Link++: Adaptive Linking of Multiple Transportation Networks

Ali Masri[1,2], Karine Zeitouni[2] and Zoubida Kedad[2]

[1] VEDECOM
Versailles, France
{first}.{last}@vedecom.fr
[2] Université de Versailles Saint-Quentin-En-Yvelines, Laboratoire DAVID
Versailles, France
{first}.{last}@uvsq.fr

Abstract. The integration of heterogeneous transportation data and services is essential towards enabling multimodality. Many new services are emerging and gaining a lot of popularity but are still isolated from multimodal solutions such as ridesharing, bike sharing. These services tend to publish their data according to the principles of linked open data in order to allow this integration. However, existing data interlinking tools are not suitable to discover and create transportation connections. The discovery of transportation connections is dependent on the timetable and geospatial information of the entities. In addition, the representation of the connection needs to be rich enough to provide the required information for further processing. In this paper, we introduce Link++, a customizable interlinking tool that enables detection and generation of customized semantic connections between transportation datasets.

1 Introduction

Multimodality is the integration of multiple modes of transportation data and services. It allows us to form links between different modes and services in order to provide richer and more optimized planning services. What is interesting about transportation data is that these links are translated to real physical links between the transportation entities. For example, when we say that there is a relation between a railway stop and a bus stop, we mean that there is a physical path at a specific moment that connects these two stops. Therefore, we give passengers the ability to use this path as a transit service to switch from one mode to another. This switch may be very important in many cases. It improves trip time, maybe cost and for sure extends trip plan possibilities and options.

In general, existing approaches tend to solve the integration problem by mapping the data they need into a unified model, then storing the unified data into a repository supported by an API e.g. Google Transit[3],

[3] http://maps.google.com/landing/transit/index.html

STIF[4]. However, they still do not take into consideration highly evolving datasets such as car sharing, bike sharing, car pooling, etc. Such services are highly dynamic and often do not have the notion of a fixed transportation stop. In turn, this makes the integration problem more challenging and demanding special needs.

Our goal is to find a simple way for operators to identify nearby transportation services by providing a connection portal enabling the identification of connections between one transportation data source and another. We are in a need of a homogeneous light-weighted representation of transportation connections (transfer points from one stop to another) and the means to discover them in a flexible and customized manner. With this representation we can link different types of transportation services regardless the mode or service they offer. All what transportation systems need to know is just how to handle these light connections and use them to connect with the outer world, which is much simpler than handling heterogeneous data and maintaining them.

Enabling such solution requires access to transportation sources which can be obtained from open data [7, 5]. Open data is gaining a great deal of popularity and numerous transportation operators are using it to publish their data on the web[5,6,7]. The main cause behind publishing the data is to increase the market visibility for each service.

Many solutions took benefit from this to provide rich data for smart cities applications. They use linked data techniques and data interlinking tools to provide extended information relevant to both transportation and passenger profile queries [13, 6]. These techniques address equivalence detection between entities to establish links between data sources. This may help in enriching data about entities. However, this is not always enough in transportation data. Further complex relations are required to reflect the nature of transportation connections.

Beyond equivalence or sameAs links, we are interested in finding connections between transportation data sources based on the geospatial and time characteristics of the data which capture the reach-ability between different transportation networks. Furthermore, using the given tools we face two main limitations. The first is the restriction to a predefined set of functions for composing linking rules, due to the lack of flexibility of existing systems in defining custom functions. For instance, to calculate information such as closeness of two transportation points of transfer (bus stop, train station, etc), we can not define custom functions to calculate walking distances, driving distances, etc. The user is forced to dig into the code (if available) and modify it directly. The second limitation

[4] `http://www.stif.info`
[5] `http://opendata.paris.fr/page/home/`
[6] `http://www.strasbourg.eu/ma-situation/professionnel/open-data/donnees/mobilite-transport-open-data`
[7] `http://www.uitp.org/tags/open-data`

is the representation of the generated output. Supporting complex relations requires more complex output patterns.

As an example, let us suppose that a link is established between two transportation points of transfers. Existing tools can provide the output *BusStop1 nextTo TrainStation132* which does not give information about the occurrence of this relation. They are next to each others but how close are they? when the connection is available? and what are the modes of transportation that we can use? etc.

