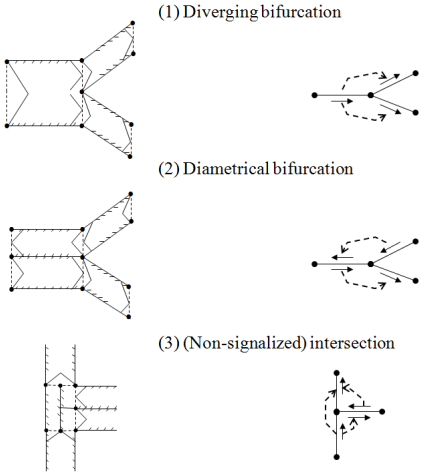

Fig. 2. Channel configurations (left) corresponding to road network data elements (right).

Every navigation oriented *road network data model* is based on an *embedded graph of street segments* as edges and intersections as nodes, not to be confused with the channel digraph. The semantics of a street segment in terms of our theory is given by a maximal support-connected sum of locations that does not cross channel portals. These portals are denoted by two intersection nodes, segment end nodes with degree > 2 (Figure 2, right). We can think of a segment as the surface of a road between two intersections, regardless of whether this surface belongs to one or more channels. Street segments therefore do not denote channels directly: Channels correspond to the 'sides' of the surface denoted by a

bidirectional segment, and only in case of one way streets they correspond to a street segment (compare Figure 2). A *LeadsTo* pair is represented as a support-connected pair of channels that is not affected by *turnoff prohibition*. So, in order to be mappable into the channel network theory, we need not only to dissect the segment graph into channels, but also to supplement it by a 'turnoff relation' on top of them. The *LeadsTo* relation is not captured by the segment graph alone: the two branching edges of a diametrical bifurcation (see Figure 2), for example, are connected at a common node, but do not lead to each other. In the case of OSM, the street segment graph is given by the sets of *intersection-free navigable way parts* (tagged as certain 'highways') and their *nodes*. We reconstructed the channel digraph from OSM using available tags and the following simplistic inference rules:

1. Dissect all navigable ways into intersection-free parts with all inner nodes of degree ≤ 2
2. All *bidirectional parts* imply 2 channels c_1, c_2, one *towards* and one *from* an incident node. The channels *may not lead into each other* (no U-turns), but are *support-connected* $SupportC(c_1, c_2)$. We abbreviate this relation as $Adjacent(c_1, c_2)$.
3. All *oneway parts* imply one channel, and all *dead end parts* imply two channels leading to each other at the dead end node.
4. At every *OSM node*, all towards-from pairs of channels that are not adjacent (see above) and not subject to any *turnoff restrictions*[2] lead to each other.

For a channel digraph D reconstructed from OSM in that way, we may now test whether it satisfies the affordance based definition of a channel network given in 2.1. This is equivalent to checking whether D is *strongly connected*, using e.g. *Tarjan's* well known SCC-algorithm [6]. For geographically bounded subgraphs, we could test for *local strong connectedness* by first closing all entry-exit pairs at those borders, connecting them to a common synthetic vertex, before running the Tarjan algorithm. Since the problem of categorizing whole road networks in this way is straightforward, we will leave the details to the interested reader.

3 Junctions and what they afford

We now turn our focus to the properties of subgraphs of a channel network that can be categorized as junctions. In this section, we summarize ideas already presented by the authors in [2]. An *induced subgraph* $F(U, O)$ has a subset of vertices $U \subseteq V$ of the channel network, and the set of arcs with elements $e \in A$ connecting any pair $\{x1, x2\} \subset U$, is exactly its set of arcs, $e \in O$. A channel network feature, like a road or junction, is an induced subgraph of D, because the selection of channels does not influence the affordance relation between them.

[2] In comparison to commercial navigation networks, turnoff restrictions are only sparsely recorded in OSM via so called *restriction relations*, see `http://wiki.openstreetmap.org/wiki/Relations/Proposed/Turn_Restrictions`. Note that this introduces some uncertainty into our evaluation.

