JOURNAL OF SPATIAL INFORMATION SCIENCE
Number 4 (2012), pp. 31–56
doi:10.5311/JOSIS.2012.4.54

RESEARCH ARTICLE

Affordance-based individuation of junctions in Open Street Map

Simon Scheider and Jörg Possin

Institute for Geoinformatics, University of Münster, Münster, Germany

Received: April 7, 2011; returned: June 22, 2011; revised: October 11, 2011; accepted: January 2, 2012.

Abstract: We propose an algorithm that can be used to identify automatically the subset of street segments of a road network map that corresponds to a junction. The main idea is to use turn-compliant locomotion affordances, i.e., restricted patterns of supported movement, in order to specify junctions independently of their data representation, and in order to motivate tractable individuation and classification strategies. We argue that common approaches based solely on geometry or topology of the street segment graph are useful but insufficient proxies. They miss certain turn restrictions essential to junctions. From a computational viewpoint, the main challenge of affordance-based individuation of junctions lies in its complex recursive definition. In this paper, we show how Open Street Map data can be interpreted into locomotion affordances, and how the recursive junction definition can be translated into a deterministic algorithm. We evaluate this algorithm by applying it to small map excerpts in order to delineate the contained junctions.

Keywords: junctions, intersections, individuation, road networks, affordances, navigation, turn restrictions

1 Introduction and motivation

Road network maps, consisting of vector representations of roads, junctions, and points of interest (POIs), are among the most widely used sorts of geodata. They enable *navigation* systems, but also allow for *georeferencing* of diverse kinds of information, ranging from traffic frequency [13] to information about public places of interest. Road network maps furthermore enable map based *orientation* of users in their environment, since roads as well as junctions can be used as landmarks in certain cases [9]. This pervasive relevance may be the reason why Open Street Map (OSM)[1], the world's biggest open user-generated road

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

network data set, is currently considered as a general spatial semantic hub for interlinking the web of data [20].

The usefulness of road network data in order to represent and guide navigation thereby depends on the following factors:

- the *quality* and *completeness* of vector geometry data, labels, and relations; and
- the possibility to *generalize* information to different levels of detail.

The first claim seems beyond question. There are at least two arguments for the second claim, one concerning performance and the other one concerning modeling of navigation. Regarding navigation performance, we often need to represent junctions as generalized objects (e.g., points) in order to use them as landmarks for orientation [10], or to schedule car navigation over large-scale distances [21]. At the same time, navigation systems need detailed representations of highway intersections in order to guide drivers through the junction's complicated sub-network. All this requires identification routines for junctions at different levels of detail. Furthermore, if one wants to statistically model traffic over the network [12], it is often necessary to take exemplars of generalized categories like roads and junctions into account. For instance, to identify street segments as parts of a larger junction is crucial in order to prevent a bias in traffic estimation [12]. The bias occurs if one tries to estimate traffic on a highway segment based on observed traffic on a highway intersection segment, e.g., a highway ramp. The latter needs to be distinguished from the former because of reduced traffic on ramps [19]. Similarly, dual carriageways are roads frequently represented in terms of parallel segments for each lane (compare Figure 2). If lane segments are confused with single road segments, then they cause a similar estimation bias [12].

Thus, there is a need to identify roads and junctions as semantic categories across different road network data sets as well as across different levels of detail. We consider our present work as a step towards this more general goal. On a high level of generalization, category instances may be represented as single vector data elements (e.g., single points for junctions, or single segments for roads). On a low generalization level, they are usually encoded as labels or relations among more detailed vector data elements. However, a persistent problem is the heterogeneity and lack of availability of category information. The authoritative standards for road network labeling (e.g., the GDF standard, ISO 14.825:2004) do not assure availability and comparability of labels [19].

In order to approach this problem independently of a certain kind of data representation, road networks can be considered in terms of data semantics [17]. From an ontological point of view [5], one can ask how to *individuate* and how to *categorize* road network junctions in general. Categorization means to determine the type of junction, e.g., the number of roads that intersect at it. Individuation means to identify the parts of a junction and, thus, to distinguish individual junctions from others. For road network maps such as OSM, which frequently do not have junction labels, these problems are challenging. Consider, for example, the motorway junction "Wuppertal Nord" in Figure 1. How many roads intersect at this junction? Where exactly does the junction start and where does it end? Which segments or nodes belong to it? For example, is the pink road segment which leads to L551 still part of the junction? Is this question answerable at all in terms of the usual topological or geometrical considerations? It seems inappropriate to identify this junction either in terms of a node with corresponding cardinality, since the junction is not a node, or in terms of a certain geometrical shape, since there is a large variety of junction shapes.

Figure 1: The difficulty of individuating junctions, illustrated by the motorway junction "Wuppertal Nord" (Source: OSM).

At first glance, it seems there is no definite answer to this question. However, in [18], we have proposed a semantic theory of *channel networks* which is grounded in observable *locomotion affordances*. An affordance is an action potential in the perceived environment [6]. We proposed a definite answer in terms of an affordance-based definition of an *n-way junction*. This definition is not based on segment geometry or topology, but on observed affordances. It is therefore independent of a specific level of representation, while capturing the intended meaning in terms of possible navigation manoeuvre. We showed that our junction definition is satisfied by a collection of the most common junction types [18]. It provides a general way of specifying the categories needed [17]. However, our junction definition is quite complex and, in particular, recursive, because of minimality criteria. Therefore, it does not translate easily into a useful algorithm.

In this paper, we develop and evaluate an algorithm to *individuate* such complex road network junctions in detailed road network maps, such as OSM, Teleatlas, or Navteq. The challenge is to translate the first order definition of [18] into an algorithm that terminates in a reasonable amount of time, and that can be applied to small map excerpts of OSM in order to delineate the contained junctions. Searching for junctions in *large datasets*, e.g., in the whole of OSM, may be based on this algorithm. However, the algorithm requires further improvements in efficiency, in particular regarding large-scale search strategies, which are not in focus in this paper.

In the remainder, we will first argue why and how far the question of junction individuation is still an open one (Section 2). We will then introduce channel networks, and explain how OSM data maps into the channel network theory proposed in [18] (Section 3). Afterwards, we will recap our junction definition from [18]. We will then discuss an algorithmic approach to the problem in Section 4, before we describe an implementation for OSM in Section 5, and evaluate it in Section 6.

2 Individuating road intersections semi-automatically

The necessity of representing road networks at different levels of granularity has been recognized early in geographic information science [21]. However, the question of how to extract and generalize junctions is a research challenge up to the present day.

A naive approach to the problem would be to consider all nodes with a certain segment cardinality in the street segment graph as a junction. However, as we have already argued, this procedure does not help to identify complex junctions such as in Figure 1. Furthermore this procedure could include erroneous cases, e.g., it may consider a bifurcating road as a 3-way junction.

More sophisticated approaches use complex shapes or the topology of the street segment graph. Most attempts are based on statistical knowledge extraction using *geometrical patterns*. For example, extraction of roads and intersections is a common problem in remote sensing [1, 8], and is solved based on pixel configurations. Road networks [24] and road categories [7], such as circular roads, can be detected by geometrical patterns in a street segment graph. For example, Mackaness et al. [11] identify junctions by spatial clustering of street nodes. This method is simple and fast. However, it relies on a context-dependent granularity parameter and cannot filter out unwanted features, such as small settlements. More recent approaches try to extract the information from GPS tracks [3].

Doubtlessly, geometry and topology of street segments can be a valuable source of information for describing categories [7, 19], especially in order to speed up search. However, as we have argued in [19] and [17], the geometry or topology of an embedded segment graph is not sufficient to capture all road network categories. The reason is that junctions and roads are constituted by their functions, i.e., by locomotion affordances, but these are only partially represented in terms of connected street segments. The main point is that a segment graph misses out *turn restrictions*.