Based on what precedes, there is no way of creating connections that are suitable to many complex matching tasks other than the standard equivalence matching tools. Our main goal is to provide a system that is flexible and rich enough to allow users to define their own way of connecting data sources. Users must be given the power to use custom functions and define any form of output needed for their tasks.

In this paper we introduce Link++, a system that uses connection patterns, custom functions and rules to enable the generation of richer connections to link data sources. These connections can be applied to provide missing connections between transportation operators and services. With this approach, we can generate rich semantic links between entities to be published as open data in the sense of improving re-usability and reducing the need for re-calculation. We evaluate our approach using a real use case on connecting two transportation modes and checking how this affects the time of trips.

The paper is structured as follows:

2 Background and Related Work

With the introduction of LinkedData[5] many approaches took chance of leveraging and exposing their data to the web. In the category of transportation data and smart cities [2][13][6] did this integration by following the linked data principles[4][5] and they succeeded in connecting data from different sources to produce applications with wider-scope services. [10] tackled the problem of cataloging, exploring, integrating, understanding, processing and transforming urban information. They proposed an approach for incremental and continuous integration of static and streaming data, based on Semantic Web technologies, while they tested their system to a traffic diagnosis scenario.

The GeoKnow[11] project and DataLift[16] platform came as a solution to help transforming data from isolated silos into linked data. They provided the necessary tools to transform, link, publish and query data extracted from multiple different sources with different formats.

The way links are created for open data datasets are through data interlinking. Data interlinking in general is a way to discover similarity relationships between RDF data sources in a semi-automatic fashion. The

process of linking requires two datasets – source and target, a distance measure and a threshold. A link between two entities of a dataset is successfully assigned if a distance measure between them exceeds a selected threshold. The main goal is to link similar instances – that are scattered between different data sources – in order to expand the knowledge graph. The MeLinDa survey[17] described data interlinking in more details and highlighted the characteristics of the most popular approaches.

We can divide the link discovery frameworks into two categories: domain specific and universal ones. The domain specific frameworks aim to discover the links between knowledge bases of a particular domain. The second category is designed to consider the linking tasks regarding the knowledge base domain. Based on [17], table1 shows the most popular interlinking tools with their properties. RKB-CRS [9], proposed an ar-

	Techniques	**Output**	**Domain**
RKB-CRS[9]	String	owl:sameAs	Publications
GNAT[15]	String, similarity-propagation	owl:sameAs	Music
ODD-Linker[8]	String	link set	Independent
RDF-AI[18]	String, WordNet	alignment format	Independent
Silk[20]	String, numerical, date	owl:sameAs, user-specified	Independent
LIMES[12]	String, geographical, numerical, date	owl:sameAs, user-specified	Independent
Link++	User-defined	User-defined	Independent

Table 1. Comparison between different interlinking tools

chitecture for managing URI equivalences on the Web of Data by using Consistent Reference Services. Their approach requires a JAVA program to be written for each pair of datasets to integrate. In each program, the coder selects the resources to compare and the their comparison function using string similarity on the property values. GNAT[15] is an automatic interlinking tool that works on music datasets described within the Music Ontology[14]. It is implemented in prolog and based on similarity aggregation algorithm to detect relations based on resource's neighbors in a graph. ODD-Linker[8] proposed an extensible framework for interlinking relational data with a high quality links. Linking rules are expressed in LinQL that is later translated to SQL queries in order to compare and identify links. LinQL supports many string matching algorithms, synonyms, hyponyms and other conditions on attributes. RDF-AI[18] is a dataset matching and fusion architecture. It takes two files, the datasets to be linked and a set of XML files describing the linking process (preprocessing, matching configuration, dataset structure, merge configuration). A local copy of the datasets is needed, and the matching is based on string similarity with an external resource (WordNet). Silk[20] is a domain independent interlinking tool. Input datasets are inserted via a SPARQL endpoint URI, a local copy, or a database access. Matching configuration can be done either with a GUI toolbox or the Silk Link