The essential *affordance properties* of such induced subgraphs now seem to arise from reachability of some of their vertices, called *entries, exits and gates* (Figure 3). Consider all arcs from $A \setminus O$ that are incident with vertices in U (connecting F with its complement in D), and call them *in-/out bridges of* F. We call a vertex $v \in U$ an *entry* of F, if and only if it is a terminal vertex of an in-bridge, and an *exit* if and only if it is an initial vertex of an out-bridge. The set of bridge-incident vertices that are either entries or exits (but not both), is called *gates of* F. Because a channel digraph D is strongly connected, it follows that for any *proper subgraph* F, there always exist *at least one entry and one exit* [2].

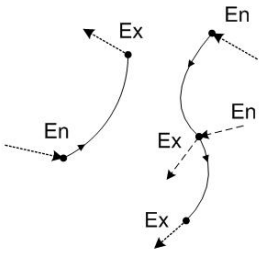

Fig. 3. An induced subgraph of a channel network and its entries and exit vertices (indicated by dotted arrows, gates are narrowly dotted).

Definition 1 (n-way junction). *An n-way junction is an induced subgraph F of a channel digraph D, which*

1. affords $n-1$ (n $\geq$ 3) navigational choices for n entries, as there is a walk from each of the n entries to $n-1$ exits (except into the opposite direction), and from $n-1$ entries to each of n exits (*total reachability*),
2. affords movements to enter and leave through gates (*discreteness of navigational action*),
3. does not contain a smaller n-way junction (*minimality I*) and
4. has entries and exits with a *minimal vertex degree of two*. Entries have either more than one internal successor or an internal predecessor. Analogously, exits have either more than one internal predecessor or an internal successor (*minimality II*).

We discuss and motivate these properties in the rest of this section by analyzing a *median u-turn junction* (see Figure 4). At a median u-turn intersection, the main road, a dual carriageway, intersects the minor road, which is a bidirectional road. It is prohibited to turn off to the left from the major road or onto the major road at the intersection point. These left turns are only possible via u-turn channels (see data model in Figure 4).

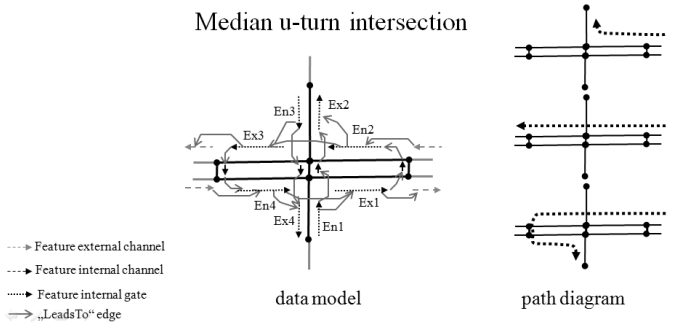

Fig. 4. A median u-turn ('Michigan left') intersection data model.

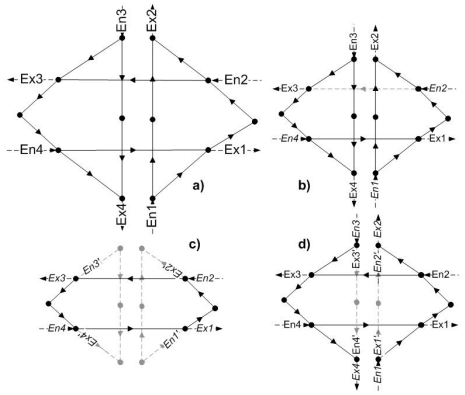

Fig. 5. Illustration of junction properties by a median u-turn channel subgraph.

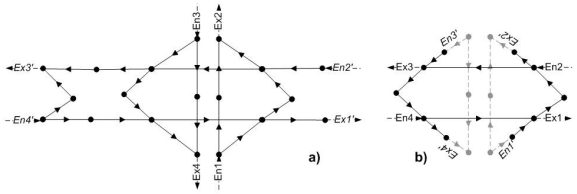

Fig. 6. Illustration of minimality properties by a median u-turn channel subgraph.

