POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di laurea magistrale

Data fusion algorithm for a touristic based web application. Case study: the city of Turin

Relatore:
Fulvio Corno

Candidato:
Marina Ciavarra

Marzo-Aprile 2020

Contents

List of Tables

List of Figures

Abstract

In this decade, data represent a valuable source for all the companies, not only the IT related one. Data quality delineates a critical problem when data from different sources are combined. The thesis has two complementary objectives. The first one is to develop a web application for the creation of customized touristic tours. The application allows the user to insert and categorize the point of touristic interest, and it allows to create tours based on the user location and the categories selected by it. In the application contest, points of touristic interest are described by name, latitude, longitude, and are categorized using six categories containing different subcategories. The categories selected for the project are six: building, museum, artistic movement, outdoor space, related figure, and activity & leisure. The building category includes subcategories that specify the use of it. Whether the point of interest reflects an architectural style, the artistic movement category should be selected. The related figure category includes famous people who lived in or built the point of interest. The other three categories are straightforward and do not need additional explanation.

The second objective of the thesis is to design and implement a data fusion algorithm to obtain, extend, and validate the database containing notable touristic places related to the city of Turin, which is the foundation of the ReactJs web application developed for this project. At first, the application was supposed to retrieve the point of touristic interest only from the user, but this idea was immediately discarded because the amount of data would be meager and difficult to validate. In this scenario, the data could be validated by an expert creating a bottleneck and a single point of failure in the validation process, or it could be created an algorithm to validate the data using a quorum system or by comparing the data with different sources. The next step was to integrate the data maintained by the Turin city hall, but its dataset does not contain complete information about points of touristic interest. It contains some data concerning different categories, but they are not linked together. The ultimate source to be taken into consideration is the data maintained by Wikipedia, which covers different data compared to the one maintained by the city hall and considers multiple categories for each point of interest. Since it does not exist a unique and trustworthy source to rely on, to retrieve these points of touristic interest, all the sources above are used. In the end, the schema heterogeneity and the need to automatically validate the data inserted by the users has led to the need to adopt data fusion techniques.

The web application follows a client-server architecture where clients can use any desktop computer with a modern browser. The server can be unique or replicated, and it communicates with a PostgreSQL database and the OTP service. The data fusion algorithm is not executed every time there is an insertion in the database, but it is executed periodically. The data fusion system is independent of the application back-end because it interacts only with the database instance. It is executed locally by the system administrator, and it only depends on the data inserted in the database. This system is seen

as a black box by the application and the system administrator. It is composed of three sequential modules that handle the problem at three different levels: schema, tuple, and values.

Data fusion is a technique allowing the integration of heterogeneous data coming from different sources in order to achieve a higher level of data quality. The latter can be expressed both qualitatively and quantitatively, depending on their usage. In the data fusion domain, high-quality data are the ones that represent more similarly a real-world entity. The first misunderstanding in approaching data fusion is to assume that it is a synonym of data integration. Both techniques deal with heterogeneous data, but data integration takes specific pieces of information from trusted sourced without merge together all the information as done by the data fusion. Data fusion was initially created for military-related problems, but despite its origin, this technique has acquired, over time, increasing emphasis in the information integration field. The growing of the information available on the internet and their easy access has encouraged interest in data fusion, based on the heterogeneity of data contained and in their representation.

The data fusion algorithm designed for the project belongs to the three steps data fusion algorithm because it handles the problem of schema heterogeneity at three different levels: schema, tuple, and values. The schema matching step deals with three different datasets with the same file format but different schema. Since the data will be used as support for the application, and already exists a schema used by the application to retrieve the data from the user, it is straightforward to use that schema. This phase of the algorithm is context-aware, and it uses two different strategies to transform the data based on the dataset considered. The AperTO dataset has two inconsistencies: the first inconsistency regards the coordinates representation and the other regards the geographical reference system used. The last transformation executed in this phase is the mapping between categories included in the AperTO and WikiData datasets in string format and their corresponding identifier stored in the application database.

The core of the algorithm is the duplicate detection phase because the data found here underlie the data fusion process. The algorithm has to deal with a massive amount of data and limited computational resources. The performances of the algorithm depend on the amount of comparison executed, to overcome this challenge, the search space must be reduced. For the thesis was selected the Duplicate Count Strategy algorithm with multiple record increase (DCS++) that is based on the Sorted Neighborhood Method, and it adjusts the size of the window based on the duplicate number found during each iteration. It requires a starting size of the window and a threshold to allow the growth of the window. The parameters that can be modified to achieve better performance during the execution of the algorithm are three: the starting size of the window, the sorting method, and the duplicate detection function. Since duplicates can be found based on the name or the coordinate inserted two sorting methods and two detection function can be implemented. The data can be sorted following two strategy: applying the order by clause to the SQL query to the database whit which data are retrieved alphabetically based on the name inserted in the database or modifying the data after they are retrieved from the database by a simple SQL select query by iterating over the data, and for each pair, calculate the Euclidean distance between them and ordering the points accordingly. Since the discriminating columns in the dataset are name, latitude, and longitude, the comparison to find duplicates can be executed on each of them individually, or they can be combined. To compare the coordinates is always used the Euclidean distance metric. The string similarity algorithm is applied to the name attribute using different types of measurements that follow different techniques to check the similarity. The first one proposed is the Ratcliff-Obershelp similarity, a sequence-based algorithm that searches the most extended sequence contained in both strings. Levenshtein distance belongs to the edit distance-based algorithms. The similarity is estimated by counting the number of operations need to transform the first string into the second. It belongs to the same category of algorithms, the Jaro-Winkler algorithm. However, it has two conditions: strings must contain the same characters with a specific distance between them, and the matching order must be the same. The last algorithm considered is the Jaccard, a token-based algorithm that works on single characters instead of strings. Four combinations came out: sort the data alphabetically and emit duplicates based on similarity measurement, sort the data alphabetically and emit duplicates based on the combination of string similarity and coordinates distance, sort the data and emit the duplicates following the Euclidean distance, sort the data following the Euclidean distance and emit the duplicates based on the combination of string similarity and coordinates distance.

For each case, the quality of the algorithm is established using the confusion matrix and the metrics: precision, recall, and F-Score. The most important is the last one that represents a weighted accuracy that can be used with unbalances datasets and is a weighted average of precision and recall. In addition to the four combinations expressed before, in the evaluation, three other parameters vary window's size win, the threshold Φ_n for string similarity, and the threshold Φ_c for the Euclidean distance. The analysis is first executed within each combination comparing the different similarity metrics when the window's size and the thresholds vary. Then all the combinations are compared to found the one that fits better in the project scope.

The last part of the algorithm executes the fusion and the validation of the data applying two different resolution strategies. The WikiData dataset is chosen as gold standard for name and coordinates. Regarding the selection of categories and subcategories, the algorithm behaves differently based on the source. When the algorithm deals with data coming from AperTO and WikiData datasets, it integrates their categories without any further check instead user data are subject to additional analysis, and to be integrated into the final categories, at least three users have to insert the same category on the selected place to be considered valid. The threshold is chosen empirically and can be a point of weakness of the algorithm.

With the algorithm presented here, the weighted accuracy F-Score achieved is about 89% without any false positive detected. It is a good approximation for the project goal that has been fully respected since the algorithm validates the data inserted by the users, integrates data coming from different sources, and increases the quality of the data used in the project. It should be clarified that this solution is not the perfect one reaching an F-Score of 100%, but for the thesis scope, it reaches an acceptable value. Another point of weakness is the quorum threshold selected because it was selected empirically without appropriate evaluations due to lack of time.

“ *Grapes must be crushed to make wine.*
Diamonds form under pressure.
Olives are pressed to release oil.
Seeds grow in darkness.
Whenever you fell crushed, under pressure, pressed, or in darkness,
you're in a powerful place of transformation and transmutation.”
-Lalah Delia-
Vibrate Higher Daily: Live Your Power (Pg.62)

Chapter 1

Introduction

According to Istat, in 2018, 428.8 million tourists came to Italy, reaching a new all-time high [44]. Italy, with its landscape and its history, is in the top five countries visited worldwide. Knowing new cultures and places are the driver elements into a new trip choice. The internet is the primary source of information both for the journey organization and the place to visit. These lead to the development of a trip-planning web application that, using both existing datasets and user-inserted data. Database plays a vital role in IT and economic base industries, the quality of data stored in its highly influence the system that relies on data to conduct business. Using different data sources increases the data quality but can lead to inconsistency. Data fusion tries to overcome this limitation to achieve a complete e consistent dataset containing quality data compared to a single source of information.

The thesis has two complementary objectives. The first one is to develop a web application for the creation of customized touristic tours. The application allows the user to insert and categorize the point of touristic interest, and it allows to create tours based on the user location and the categories selected by it. In the application contest, points of touristic interest are described by name, latitude, longitude, and are categorized using six categories containing different subcategories. The categories selected for the project are six: building, museum, artistic movement, outdoor space, related figure, and activity & leisure. The building category includes subcategories that specify the use of it. Whether the point of interest reflects an architectural style, the artistic movement category should be selected. The related figure category includes famous people who lived in or built the point of interest. The other three categories are straightforward and do not need additional explanation.

The second objective of the thesis is to design and implement a data fusion algorithm to obtain, extend, and validate the database containing notable touristic places related to the city of Turin, which is the foundation of the ReactJs web application developed for this project. At first, the application was supposed to retrieve the point of touristic interest only from the user, but this idea was immediately discarded because the amount of data would be meager and difficult to validate. In this scenario, the data could be validated by an expert creating a bottleneck and a single point of failure in the validation process, or it could be created an algorithm to validate the data using a quorum system or by comparing the data with different sources. The next step was to integrate the data maintained by the Turin city hall, but its dataset does not contain complete information about points of touristic interest. It contains some data concerning different categories, but they are not linked together. The ultimate source to be taken into consideration is the data maintained by Wikipedia, which covers different data compared to the one maintained by the city hall and also considers multiple categories for each point of interest. Since it does not exist a unique and trustworthy source to rely on, to retrieve these points of touristic interest, all the sources above are used. In the end, the schema heterogeneity and the need to automatically validate the data inserted by the users has led to the need to adopt data fusion techniques.

The case study of the touristic web application for the city of Turin is presented in chapter 2. Here are reported the state of the art in the touristic application field, then the requirements, the system architecture, and the development of the web application. Chapter 3 introduces data fusion in general and explains the different classification of data fusion algorithms. In this chapter are described the three steps data fusion algorithm widely used in research. Chapter 4 includes the specification of the data fusion algorithm created for the project. It explains in detail the three phases of the algorithm as well as the datasets used. Some pseudocode is inserted to explain the algorithm better and the solution adopted. The analysis of the algorithm, the evaluation metrics, and the result are presented in chapter 5, while chapter 6 contains conclusions and future works.

Chapter 2

Case Study

In modern society, people travel all around the world for many different reasons but knowing new cultures and places is the driver element into a new trip choice. Nowadays, the internet is the primary source of information both for the journey organization and the place to visit. These lead to the development of a trip-planning web application that, using both existing datasets and user-inserted places, generates a touristic tour based on specific categories inserted by the user on the GUI. The application is specifically focused on the city of Turin.

In this chapter is described the touristic-application developed. In section 2.1 are described the scientific and technological progress in data collection, map creation, and trip planning. Functional and non functional requirements are reported in section 2.2, followed by the description of the system architecture in section 2.3 and its development in the last section 2.4.

2.1 State of the art

This project affects many specialists and large areas of the IT world. The primary concern is if there are existing datasets that fit the project needs or, in the contrary case, how to collect data from the users. Subsection 2.1.1 defines what a Dataset is and describes different existing datasets, which can be used. There is also a discussion on why they don't fit the project goal. When the problem of data is solved, the development of the touristic tour turned out, subsection 2.1.2 analyze the existing tools for map creation with their advantage and disadvantage. Subsection 2.1.3, instead analyzes the existing software for trip planning development and the existing applications for trip planning.

2.1.1 Existing Data Set

Before list the dataset related to touristic information found during the researches, it is essential to give the dataset definition. Dataset is a collection of information composed of separate elements organized in block structures that can be manipulated as a unit. It exists many different types of dataset, differentiated on the data storage and structure.

Different website maintains various dataset that classify data from all kinds. One need only think about government entities as the European Union, dates September 2019, it provides 13879 up-to-date datasets [40]. Google created in 2008 a public data explorer allowing the user to search and examine large third-part datasets in the form of graphs, plots, or on map [38]. Torino city hall maintains 1750 datasets divided into: people and society, public administration, economics and finance, cities, environment, education and culture, transportation, tourism, health, elections, science and technology, agriculture and fishing, and energy [39]. Between all of them, only three fit in the project scope: tourism, education and culture, and environment. Into these categories were selected seven datasets that fit in the application scope: libraries, historical places, theaters, museums, market, themed market, and green areas.

Even if the *AperTO* dataset covers different domains in the application scope, many of the most important sites of cultural interest are not considered here, to overcome this limitation, the largest free online encyclopedia can be used. Wikipedia contains two different datasets: DBpedia and WikiData. DBpedia extracts information from Wikipedia pages and publishes them as Linked Open Data [43]. WikiData is a secondary database that collects structured data [42], it gathers data and their sources imposing a very structured schema to ensure easy reuse of the data. Both of them include data concerning the city of Turin, precisely 31 *DBpedia* entries and 1294 *WikiData* ones, and after detailed research, only the *WikiData* data were selected for the application.

Given that, the two dataset were not exhaustive, to have the data needed for the scope, the next step was to collect the categorized data from the most potent source of information: the users.

2.1.2 Map creation tools

Software technologies are always under evolution, although the tech giant are the driving force of this industry, they are not the undisputed market leaders. When a developer has to insert a map with some functionality, the first API considered is the Google Maps one.

Google Maps is a project born in 2005 that allows map visualization, but the services built on top of it is its strength. It allows to research restaurant, monument, bus station, airport and also to calculate the road routs between two or more points. Street view and traffic information are two additional services integrated with it [31]. The API renders four basic map types:

Roadmap is the default road view.

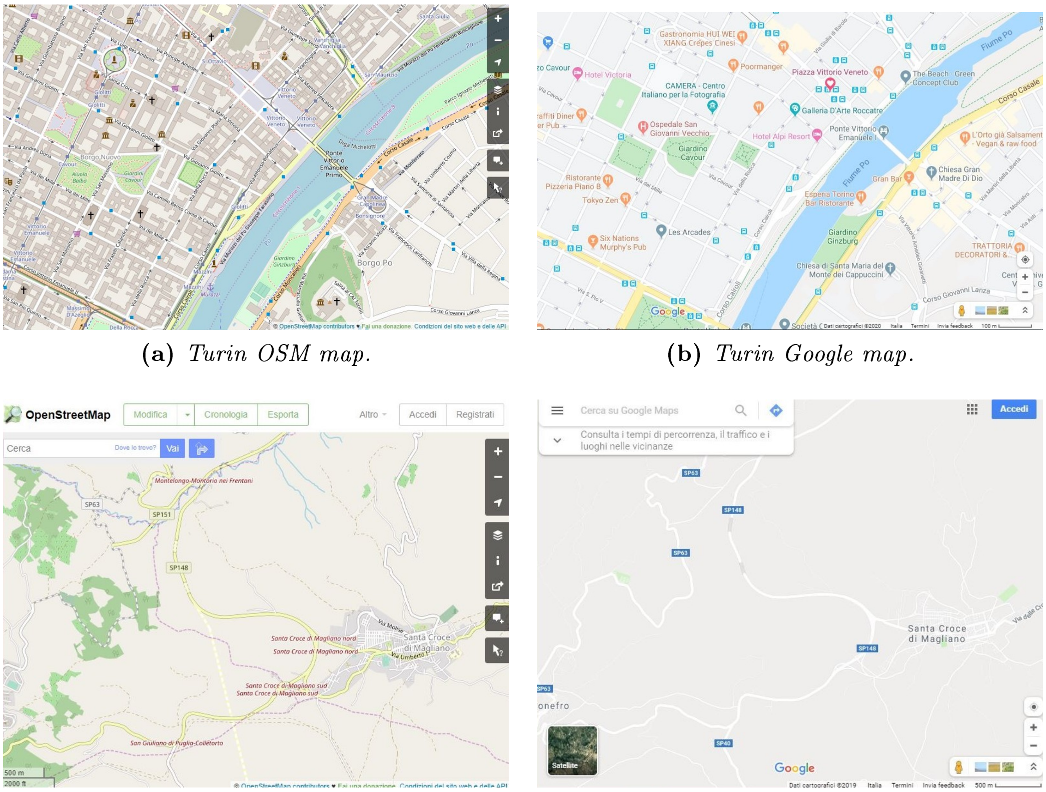
Satellite includes the Google Earth images.

Hybrid is a combination of the previous two.

Terrain displays a physical map based on terrain information.

Both the element and the style of this basic map can be customized. The API supports the overlay map type which is designed to work on top of an existing map type, and it adds a new layer showing additional information to the user. There is also the possibility of creating a custom map type.

At its first release, Google Maps was a free service but over the years have been applied overly stringent rules. Nowadays, to use the API, the developer must include an API key, enable billing on all of its projects, and it has a maximum of 500 requests per second [32]. Moreover, the developer has no control over the policy applied by Google. On the market, there are various alternatives. Apple MapKit is an Apple library that has higher limits for map views and service calls, but it is still a beta version [45]. Bing Maps [46]: is similar to Google one, but Microsoft owns it. It has 125000 free calls per year. These three API are not for free, and they have a closed approach to data collection and distribution.



(a) *Turin OSM map.*

(b) *Turin Google map.*

(c) *South of Italy OSM map.*

(d) *South of Italy Google map.*

Figure 2.1: Examples of detail level comparison between OpenStreetMap and Google maps in two different locations. Figure (a) and (b) is a capture of the area near the Po river in Turin. Figure (c) and (d) opposes the detail level in a not touristic area in the south of Italy.

OpenStreetMap (OSM for conciseness) instead adopts an open philosophy: it is a free and editable map released under the open-content license. Their goal is to create a database which can be used by third parties. Nowadays, OSM maps are used both for humanitarian aid and scientific research. The OSM project collects data using a collaborative model similar to the Wikipedia one. It creates a free editable map of the world entirely updated by the volunteers. The updates are immediately visible to all the users, and the user and the OSM community own them. OSM also uses satellite images and government data for map production[34]. Quality and granularity, as well as equal coverage, is guaranteed by the OSM community. Google instead puts more effort and resources into areas that are most profitable to sell. The figure 2.1 shows well this concept comparing the Google and OMS maps in two different places: Turin and a small town in the south of Italy. Even if the Google map in Turin is detailed, this map is still less updated that the OSM one. For example, in figure 2.1 (a) we can see the real shape of Vittorio Veneto square, that is simply represented as a rectangle in the Google map. OSM maps can be used both directly for their server or through an open-source Javascript library. All the features of Google Maps are supported in OpenStreetMap and in some cases, they are even better.

2.1.3 Trip planning

Over the last decades, various Trip Planner systems have been developed, and researches are still ongoing. A transit trip planner is a particular engine that assists users in planning their trip inserting origin and destination, departure or arrival time, and the transport means. Most systems are based on static schedules that don't take into consideration the traffic congestion and so the possible delay. There are also real-time transit trip planners includes Vehicle Mounted Unit (VMU), on the bus. This system, based on the GPS mounted in the vehicle, provides real-time location information which is used to track the movement and, accordingly, the arrival time.

In 1999, the first multimodal itinerary planner was developed to provide the tourist with schedules of public transports. The Itinerary Planner was developed to help travelers find a suitable itinerary by generating alternative travel plans for a single Origin-Destination pair with time and mode of travel constraints. It didn't consider the real-time scenario but the unique one to start the Multimodal Trip Planner with constraints. In 2010 was introduced the Park-n-Ride mode support for multimodal trip planner. It considers parking lots near public transportation access points. During the years, different optimization algorithms have been created by inserting data caching and applying more strict search criteria. For transit route planners to work, transit schedule data must always be kept up to date. To facilitate data exchange and interoperability, in 2006 was developed a standard data formats have emerged called General Transit Feed Specification or GTFS [36]. The most advanced trip planner until today is Google Transit [35].

There are another two competitors with similar performance and free: OpenTripPlanner and Navitia. OpenTripPlanner or OTP is an open-source and multi-platform multimodal trip planner. It has a monolithic architecture that makes it easy to configure and run, and It uses the GTFS standard. Real-data informations are available as a continuous stream from the GTFS-RT. Navitia is an open-source framework provided as a hosted open service preloaded with open data from several regions. It has a modular system more suitable for high-throughput service and uses an extended transit data similar to GTFS but incompatible with it. The integration with real-time data is slow due to the protocol applied [37].

Journey planners use an in-memory representation of the transportation network and timetable to enable an efficient and rapid search and routing algorithms to search the transport network graph. The routing can use Dijkstra's algorithm when it is independent of time. Users can create a custom itinerary by using different websites that have a pre-built database of points of interest.

Google travel put together flights, and hotels deal with information on the trip destination. In the explore section, the user gets an overview of the main activities in the selected city. Then it sees the popular things to do and some predefined Day plans based on actual travelers' visits.

Itineree helps the trip planning. It has a section called Do & See This Day, where the user can see the top ten places for TripAdvisor or the most recommended attractions. It can also search for places inserting a keyword, but for this feature, the website uses an external resource like Google and TripAdvisor. Besides, it also includes flight, hotel, and restaurant facilities.

Sygic Travel Maps shows attractions, hotels, restaurants and shops directly on the map using a proprietary database of touristic attractions. There are two other sections where the main attraction and the exiting touristic tours are presented.

TripHobo is an itinerary building website that based on seven categories, and the number and typology of travelers create a customized tour. The categories available are adventure, arts and culture, entertainment, historical, leisure, outdoors, and museum. It also includes transportation and hotel facilities.

Roadtrippers has two main functionality: explore places or plan trip. The explore place section allows the user to select some feature as accommodation, attraction and culture, food and drink, outdoors and recreation, point of interest, entertainment and nightlife and some others more. It uses a map visualization where the user selects one or more points, and the system renders the road trip between them.

Inspirock is like TripHobo, but it is limited to selected countries, and it doesn't allow the user to edit the trip after its creation. The filter mechanism enables the user to select the degree of point of interest popularity and one or more categories between culture, romantic, museum, outdoors, beaches, shopping, relaxing, wildlife, and historical sites. The daily trips proposed are based on existing touristic tours.

Ixigo uses the same style of Google travel. When the user inserts the city, it gives him some tips about the best period of the year to visit the city with the weather forecast and which fabric type is more suitable for the selected period.

There are also countless other web sites that allows the user to insert the itinerary manually and to keep track of his trip without give any suggestion on the journey creation.

2.2 Requirement

The following chapter contains the requirements, both functional and non-functional, that ensure the system quality. In the development, they were the milestones to monitor the progress during the months. Since there wasn't a definite client, the requirements are the result of market research and the opinion of the project advisor.

Before any further specification, it is mandatory to explain the difference between standard and authorized users. A standard user can access the website, navigate through pages, insert categories and places, and generate a tour based on its preference. The authorized user has specific priviledge given by the system administrator. It can review the data inserted by the standard user and validate them.

2.2.1 Functional Requirement

The functional requirement specifies the detailed requirements which the system shall meet. The different scenarios are presented using the Use Case diagram. Moreover, each use case is described specifing:

- Main actor
- Precondition
- Input
- Output

Use Case UC01: Register

RF01	User registration
Main Actor	User, both standard and authorized
Precondition	The user didn't have an open session, and he is not registered in the system.
Input	The user wants to access the system

Continued from previous page

Output	1. The user opens the application. 2. The system shows the main page. 3. The user selects *"Sign Up"* button. 4. The system shows the sign-up form. 5. The user inserts username, email and the password twice. 6. The user clicks on the *"Sign Up"* button 7. The system validates all the fields. 7.1 If the username or the email already exists, the system shows an alert. The user has to change them to sign-up. 7.2 If the password, the email, or the username doesn't respect the criteria, the system shows an alert. 7.3 If the two passwords inserted, don't match the system shows an alert. 8. The system redirects the user to his main page.

Continues from previous page

Table 2.1: Describes the main actor, the precondition, and the intentions of the Registration process. In the output section are described the steps to achieve the registration.

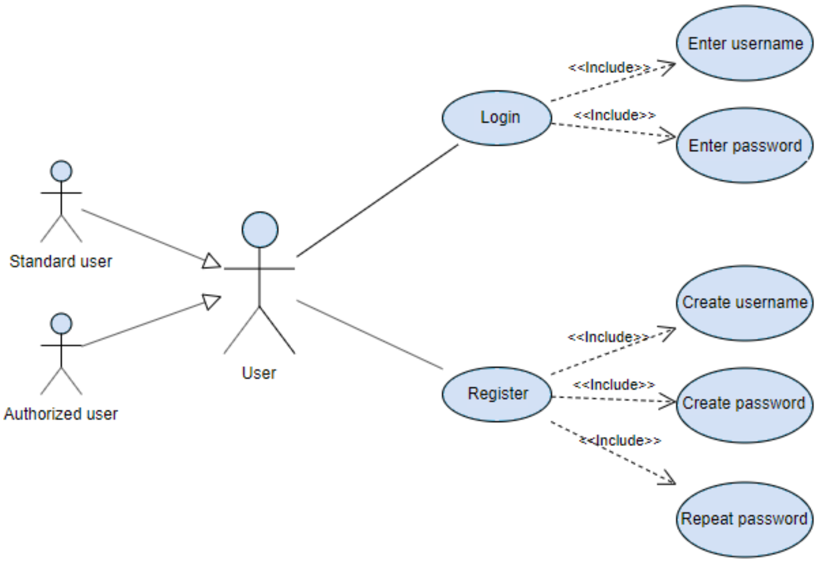


Figure 2.2: The use case describes the access process to the system, both for authorized and standard users. The detailed description is reported in the table 2.1 and 2.2

Use Case UC02: Login

RF02	User Login
Main Actor	User, both standard and authorized
Precondition	The user didn't have an open session, and he is already registered in the system.
Input	The user wants to access the system
Output	1. The user opens the application. 2. The system shows the main page. 3. The user selects *"sign-in"* button. 4. The system shows the sign-in form. 5. The user inserts username and password. 6. The user clicks on the emph"Sign In" button. 7. The system checks if the user exists and validates the two fields. 7.1 If the username or the password is wrong, the system shows an alert. The user has to change them to sign-in. 7.2 If the user with that username doesn't exists, the system shows an alert. 8. The system redirects the user to his main page.

