Free University of Bolzano Faculty of Computer Science

Master of Science Thesis

The Allocation of Dynamic Objects Within an Isochrone

by

Sarunas Marciuska

Supervisor: Johann Gamper

Bolzano, 2010

Contents

Contents **i**

1 Introduction **5**

2 Review of the State of the Art **9**

3 Problem Statement **15**
- 3.1 Buffer Approach 16
- 3.2 Polygon Approach 17

4 Buffer Solution **19**

5 Polygon Solution **23**
- 5.1 Motivation 23
- 5.2 Background information 24
- 5.3 Basic Principals 25
- 5.4 Algorithm 27
- 5.5 Time Complexity 29
- 5.6 Implementation 30

6 Experiments and Discussion **31**
- 6.1 Experimental Environment 31
- 6.2 Precision, Recall, and F-measure 32
- 6.3 Experimental Analysis 36
- 6.4 Data Inconsistency 37
- 6.5 Summary 38

7 Conclusion and Further Studies **39**

Bibliography **41**

List of Figures **43**

List of Tables **44**

Acknowledgements

I am extremely thankful to my supervisor, Johann Gamper. His encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. I am especially thankful that I could jump into his office at any time - even without an appointment.

I owe my deepest gratitude to Andrea Janes. First of all, for remaining my friend, even though, I was unbearable during the time when I was writing this thesis. Thanks for reading my work and putting millions of articles in places (it is an open secret that Lithuanians are particularly bad at it).

It is a pleasure to thank Egle Kirdulyte. The designer's touch on some pictures saved me a huge amount of energy: I thought that it would take ages for the ugly duckling to become a beautiful swan.

I am indebted to Ingrida and Sveta for inviting me for lunch on Sundays. Because of my laziness to go to shops on Saturdays, I would have starved without their delicious food, and this thesis would not have been finished at all. I still remember the taste of the juicy beef that was the wind beneath my wings.

It is an honor for me to thank Linas, Marius and Max for going to ski-touring in the evenings. The tiredness and relaxation afterwards kept me from going crazy during the most difficult moments of this thesis.

Abstract

Urban planning addressed issues such as: putting strategic objects in optimal places and sending emergency forces in the area of an accident in the most optimal way. In this thesis we show how isochrone can be used to solve the before mentioned problems. An isochrone is all set of points that can be reached from the query location within a given time span. In this thesis we present two algorithms for obtaining objects within an isochrone. The first approach is based on constructing buffers around the links of the isochrone and making an intersection with those buffers to obtain the objects that we are searching for. The second approach is based on the computation of the area that the isochrone is covering. In this work we empirically evaluate both solutions using a real-world data set. We measure the quality of each solution by implementing the precise approach that is expensive in terms of time complexity. As quality estimators we use recall, precision, and f-measure. We present the running time of each approach by using a real-world data set. During the experiments we found data inconsistencies. We briefly describe how to overcome those inconsistencies.

Chapter 1

Introduction

Urban planning addressed issues such as: putting strategic objects in optimal places and sending emergency forces in the area of an accident in the most optimal way. An example of the former problem is to choose where to build a bus or metro station maximizing the number of people living nearby. In this way, we can maximize the utilization of the bus or metro station. Usually, the computation of the location of such objects is based on the time that is needed for people to reach them. The optimal position for the bus or metro stop is where the largest number of people can reach that place within the given time span. An example for the latter problem is the optimal emergency force distribution in the case of an accident. Usually, emergency forces are moving in a city and it is needed to sent the forces that can arrive fastest to the place of emergency.

In this work we show how an isochrone can be used to solve the before mentioned problems. An isochrone is a set of points that can be reached from a given location within a given time span. In our case given location is a place of accident, bus or metro stop and set of points are emergency forces and people. By grouping citizens or emergency forces databases with isochrones, it is straightforward to compute the number of inhabitants or emergency forces that are within an isochrone. To solve the first problem we have to find the biggest number of inhabitants by moving the center of the isochrone (bus or metro stop). To solve the second problem we have to find a needed number of emergency forces by increasing the time span of an isochrone.

An isochrone in a spatial database is represented as a set of links within a street network and the objects that we are searching for can be outside that network. For example, pedestrians can be in the park but not on a street. This means that in order to find objects within an isochrone we have to consider area that is reachable within it. The problem of finding objects within an isochrone is to find the objects that are within the area that can be reached from the isochrone. One way to construct such area is to draw a buffer around each link of the isochrone. Another way is to devise an algorithm that constructs that area from links of the isochrone. An intuitive solution for finding objects within an isochrone is to add them into the result together with the creation of the isochrone. However, such an approach is very expensive in terms of time complexity, since there is no a relation between the objects and the isochrone itself in the database.

Example 1

Figure 1.1 shows a 20 minutes isochrone with a single query location and a database of houses. The query location is presented as a star, all places that can be reached from it within 20 minutes are presented as a bold lines, houses are presented as dots. This figure represents a snapshot of a data that we were using in our experiments.

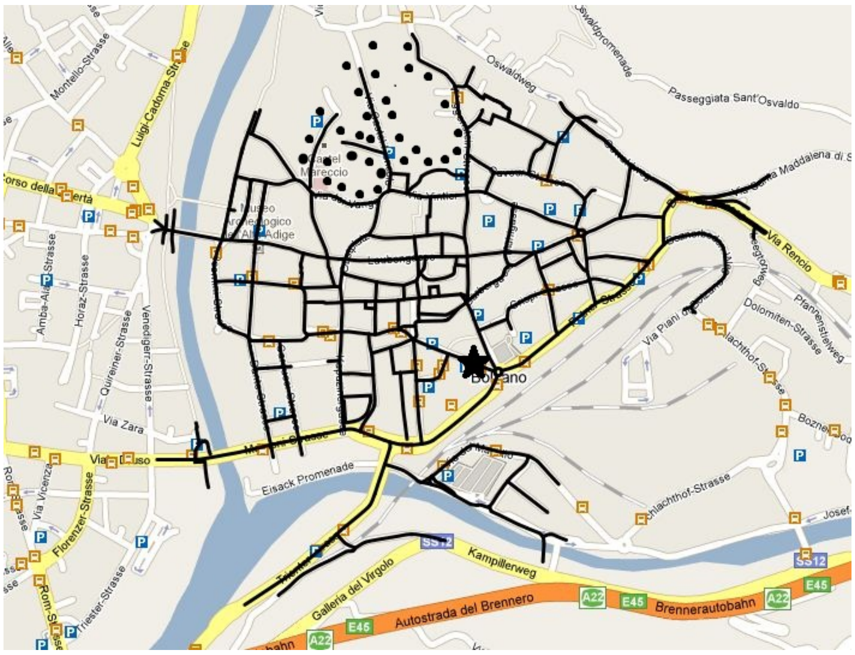


Figure 1.1: The isochrone

To achieve the desired result we propose to make an intersection with the area that the isochrone is covering or buffers around links of the isochrone and the objects that are within that area. The computation of the area that an isochrone is covering is similar to the computation of a footprint of a set of points. The most famous methods for this computation are concave hull and alpha shapes. However, to apply the concept of isochrones in the field of urban planning, we repesent isochrones as a set of links rather a set of points. For computing the covering area we adapted an algorithm that is similar to the computation of a convex hull. There are two main algorithms for computing the convex hull: Jarvis march and Graham scan. In our work we adapt the algorithm following the Jarvis march approach.

In this thesis we present two algorithms for obtaining objects within an isochrone. The first approach is based on constructing buffers around the links of the isochrone and making an intersection with those buffers to obtain the objects that we are searching for. The implementation of this solution is mainly based on existing spatial database functionalities. However, in existing spatial databases a link buffer is created only virtually. Such buffers are used for making an intersection with objects, but after obtaining the results they are not stored in the database.

The second approach is based on the computation of the area that the isochrone is covering. This area is represented as a polygon. As final result we make an intersection with that polygon and the objects that we are searching for. In this work we give a brief description of the algorithm that computes the before mentioned polygon. We present the pseudo code and the runtime complexity of the algorithm.

In this work we empirically evaluate both solutions using a real-world data set. We measure the quality of each solution by implementing the precise approach that is expensive in terms of time complexity. As quality estimators we use recall, precision, and f-measure. We present the running time of each approach by using a real-world data set. During the experiments we found data inconsistencies. We briefly describe how to overcome those inconsistencies.

The thesis is organized as follows: in chapter 2 we present related work, in chapter 3 we describe the scope of the problem and the overview of the possible solutions, chapter 4 and chapter 5 present two different algorithms for finding objects within an isochrone, in chapter 6 we present an experimental results, and chapter 7 presents conclusions and future work.

Chapter 2

Review of the State of the Art

In this chapter we present related works for finding objects within an isochrone. We begin introducing research areas that are closest to the computation of an isochrone. Those areas deal with range queries and shortest path algorithms. In order to obtain objects within an isochrone in the most efficient way, an index must be created. In this chapter we present research that deals with the indexing system of spatial databases. The last part presents related works to the computation of the covering area of an isochrone. Those works deal with the computation of convex hull, concave hull, and alpha shapes.

