Proximity Graphs for Crowd Movement Sensors

Cristian Chilipirea, Andreea-Cristina Petre, Ciprian Dobre
Faculty of Automatic Control and Computers
University Politehnica of Bucharest
Bucharest, Romania
Email: {cristian.chilipirea; ciprian.dobre}@cs.pub.ro, andreea.petre@cti.pub.ro

Maarten van Steen
CTIT
University of Twente
Enschede, Netherlands
Email: m.r.vansteen@utwente.nl

***Abstract*—Sensors are now common, they span over different applications, different purposes and some over large geospatial areas. Most data produced by these sensors needs to be linked to the physical location of the sensor itself. By using the location of a sensor we can construct (mathematically) proximity graphs that have the sensors as nodes. These graphs have a wide variety of applications including visualization, packet routing, and spatial data analysis.**

We consider a sensor network that measures detections of WiFi packets transmitted by devices, such as smartphones. One important feature of sensors is given by the range in which they can gather data. Algorithms that build proximity graphs do not take this radius into account.

We present an approach to building proximity graph that takes sensor position and radius as input. Our goal is to construct a graph that contains edges between pairs of sensors that are correlated to crowd movements, reflecting paths that individuals are likely to take. Because we are considering crowd movement, it gives us the unique opportunity to construct graphs that show the connections between sensors using consecutive detections of the same device. We show that our approach is better than ones that are based on the positioning of sensors only.

Keywords: **crowd movement, sensors, geolocation, graphs**

I. Introduction

Increasingly more applications require the use of multiple sensors that spread over a geospatial area. Most sensors have a geolocation characteristic: data generated by these sensors has meaning only when the location of the sensor is considered. When we consider multiple sensors that measure the same characteristic the location of these sensors in relation with each other is also important.

The physical location characteristic is most apparent in applications that take sensor data and construct map-based visualization of an area. An application area of increasing interest is crowd monitoring using sensors that process WiFi packets [1]. This is the application area we focus our analysis on in this paper. Our sensors are able to detect and parse WiFi packets and extract MAC addresses that can then be used to track a WiFi-enabled device.

When one considers multiple sensors that gather the same or correlated data, it is important to consider the spatial locality of the sensors themselves and in relation to each other. Regardless of sensor type, measurements from closely positioned sensors have stronger correlations and more similarity than distant sensors. Proximity graphs achieve such a distinction by mapping the relation between sensors from a spatial perspective.

We present multiple methods for constructing proximity graphs for distributed sensor networks and analyze these graphs for two distinct deployments of crowd-monitoring sensors based on WiFi detections. These algorithms take as input the location of the sensors and generate a proximity graph. Proximity graphs generally do not consider the range that a sensor is expected to "cover." We present a new proximity-graph generator algorithm that takes as input the location of sensors as well as the range at which sensors function and show how the resulting graph compares with other algorithms.

II. Related Work

When multiple sensors are measuring the same feature that is expected to vary linearly between two neighboring sensors it is possible to obtain all values between them by interpolating the values they produce. A concrete example is computing noise levels between two sensors by using just the two data points from the sensors. In the case of crowd monitoring, interpolation can be applied to crowd densities, if we expect them to vary linearly between the sensors. A solution for this problem uses a proximity graph for the sensor to compute the interpolated values between multiple sensors [2].

A different use case is given in [3] where the proximity graph is used to make a sensor network more energy efficient by improving the communication between multiple sensors while still maintaining redundancy unlike the case of a minimum spanning tree.

Localization is a problem that is still being researched. Even though at a global scale the GPS, [4] solves this problem with a reasonable error margin, at finer scales, for instance inside a building the problem requires a level of accuracy that is difficult to achieve (apart from the fact that using GPS indoors is difficult to oftentimes impossible). Proximity graphs are used in these cases in different ways. One way is to better estimate the location of the sensors themselves [5]. Another is to predict the performanaace of positioning systems through fingerprinting [6]. Yet another is in tracking a target in a mobile sensor network [7].

In contrast to the situation where we take a known sensor network and try to build a proximity graph that best matches real-life data, [8] and [9] use data obtained from mobile devices about static WiFi access points. Then, like us, they build a proximity graph of these access points. They do not compare multiple proximity-graph generation methods, as they

do not have the location of the access points, but instead generate a graph by putting an edge between any two access points that are seen by a mobile device at the same time. This method is similar to the way we obtain our proximity graphs from real-life data. It differs in the way that it does not construct edges between access points with nonoverlapping ranges even though possible pathways or correlations between them may exist.