Specification Language (Silk LSL). User specifies the properties to be matched, the pre-processing functions and the matching technique to be used. Matching function are combined via aggregation functions (MAX, MIN, AVG). Silk provides a load of pre-processing function on Literals and numeric data types. Many comparison functions are defined including string similarity, numerical distance, date-time. The only distance function available for matching geospatial datasets, is the geometric distance — based on the Euclidean distance. It takes the latitude and the longitude of both entities and matches them according to a given threshold. The good thing in this distance function is that the threshold is well formatted, the user can define the minimal distance in meters or kilo-meters. LIMES[12] is one of the tools provided by the GeoKnow[1] project, it handles the matching in a very fast speed compared to other link discovery frameworks. LIMES provide better distance functions for geospatial data, thus we have more options to match. It supports basic string metrics, numeric vectors such as Euclidean and Orthodromic distance metrics and many other similarity metrics e.g. Hausdorff, Sum of minimum, Frchet, etc. Writing a linkage rule in LIMES is done via XML, and no GUI is available to support the process. Although many distance metrics are supported, these functions are only good for geographical data. Geographical data is a subset of transportation data, so more powerful functions are needed, those who enable richer linking between transportation units or objects.

Analyzing existing link discovery approaches shows that they are more suitable to equivalence matching. They provide functions and aggregations to detect sameAs, part-of or subClass relationships. These approaches may be suitable in some cases for geospatial data (the GeoKnow project [1] and LinkedGeoData [19]), but they are not sufficient for transportation data. Interlinking solutions must take into account both the spatial and temporal characteristics of transportation data in addition to the real-time state. Consider that we want to connect two transportation data sources with the intention of discovering how we can reach one stop from another. Doing so with existing tools limits us to equivalence detection due to the provided functions and output format. What is required is a more representative and semantic way to connect these sources [3] showing how they can be connected from a transportation point of view. As a conclusion, the output of an interlinking process mainly focuses on detecting a set of owl:sameAs links. However, we need to have more information in the generated links to enable better post-processing and analysis and to reduce re-calculation costs (e.g., include information about a connection status and the distance between two connected entities in transportation links).

3 Link++

As we have discussed earlier, existing interlinking tools are not suitable to connect transportation data sources. What is required is a more customizable connection generation process to enable richer and more flexible connection representation. We introduce Link++ (shown in Figure

1), a system that enables flexible connection discovery and customized output definition using connection patterns, custom functions and linking rules. Connection patterns are templates for connection generation used to define both the content and format of a linking process output.

In general, the approach consists of two main phases:

- The definition phase, where users define the connection patterns, the required functions and linking rules.
- The generation phase, where: 1) the definitions are taken and applied to the datasets 2) the rule is applied to the entities 3) and when valid, a connection is created and stored in a repository.

In a formal definition, a linking task T requires the following input for the process:

- Input data sources $D1$ and $D2$ representing the datasets to be linked
- O is the custom-defined connection pattern
- R is the linkage rule that defines when a connection must be generated
- F is a set of functions required for the linking task
- L is a set of pre-defined libraries implementing the dependencies of F

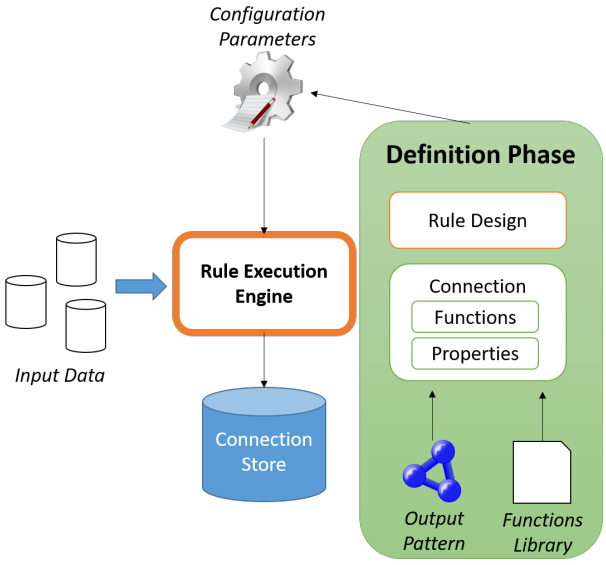

Fig. 1. Link++: An approach for flexible and customizable connection generation.

The following sections explain in detail the tasks required for an interlinking process.

3.1 Specifying Custom Functions and External Libraries

Enabling users to define their own functions is crucial for a complete system that supports all required matching tasks. Thus, the first task is to enable users to write any functions to be used in their linking rules or similarity calculations. Users can simply do it with Link++ with a custom Java class. In addition, external libraries are needed to support third-party code. It is supported and can be used in our system by simply adding the JAR file to the code repository.