To illustrate, consider the generalization of a dual carriageway and a roundabout in Figure 2 (example taken from [19]). The former is a road generalizable into a segment; the latter is a junction generalizable into a node. The vertical arrow indicates the direction of generalization. Black arrows next to segments indicate driving directions. Note that even though in our figures we depicted right-hand traffic, all our arguments in this paper are independent of the main driving side. We may strip away nodes o and q (selection) in the dual carriageway and amalgamate nodes p and r to generate a single segment j. In the roundabout, we may amalgamate nodes o, q, p, m, n, and r to a single node u. But on what grounds? Obviously, there is no topological difference between the two subgraphs that would allow us to distinguish between these two cases, and, thus, between junctions and roads.

We suggest that the semantic difference between the two categories is not captured by segment topology, but rather by turn restrictions. For in the dual carriageway, it is not possible to turn from segment a into f crossing node m, while it is certainly possible to drive from a to f in the roundabout. Note that this information is simply not contained in the segment graph. It rather requires an additional turn off relation on top of it, as was proposed already by Winter in [23].

The difference between a roundabout and a dual carriageway may also be based on its elongated shape. However, this is an unreliable ad-hoc rule. For in the case of complex junctions, we have to deal with a great variety of shapes and can have very many turn restrictions (for examples, see Section 6.1). Our argument is that an affordance-based ap-

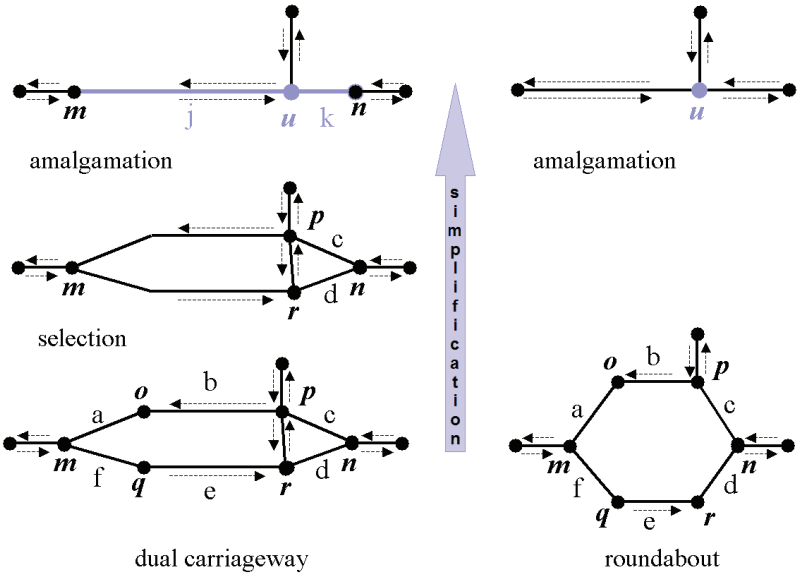


Figure 2: Generalization of two embedded street segment graphs, one being a common data model of a dual carriageway (a road), the other being a common data model of a roundabout (a junction) [19].

proach to individuation is semantically prior to topology and shape, and thus more reliable, while topology and shape are rather indirect proxies. Therefore, the cost of basing individuation on affordances corresponds to a gain in correctness and completeness, provided that the information about affordances is somehow contained in the data.

3 Channel networks and junctions

Roads are parts of a medium for *supported locomotion*, i.e., they allow movements supported by a flat surface. But roads additionally exhibit conventional turn restrictions, observable, e.g., by road signs or dotted versus full lines on a road surface. Turn restrictions are the main instrument for traffic management, because they restrict the supported movements to a regulated more or less homogeneous flow into discrete directions. A road is thus a case of a *physically manifested social affordance* [22], because it presupposes the interpretation of symbols on the road surface.

We defined a *channel network* in [16, 18] as a medium for *turn-compliant supported motion*. 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 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 3). Channels do not overlap each other and are not self-connected at their portals. They exist in finite amounts and can be perceived based on traffic conventions: channels are discrete

parts of the environment in which supported movements are restricted *by convention*, for example, by perceivable signs on the road surface.

If two channels are connected such that one can move from one to the other via appropriate portals, this is expressed by the irreflexive affordance relation *LeadsTo*(x, y) [16, 18], whose domain and range consists of channels in the environment of the observer. The transitive closure of this relation, which was called *ReachableFrom* in [16, 18], denotes those paths of channels that afford turn-compliant locomotion.

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 [18]. 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 can be described by a directed graph $D(V, A)$ [18].

3.1 Interpreting channel networks in terms of OSM data

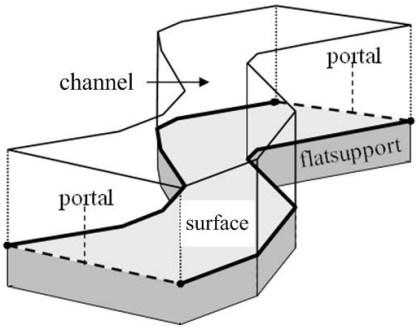


Figure 3: Illustration of a channel [19].

Every navigation-oriented road network data model is based on an *embedded graph of street segments*. In this data model, nodes and undirected edges (segments) are (non-planarly) embedded into the Euclidean plane. However, a channel network is not the same as a street segment graph.

Channels can be embedded into the plane in terms of street segments whose end nodes indicate some kind of intersection (Figure 4, right). But channels are not identical with these segments, since channels may correspond to the "side lanes" of a bidirectional segment, as in Figure 4(2), as well as to a single street segment in the case of one way streets, shown in Figure 4(1).

Furthermore, as we have argued in Section 2, the data model of a channel network is more complex than an embedded street segment graph. It also requires a relation that accounts for *LeadsTo*, i.e., for turn-compliant navigation among channels. In Figure 4, we have depicted example channel configurations (left) together with their equivalent road-network data models (right). In the data model subfigures, the *LeadsTo* relation is depicted by dotted arrows; channels are represented as full arrows; and the street segments are represented as a separate undirected embedded graph with lines and nodes. As one can see, *LeadsTo* is not captured by the segment graph alone, since the two branching edges of a

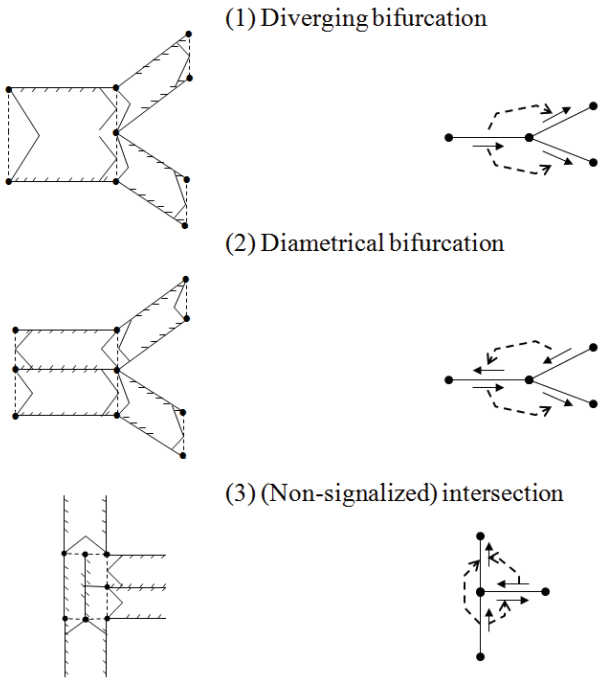


Figure 4: Channel configurations (left) corresponding to road network data elements (right) [19].

diametrical bifurcation, as in Figure 4(2), are connected at a common node, but do not lead to one another.

For our purpose, the street segment graph of OSM can be simplified into a set of *intersection-free navigable way segments*. We propose the following rules to reconstruct channels and the *LeadsTo* relation from available tags and relations:

1. Dissect or merge all navigable ways (tagged as "highway") into maximal intersection-free way segments with all inner nodes of degree ≤ 2.
2. For every bidirectional way segment, construct 2 channels c_1, c_2, one *towards* and one *from* an incident node. These channels *may not lead into one another*, but are always *support-connected SupportC*(c_1, c_2). We call c_1, c_2 *adjacent* in the following.
3. For every one-way segment, construct one channel. For all dead end segments, construct 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 point 2) and not subject to any turn restrictions[2] are also connected by *LeadsTo*.

