Determining Objects within Isochrones in Spatial Network Databases

Sarunas Marciuska and Johann Gamper

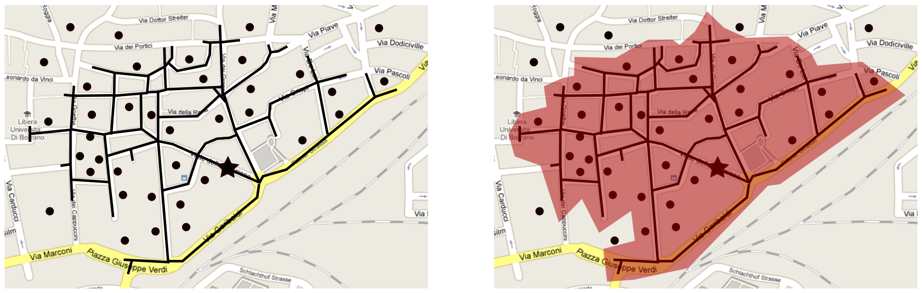
Free University of Bolzano-Bozen, Italy
{gamper,marciuska}@inf.unibz.it

Abstract. Isochrones are generally defined as the set of all space points from which a query point can be reached in a given timespan, and they are used in urban planning to conduct reachability and coverage analyzes in a city. In a spatial network representing the street network, an isochrone is represented as a subgraph of the street network. Such a network representation is not always sufficient to determine all objects within an isochrone, since objects are not only *on* the network but might be in the immediate vicinity of links (e.g., houses along a street). Thus, the spatial area covered by an isochrone needs to be considered.
In this paper we present two algorithms for determining all objects that are within an isochrone. The main idea is to first transform an isochrone network into an isochrone area, which is then intersected with the objects. The first approach constructs a spatial buffer around each edge in the isochrone network, yielding an area that might contain holes. The second approach creates a single area that is delimited by a polygon composed of the outermost edges of the isochrone network. In an empirical evaluation using real-world data we compare the two solutions with a precise yet expensive baseline algorithm. The results demonstrate the efficiency and high accuracy of our solutions.

1 Introduction

Urban planning has to deal with tasks such as to analyze the reachability of strategic objects in a city and to place these objects in optimal positions. For example, what is the best place to build a metro station or a school such that a large number of people can reach that place in comfortable times?

Isochrones, which are defined as the set of all space points from which a query point can be reached in a given timespan, are used as an instrument to perform such analyses. By joining an isochrone with the inhabitants database the number of citizens living in a certain distance from a query point can be determined. Figure 1(a) shows the 5 minutes isochrone for a single query location (star) in the city of Bozen-Bolzano. The isochrone is represented by the street segments in bold and covers all points in the street network from where the query point can be reached in 5 minutes, assuming a walking speed of 1.6 m/s and considering walking as the only mode of transportation. In general, the computation of isochrones needs to consider multiple modes of transportation, e.g., walking, bus, train, metro, etc. An isochrone is then a possibly disconnected set of space points: a large area around the query point and smaller areas around bus/metro/train stops, from where the query point can be reached by a combination of walking and using public transportation. For the sake of simplicity, in this paper we consider mainly isochrones in a pedestrian network (walking mode only). The proposed solutions can easily be extended for multiple modes of transportation, as we briefly discuss in Sec. 4.



(a) Isochrone Network (b) Isochrone Area

Fig. 1. Isochrone Representation as Network and Area.

The representation of isochrones as a subgraph of the street network is not always sufficient. A representation as an area, such as illustrated in Fig. 1(b), is often desirable for several reasons. First, the objects within an isochrone we are looking for might not lie exactly on the network but in the immediate vicinity of links, e.g., houses in a street have usually a distance of up to 50 meters or more from the street. In Fig. 1 such objects are represented as dots. With a representation as an area, it is straightforward to determine all (static or dynamic) objects within an isochrone. Second, for human users isochrones are usually visualized as an area rather than as a subgraph. Therefore, we aim at transforming an isochrone network representation into an isochrone area representation that covers the (immediate) vicinity of the isochrone network.