The creation of an isochrone is a new research area in spatial databases. It is related to range queries. For example, one problem could be to find all houses that are witin 50 meters from the street. Papadias et. al. [12] present a range query algorithms: Range Euclidean Restriction algorithm and Range Network expansion algorithm. A range query retrieves all objects that are within a given network distance d. The Range Euclidean algorithm retrieves all objects that are within Euclidean distance d. The Range network expansion algorithm retrieves all objects that are within network distance d.

Even though range queries are the closest concept to isochrones, the computation of isochrones adapts Dijkstra's [1] incremental network expansion strategy. It is the most popular shortest path algorithm. It starts from the source and expands route till the destination is reached.

Gamper et al. [5] present a computation of isochrones in a bimodal spatial network that consists of two networks: a pedestrian network and one or more bus networks. The isochrone is computed using a bimodal incremental network expansion algorithm that incrementaly expands each individual network. In our work we consider a spatial network that consists of one pedestrian network.

In order to retrieve the objects efficiently within an isochrone one way is to put an index on them. An example of such use of indexes can be found in spatial databases to retrieve spatial objects. Beckmann et al. [10] introduce the R-tree method for storing complex spatial objects. For each spatial object a minimum bounding rectangle (MBR) is created that encloses the object. All MBRs create a hierarchy, which is stored in a R-tree structure. This method is used in spatial databases in order to index spatial objects. R-tree index is a B-tree-like hierarchical structure. It is used to store real-world geometric objects such as streets, houses, parks, lakes, rivers etc. R-tree indexing is widely used to answer range queries. The structure of R-tree is presented in Figure 2.1.

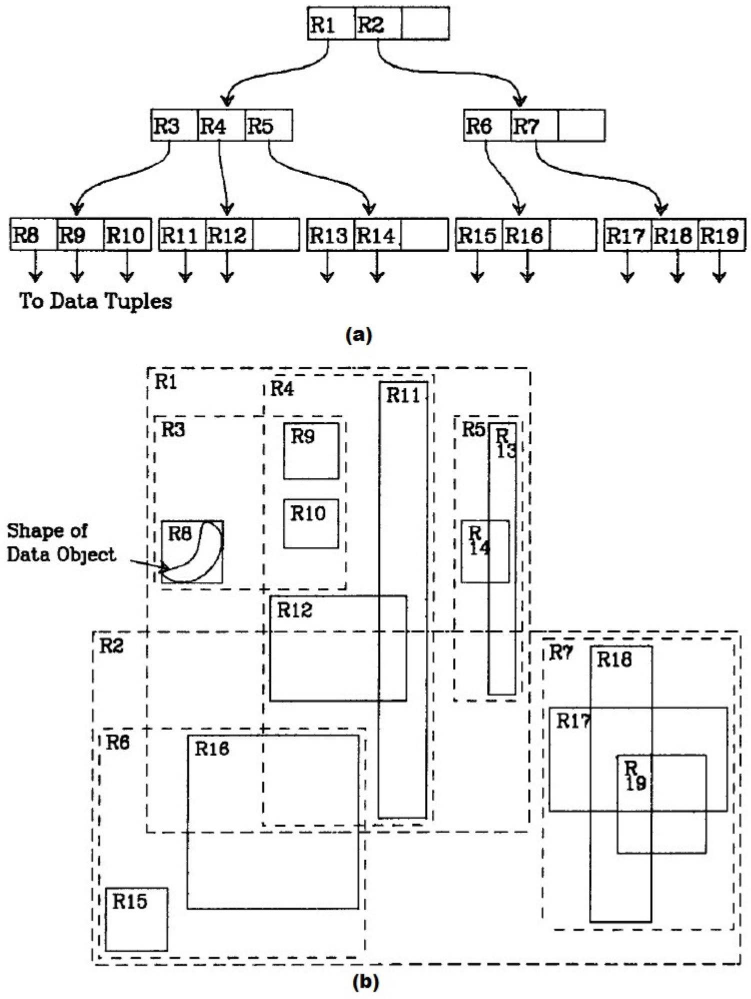


Figure 2.1: R-tree for 2D objects

Each node of R-tree stores a link to a child (Figure 2.1 (a)) and a bounding box of all entities that are within its child nodes (Figure 2.1(b)). Using an R-tree indexing system the answer if two object are intersecting is straight forward. The main assumption that the algorithm presented in chapter 4 will work faster then algorithm presented in chapter 3 is based on the R-tree indexing system.

R-trees are based on B-tree that was presented by Bayer [8]. A B-tree is designed to contain a large number of keys in each node so that the height of the tree is relatively small. This means that only a small number of nodes must be read from the disk to obtain an item. The aim is to get fast access to the data by reading a very small number of records.

The algorithm presented in this thesis is built on the same idea to create an index to retrieve spatial objects and consists of a part that describes how to compute a bounding polygon from the links of an isochrone. This part is closest to a convex hull computation. However, it contains $2D$ links while a convex hull computation is based on $2D$ points.

We can not use convex hull approach in our algorithm by splitting links into $2D$ points, because it can happen that links that originally are not within isochrone will be added to the convex hull. The convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior. Figure 2.2 presents the convex hull for a set of points.

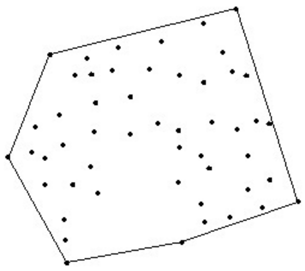


Figure 2.2: Convex hull

Two main algorithms are known for creating a convex hull: Jarvis March and Graham scan.

The *Jarvis March* [6] approach describes a way to find a convex hull for a given finite number of $2D$ points. The main idea of the algorithm is to put a point on a convex hull that has the smallest polar angle with respect to a previous point. The first point is taken as a left most point from all points. This algorithm runs in $O(nh)$ time, where n is a given number of points in the data set and h is a number of points on the convex hull.

The graham scan approach is based on computation of left and right turns. To determine if three points p_1, p_2 and p_3 make a right turn we have to compute the cross product.

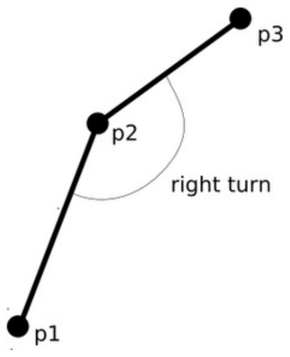


Figure 2.3: Right turn

Let $v_1 = p_1 - p_2$ and $v_2 = p_3 - p_2$. If a cross product v_1 x v_2 is greater than 0, then the set of points p_1, p_2 and p_3 make a right turn, otherwise they make a left turn.

The *Graham scan* [4] approach computes the convex hull in $O(n\ log\ n)$ time for a given finite number of $2D$ points. The algorithm is processed in three steps. First, the point that has the smallest y coordinate among all points is chosen as the first point. Second, the remaining points are sorted according to the x axis increasingly. Finally, for each next point in a convex hull, the turn between the point and the previous two points is computed. If it is a right turn, this means that the second to last point has to be removed from the convex hull. This process is continued while last three points make a right turn. If a left turn occurs, the last point is put to the convex hull and the next point is taken from the sorted array.

The main idea of our algorithm is similar to Jarvis March's approach. The visual result of our computed polygon is closer to a concave hull then to a convex hull. We can not use an existing concave hull algorithm because our data model is based on $2D$ links but not on $2D$ points. Even if we split links into $2D$ points it can happen that the edge of a concave hull intersects with the edge of a link.The computation of a concave hull was presented by Moreira et. al. [7] . Their approach is based on the k-nearest neighbors computation in the region occupied by a set of points. A Concave hull is a hull that is not convex. Usually, concave hulls are used to find footprints or patterns within given set of points. It is difficult to judge which concave hull gives the best results, because more than one concave hull can be created for the same set of points. Figure 2.4 presents one of the possible concave hulls.

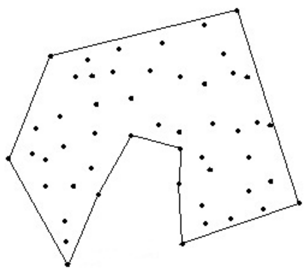


Figure 2.4: Concave hull

A similar problem for finding footprints from a set of $2D$ points was discussed by Galton [2] and Edelsbrunner ([11],[3],[9]). Those approaches are based on alpha shapes. The main idea of the alpha shape algorithm is to put the circuits with a radius $1/alpha$ that they would touch at least two points and none of the rest points would be inside those circuits. Then all points that touch the circuits are selected and connected with lines. The result is a footprint of those points. If the radius is big enough the result will be a convex hull. If radius is too small then the result will be all points but not connected by lines. The alpha shape algorithm can not be used as well to achieve the desired result, because it is based on a k-nearest neighbor approach and the number of the neighbors have to be adapted to each isochrone separately. Figure 2.5 presents the result of alpha shape.