[2]In comparison to commercial navigation networks, turn 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.

3.2 Junctions and what they afford

Following [18], a junction can be conceived as an induced subgraph $F(U, O)$ of a channel digraph $D(V, A)$ with a special kind of affordance. We already said that any part of a road network reflects social locomotion affordances. But over and above this, junctions essentially afford a certain human choice among different locomotion paths. This was called *mental affordance* by Raubal [15]. The set of paths to choose from arises from reachability of some vertices, called *entries*, *exits*, and *gates*. 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 in U that are either entries or exits (but not both), is called *gates of* F [18].

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* (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 explain and motivate these properties in the remainder by analyzing a *median U-turn junction* (see Figure 5). 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 5).

In Figure 5, the *LeadsTo* relation is indicated by gray curved arrows that lead from a channel to a successor. Exits and entries of the subgraph are numbered anticlockwise. In the corresponding channel digraph in Figure 6a, channels are depicted as vertices and the *LeadsTo* relation as arcs. Exits and entries have equivalent numbers.

The first and most obvious property is that junctions must 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, the driver 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 6b, 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. If people are speaking of "staying on or leaving a junction," they mean discrete actions: the process of entering, staying on, or leaving a junction is unambiguous for a moving object inside of a channel. The subgraph in Figure 6c (in black) is a part of the median U-turn that satisfies the total reachability assumption. But because neighboring exits and entries coincide, 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 still satisfies the other properties. In a first sense, a junction is minimal because it *never contains a smaller junction*.

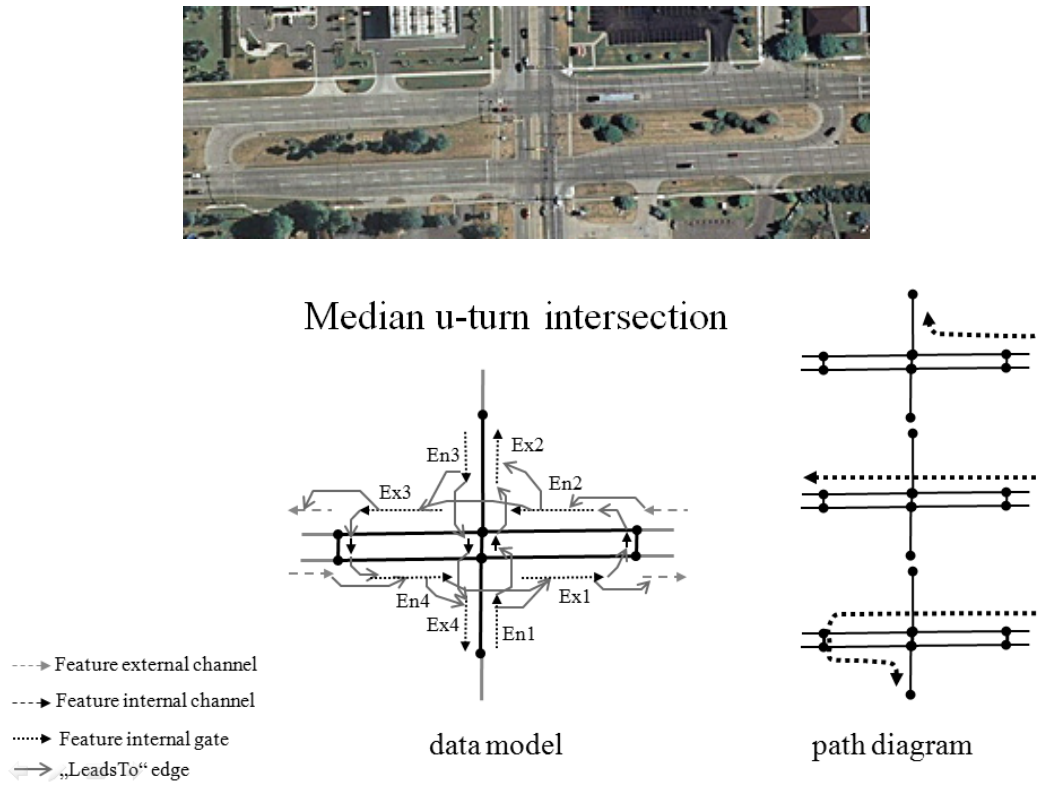


Figure 5: A median U-turn ("Michigan left") intersection, including an example from Arizona (top) and its associated segment data model (bottom).

In a second sense, we require a *minimal vertex degree* for entries and exits. For details, compare the discussion in [18].

4 Algorithm for individuating junctions

In this section, we develop an algorithm that makes affordance-based Definition 1 computable, i.e., that can be used to identify junctions as subgraphs in a road network, such as OSM. The pseudocode in this section makes use of some well known operations on common data types. A universal super type is denoted by letter T. T^* is the type of an array of T instances. Well-known data types are denoted by their usual names. All operations and constants are explained in the text. The most important ones are listed in Table 1. A synopsis of the entire procedure will be given in Section 5.

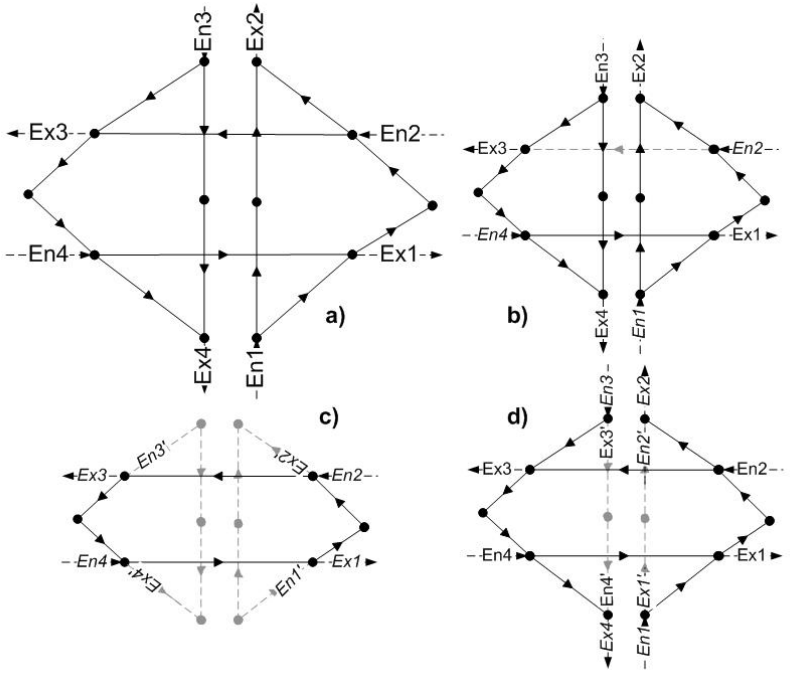


Figure 6: A median U-turn channel subgraph.

4.1 Checking junction properties by DFS back-propagation

In order to inspect the total reachability property of Definition 1, we have to collect all paths from all entries of a subgraph F to all of its exits. For the sake of simplicity, we collect all paths from all vertices to all other vertices using a time-efficient solution based on *depth-first search* (DFS) [4]. If there is a problem with storage space, then this algorithm could be slightly modified such that only paths between entry/exit candidates (see definition in Section 4.2) are collected.

Suppose that a channel network $D(V, A)$, a directed graph, is given by a domain of vertices V and the functions *Succ* and *Pred* denoting the arcs. We intend to check whether a given subset of vertices U of induced subgraph $F(U, O)$ is a junction or not. 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 gray while moving to unvisited (white) vertices; stops if vertex a 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 the list *Cycleedges*.