The computation of an isochrone area from an isochrone network is similar to the computation of a footprint for a set of points. The most common methods for this are concave hull [10] and alpha shapes [4]. Since these methods compute an area from a set of 2D points and we have a set of 2D links, they cannot directly be applied. By transforming the links into a set of points, we loose the edge information which might result in large errors, as illustrated in Fig. 2. The isochrone network in Fig. 2(a) is transformed into a set of points in Fig. 2(b). Figure 2(c) shows the area that is obtained with the alpha shapes or concave hull method; the large area indicated by the letter “A” is missing. While a parameter allows to control the computation of the area, it is generally impossible to find the right parameter to obtain the correct area, and the problem remains that with the transformation into a set of points relevant pieces of spatial information are lost.

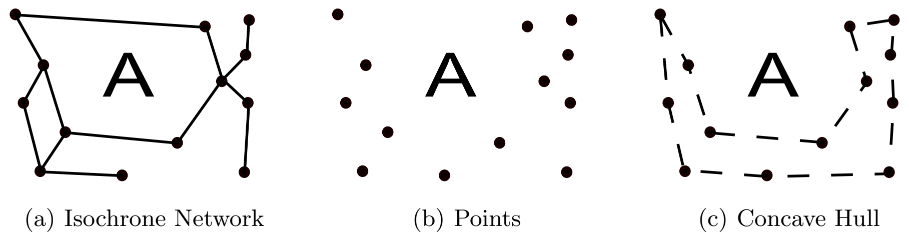


Fig. 2. Concave Hull (and Alpha Shapes) Method.

In this paper we present two different solutions to transform an isochrone network into an isochrone area. The *link-based approach* constructs a buffer of a user-specific size around each individual link of the isochrone network, yielding an area that possibly contains holes. This solution exploits existing spatial database functionalities. The *surface-based approach* computes first a polygon that covers the isochrone network and then creates a buffer around this polygon. The obtained area doesn't contain holes. To determine all objects within an isochrone, the constructed area is intersected with the relation that stores the objects. We empirically evaluate the two solutions using real-world data and determining all objects within an isochrone. We measure the quality of each solution by comparing it with a baseline solution that is precise but expensive in terms of runtime. As quality estimators we use recall, precision, and f-measure. The experiments show that the quality of the surface-based approach is higher for buffers smaller than approximately 60 meters. For a larger buffer size both approaches give similar results, since the area constructed by the link-based approach contains less holes and becomes more similar to the area constructed by the surface-based approach. The surface-based approach is faster than link-based approach, though both approaches scale almost linearly with the size of the isochrone.

The rest of the paper is organized as follows. In Section 2 we discuss related work. Sections 3 and 4 present the two different approaches for the computation of an isochrone area. Section 5 presents the experimental results, and Section 6 draws conclusions and points to future work.

2 Related Work

Isochrones are first introduced in [1, 7] as a new query type in spatial network databases, which is used by city planners as an instrument to analyze coverage and reachability queries in a city. The work in [1, 7] computes isochrones for bimodal networks, consisting of a pedestrian network and one or more bus networks. Since isochrones are represented as a subgraph of the pedestrian network, only objects that lie exactly on the network edges can be determined. In this paper we extend this work to represent isochrones as areas and to determine all objects that lie within this area.

Among the various queries in spatial network databases, range queries are closest to isochrones. A range query determines all objects (of a specific type) that are within a specific distance from a query point. The main difference is that an isochrone represents an area (containing all space points) from where a query point is reachable in a given timespan, while a range query returns all objects within a given distance. An isochrone can be intersected with any type of objects without recomputing it from scratch, and it can also be graphically visualized for a human user.

Range queries for spatial network databases are first introduced in [12], where the Range Euclidean Restriction (RER) algorithm and the Range Network Expansion (RNE) algorithm are presented. Deng et al. [2] improve over the work in [12] by performing less network distance calculations and therefore accessing less network data. Mouratidis et al. [11] present a solution for continuous nearest neighbor monitoring in road networks, where the query points and the data objects move frequently and arbitrarily in the network. All these frameworks for range queries and nearest neighbor queries assume that the objects lie exactly on the network links. Isochrones, in particular the isochrone areas as constructed in this paper can also catch objects that are in a (user-specified) immediate vicinity of the links.