Table 2.2: This table describes the login process in all of its parts. First, define the main actor and the preconditions. Then it explains what the user wants to do. In the last section are reported all the subsequent actions performed by the user and the system with the possible outcome.

Use Case UC03: Add new category

RF03	User add new category
Main Actor	Standard User
Precondition	The user has an open session, and he is a standard user.
Input	The user wants to insert a new category.
Output	1. The user selects, on the navigation bar on the left, the *"Add new Categories"* section. 2. The system shows the related page. 3. The user selects the form and inserts the category name. 4. The user selects the *"Category"* radio button. 5. The user clicks on the *"Insert"* button. 6. The system checks that this new category isn't already in the database. 6.1 If the category already exists, the system shows an alert. The user can insert a new one or continue to navigate in the system. 6.2 If the insertion succeeds, the system shows an alert to notify the user. The user can repeat this action as many times as he desires.

Table 2.3: Describes the main actor, the precondition, and the intentions when the user wants to add a new category. In the output section are described the steps to achieve the action and the possible outcome.

Use Case UC04: Add new subcategory

RF04	User add new subcategory
Main Actor	Standard User
Precondition	The user has an open session, and he is a standard user.
Input	The user wants to insert a new subcategory.
Output	1. The user selects, on the navigation bar on the left, the *"Add new Categories"* section. 2. The system shows the related page. 3. The user selects the *"Subcategory"* radio button. 4. The system shows the validate category already inserted in the database. 5. The user selects the form and inserts the subcategory name. 6. The user selects one or more parent categories between the ones rendered on the page. 7. The user clicks on the *"Insert"* button. 8. The system checks that this new subcategory isn't already in the database. 8.1 If the subcategory already exists, the system shows an alert. The user can insert a new one or continue to navigate in the system. 8.2 If the insertion succeeds, the system shows an alert to notify the user. The user can repeat this action as many times as he desires.

Table 2.4: Describes the main actor, the precondition, and the intentions when the user wants to add a new subcategory. In the output section are described the steps to achieve the action and the possible outcome.

Use Case UC05: Add new point of interest

RF05	User add new point of interest
Main Actor	Standard User
Precondition	The user has an open session, and he is a standard user.
Input	The user wants to insert a new point of interest.
Output	1. The user selects, on the navigation bar on the left, the *"Add new points"* section. 2. The system shows the related page. 3. The user selects one point on the map. 4. The system renders a marker on the selected point, it blocks the map and it shows a form. 5. The user inserts the place name and its description. It selects one or more categories and subcategories. 6. The user clicks on the *"Insert"* button. 7. The system checks that this new point of interest isn't already in the database. 7.1 If the place name already exists, the system shows an alert. The user can insert a new one or continue to navigate in the system. 7.2 If the insertion succeeds, the system shows an alert to notify the user. The user can repeat this action as many times as he desires.

Table 2.5: Describes the main actor, the precondition, and the intentions when the user wants to add a new point of interest. In the output section are described the steps to achieve the insertion and the possible outcome.

Use Case UC06: Generate a touristic tour

RF06	User generate a tour
Main Actor	Standard User
Precondition	The user has an open session, and he is a standard user.
Input	The user wants to generate a tour.
Output	1. The user selects, on the navigation bar on the left, the *"Create points of interest"* section. 2. The system shows the related page. 3. The user selects one or more categories. 4. The system renders the subcategories related to the category selected in the previous step. 5. The user optionally selects one or more subcategories. 6. The user clicks on the *"Generate Points"* button. 7. The system checks that there is in the database at least one point linked to those categories and subcategories. 8. In case of success: 8.1 The system renders a map with the points found. 8.2 It also shows on the map a path between the points with public transportation and a textual description of the journey. 9. In case of failure: 9.1 The system renders a map with no points. The user can repeat this action as many times as he desires.

Table 2.6: In the tour generation process are described the main actor, the precondition, and the intentions of the user. In the output section are described the steps to achieve the action and the possible outcome.

Use Case UC07: Review and validation of categories and subcategories

RF07	User reviews and validates categories
Main Actor	Authorized User
Precondition	The user has an open session, and he is a authorized user. The categories and subcategories are still not reviewed by no one.
Input	The user wants to validate a category inserted by standard users.
Output	1. The user selects, on the navigation bar on the left, the *"Review category"* section. 2. The system shows the related page. 3. The user sees two different columns, one for the categories and one for the subcategories. Each row has two buttons: *"Validate"* and *"Invalidate"*. The subcategory column also shows the reference to its parent category. 4. The user can choose to validate or invalidate each category and subcategory. 5. The user clicks on the *"Validate"* button associated to the category that he wants to validate. 6. The system stores the information on the database. The user can repeat this action as many times as he desires until there are categories to review. The user also can validate or invalidate a category after this first validation stage. It selects on the navigation bar on the left, the *"Validated categories"* to invalidate or the *"Invalidated categories"* to validate an already reviewed category

Table 2.7: Describes the main actor, the precondition, and the intentions when the user wants to review a category. In this scenario it can validate or invalidate a category, as well as do nothing. In the output section are described the steps to achieve the validation of a category. The process to invalidate a category is exactly the same, the only difference is the button clicked in the fifth step. The same process can be executed as well for the subcategories.

Use Case UC08 Review and validation of point of interest

RF07	Data fusion algorithm reviews and validates categories
Main Actor	System administrator
Precondition	The database contains points of interest still not validated.
Input	The main actor wants to validate the point of interest inserted by standard users.

Continued from previous page

Output	1. The administrator runs the algorithm 2. The system queries the database. 3. The algorithm checks for duplicate in the data 4. In case of success: 4.1 The system runs the data fusion algorithm 4.2 It stores the new data in the database. 5. In case of failure: 5.1 The algorithm terminates without any error. The system administrator can repeat this action as many times as he desires until there are point of interest to review. The frequency of the operation is arbitrary and it depends on the quantity of data inserted by the user. It is executed at least once to create a reliable database as basis for the application

Continues from previous page

Table 2.8: Describes the main actor, the precondition, and the intentions when the system administrator wants to review points of interest inserted by standard users. In this scenario it can execute the data fusion or not. In the output section are described in general the steps to achieve the data fusion.

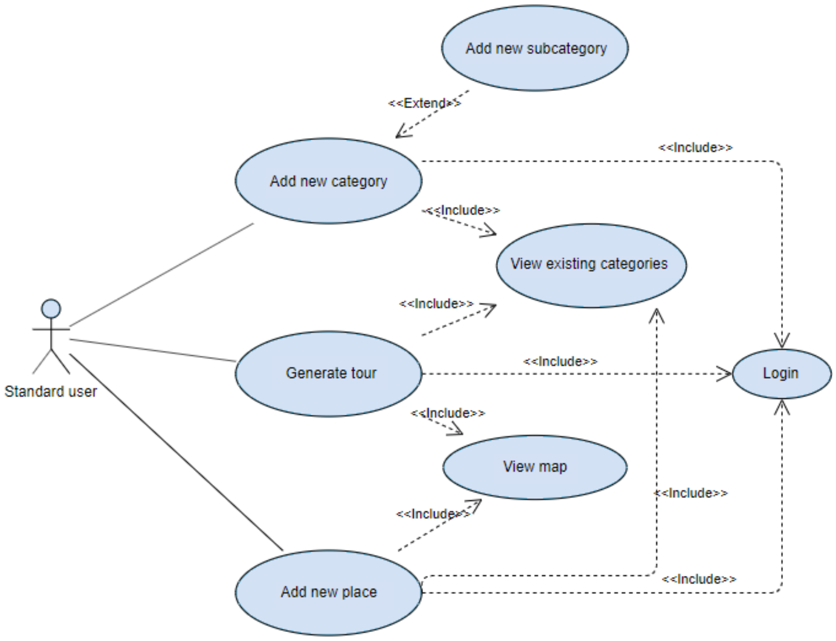


Figure 2.3: The use case describes all the actions that a standard user can execute in the system. All the actions require that the user is logged in and is of type standard. The detailed description is reported in the table 2.3, 2.4, 2.5 and 2.6

2.2.2 Non Functional Requirement

Non-functional requirements state how the functional requirements need to be achieved. In the following section, they are presented using tables that specify the identification number, the type, and the description.

Usability

RNF01	Usability
Description	The UI design is attractive for the user. The combination of shapes and colors made the interaction more natural and intuitive. Moreover, UI follows the logic of similar applications making the interaction more comfortable and error-free. The users don't need the training to use the application.

Table 2.9: Describes the usability requirement of the application.

Maintainability

RNF02	Maintainability
Description	The developer has to easy correct defects or their causes, prevent unexpected working conditions, and repair or replace components without having to rewrite the whole code. Besides, the useful system life must be maximized along with its efficiency, reliability, and safety.

Table 2.10: Describes the maintainability requirement of the application.

Performance

RNF03	Performace
Description	The system must have a short response time both in the UI rendering and in the interaction with the backend. It should be acting in the same way with different users' load. The database size and the length of the path for the tour to calculate don't have to affect significantly the response time that should be under 5 seconds.

Table 2.11: Describes the performance requirement of the application.

Platform compatibility

RNF04	Platform compatibility
Description	The software execution isn't affected by the platform where it runs. The system must work well on every web browser. This non-functional requirement also includes the portability one, in terms of abstraction between the application logic and the system interface to reduce the development cost.

Table 2.12: Describes the platform compatibility requirement that also satisfies the portability one.

Reliability

RNF05	Reliability
Description	The system can work under a defined condition for a specified period. It reflects design perfection and resistance to failure of a component or the whole application in terms of probability of success. It is strictly related to availability and the users' behaviors. The possible failure of the system can be related to human interaction, maintenance-induced failure, and software failure.

Table 2.13: Describes the reliability requirement of the application.

Robustness

RNF06	Robustness
Description	The system has to deal with errors during the execution and erroneous input. It has to notify the user and continue without generate misleading behaviors.

Table 2.14: Describes the robustness requirement of the application.

Operability

RNF07	Operability
Description	All the system parts, frameworks, databases, and UI, have to work together to accomplish the common task. The system works without needing application restart or any other non-automated interventions. It is closely related to reliability and maintainability.

Table 2.15: Describes the operability requirement of the application.

Security

RNF08	Security
Description	The data integrity is mandatory for the security of all the information stored in the database. The system has to implement access restrictions and separation of jurisdiction between the users. Further, it has to implement a mechanism to prevent informatic attacks.

Table 2.16: Describes the security requirement of the application.

2.3 Architecture and Design

This section explains first the technologies used, then it presents in detail the system architecture and the design with a focus on the frameworks used. Subsection 2.3.2 describes first the general architecture of the project. Then it reports the detailed architecture used for the back-end and the front-end development. Subsection 2.3.3 includes the database schema and the data representation used in the project.

2.3.1 Technologies

Before discussing about the system architecture and the developing process, the technologies used must be introduced. The ones employed in the frontend will be presented first, followed by the backend technologies and, in the end, the database.

React is a javascript library purposely designed for the UI creation with the goal of have an intuitive language capable of building dynamic and increasing complex UIs. It is a declarative and stateless language made of immutable and reusable elements nested together. React supports several frameworks and external plugin to manage routing, API interaction, and more sophisticated features [53] [54]. Due to is nature, React well fits into the Model-View-Controller software design pattern. This paradigm decouples the presentation logic and the data representation, enhancing code reusability, and the usage of any backend language without restrictions.

Although there are several libraries for web application developing, the choice fell on React because:

- Unlike JQuery it excludes a direct operation on the DOM. Instead, it provides a virtual DOM to the developers to guarantee the best performance
- It is less complicated compared to AjaxJS
- It applies the Separation of Concerns. React isolate state and code based on their scope of use to make the components more reusable at the expense of the inherent separation of the MVC pattern.

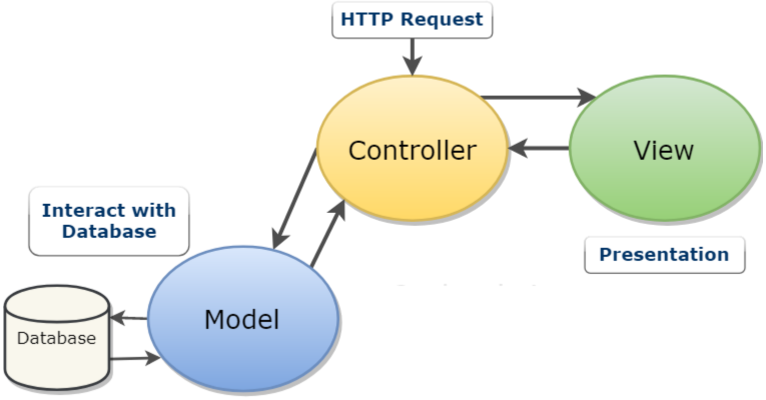


Figure 2.4: Model-View-Controller design pattern.

- React's component uses the JSX syntax allowing the developer to include HTML in React.

React Redux is a state management library that synchronize state and UI components to ensure the availability of the state information for all the components in the application tree. It is a predictable state container that uses a central data store to manage the state of the application. The store act as a source of truth on which components can rely on state management [57].Three basic concepts are Redux's cornerstone: store, action, and reducers. The whole app state is stored in an object tree inside a single store. The store makes it easier to debug and inspect the application. The only way to change the state tree is to emit an action, an object describing what happened. All changes are centralized and follow a strict order avoiding race conditions. Reducers are functions that specify how the actions transform the state tree. It takes the previous state and generates a new one without modifying the previous one [58]. This patter can be challenging for small applications, but it scales to large and complex apps very well. The interaction between the three components is shown in the following picture.

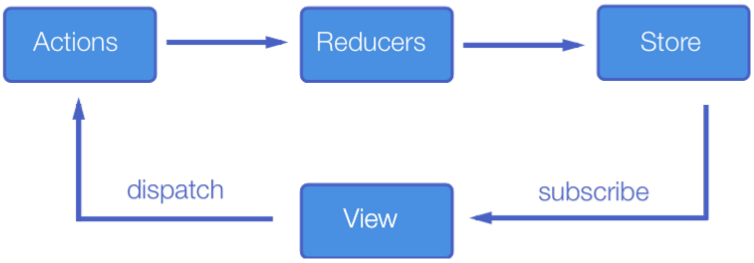


Figure 2.5: Interaction between store, action, and reducer in React Redux.

React Router is an API for React application that allows the creation of a single-page web application with dynamic navigation and it embraces the React's philosophy of component-based architecture. The difference between static and dynamic routing stands where the routes are declared. Static routing is declared when the app is initialized outside of a running application. Dynamic routing instead, takes place as the app is rendering. There are three main components: router, route matching, and navigation. The router component is the core of the application; it must be the outer component given to the DOM in the render method. It creates a history object and enables these components to interact with it to manage the routing. The router component is in its turn divided into two parts: Route and Switch. The Route is mandatory, and it contains the path and a component to render when a location matches the Route's path. The Switch can be omitted, but it is useful to group the different Routes and iterate among them to do the matching. Navigation links are Link components to create links with styling attributes [56]. React Router also enables the user to utilize the browser button, maintaining the correct application view.

Node.js is a cross-platform and open-source Javascript runtime environment built on Chrome's V8 engine. It allows the usage of javascript both on the client and server-side letting to implement the Javascript everywhere paradigm, unifying the Web application development under a single programming language. This choice makes development faster and bug-fixing more effective. A Node.js application runs on a single process without creating a new thread for each request going against the model of the classical web server. It accesses the operative system resources through the event-driven model. This model is based on a simple concept: every time there is a change an event is fired. Node.js uses a non-blocking paradigm, making the blocking behavior the exception; in this way, it avoids the waste of memory and CPU cycles, in favor of increased performance. This characteristic makes it perfect for the creation of a data-driven application where the I/O is put to the test. Node.js is very flexible in terms of development architecture because it has few dependencies and loosely Node.js came along with a standard package management useful for download and installed several plug-in and dependencies. Even if npm is the largest javascript repository, its quality is still under verification. The lack of a control mechanism on these modules leads to a careful choice of them from the developer [60] [61]. Node.js is a low-level platform, and there are thousands of libraries build on it to make the development easier.

Express.js is the de facto standard server framework for Node.js. It is an unopinionated framework that has few restrictions on how to achieve goals and which component to use. It provides common tasks not directly provided by Node as routing and middleware integration. Routing determines how the server responds to a request to a particular URI and HTTP request method. Route path can be string, string pattern, or regular expressions. The server listens for requests on a specific route and method when it detects a match execute the callback function associated with it. Route handlers behave like a middleware with the only exception of the next()

function, that can be used to skip the remaining route callbacks. The callback function has to send a response back to the client to avoid letting the request hanging. The express.Router class creates modular route handlers that is a complete middleware [59]. Middleware functions can access the request-response object and the next middleware. They can execute code, change the request-response object, end the function exiting the request-response cycle and call the next middleware function in the stack. Express can use different types of middleware:

- Application-level is bind to the app object through the app.use() function. It takes three arguments request, response, and next.
- Router-level is bound to the router object and works in the same way as the previous one.
- Error-handling is bind to the app object but takes four arguments: error and the usual ones.
- Built-in includes the JSON parser for incoming requests.
- Third-party is installed using npm and is applied using the app.use() function.

In the specific case of the thesis, express interact with three leading middleware: cors, passport, and sequelize. Cors implement the cross-origin resource sharing a security policy that allows the server to specify who can access it and which HTTP request use. Passport is an authentication middleware designed to manage both traditional and through an OAuth provider as Facebook, Twitter, and Google. Sequellize is an Object-Relational-Mapper for Node.js. Its basic idea is to abstract the complexity of interfacing with the database writing SQL queries using the object-oriented paradigm instead of pure SQL [62].

PostgreSQL is an open-source object-relational database management system. It uses a client/server model where the two processes cooperate and communicate through a TCP/IP network connection. The server process, called Postgres, manages the database and performs actions on it on behalf of the client. The client application can have different natures: web server, text-oriented application, or database management tool. PostgreSQL supports standard SQL and adds more complex features that simplify management and prevent data loss and corruption [62].

Python is a high-level programming language that can rely on several resources and libraries. It is a highly versatile language that hides from the user its complexity and allows fast application development. This language supports different programming paradigms, it is no variable dependency and it outperforms in terms of execution and development speed. All these features make it the best language to code algorithms.

2.3.2 System architecture

The system architecture is composed of sub-systems that work together to implement the overall system. This project is based on a client-server architecture, shown in the figure 2.6, where clients can use any desktop computer with a modern browser. The server can be unique or replicated, and it communicates with a PostgreSQL database and the OTP service. The data fusion algorithm is not executed every time there is an insertion in the data base, but it is executed periodically.The system administrator triggers the execution when the number of places inserted exceeds a fixed threshold. The client-server architecture has several advantages that fit the project scope.

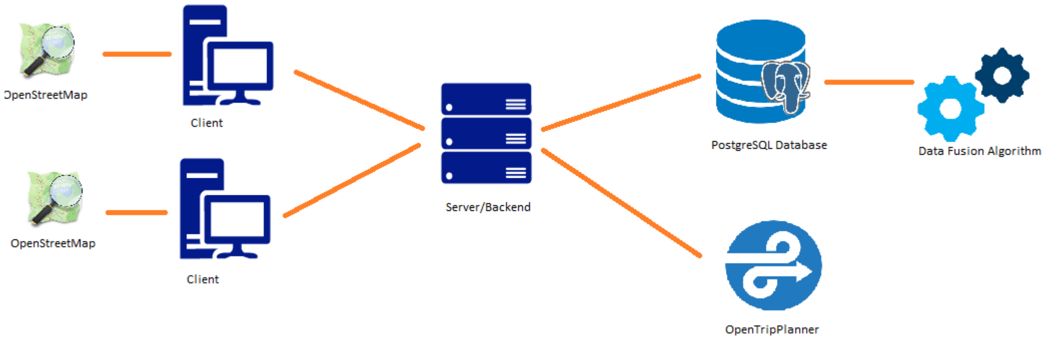


Figure 2.6: Client-server architecture of the project. It shows the connection between the components and also includes the server connection with the PostgreSQL database and the OpenTripPlanner service. Each client has its instance of the OpenStreetMap. It shows the interaction between the application and the data fusion algorithm

First of all, the system ensures the separation between the presentation and business logic. Business logic includes all the processes invisible to the user that are the core of the application; the presentation logic consists of the UI representation instead. This separation ensures uncomplicated maintainability because any changes in the back-end don't affect the presentation layer, and the changes are centralized. Moreover, the client can access the system without any specific configuration. The front-end development can be uncoupled because the communication between client and server should agree only on the data representation and the communication protocol. It is a cheap architecture in term of human cost, the security and the maintainability are centralized, so fewer support staff is required [48].

Back-end architecture

The back-end architecture is divided into two subsystems: the first one regards the services related to the web application and the other one concerns about the data fusion algorithm. The application back-end follows the REST architectural style. The representation state transfer defines a set of constraints for web services creation that provides interoperability between systems over the internet [49] [50]. A RESTful API is based on the following guideline:

Separation of concern between client and server enables the components to grow autonomously. It improves client portability across different platforms, and it increases the server scalability by streamlining its components.

Responses cacheability can be enabled by clients or servers to improve performance or disabled to avoid to receive stale data.

Stateless server protocol binds the session state to the client. In each request sent, the client must include all the information required for the communication.

Layered system used to improve scalability and security. This feature is liked to the separation of concern explained before. The client is not aware of who he is communicating with because additional layers can be added to the server to improve load balance, security policies, or caching mechanism. Furthermore, the server can call other web services to introduce additional functionality.

Uniform interfaces to simplify and decouple the architecture. Unique URI identifies individual resources that are conceptually separated by the data representation returned. The message returned includes enough metadata information to be manipulated or deleted. A standard interface like HTTP is used for communication.

RESTful systems aim for fast performance, reliability, and scalability, as explained before. Following this architectural style, components can be managed and updated without affecting the system even while it is running.

The data fusion system is independent of the application back-end because it interacts only with the database instance. It is executed locally by the system administrator, and it only depends on the data inserted in the database. This system is seen as a black box by the application and the system administrator. It is composed of three sequential modules that handle the problem at three different levels: schema, tuple, and values, as showed in figure 2.7.

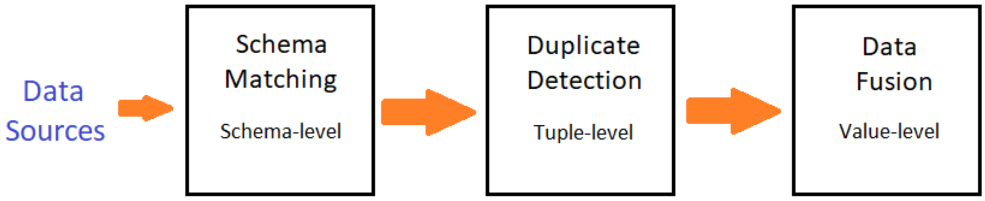


Figure 2.7: High-level representation of the data fusion algorithm. It shows the three different modules of the algorithm and the interactions between them.

Front-end architecture

The implicit connection between React and the Model-View-Controller design pattern described in section 2.3.1 leads to think of embracing the MVC design style. The problem with MVC is the bidirectional communication that made the code less maintainable and challenging to debug. Facebook, which developed React, presented a new architecture called Flux for building React web application with unidirectional data flow [41]. Four elements are the basis of Flux: Action is an object with property and data, Store contains the application state and logic, View listen to store changes and re-render accordingly, and dispatcher is a process implementing actions and callback functions [51]. Based on this architecture, the thesis is developed using Redux, an architecture based on Flux that differs from it for the absence of the dispatcher and the concept of data immutability.

In the project, React is the View layer, and Redux is the Store.The overall architecture with components and relationships is represented in figure 2.8.

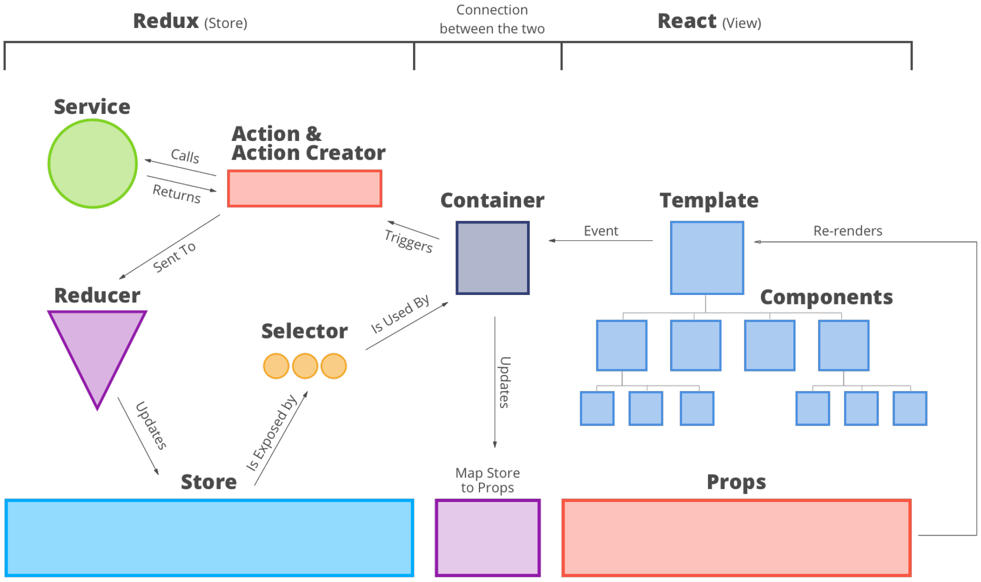


Figure 2.8: React+Redux architecture details with highlights on the components and the relations between them.