The user specified functions play an important goal in the matching tasks since they can form a linking rule, a similarity metric, a transformation/preprocessing operations or any other function based on users' needs. The functions are gathered in a JAVA file accompanied with the used jar libraries, then they are compiled at run time and used when needed. To reference each method the user simply address the function as follows: "class-name"."method-name". For example: MyClass.walkingDistance.

3.2 Defining a Linking Rule

A linking rule specifies the conditions required to generate a connection between a given pair of entities. The main goal is to apply this rule to each entity pair in order to seek for a match and create the specified connection. Defining a rule requires a set of functions (similarity metrics and preprocessing functions) previously defined by the user. Each rule is defined with a root node that is either an aggregation or a comparison operator and sub-nodes specifying any other function chained in a way to suit the linking task.

An aggregation operator combines the values of different operators/values by applying the specified aggregation method, e.g., max, min, average, etc. It is defined by an aggregation function and a threshold. Each function contains a set of parameters that can be specified from the given data sources or directly by the user. The threshold defines whether the value of the operator must be evaluated as true or false in the linking rule.

Since data sources can be represented in different ways, we can use the transformation operator to modify this representation. To this end, we define a function that takes its parameters from the data sources or from the composition of other transformation operators, e.g., lowercase, uppercase, concatenation, round, ceiling, etc.

Finally, the comparison operator is used to define the similarity (or the relatedness) between two properties of the given data sources. A comparison is valid between operators themselves or with other transformation functions, and a threshold defines whether the value is accepted or not for the rule to be valid, e.g., distance, equality, etc.

3.3 Configuring a Connection Pattern

Connections are the final outputs of the interlinking task specified by connection patterns. Therefore, it is important to be precise when defining a connection pattern. A pattern specifies the format of the generated connections and the required information they must contain. In other words, it represents a template that will be filled when a connection is instantiated.

A connection pattern is composed of a set of properties, where each property is defined by a function that calculates it. Function parameters can be the inputs from the data sources or predefined by the rule composer. A connection pattern is freely chosen by a user according to the interlinking task and the post-processing needs. The formal definition of a connection pattern O is as follows:
Let D_1 and D_2 be two data sources. Given V any data type and F a set of custom functions required to generate the patterns, Pr is a set of properties where each property is represented by a property name n, a value v and a corresponding function f, which calculates the property value during the generation process.

$$Pr = \{(n, v, f) | n \in String, v \in V \, f \in F\} \tag{1}$$

A connection pattern is formalized as:

$$O = (d_1, d_2, pr) | d_1 \in D_1, d_2 \in D_2, pr \subseteq Pr \tag{2}$$

Both the connection pattern and the linking rule files are described in XML files that conform a data type definition (DTD); custom functions are written using JAVA (users can write any JAVA file and use the defined methods in his/her connection pattern or rule), and the output is generated in RDF.

We will give a demonstration case with a real scenario of defining both the linking rule and the connection pattern in the evaluation section. It illustrates the configuration process and shows an instance of the XML files (output pattern and rule).

3.4 Connection Discovery Algorithm

Once the configuration step is completed, the connection discovery process starts as described in the sequel. Algorithm 1 represents the pseudocode of the implemented linking process. The algorithm iterates over each pair of entities in the two data sources and evaluates the linking rule between them. Based on the rule evaluation, the algorithm decides if a connection must be created or not. If a rule is triggered, a new connection is generated by evaluating the connection pattern and applying the corresponding function of each property. The values are calculated by the specified functions in the output pattern, and their parameters are

filled from the currently-compared entities. Here, we instantiate the connection and fill in its information from the return values of the functions. The connection is stored in a specified repository, and the algorithm continues on the remaining pairs until all are treated. In the worst cases,

```
Data: D1, D2, O, R, F
Result: Discover the list of connections and add them to the connections store
/* iterate over the elements of D1                                    */
foreach e1 in D1 do
    /* iterate over the elements of D2                                */
    foreach e2 in D2 do
        /* evaluate the linking rule                                  */
        if evaluateRule(e1, e2, R) is true then
            /* if the rule holds, create new connection based on the
               output pattern                                         */
            c ← createConnection(e1, e2, O);
            /* calculate the value of each property in the pattern
               based on the specified function                        */
            foreach p in c.properties do
                f ← F.getFunction(p.getFunction);
                value ← f.calculate(p.getProperties);
                c.addProperty(p.name, value);
            end
            /* the connection is instantiated and ready to be added to
               the connection store                                   */
            add c to connections store;
        end
    end
end
```

Algorithm 1: Connection discovery algorithm.

the time complexity of the algorithm is O(n * m), where n and m are the sizes of the input datasets. The storage complexity (in terms of data pages) is the same as a nested loop join in databases that is equal to the size of the smallest dataset + one page, which usually fits in memory. This complexity may be reduced by using some pre-filtering techniques that the system may offer in a future version; for instance, using a spatial index to replace the inner loop by a search in an index (which reduces the cost to log(n)). Then, the specific rules and function defined by the user will be applied in a refinement phase automatically by the system.