The work which is closest to our work is the continuous intersection join presented in [13]. It proposes a solution to determine all objects that can be reached from a moving query point within a specified time. The main idea is to use a distance range query from the query point in order to determine the objects that can be reached from there. However, the range query uses the Euclidean distance resulting in a circular area around the query point, which is intersected with the objects. Isochrones use the network distance, and the main challenge is to construct a minimal area around the isochrone network which represents all space points within the isochrone.

The computation of an isochrone area from an isochrone network is similar to the computation of a convex or concave hull for a finite set of 2D points. Two main algorithms are known for the concave hull: Jarvis March [9] and Graham scan [8]. The main idea of the Jarvis March approach [9] is to include a point in the convex hull that has the smallest polar angle with respect to a previous point. As the initial point, the left-most point among all points is taken. The algorithm runs in $O(nh)$ time, where n is the number of points in the data set and h is a number of points on the convex hull. The Graham scan approach [8] works in three steps. First, the point with the smallest y-coordinate is chosen. Second, the remaining points are increasingly sorted according to the x-coordinate. 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, the link from the second to the last point is removed. If a left turn occurs, the last point is included into the convex hull, and the next point is taken from the sorted array. The algorithm runs in $O(n \log n)$ time for a finite set of n points.

In general, the shape of an isochrone area is closer to a concave hull than to the convex hull. Different from the convex hull, there is no unique concave hull. An algorithm for the computation of concave hulls for a set of 2D points is presented in [10]. The algorithm is based on the k-nearest neighbors. Depending on the choice of k, different concave hulls are generated. With a higher number of k, the shape becomes smoother. With $k = n$, the concave hull coincides with the convex hull. It is difficult to determine the right value of k to get a good shape for the isochrone area.

A similar problem of finding footprints for a set of $2D$ points is discussed in [6] and [3–5]. These approaches are based on so-called alpha shapes. The main idea of the alpha shape algorithm is to draw circles with a radius of $1/alpha$ such that they touch at least two points and none of the other points is inside those circles. All points that touch a circle are selected and connected. If the radius is big enough, the result is the convex hull. If the radius is too small, the result is the set of all points without any connections.

Since isochrone networks are represented as 2D links, the above approaches for the computation of convex/concave hulls and alpha shapes cannot be directly applied to compute isochrone areas. If we first transform the links into points, we loose important spatial information, which might lead to significant errors in the shape of the isochrone area.

3 Link-Based Approach

In this section we describe the link-based approach, which draws a buffer around each individual link of the isochrone network and returns the union of these buffers as the isochrone area.

An isochrone network is represented as a graph $G = (V, E, \gamma)$, where V is a set of vertices (or nodes), $E \subseteq V \times V$ is a set of links (or edges), and γ is a function that assigns a geometry to each edge. The geometry is a polyline, $\gamma((u, v)) = \{p_1, \dots, p_n\}$, where $p_1, \dots, p_n$ are space points that are connected by lines.

To create the buffers we use Oracle's built-in function SDO_BUFFER(A,d), which creates a buffer of size d around the spatial object A. Unfortunately, by applying this function for a link, the border of the buffer is not going through the endpoints of the link, as illustrated in Figure 3(a). The isochrone consists of the nodes $V = \{q, a, b\}$ and the links $E = \{(q, a), (q, b)\}$, where a and b represent the outermost points from where the query point q is reachable in the given timespan t_{max}. Drawing a buffer of distance d around each of the two links introduces an error near the nodes a and b.

To remedy from this problem, we reduce the maximal timespan, t_{max}, of the isochrone by an amount that corresponds to the buffer size d. That is, we determine $t'_{max} = t_{max} - \frac{d}{s}$ as the new timespan for the computation of the isochrone network; s is the walking speed used for the computation of the isochrone. Using t'_{max} results in a smaller isochrone network. By constructing a buffer for each link in the reduced isochrone network, the buffers cross exactly the outermost points a and b of the original network (see Fig. 3(b)).

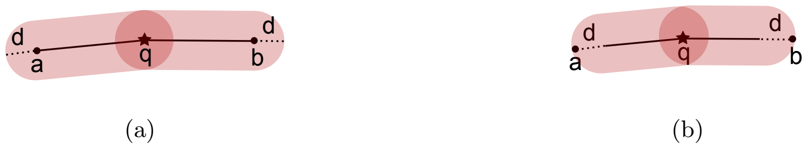


Fig. 3. Decreasing the Timespan for the Computation of the Isochrone Network.