The container is an abstraction layer that allows Redux and React to be decoupled, therefore, to change and grow autonomously. It translates the changing in the Redux state to React props, and when in React, an event happens, it triggers the corresponding action in Redux. A Component is a piece of code that renders and uses a set of props passed by its parent that can be a component or a template. A template is merely an additional abstraction level. The Action is an object containing the type of action, and the state changed because of the action. It communicates with the server to execute the action required by the React event, and it sent the result to the Reducer. The Reducer is the only component allowed to change the state creating a new one. The selector is an abstraction level between the Store and the Container [52].

2.3.3 Data representation

The data used by the application is stored in a relational database. An ORM technique can be used to simplify the development and increase maintainability. The Object Relational Mapping technique allows the developer to manipulate and query the data in the database using an object-oriented approach. It implements a data layer that creates a "virtual object database" and acts as a translator between the object-oriented language and the database, reducing the need for SQL language. The ORM translates the logical object representation in a suitable form that can be stored in the database preserving the object properties [47].

This project uses Sequelize, an ORM promise-based for Postgres, MySQL, and other relational databases [33]. The first step is to create a Sequelize instance to connect to the database. Then the developer has to define for each database table a model in Sequelize. By default, the *createdAt* and *updateAt* field are inserted in the model to track the modification done on the table. The application will interact with the database directly with this model. After the model creation, it must be deployed in the database using the Sequelize migration.

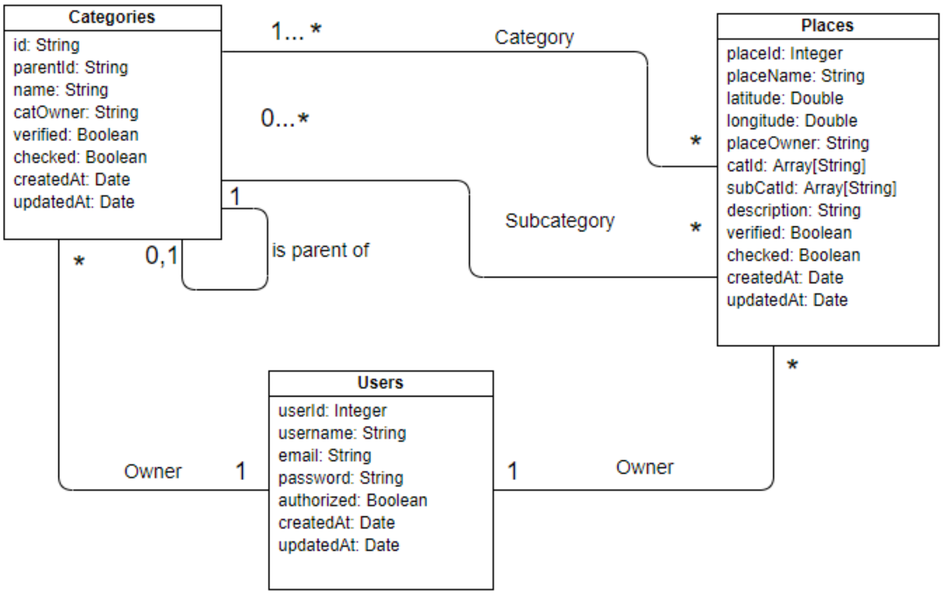


Figure 2.9: Schema of the project database.

Figure 2.9 shows the database model created using the migration. The Categories and Places tables are linked with the Users one through a foreign key on the user *email.* The Places table also has two more foreign keys referring to the categories and subcategories stored in the Categories table. The Categories table includes both categories and subcategories that are defined using a foreign key applied between the *id* and *parentId* attributes.

2.4 Development

This chapter details the implementation of the project, explaining the functionality and the most significant element employed. It includes UML diagrams and snapshots of the application both of code and UI. Before entering the implementation details, let's introduce the development environment.

The development environment is a set of processes and tools used to develop source code and debug it. WebStorm is the JavaScript IDE created by JetBrains that covers all the modern JavaScript languages as React and Node.js. It provides to the developer intelligent code completion and on-the-fly error detection. For these features and the built-in integration with Git and GitHub, it was chosen for the development of the entire project. Git is a revision control system and a source code management that was used in the project to keep track of its history. It was used together with a GitHub account where the project information was stored. GitHub is a web-based hosting service that uses Git.

2.4.1 Homepage

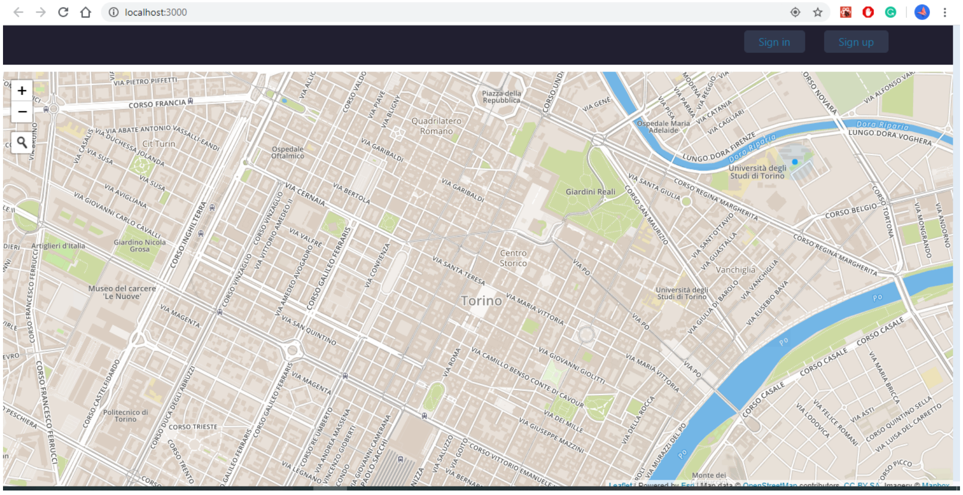


Figure 2.10: App homepage

The first view presented to the user is a map showing the point of interest, inserted and already validated, as markers on the map. This is a public page where both registered and non-registered users can access it. In the top-right corner, there are the Sign-in and Sign-up button to allow access to the restricted part of the system. In the development, Leaflet was used to show the OSM map in the application. Leaflet is an open-source Javascript library designed to ensure simplicity, performance, and usability. It is highly customizable and supports several plugins to extends its features [63]. When a user navigates to the homepage, the browser asks him to allow the localization. If it is enabled, the map will zoom in on his position, and a localization marker appears. Otherwise, nothing happens, but the system could not work correctly in the next phase of tour creation. The user can navigate on the map, and when he goes on a marker, the system shows the place name as shown in figure 2.10.

The system to render the point of interest on the map, interact with the server. The only API call that can be executed by an unregistered user is the *getAll* one showed in figure 2.11. The system asks for the points of interest already validated by the authorized user. Since this page is public, the server returns only the name and the geographic coordinates of the point without any information about who inserted it.

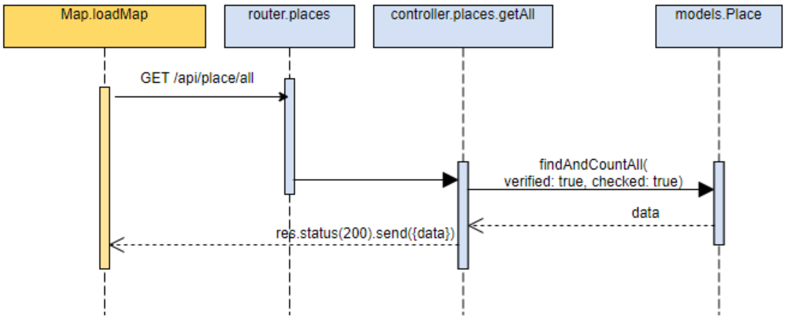


Figure 2.11: Sequence diagram for points of interest retrieval.

2.4.2 User login

As said before, the user to access all the application features has to log in and start a new session. To handle the authentication, a specific service was developed using JSON Web Tokens. JWT is a compact and self-contained solution for transmitting information between parties that can be verified and trusted because it is digitally signed. JWT is composed of three different parts: header, payload, and signature. The header contains the specification for the type of token and the algorithm used. User-related information and token metadata are stored in the payload. The signature is used to verify the token, and it is created trough the encoded header, the encoded payload, and the secret known by the token issuer. Any changes in the token modifies its signature and invalidate it. The JWT is used as an access token for the protected features of the system.

Every time the user enters the application, he receives an access token. For design choices, the client stores the access token and sent it to the server through cookies. Figure 2.12 shows all the calls executed by the system in the login process. This scenario represents the login of an already registered user that inserts all the correct parameters, explained afterward. The *signInValidation* function validates the data inserted by the user and returns the error if it founds some illegal data. It the case represented, the user has inserted the correct ones, so the validation returns an undefined error that means success. The JTW token and the username are stored by Redux in the client-side.

The user interface showed in figure 2.13 follows the style of the most popular software to make the user comfortable with a familiar scenario. The user as to insert the username and the password, then he selects the *SignIn* button. In case he miswrote any of the two

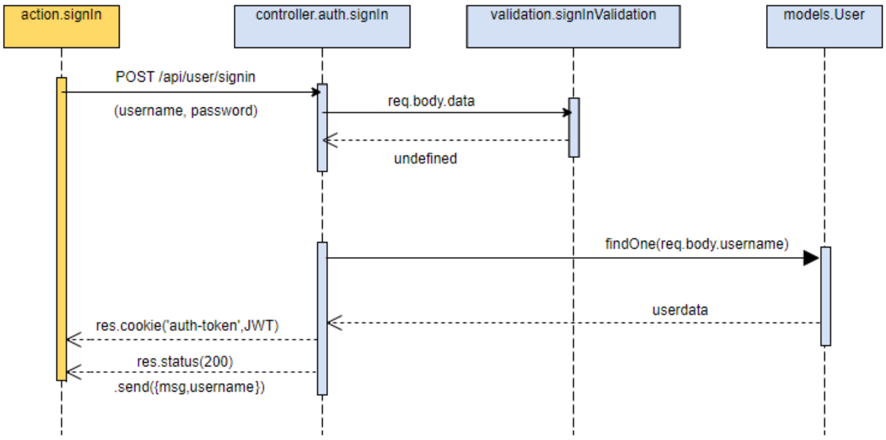


Figure 2.12: Sequence diagram of the login process. The first lifeline on the left it the one related to the client. The others are all calls within the server.

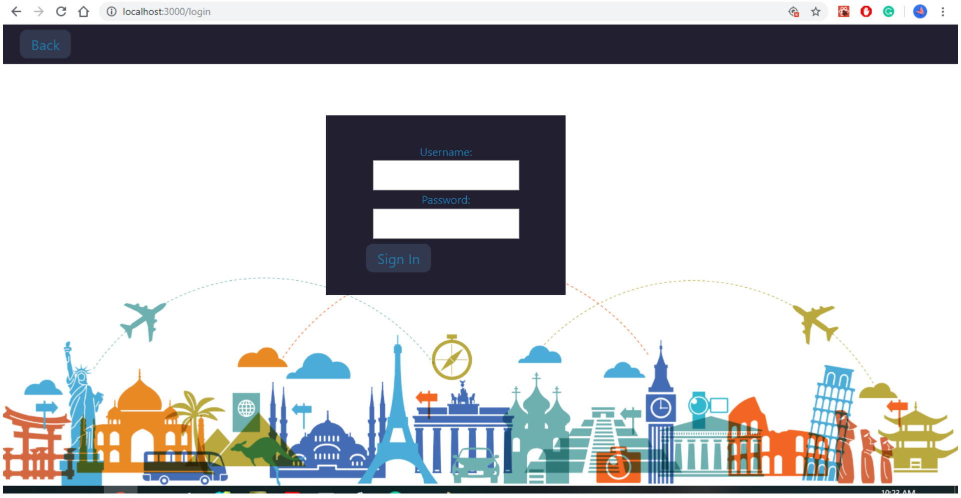
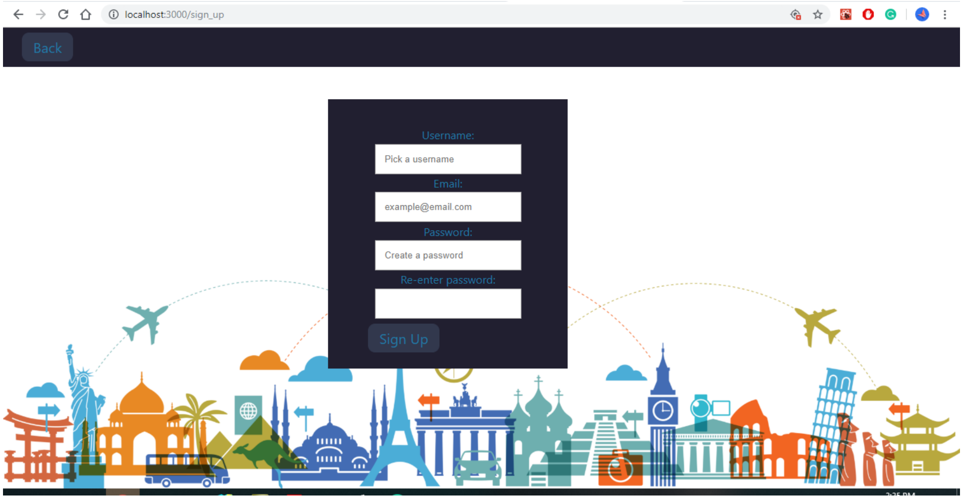


Figure 2.13: App login page

fields, leaves one of them empty, or inserts illegal code in them, he is notified with an alert. In case the login succeeds, the main page connected with his state of normal or authorized user is automatically rendered.

2.4.3 User Registration

During the registration process, the user must follow some specifications for username and password creation. The username must be at least 6 characters long and contain only alphanumeric characters and underscores. The password requires at least 8 characters, including an uppercase letter, a digit, and a special symbol. It has to be repeated another time for safety. The form used for the registration process also uses a structure similar to the other popular website. In case any field showed in figure 2.14 is empty or doesn't respect the constraint explained before, the user is notified.



localhost:3000/sign_up

Back

Username:

Pick a username

Email:

example@email.com

Password:

Create a password

Re-enter password:

Sign Up

Figure 2.14: App registration page

When the user hits the *SignUp* button, the client sends a POST request to the server. The input validation process is called and permit to proceed with the new user insertion. This process showed in figure 2.15 is practically the same described in figure 2.12. The only differences are the URI of the request, and the parameter send with it. If the user inserted a username already taken or the email already exists, the user is notified to allow him to change them and start the process again. When the process is completed successfully, the user is directly redirected to his homepage and receives with the response the JWT and his username.

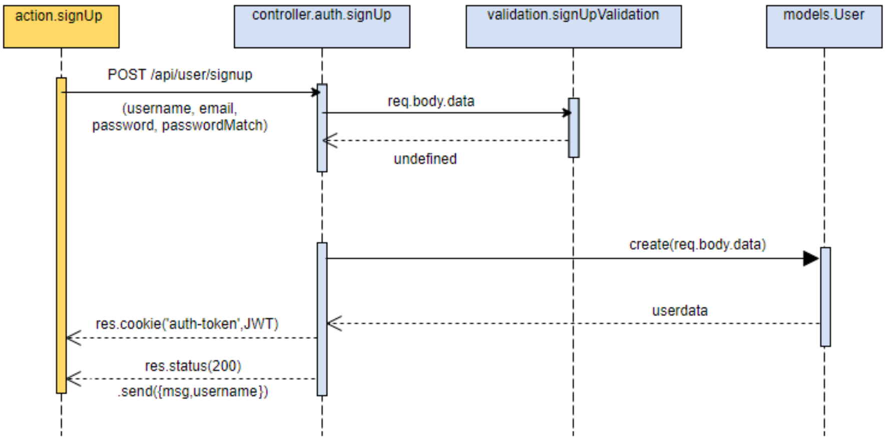

action.signUp
controller.auth.signUp
validation.signUpValidation
models.User
POST /api/user/signup
(username, email, password, passwordMatch)
req.body.data
undefined
create(req.body.data)
userdata
res.cookie('auth-token',JWT)
res.status(200) .send({msg,username})

Figure 2.15: Sequence diagram of the registration process.

2.4.4 Token Validation

The system has to validate the token every time the user requests a private route. The system specifies three functions to verify the different access privileges. The *verify.normal* is used to check if the user that wants to access the resource is registered and is a standard user. The *verify.authorized* one examine if the user has the authorized privilege. There is also a general function called *verify.general*, used to check only if a user is registered in the system because both types of users can access some resources in different contexts.

Figure 2.16 reports the sequence diagram for the token validation. First, the server has to retrieve the JWT from the request cookies. In case the function found it, the token is validated with a proprietary function of JWT that decrypts the payload and takes the *userId* previously stored there. The server then queries the database to find if the user corresponding with that id is registered and notifies the user accordingly. If the user tries to access unauthorized resources, the system automatically redirect him to the login page.

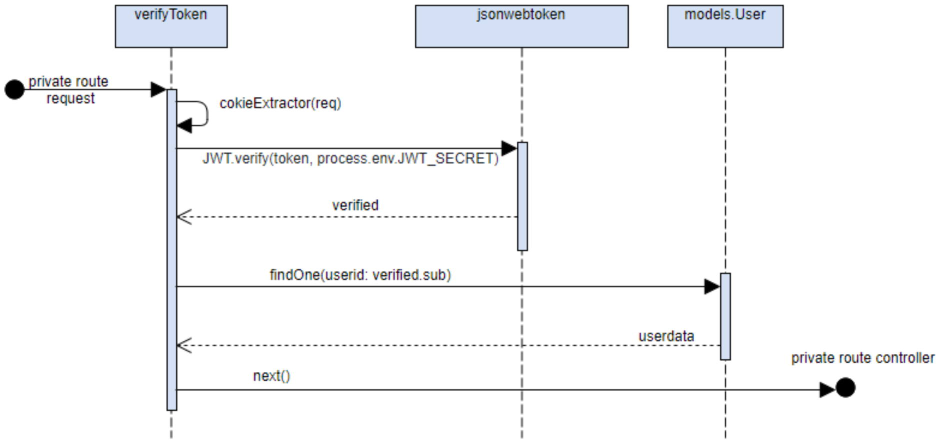


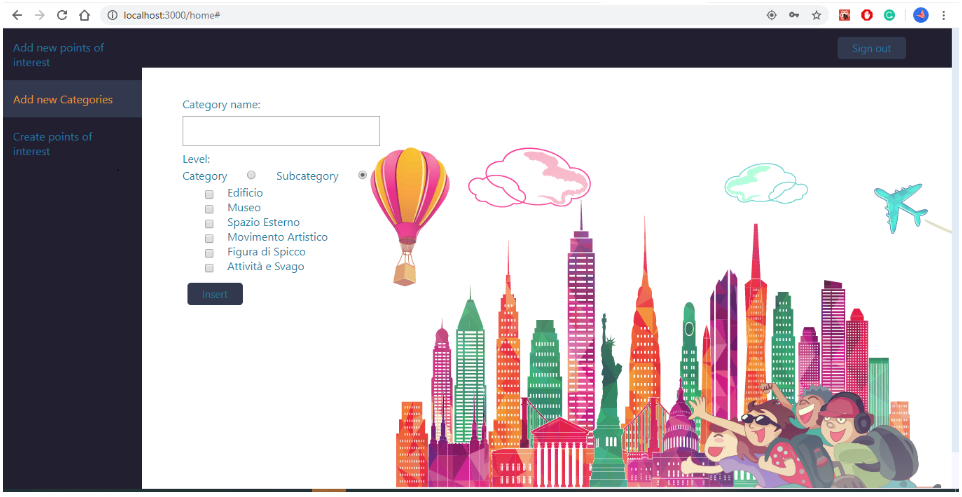
Figure 2.16: Sequence diagram of the authentication process.

2.4.5 Standard user features

The standard user is the one that can add valuable information on the application and the one with more sophisticated features. On his homepage can choose three different activities explained in the following subsections, or he can logout using the *SignOut* button in the header showed in figure 2.17, 2.20, 2.23 and 2.25 using a straightforward process. The *SignOut* button triggers an action on the Redux reducer and sets the global state to the initial state before the login.

Insertion of a new Category

The first feature of the software is the categorization of point of interest. By default, the system already has six categories: Building, Museum, Activity and Leisure, Outdoor, Leading Figure and Artistic Movement. Each of them has several subcategories that can be liked to one or more categories. Even if the categories cover many touristic aspects, the possibility for the user to add a new one was inserted.



Category name:

Level:
Category ○ Subcategory ●
□ Edificio
□ Museo
□ Spazio Esterno
□ Movimento Artistico
□ Figura di Spicco
□ Attività e Svago

Figure 2.17: Add new Category tab of the application. The user has already selected the subcategory field, and the system has rendered the possible parent category between choose.

The user to insert the category selects the *Add new category* tab in the left navigation bar showed in figure 2.17. He can choose between the insertion of a category or a subcategory. In the category case, he inserts the name, selects the Category button, and clicks on *Insert*. The sequence diagram, figure 2.18, exposes the API calls executed. First, the system checks if the user is a standard one and is registered, and it validates the input. Then it queries the database three times to ensure that there isn't any category with that name, to count the existing categories to give a sequential id to the categories, and if any error occurs, it inserts the new category. In case of success or error, the user is notified.

For the subcategory insertion, the user has to select the subcategory button; the system requests the main categories through a GET request following the steps reported in figure 2.19. The user then selects one or more categories and continues as explained for the category insertion. To get the categories, the system checks first if the user is registered in the system, and afterward, it retrieves from the database the category already validated. The process to insert a subcategory is the same; the only difference is the request URI, and the parameters passed with it.

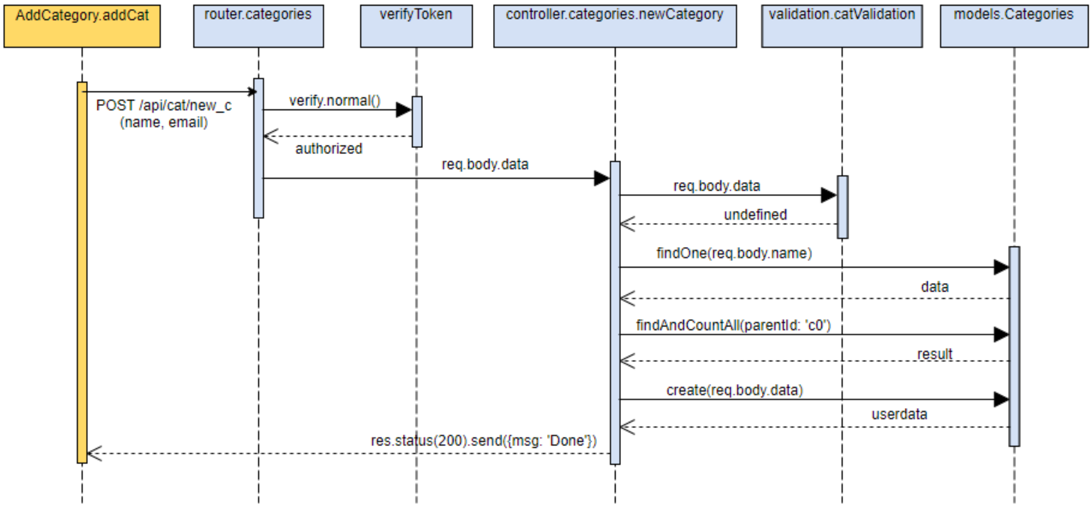


Figure 2.18: Sequence diagram of the category insertion process.

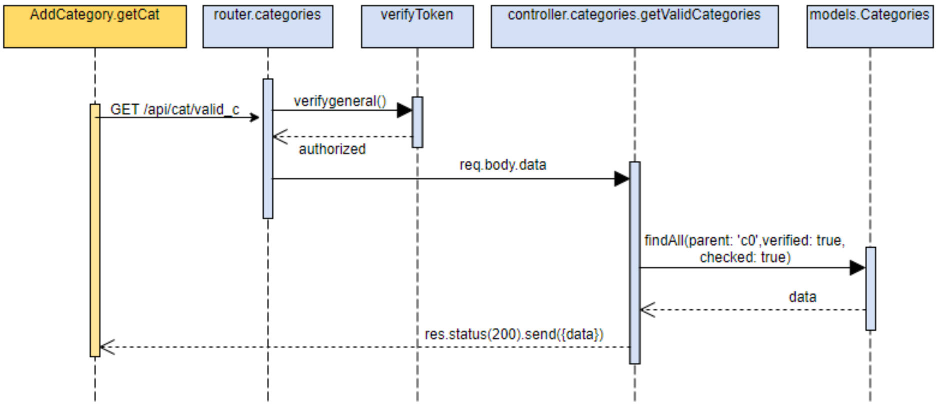
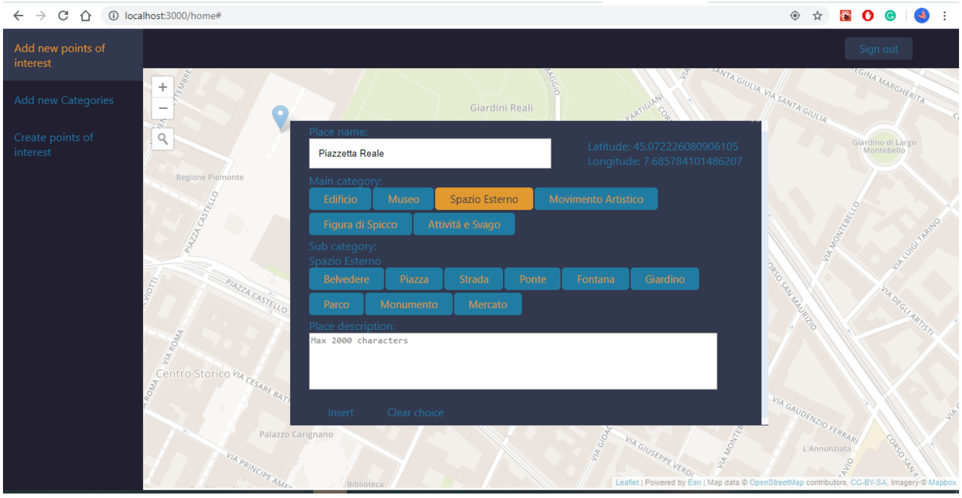


Figure 2.19: Sequence diagram of the process for get the categories already validated by the authorized user.