Link++ is implemented and an executable version of it can be found online via the link `https://github.com/alimasri/link-plus-plus.git`; in addition to a video tutorial on: `https://youtu.be/u2gr7Wa4eT4`. A screen-shot of the system is shown in Figure 2. It shows the project-based graphical user interface where users can add their data sources,

functions, linking rule and connection pattern. The output is shown in a separate directory.

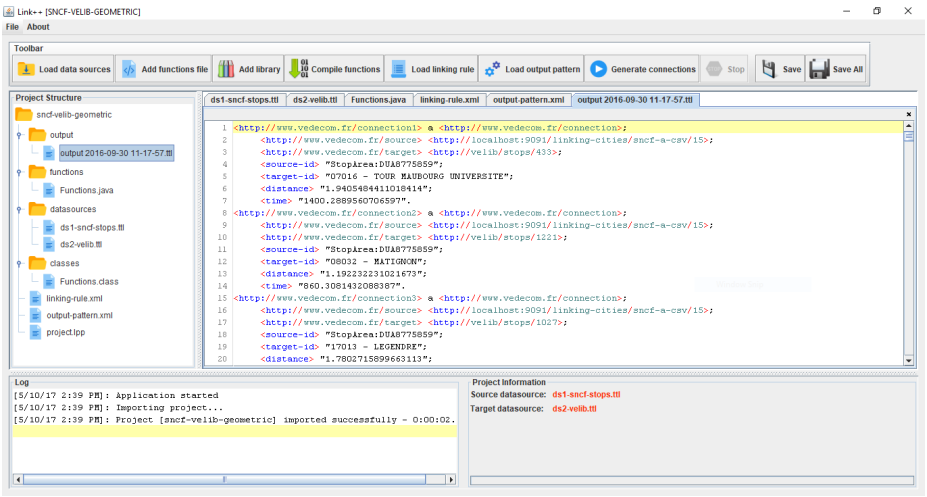

Fig. 2. A screen-shot of Link++

4 Evaluation

To recall, in transportation networks, a connection can be described as an accessible path from one transportation point of transfer to another. A point of transfer is any stop that allows users to change a transportation unit or mode. A connection contains properties describing both the departure and arrival stops in addition to other properties. We define a transportation connection as one of the following two types:

- Timetable connection that has specific departure and arrival times. This type of connection will be referred to as a scheduled connection. It has the following properties: departure-time, arrival-time, departure-stop and arrival-stop.
- Other connections that have no schedule information and for which availability is not restricted by timing constraints. We will refer to these connections as unscheduled connections. They have the following properties: departure-stop, arrival-stop and distance.

We evaluate the approach using the datasets from SNCF[8] and Autolib[9] companies. The number of instances in each of the SNCF and Autolib datasets is 1067 and 869, respectively.

[8] `http://gtfs.s3.amazonaws.com/transilien-archiver_20160202_0115.zip`

[9] `http://opendata.paris.fr/explore/dataset/stations_et_espaces_autolib_de_la_metropole_parisienne/`

In the following, we describe the evaluation phases from preparing the data, setting-up the system and visualizing the generated output.

Data Preparation In this phase, the goal is to represent the timetable information in a format compatible with our definition of connection. Instead of designing a network by a series of stops or other representations, we want to represent it by a series of connections between stops. Since SNCF is a public transportation company with data described in timetables, the task here is to extract scheduled connections from the given data.

To this end, we have proposed an algorithm that transforms timetable data from GTFS files into scheduled connections. The algorithm iterates over the timetable information for each stop and creates a connection that starts from a departure stop at a departure time and ends with an arrival stop with the specified time. The process is repeated to a predefined date range to limit the number of connections created.