Figure 4 shows the algorithm lISO for the computation of an isochrone area using the link-based approach. The algorithm has two input parameters: an isochrone network I; a buffer size d. The algorithm iterates over all links in the isochrone network I and constructs a buffer of size d around each link. These buffers are collected in B and are returned as area representation of the isochrone I, covering all space point on the network and in the immediate vicinity from where q is reachable in the given timespan.

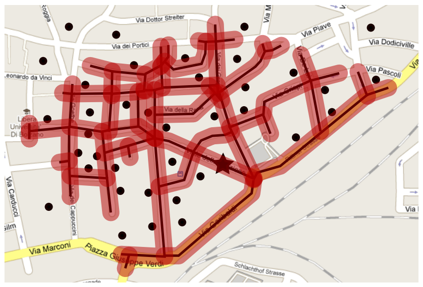
```
**Algorithm:** lISO($I, d$)
**Input**: Isochrone $I = (V, E, \gamma)$; distance $d$;
**Output**: Isochrone area $B$;
$B \leftarrow \emptyset$ ;
**foreach** link $l \in E$ **do**
    $B \leftarrow B \cup \{SDO\_BUFFER(l, d)\}$;
**end**
**return** $B$;
```

Fig. 4: Link-based Approach for the Computation of an Isochrone Area.

Figure 5(a) shows the isochrone area computed with the link-based approach, using a buffer size of 30 meters. Depending on the size of the buffer, the isochrone contains more or less holes. The isochrone in Fig. 5(b) uses a buffer size of 50 meters, resulting in less holes.

4 Surface-Based Approach

The surface-based approach computes first the minimum bounding polygon of the isochrone network, termed its surface, and draws then a buffer around the surface. The *surface* of an isochrone $I = (V, E, \gamma)$ is defined as the minimal set


(a) Buffer Size = 30 m

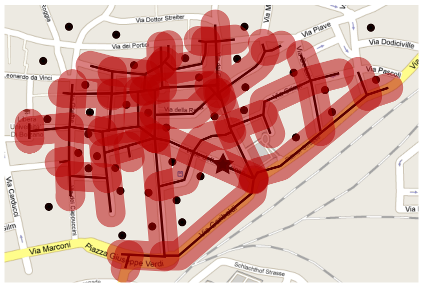
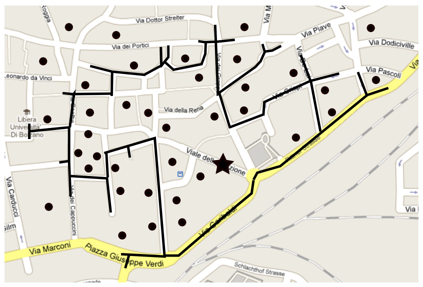

(b) Buffer Size = 50 m

Fig. 5. Isochrone Area with the Link-Based Approach.

of links $S \subseteq E$ that form a polygon and cover all other links in E. Figure 6(a) shows the surface of the isochrone in our running example, which covers all street links of the isochrone network.

Next, we construct a buffer of a user-specified size d around the surface polygon in order to include also space points in the outer vicinity of the surface polygon. Figure 6(b) shows the isochrone area that is constructed with the surface-based approach. Obviously, the isochrone area does not contain any holes (different from the link-based approach).


(a) Surface of the Isochrone

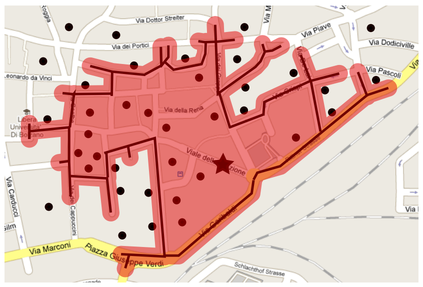

(b) Isochrone Area

Fig. 6. Isochrone Area with Surface-Based Approach.