Operations	Type	Explanation
	V	Vertices (channels) of channel digraph D
	U	Vertex subset of subgraph F in V
$Succ(), Pred()$	$V \to V$	Returns the successor (predecessor) vertex in D
$Subgraph()$	$V \to Bool$	Flag for vertices U
$Entries, Exits$	$List$	List for entries (exits) of F
$Forwardedge()$	$V \times V$	DFS spanning tree edges
$Cycleedges()$	$V \times V$	DFS spanning tree back edges
$Reaches().vertices$	$V \to List$	Returns a vertex list reachable from a vertex
$Reaches().paths$	$V \to List$	Returns a path list reachable from a vertex
$Visit()$	$V \to w\|b\|g$	Color flag for vertices visited in DFS
$Out(In)degree()$	$V \to \mathbb{N}$	Returns the out(in)-degree of a vertex
$SelectC()$	$\mathbb{N} \times \mathbb{N} \to List$	$SelectC(n, m)$ Returns m-combinations out of n
[ ]	$T^* \to List$	Takes array $a, ..., b$ and returns list $[a, ..., b]$
. [ ]	$List \times \mathbb{N} \to T$	Returns a list element, e.g., $[a, b].[0] = a$
.append	$List \times T \to List$	Appends elements at the end of a list, e.g., $[a].append(b) = [a, b]$
$length()$	$List \to \mathbb{N}$	Returns the length of a list, e.g., $length([a]) = 1$
.Contains	$List \times T \to Bool$	Tests whether a list contains an element, e.g., $[a].Contains(a)$
[]	$List$	The empty list

Table 1: Inventory of important types, pseudocode operations, and constants.

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**
end for
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}
    end if
  end while
  while $Succ(v) = v'$ **do**
    if $Subgraph(v') =$ **false then**
      $Exits \leftarrow Exits.append(v)$
    else
      $Outdegree(v)$++ {Countup internal out-degree of v}
    end if
  end while
end for

The essential idea in Algorithm 2 is to use the DFS-principle of *back-propagation* 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 gray), then each vertex v in the tree has a list $\mathit{Reaches}(v).\mathit{vertices}$ of all vertices *reachable in this tree* (as well as a list of corresponding paths $\mathit{Reaches}(v).\mathit{paths}$). The 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.

Algorithm 2 TR-DFS(v): Total reachability depth-first search.

Require: $\mathit{Visit}(v) = \mathit{white}$; $\mathit{Subgraph}(v) =$ **true**;
1: $\mathit{Visit}(v) \leftarrow \mathit{gray}$ {Color vertex as visited but not yet backtracked}
2: $\mathit{Reaches}(v).\mathit{vertices} \leftarrow \mathit{Reaches}(v).\mathit{vertices}.\mathit{append}(v)$ {Register new reachable vertex locally}
3: **while** $\mathit{Succ}(v) = v'$ **do**
4: **if** $\mathit{Subgraph}(v') =$ **true then**
5: **if** $\mathit{Visit}(v') = \mathit{white}$ **then**
6: $\mathit{Forwardedge}(v, v') \leftarrow$ **true** {Mark edge as forward}
7: TR-DFS(v') {Call recursion}
8: **else if** $\mathit{Visit}(v') = \mathit{gray}$ **then** {Collect back-edges to update cycles}
9: $\mathit{Cycleedges} \leftarrow \mathit{Cycleedges}.\mathit{append}(v, v')$
10: **end if**
11: **for all** $i = 0; i < \mathit{length}(\mathit{Reaches}(v').\mathit{vertices}); i{+}{+}$ **do**
12: $\mathit{Reaches}(v).\mathit{vertices} \leftarrow \mathit{Reaches}(v).\mathit{vertices}.\mathit{append}(\mathit{Reaches}(v').\mathit{vertices}.[i])$ {Back-propagate reachable vertices}
13: $\mathit{Reaches}(v).\mathit{paths} \leftarrow \mathit{Reaches}(v).\mathit{paths}.\mathit{append}(\mathit{Reaches}(v').\mathit{paths}.[i].\mathit{append}(v'))$ {Back-propagate the paths to these vertices}
14: **end for**
15: **end if**
16: **end while**
17: $\mathit{Visit}(v) \leftarrow \mathit{black}$ {Color vertex as already backtracked}

But the subgraph F will frequently contain *circuits*. This complicates the situation, since the collected tree paths do not necessarily contain the paths to 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 backtracked into such a circuit (compare lines 10-12 in Algorithm 2). But if this vertex happens to be still in gray, then the reachability information is not there yet. We therefore have to do a separate back-propagation after the execution of 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 are visited. 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. Since the call to TR-DFS() is done only once for all visited vertices and edges, the time complexity is unchanged. 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 3 BACKP(y, z, z'): Independent back-propagation for cycles.

for all $i = 0; i <$ *length*(*Reaches*(z).*vertices*$); i$++ **do** {Back-propagation of paths}
  Reaches(y).*vertices* $\leftarrow$ *Reaches*(y).*vertices.append*(*Reaches*(z).*vertices*.$[i]$)
  Reaches(y).*paths* $\leftarrow$ *Reaches*(y).*paths.append*(*Reaches*(z).*paths*.$[i]$.*append*(z))
end for
while *Pred*$(y) = x$ **do**
  if *Forwardedge*(x, y) = **true** AND $z' \neq x$ **then**
    BACKP(x, y, z') {Recursion}
  end if
end while

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

Require: Flag *Subgraph* is set, and list *Entries* is filled
Set all vertices $v \in V$: *Visit*$(v) \leftarrow$ *white*; *Reaches*$(v) \leftarrow$ [[], [[]]]
For all edges $(x, y) \in A$ set *Forwardedge*$(x, y) \leftarrow$ **false**
Cycleedges $\leftarrow$ [] {Create empty lists}
for all v in *Entries* **do** {Call recursive TR-DFS() until exhaustion}
  if *Visit*(v) = *white* **then** {Start call only if entry was not visited yet}
    TR-DFS(v)
    for all (y, z) in *Cycleedges* **do** {Update cycles}
      BACKP(y, z, z)
      Delete (y, z) from *Cycleedges*
    end for
  end if
end for

4.2 Searching for junctions in an efficient way

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 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 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

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

Require: n is the user-specified 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).*vertices*.*Contains*(ex) **then**
9: $exnr$++ and *Ennr*(ex)++
10: **end if**
11: *Check* $\leftarrow (en \neq ex)$ {Check discreteness}
12: **end for**
13: *Check* $\leftarrow exnr \geq (n-1)$ {Check total reachability for entries}
14: **end for**
15: **for all** ex in *Exits* **do**
16: *Check* $\leftarrow$ *Ennr*$(ex) \geq (n-1)$ {Check total reachability for exits}
17: **end for**
18: **return** *Check* {If false, test has failed}

sets "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: 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 $2n$ 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 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.

For this purpose, we need to take into account reachability relations among vertices in E. We can construct a *bipartite graph* B as in Figure 7, mapping E to itself, E', where E are potential entries and E' potential exits. 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:

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
Check ← **true**; *OrEn* ← *Entries*; *OrEx* ← *Exits* {Store original entries/exits}
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.}

for all $i = 0; i < |E|; i$++ **do** {for all columns i in the grid}
  for all $h = 0; h < \binom{|E|}{n}; h$++ **do** {for all entry-subsets h of length n}
    Generate subcolumn $C_{i,h}$ of length n from grid column $C_i = [(x_i, y_0), ..., (x_i, y_{|E|-1})]$ 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.
  end for
end for
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.
for all $h = 0; h < \binom{|E|}{n}; h$++ **do** {for all entry-subsets h of length n}
  $m \leftarrow$ the number of columns in L with identical index h
  for all $k = 0; k < \binom{m}{n}; k$++ **do** {for all $n \times n$ subgrids of $m \times n$ grid h in L}
    Select subgrid k using indices *SelectC*(m, n)
    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**
      For all pairs (x_i, y_j) with value 1 in subgrid k, add all paths (*Reaches*(y_j).*paths*.[p]).*append*(y_j) having *Reaches*(y_j).*vertices*.[p] $= x_i$ to a list of vertices U_k.
      J-PREPARE(U_k) {Update the entry and exit lists of the subgraph}
      if J-CHECK(n) = **true then**
        Check ← **false** and BREAK
      end if
    end if
  end for
end for
return *Check* {If false, test has failed}

1. v is not an exit, and v' not an entry, and there is a path from v to v' in F; and
2. v is not the same vertex as v', i.e., same vertices are not connected by default.