Insertion of a new point of interest

One of the central purposes of the application is giving value to user knowledge and ensure the most trustworthy and free journey. The idea is to insert points of interest that cover not only the major sightseeing but also hidden or less know places. The user can do it within the website, selecting the tab Add new point of interest. If he has enabled the localization, it sees a map zoomed on his position; otherwise, he sees the general map, as explained in section 2.4.1. In this case, the points of interest already validated are represented with orange markers; the new point selected by the user is a blue marker.

When the user selects a point on the map, the system blocks the map and renders a popup



localhost:3000/home#

Add new points of interest

Add new Categories

Create points of interest

Sign out

Place name:

Piazzetta Reale

Latitude: 45.072226080906105
Longitude: 7.685784101486207

Main category:

Edificio | Museo | Spazio Esterno | Movimento Artistico | Figura di Spicco | Attività e Svago

Sub category:
Spazio Esterno

Belvedere | Piazza | Strada | Ponte | Fontana | Giardino | Parco | Monumento | Mercato

Place description:

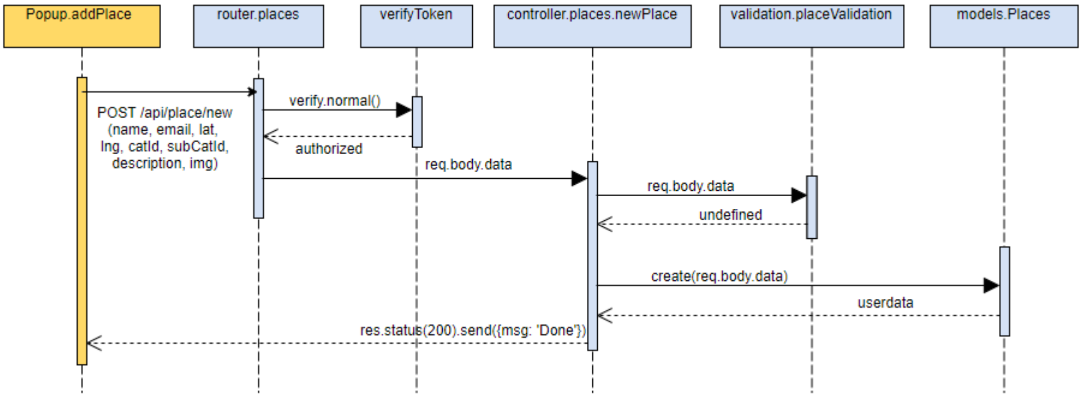
Max 2000 characters

Insert | Clear choice

Giardini Reali
Regione Piemonte
Piazza Castello
Centro Storico
Palazzo Carignano
Giardino di Largo Montebello
Leaflet | Powered by Esri | Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © Mapbox

Figure 2.20: Add new point of interest tab of the application. The user has already selected the point on the map and the *Outdoor* category,and inserted the place name. The subcategory rendered by the system are the one related to the *Outdoor*category already selected.

form, as in figure 2.20, where the user can insert name, categories, subcategories, and a description. Between them, the name and at least one category must be inserted. The system shows in the form latitude and longitude where the user clicked and store them with the username and all the other information in the database, as reported in figure 2.21.



Popup.addPlace
router.places
verifyToken
controller.places.newPlace
validation.placeValidation
models.Places
POST /api/place/new (name, email, lat, lng, catId, subCatId, description, img)
verify.normal()
authorized
req.body.data
req.body.data
undefined
create(req.body.data)
userdata
res.status(200).send({msg: 'Done'})

Figure 2.21: Sequence diagram of the point of interest insertion process.

A post request is sent to the server that validates token and input and creates an instance on the database. The flow of the process is similar to others already explained in the previous sections. But this one requires two other API calls. The first one is included in the map creation, as explained in the 2.4.1 section, for the retrieval of point of interest already validated. The second API call, described in figure 2.22, can be repeated every time the user selects a category button to retrieve the subcategories linked to the selected category. There is a POST request to the server that verifies the token and asks the database for the subcategories validated that have as parent the one stored in the request body.

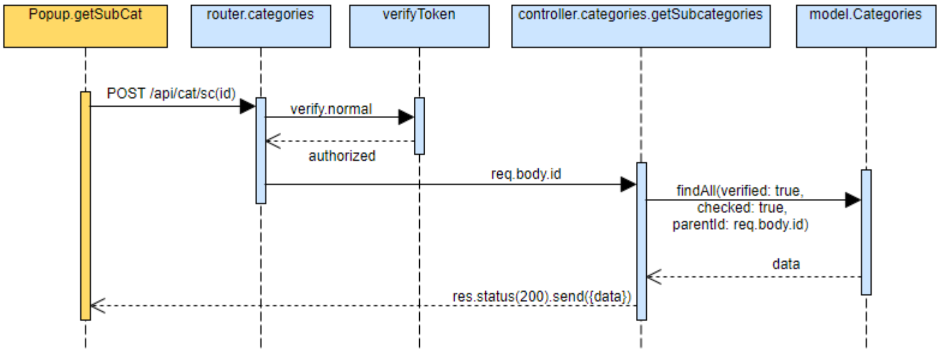


Figure 2.22: Sequence diagram of subcategory request process.

Tour Creation

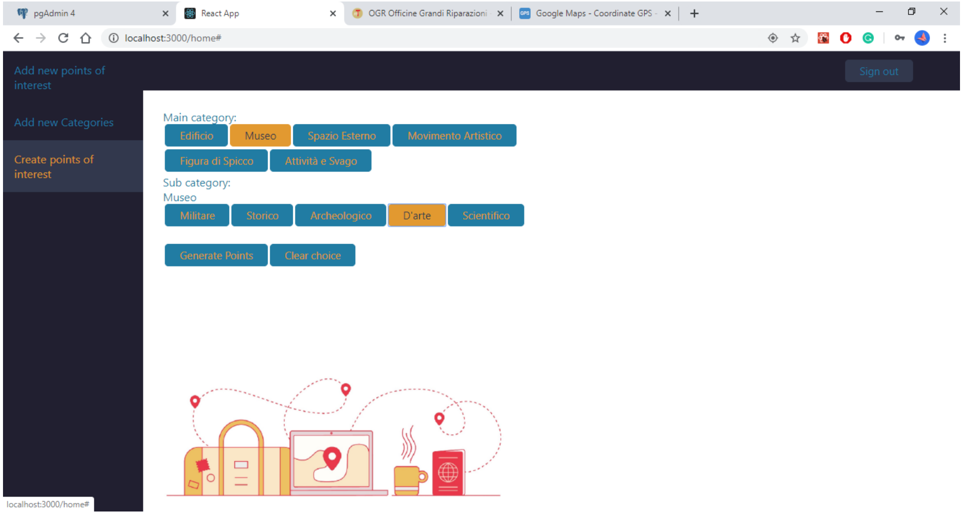


Figure 2.23: Creates point of interest tab of the application. The user has already selected the Museum category and the system renders the subcategory related to it. The user selects the Art subcategory.

The last, but not least feature available for a non-privileged user is the customized tour creation. The user navigates to the *Create points of interest* tab on his homepage. The system renders the first screen, reported in figure1, where the user sees and chooses the validated categories. To do so, it executes the GET call explained in section 2.4.5 figure 2.19. For each category selected by the user, the system executes the same call to the API explained in section 2.4.5 figure 2.22. The user can choose many subcategories as well. To help the user in the visualization of the categories chosen, the system renders them with a different color. The user can select and deselect each category manually or undo all by selecting the *Clear choice* button.

When the user is satisfied with the filter selection, it clicks on the *Generate points* button, and the system executes the API call reported in figure 2.24. As usual, the server checks if the user has the right privileges to execute that call. It queries the database to found the places that were categorized with the tag selected by the user. The result of the database aren't returned directly to the user because it is a waste of bandwidth to execute another call to the server for the tour generation. The server automatically calls the *sortPoint* function passing the data retrieved from the database and the current position of the user. In this scenario, the user must allow the localization to use the service.

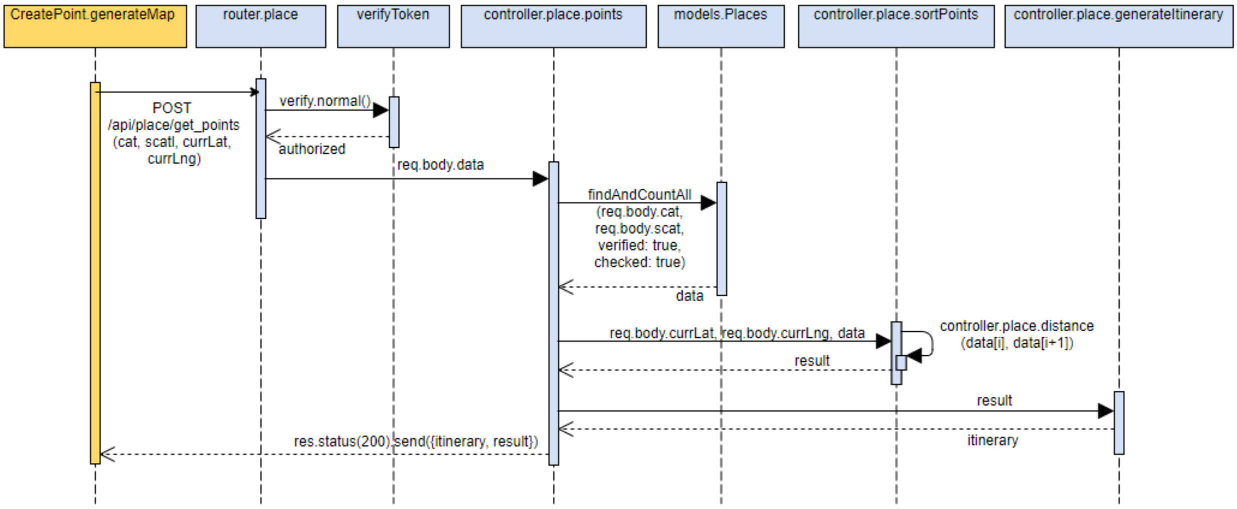
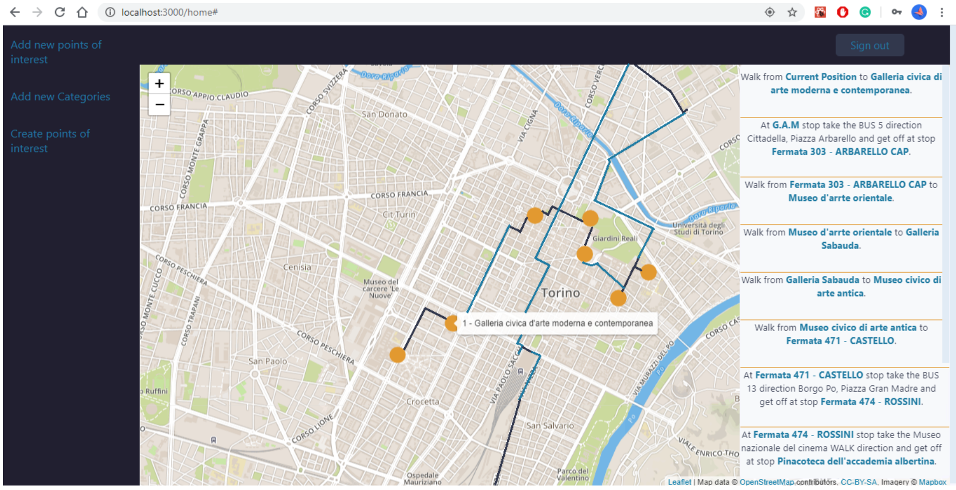

CreatePoint.generateMap
router.place
verifyToken
controller.place.points
models.Places
controller.place.sortPoints
controller.place.generateItinerary
POST /api/place/get_points (cat, scatl, currLat, currLng)
verify.normal()
authorized
req.body.data
findAndCountAll (req.body.cat, req.body.scat, verified: true, checked: true)
data
req.body.currLat, req.body.currLng, data
controller.place.distance (data[i], data[i+1])
result
result
itinerary
res.status(200).send({itinerary, result})

Figure 2.24: Sequence diagram of the tour creation process.

The sort function uses a brute force approach because during the design was assumed that point categorization was focused and careful to give the best performance possible. This function computes in the first iteration, the distance between the current user position and all the points returned by the database. The nearest point goes in the array as the second element, and the process iterate again. This time the system computes the distance between the second element of the array and the remaining point, putting the new solution in the third position. The function ends when there aren't any more points on which iterate.

The OTP service can create the itinerary only between pairs, so it can't receive the array as a whole. Therefore, the system passes the sorted array to the *generateItinerary* function that iterates on the array, takes a point and its following one, and sends them to the OTP. To work the OTP also needs the time and date of departure, which are set in our system by default with the current date and time, the maximum walking distance between the different bus transit, set to 500 meters, and the transport modes to consider, in this case, walk and public transportation. The function creates a new array with the itinerary linked together and returns it finally to the user.

If the research of points didn't produce any result, the user is notified. Otherwise, the system renders a new screen with a map and a description box, as shown in figure 2.25. In the map are present only the point of interest received back from the server and not all the points as in sections 2.4.1 and 2.4.5. The user can navigate the map and see the point name as in section 2.4.1. The system renders with two different colors, the bus path and the path where he has to walk, and when he passed over a path with the cursor, the line name is shown. To make the user comfortable was inserted a description box where the journey is described in a discursive way. If the user is not satisfied with the tour, he can select the *Create points of interest* tab on his homepage again.


localhost:3000/home#
Add new points of interest
Add new Categories
Create points of interest
Sign out
+
−
1 - Galleria civica d'arte moderna e contemporanea
Torino
Leaflet | Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © Mapbox

Walk from **Current Position** to **Galleria civica di arte moderna e contemporanea**.

At **G.A.M** stop take the BUS 5 direction Cittadella, Piazza Arbarello and get off at stop **Fermata 303 - ARBARELLO CAP**.

Walk from **Fermata 303 - ARBARELLO CAP** to **Museo d'arrte orientale**.

Walk from **Museo d'arrte orientale** to **Galleria Sabauda**.

Walk from **Galleria Sabauda** to **Museo civico di arte antica**.

Walk from **Museo civico di arte antica** to **Fermata 471 - CASTELLO**.

At **Fermata 471 - CASTELLO** stop take the BUS 13 direction Borgo Po, Piazza Gran Madre and get off at stop **Fermata 474 - ROSSINI**.

At **Fermata 474 - ROSSINI** stop take the Museo nazionale del cinema WALK direction and get off at stop **Pinacoteca dell'accademia albertina**.

Figure 2.25: Creates point of interest tab of the application after the tour creation. The user navigates on the map and points on the first stop of the journey *Galleria civica di arte moderna e contemporanea*. The light blue line represent the path with public transportation and the dark blue the path where the user has to walk. On the right are reported the steps to be execute.

Chapter 3

Data Fusion

Data fusion is a technique allowing the integration of heterogeneous data coming from different sources in order to achieve a higher level of data quality. The latter can be expressed both qualitatively and quantitatively, depending on their usage. In the data fusion domain, high-quality data are the ones that represent more similarly a real-world entity. Regarding instead the algorithm quality, four measurements are used to establish its quality: accuracy, precision, recall, and F-score. The accuracy is the ratio between the correct prediction over all the observations; it is a proper measurement with symmetric datasets, where the number of true positives and false negatives are balanced. The precision is the ratio between the correctly predicted positive over the total positive predicted. The recall is the ratio between the correctly predicted positive over the whole observation. The F-score represents a weighted accuracy that can be used with unbalances datasets and is a weighted average of precision and recall.

Data fusion was initially created for military-related problems, and it is not a coincidence that the most widely researched and documented area is the multisensor one, indeed [1]. Nonetheless its origin this technique has acquired, over time, increasing emphasis in the information integration field. The growing of the information available on the internet and their easy access has encouraged interest in data fusion, based on the heterogeneity of data contained and in their representation. For more than 30 years, researches go on this field solving the problem of missing and uncertain data as well as schema heterogeneity, but usually, the semantic aspect of data is ignored. The first misunderstanding in approaching data fusion is to assume that it is a synonym of data integration. Both techniques deal with heterogeneous data, but data integration takes specific pieces of information from trusted sourced without merge together all the information as done by the data fusion.

The first step before explaining the different data fusion models is to focus on the schema used to create the algorithm. It is composed of three steps:

Chose source data is the crucial step in the process to understand the problem that we are dealing with and choose the amount of data to use and the type that will add value to the final dataset. This step is essential because data that apparently may seem relevant for the scope after in-depth investigation are just useless and source of misleading information.

Follow a rigorous model in order to deal with the different degrees of uncertainty inherent in data fusion. The best strategy is to use the divide and conquer one to apply different techniques for each subproblem but not the only one. This phase is the most general one where the optimization can be executed and is the most documented one.

Create a trustworthy model based on the previous step can also be feasible in terms of cost, time, and resources.

3.1 Data fusion models

Data fusion can be applied in different areas and at different levels, to merge data coming from sensors, IoTs, or general hardware devices or can be applied to text-based data. In order to give a general overview of data fusion techniques, the classification can follow the criteria presented in [3]:

Relation between data proposed by Duran-Whyte [2] divides the data into redundant, complementary, and cooperative. Complementary data provides complete information by put together two different sources that alone do not have any meaning for the scope. Redundant data are the ones that are precisely equal and can be used to ensure faith but do not add any additional information to the dataset. The last category is the most complex because this type of data is the core of the data fusion process since when they are combined, returns a more complex and complete data.

I/O data nature also known as Dasarathy's classification [4], classify the data fusion techniques based on the pair of input and output data that interact with the algorithm. The data pairs are divided into five types: raw data-in and out coming directly from the sensors, sensor data-in processed to extract feature, feature elaborated more in detail, feature inserted to extract decisions, and the last one is decision fusion.

Abstraction level described in [5] is composed of four levels: signal for the data sensors, pixel for image processing, characteristic are metadata extracted from the previous levels and symbol level where the information is presented in its higher form.

JDL data fusion classification [6] is divided into five processing levels: source preprocessing, object refinement, situation assessment, impact assessment, and process refinement. The first process extracts the data from the source and executes a low-level fusion on raw data. Then these data are cleaned up using different techniques and stored in a uniform data structure. The situation assessment is similar to the previous process, but it is focused on the relationship between the data. Inside the fusion domain, the process executed so far is evaluated, and all the risks are taken into account. The last process is exterior to the fusion domain and concerns the system resource management.

Architecture type where the data fusion is performed. In a centralized architecture, all the computation is focused on the central processor. The drawbacks of this approach are time and bandwidth cost used for communication between input data and processor. In a distributed architecture, each information source preprocessed its data individually and then sent the data to a central node only for the fusion step. This approach overcomes the centralize architecture limitation, but each node as only a partial view of all the data that may lead to error. The decentralized architecture works with a network of peers, each one performs all the tasks of a centralized architecture, but it works with the data coming from its pears. This method is not scalable due to the communication cost. The last model combines the distributed and decentralized one hierarchically.

Apart from the model used in the data fusion algorithm, there are several fields where this technique can be applied. In the field of Linked Open Data, it exists a framework called Linked Data Integration Framework (LDIF)[16] that automatically integrate linked data. In particular, the Slieve module includes a module for data fusion and quality assessment [15]. It uses the metadata of named graphs to evaluate data quality. Slieve is high configurable, allowing the user to select the metadata and the fusion function to use using an XML file. In [14] is presented a specific resolution strategy based on the Bayesian model. Also [13] presents an algorithm based on voting and probabilistic models.

3.2 Three step algorithm

Data fusion, in general, address the inconsistency problem at three different levels: schema, tuple, and values. These problems were addressed separately in the literature, and only in some cases, the researches are focused on the whole data fusion process. Following the literature [17], data fusion algorithms can be divided into three different phases, where each one handles a level separately.

3.2.1 Schema Matching

Schema matching handles the problem of data heterogeneity at the schema level, and it can be divided again into two sub-section: schema matching and data transformation. The first step is closely related to the data source that will be used because it is the only part that must be context-aware. The schema of the different data source are compared to understand the relations between them, in this case, the algorithm search for duplicate in the column declaration. The next decision is about the schema to use for the rest of the algorithm. There are three possibilities: use a standard schema, use a new one explicitly created based on the context, or choose one between the data sources. The first possibility is the more general one, but it can generate problems with multiple null values in the next algorithm phases and also discard important feature context related. The other two choices are highly related to the context, and for this reason, they benefit from the optimization of a specialized data fusion algorithm.

The data transformation is where all the data coming from different sources are retrieved and stored under the unique schema selected previously. In this phase, the algorithm also performs the first data transformation since there is not only one standard representation of the data. Data can have different formats as linked data, CSV files, XML files, or they can use UTF-8 or ASCII standards, to mention a few. This stage of the algorithm can be fully automated, and user interaction is required only for the input insertion because the user needs to choose the data sources and the schema to use.

In literature, there are different studies about schema matching prototype: MOMIS [7], DIKE [8], CUPID [9] are schema-based and use real-world schema to perform their evaluation. In [10] different type of schema matching algorithm are compared, they are divided based on the level where they act (schema, instance, element, or structure) and on the type of matcher they use (language or constraint-based).

3.2.2 Duplicate Detection

Data duplication has as its goals to find the instance in the dataset that represents the same real-data object. Even if there are different types of approaches to the problem, all of them are based on a fundamental principle: compare entries pairs using similitude measurement and acceptance threshold [11]. Although the theory behind this phase is simple, due to the amount of data to process and the bound in terms of time, cost and resource data fusion is one of the more researched fields in the data mining for the last fifty years. The duplicate detection problem is addressed in [18] using a sorted method neighbor where the data are first sorted and then compared in smaller sets. The approach presented in [19] is based on the [18], but it uses an algorithm of Duplicate Count Strategy instead. In [20] is proposed a solution using a priority queue of duplicate records where each time a duplicate is found, it is put at the head of the queue.

Data duplication has to deal with two main problems: multiple representations that have only slight differences and the possibility that it must compare all the records using a brute force algorithm. The duplicate data can be of two types: exact matching that can be considered as redundant data and near matching that can contain spelling errors, abbreviations, or missing values. The performance of the data duplication algorithm can be measured in terms of effectiveness and efficiency. The effectiveness is linked to the similarity measurement, and the threshold used because this is the core of the algorithm. The efficiency is related mostly to the kind of algorithm used to check the similarity. There are four parameters used to evaluate the algorithm: time, cost space, and accuracy.

In the literature, there are three different techniques to create an algorithm for duplicate detection: knowledge-based, probabilistic, and empirical [12]. The knowledge-based requires more human interaction than the other because it requires knowledge and training to perform the decision. The probabilistic method is based on statistical methods, and it extracts the knowledge from the data using a bottom-up approach. This means that the algorithm does not have knowledge about the field it is working with, and in this way, it can be more generic. The empirical technique is based on the observation of previous researches to derive a new algorithm that has to be tested and evaluated in order to validate it. The algorithm following the empirical technique is composed of two steps: sorting and comparison. The data are sorted because choosing proper sorting-key increases the possibility of having duplicates in the same neighborhood. The comparison of the records can follow two different methods to decrease the comparison number as shown in figure 3.1:

Blocking : the data are partitioned in fixed blocks, and the search for duplicates is executed only between the data within the block.

Windowing : only the data in the window are compared to detect duplicates, but when the search is completed, the windows slice of one entry and the search is executed again.

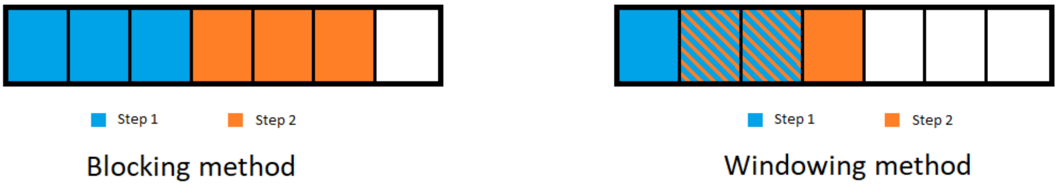


Figure 3.1: Comparison between blocking and windowing methods in the selection of data in two consecutive steps. In blue are represented the data selected in the first step, the second one is shown in orange. The figure on the left represents the blocking method, where each step considers a different set of data. On the right, is displayed the windowing method where the data selected by the two steps are overlapped.

3.2.3 Data Fusion

The data fusion goal is to combine the different representations of the same real-world object found in the previous steps to have a single representation of that entity. This step deals with two different types of conflicts: uncertainty is the conflict between a non-null representation and one or more null ones, contradiction is the conflict between two or more non-null representation. These conflicts may have been generated by the integration or may already exist in the dataset. Before entering the details of the fusion procedure and function, it is necessary to present the macro-categories for the data fusion strategy:

Conflict-ignore strategy does not make any decision on the conflicting data and sents it back to the user that has to solve the conflict using the *PassItOn* strategy. Sometimes this strategy is not even aware of the conflict. Pegasus [22] allows the user to access heterogeneous distributed data. It detects potential duplicates and maps them in a mapping table that is reviewed by the system administrator who sent back to the user the view without conflicts. Nimble [21] is the first integration system to use the XML data model, and it relies on an expert user to solve the conflicts. The Carnot system [23] allows the integration between structured and non-structured data using a global schema and an automatic transaction between them. It uses experts to execute the mapping and passes all the duplicates to the user. The InfoSleuth project [24] is the successor of the last one, and it allows to add or remove data sources at run time.

Conflict-avoid strategy applies a unique decision for all the conflict in general using the *TrustYourFriend* strategy. The TSIMMIS [25] and the SIMS [26] system are two mediator-wrapper systems that recognize the duplication and solve them by taking the data from a specific source previously selected. The first one uses an expert or machine learning method as mediator. Infomix [27] also is a mediator-wrapper system, but it can answer general queries incorporating different data models like XML, HTML, relational database.