Channels are indicated by arrows alongside their embedded street segment lines. The *LeadsTo* relation among channels is indicated by grey curved arrows that lead from a channel to its successor. *Exits and entries* of the subgraph are numbered anticlockwise. In the remainder, we will refer to the correspondent *channel digraph* in Figure 5 a, in which channels are depicted as vertices and the LeadsTo relation as arcs. The exits and entries in these figures have equivalent numbers. The first and most obvious property is that junctions afford paths

from each entry to each exit (except the one in the opposite direction) (*total reachability*). Thus they enforce a navigational choice. If a driver has entered a junction, he is afterwards forced to take a directional decision by taking one of a set of $n-1$ paths inside of it. If one leaves out the dotted arc in Figure 5 b, then entries one, two and four are affected and lose their paths to exits three and four, so that this subgraph is not a junction anymore. The second property, *discreteness of navigational action*, seems to be common to every *road network feature*, thus also to a road. If people are speaking of 'staying on or leaving a road/junction', we suggest that they mean *discrete actions*: The process of entering, staying on, or leaving a road/junction is unambiguous for a moving object inside of a channel. The subgraph in Figure 5 c (in black) is a part of the median u-turn that satisfies the *total reachability* assumption. But because neighboring exits and entries collapse, this subgraph violates the discreteness property.

We furthermore need *minimality assumptions* for junctions. This is because one can usually supplement a junction with further channels such that it satisfies the other properties. In Figure 6 a, we have extended the median u-turn at the left end by completing the dual carriageway road. In a first sense, a junction is minimal because it *never contains a smaller junction*. This is because junctions keep the first two properties if one adds an extra road to them (Figure 6 a). In a second sense, we require a *minimal vertex degree* for entries and exits. Why is this a legitimate property of junctions? Take for example the entry En3' with degree one in Figure 6 b. All paths from this entry must cross the one edge incident to En3'. And because no other path (from another entry) uses this edge, the subgraph can safely be shortened by this edge while retaining its total reachability property. The same applies for exits. The second minimality criterion is necessary because it is generally not implied by the first. Compare the discussion in [2].

4 Algorithm for Categorizing Junctions

In this section, we will develop an algorithm that implements Definition 1 in a tractable way. The pseudocode in this section makes use of some well known operations on common data types and lists. Type variables are denoted by letter A and well known types by their uppercase names. For example,

- $[\,] : A^* \to List$, takes an array of something and constructs a list, as in $[a, b]$,
- $.[\,] : List \times Integer \to A$, returns the element of a list at the specified index, e.g. $[a, b].[0] = a$,
- $.append : List \times A \to List$, appends elements at the end of a list, e.g. $[a].append(b) = [a, b]$,
- $length : List \to Integer$, returns the length of a list,
- $.Contains : List \times A \to Bool$, tests whether a list contains an element, and
- $[]$ denotes the empty list.

All other functions are explained in the text of the following subsections.

4.1 Checking Junction Properties by DFS-Backpropagation

In order to inspect the property *total reachability* of Definition 1, we have to collect all paths from all entries of a subgraph F to all of its exits. As we will see in Section 4.2, in order to check for minimality I, we will have to collect these paths also for *'entry/exit candidates'* inside of the subgraph F. For the sake of simplicity, we propose to collect all paths from all vertices to all other vertices using a time efficient solution based on *depth-first search*. If there is a problem with storage space or efficiency, then this algorithm can be slightly modified such that only paths between entry/exit candidates (see definition in Section 4.2) will be collected.