This bipartite graph represents the reachability properties among entry/exit candidates in F, and can thus be used to restrict our search to subgraphs that satisfy total reachability. Our problem can then be reformulated as finding a certain bipartite subgraph K_{n-1} in B, 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).

The bipartite subgraph satisfies the total reachability property among entry/exit candidates, but does not yet correspond to a junction since the other properties need to checked

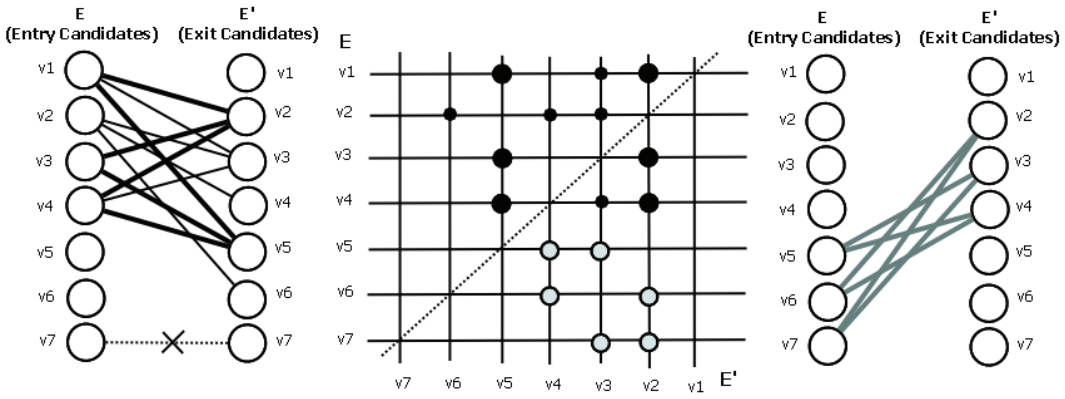


Figure 7: Bipartite graphs and their grid representations. A $K_{2,3}$ complete subgraph (thick edges, left), corresponds to a subgrid (thick points, middle). A K_{3-1} subgraph (right) corresponds to an incomplete subgrid with a missing diagonal (gray points, middle).

on the corresponding induced subgraph. Once we found such a K_{n-1}, we can easily reconstruct the corresponding F-subgraph $K(U_k, O_k)$ by adding vertices of a path to its set of vertices U_k for each K_{n-1}-edge, drawing on the already stored paths between the candidates in F. 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.

How can we search for K_{n-1} in an efficient way? As was shown in [2], the 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 lexicographically all subcolumns of length n in this grid. For example, in Figure 7, a $K_{2,3}$ subgrid with 2 subcolumns of length 3 is depicted by the thick points in the middle.

But in contrast to [2], we need to find a K_{n-1} subgrid with an empty diagonal instead, since we want to allow for one non-reachable exit per entry (compare Figure 7, middle, gray points). This corresponds to the impossibility of leaving the junction in the opposite direction of entry (compare Section 3.2). This is slightly more complex (compare Algorithm 6). In this case we need to collect all grid subcolumns having $(n-1)$ entries in common with any given entry subset $h \subset E$ of length n, order them lexicographically in list L using an index for h, select all $n \times n$ subgrids k for each subgrid with index h, and check whether all rows in k have at most one missing exit. This has a slightly larger complexity of $O(|E|\binom{|E|}{n}n)$, since for every entry subset h of size n, we have to enumerate n subcolumns 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.

The actual runtime is not pre-determinable since it depends on the size of E and the structure of B, which both depend on the connectivity of the channel subgraph F. While the algorithm is bound to have a polynomial runtime complexity, it is efficient only if F is sufficiently connected and thus $|E|$ increases slower than $|U|$. The size of E would increase

dramatically if we increased the number of unconnected components in F, since all their bordering vertices automatically become entry/exit candidates. However, this possibility can be excluded for junctions, since their subgraphs are always connected, and so this is in fact a comparatively efficient solution.

5 Implementation

In this section we present a plugin for the OSM editor JOSM that implements the algorithms described in Section 4. A general description of the plugin can be found in http://wiki.openstreetmap.org/wiki/JOSM/Plugins/JunctionChecking and [14]. JOSM is the most frequently used OSM editor. Our plugin enables the creation of a channel digraph from OSM data and searching and checking this digraph for junctions. We discuss its use by a junction example in Figure 8. The channels of the digraph are shown as red lines. In this diagram, a way segment that is drivable in both directions, and thus consists of two channels, is shown as red line without arrows. Single channels of one-way street segments are indicated by arrows.

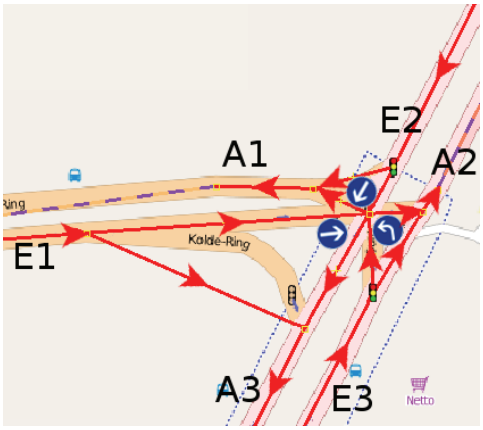
The data model of JOSM is like other popular GIS. JOSM can show different kinds of geographical data, for example GPS tracks or OSM data. Each data source has its own layer. A channel digraph can be generated from a layer that contains OSM data. If the plugin is loaded there is a new dialog window in the lower right part of the JOSM window. In this dialog window, the user can create a channel digraph, analyze junctions or search for junctions.

In order to create a channel digraph, an OSM layer must be made active by clicking on it. Before transforming the layer (the transformation rules are described in Section 3.1), the user can choose whether he or she wants the plugin to analyze the strongly connected components of the created channel digraph. This is recommended and is selected by default. Checking junctions in subgraphs that are not strongly connected results in errors due to artificial gates. After analysis, components that are not strongly connected are shown in a different color in JOSM and should not be used for further junction tests.

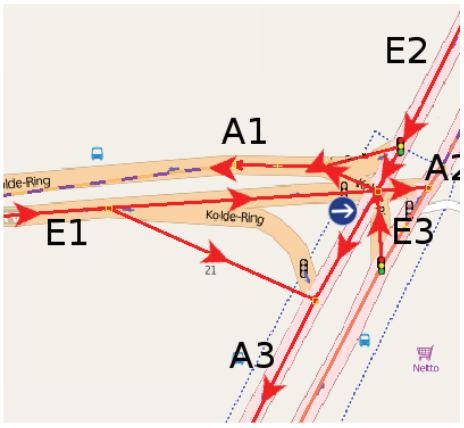
After clicking the channel digraph creation button, the channel digraph is shown as a separate layer in JOSM. In this layer, the user can mark channels for junction tests. For example, let us analyze the junction "Koldering/Weseler Strasse" in Münster, Germany. First, we mark the channel subgraph candidate by clicking or drawing a rectangle (see Figure 8a). Then we need to set the junction order n in the plugin dialog window. Since there are three possibilities to enter or leave the junction, it is a 3-way junction.

In order to perform a junction check, we need to push the "junction check" button. This invokes the procedure depicted in the flowchart of Figure 10. There are three possible results :

1. Positive: The marked channels represent a junction. In this case (and if a default option is chosen), an OSM relation will be generated that contains the junction and stores its order. The discovered junction will be displayed in another color.
2. Negative: The marked channels do not satisfy all of the junction criteria total reachability, discreteness of navigational action and minimality II (Section 3.2). In this case the user will be informed in an additional dialog window.
3. Negative: The marked channels satisfy the three junction criteria total reachability, discreteness of navigational action, or minimality II, but not the criteria minimality I.



(a) Marked channels at the beginning of the analysis. Blue turn off signs depict actually existent turn restrictions.