Conflict-resolve strategy solves the conflict using two different strategies by deciding which data to keep or mediating and create a new value that is representative of the data considered. This resolution strategy applies relational operation and aggregate function to the data to solve the conflicts. Multibase [28] implements a solution using outer join operation and basic aggregation function to fuse data. In the same way, HumMer [29] implements the fusion through grouping and aggregation. Hermes [30] integrates data sources and reasoning facilities combined by an expert user acting as a mediator. In the conflict-resolution case, two different strategies can be applied: *CryInTheWolves*, which chooses the most used data in the dataset and *MeetInTheMiddle* that creates a new value that represents the conflict ones.

In conflict resolution, several functions can be applied to decide which data between the duplicate to keep in the final representation. In the deciding strategy, the algorithm can decide to take a value based on its insertion on the dataset, its last update, the number of most frequent appearances, and on the ownership to a specific data set. It can also decide to take the first non-null value found. When embracing the mediation strategy, all the aggregation function can be used to create new representative data.

Chapter 4

Proposed Solution

The goal of the thesis is to develop a data fusion algorithm to extend and validate the data inserted in a database containing points of touristic interest related to Turin. Each point of interest is represented by name, latitude, longitude, categories, and eventually subcategories. The categorization of these points is the principal element of the application because the user selecting the categories receives back a touristic tour related to them. For the creation of the application's dataset are selected three different sources: open data of Turin city hall (AperTO), WikiData query results, and user inserted data as explained in section 2.1.1. The schema heterogeneity and the need to automatically validate the data inserted by the users has led to the need to adopt data fusion techniques. The algorithm developed to overcome these limitations falls into the category of three steps data fusion algorithm. To better explain the proposed solution, each step of the algorithm is contained in a different section of this chapter.

In section 4.1 is described the algorithm performing the schema matching. This section underlines the difference between the sources at the schema level with the solution adopted. Duplicate detection is reported in section 4.2 where is presented the DCS++ algorithm and the similarity measurement used. Section 4.3 reports the conflict resolution performed by the data fusion algorithm and the function used.

4.1 Schema Matching

Data retrieved from different sources can be published with different schemas and formats. The datasets selected for the case study have all the same CSV format but different schemas. It is necessary to point out that the dataset AperTO is composed of seven different CSV files that enclose specific points of interest for one category or sub-category. From now on, every time the AperTO dataset is reported, it refers to all the seven files in it. Before dealing with specific problems inherent in each dataset, it is crucial to select the schema to use for the next phases. Since the data will be used as support for the application, and already exists a schema used by the application to retrieve the data from the user, it is straightforward to use that schema. The schema used is described in section 2.3.3, and more specifically, it is the Places table shown in figure 2.9.

As explained in chapter 3, this phase of the algorithm is context-aware, and it uses two different strategies to transform the data based on the dataset considered. Multiple data transformations must be executed regards the datasets of AperTO.The first inconsistency regards the coordinates representation. In the AperTO datasets, coordinates are reported as a pair in one column following the representation of the Geospatial point, in the selected schema, latitude and longitude are stored in two different columns instead. There is another disagreement between these two schemas regarding the geographical reference system used. Coordinates in AperTO datasets follow the Gauss-Boaga Roma40 western projection, a grid system based on the Roma 1940 geodetic datum whose origin lies at Monte Mario near Rome, and it is divided into western and eastern zone. WikiData and user inserted data follow the World Geodetic System (WGS), which is a standard used by the Global Positioning System (GPS). These transformations are executed together by the algorithm. For each file in the AperTO dataset, it loops over the row, selects the corresponding column, transforms the Geospatial point in two different values, converts the coordinates in WSG, and stores them in two different columns.

For the next phase of data transformation, an intermediate schema was inserted to facilitate the mapping between categories included in the AperTO and WikiData datasets in string format and their corresponding identifier stored in the application database. The algorithm first creates a new schema that combines together columns of the WikiData dataset, and the selected schema explained before. Then it inserts in this schema the data from WikiData and AperTO datasets. At this point, the algorithm retrieves from the database the mapping between categories, subcategories, and identifier and stores it into a map. It executes, for each row in the new schema, a comparison between the category in string format and the map created before. After a row is converted in the right category format, it is inserted in the Places database. At the end of this phase, all the data are inserted in one table using a common schema; therefore, the problem of schema heterogeneity is solved.

4.2 Duplicate Detection

The core of the algorithm is the duplicate detection phase because the data found here underlie the data fusion process. The algorithm has to deal with a massive amount of data and limited computational resources. The performances of the algorithm depend on the amount of comparison executed, to overcome this challenge, the search space must be reduced. Two different methods exist, as explained in section 3.2.2: blocking and windowing.The blocking method split the data into fixed non-overlapping blocks, and it searches for duplicates only inside the block. The windowing method, instead, creates a window that slides over the data. The limitation of the blocking method is that if two duplicates are in different blocks, the algorithm does not find them because they will be not compared. For this reason, it is preferable to use the windowing method instead. The size of the windows can be fixed or dynamically adjusted during the execution. Since it is challenging to select the perfect window size a priori and, even if it is found, the number of duplicates can vary in the dataset, the size of the window chosen for the project is modified dynamically during the execution.

Duplicates may be located in the same windows or at entirely different locations. In order to increase the likelihood that duplicates are in the same neighborhood, data can be sorted according to the discriminating columns for the project scope. Based on the consideration presented so far, the existing algorithm that fits better in the requirements is the Duplicate Count Strategy with multiple record increase (DCS++). This algorithm is based on the Sorted Neighborhood Method, and it adjusts the size of the window based on the duplicate number found during each iteration. It requires a starting size of the window and a threshold to allow the growth of the window. As explained in [19], if the threshold Φ depends on the size of the window w following equation 4.1, the algorithm is at least as efficient as SNM with the same window size.

$$\Phi = \frac{1}{w-1} \tag{4.1}$$

As reported in algorithm 1, it first sorts the data and populates the starting windows, and then it iterates over all data searching for duplicate. The algorithm always compares the first element of the window with all the others; when it founds a duplicate, it increases the duplicate count *duplicates* and inserts in the *skipRecords* array the $win[k]$ element, in other words, the duplicate of the first element. Then it checks if the size of the window should be increased and it behaves accordingly. For each comparison executed, the algorithm increases the comparison count *comparisons* and executes the second check on the window growth based on the threshold reported before. Every time the window slides over the data, the algorithm has to check ahead if the first element in the window is present in the *skipRecords* array. If the element is not in the *skipRecords* array, the algorithm executes all the tasks explained before. Otherwise, it just sets $k = w$; this is a modification to the standard DCS++ algorithm because, during the development of the algorithm, it inserted wrong values after it found a record to skip due to wrong k values. The last part of the algorithm slides the window by removing the first element from it, adding a new value to the window, and eventually resize the window to the starting value of w.

The parameters that can be modified to achieve better performance during the execution of the algorithm are three: the starting size of the window w, the sorting method, and the *isDuplicate* function that returns the duplicates. Given that in the project, duplicates can be found based on the name or the coordinate inserted, four different possibilities can be studied. Data can be sorted alphabetically based on their name or by their distance checking the coordinates. The string similarity of two places names or the distance between the pair of coordinates of two points can be used to check if these points are duplicates. Combining these two considerations, in the following section are described in detail the four combinations of sorting and detecting duplicate methods.

Algorithm 1 Duplicate Count Strategy with multiple records increase (DCS++)

```
Require: $w > 1$ and $0 < \Phi \leq 1$

Sort records
Populate windows $win$ with first $w$ records of $record$
$skipRecords \leftarrow null$

for $j \leftarrow 1$ to $records.length$ do
    $duplicates \leftarrow 0$
    $comparisons \leftarrow 0$
    $k \leftarrow 1$
    if $win[0]$ not in $skipRecords$ then
        while $k \leq win.length$ do
            if $isDuplicate(win[0], win[k])$ then
                emit duplicate pair $(win[0], win[k])$
                $skipRecords.add(win[k])$
                $duplicates \leftarrow duplicates + 1$
                while $win.length < k + w - 1$ and $j + win.length < records.length$ do
                    $win.add(records[j + win.length + 1])$
                end while
            end if

            $comparisons \leftarrow comparisons + 1$
            if $k = win.length$ and $j + k < records.length$ and $\frac{duplicates}{comparisons} \geq \Phi$ then
                $win.add(records[j + k + 1])$
            end if
            $k \leftarrow k + 1$
        end while
    else
        $k \leftarrow w$
    end if

    $win.remove(0)$
    if $win.length < w$ and $j + k < records.length$ then
        $win.add(records[j + k + 1])$
    else
        while $win.length > w$ do
            $win.remove(win.length)$
        end while
    end if
    $j \leftarrow j + 1$
end for
```

4.2.1 Sorting methods

The sort key can be composed of one or more columns in the dataset that manages the data sequence. In this project, the leading columns are latitude, longitude, and place name. Creating a sort key that combines significant parts of all of them is challenging because it is not easy to select just a part of all the three values maintaining a significant part of all of them. For this reason, two different strategies are applied to sort the data:

Order by clause is inserted directly in the SQL query to the database. With this method, data are retrieved alphabetically based on the name inserted in the database. It does not require any data modification afterward because the SQL query is already optimized. This clause cannot be used with latitude and longitude since it does not exist a way to execute the “order by” on a pair with the same weight, but it must choose a major and minor sort key.

Euclidean distance method modify the data after they are retrieved from the database by a simple SQL select query. A function iterate over the data, and for each pair, calculate the Euclidean distance between them ordering the points accordingly. It is necessary to clarify that the coordinates are represented following the World Geodetic System (WGS), which, from a geometric point of view, can be seen as a Cartesian reference system. Based on this assumption, the Euclidean distance gives an accurate distance measurement calculated following the formula 4.2.

$$d = \sqrt{(P1_x - P2_x)^2 + (P1_y - P2_y)^2} \tag{4.2}$$

4.2.2 Duplicate Detection functions

In algorithm 1, the function *isDuplicate* represents the core of the algorithm. Since the discriminating columns in the dataset are name, latitude, and longitude, the comparison to find duplicates can be executed on each of them individually, or they can be combined. All three possible solutions have been implemented because it is not possible to establish in advance which one will reach better performance. The *isDuplicate* function can use one metric between the following: string similarity, Euclidean distance, a combination of both of them.

String similarity algorithm is applied to the name attribute. Different types of measurement can be used because they follow different techniques to check the similarity. The first one proposed is the Ratcliff-Obershelp similarity, a sequence-based algorithm that searches the most extended sequence contained in both strings. The score is calculated by doubling the number of matching characters divided by the total characters number of the two strings. Levenshtein distance belongs to the edit distance based algorithms. The similarity is estimated by counting the number of operations need to transform the first string into the second. It allows three transformations on a single character: insert, delete, and replace. It belongs to the same category of algorithms, the Jaro-Winkler algorithm.

However, it has two conditions: strings must contain the same characters with a specific distance between them, and the matching order must be the same. The last algorithm considered is the Jaccard, a token-based algorithm that works on single characters instead of strings. The similitude is calculated by dividing the mutual tokens by the number of unique tokens in the strings.

Euclidean distance metric is applied to a pair of points, as explained in section 4.2.1. The latter method inserts in the *isDuplicate* function execute a check on both string similarity and coordinates distance. For each metric used, a threshold is established to allow the emission of the duplicate only when the metrics respect the threshold. Based on the sorting methods reported in section 4.2.1 and the functions explained before, four different scenarios are created:

- sort the data alphabetically and emit duplicates based on similarity measurement
- sort the data alphabetically and emit duplicates based on the combination of string similarity and coordinates distance
- sort the data and emit the duplicates following the Euclidean distance
- sort the data following the Euclidean distance and emit the duplicates based on the combination of string similarity and coordinates distance

The output of the duplicate detection phase of the algorithm is a CSV file where each row contains a set of duplicated data.

4.3 Data Fusion

TThe last part of the algorithm executes the fusion and the validation of the data applying two different resolution strategies. It iterates over the CSV file received from the previous phase, and for each row, it retrieves the duplicated data from the database and execute the fusion. The data with which the algorithm works come from different sources, as described in the introduction of chapter 4, with different levels of trust. To have high-quality data, a gold standard is introduced in order to keep the leading information coming from the most trustworthy source. Initially, the AperTO dataset was selected as gold standard since the local administration maintains it, but after some researches, it has been discovered that this dataset contains misleading information. For this reason, the new gold standard selected is the WikiData dataset.

The algorithm between the duplicated data first checks the data source, if it founds an entry coming from WikiData, marks it as the gold standard. If no entry from WikiData is found, the algorithm checks for the AperTO ones, and if it found it, this entry is chosen for the gold standard. When an entry became the gold standard, the name and the coordinates that it contains will be the one selected for the unique data returned after the fusion process.

Algorithm 2 Data Fusion pseudocode for the categories and subcategories integration

Require: *duplicateList* and *goldStandard*

```
$\Phi \leftarrow 3$
$userCat \leftarrow null$
$userSubCat \leftarrow null$

for $item$ in $duplicateList$ do
    if $item.get(placeOwner)$ = ( Wikidata  or  AperTO ) then
        for $element$ in $item.get(catId)$ do
            if $element$ not in $goldStandard.get(catId)$ then
                insert $element$ in the gold standard
            end if
        end for
        for $element$ in $item.get(subCatId)$ do
            if $element$ not in $goldStandard.get(subCatId)$ then
                insert $element$ in the gold standard
            end if
        end for

    else
        for $element$ in $item.get(catId)$ do
            if $element$ not in $goldStandard.get(catId)$ then
                if $userCat.count(element) < \Phi$ then
                    insert $element$ in $userCat$
                else
                    insert $element$ in the gold standard
                end if
            end if
        end for
        for $element$ in $item.get(subCatId)$ do
            if $element$ not in $goldStandard.get(subCatId)$ then
                if $userSubCat.count(element) < \Phi$ then
                    insert $element$ in $userSubCat$
                else
                    insert $element$ in the gold standard
                end if
            end if
        end for
    end if
end for
```

Regarding the selection of categories and subcategories, the algorithm behaves differently based on the source, as shown in algorithm 2. When the algorithm deals with data coming from AperTO and WikiData datasets, it integrates their categories without any further check because these two are trustworthy sources. Data coming from users are subject to additional analysis, and to be integrated into the final categories, at least three users have to insert the same category on the selected place to be considered valid. The threshold is chosen empirically and can be a point of weakness of the algorithm.

At the end of each duplicated set, the algorithm updates the database entry previously selected as the gold standard inserting the new categories and subcategories retrieved during the process, and mark this entry as verified. Now the entry can be used by the front-end application to generate the customized touristic tour.

Chapter 5

Result Analysis

Before executing a detailed analysis of the different possible cases, it is mandatory to introduce the metric used to evaluate the performances. As explained in chapter 4, the core of the algorithm is the duplicate detection phase that falls into the categories of the classification algorithm. The confusion matrix reported in table 5.1 is the basis of all the performance metrics linked with classification algorithm. It contains four different data type based on the combination of actual value and predicted value of a set:

TP – True Positive where the actual class of the considered point is true, as well as the predicted one.

TN – True Negative is the case in which both the actual class of the point and predicted one is false.

FP – False Positive when the actual class is false, but the predicted one is true

FN – False Negative is predicted negative even if it is actually positive

		Prediction	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

Table 5.1: Confusion Matrix

The performance metrics based on the confusion matrix that are taken into consideration are reported below. The accuracy is the easiest one, and it is the ratio of the correctly predicted over all the observations, as reported in equation 5.1. This metric is meaningful when the number of FP and FN are balanced.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

Equation 5.2 reports the precision that represents the number of the true positive predicted over the total predicted positive.

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

The recall reported in equation 5.3 represents the percentage of the duplicates correctly classified by the algorithm.

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

F-score represents a weighted average of precision and recall, as expressed in equation 5.4. It is useful when the system is unbalanced, and the accuracy cannot be used.

$$F - Score = \frac{2 * (Recall * Precision)}{Recall + Precision} \tag{5.4}$$

To find the configuration for the algorithm that outperforms in duplicate detection, four cases are created to combine sorting metrics and duplicate detection function. Besides, an evaluation of the starting size of the window is executed in each developed case.

5.1 Place Name Sorting with string similitude detection

The first combination taken into consideration is to sort the data alphabetically and detect duplicates based on the name similarity without considering the coordinates. In this scenario, four similarity metrics are used and compared: Ratcliff-Obershelp, Jaro-Winkler, Jaccard, and Levenshtein.For each similarity used, different threshold values of similarity are used to detect the duplicate. The algorithm first computes the similarity *distance*, then following equation 5.5, returns *true* if the two elements are duplicates, $false$ otherwise.

$$\begin{cases} True & \text{if } distance > \Phi \\ False & \text{if } distance \leq \Phi \end{cases} \tag{5.5}$$

The evaluation of each metric is presented individually in terms of precision, recall, and F-score to evaluate the different starting values of the window size. In the end, the similitude metrics are compared to select the most accurate in terms of F-Score.

The first case taken into consideration is the Ratcliff-Obershelp metrics, in figure 5.1 is reported the evolution of the precision for different windows size. The X-axis contains the different threshold values, and the Y-axis contains the precision value Φ_n instead. The precision follows an upward trend, without any difference between the size of the windows when considering a threshold between 0.95 and 1, where 1 represents the exact similitude. Between 0.9 and 0.95, the size of the windows *win* equal to 7 performs a little bit better, but in general, they perform the same. When considering the lowest threshold, the algorithm performs differently based on the initial window size. This last case considers as duplicates strings that have a similarity higher than 80%, so the number of false-positive FP increases according to the initial window size.

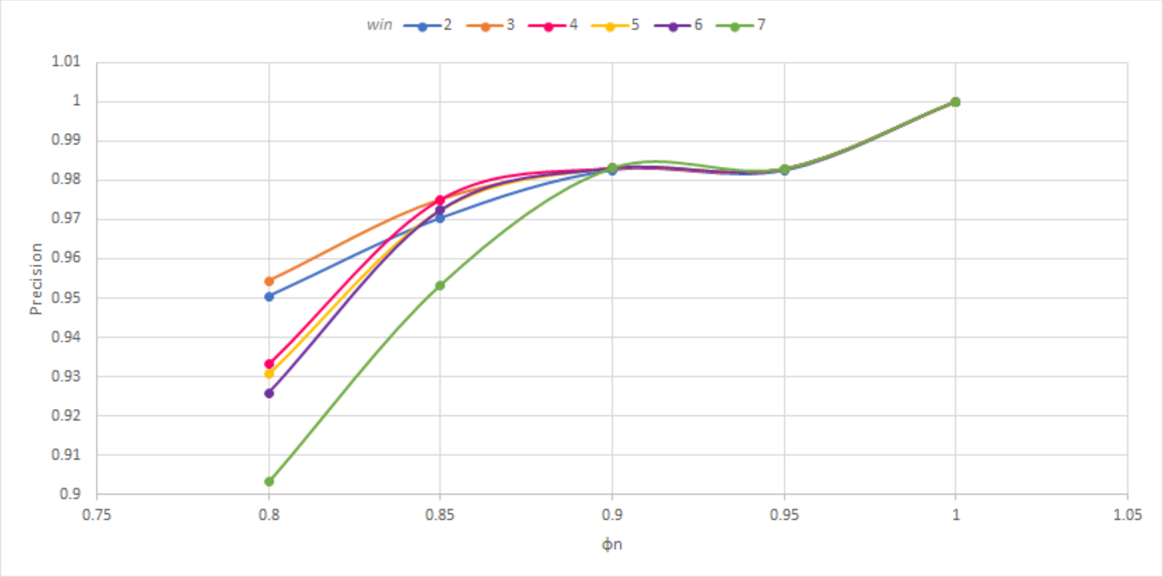


Figure 5.1: Precision of the Ratcliff-Obershelp similitude metrics.

The recall of the Ratcliff-Obershelp is shown in figure 5.2, it uses the same reference system with thresholds Φ_n on the x-axis and precision on the y-axis, and the different size of the window win is reported on the chart.

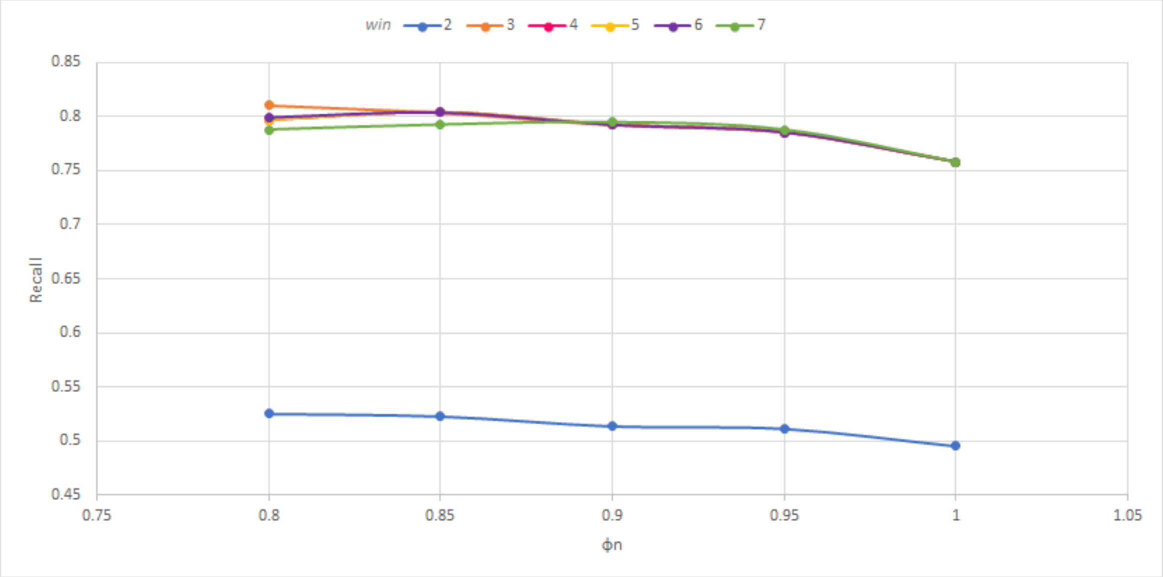


Figure 5.2: Recall of the Ratcliff-Obershelp similitude metrics.

The worst recall performances are reached with the smallest initial size of the window. For all the window sizes, the recall has a downward trend. Some differences between the size of the window can be seen at the lowest threshold Φ_n, with recall varying between the 0.79 and 0.81. However, in general, the windows between 3 and 7 have the same performance in terms of recall.

F-score, in figure 5.3, takes into consideration both recall and precision. High F-score means a high number of duplicate detected and correctly labeled. As might be expected, the performance whit window size equal to 2 is less than 0.7 since its recall was low. The trend is decreasing, and when considering thresholds higher than 0.9, the window size does not affect the performances. The initial size of the window equal to 3 outperforms in all the metrics when the threshold is set to 0.8.

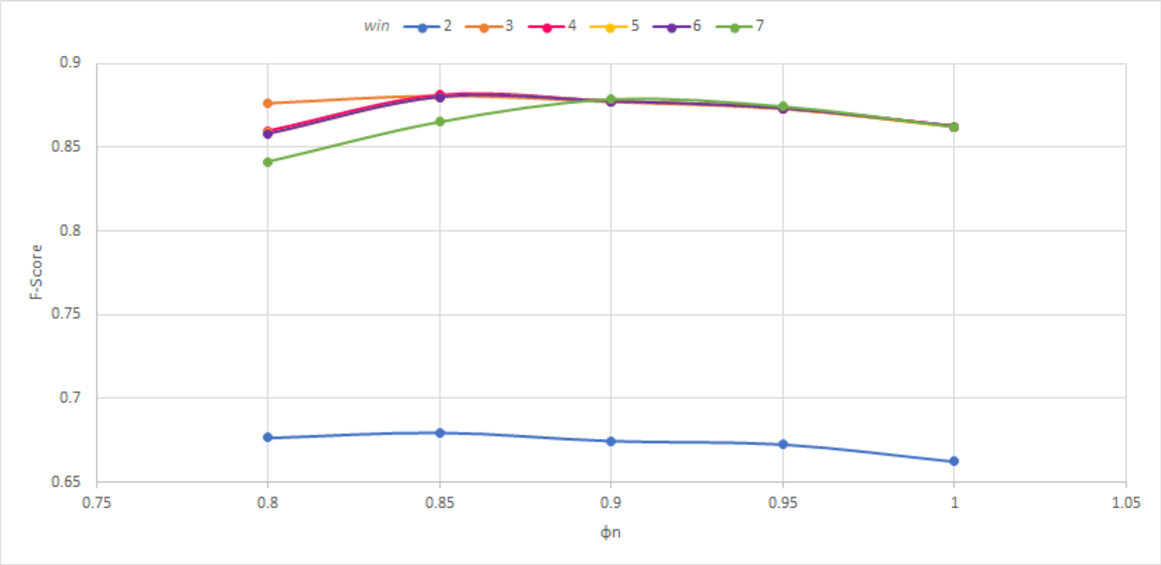


Figure 5.3: F-Score of the Ratcliff-Obershelp similitude metrics.

Jaro-Winkler precision varies a lot using different thresholds, as described in figure 5.4, passing from less than 20% to 100% when using the exact similarity. The window size that performs better with all the threshold is 2, reached by the other when the threshold Φ_n is higher than 0.95. It has the same reference system of the previous metric and follows an upward trend.

Regarding the recall presented in figure 3, the window with size *win* equal to 2 performs better whit a threshold Φ_n of 0.8, but never reaches a recall higher than 55%. The windows having size between 3 and 7 follow the same trend, reaching the maximum with a threshold equal to 3 and then decrease as the threshold increases.