The algorithm to compute the surface of an isochrone is a generalization of Jarvis' algorithm [9] for the computation of the convex hull. The main idea is to find first the link with the left-most endpoint (i.e., smallest x-coordinate) and the smallest counter-clockwise angle with the y-axis. (Any other link which lies on the surface polygon could be used as the initial link as well). Starting from the initial link, the algorithm iteratively adds an adjacent link to the surface, which has the smallest counter-clockwise angle with the link that has been added previously. The algorithm terminates when it returns to the initial link.

```
**Algorithm:** sISO$(I, d)$
**Input**: Isochrone $I = (V, E, \gamma)$; distance $d$;
**Output**: Isochrone area $B$
$u_0 \leftarrow \underset{v \in V}{\text{argmin}}\{v.x\}$;
$\alpha \leftarrow 360°$;
foreach link $(u, v) \in E$ such that $u = u_0$ do
    $\alpha' \leftarrow angle(u-(0,1), u, v)$;
    if $\alpha' < \alpha$ then
        $\alpha \leftarrow \alpha'$;
        $(u_0, v_0) \leftarrow (u, v)$;
    end
end
$S \leftarrow \{(u_0, v_0)\}$;
$(u_p, v_p) \leftarrow (u_0, v_0)$;
repeat
    $\alpha \leftarrow 360°$;
    foreach link $(u, v) \in E$ such that $u = v_p$ do
        $\alpha' \leftarrow angle(u_p, u, v)$;
        if $\alpha' < \alpha$ then
            $\alpha \leftarrow \alpha'$;
            $(u', v') \leftarrow (u, v)$;
        end
    end
    $S \leftarrow S \cup \{(u', v')\}$;
    $(u_p, v_p) \leftarrow (u', v')$;
until $(u', v') \neq (u_0, v_0)$;
$B \leftarrow SDO\_BUFFER(S, d)$;
return $B$
```

Fig. 7: Algorithm sISO.

Figure 7 shows the algorithm, which has two input parameters: an isochrone network I and the size d of the buffer. The algorithm returns the area representation of the isochrone I, using the surface-based approach.

The algorithm determines first the left-most node, u_0, of the isochrone network. Then, the link through u_0 which has the smallest counter-clockwise angle with the y-axis is determined. Figure 8(a) illustrates this step. The left-most node is n_1, which has two links (n_1, n_3) and (n_1, n_2). The link (n_1, n_3) has the smaller angle, α_1, and is chosen as the initial link and added to the surface S. Next, the algorithm enters a loop, in which the surface is incrementally extended with a new link on each iteration until the initial link is encountered again. (u_p, v_p) represents the link that has been added in the previous iteration (or the initial link on the first iteration). On each iteration all links that are connected to (u_p, v_p), i.e., have v_p as source node, are considered. The link with the smallest counter-clockwise angle with (u_p, v_p) is on the surface and is added to S. This step is illustrated in Fig. 8(b). The links (n_3, n_4) and (n_3, n_5) are considered as possible extensions of the initial link (n_1, n_3). Since α_3 is smaller than α_4, the link (n_3, n_4) is added to the surface S. The loop terminates when the initial link (u_0, v_0) is encountered again. Figure 8(c) shows the completed surface. As a last step, the algorithm creates a buffer of size d around the surface of the isochrone, which is returned as a result.

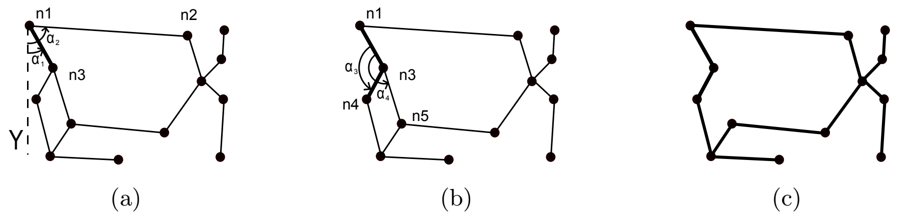


Fig. 8. Step-Wise Computation of the Surface of an Isochrone.

Finding the left-most link at the beginning and the next link in each iteration takes $O(n)$ time, where n is the number of nodes in the isochrone. In the worst case, the extension step iterates over all nodes (if all nodes are on the surface), yielding an overall complexity of $O(n^2)$.