(b) Junction candidate found by the plugin. Blue turn off sign depicts those turn restrictions that were recorded in OSM.

Figure 8: Example for a junction check: Koldering in Münster, Germany. Red lines depict channels.

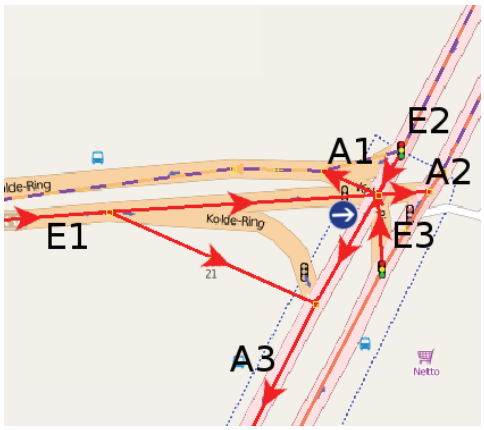


Figure 9: Smallest junction found by the plugin: Koldering in Münster, Germany.

This means the subgraph contains a smaller junction. The minimality I test stops as soon as it finds a subgraph within the original subgraph that satisfies the three other junction criteria. This subgraph is displayed in another color and represents another junction candidate for testing. To check this candidate, we have to perform another minimality test, as indicated by the loop around the minimality check in Figure 10.

In our example, the plugin discovered a smaller junction candidate (see Figure 8b) which needs to be tested again. This second run results in a new junction candidate (see Figure 9). After we checked this junction again, the result is positive and all criteria are satisfied. The junction found in this example will be evaluated in Section 6.1.

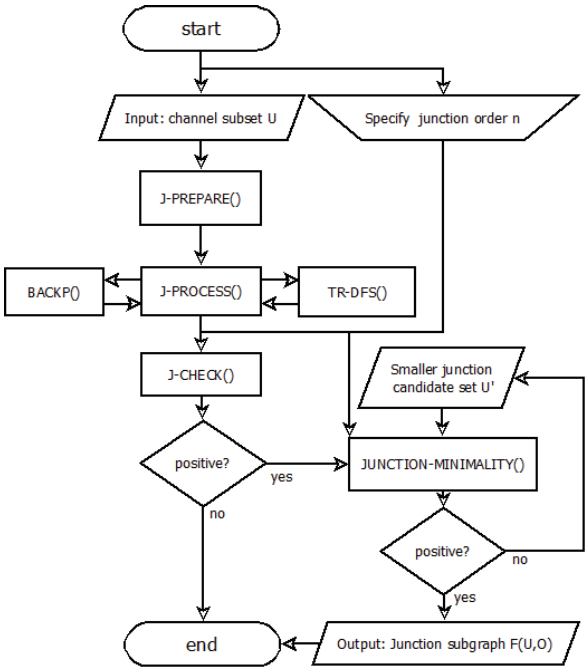


Figure 10: Flowchart of the junction check procedure. Process names correspond to algorithms of Section 4.

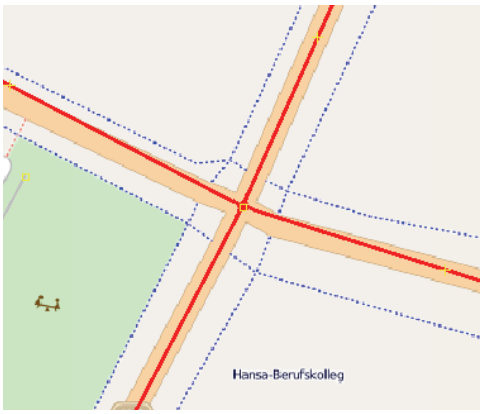
The junction search is just an exhaustive minimality check. The user marks an arbitrary subset of channels in the channel digraph layer and pushes the "junction search" button. The plugin then searches for junction candidates with the preset junction order (default value is $n = 3$). This is done by a slightly modified Algorithm 6: we do not stop after finding a junction candidate, but search for all of them and store them in one list. Afterwards, the candidates in the list are compared for mutual inclusion. If a candidate includes another one, it cannot be a junction because of criteria minimality I, and thus is removed from the list. This proceeds until only true junctions remain.

6 Evaluation

In this section, we will evaluate our approach in a twofold way. We will first analyze some example junctions found by the plugin and discuss their quality. Afterwards we analyze the computing time of searching in a subset of channels for junctions.

6.1 By examples from OSM

The first example is the minimal form of a 4-way junction. It consists of just eight channels: four entries and four exits, as for example the junction "Wolbecker Strasse /Hansaring" (see Figure 11a). The eight channels are connected by a single OSM node. The junction check is positive for this example.



(a) Example for a smallest 4-way junction: "Wolbecker Strasse/Hansaring" in Münster, Germany.

(b) Example for another 4-way junction: "Weseler Strasse/Inselbogen" in Münster, Germany.

Figure 11: Two examples for 4-way junctions.

A more complex successfully discovered example is the junction "Weseler Strasse /Inselbogen" (see Figure 11b). It consists of a way with separate lanes for both directions (Weseler Strasse) and a way with only one lane for both directions (Inselbogen). The street with two lanes is represented by two OSM ways which are tagged as "one-way." In accordance with our rules in Section 3.1, each one-way corresponds to one channel. This example is also a 4-way junction although there are two more channels in the subgraph. They are the result of the physical separation of the lanes of the Weseler Strasse.

The examples so far did not require any turn restrictions, since every topologically possible manoeuvre was actually allowed. But we already discussed a junction with turn restrictions in Section 5. In this junction, not every topologically possible turn is allowed (compare Figure 8a). But only one out of three existing turn restrictions were observed by OSM mappers and recorded in terms of an OSM relation (see Figure 8b). Therefore the junction could not be identified properly by the affordance based algorithms. In fact, the plugin erroneously detects a smaller junction due to fictitious turn possibilities. The existing turn restriction disallows movement from the Koldering entry E1 to exit A3. But in the discovered junction candidate, it is still possible to move from entry E3 (Weseler Strasse) to exit A2. In reality this would mean taking the left turn lane at first, but then suddenly return to Weseler Strasse via exit A2.

Therefore, the existing OSM dataset is incomplete and only partially suited in order to represent the actual affordance, as seen in Figure 8a. There seems to be a problem with the given OSM data quality in this respect: we discovered not one junction in the main city of Münster where all necessary turn restriction were recorded. The implications of this insight are discussed in Section 7.

At last, we will analyze motorway junctions and show that an even more complex structure can be mastered without problems if the affordance information is complete. Motorway junctions are complex but do not require turn restrictions since they are not signalized. Normally one can only turn off to one side of the road, and the only existing turn restriction results from the topology of the OSM channels: a restriction to drive into the end of one-

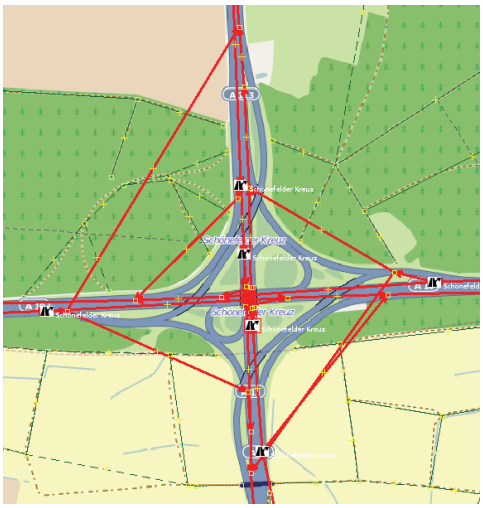


Figure 12: Example of a 4-way motorway junction: "Schönfelder Kreuz."

way streets need not be explicitly recorded. So we have no reason to expect errors in this case.

The first motorway example is the motorway junction "Schönfelder Kreuz" at Berlin, Germany (Figure 12). It is a 4-way junction that consists of 47 channels. The last example is a complex 5-way junction with 71 channels (see Figure 13). The motorway junction "Wuppertal Nord," which motivated our paper in Figure 1, consists of two motorways crossing one primary street. The primary street as well as the motorways have two lanes for each direction. Both junctions were correctly identified by the plugin.