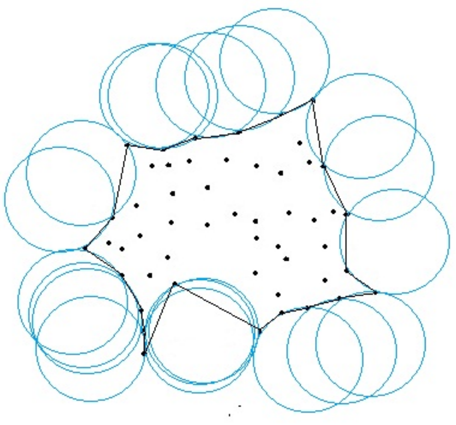


Figure 2.5: Alpha shapes

In this thesis we present a Buffer solution for finding objects within an isochrone. Moreover, we present a Polygon solution to achieve the desired result. The Polygon solution improves the Buffer solution by preparing the data in such a way that the R-tree indexing system manages to analyze it faster. In our work we modified Jarvis March's approach to calculate the bounding polygon. The main modification was made to adapt the algorithm for a $2D$ connected network links instead of $2D$ points. We have chosen this algorithm because of the following reason: we assumed that the number of links on the bounding polygon is smaller than *log* n where n is the number of links within the *isochrone*.

Chapter 3

Problem Statement

In this chapter we define the problem of finding objects within an isochrone. We present the intuitive solution for finding objects within an isochrone that is precise, but expensive in terms of time complexity. In order to achieve the desired results in a more efficient way we introduce approximate solutions based on the creation of buffers around each link of an isochrone and based on the creation of a bounding polygon of the isochrone.

In order to define the problem of finding objects within an isochrone it should be said that an isochrone is stored as a set of links obtained from a street network and objects that we are searching for can be outside that network. For example, pedestrians can be in the park but not on a street. This means that in order to find objects within an isochrone we have to consider the area that is reachable within it. The problem of finding objects within an isochrone is to find the objects that are within the area that can be reached from the isochrone.

The Baseline solution for finding objects within an isochrone is based on adding objects to the result set together with the creation of the isochrone by checking if there is enough time to reach the object. Firstly, we would have to make a projection of all objects on the links, because it is most probable that objects can be reached from the closest link. Secondly, distances between objects and links should be computed on the projection point.

Example 2
An example of before mentioned idea is shown in Figure 3.1.

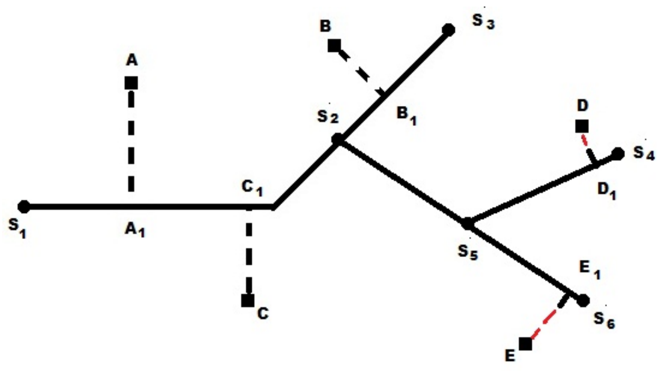


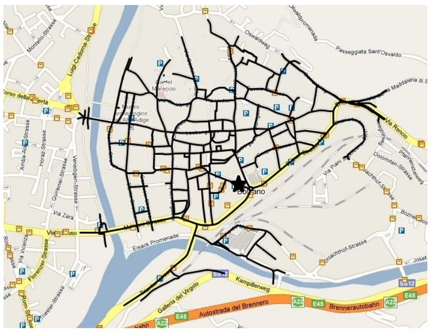
Figure 3.1: Projected points

First of all, the projections of all points are made on the closest links. Point A is projected as A_1 on link S_1S_2, point B is projected as B_1 on link S_2S_3, point C is projected as C_1 on link S_1S_2, Point D is projected as D_1 on link S_5S_4, Point E is projected as E_1 on link S_5S_6. Secondly, the distances the projection points until the starting points of links on which those points are projected have to be calculated: S_1A_1, S_1C_1, S_2B_1, S_5D_1, S_5E_1. Then the distances from the projection points till the initial points have to be calculated: A_1A, B_1B, C_1C, D_1D, E_1E. Finally, the points that can be reached are added together with construction of the isochrone. Let say that the starting point of the isochrone is S_1 then we compute the distance that can be covered from that point by the multiplication of a default walking speed and time left. We compare that distance with the distance needed to reach points that are projected on link S_1S_2. The distance needed to reach point A is $S_1A_1 + A_1A$ and the distance to reach point C is $S_1C_1 + C_1C$. From the figure we can see that those points can be reached from point S_1 and they have to be added to the result set. In the same way we add point C into the result set. However, points D and E can not be reached within an isochrone and they are not added to the final result set.

The main problem of the Baseline solution is that it is very ineffective, because the computation of the distances between objects and links has to be split into complex geometry computations that are very expensive in terms of time complexity. We propose two alternative solutions that are more efficient than the Baseline approach. The first solution is based on the creation of a buffer around each link of the isochrone. The second solution is based on the creation of a polygon around the isochrone. Both approaches are approximate, because buffers and polygons are just an approximation of an area that can be reached within the isochrone.

3.1 Buffer Approach

The Buffer solution to find objects within isochrone is based on the creation of a buffer around each link of the isochrone. To achieve the desired result we have to compute the intersection between those buffers and objects. The main element of such solution is the isochrone. The isochrone itself can be constructed from a multimodal network that contains bus, train, pedestrian networks, etc. In this work we use isochrones that contain only a pedestrian network. A pedestrian network is a network that contains streets where pedestrians can walk. It is represented as a set of links. Isochrones are then computed based on the pedestrian network. Even though a pedestrian network is represented by full links - isochrones can contain a partial links. Partial links are added to the isochrone when they are partially reachable because of a time constraint. In order to have the same data structure for full and partial links, we split partial links on the last reachable point and create a new link. In such a way we convert partial links to normal links and the isochrone contains links that are fully reachable. The isochrone is stored in the database in the *isochrone* table presented in Figure 3.2 (b).



(a)

Link ID	Geometry
1	geometry
2	geometry
...	...
3399	geometry
3400	geometry

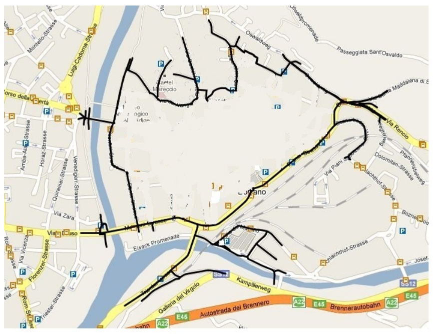
(b)

Figure 3.2: The structure of an isochrone

In this application, an isochrone is a set of all pedestrian network links that are fully reachable from a query location within a specified time constraint and a set of all transformed pedestrian network links that are partially reachable from a query location within a specified time constraint. The approach that is creating buffers around links of the isochrone to achieve the desired result is presented in chapter 4.

3.2 Polygon Approach

The last approach presented in this work is based on the computation of a bounding polygon of the isochrone. To achieve the desired result the intersection between objects and the bounding polygon is computed. The Polygon approach is presented in chapter 5. This bounding polygon is called the surface of the isochrone. The surface of an isochrone is a minimum set of links from the isochrone that creates a polygon, enclosing the isochrone.



(a)

Link ID	Geometry
1	geometry

(b)

Figure 3.3: The structure of an isochrone's surface

The surface of the isochrone is stored in the database in a *surface of isochrone* table presented in Figure 3.3(b). The surface of the isochrone that is shown in Figure 3.2 (a) is presented in Figure 3.3 (a).

The main scope of this thesis is the implementation and analysis of the Buffer and the Polygon solution. Those solutions are approximate solutions. To measure the quality of the results of each approach we compared them with the Baseline solution in chapter 6.

Chapter 4

Buffer Solution

In this chapter we describe the solution that is based on the creation of buffers in order to find objects within an isochrone. We present the relation between objects and the isochrone. For the implementation of before mentioned solution we present the main spatial operators, because they will be used in order to create buffers. As the final result of this chapter we present the implementation of the *SDO_WITHIN_DISTANCE* operator for obtaining the objects within an isochrone. It creates a virtual buffer around each link and obtains objects that are intersecting with that buffer. In this particular solution buffers are not stored in the database.

In order to solve the problem of finding objects within an isochrone the structure of the isochrone and objects have to be described in more detail. First of all, we assume that the objects we are searching for are points and the isochrone is made of a set of links. Since we are considering that points are moving with respect to the isochrone we assume that there is no relationship between points and links. This means that a point can be anywhere on a given time point and it is not assigned to any link. As a practical demonstration of the feasibility of our approach we use information about Bolzano houses and streets. We use houses as points and streets as a set of links from which the isochrone is created. A Search in the isochrone gives different results during different time moments, because objects are moving. We assume that Bolzano houses represent the snapshot of moving objects on a certain time moment.

We use the Oracle spatial database management system for our experiments. It has three main spatial operators: *SDO_WITHIN_DISTANCE*, *SDO_NN* and *SDO_RELATE*. *SDO_WITHIN_DISTANCE* is used to find all data within specified distance, *SDO_NN* finds the nearest neighbors to the a query location, and *SDO_RELATE* retrieves the neighbors that intersect or relate to a query location. To solve our problem the *SDO_NN* operator can not be used because we do not know the exact number of houses within an isochrone. This is the reason why why can not specify the number of neighbors that *SDO_NN* function has to return. The best solution for this problem is to use the *SDO_WITHIN_DISTANCE* (*within-distance*) operator, because it does not require the manual creation of a buffer around links and retrieves all objects within specified distance.