In case of Autolib, we do not have timetable information, so we need a way to discover the connections between its stops. Using our approach, we can match Autolib's dataset with itself (in order to know when a Autolib station is reachable from an another) to discover these unscheduled connections between. Since the configuration task is common and independent, the following section describes how to use our approach to discover the unscheduled connections for Autolib-Autolib and Autolib-SNCF.

Discovering New Connections Two tasks are required one for Autolib-Autolib connections and one for Autolib-SNCF connections. In this example, unscheduled connections are driving or walking connections between Autolib-VELIB and Autolib-SNCF, respectively. We use our approach to search for connections that match a predefined criteria. Since our approach works on RDF data, we have used the DataLift [16] platform to transform both SNCF stops and VELIB CSV files into RDF turtle formats. In the sequel, we describe in detail all of the required tasks to achieve our goal.

- Defining custom functions: Our system is flexible as it allows users to create any custom function to be used in the linking task. Users can use external dependencies, as well. In our example, we define the functions getWalkingDistance, getWalkingTime, getDrivingDistance and getDrivingTime. In a real scenario, we get this information from a web service, such as Google's distance matrix API (`https://developers.google.com/maps/documentation/distance-matrix/`). However, due to the query limit, we have chosen to implement them by local functions based on mathematical calculations (`http://www.movable-type.co.uk/scripts/latlong.html`).
- Define the linking rules: Recall that the linking rule describes the condition that triggers the creation of a connection. Two rules are required, one for Autolib-Autolib and the other for Autolib-SNCF.

For the first one, the condition of the defined rule is the following: "If a driving path exists within 200 km (the time before the battery is totally discharged), create a connection". For Autolib-SNCF connections, the rule is: "If a walking path exists from one stop to another within one kilometer, create a connection".

Rules are written in XML format, and the functions that calculate the walking distance and time are referenced from the custom functions file. We note that the parameters "200 km" and "1 km" are given by the user who is responsible for the configuration. We set these parameters as the maximum feasible scope for a person to ride the car or walk from one station to another. Figure 3 shows an example of how a rule can be defined.

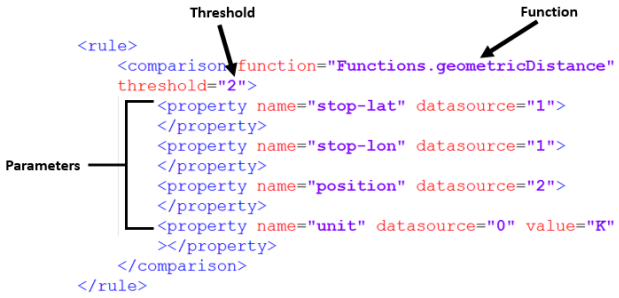

Fig. 3. An example of rule definition in XML.

– Defining the connection pattern: We define the output generated by the system at each valid rule. We have chosen the following properties to be represented in a connection pattern: source-id, target-id, walking/driving distance and walking/driving time. This pattern is the same for both tasks, and an example is shown in Figure 4.

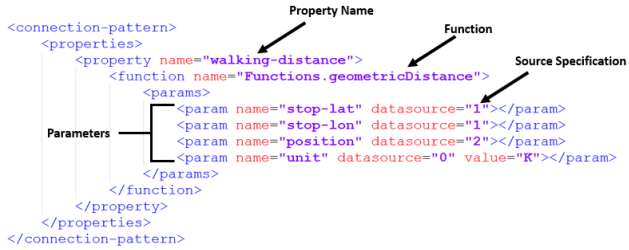

Fig. 4. An example of a connection pattern in XML.

Executing these tasks with the above configuration enabled us to enrich the network with discovering 535,966 internal connections between Autolib car stations and 272 new connections between the two different transportation modes SNCF and Autolib.
Compared to the existing link discovery frameworks, our approach succeeded in discovering links with richer representations and customized properties that can be used for numerous tasks.

5 Conclusion

Open data made it possible for new companies and services to gain visibility in the market. New transportation services are taking advantage of this to provide the needed data to support multimodality. As we have seen, existing open data interlinking tools are not suitable to support transportation data linking which represents real connections in the physical world. In this paper, we introduce Link++, a customizable interlinking tool that enables detection and generation of customized semantic connections between transportation datasets.

Link++ provides the means for existing transportation systems to access information about nearby services and integrate them. We have introduced how Link++ defines custom connections and used these connections to expand a transportation network containing trains and car sharing networks. In the future, our focus will target the dynamic part of the connections since they are not always static and may be affected by external events. On the other hand, we aim to take profile information while generating connections which introduces the need of real-time connection generation. Furthermore, we will consider optimizing the Cartesian product in the discovery algorithm.