6.2 Performance evaluation

In this section, we evaluate the computing time of the algorithm (compare Section 4.2). The plugin has a command-line interface to record statistics about the computing time of junction searches. The time complexity of junction search depends mainly on two methods in the Java source code: GenerateSubColumns() (rows 3–6 in Algorithm 6) and IterateThroughKN() (rows 7–16 in Algorithm 6). The first method searches for all subcolumns of length $n-1$ in the constructed grid (see Section 4.2). The second constructs corresponding totally reachable subgraphs and tests them for discreteness of navigational action and minimality II. We analyzed only junction search since it is the worst case of a junction check.

We evaluated time complexity by performing automated search in a constantly growing connected subset of channels. The search was started with five channels in a subset, thereafter with six channels, and so on, until a predetermined upper limit was reached. The candidate subgraphs were selected in the following manner: a seed channel was selected incidentally, and outgoing channels were added recursively until the maximum subset size was reached. In this case, the subgraphs are assured to be connected and hence the grid size could not increase dramatically, as argued in Section 4.2.

For each subgraph, the computing time for GenerateSubColumns() was measured in milliseconds. We performed multiple runs for a subset size and averaged computing time.

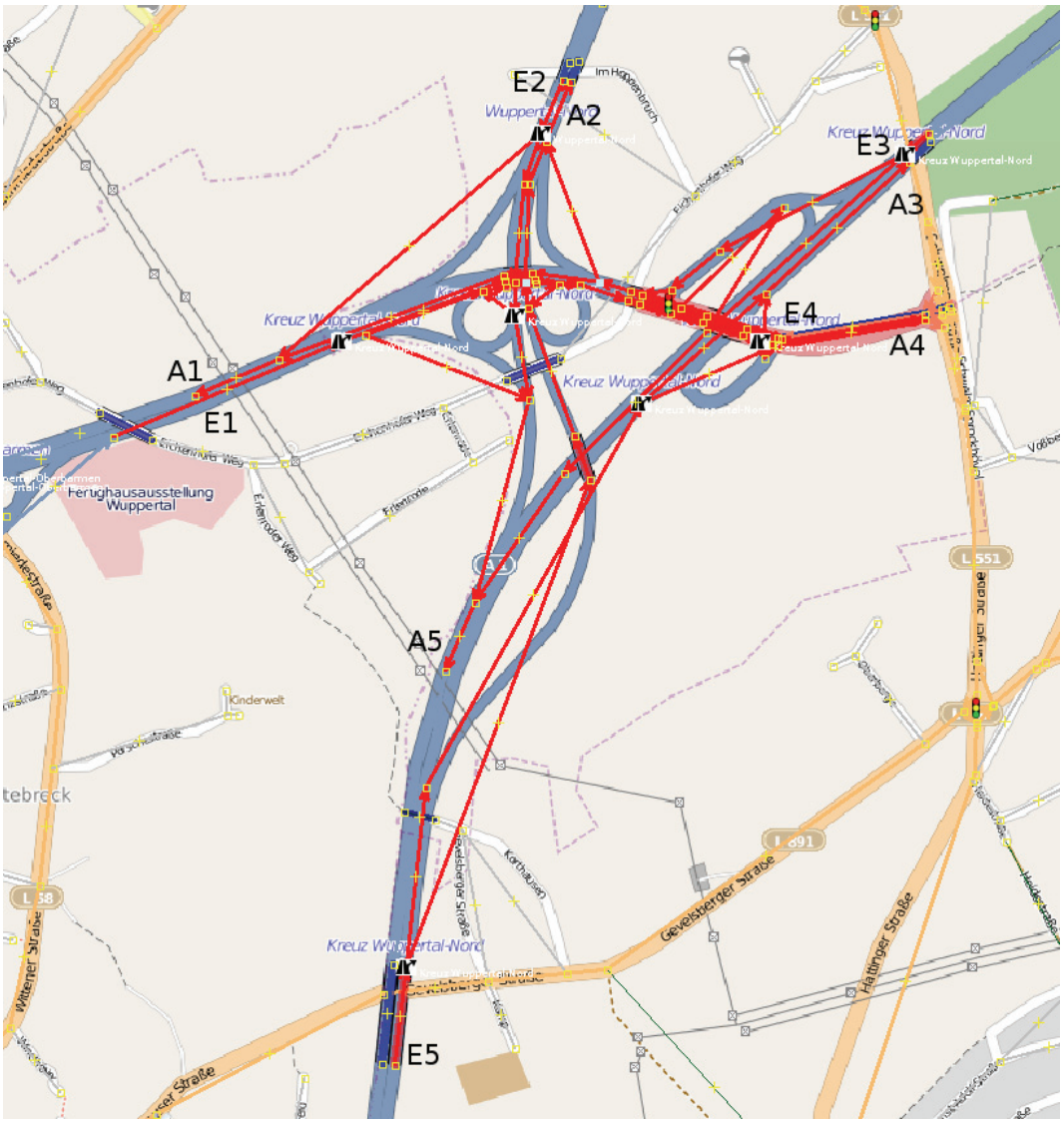
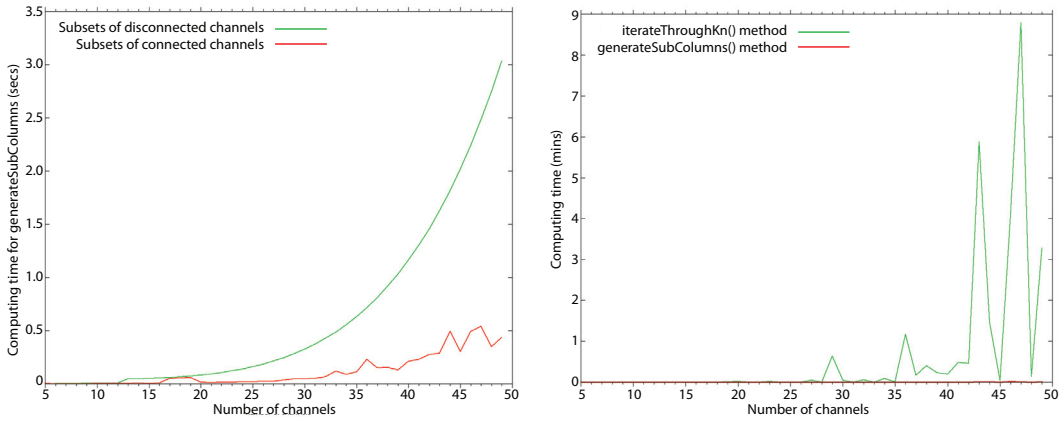


Figure 13: Example of a 5-way motorway junction: "Wuppertal Nord."

The performance of GenerateSubColumns() was tested in five runs up to a subset of 50 channels by averaging values. The result is shown in Figure 14a in terms of the red curve. Computing time grows moderately but does not exceed one second. This seems reasonably fast for practical purposes. For comparison, we have also plotted the performance for random channel selection (green curve), in which subgraph connectedness is not assured. This has inefficient cubic growth, but represents unrealistic conditions.

The computing time of the IterateThroughKN() method is depicted in Figure 14b in terms of the green curve. In this case, values were not averaged, but represent a typical



(a) Red curve shows average computing time for the GenerateSubColumns() method in seconds.

(b) Green curve shows typical computing time of IterateThroughKN() in minutes.

Figure 14: Computing time of junction search algorithm for different subgraph sizes.

result of a run. Expectation values evidently grow in connection with the growing number of channels, while the actual computing time often remains zero if no adequate channel configuration was found in the preceding method (see subgraph size 45 and 49). Inside the tested range of graph sizes, it can happen that the method needs several minutes to terminate.

In summary, we note that algorithm performs sufficiently fast for the intended purpose, which was to individuate junctions in a small connected subgraph of a channel network, e.g., in an OSM map excerpt in which one wants to delineate particular junctions. For search in larger datasets, however, further research effort is needed to increase efficiency.