Suppose that a channel network $D(V, A)$, a directed graph, is given by a domain of vertices V and the functions $Succ : V \to V$ and $Pred : V \to V$ denoting the arcs. We furthermore suppose that we intend to check whether a given set of vertices U of induced subgraph $F(U, O)$ is a junction or not[3]. For reasons of efficiency, we will use a flag $Subgraph$ to identify vertices in U. This flag is set in Algorithm 1, which also identifies *exits, entries* (stored in lists $Entries$ and $Exits$) and the *vertex degrees* in subgraph F. Beginning at one entry of F, Algorithm 2 recursively visits vertices of the subgraph F in DFS fashion: It colors them grey while moving to unvisited (white) vertices, stops if vertex was already visited or does not have successors, backtracks to the last forking vertex coloring all vertices on its way in black, and then moves forward until exhaustion, producing a 'spanning tree' subgraph. Edges of this tree are marked by the flag $ForwardEdge$. Back-edges, i.e. edges pointing from a tree vertex to a vertex up in the tree (thus producing a cycle), are stored in list $Cycleedges$.

The essential idea in Algorithm 2 is to use the DFS-principle of *backpropagation* to broadcast the information about (1) reachable vertices and (2) the paths to these reachable vertices from each vertex of the subgraph upwards in the tree. So when the algorithm has stopped building the tree, and each vertex has been backtracked (and is thus colored in black, not just grey), then each vertex v in the tree has a list $Reaches(v).[0]$ of all vertices *reachable in this tree* (as well as a list of corresponding paths $Reaches(v).[1]$). Time complexity of this algorithm is, unlike ordinary DFS, quadratic in the size of the subgraph $F(U, O)$ in the worst case, because for each vertex, we collect all paths to other vertices[4].

But the subgraph F will frequently contain *circuits*. This complicates the situation, since the collected tree paths do not necessarily contain the paths through those vertices upwards in the tree that are reachable from downwards in a cycle. The paths stored in a vertex that was already visited are indeed also backtracked into the circuit (compare lines 10-12 in Algorithm 2). But if this vertex happens to be still in grey, then the reachability information is not there yet. We therefore have to do a separate backpropagation after the execution of

[3] We will see in Section 4.2, that our proposal also involves an efficient solution to search for junctions in arbitrary channel digraps.

[4] If we restrict the collected paths to entry exit candidates, i.e. vertices of degree 3, time complexity will be much closer to linear time.

Algorithm 1 J-PREPARE(U): Prepare subgraph and detect entries and exits

Require: U is a subset of vertices V of channel digraph D

For all $v \in V$, empty flag $Subgraph(v) \leftarrow$ **false** and $In(Out)Degree(v) \leftarrow 0$
$Entries \leftarrow []$; $Exits \leftarrow []$; // Create empty lists
for all $v \in U$ **do** // Set flag for subgraph membership
    $Subgraph(v) \leftarrow$ **true**
for all $v \in U$ **do** // Collect all entries and exits of U
    while $Pred(v) = v'$ **do**
        if $Subgraph(v') =$ **false then**
            $Entries \leftarrow Entries.append(v)$
        else
            $InDegree(v)$++ // Countup internal in-degree of v
    while $Succ(v) = v'$ **do**
        if $Subgraph(v') =$ **false then**
            $Exits \leftarrow Exits.append(v)$
        else
            $OutDegree(v)$++ // Countup internal out-degree of v

Algorithm 2 TR-DFS(v): Total Reachability Depth-First Search

Require: $Visit(v) = white$; $Subgraph(v) =$ **true**;

$Visit(v) \leftarrow grey$ // Color vertex as visited but not yet backtracked
$Reaches(v).[0] \leftarrow Reaches(v).[0].append(v)$ // Register new reachable vertex locally
while $Succ(v) = v'$ **do**
    if $Subgraph(v') =$ **true then**
        if $Visit(v') = white$ **then**
            $ForwardEdge([v, v']) \leftarrow$ **true** // Mark edge as forward
            TR-DFS(v') // Call recursion
        else if $Visit(v') = grey$ **then** // Collect back-edges to update cycles
            $Cycleedges \leftarrow Cycleedges.append([v, v'])$
        for all $i = 0; i < length(Reaches(v').[0]); i$++ **do**
            $Reaches(v).[0] \leftarrow Reaches(v).[0].append(Reaches(v').[0].[i])$
            // Backpropagate reachable vertices
            $Reaches(v).[1] \leftarrow Reaches(v).[1].append(Reaches(v').[1].[i].append(v'))$
            // Backpropagate the paths to these vertices
$Visit(v) \leftarrow black$ // Color vertex as already backtracked