When multiple modes of transportation are considered, isochrones get typically disconnected. For instance, Fig. 9(a) shows an isochrone when walking in combination with buses are considered. There is a large island (area) around the query point and small islands around the reachable bus stops. While lISO correctly handles disconnected isochrones, the surface-based algorithm sISO works only for isochrones that form a connected graph. To adapt the algorithm for disconnected isochrones (as produced when multiple transportation modes are considered), a pre-processing step is required to determine the connected components (i.e., maximal connected subgraphs) of the isochrone, which can be done in linear time. Then for each connected component the algorithm sISO is called.

5 Experimental Evaluation

In this section we present the results of an experimental evaluation of the two algorithms using real-world data.

5.1 Setup and Data

The two algorithms for the computation of an isochrone area and the intersection of the isochrone area with objects (e.g., houses) were implemented in Java on top of the Oracle Spatial DBMS. The algorithms use built-in functionalities of Oracle Spatial to construct buffers and to compute the intersection between areas and objects. To compute the initial isochrone network, which is passed as input to sISO and lISO, we use the algorithm in [7]. The spatial data, including

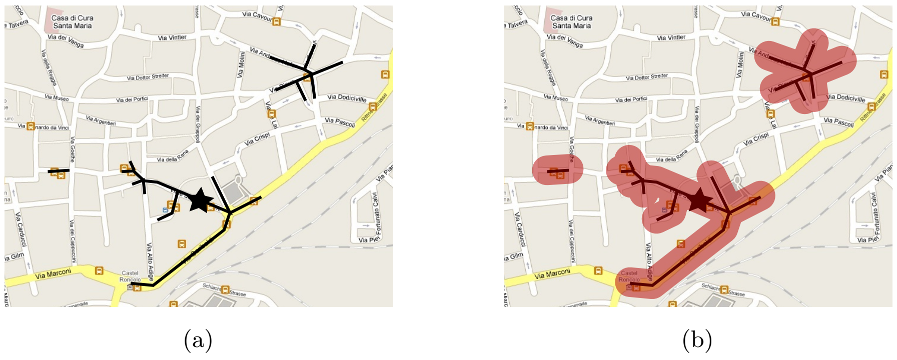


Fig. 9. Isochrone for Multiple Modes of Transportation.

the isochrones and the objects, are stored in the database. All experiments were run on a computer with a 2GHz CPU and 1.5 GB RAM.

For the experiments we used the street network of the city of Bolzano-Bozen, which consists of approximately 3500 links (streets segments). As objects within an isochrone we used the houses in Bolzano-Bozen (approximately 12300 houses), which are stored in a separate table.

To measure the quality of lISO and sISO we implemented a baseline approach as a reference solution, which essentially works as follows. Each house h is projected perpendicularly to the closest edge (u, v) in the pedestrian network. More specifically, it is projected to a segment (p_i, p_{i+1}) of the polyline $\gamma((u, v)) = \{p_1, \ldots, p_n\}$ that represents the edge's geometry. Assume that a house is mapped to point p on edge (u, v). Then a house is considered to be within an isochrone if its Euclidean distance to p plus the network distance from p to query point q is smaller than the maximal timespan of the isochrone. While the basline approach is slow, since it needs to determine for each house the distance to each individual segment of all edges, we use it as a reference solution to measure precision, recall, and f-measure of the surface-based and link-based solutions.

5.2 Precision, Recall, and F-measure

Varying the Location of the Query Point. In the first experiment we use a fixed timespan of 15 min, a walking speed of 1.6 m/s, and a buffer size of 30 m, and we vary the location of the query point between locations in the center and the border of the city as well as between dense, average, and sparse areas. To measure the quality of the two approaches the f-measure is used. The result of this experiment is presented in Fig. 10 and shows that the location of the query point (and hence different densities of houses) has almost no impact on the quality. Therefore, in the remaining experiments we use a fixed query point in the city center.