References

1. Spiros Athanasiou, Daniel Hladky, Giorgos Giannopoulos, Alejandra Garcia Rojas, and Jens Lehmann. Geoknow: Making the web an exploratory place for geospatial knowledge. *ERCIM News*, 96:12–13, 2014.
2. Sören Auer, Jens Lehmann, and Sebastian Hellmann. *Linkedgeodata: Adding a spatial dimension to the web of data.* Springer, 2009.
3. Montserrat Batet, Sébastien Harispe, Sylvie Ranwez, David Sánchez, and Vincent Ranwez. An information theoretic approach to improve semantic similarity assessments across multiple ontologies. *Information Sciences*, 283:197–210, 2014.
4. Christian Bizer. Evolving the web into a global data space. In *BNCOD*, volume 7051, page 1, 2011.
5. Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pages 205–227, 2009.

6. Sergio Consoli, Misael Mongiovì, Diego Reforgiato Recupero, Silvio Peroni, Aldo Gangemi, Andrea Giovanni Nuzzolese, and Valentina Presutti. Producing linked data for smart cities: the case of catania.
7. Michael B Gurstein. Open data: Empowering the empowered or effective data use for everyone? *First Monday*, 16(2), 2011.
8. Oktie Hassanzadeh, Lipyeow Lim, Anastasios Kementsietsidis, and Min Wang. A declarative framework for semantic link discovery over relational data. In *Proceedings of the 18th international conference on World wide web*, pages 1101–1102. ACM, 2009.
9. Afraz Jaffri, Hugh Glaser, and Ian Millard. Managing uri synonymity to enable consistent reference on the semantic web. 2008.
10. Spyros Kotoulas, Vanessa Lopez, Raymond Lloyd, Marco Luca Sbodio, Freddy Lecue, Martin Stephenson, Elizabeth Daly, Veli Bicer, Aris Gkoulalas-Divanis, Giusy Di Lorenzo, et al. Spud semantic processing of urban data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 24:11–17, 2014.
11. Jon Jay Le Grange, Jens Lehmann, Spiros Athanasiou, A Garcia-Rojas, Giorgos Giannopoulos, Daniel Hladky, Robert Isele, A-C Ngonga Ngomo, M Ahmed Sherif, Claus Stadler, et al. The geoknow generator: Managing geospatial data in the linked data web. *Linking Geospatial Data*, 2014.
12. Axel-Cyrille Ngonga Ngomo and Sören Auer. Limes-a time-efficient approach for large-scale link discovery on the web of data. *integration*, 15:3, 2011.
13. Julien Plu and François Scharffe. Publishing and linking transport data on the web: extended version. In *Proceedings of the First International Workshop on Open Data*, pages 62–69. ACM, 2012.
14. Yves Raimond, Samer A Abdallah, Mark B Sandler, and Frederick Giasson. The music ontology. In *ISMIR*, pages 417–422. Citeseer, 2007.
15. Yves Raimond, Christopher Sutton, and Mark B Sandler. Automatic interlinking of music datasets on the semantic web. *LDOW*, 369, 2008.
16. François Scharffe, Ghislain Atemezing, Raphaël Troncy, Fabien Gandon, Serena Villata, Bénédicte Bucher, Fayçal Hamdi, Laurent Bihanic, Gabriel Képéklian, Franck Cotton, et al. Enabling linked data publication with the datalift platform. In *Proc. AAAI workshop on semantic cities*, pages No–pagination, 2012.
17. François Scharffe and Jérôme Euzenat. Melinda: an interlinking framework for the web of data. *arXiv preprint arXiv:1107.4502*, 2011.
18. François Scharffe, Yanbin Liu, and Chuguang Zhou. Rdf-ai: an architecture for rdf datasets matching, fusion and interlink. In *Proc. IJCAI 2009 workshop on Identity, reference, and knowledge representation (IR-KR), Pasadena (CA US)*, 2009.
19. Claus Stadler, Jens Lehmann, Konrad Höffner, and Sören Auer. Linkedgeodata: A core for a web of spatial open data. *Semantic Web*, 3(4):333–354, 2012.
20. Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Silk-a link discovery framework for the web of data. *LDOW*, 538, 2009.