Algorithm 3 BACKP(y,z,z'): Independent backpropagation for cycles

for all $i = 0; i < length(Reaches(z)[0]); i$++ **do** // Backpropagation of paths
    $Reaches(y).[0] \leftarrow Reaches(y).[0].append(Reaches(z).[0].[i])$
    $Reaches(y).[1] \leftarrow Reaches(y).[1].append(Reaches(z).[1].[i].append(z))$
while $Pred(y) = x$ **do**
    if $ForwardEdge([x, y]) =$ **true** AND $z' \neq x$ **then**
        BACKP(x,y, z') // Recursion

Algorithm 2, which backtracks vertices at cycleedges, see Algorithm 3. Since subgraph F is not necessarily strongly connected, nor even connected, it has to be covered by a forest, i.e. a set of trees. To this end we have to execute Algorithm 2 iteratively until exhaustion. The exhaustive execution of TR-DFS and also the cycle backtracking of Algorithm 3 are both controlled by Algorithm 4. This procedure starts TR-DFS at a yet unvisited entry and goes on until all entries were visited[5]. Since the call to TR-DFS is done only once for all visited vertices and edges, time complexity is still the same. Once Algorithm 4 terminates, all necessary information for testing the junction properties are collected. Algorithm 5 implements the test for total reachability, junction order, discreteness and minimality II for the subgraph currently marked with the flag *Subgraph*.

Algorithm 4 J-PROCESS(): Build a spanning forest for the current subgraph

Require: Flag *Subgraph* is set, and list *Entries* is filled
1: Set all vertices $v \in D$: $Visit(v) \leftarrow white$; $Reaches(v) \leftarrow [[], [[]]]$
2: For all edges $(x, y) \in D$ set $ForwardEdge([x, y]) \leftarrow$ **false**
3: $Cycleedges \leftarrow []$ // Create empty lists
4: **for all** v in *Entries* **do** // Call recursive TR-DFS until exhaustion
5: **if** $Visit(v) = white$ **then** // Start call only if entry was not visited yet
6: TR-DFS(v)
7: **for all** (y,z) in *Cycleedges* **do** // Update cycles
8: BACKP(y,z,z)
9: Delete (y,z) from *Cycleedges*

4.2 Checking for minimality I

In order to check for minimality I, we have to test whether a given induced subgraph $F(U, O)$, the candidate junction, contains no smaller induced subgraph which is also a junction candidate (see Definition 1). It is obvious that by solving this problem, we also solve the problem of searching for a junction in an arbitrary network. In a naive approach, we could just iterate through all true subsets U' of its set of vertices U, each time run J-PREPARE(U') and J-CHECK(n), and return a positive result if these checks are all negative. This procedure has inefficient exponential time complexity of $O(2^{|U|})$, it is *intractable*, since we have to iterate through the power set of U by switching all vertices on and off in all combinations.

A more efficient solution would have to constrain the size of the subset we are searching for. Time complexity is then only polynomial in data size, with the polynomial bounded by the size of the target subset m. For example, it

[5] This is more efficient than starting from arbitrary vertices, because it helps averting unnecessarily many subtrees in the forest if entries happen not to be reachable from such a starting vertex.