7 Conclusion

In this paper, we treated the problem of how to individuate junctions in road network map excerpts, i.e., how to automatically identify them in terms of a street segment subgraph. We argued that current approaches ignore a crucial source of information (Section 2), namely information about turn-compliant locomotion affordances. The channel network theory is a semantic theory that directly represents these affordances [18]. In the paper, we showed how this theory can be interpreted into a road network database like OSM, and how a recursive definition of an n-way junction can be translated into an algorithm with testable correct results, and with acceptable runtime behavior for small map excerpts.

The algorithm requires a channel digraph to be reconstructed from the road network dataset. For this purpose, the information about turn restrictions must be available in it. In OSM, turn restrictions are in principle recorded, but are often incomplete. As a result, affordance-based search sometimes produced incorrect results in our evaluation. However, this problem disappears as soon as the algorithm is applied to a dataset of higher quality, such as Navteq or Teleatlas. This reveals that beyond cartographic displays, crowd-sourced navigation networks still seem to miss some of the quality required to take over from com-

mercial datasets. It also indicates that our affordance-based approach may be used as part of a data quality assurance process for vehicle navigation purposes.

Alternative approaches to junction search, based solely on the geometry of the street segment graph, may be simpler and faster since they do not dependent on affordance observations. However, they are not sufficient since they miss out essential turn restrictions that are not reflected in this graph. As soon as one takes the latter into account, one effectively needs to reason about affordances.

There are several ways in which this work can be extended. First of all, the general method should also be usable for roads. Regarding junctions, the definition in [18] still excludes certain types of them, e.g., roundabouts. Furthermore, the definition still allows for overlapping individuals, which may be unwanted. A more general search over different junction types, e.g., of different way cardinalities, seems desirable and feasible. Most importantly, however, the efficiency of search must be further improved in order to be applied to large datasets in a batch mode. One idea is to use geometric information to expedite search, and use affordance-based search only on appropriate geometric candidates.

Acknowledgments

This work was funded by the German Research Foundation (DFG) under the Semantic Reference Systems II project (DFG KU 1368/4-2), as well as by the European Commission (ICT-FP7-249120 ENVISION project). The authors would also like to thank the anonymous reviewers for their detailed comments.

References

[1] BARSI, A., AND HEIPKE, C. Detecting road junctions by artificial neural networks. In *2nd GRSS/ISPRS Joint Workshop on Remote Sensing and Data Fusion over Urban Areas. URBAN 2003* (2003), pp. 129–132.

[2] BERG, M. D., OVERMARS, M., AND KREVELD, M. V. Finding complete bipartite subgraphs in bipartite graphs. Tech. Rep. RUU-CS-89-30, Utrecht University (Netherlands), 1989.

[3] EKPENYONG, F., PALMER-BROWN, D., AND BRIMICOMBE, A. Extracting road information from recorded GPS data using snap-drift neural network. *Neurocomputing 73*, 1–3 (2009), 24–36.

[4] EVEN, S. *Graph Algorithms*. Computer Software Engineering. W. H. Freeman & Co, New York, 1979.

[5] GANGEMI, A., GUARINO, N., MASOLO, C., AND OLTRAMARI, A. Understanding top-level ontological distinctions. In *Proc. IJCAI 2001 Workshop on Ontologies and Information Sharing*, H. Stuckenschmidt, Ed. 2001.

[6] GIBSON, J. *The ecological approach to visual perception*. Houghton Mifflin, Boston, 1979.

[7] HEINZLE, F., ANDERS, K.-H., AND SESTER, M. Pattern recognition in road networks on the example of circular road detection. In *Geographic, Information Science*, M. Raubal,

H. J. Miller, A. U. Frank, and M. F. Goodchild, Eds., vol. 4197. Springer, Berlin, 2006, pp. 153–167.

[8] HU, J., RAZDAN, A., FEMIANI, J., CUI, M., AND WONKA, P. Road network extraction and intersection detection from aerial images by tracking road footprints. *IEEE Transactions on Geoscience and Remote Sensing 45*, 12 (2007), 4144–4157. doi:10.1109/TGRS.2007.906107.

[9] KLIPPEL, A., RICHTER, K.-F., AND HANSEN, S. Structural salience as a landmark. In *Mobile Maps. Workshop at MobileHCI'05* (Salzburg, Austria, 2005).

[10] KLIPPEL, A., TAPPE, H., KULIK, L., AND LEE, P. U. Wayfinding choremes: A language for modeling conceptual route knowledge. *Journal of Visual Languages and Computing 16*, 4 (August 2005), 311–329. doi:10.1016/j.jvlc.2004.11.004.

[11] MACKANESS, W. A., AND MACKECHNIE, G. A. Automating the detection and simplification of junctions in road networks. *Geoinformatica 3*, 2 (1999), 185–200. doi:10.1023/A:1009807927991.

[12] MAY, M., HECKER, D., KÖRNER, C., SCHEIDER, S., AND SCHULZ, D. A vector-geometry based spatial knn-algorithm for traffic frequency predictions. In *Proc. 8th IEEE International Conference on Data Mining (ICDM 2008)*. IEEE Computer Society, 2008, pp. 442–447. doi:10.1109/ICDMW.2008.35.

[13] MAY, M., SCHEIDER, S., RÖSLER, R., SCHULZ, D., AND HECKER, D. Pedestrian flow prediction in extensive road networks using biased observational data. In *Proc. 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, W. G. Aref, M. F. Mokbel, and M. Schneider, Eds. ACM, New York, 2008, pp. 471–480.

[14] POSSIN, J. Affordance-basierte interpretation von Openstreetmap-Daten für Navigationsanwendungen. Diplomarbeit, University of Münster, Münster, Germany, 2011.

[15] RAUBAL, M. Ontology and epistemology for agent-based wayfinding simulation. In *Geographical Domain and Geographical Information Systems - EuroConference on Ontology and Epistemology for Spatial Data Standards* (France, 2001), S. Winter, Ed., vol. 19 of *GeoInfo Series*, La Londe-les-Maures, pp. 85–87.

[16] SCHEIDER, S. *Grounding geographic information in perceptual operations*. PhD thesis, University of Münster, Münster, Germany, 2011.

[17] SCHEIDER, S., AND KUHN, W. Road networks and their incomplete representation by network data models. In *Proc. 5th International Conference Geographic information science*, T. J. Cova, H. J. Miller, K. Beard, A. U. Frank, and M. F. Goodchild, Eds. Springer, Berlin, 2008, pp. 290–307.

[18] SCHEIDER, S., AND KUHN, W. Affordance-based categorization of road network data using a grounded theory of channel networks. *International Journal of Geographical Information Science 24*, 8 (2010), 1249–1267. doi:10.1080/13658810903514198.

[19] SCHEIDER, S., AND SCHULZ, D. Specifying essential features of street networks. In *Proc. 8th International Conference on Spatial Information Theory (COSIT 2007)*, S. Winter, M. Duckham, L. Kulik, and B. Kuipers, Eds. Springer, Berlin, 2007, pp. 19–23.

[20] STADLER, C., LEHMANN, J., HÖFFNER, K., AND AUER, S. Linkedgeodata: A core for a web of spatial open data. *Semantic Web Journal* (2012), forthcoming.

[21] TIMPF, S., VOLTA, G., POLLOCK, D., AND EGENHOFER, M. A conceptual model of wayfinding using multiple levels of abstraction. In *GIS - From Space to Territory: Theories and Methods of Spatio-Temporal Reasoning in Geographic Space* (London, UK, 1992), Springer, pp. 348–367.

[22] TURNER, P. Affordance as context. *Interacting with Computers 17*, 6 (2005), 787–800. doi:10.1016/j.intcom.2005.04.003.

[23] WINTER, S. Modeling costs of turns in route planning. *GeoInformatica 6*, 4 (2002), 345–360. doi:10.1023/A:1020853410145.

[24] XIE, F., AND LEVINSON, D. Measuring the structure of road networks. *Geographical Analysis 39*, 3 (2006), 336–356.