The *within-distance* operator determines if two spatial objects, A and B, are within a specified distance of one another. This operator first constructs a distance buffer D_b around the reference object B. It then checks that A and D_b are non-disjoint. Two objects are disjoint if they do not intersect. In our case A is the object that we are searching for and B is the link in the *isochrone*. Note: the construction of buffers is only virtual, they are not stored in the database and can not be visualized.

We implemented the within-distance function to achieve the desired result. The implementation of the *within-distance* function is shown in Figure 4.1. As an input we use two parameters: all links from the Isochrone table (Isochrone); all objects from the Bolzano houses table (Objects). The before mentioned tables are presented in chapter 6. The algorithm retrieves a set of of objects that are within specified distance from the isochrone.

```
Algorithm: GetObjectsFromIsochrone(Isochrone, Objects, d)
Input: Isochrone; Objects; d
Output: a set of of objects that are within specified distance d from the
        isochrone;
SELECT DISTINCT $Objects.ID$
FROM $Objects$ O, $Isochrone$ I
WHERE SDO_WITHIN_DISTANCE(
$O.Geometry$, $I.Geometry$,$'DISTANCE = d\ UNIT = METER') =' TRUE'$
```

Figure 4.1: Buffer solution for finding dynamic objects

The query is processed in the following way:

1. All links from the isochrone are selected. Those links are taken from the database from the *Isochrone* table.
2. A virtual buffer of size *d* is created around all selected links. This buffer is created by *SDO_WITHIN_DISTANCE* function.
3. All objects are selected that are intersecting with that virtual buffer. In our experiments, houses were used as a representation of those objects.

Example 3
Figure 4.2 shows how the *within-distance* function computes the result:

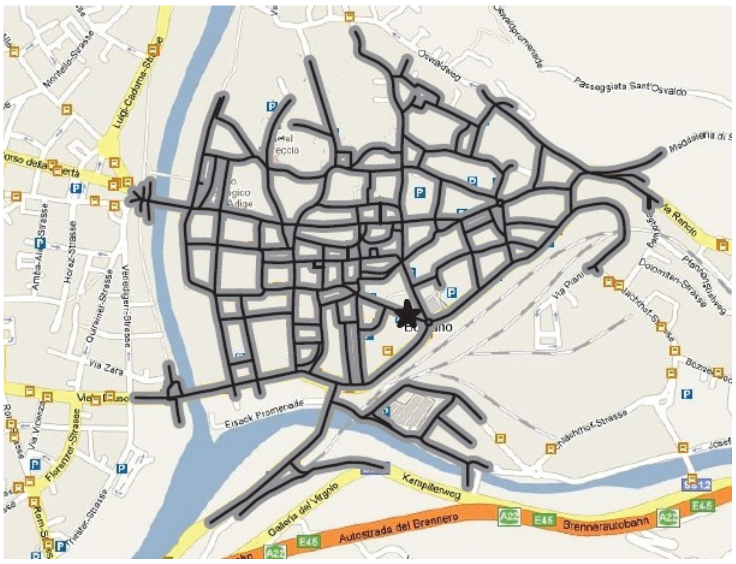


Figure 4.2: The distance buffer

In our example we would get 2893 objects as result if we took the isochrone with the following parameters: default walking speed 30 m/min, the starting point of the isochrone is located in the Bolzano city center, the size of the isochrone is 20 min and the distance buffer size is 10 m.

If the *isochrone* has more than one link, more than one distance buffer is created. For searching objects within the distance buffers the function *within-distance* is using an R-tree index. It looks for all objects that are intersecting with distance buffers within that tree.

Chapter 5

Polygon Solution

In this chapter we present the main idea why we should use a covering polygon for allocating the objects within the isochrone. We give all background information needed to understand the algorithm that is creating the surface of the isochrone. In this chapter we present the algorithm, pseudo code and the runtime complexity of the approach that is creating the polygon that covers an isochrone. At the end we present an implementation of the Polygon solution.

5.1 Motivation

In this section we present the main idea why a covering polygon should be used for allocating the objcets within an isochrone. We give an overview why simillar existing solutions can not be applied.

The $within - distance$ function uses R-Tree index to compute objects that are intersecting. An isochrone contains a big number of links and all those links are connected. To improve the performance of the $within - distance$ function we propose an improvement for the object allocation: instead of making an intersection with all link buffers, one covering polygon should be created in such a way that the intersection would be calculated only with one polygon instead of many.

The covering polygon would be similar to a concave hull. However, we can not use an existing concave hull algorithm because our data model is based on $2D$ links but not on $2D$ points. Even if we split links into $2D$ points it can happen that the edge of a concave hull intersects with the edge of a link. The alpha shape algorithm can not be used as well, because it is based on a k-nearest neighbor approach and the number of the neighbors have to be adapted to each isochrone separately. This is why this solution is not robust. To avoid the intersection of edges we could use the convex hull approach by splitting links in $2D$ points. In such a case it can happen that links that originally are not within isochrone will be added to the convex hull. In our work we propose to use a modified convex hull approach to compute the covering polygon of an isochrone.

5.2 Background information

In this section we explain the structure of a link. We present the concepts of clockwise and counter clockwise angles because the selection of links to the surface of the isochrone is based on the computation of the counterclockwise angle.

All links of an isochrone are defined by a set of points $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ where (x_1, y_1) is the first point, (x_2, y_2) is the second point, $\ldots$, (x_l, y_l) is the last point. In our algorithm we use only four points of the link: the first, the second, the last before last and the last. We use a notation where l from (x_l, y_l) doesn't represent the number of the point in the sequence, but it represents the last point. For example, $L_1\{(x_l, y_l)\}$ represents the last point of link L_1 and $L_2\{(x_l, y_l)\}$ represents the last point of link L_2, but it doesn't mean that L_1 and L_2 have the same number of points. Formally, link is a set of points connected with lines or arcs. Usually, a street network is represented as a set of links. A single street can be represented as a set of links or only one link. If a street has cross roads then it is represented as a set of links and in the other case as a one link. In our database a single link is stored as a single geometry object. In order to obtain a list of points that describe that link we have to extract them from the geometry object.

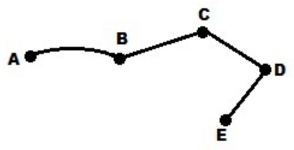


Figure 5.1: Link

In Figure 5.1 we present the structure of a link that is used in our algorithms. It contains five points: A, B, C, D, E. A and B points are connected with an arc and the rest points are connected with lines.

The computation of a counterclockwise angle is the main issue that defines if the link belongs to the surface of the isochrone or not. To define clockwise and counter clockwise angles we are using Figure 5.2.

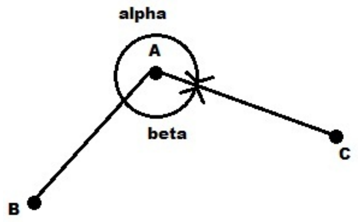


Figure 5.2: Clockwise and counterclockwise angles

Angle is clockwise between two lines AB and AC with respect to line AB if the lines AC are making clockwise turn with respect to line AB. Angle α is clockwise between lines AB and AC with respect to the line AB. Angle β is counterclockwise between lines AB and AC with respect to the line AB.

5.3 Basic Principals

In this section we present the main idea of the algorithm for computing the surface of an isochrone. We give a definition of the left most link and prove that the left most link belongs to the surface of the isochrone, because the left most link is the starting point of the Polygon algorithm.

Lemma 1. *A link that has the smallest x value of the first point (x_1) and the smallest clockwise angle with the Y axis and the line made from (x_1, y_1) and (x_2, y_2) points (comparing to the other links that have the same first point) belongs to the surface of the isochrone. Such link is called the left most link of the isochrone.*

Let the first point of the left most link of the isochrone does not belong to its surface.Then it can be inside or outside of the surface polygon. If it is outside that polygon then the polygon does not cover the isochrone and it is a contradiction to the isochrone's surface definition. If the left most link is in the isochrone's surface polygon then the x_1 value is not a minimum and it is a contradiction to the definition.

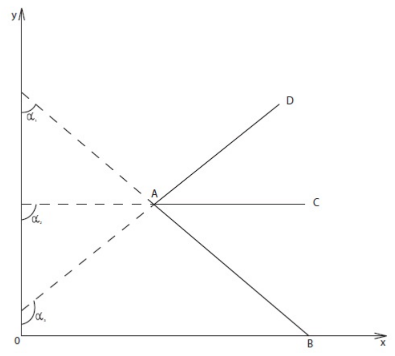


Figure 5.3: Angles between links

To prove the second part of Lemma 1 we use Figure 5.3. Let point $A = (x_1, y_1)$ be the left most point among all links such that has the smallest x_1 value. Let the points $B = (x_2, y_2), C = (x_2, y_2), D = (x_2, y_2)$ be the second points of links. If the angle OYB is the smallest comparing with OYC and OYD it means that AB is a covering line because there are other lines that start on point A and create a polygon where line AB would be inside that polygon but not on the edge. If A belongs to the surface of the isochrone then AB belongs to the surface of the isochrone. Since A has the left most point among other links it means that A belongs to the surface of the isochrone.$\Rightarrow$ AB belongs to the surface of the isochrone. $\Rightarrow$ Then link that contains line AB belongs to surface of the isochrone.