Algorithm 5 J-CHECK(n): Junction Checks (except minimality I)

Require: n is the junction order
Require: J-PREPARE() and J-PROCESS() have been executed
1: Set $Check \leftarrow$ **false**, For all ex in $Exits$, set $Ennr(ex) \leftarrow 0$
2: $Check \leftarrow length(Exits) = length(Entries) = n$ // Check junction order
3: **for all** en in $Entries$ **do**
4: $Check \leftarrow (OutDegree(en) + InDegree(en) \geq 2)$ // Check minimality II
5: $exnr \leftarrow 0$
6: **for all** ex in $Exits$ **do**
7: $Check \leftarrow (OutDegree(ex) + InDegree(ex) \geq 2)$ // Check minimality II
8: **if** $Reaches(en).[0].Contains(ex)$ **then**
9: $exnr{+}{+}$ and $Ennr(ex){+}{+}$
10: $Check \leftarrow (en \neq ex)$ // Check discreteness
11: $Check \leftarrow exnr \geq (n-1)$ // Check total reachability for entries
12: **for all** ex in $Exits$ **do**
13: $Check \leftarrow Ennr(ex) \geq (n-1)$ // Check total reachability for exits
14: **return** $Check$ // If false, test has failed

would be $O(|U|^m)$ if we naively iterated through all possible m-fold one-element selections from $|U|$. If we furthermore rule out 'same element' and 'same set' selection sequences, we arrive at algorithms for selecting *'m out of U'*. This can be solved even more efficiently, namely in $O(\binom{|U|}{m})$, using e.g. *lexicographically ordered m-combinations* from $|U|$ numbers. In the following, we assume that we have such an algorithm available for integer sets[6]: Function $SelectC(u, m)$ (for "select combination") returns integer lists consisting of m lexicographically ordered members of the integer set $\{0, ..., u-1\}$ in a lexicographically ordered sequence. For example, if $u = 3$ and $m = 2$, the first call $SelectC(3, 2)$ would produce $[0, 1]$, and then iteratively $[0, 2]$ and$[1, 2]$, to produce exactly the $\binom{3}{2} = 3$ different combinations of 2 out of 3 numbers.

The main question is then how to constrain the target set in size. We do not know the size of a target junction a-priori. But since Definition 1 of an *n-way junction* is exclusively based on the set of n entries and n exits of a subgraph, we can reformulate the problem: Search for a fixed sized set of $m = n + n$ vertices in a set of entry/exit candidates of F of the junction $F(U, O)$, such that they are totally reachable, discrete and minimal II in the sense of Definition 1, but do not coincide with the original entries and exits in F.

We can meaningfully restrict the set E of *entry/exit candidates* in F to only those vertices of U that are either actual exits/entries of F, or have a *degree of at least 3*. This is because F-internal vertices with degree 2 can never satisfy the *discreteness property*, as they are bound to lose at least one edge in order to become an entry or exit. We could then use $SelectC(|E|^2, 2n)$ to enumerate all $2n$-subsets in time $O(\binom{|E|^2}{2n})$. But by incorporating the first requirement of total reachability, we can further simplify the problem. We can construct a *bipartite*

[6] See e.g. the following implementation:
`http://msdn.microsoft.com/de-de/magazine/cc163957(en-us).aspx`

graph B like in Figure 7 between E and itself, E'. Then we have a function for mapping the same vertices in both sets, $Refl : E \to E'$.

Now suppose we construct the bipartite graph B such that there is an edge from vertex $v \in E$ to vertex $v' \in E'$ if and only if:

1. v is not an exit, and v' not an entry, and there is a path from v to v' in F,
2. and $Refl(v) \neq v'$, so same vertices are not connected by default[7].

Then our problem can be reformulated as finding a certain *bipartite subgraph* K_{n-1} *in* B satisfying further properties, namely a subgraph in which n vertices from E are connected to at least $n-1$ vertices from E', and n vertices from E' are connected to at least $n-1$ vertices from E (compare Figure 7, right).