Since both the recall and the precision of the Jaro-Winkler metric vary considerably using different threshold Φ_n, the F-Score in figure 5.6, reports a high variation in the performance between 25% and 90%. Of course, the trend of *win* equal to 2 dissociates from the other sizes of windows, performing better with a threshold less than 0.85 and worst otherwise. The window with size equal to 3 has the best performance until the

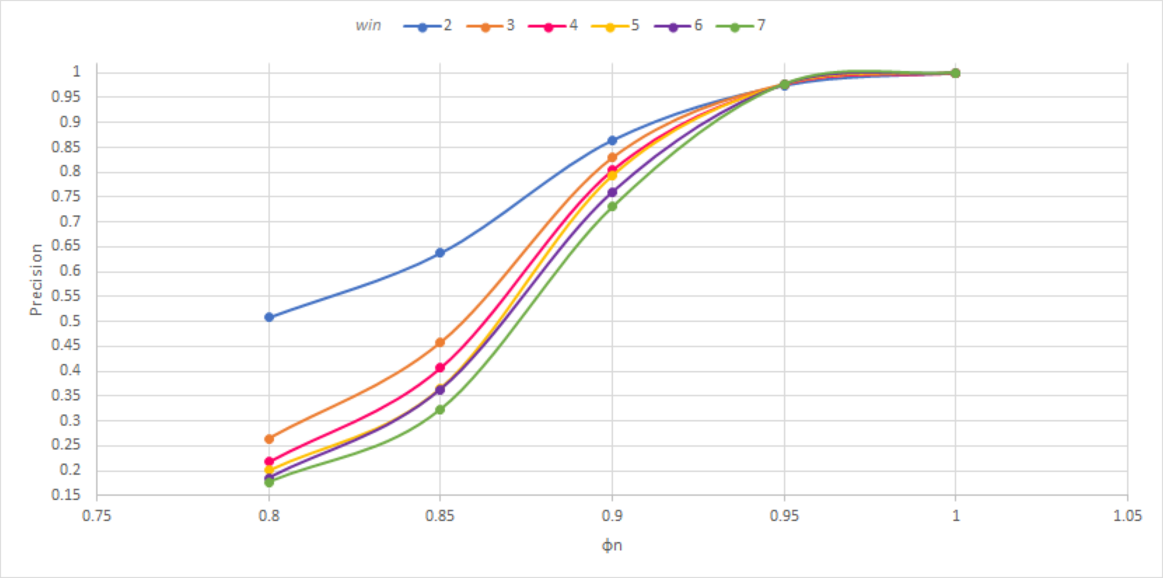


Figure 5.4: Precision of the Jaro-Winkler similitude metrics.

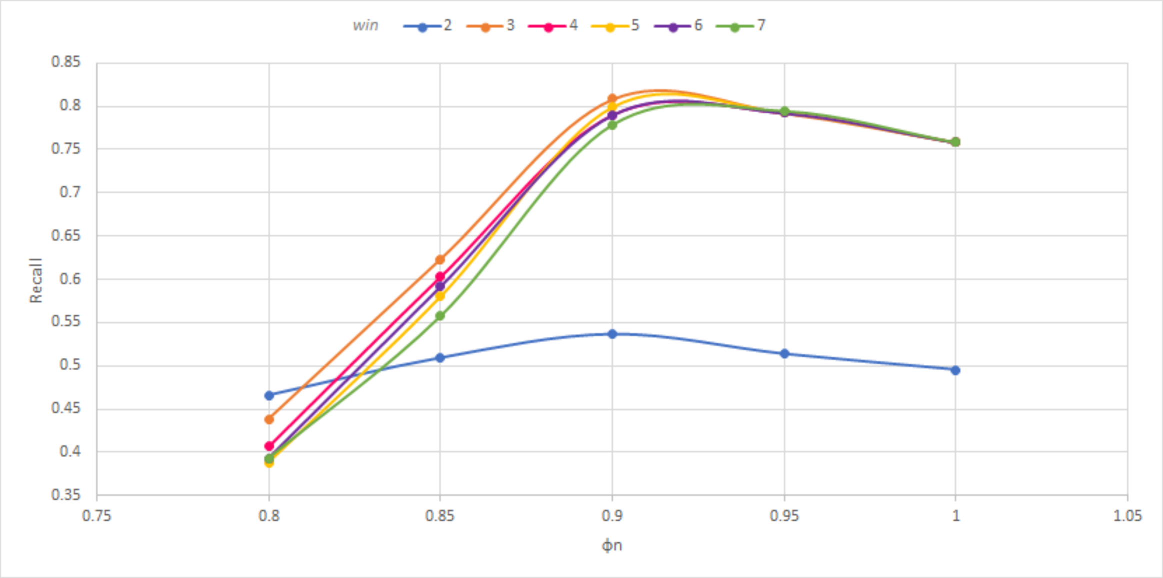


Figure 5.5: Recall of the Jaro-Winkler similitude metrics.

threshold equal to 0.95, and then it follows the same trend of window size between 4 and 7. Compared to the Ratcliff-Obershelp metric, Jaro-Winkler needs a higher threshold to have performance similar to it.

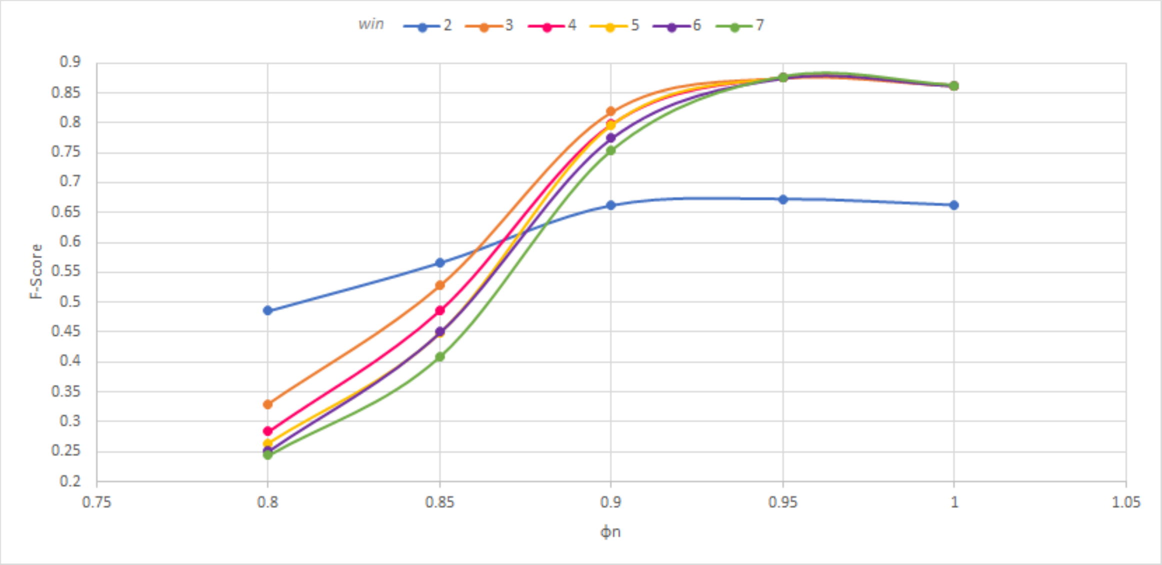


Figure 5.6: F-Score of the Jaro-Winkler similitude metrics.

Between all the similarity metrics used in the project, the Levenshtein one is the slowest and the one less affected by the window size. The precision reported in figure 5.7, is very high between 95% and 100%. The windows with size 3, 4, 5, and 6 follow the same trend, in fact, in the chart are overlapped, and only *win* equal to 6 is visible. The window having size equal to 7 has the lowest performance with a threshold Φ_n equal to 0.8, but increasing the threshold, it performs exactly like the other.

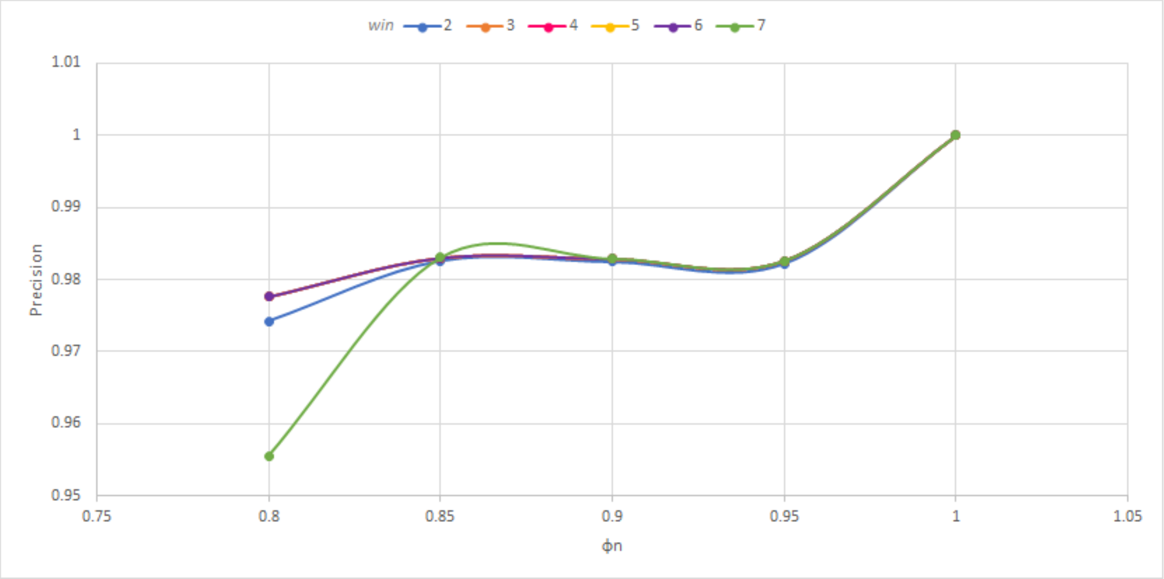


Figure 5.7: Precision of the Levenshtein similitude metrics.

Figure 5.8 reports the recall of the Levenshtein metrics based on different initial size of the window. Like the recall of Jaro-Winkler, the window with size equal to 2 has the worst performance lower than 70%. The other size of the windows follows the same decreasing trend, with an overlap with the window between 3 and 6 that performs better with the threshold Φ_n equal to 0.8.

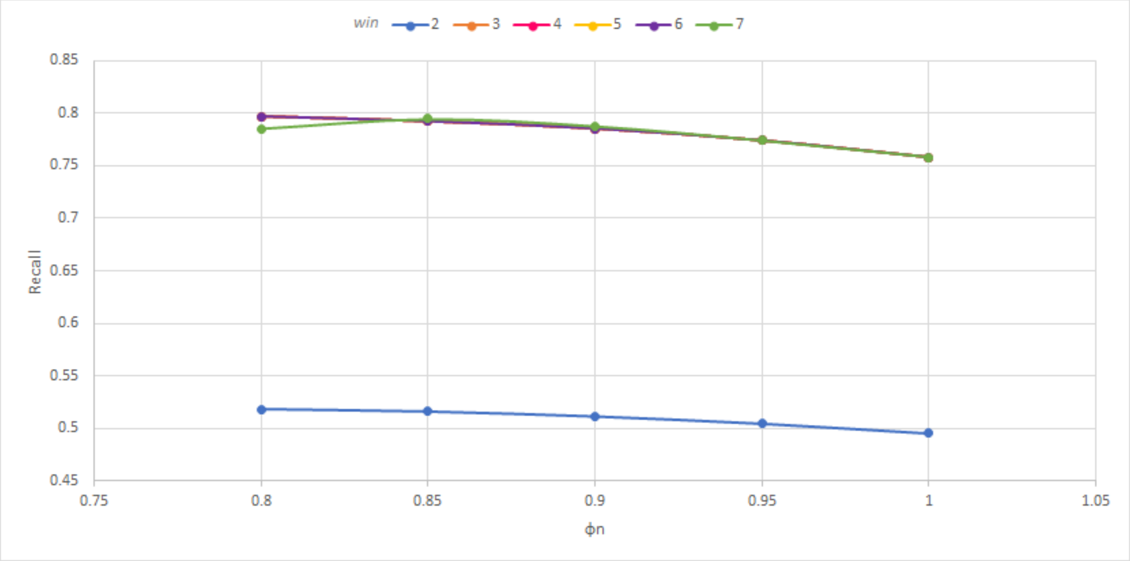


Figure 5.8: Recall of the Levenshtein similitude metrics.

F-score, shown in figure 5.9, is very similar to the recall in chart 5.8. It presents the window with *win* equal to 2 at the bottom with an F-Score less than 70%, and the other window having size between 3 and 7 following the same descending trend explained for the recall before.

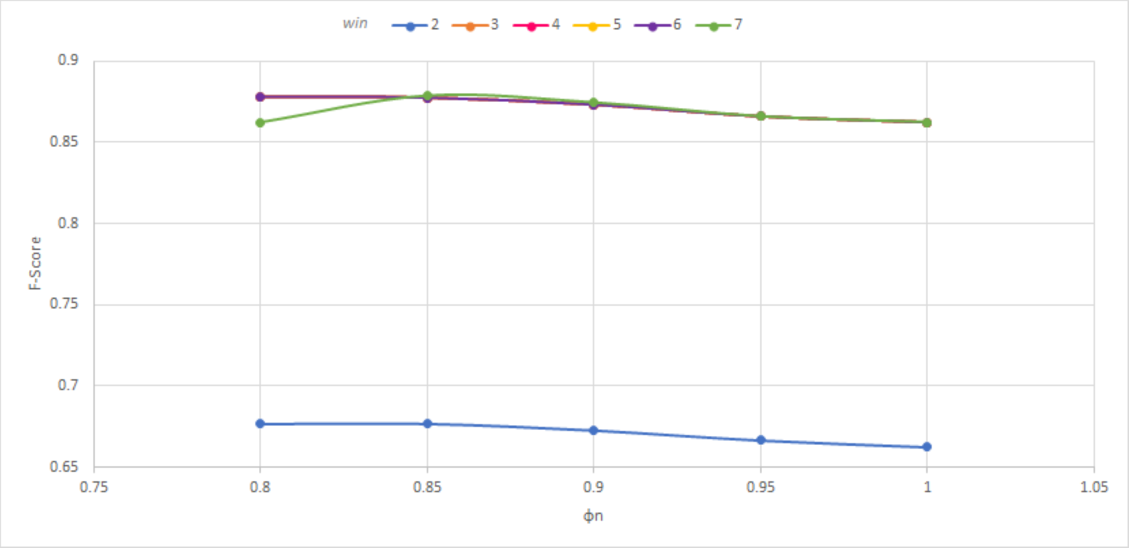


Figure 5.9: F-Score of the Levenshtein similitude metrics.

To evaluate the Jaccard metric first was considered the threshold range from 0.8 to 1, but since neither the size of the windows *win* and the threshold Φ_n did not influence the performance, the range was extended until 0.5. Figure 5.10, reports the precision of the Jaccard similarity metric using the extended threshold range. The precision has an upward trend until a threshold of 0.8 where it reaches its maximum precision of 100%. In the extended range, the window with size *win* equal to 7 is the one with the worst performance reaching a precision of 90%.

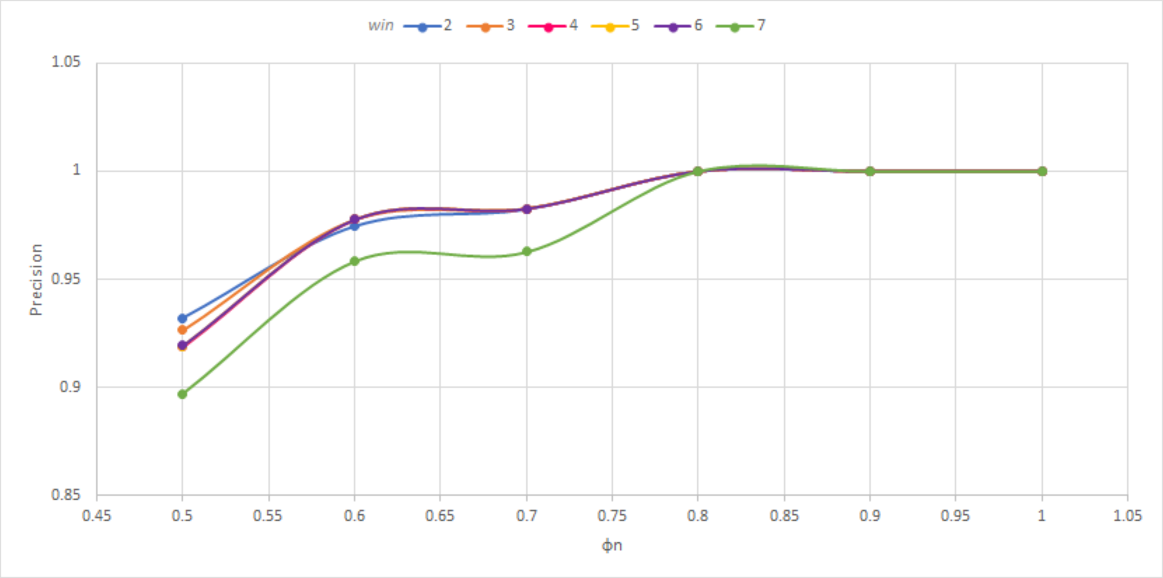


Figure 5.10: Precision of the Jaccard similitude metrics.

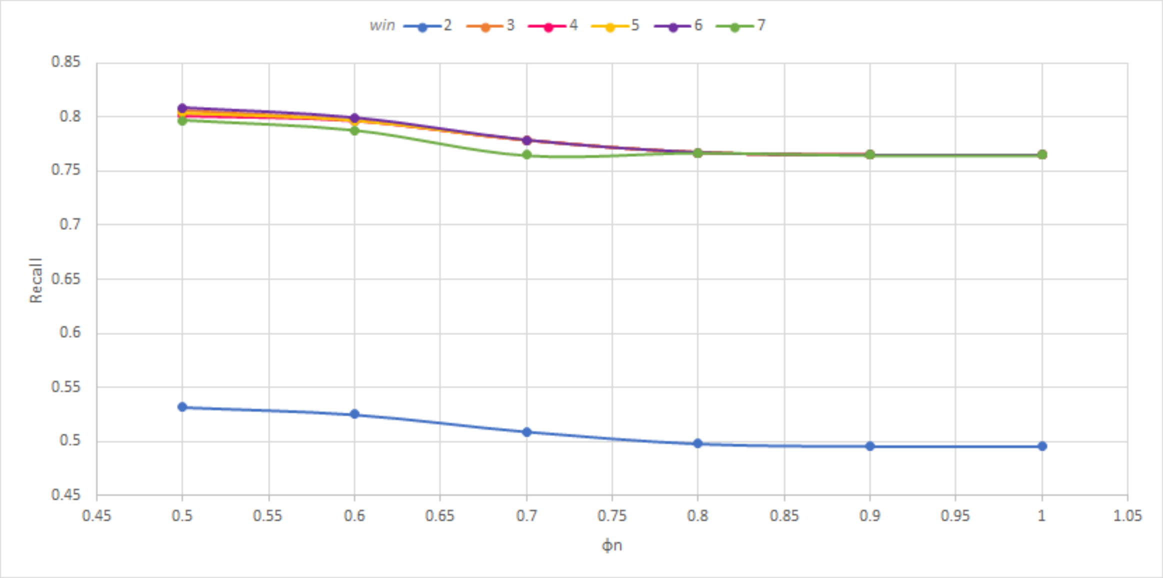


Figure 5.11: Recall of the Jaccard similitude metrics.

The recall of Jaccard is reported in figure 5.11 with a downward trend reaching the local minimum with a threshold Φ_n higher than 0.8. As already seen with the previous two metrics, the window with size equal to 2 has the worst recall performance, diverging from the other values. The F-Score in figure 5.12, agrees with the recall, having the worst performance with *win* equal to 2. The best performance, instead, is reached with a threshold of 0.6 using a window size between 3 and 6.

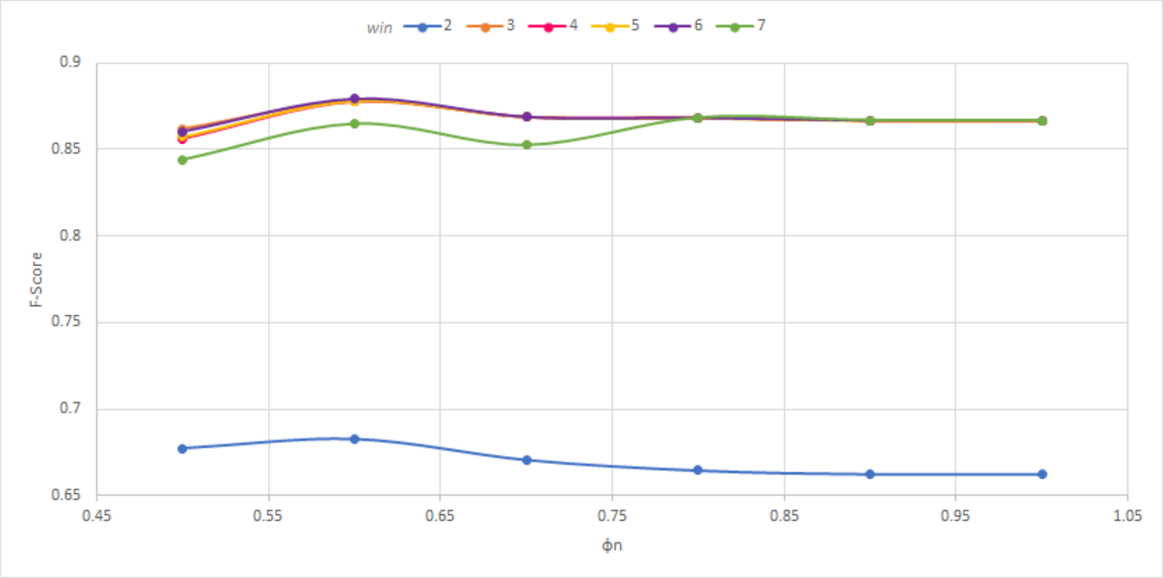


Figure 5.12: F-Score of the Jaccard similitude metrics.

All the similarity metric can reach an accuracy above 85%, and any of them can be used with a specific value of the size of the window *win* and threshold Φ_n. Between all the metrics, the Levenshtein is too slow and can be discarded since it performs like the other. Jaro-Winkler is the metrics that vary the most, in the range of threshold from 0.8 to 1. The Jaccard metrics perform better with very small threshold, the Ratcliff-Obershelp peforms better with threshold near 0.8.

5.2 Coordinates Sorting with euclidean distance detection

The second evaluation is done taking into consideration the other extreme case, sorting the dataset and detecting duplicates based on the increasing distance between the coordinates of the point of interest. In this scenario does not vary the metric to calculate the distance between points, but, as before, the window's size and the threshold vary to select the best combination of them. The window's size *win* can vary between 2 and 7. The threshold values Φ_c considered starts from the exact correspondence where the two points are overlapped until a distance beyond 0.001. The precision of this method, reported in figure 5.13, has a downward trend as expected. Increasing the area where two points are considered duplicates, it is more likely to found false-positive pairs. It was impossible to utilize only the exact match because different users inserting the same point of interest can insert coordinates slightly different. The x-axis contains the threshold values Φ_c, and the y-axis contains the precision instead.

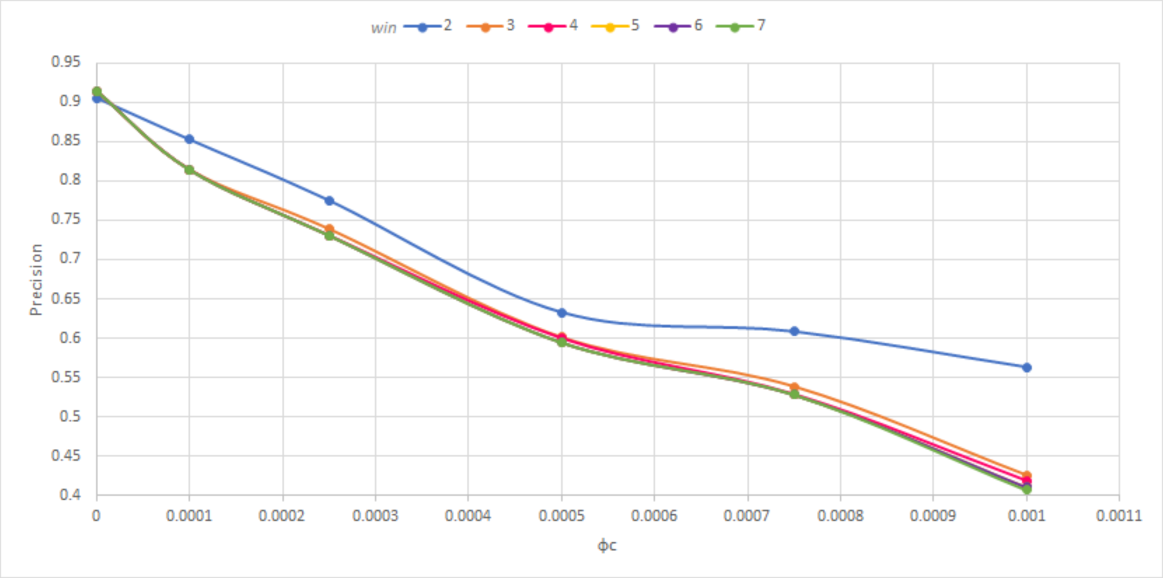


Figure 5.13: Precision of the Euclidean Distance.

The motivation behind the choice to increase the search area of duplicates is better explained considering the recall performance in figure 5.14. Using the distance equal to 0, the duplicates recognized are between 20% and 25%. With a higher threshold, the number of duplicates detected increases but at the expense of precision.

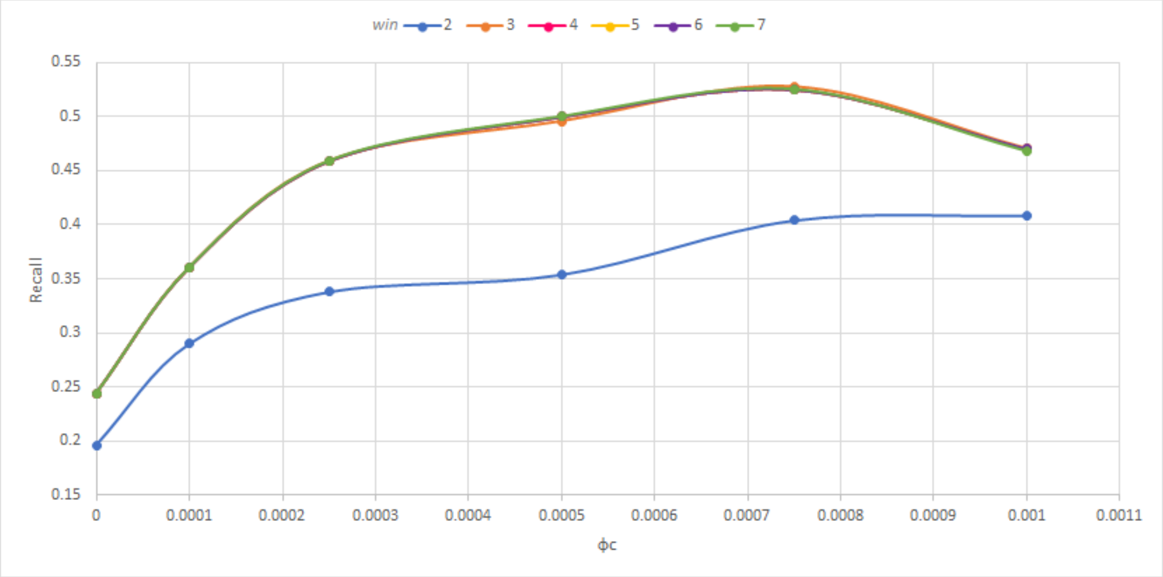


Figure 5.14: Recall of the Euclidean Distance.