Having an *isochrone* as an input, the algorithm returns the surface of the isochrone as an output. The main idea of the algorithm is to find the left most link in the isochrone and walk around the edge of the polygon that is covering an isochrone using that link as the starting link until it was reached again.

The algorithm can be defined using the following rules:

1. The initial link must be on the surface of the isochrone. This is the reason why we are taking the left most link from the isochrone. If more then one *link* was found

- the *link* that has the smallest counter clockwise angle with respect to the x axis must be taken.

When the initial link is chosen the next links must be repeatedly chosen in the following way:

a) The next link must have a common point with the previous link.

b) The next link must have the smallest counter clockwise angle out of all links that satisfy the (a) condition.

2. The surface of the isochrone is calculated when next link has the same last point as the first point of the initial link. The last point of the next link is the point that is not common with the previous link. The first point of the initial link is the most left point in that link according to the x ordinate.

Example 4
The algorithm's steps can be depicted with the following example:

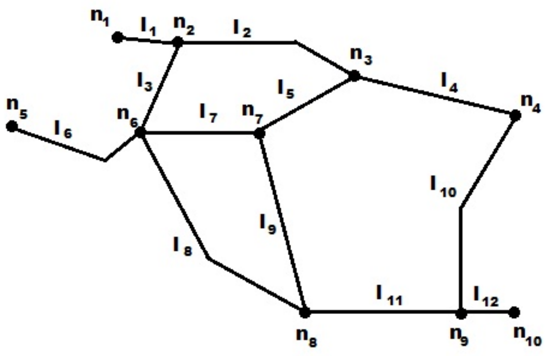


Figure 5.4: Example

In this example isochrone consists of twelve links $\{l_1, \ldots, l_{12}\}$ and ten nodes $n_1, \ldots, n_{10}\}$. As a starting link we take link l_6 because it is the most left link. We add all its points to the polygon that will be the surface of the isochrone. In the next step l_3, l_7 and l_8 links are selected as a possible links that could be added to the surface of the isochrone, because they have a common point with the previous link l_6 on a node n_6. The link l_8 has the smallest counterclockwise angle with the link l_6. We add link's l_8 points to the polygon that will be the surface of the isochrone. During the next step we chose links the l_9 and $l_1 1$ because they have a common point with the link l_8 on a node n_8. The link $l_1 1$ has the smallest counterclockwise angle with the link l_8. We add all its points to the polygon that will be the surface of the isochrone. In the same way we add link $l_1 2$ to the surface of the isochrone. It has a common point only with itself, in such a case we add $l_1 2$ points to the surface of the isochrone in a reverse order. The algorithm terminates when the reverse link's l_6 points are added to the surface of the isochrone. In our data set links do not have direction. Reverse notation is used in order to distinguish that the points of the same link are added in the different order for the construction of polygon. As a final result the surface of the isochrone contains the points of the following links: l_6, l_8, $l_1 1$, $l_1 2$, reverse $l_1 2$, $l_1 0$, l_4, l_2, l_1, reverse l_1, l_3 and reverse l_6. As a final result the surface of the isochrone drops information about three links: l_5, l_7, l_9. When the surface of the isochrone is computed from an isochrone that has a lot of links this information drop is even more visible.

Note:If we would remove links l_1 and l_6 from our example then the left most link would start on the node n_6. In such a case the left most link could be: l_3, l_7 or l_8. Link l_7 obviously does not belong to the surface of the isochrone. This is the reason why we proved that the left most link must have the smallest counter clockwise angle with respect to the Y axis.

5.4 Algorithm

In this section we present the pseudo code of the algorithm and its separate parts.

In the Figure 5.5 we present the pseudo code for the entire algorithm.

```
Algorithm: CreateIsochronesSurface(Isochrone, Objects)
Input: isochrone = $\{L_1, \dots, L_n\}$
Output: a set of links of the surface of the isochrone (surface)
initLink = FindTheMostLeftLink(isochrone);
$\{surface\} \leftarrow$ initLink ;
firstPoint = initLink $(x_1, y_1)$;
lastPoint = initLink $(x_n, y_n)$ ;
$L_l = initLink$;
while firstPoint $\neq$ lastPoint do
    $\{L_c\}$ = FindAllCommonLinks(isochrone,$\{L_c\}$);
    nextLink = FindTheLinkWithTheSmallestAngle($L_l$,);
    $L_l$ = nextLink;
    lastPoint = $L_l$ $(x_n, y_n)$ ;
    $\{surface\} \leftarrow$ nextLink;
end
return $\{surface\}$
```

Figure 5.5: Pseudocode for entire algorithm

The algorithm takes an isochrone as an input, finds the left most link of the isochrone and connects it to other links that belong to the surface of the isochrone.

Figure 5.6 shows the pseudo code of the algorithm that finds the left most link of the isochrone (L_k). The algorithm has one input parameter: isochrone. Isochrone contains of a set of links $\{L_1, \dots, L_n\}$. Each link is defined by a set of ordinates $L_k = \{(x_1, y_1), \dots, (x_l, y_l)\}$.

```
Algorithm: FindTheMostLeftLink(isochrone)
Input: isochrone = $\{L_1, \dots, L_n\}$
Output: $L_k$
$L_k = L_1$;
foreach $L_b \in$ isochrone do
    if $L_b(x_1) < L_k(x_1)$ then
        $L_k = L_b$;
    end
    if $L_b(x_1, y_1) = L_k(x_1, y_1)$ AND $L_b\{(x_1, y_1), (x_2, y_2)\}$ angle with respect to $Y$
    axis $< L_k\{(x_1, y_1), (x_2, y_2)\}$ angle with respect to $Y$ axis then
        $L_k = L_b$;
    end
end
return $L_k$
```

Figure 5.6: Pseudo code for finding the left most link

To find the left most link we use the first point (x_1, y_1) and the second point (x_2, y_2) of each link . The first link (L_1) from the isochrone is taken as the initial the let most link (L_k). Then the algorithm goes through all links and finds the left most link that has the smallest x ordinate on its first point (x_1). If the algorithm finds another link that has the same point as the left most link then it is taken as the new left most link if it has the smallest angle with respect to Y axis.

Figure 5.7 presents the pseudo code to obtain a set of links $(\{L_c\})$ that have the common point with the last inserted link to the surface of the isochrone. It takes two parameters as an input: a set of links of the isochrone $\{L_1, \dots, L_n\}$, and the last inserted link to the surface of the isochrone (L_l).

```
Algorithm:FindAllCommonLinks(isochrone,$L_l$)
Input: isochrone = $\{L_1, \dots, L_n\}$ ;
$L_l$
Output: $\{L_c\}$
$\{L_c\} = \phi$ ;
foreach $L_b \in$ isochrone do
    if $L_l(x_l, y_l) = L_b(x_1, y_1)$ then
        $\{L_c\} \leftarrow L_b$ ;
    end
    if $L_l(x_l, y_l) = L_b(x_l, y_l)$ then
        $\{L_c\} \leftarrow inverseL_b$ ;
    end
end
return $\{L_c\}$
```

Figure 5.7: Pseudo code to obtain links that have common point with last inserted link

The algorithm goes through all the links of the isochrone and returns those that have the same first or last point comparing to the last point of the last inserted link to the isochrone. If the last point is equal then the link is inverted in order to simplify next step of the algorithm where the angle between links is computed.

Figure 5.8 presents the pseudo code for obtaining a link that has the smallest counter-clockwise angle with the last link from the surface of the isochrone (L_s). The algorithm takes two parameters as an input: a last inserted link to the surface of the isochrone (L_l); a set of links that have common point with the last inserted link to the surface of the isochrone ($\{L_c\}$).

```
Algorithm: FindTheLinkWithTheSmallestAngle($L_l$,$\{L_c\}$)
Input: $L_l$ ;
$\{L_c\} = \{L_1, \ldots, L_n\}$
Output: $L_s$
smallestAngleLink = $L_1$ ;
foreach $L_b \in \{L_c\}$ do
    if the angle between $L_l\{(x_l - 1, y_(l - 1)), (x_l, y_l)\}$ and $L_b\{(x_1, y_1), (x_2, y_2)\} \leq$
    the angle between $L_l\{(x_l - 1, y_(l - 1)), (x_l, y_l)\}$ and $L_s$ $\{(x_1, y_1), (x_2, y_2)\}$ then
        $L_s = L_b$ ;
    end
end
return smallestAngleLink
```

Figure 5.8: Pseudo code to obtain link that has the smallest counterclockwise angle with the last inserted link

The algorithm goes through the all links that have a common point with the last inserted link and finds the link that has a smallest counter clockwise angle with the last link. To compute the angle the algorithm takes a line made from two last points of the last link $L_l\{(x_l - 1, y_(l - 1)), (x_l, y_l)\}$ and a line made from two first points of a common link $L_b\{(x_1, y_1), (x_2, y_2)\}$. This is the reason why we inverted links in algorithm that is presented in Figure 5.7 when they had common last point, because now that point is always first and double checking if it is the last is not needed.