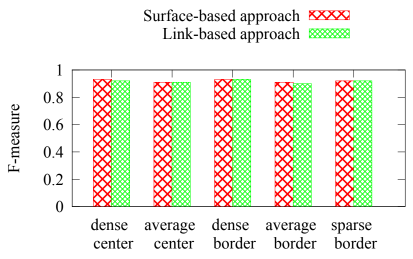
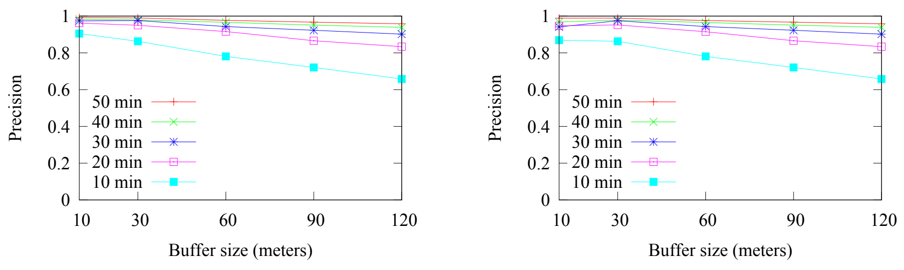


Fig. 10. F-measure for Different Query Points.

Varying Buffer Size and Maximum Timespan. In the second experiment we use a fixed query point in the city center and a walking speed of 1.6 m/s, and we vary the buffer size between 10 and 120 m and the maximal timespan for the isochrone between 10 and 50 min.

Figure 11 presents the precision of sISO and lISO. For both solutions the precision depends on the size of the isochrone. The bigger the maximal timespan, the higher is the precision. Vice versa, if the size of the buffer is too large, the precision is decreasing, since many false positives are included. There is no substantial difference in precision between sISO and lISO. The best precision is obtained when the buffer size is between 10 and 30 m.



(a) sISO (b) lISO

Fig. 11. Precision

Figure 12 shows the recall, which for both solutions increases with the size of the isochrone and with the size of the buffer, though the size of the buffer has less impact in lISO. While the precision is almost identical for both solutions, the surface-based approach has a significantly higher recall for small buffers up to a size of 30 m. When the buffer size is small, the link-based approach misses many objects that are located in the holes.

Figure 13 shows the f-measure. For a small buffer size up to approximately 60 m the surface-based approach is superior. For large buffers the difference be-

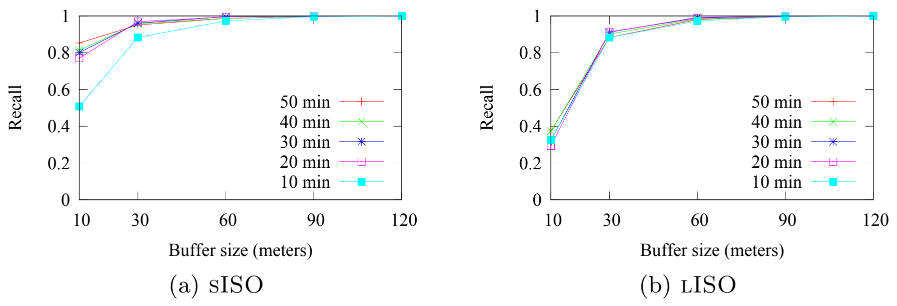


Fig. 12. Recall.

tween the two solutions disappears, since the isochrone area produced by the link-based approach becomes more and more similar to the isochrone area produced by the surface-approach. The highest f-measure for both solutions is obtained with a buffer size of approximately 60 m.

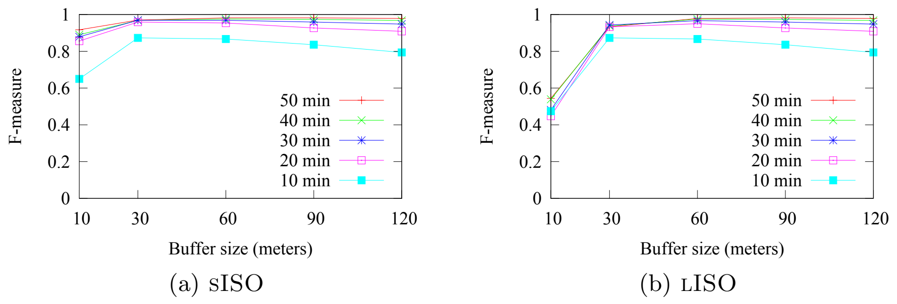


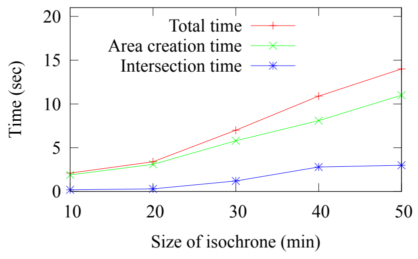
Fig. 13. F-measure.