In figure 5.15, the F-Score underlines what said before; in fact, with a threshold Φ_c of 0.001, the accuracy is approximately 45% meanwhile it rises above the 55% with a threshold of 0.00025. The window's size that has the worst performance is the one with size 2 like in the cases presented in section 5.1. Since with this method, the accuracy reached is very low, the combination of the two different methods is inserted.

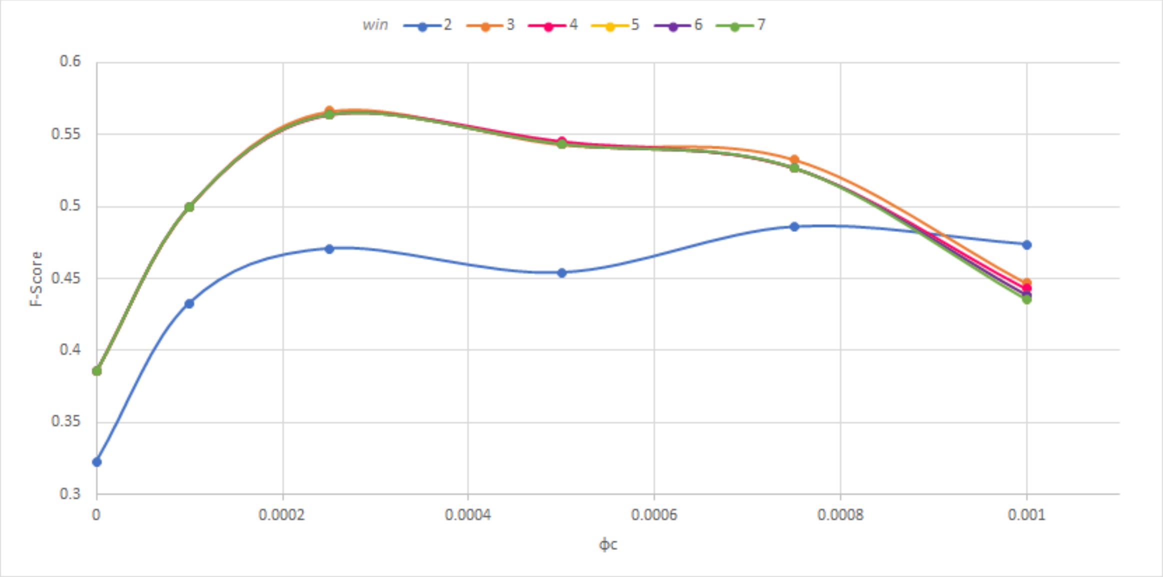


Figure 5.15: F-Score of the Euclidean Distance.

5.3 Place Name Sorting with combined duplicates detection

To achieve better accuracy and take into consideration both discriminating factors of name and coordinates, it was introduced a combined method to detect duplicates. The first one reported sorts the dataset alphabetically, still considers different window's size *win* and two different thresholds, Φ_c for the coordinates and Φ_n for the name. The combination of the thresholds is expressed in equation 5.6.

$$\begin{cases} True & \text{if } distanceN > \Phi_n \text{ and } distanceC < \Phi_c \\ False & \text{otherwise} \end{cases} \tag{5.6}$$

For this evaluation, the variables to consider are four, because besides the three expressed before, *win*, Φ_n and Φ_c, the similarity metric used in combination with the threshold Φ_n can be vary too. Since into the evaluation done in section 5.1, the Levenshtein metric has the same performance of the others, but it is much slower, the similarity metric considered are only: Jaccard, Jaro-Winkler, and Ratcliff-Obershelp. Given that the evaluation in terms of precision and recall for the similarity metrics have been already done in section 5.1, the next evaluation reports only the accuracy of each similarity metric.

The results report the accuracy of the method through F-Score, and they are divided by similarity metrics used and threshold Φ_c. The threshold Φ_c to evaluate the Euclidean distance is incremented by an order of measurement for each evaluation. Reach a threshold Φ_c of 0.01 is made possible by the combination of the two metrics because without the insertion of the similarity metric, the accuracy of the model presented in figure 5.15 decreases from 45% to 30%.

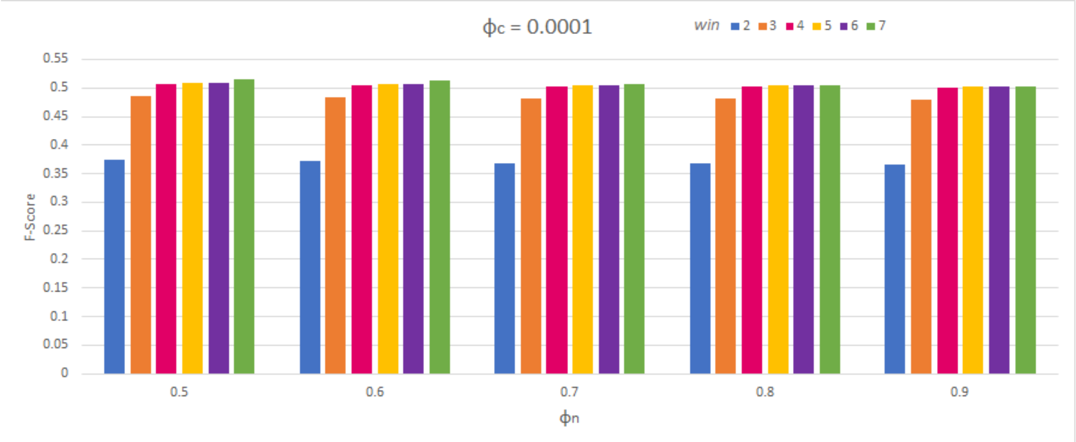


Figure 5.16: F-Score with threshold of 0.0001 Jaccard similarity metric.

Figure 5.16 presents the F-Score of the Jaccard similarity metric for a threshold Φ_c of 0.001. Based on the F-Score of figure 5.12 the model should reach the best performance with a window's size *win* from 3 to 6, but the combination with the Euclidean distance allows the window with size 7 to perform better. As expected, the best performance is reached with a small threshold Φ_n.

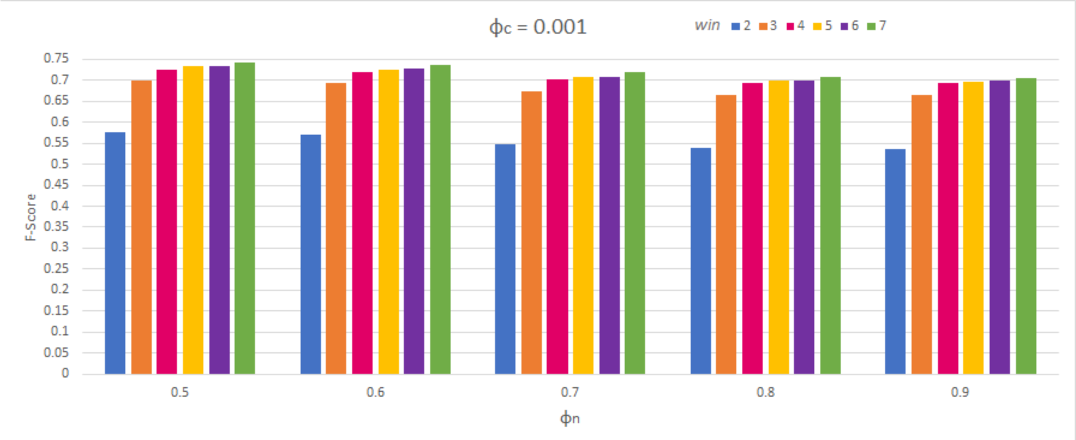


Figure 5.17: F-Score with a fixed threshold Φ_c of 0.001 Jaccard similarity metric.

With a threshold Φ_c of 0.001, reported in figure 5.17, the behavior expressed before is repeated, reaching the best F-Score performance with a threshold Φ_n of 0.5 and a window's size *win* of 7. The accuracy increases from 50% to 75% only with the increment of one order of measurement of the threshold Φ_c. In figure 5.18, increasing the threshold Φ_c by another order of measurement, there is an additional increment of the 10% from the F-Score always with a window's size *win* equal to 7 and a threshold Φ_n of 0.5.

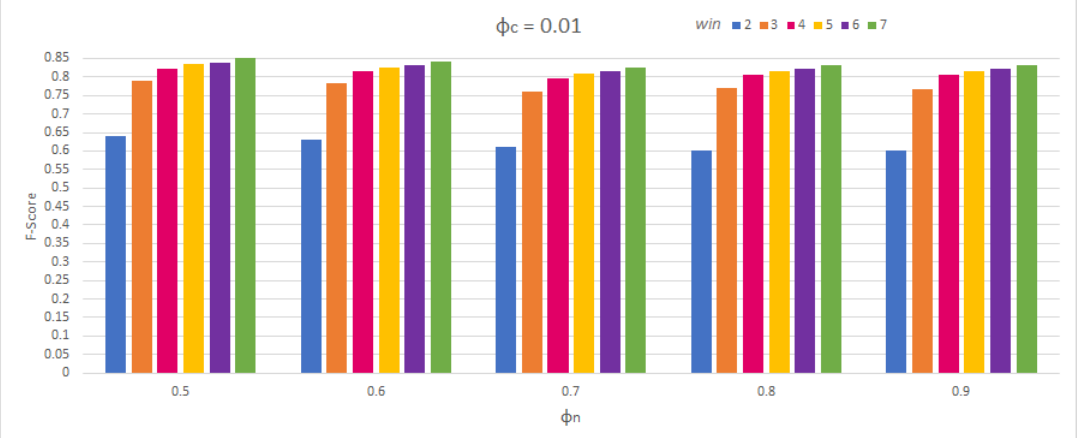


Figure 5.18: F-Score with a fixed threshold Φ_c of 0.01 Jaccard similarity metric.

Figure 5.19 reports the F-Score of the Jaro-Winkler similarity metric for a threshold Φ_c of 0.0001. Based on the evaluation of the accuracy done in figure 5.6, Jaro-Winkler metrics perform better selecting a high threshold of Φ_n, but here the best performance is reached with a threshold Φ_n equal to 0.8. With this threshold, the most performing window's size *win* should be the one equal to 3 according to the evaluation done in figure 5.6, but like the previous case, the best window's size *win* is equal to 7.

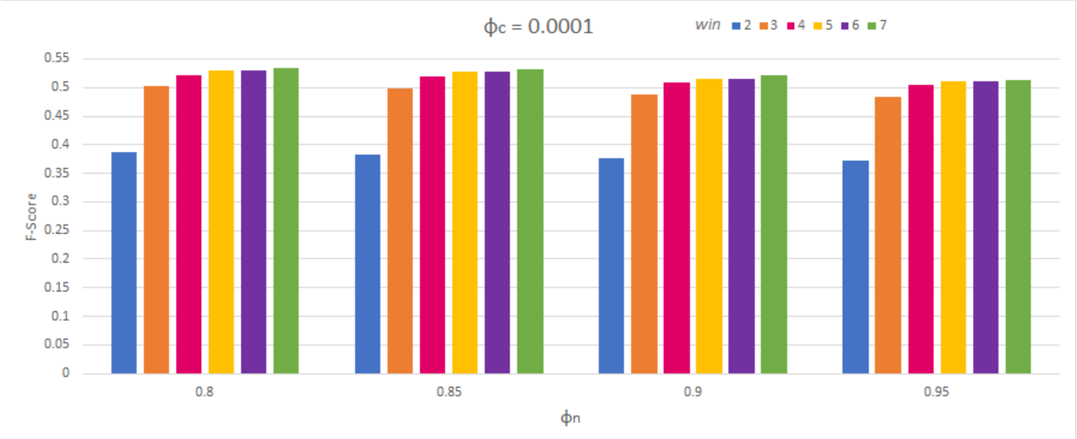


Figure 5.19: F-Score with a fixed threshold Φ_c of 0.0001 Jaro-Winkler similarity metric.

When setting the threshold Φ_c to 0.001, the F-Score increases from 50% to almost 80%. In this case, also the best threshold Φ_n changes from 0.8 to 0.85, as reported in figure 5.20. Figure 5.21, with a threshold Φ_c equal to 0.01 regarding the F-Score performance, is in accordance with the evaluation of the F-Score reported figure 5.6. The best performance is reached with a threshold Φ_n of 0.95 and a window's size *win* equal to 7. Compared to the previous case using the Jaccard metric, this combination performs better reaching an F-Score near 90%

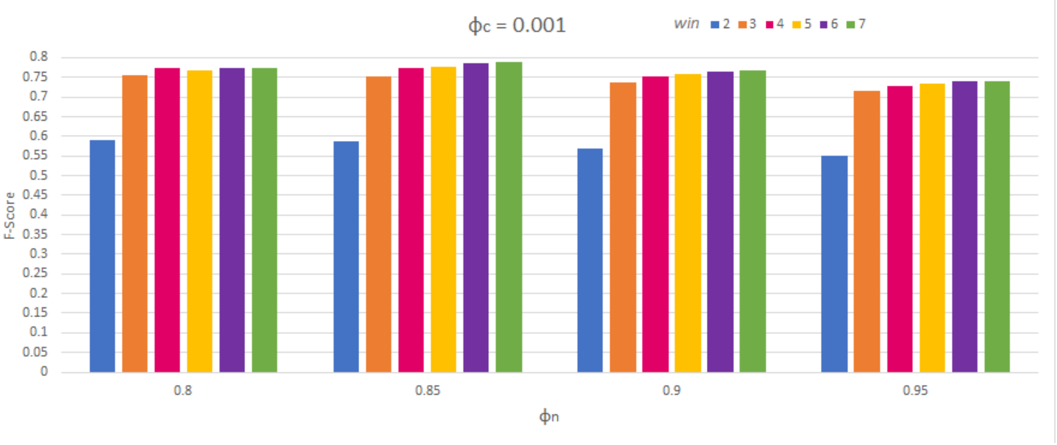


Figure 5.20: F-Score with a fixed threshold Φ_c of 0.001 Jaro-Winkler similarity metric.

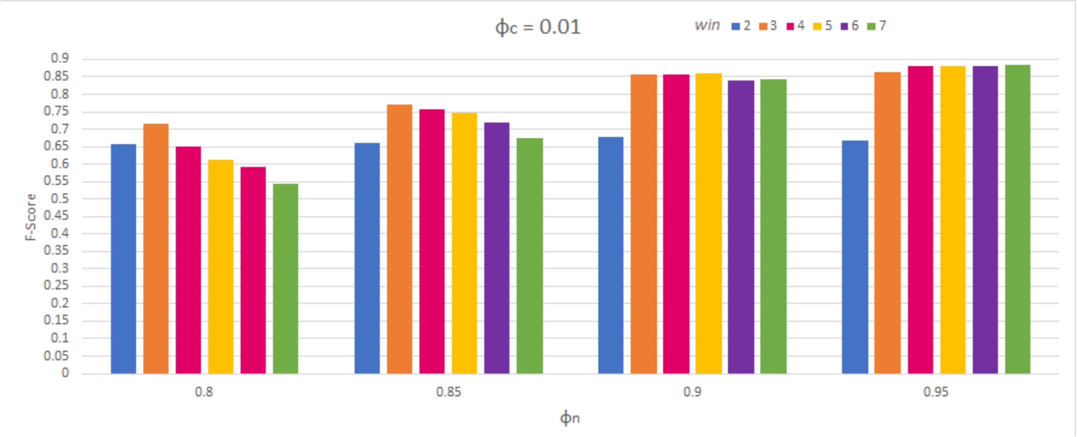


Figure 5.21: F-Score with a fixed threshold Φ_c of 0.01 Jaro-Winkler similarity metric.

Ratcliff-Obershelp is evaluated to select the best configuration to reach higher F-Score. Figure 5.22 reports the F-Score with a fixes threshold of Φ_c of 0.0001. The higher F-Score is reached with a threshold Φ_n of 0.8 as can be expected following figure 5.3, but the best window's size *win* is always 7 in contrast with figure 5.3, where it is equal to 3. Also, with a threshold Φ_c equal to 0.001, the best performance is reached with a window's size *win* of 0.8 and threshold Φ_n of 0.8, as shown in figure 5.23. Here the F-Score increases from 50% to 75%.

Figure 5.22 reports the F-Score with a threshold Φ_c of 0.01 that reaches the maximum F-Score within the Ratcliff-Obershelp metric. The threshold Φ_n selected is equal to 0.85, and the window's size *win* can be one between 4 to 6 because they reach the same performance.

To summarize the performance of the duplicate detection based on the alphabetically sorted dataset, it is clear that using the smallest threshold Φ_c of 0.0001, the algorithm

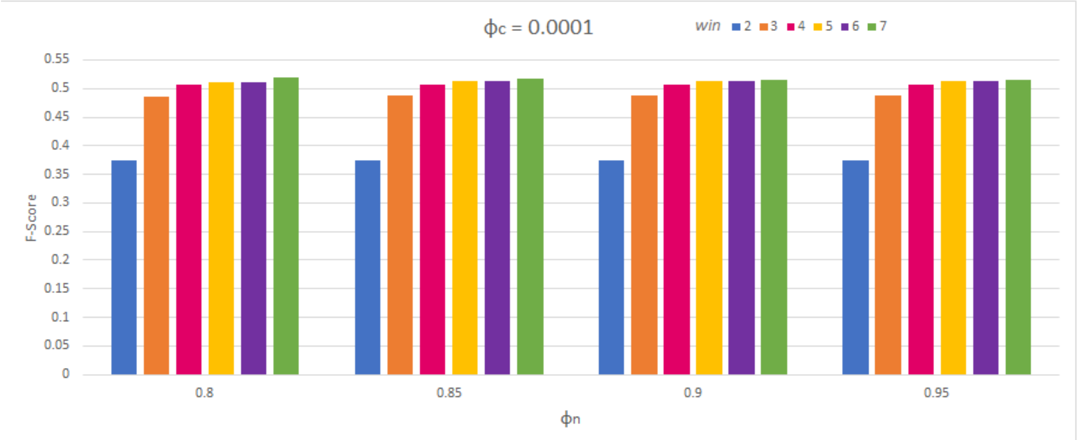


Figure 5.22: F-Score with a fixed threshold Φ_c of 0.0001 Ratcliff-Obershelp similarity metric.

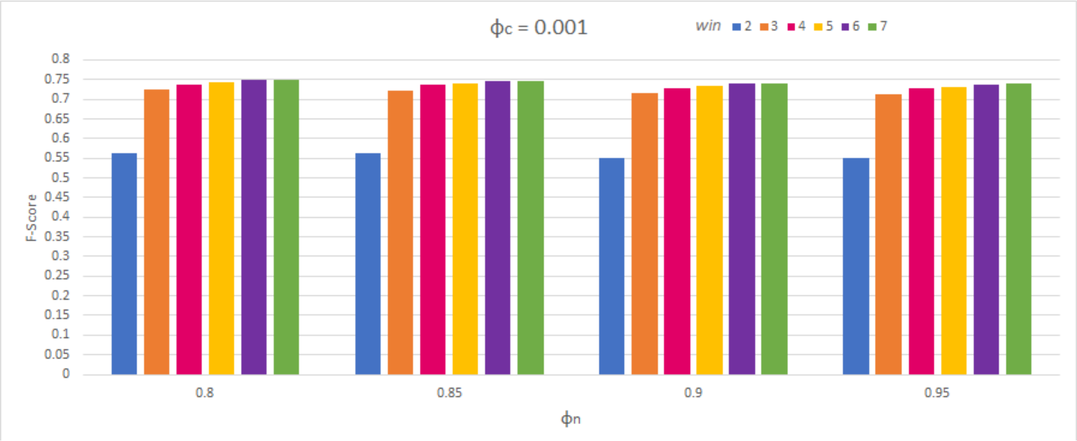


Figure 5.23: F-Score with a fixed threshold Φ_c of 0.001 Ratcliff-Obershelp similarity metric.

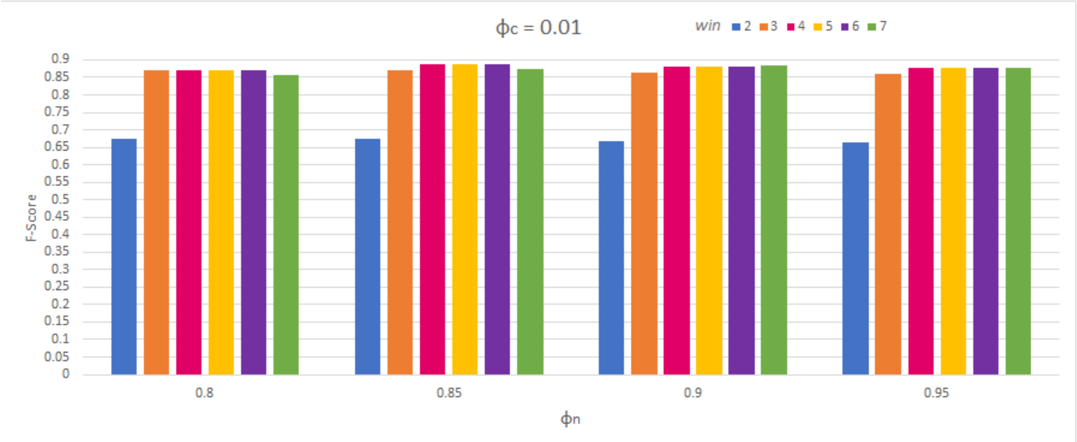


Figure 5.24: F-Score with a fixed threshold Φ_c of 0.01 Ratcliff-Obershelp similarity metric.

that performs best is the one using the Jaro-Winkler and a threshold Φ_n of 0.8. When considering a threshold Φ_c of 0.001, the Jaro-Winkler metric is still the best with a threshold Φ_n of 0.85 and a window's size *win* equal to 7. The best performance for this configuration is obtained with a threshold Φ_c of 0.01 with all the different similarity metrics used, but the best one is the Ratcliff-Obershelp with an F-Score of 88.83% followed by the Jaro-Winkler with 88.26%. The best configuration is achieved with a threshold Φ_n of 0.85 and a window with a size *win* equal to 6.

5.4 Coordinates Sorting with combined duplicates detection

The last configuration tried to achieve the best performance is the duplicate detection algorithm sorting the dataset based on the distance between the coordinates of the point of interest. The duplicate detection combines the coordinates distance and the similarity metrics applied to the name of the point of interest following equation 5.6. The same schema used for section 5.3 is used here, the F-Score for each similarity metric is reported for different thresholds Φ_c, and the comparison within the similarity metric is executed blocking the threshold Φ_c and let vary the window's size and the threshold Φ_n.

Figure 5.25 reports the F-Score of the Jaccard metric with a threshold Φ_c equal to 0.0001. The best configuration is achieved with a threshold Φ_n of 0.6, contrary to figure 5.16, where the best performance is reached with a threshold Φ of 0.5. This configuration agrees with the one in figure 5.12 because they both reach the maximum at Φ_n equals to 0.6.

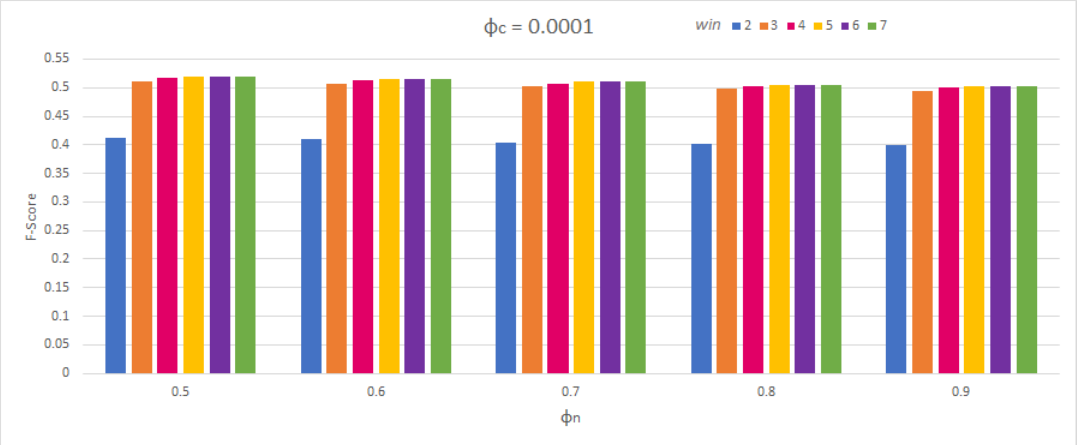


Figure 5.25: F-Score with a fixed threshold Φ_c of 0.0001 Jaccard similarity metric.

In figure 5.26 is described the F-Score with threshold Φ_c of 0.001 that performs better with a threshold Φ_n of 0.5 like the previous configuration. In this case, there is an abnormality with the threshold Φ_n equals to 0.7, where the F-Score dizzying decreases from the average 70% to less than 50%.