5.5 Time Complexity

In this section we present the time complexity of the algorithm and its separate parts.

The rule number (1) presented in Algorithm section can be run in $O(n)$ time because this part can be done while going through all the isochrone's links once.

The rule (a) presented in Algorithm section runs in $O(n)$ time as well because it needs a single isochrone links scan.

The rule (b) presented in Algorithm section can be run in $O(n)$ time because it needs to calculate the angles for the links that has common point with the previous link. The street network used in our experiments usually doesn't exceed $O(3)$ time complexity because most of the links are connected in cross roads of four links and it is necessary to calculated the angles between one of those links with the remaining three links.

The time complexity for entire algorithm is $O((n + n + n)h) = O(nh)$ where h is the number of links on the isochrone's surface.

5.6 Implementation

In this section we present the implementation of the Polygon solution for allocating objects within an isochrone.

The query for allocating objects within an isochrone using the Polygon approach is almost the same as in the Buffer's approach. The main difference lies in the input parameters. Instead of the isochrone as an input parameter the Polygon approach is using the surface of the isochrone. While the isochrone contains a set of links, the surface of the isochrone is represented as a single polygon. The algorithm to achieve the desired result using the Polygon approach is presented in Figure 5.9.

```
Algorithm: GetObjectsFromIsochrone(IsochroneSurface, Objects, d)
Input: IsochroneSurface; Objects; d
Output: a set of of objects that are within specified distance from the the surface
        of the isochrone;
SELECT DISTINCT $Objects.ID$
FROM $Objects$ O, $IsochroneSurface$ IS
WHERE SDO_WITHIN_DISTANCE(
$O.Geometry$, $IS.Geometry,'DISTANCE = d\ UNIT = METER') =' TRUE'$
```

Figure 5.9: Polygon solution for finding dynamic objects

Chapter 6

Experiments and Discussion

In this chapter we present the experimental environment in which all experiments were done. We introduce estimators such as recall, precision and f-measure. Using those estimators we measure the quality of each approach. In this chapter we present the runtime of each algorithm for allocating objects within an isochrone. During the experiments we found data inconsistencies. We briefly describe how to overcome those inconsistencies.

6.1 Experimental Environment

In this section we present data that we used for our experiments. We describe the environment in which we run those experiments.

For the experiments we used a map of the city of Bolzano that is located in Italy. Map objects were stored in an Oracle Spatial database. For the experiments we used the following tables: street network (Table 6.2) and bolzano houses (Table 6.1). The *street network* table is used for computing the isochrone. It had around 3400 links. The *bolzano houses* table is used as a representation of the objects(houses in our case) that we are searching for within the isochrone. It had around 12300 houses.

Table 6.1: Street network (Network)

Link ID	Geometry
1	geometry
2	geometry
...	...
3399	geometry
3400	geometry

Table 6.2: Bolzano houses (Objects)

House ID	Geometry
1	geometry
2	geometry
...	...
12299	geometry
12300	geometry

We ran the performance comparison on a Unix server that contains an Oracle Spatial database management system. The approaches presented in this work for allocating objects within an isochrone were implemented using the Java programming language. The isochrones were computed from the real city network of Bolzano, Italy.

6.2 Precision, Recall, and F-measure

In this section we present the quality measures to compute the goodness of the Polygon and Buffer algorithms. At the end of this section we show the results that were obtained using the before mentioned quality measures.

The $within - distance$ function creates a buffer around the links of the isochrone (in case of aBuffer solution) or the surface of the isochrone (in case of a Polygon solution).The user's specified buffers size d increases the area where objects are searched for. All objects that are within distance d from an isochrone or the surface of the isochrone, but are not reachable within given time, are included to the result. To avoid this inconsistency we decreased the initial size of the isochrone in the following way:

$$t_0 = t - d/w$$

Where t_0 is the decreased size of an isochrone, t is the initial size of the isochrone, d is the size of a buffer, and w is the walking speed.

According to the buffer's size, the Buffer solution and the Polygon solution will produce different results, because the Buffer solution is making an intersection with links and the Polygon solution is making an intersection with the minimum bounding polygon. The bigger the value of a distance buffer, the similar the results. We get such results because the objects obtained from the Buffer approach are a subset of the objects obtained from the Polygon approach.

To measure the precision we implemented the Baseline approach that uses to prestored distances between the objects and the starting nodes of links to which they are mapped. The execution time of the query that is prestoring the before mentioned distances is 30 minutes. We use the Baseline approach as the most precise one and compute precision, recall and f-measure for the Polygon and Buffer approaches. Recall, precision and f-measure are computed from two parameters: total retrieved objects, and total relevant objects. The relationship between those parameters is presented in Figure 6.1.

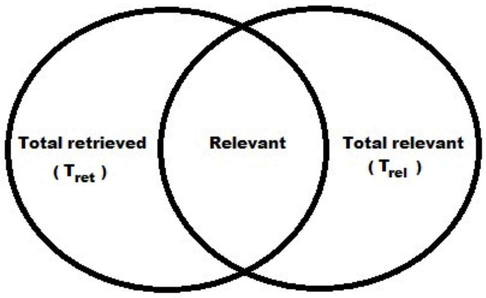


Figure 6.1: Relationship between recall and precision

Precision is defined as the number of relevant objects retrieved divided by the total number of objects retrieved. Recall is defined as the number of relevant objects retrieved divided by the total number of existing relevant objects (which should have been retrieved).

$$Precision = (T_{rel} \cap T_{ret})/(T_{ret})$$

$$Recall = (T_{rel} \cap T_{ret})/T_{rel})$$

Usually, if recall is increasing then precision is decreasing and vice versa. To measure the goodness of recall and precision f-measure is used.

$$F - measure = (2 * recall * precision)/(recall + precision)$$

In our experiments the total number of retrieved objects are the objects returned by the Buffer or Polygon approach and the total number of relevant objects are the objects returned by the Baseline solution.

The results obtained using the Buffer and Polygon solutions are influenced by three parameters: the location of the starting point of an isochrone, the size of the isochrone and the size of the distance buffer that is used in the $within - distance$ function.

To check the influence of the location of the starting point we arbitrary put it in five places of the city: in the center where the density of houses is high, in the center where density of houses is average, on the border of the city where the density of houses is high, on the border of the city where the density of houses is average, on the border of the city where the density of houses is low. The point in the center with low density is missing because there is now such place in Bolzano city. For experiments we were using fixed 15 min size of an isochrone with 30 m/ min walking speed and 30 m distance buffer that is used by the $within - distance$ function. To compare the quality of the results we used a f-measure estimator. The results of this experiment are presented in Figure 6.2.

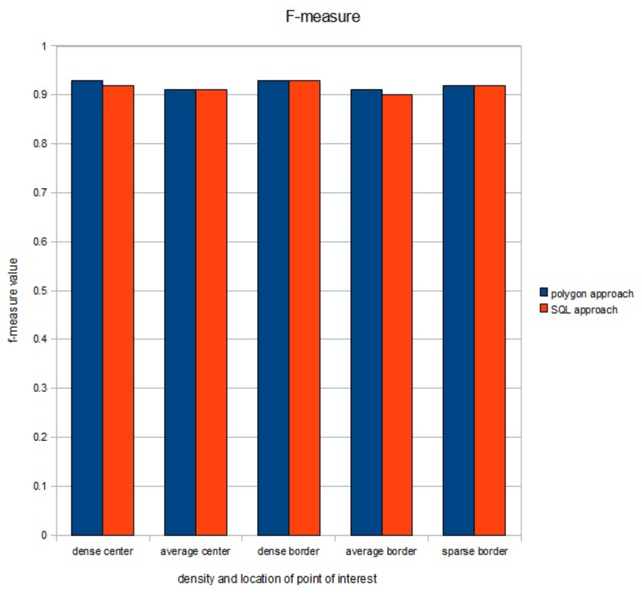


Figure 6.2: F-measure using different starting points

Figure 6.2 shows that the location of the starting point of an isochrone doesn't effect f-measure value significantly. The difference between the best and the worst case is 3 %. This is the reason why for the remaining experiments we use fixed starting point in the center and change the other two parameters.

To compute the f-measure, precision and recall we run experiments with different size of the isochrone and different buffer's size that is used in the $within - distance$ function.

The buffer's size was increased from 10 m to 120 m and the isochrone's size was increased from 10 min till 50 min. We chose 30 m/min as a default walking speed.

Figure 6.3 presents the precision of the Polygon and Buffer approaches using different parameters. Figure 6.3 (a) presents the precision of the Polygon approach and Figure 6.3 (b) presents the precision of the Buffer approach.