Algorithm 6 JUNCTION-MINIMALITY(n): Check for minimality I

Require: J-CHECK(n)= **true** has been executed successfully
Require: Entry/exit candidates E have been collected as described in the text
1: $Check \leftarrow$ **true**; $OrEn \leftarrow Entries$; $OrEx \leftarrow Exits$ // Store original entries/exits
2: ConstructGrid($X \leftarrow E, Y \leftarrow E$)// Construct a grid (matrix) from candidates E of junction vertices U as described in the text, with X (columns) denoting exit- and Y (rows) entry-candidates, and with value 1 for each reachable entry/exit pair, and 0 otherwise.
3: **for all** $i = 0; i < |E|; i{+}{+}$ **do** // for all columns i in the grid
4: **for all** $h = 0; h < \binom{|E|}{n}; h{+}{+}$ **do** // for all entry-subsets h of length n
5: Generate subcolumn $C_{i,h}$ of length n from grid column $C_i = \left[(x_i, y_0), ..., (x_i, y_{|E|-1})\right]$ using index list $list_h \leftarrow SelectC(|E|, n)$, and if $C_{i,h}$ contains *at most one* 0, add it to a list L. Otherwise, do nothing.
6: Order L lexicographically using index h of each column. Now columns with identical n entries plus columns sharing subsets of size $(n-1)$ are exactly adjacent.
7: **for all** $h = 0; h < \binom{|E|}{n}; h{+}{+}$ **do** // for all entry-subsets h of length n
8: $m \leftarrow$ the number of columns in L with identical index h
9: **for all** $k = 0; k < \binom{m}{n}; k{+}{+}$ **do** // for all $n \times n$ subgrids of $m \times n$ grid h in L
10: Select subgrid k using indices $SelectC(m, n)$
11: **if** (every row of k has *at most one* 0) and (not for all x_i, y_j in k, $OrEx.Contains(x_i)$ and $OrEn.Contains(y_j)$) **then**
12: For all pairs (x_i, y_j) with value 1 in subgrid k, add all paths $(Reaches(y_j).[1].[p]).append(y_j)$ having $Reaches(y_j).[0].[p] = x_i$ to a list of vertices U_k.
13: J-PREPARE(U_k) // Update the entry and exit lists of the subgraph
14: **if** J-CHECK(n) = **true then**
15: $Check \leftarrow$ **false** and BREAK
16: **return** $Check$ // If false, test has failed

This subgraph will automatically satisfy the total reachability property among entry/exit candidates. Once we found such a K_{n-1}, we can reconstruct a corresponding full induced F-subgraph $K(U_k, O_k)$ by adding a path to its set of

[7] This is because we want to rule out non-discrete candidates a-priori.

vertices U_k for each K_{n-1}-edge, drawing on the already stored paths between the candidates in F[8]. As was shown in [7], the very similar problem of counting all 'complete' $K_{n,n}$ subgraphs of a bipartite graph (in our case an $|E| \times |E|$-graph) can be solved in roughly $O(|E|\binom{|E|}{n})$ by transforming B into a plane grid of points with coordinates $X = E'$ (exits) and $Y = E$ (entries), as in Figure 7, middle, and by collecting and ordering all subcolumns of length n in this grid. Finding a K_{n-1} subgrid requires additionally to collect all subcolumns having $(n-1)$ entries in common with any given entry subset h of length n, and to check whether all rows of an $n \times n$ subgrid with entry subset h have at most one missing exit. This has a slightly larger complexity of $O(|E|\binom{|E|}{n}n)$, since for every entry subset of size n, we have to enumerate n subsets of size $n-1$. Algorithm 6 contains the pseudocode for a procedure that enumerates all such subgrids, reconstructs the corresponding channel subgraphs from paths, and checks them for junction properties.