5.3 Runtime

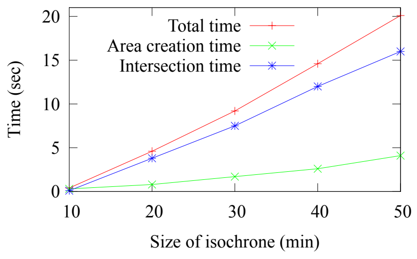
In the last experiment we analyze the efficiency of the proposed solutions, by varying the size of isochrone, since the number of links in an isochrone is the most influencing factor for the running time. All other parameters are fixed: buffer size 60 m, walking speed 1.6 m/s, and query point in the city center.

Figure 14 shows the results of the runtime experiment, distinguishing between the time for computing the isochrone area, intersecting the area with the objects, and the total runtime. The creation of the link-based area is more efficient. However, the intersection of the isochrone area with the objects is faster in the surface-based approach, since only one large area needs to be intersected.

Overall, both solutions scale almost linearly with the size of the isochrone, and the surface-based approach is faster than the link-based approach in terms of total runtime.



(a) sISO



(b) lISO

Fig. 14. Runtime.

6 Conclusion and Future Work

In this paper we present two different solutions, termed link-based approach and surface-based approach, to determine all objects that lie within an isochrone. Both solutions first transform an isochrone network into an isochrone area and then perform an intersection with the relation that stores the objects. The link-based approach constructs a buffer around each individual link of the isochrone network. The surface-based approach computes first a polygon that covers the isochrone network and then creates a buffer around this polygon. We run experiments with real-world data to measure the quality and efficiency of the two solutions. Both approaches achieve a high quality (compared to a precise yet slow reference solution). The surface-based approach is superior for small buffers, and it is more efficient than the link-based approach.

Future work is possible in various directions. More specifically, we will conduct more extensive experiments both to study the quality of the two solutions for different types of objects and to analyze the scalability for very large isochrones.

References

1. Veronika Bauer, Johann Gamper, Roberto Loperfido, Sylvia Profanter, Stefan Putzer, and Igor Timko. Computing isochrones in multi-modal, schedule-based transport networks (demo paper). In *ACMGIS-2008)*, pages 1–2, Irvine, CA, USA, November 5–7 2008.
2. Ke Deng, Xiaofang Zhou, Heng Tao Shen, Shazia W. Sadiq, and Xue Li. Instance optimal query processing in spatial networks. *VLDB J.*, 18(3):675–693, 2009.

3. H. Edelsbrunner. Weighted alpha shapes. *Technical Report:UIUCDCS-R-92-1760*, 1992.
4. Herbert Edelsbrunner, David G. Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–558, 1983.
5. Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. In *VVS*, pages 75–82, 1992.
6. Antony Galton and Matt Duckham. What is the region occupied by a set of points? In *GIScience*, pages 81–98, 2006.
7. J. Gamper, M. Böhlen, W. Cometti, and M. Innerebner. Scalable computation of isochrones in bimodal spatial networks. Technical report, Free University of Bolzano-Bozen, 2010.
8. Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1(4):132–133, 1972.
9. R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Inf. Process. Lett.*, 2(1):18–21, 1973.
10. Adriano J. C. Moreira and Maribel Yasmina Santos. Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points. In *GRAPP (GM/R)*, pages 61–68, 2007.
11. Kyriakos Mouratidis, Man Lung Yiu, Dimitris Papadias, and Nikos Mamoulis. Continuous nearest neighbor monitoring in road networks. In *VLDB*, pages 43–54, 2006.
12. Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao. Query processing in spatial network databases. In *VLDB*, pages 802–813, 2003.
13. Rui Zhang, Dan Lin, Kotagiri Ramamohanarao, and Elisa Bertino. Continuous intersection joins over moving objects. In *ICDE*, pages 863–872, 2008.