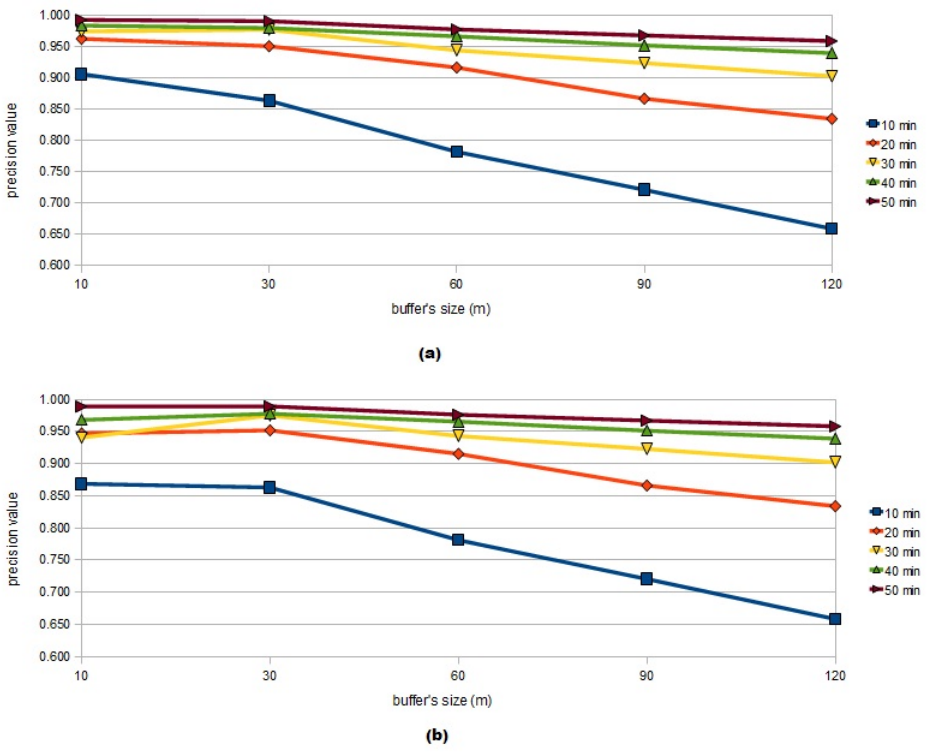


Figure 6.3: Precision

The experiments showed that the precision is influenced by the size of the isochrone in both approaches. The bigger the size of the isochrone the higher precision was obtained. The precision depends on the buffer's size in the Polygon and the Buffer approach. The experiments showed that the best precision was obtained when the buffer size was between 10 and 30 m.

Figure 6.4 presents the recall of the Polygon and Buffer approaches using different parameters. Figure 6.4 (a) presents the recall of the Polygon approach and Figure 6.4 (b) presents the recall of the Buffer approach.

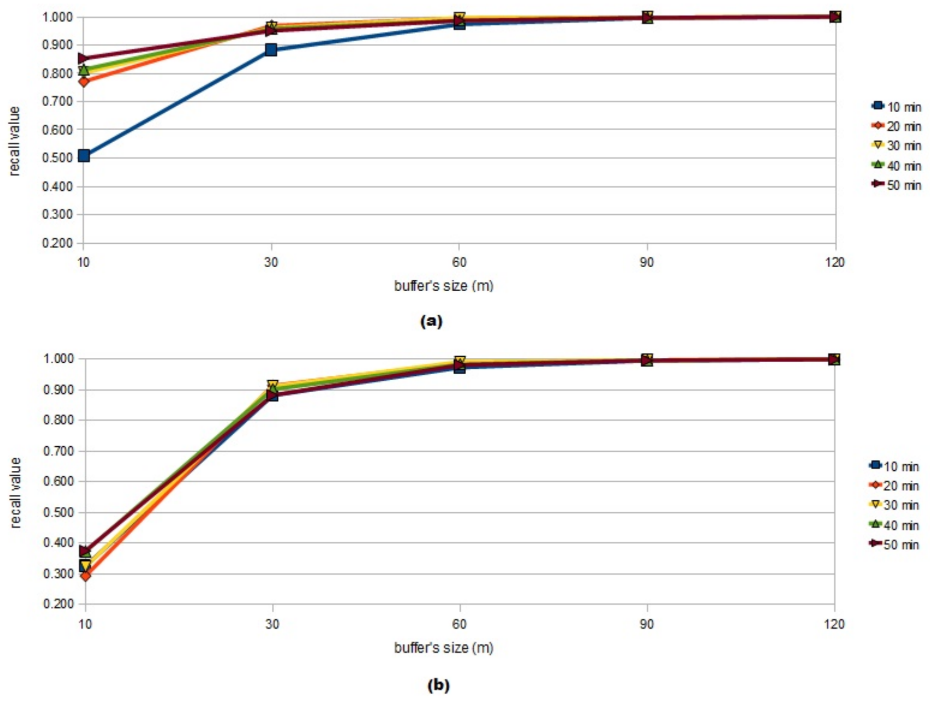


Figure 6.4: Recall

The experiments showed that recall depends on the size of the isochrone and the buffer's size in both approaches. The bigger the size of the isochrone or the bigger the buffer's size that was chosen, the higher the recall that was obtained. While precision wasn't significantly different in both approaches using the same parameters - the recall is higher in the Polygon's approach when the buffer distance is between 10 and 30 m. This is because Buffer approach misses a lot of objects near the center of the isochrone when the buffer size is small.

Figure 6.5 present the f-measure of the Polygon and Buffer approaches using different parameters. Figure 6.5 (a) presents the f-measure of the Polygon approach and Figure 6.5 (b) presents the f-measure of the Buffer approach.

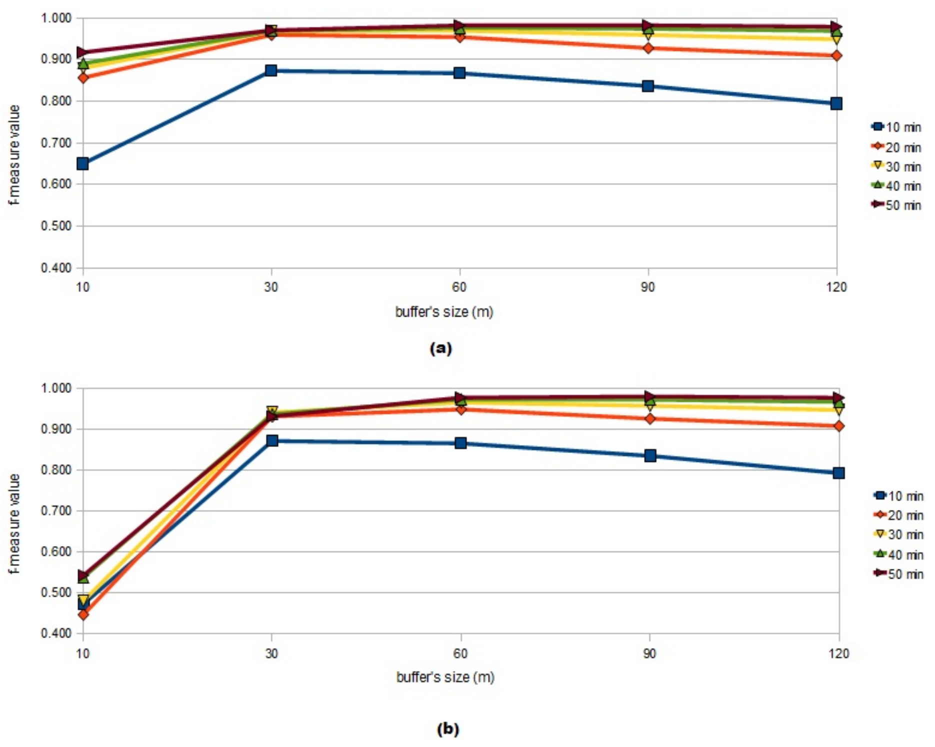


Figure 6.5: F-measure

The experiments showed that the Polygon approach works better according the the f-measure estimator when the distance buffer's size is between 10 and 30 m. In other cases there are no significant differences between the algorithms. Moreover, the highest f-measure was obtained in both case when the size of the isochrone was 50 min and the distance buffer size was between 30 and 60 m.

6.3 Experimental Analysis

In this section we present the runtime of Buffer and Polygon approaches.

To test the running time of Buffer and Polygon approaches we used different size of isochrone, because the link size in the isochrone is the most influencing factor of the running time. All other parameters were fixed: buffer size 60 m, default walking speed 30m/min, starting point location of the isochrone was taken in the center.

Figure 6.6 presents the running time performance of Buffer and Polygon approaches.

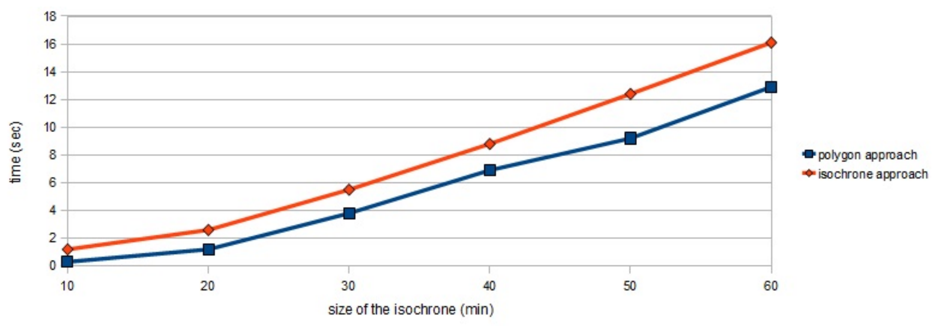


Figure 6.6: Runing time

Experiments showed that the Polygon approach is faster than the Buffer approach.