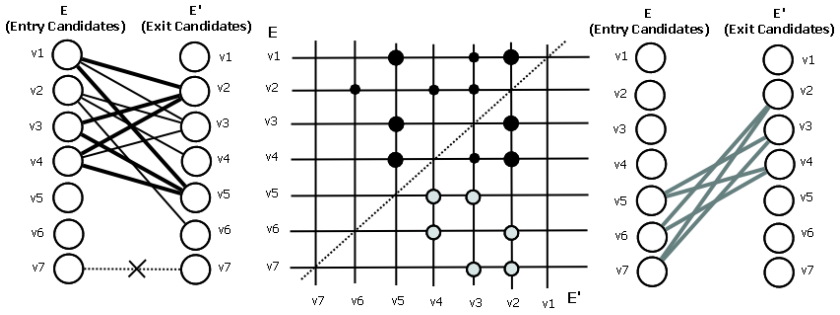

Fig. 7. Bipartite graphs and their grid representations. A $K_{2,3}$ complete subgraph (thick edges, left), corresponds to a subgrid (thick points, middle). A K_{3-1} subgraph (right) corresponds to an incomplete subgrid with a missing diagonal (grey points, middle).

5 Evaluation and Conclusion

We tested the algorithm in Section 4 on a set of 14 junction candidate subgraphs of a generated OSM-channel digraph for the city of Münster. Since the junction tag in OSM is very infrequent and unreliable, we selected 7 positive and 7 negative example subgraphs by hand, sometimes adding missing turn off restrictions.

[8] Note that entry/exit *candidates* do not necessarily correspond to *true entries and exits* in the corresponding induced subgraph. Still, the existence of a set of totally reachable candidates in this subgraph is a necessary property, which is used here to speed up search.

The *positive examples* were generated from junctions of different type[9]. We then added adjacent segments to generate the *negative examples*, for example complete roads satisfying all junction properties except minimality I. By the time of submitting this paper, we could produce good results on these examples, but are still working on the implementation and further tests. Despite this laborious kind of evaluation, the authors hope to provide a free reliable implementation online very soon, which could be further improved and tested inside of the OSM community.

We proposed algorithms for categorization of road network data. We showed how such data, e.g. OSM, map into the channel network theory which is grounded in affordances. We elaborated an efficient algorithm for checking wether a given subgraph of a channel network is a an n-way junction, n being the number of entries/exits. The junction test also involves an efficient solution for finding junctions in arbitrary networks. This supports the conviction of the authors that grounding data in affordances is fruitful not only for communicating its meaning, but also for designing computational solutions.

Acknowledgements

This work is funded by the Semantic Reference Systems II project (DFG KU 1368/4-2).

References

1. Scheider, S., Schulz, D.: Specifying essential features of street networks. In Winter, S., Duckham, M., Kulik, L., Kuipers, B., eds.: Spatial information theory: 8th Int. Conf., COSIT 2007; Proc. Springer, Berlin (2007) 19–23
2. Scheider, S., Kuhn, W.: Affordance-based categorization of road network data using a grounded theory of channel networks. International Journal of Geographical Information Science (in Press)
3. Gibson, J.: The ecological approach to visual perception. Houghton Mifflin, Boston (1979)
4. Scheider, S., Janowicz, K., Kuhn, W.: Grounding geographic categories in the meaningful environment. In K.S. Hornsby, C. Claramunt, M.D., Ligozat, G., eds.: Spatial Information Theory, 9th Int. Conf., COSIT 2009; Proc. Springer, Berlin (2009) 69–87
5. Gangemi, A., Guarino, N., Masolo, C., Oltramari, A.: Understanding top-level ontological distinctions. In Stuckenschmidt, H., ed.: Proc. IJCAI 2001 Workshop on Ontologies and Information Sharing. (2001)
6. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM Journal on Computing **1** (1972) 146–160
7. Berg, M.d., Overmars, M., Kreveld, M.v.: Finding complete bipartite subgraphs in bipartite graphs. Technical Report RUU-CS-89-30, Utrecht University (Netherlands) (1989)

[9] Amongst others a cloverleaf (4-way) ("Kreuz Münster-Nord"), a signalized T (3-way) ("Kolde Ring") and a diamond interchange (4-way) ("Albersloher Weg/B51").