Like the case presented in section 5.3, the configuration with a threshold Φ_c equals to 0.01reported in figure 5.27, is the one that performs better with the Jaccard metrics

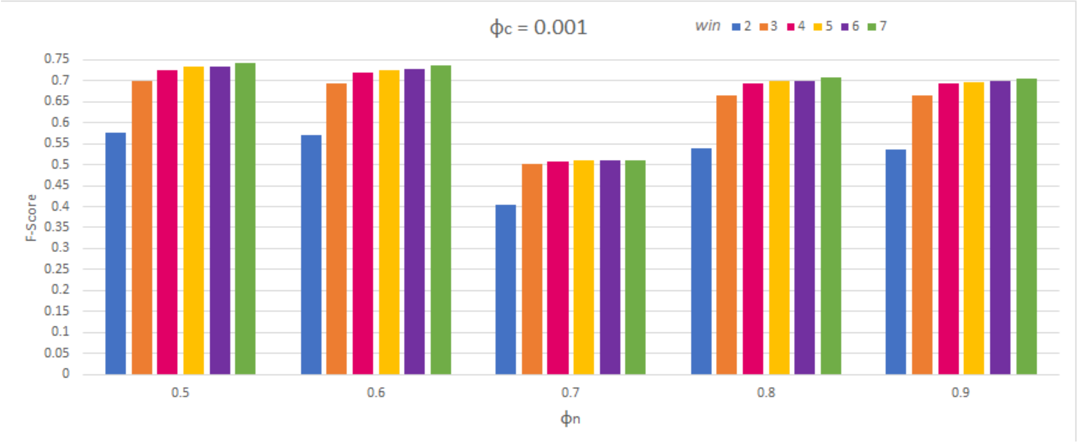


Figure 5.26: F-Score with a fixed threshold Φ_c of 0.001 Jaccard similarity metric.

reaching an F-Score of 85%. This result is obtained with a window's size *win* of 7 and threshold Φ_n of 0.8.

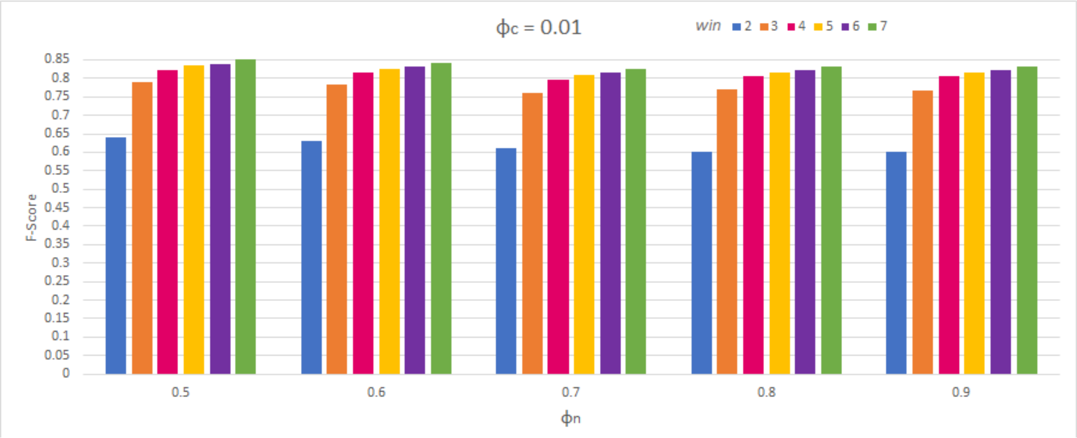


Figure 5.27: F-Score with a fixed threshold Φ_c of 0.01 Jaccard similarity metric.

F-Score of the Jaro-Winkler metric with a threshold Φ_c of 0.0001 is reported in figure 5.28. Like the case presented in the previous section in figure 5.19 the best configuration is achieved with a threshold Φ_n equals to 0.8, but here, the window's size *win* can be chosen between 4, 5, 6 and 7.

Selecting a threshold Φ_c equals to 0.001, the best configuration to achieve a higher F-Score is presented in figure 5.29 and includes a threshold Φ_n equals to 0.85 and a window with size *win* of 7 like the case in figure 5.20.

The difference between the case reported in figure 5.21 and the F-Score described here in figure 5.30 it presents with a threshold Φ_c of 0.01. Since in figure 5.21, the best configuration contains the threshold Φ_n of 0.95, and here the best one is found when the threshold Φ_n is equal to 0.9. Like the case presented in section 5.3, the Jaro-Winkler metric reaches the higher performance with a threshold Φ_c of 0.01, where the F-Score is

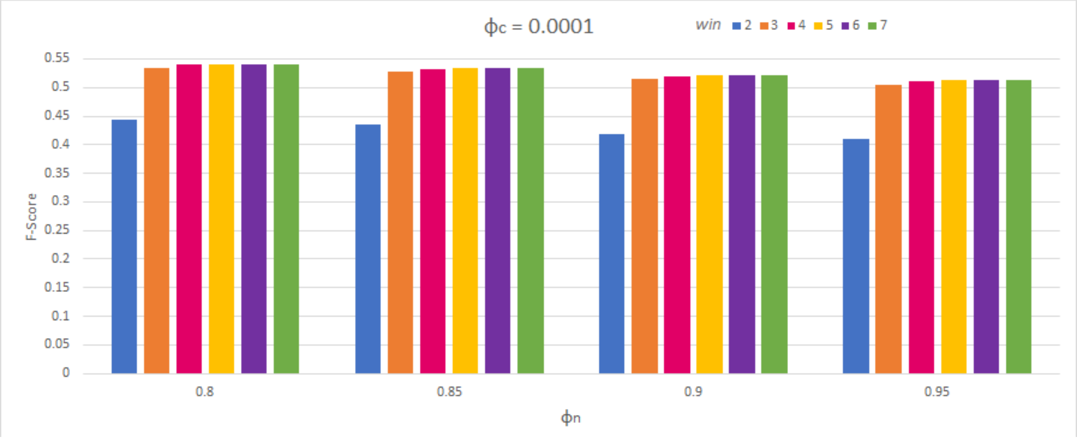


Figure 5.28: F-Score with a fixed threshold Φ_c of 0.0001 Jaro-Winkler similarity metric.

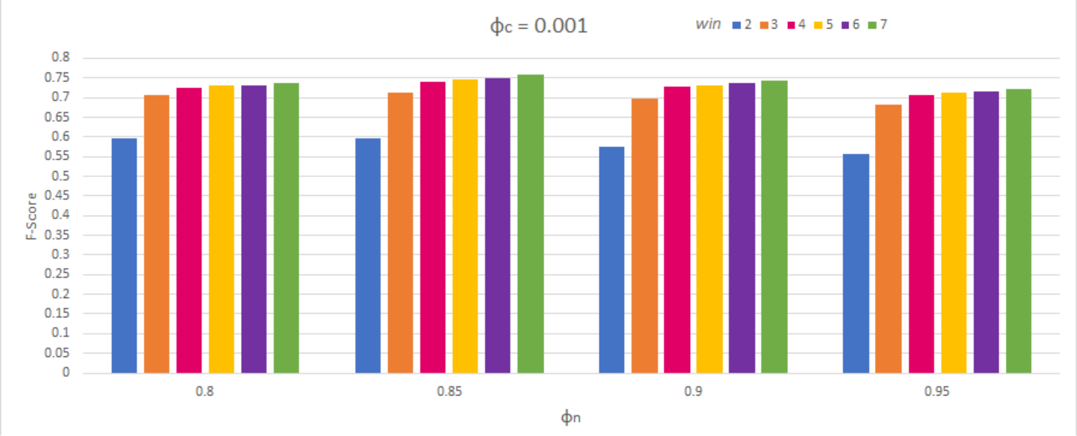


Figure 5.29: F-Score with a fixed threshold Φ_c of 0.001 Jaro-Winkler similarity metric.

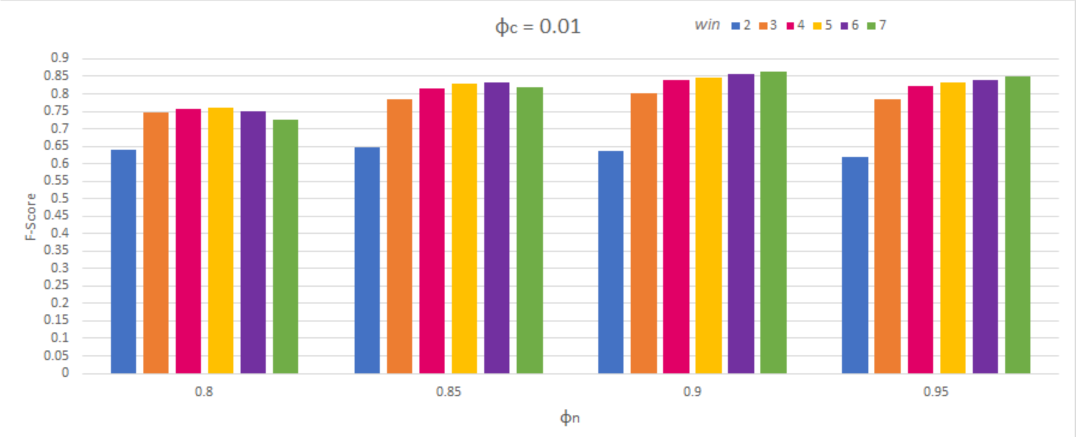


Figure 5.30: F-Score with a fixed threshold Φ_c of 0.01 Jaro-Winkler similarity metric.

equal to 86.40%, but the best configuration for this metric is still the one presented in section 5.3.

Figure 5.31 reports the F-Score of the Ratcliff-Obershelp similarity metric with a threshold Φ_c of 0.0001. The best configuration is precisely the same as figure 5.22, with a window's size *win* of 7 and a threshold Φ_n of 0.8. Also, for the case with a threshold Φ_c equals to 0.001 in figure 5.32, there are no changes compared to figure 5.23 since the best choice is still *win* equals to 7 and Φ_n equals to 0.8.

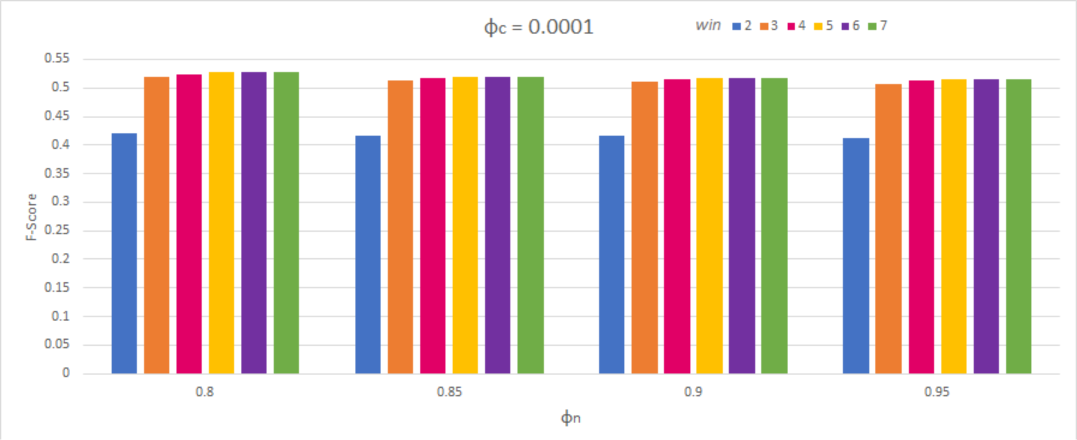


Figure 5.31: F-Score with a fixed threshold Φ_c of 0.0001 Ratcliff-Obershelp similarity metric.

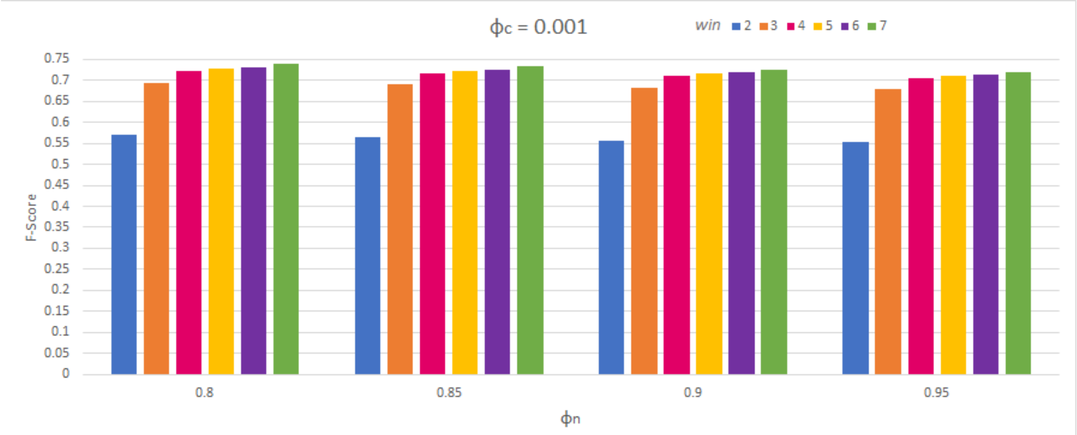


Figure 5.32: F-Score with a fixed threshold Φ_c of 0.001 Ratcliff-Obershelp similarity metric.

Like the case presented in section 5.3, Ratcliff-Obershelp performs better using a threshold Φ_c equal to 0.01, as presented in figure 5.33. The best configuration is achieved with a window's size *win* of 7 and a threshold Φ_n of 0.9 unlike the case in figure 5.24 where the best threshold Φ_n was equal to 0.85.

Between all the different combinations of sorting methods and duplicate detection functions, the worst in terms of performance is reported in section 5.2, where the dataset is sorted based on the coordinate's distance, and it also uses the coordinate's distance

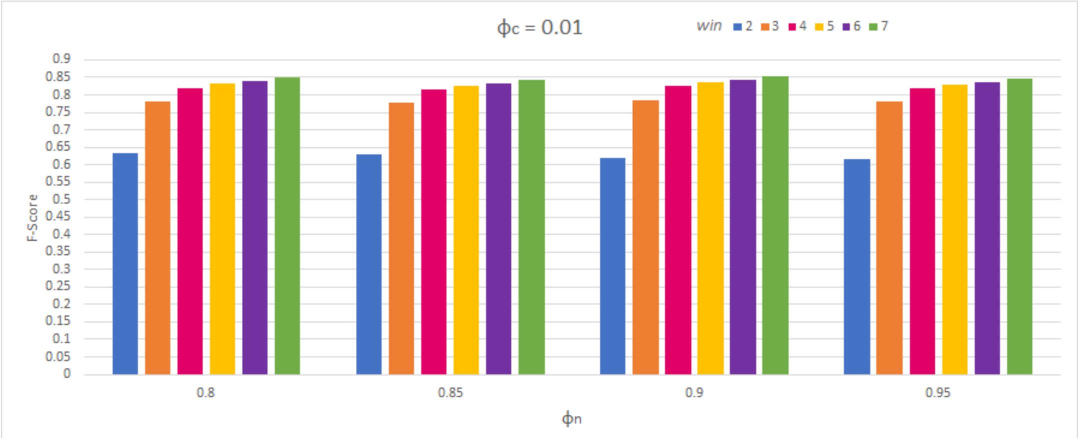


Figure 5.33: F-Score with a fixed threshold Φ_c of 0.01 Ratcliff-Obershelp similarity metric.

to detect duplicates, reaching an F-Score of 55%. Tho other combinations reach all an F-Score between 85% and 90%. In the end, the best combination proves to be the one with the dataset alphabetically sorted where the duplicate detection function uses both the Euclidean distance and the Ratcliff-Obershelp similarity metric. The configuration for the thresholds and the window's size is the one presented in figure 5.33 having the window's size *win* of 7, the threshold Φ_n of 0.9, and the threshold Φ_c equal to 0.01. This configuration has 0 false-positive, so at the end, the point of interest fused will not contain points of interest that are not duplicates.

Chapter 6

Conclusion and Future works

The data fusion algorithm developed in this project fits in the three-step algorithms domain. The schema matching phase deals with three different datasets: user inserted data, Wikidata query results, and dataset maintained by Turin city hall. The algorithm has to deal with different standards and representations in this phase; to do so, it selects the schema created for the user inserted data and modify the data accordingly. Duplicate detection phase follows the principles of DCS++, an algorithm that sorts the dataset based on a sorting key and compares the data with a windowing method. The additional configuration parameters selected for the duplicate detection algorithm are based on the evaluation reported in chapter 5. The configuration selected contains a dataset sorted alphabetically with a duplicate detection function considering both string similarity and Euclidean distance. Specifically, it uses the Ratcliff-Obershelp similarity metric with a threshold Φ_n equals to 0.9, a threshold Φ_c equals to 0.01 for the Euclidean distance and a window with size win equal to 7. The last phase of the algorithm follows a conflict-resolution strategy with the selection of a Gold standard to select the leading information from the most truthful source. Additionally, it implements a quorum strategy to integrate and validate the data inserted by the users.

With the algorithm presented here, the weighted accuracy F-Score achieved is about 89% without any false positive detected. It is a good approximation for the project goal that has been fully respected since the algorithm validates the data inserted by the users, integrates data coming from different sources, and increases the quality of the data used in the project. It should be clarified that this solution is not the perfect one reaching an F-Score of 100%, but for the thesis scope, it reaches an acceptable value. Another point of weakness is the quorum threshold selected because it was selected empirically without appropriate evaluations due to lack of time.

Further works can be done searching different configuration for the duplicate detection phase to reach a higher accuracy, or selecting a new duplicate detection algorithm. If the data inserted in the application grows exponentially should be good to implement a machine learning algorithm, since the amount of data now available is not large enough to require the use of machine learning.

Bibliography

[1] I. R. Goodman, Ronald P. S. Mahler, Hung T. Nguyen,"Mathematics of data fusion", *Springer-Science*

[2] H. F. Durant-Whyte, "Sensor models and multisensor integration", *International Jurnal of Robotic Research*, *Vol. 7*, No. 6 ,pp. 97-113, 1988.

[3] Federico Castanedo, "A Review of Data Fusion Techniques", *The Scientific World Journal.*

[4] B. V. Dasarathy, "Sensor fusion potential exploitation-innovative architectures and illustrative applications", *Proceding of the IEEE*, *Vol. 85*, No. 1, pp. 24-38, 1997.

[5] R. C. Luo, C.-C. Yih, K. L. Su, "Multisensor fusion and integration:approaches, application, and future research directions", *IEEE Sensors Jurnal*, *Vol. 2*, No. 2, pp. 107-199, 2002.

[6] "JDL", *Data Fusion Lexicon. Technical Panel For C3*, F.E. White, San Diego, California, USA, Code 4 20, 1991.

[7] S. Bergamaschi, S. Castano, M. Vincini, "Semantic Integration of Semistructured and Structured Data Sources", *SIGMOD Record*, *Vol. 28*, pp. 54-59, 1999.

[8] L. Palopoli, G. Terracina, D. Ursino, "The system DIKE: Towards the Semi-Automatic Synthesis of Cooperative Information System and Data Warehouses", *ADBIS-DASFAA*, pp. 108-117, 2000.

[9] J. Madhavan, P.A. Bernstein, E.Rham, "Generic schema matching with Cupid", *Proceeding of the International Conference on Very Large Databases(VLDB)*, Rome, Italy, 2001.

[10] E. Rahm, P. A. Bernstein, "A Survey of approaches to automatic schema matching", *VLDB Journal*, *Vol. 10*, No. 4, pp. 334-350, 2001.

[11] A. Skandar, M. Pehman, M. Anjum, "An Efficient Duplication Record Detection Algorithm for Data Cleansing", *Internationa Journal of Computer Applications(0975-8887)*, *Vol. 127*, No. 8 , October 2015.

[12] M. Rehman, V. Esichaikul,"Duplicate record detection for database cleansing" in Machine Vision, *ICMV'09. Second International Conference*, Dubai 2009, pp. 333-338.

[13] A. Galland, S. Abiteboul, A. Marian, P.Senellart, ”Corroborating information from disagreeing views”, *3rd ACM International Conference on Web Search and Data Mining (WSDM)*, February 2010, pp. 131-140.

[14] X. L. Dong, L. Berti-Equille, D. Srivastava, ”Data fusion: Resolving conflicts from multiple sources”, *Web-Age Information Management (WAIM)*, 2013, pp. 64-76.

[15] Pablo N. Mendes, Hannes Mühleisen, Christian Bizer, ”Slieve: Linked Data Quality Assesment and Fusion”, *Proceedings of the 2012 Joint EDBT/ICDT Workshops* pp. 116-123, ACM.

[16] http://ldif.wbsg.de/

[17] Jens Bleiholder, Felix Naumann, ”Data Fusion”, *ACM computing surveys (CSUR)*, *Vol. 41*, No. 1, pp. 1-41, 2009.

[18] M. Hernández, S. Stolfo, ”The merge/purge problem for large databases”, *Proceeding of the ACM SIGMOD International Conference on Management of Data*, pp. 127-138, May 1995.

[19] U. Draisbach, F. Naumann, S. Szott, O. Wonneberg, ”Adaptive Windows for Duplicate Detection”, *International Conference on Data Engineering (ICDE)*, IEEE, pp. 1073-1083, 2012.

[20] Alvaro E. Monge, ”Matching Algorithms within a Duplicate Detection System”, *IEEE Data Eng. Bull.*, *Vol. 23*, No. 4, pp. 14-20, 2009.

[21] D. Draper, A. Y. Halevy, D. S. Weld, ”The Nimble XML data integration system”, *Proceeding of the International Conference on Data Engineering(ICDE)*, IEEE Computer Society, pp. 155-160, 2001b.

[22] R. Ahmed, P. De Smedt, W. Du, W. Kent, M. A. Ketabchi, W. A. Litwin, A. Rafii, M.-C. Shan, ”The Pegasus Heterogeneus multidatabase system”, *IEEE Comput.*, *Vol. 24*, No. 12, pp. 19-27, 1991.

[23] C. Collet, M. N. Huhns, W.-M. Shen, ”Resource integraation using a large knowledge base in Carnot”, *IEEE Comput.*, *Vol. 24*, No. 12, pp. 55-62, 1991.

[24] R. J. Jr. Bayardo, W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, D. Woelk, ”InfoSleuth: Agent-Based semantic integration of information in open and dynamic environments”, *Proceeding of the ACM International Conference on Management of Data SIGMOD*, ACM Press, New York, pp. 195-206, 1997.

[25] J. Hammer, J. McHugh, H. Garcia-Molina, ”Semistructured data: The TSIMMIS experience.”, *Proceeding of the East European Conference on Advances in Databases and Information Systems (ADBIS)*, pp. 1-8, 1997.

[26] J. L. Ambite, C. A. Knoblock, I. Muslea, A. G. Philpot, ”Compiling source descriptions for efficient and flexible information integration”, *J. Intell. Inf. Syst.*, *Vol. 16*, No. 2, pp. 149-187, 2001.

[27] N. Leone, G. Greco, G. Ianni, V. Lio, G. Terracina, T.Eiter, W. Faber, M. Fink, G. Gottlob, R.Rosati, D. Lembo, M. Lenzerini, M. Ruzzi, E. Kalka, B. Nowiciki, W. Staniszkis, ”The INFOMIX system for advanced integration of incomplete and inconsistent data.”, *Proceeding of the ACM International Conference on Management of Data SIGMOD*, pp. 915-917, 2005.

[28] U. Dayal, ”Processing queries over generalization hierarchies in a multidatabase system”, *Proceeding of the International Conference on Very Large Database (VLDB)*, pp. 342-353, 1983.

[29] A. Bilke, J. Bleiholder, C. Böhm, K. Draba, F. Naumann, M. Weis, ”Automatic data fusion with HumMer”, *Proceedings of the International Conference on Very Large Databases (VLDB)*, pp. 1251-1254, 2005.

[30] V. S. Subrahmanian, S. Adali, A. Brink, R. Emery, J. Lu, A. Rajput, T. Rogers, R. Ross, C. Ward, ”Hermes: a heterogeneous reasoning and mediation system”, Tech. Rep., University of Maryland, 1995.

[31] https://www.techeconomy.it/2016/03/25/openstreetmap-e-google-maps/

[32] https://developers.google.com/maps/documentation/javascript/

[33] https://sequelize.org/master/

[34] https://en.wikipedia.org/wiki/OpenStreetMap

[35] N. Borolea, D. Routa, N. Goela, Dr. P. Vedagirib, Dr. Tom V. Mathewb, ”Multimodal Public Transit Trip Planner with Real-Time Transit Data ”, *Procedia - Social and Behavioral Sciences*, *No. 104* , 2013, pp. 775 – 784

[36] https://beyondtransparency.org/chapters/part-2/pioneering-open-data-standards-the-gtfs-story/

[37] https://github.com/CanalTP/navitia/wiki/OpenTripPlanner-and-Navitia-comparison

[38] https://www.predictiveanalyticstoday.com/google-public-data-explorer/

[39] https://aperto.comune.torino.it

[40] https://data.europa.eu/euodp/en/data/

[41] https://facebook.github.io/flux/

[42] https://www.wikidata.org/wiki/Wikidata:Introduction

[43] https://en.wikipedia.org/wiki/DBpedia

[44] https://www.istat.it/it/archivio/236148

[45] https://developer.apple.com/documentation/mapkit

[46] https://www.bingmapsportal.com/

[47] https://en.wikipedia.org/wiki/Object-relational_mapping

[48] https://en.wikipedia.org/wiki/Client%E2%80%93server_model

[49] https://searchapparchitecture.techtarget.com/definition/RESTful-API

[50] https://en.wikipedia.org/wiki/Representational_state_transfer

[51] https://medium.com/of-all-things-tech-progress/understanding-mvc-architecture-with-react-6cd38e91fefd

[52] https://medium.com/mofed/react-redux-architecture-overview-7b3e52004b6e

[53] https://reactjs.org/

[54] https://en.wikipedia.org/wiki/React_(web_framework)

[55] https://www.html.it/guide/react-la-guida/

[56] https://reacttraining.com/react-router/web/guides/quick-start

[57] https://redux.js.org/api/api-reference

[58] https://medium.com/the-web-tub/managing-your-react-state-with-redux-affab72de4b1

[59] http://expressjs.com/en/api

[60] https://nodejs.org/en/docs/

[61] https://nodejs.dev

[62] https://www.postgresql.org/docs/

[63] https://leafletjs.com/reference-1.6.0.html