6.4 Data Inconsistency

During the experiments we found data inconsistencies. We briefly describe how to overcome those inconsistencies in this section.

While running experiments we noticed that the Polygon approach sometimes created polygons that were not complete. We examined why this was happening and found out that it was influenced by the inconsistency of the data in the map. We found two types of such inconsistency:

1. *Links* looked connected visually, but the end points on the connection point had slightly different coordinates. This situation is presented in the Figure 6.7.

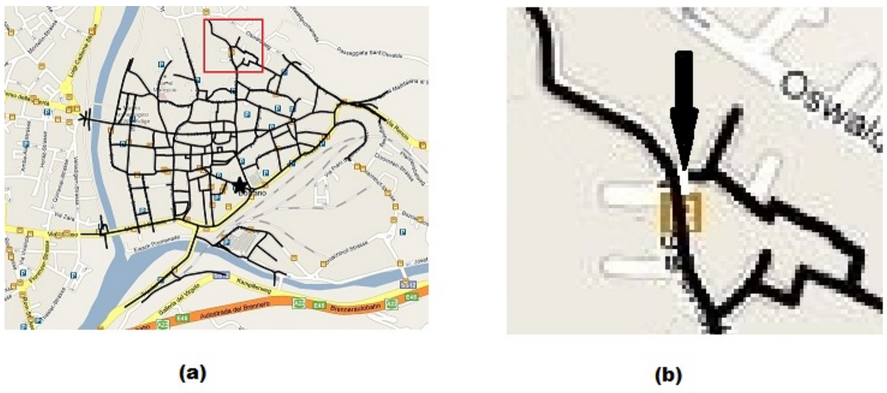


Figure 6.7: Common point inconsistency

In reality the difference on the intersection points in the before mentioned data inconsistency was few millimeters. To solve this problem we introduced maximum possible difference (ϵ) between two *links* on the intersection point. If the difference was smaller than (ϵ) we considered that *links* are intersecting. In our experiments we were using 10 centimeters as the value of (ϵ).

2. Intersected *links* were not split on the intersection point.

The Figure 6.8 presents *links* that are intersected but not split on the intersection point. We were not able to solve this inconsistency with the algorithm. We had to

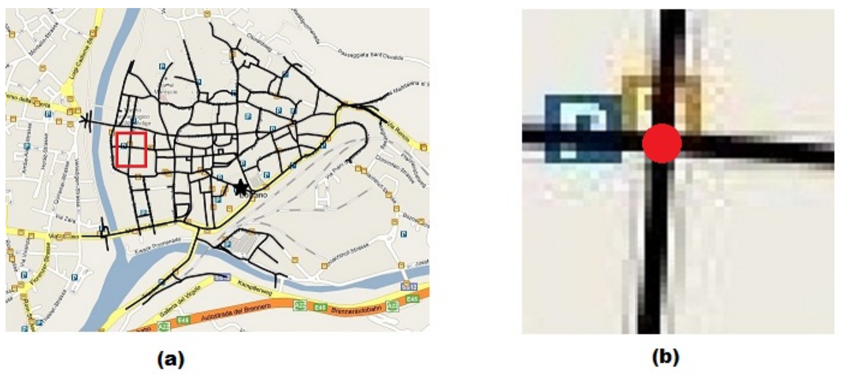


Figure 6.8: Intersection inconsistency

change such data manually in the database by splitting intersecting *links* and entering split *links* in the database.

Those inconsistencies are influencing only the Polygon approach, because this approach deals with the construction of a bounding polygon and the detailed knowledge of links is needed.

6.5 Summary

In this section we present the summary of our experimental results.

The main scope of this work was to present algorithms that find objects within an isochrone. We presented two approaches: the Buffer approach and the Polygon approach. We used estimators such as precision, recall and f-measure to measure the goodness of each approach. By running experiments we showed that Polygon approach gives better precision, recall and f-measure values comparing to the Buffer's approach if the size of the distance buffer is smaller then 60 meters. If distance buffer is larger then 60 meters, both approaches produce the same results. Experiments showed that the running time of the Polygon approach is faster than the running time of Buffer's approach. While running experiments e noticed that the the Polygon approach sometimes created polygons that were not complete. We examined why this was happening and found out that it was influenced by the inconsistency of the data in the map. However, the Buffer approach is not influenced by data inconsistency.

Table 6.3: Summary of the Polygon and the Buffer approach

	Polygon approach	Buffer approach
Precision	higher or the same	worse or the same
Recall	higher or the same	worse or the same
F-measure	higher or the same	worse or the same
Running time	faster	slower
Sensitivity to data inconsistencies	sensitive	is not influenced

Table 6.3 presents the comparison between the Polygon and the Buffer approach.

Chapter 7

Conclusion and Further Studies

In this paper we presented two approaches to allocate objects within an isochrone: the Buffer solution and the Polygon solution. To measure the quality of each approach we used precision, recall and f-measure estimators. As a precise solution for those estimators we used the Baseline approach that is expensive in terms of time complexity. By running experiments on the real world data set we showed that the Polygon solution is at least as good as the Buffer solution according to the before mentioned estimators. Experiments showed that the Polygon approach is faster than the Buffer approach. During the experiments we found data inconsistencies. We briefly described how to overcome those inconsistencies in this section. However, those inconsistencies are influencing only thePolygon approach.

We made experiments only with a pedestrian network. As future work we should modify our algorithm to work with a multimodal network that contains buses, trains, etc. The difference betweena pedestrian network and a multimodal network is that the multimodal network can have islands. For example some places are not reachable on foot within given time, but are passed by bus, but can not be included into the isochrone because the bus doesn't have a stop there and such places are not reachable on foot from the closest bus station.

We were considering that all streets within pedestrian network are open. However, some streets can be under construction and closed. If such streets will appear in the isochrone - they will create holes. As future work we should modify the algorithm to worth with the isochrone that contains holes.

Bibliography

[1] E.W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1959. [cited at p. 9]

[2] A. Galton M. Duckham. What is the region occupied by a set of points? *Procceedings of the Fourth International Conference on Geographic Information Science*, 2006. [cited at p. 12]

[3] H. Edelsbrunner. Weighted alpha shapes. *Technical Report:UIUCDCS-R-92-1760*, 1992. [cited at p. 12]

[4] R.L. Gragham. An efficient algorithm for determining the convex hull of a planar set. *Information Processing Letters 1*, 1972. [cited at p. 11]

[5] J. Gamper M. Bohlen W. Cometti M. Innerebner. Scalable computation of isochrones in bimodal spatial networks. *Technical report: Free university of Bolzano-Bozen*, 2010. [cited at p. 9]

[6] R.A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters 2*, 1973. [cited at p. 11]

[7] A. Moreira Y. Maribel. Concave hull: a k-nearest neighbors approach for the computation of the region occupied by a set of points. *Proceedings of the International Conference on Computer Graphics Theory and Applications*, 2007. [cited at p. 12]

[8] R. Baryer E. McCreight. Organization and maintenance of large ordered indexes. *Mathematical and Information Sciences Report*, 1970. [cited at p. 10]

[9] H. Edelsbrunner E.P. Mucke. Three dimensional alpha shapes. *Workshop on Volume visualization*, 1992. [cited at p. 12]

[10] N. Beckmann H.P. Begel R. Schneider B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. *ACM SIGMOD*, 1990. [cited at p. 9]

[11] H. Edelsbrunner D.G. Kirkparick R. Seidel. On the shape of the set of points in the plane. *IEEE Transactions on Information Theory*, 1983. [cited at p. 12]

[12] D. Papadias J. Zhang N. Mamoulis Y. Tao. Query processing in spatial network databases. *VLDB*, 2003. [cited at p. 9]

List of Figures

1.1 The isochrone . . . 6

2.1 R-tree for 2D objects . . . 10
2.2 Convex hull . . . 11
2.3 Right turn . . . 11
2.4 Concave hull . . . 12
2.5 Alpha shapes . . . 13

3.1 Projected points . . . 16
3.2 The structure of an isochrone . . . 17
3.3 The structure of an isochrone's surace . . . 18

4.1 Buffer solution for finding dynamic objects . . . 20
4.2 The distance buffer . . . 21

5.1 Link . . . 24
5.2 Clockwise and counterclockwise angles . . . 24
5.3 Angles between links . . . 25
5.4 Example . . . 26
5.5 Pseudocode for entire algorithm . . . 27
5.6 Pseudo code for finding the left most link . . . 28
5.7 Pseudo code to obtain links that have common point with last inserted link . 28
5.8 Pseudo code to obtain link that has the smallest counterclockwise angle with the last inserted link . . . 29
5.9 Polygon solution for finding dynamic objects . . . 30

6.1 Relationship between recall and precision . . . 32
6.2 F-measure using different starting points . . . 33
6.3 Precision . . . 34
6.4 Recall . . . 35
6.5 F-measure . . . 36
6.6 Runing time . . . 37
6.7 Common point inconsistency . . . 37
6.8 Intersection inconsistency . . . 38

List of Tables

6.1 Street network (Network) 31
6.2 Bolzano houses (Objects) 31
6.3 Summary of the Polygon and the Buffer approach 38