III. Proximity Graphs

Consider a WiFi-packet sensor system. The sensors are the nodes of a graph $G = (V, E)$ with vertices $V = \{v_i | v_i \textit{ is a sensor}\}$ and edges $E = \{e_{ij} = (v_i, v_j) \in V^2 | i < j\}$. G represents the graph with edges between all nodes (a ***Full Mesh***). We note that V is the same for all the graphs we present here, the sensor list does not change, and they also have a location and a radius that do not change.

The ***Relative Neighborhood Graph*** (RNG) [10] is defined as $G_{rng} = (V, E_{rng})$ with $E_{rng} = \{e_{ij} = (v_i, v_j) \in V^2 | i < j; \forall v_k \in V : d_{ij} < max(d_{ik}, d_{jk})\}$ where d_{ij} is the Euclidean distance between v_i and v_j. In other words, two vertices are joined if there is not another vertex that is closer to both than they are to each other.

An RNG is also a ***Gabriel Graph*** (GG) [11] which is defined as $G_{gg} = (V, E_{gg})$ with $E_{gg} = \{e_{ij} = (v_i, v_j) \in V^2 | i < j; \forall v_k \in V : d_{ij}^2 < d_{ik}^2 + d_{jk}^2\}$ where d_{ij} is again the Euclidean distance between v_i and v_j. Two vertices are joined if the smallest circle having the two on its circumference does not contain another vertex.

Finally, Gabriel graphs form a subset of graphs constructed through a ***Delaunay Triangulation*** (DT) [12] in which v_i, v_j, v_k are connected as a triangle if the circle circumscribing them does not contain any other vertex.

The ***Sphere of Influence Graph*** (SIG) [13] has $G_{sig} = (V, E_{sig})$ with $E_{sig} = \{e_{ij} = (v_i, v_j) \in V^2 | i < j; \forall v_k, v_l \in V : d_{ij} < d_{ik} + d_{il}\}$ where two vertices are joined if their Euclidean distance is less than the sum of the distances to each of their respective nearest neighbor.

We define ***Inferred Graphs*** (IG) as a set of graphs obtained from real-life data, equal in size with the number of edges of a full mesh. $G_{ig,k} = (V, E_{ig,k})$ with $E_{ig,k} =$ first k elements of $\{e_{ij} = (v_i, v_j) \in V^2 | i < j\}$ ordered by f_{ij} where f_{ij} is the number of unique devices that have consecutive detections at the sensors at the ends of the edge e_{ij}. Here the smallest graph would include only the most popular edge (the one were most devices have consecutive detections) and the following graphs would add edges with an increasingly lower popularity until the full mesh is reached. We have yet to find an optimal number of edges for these graphs and we believe it to be application dependent.

As stated previously, our goal is to create a graph with a minimal number of edges that contains all the edges between pairs of sensors that are correlated, or, in the case of crowd monitoring that contain paths people are likely to take. Given this, an edge is part of our graph if and only if the area between v_A and v_B is not covered by other nodes, as can be seen in Fig. 1. We also join v_i and v_j when their respective sensors are in each other's range. In Fig. 1 we see how our algorithm works with nodes that have different radiuses. If the straight line between A and B is partly covered by the range of another node, then there is a very low probability that when a device moves from A to B we obtain a detection at v_A followed by a detection at v_B without there being a detection at the node between the two.

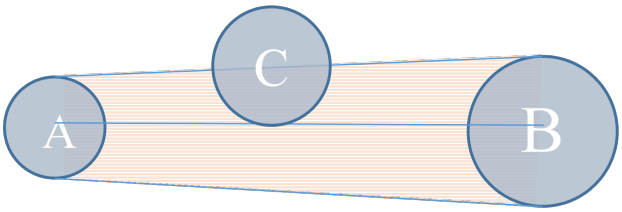


Fig. 1: Test if edge exists between A and B.

An algorithm that detects edges e_{ij} for our proximity graph takes a set of possible lines from C_i (the circle representing the radius of v_i) to C_j and counts how many intersect the radius circle C_k of any other sensor k. The proximity graph algorithm can be compared to the Ray-Casting algorithm. The number of lines that need not intersect any other sensor was empirically determined to be 80% of the total number of lines drawn between the two sensors.

We define the proximity graph obtained in our solution as $G_H = (V_H, E_H)$. Here we have the vertices $V_H = \{v_i = (c_i, r_i) | v_i \textit{ is a sensor}\}$ with c_i as the location, a GPS coordinate of the sensor and r_i the range from which our sensor can receive packets. We consider the signal radius of a sensor to be an ideal circle. In reality the shape of this area varies because of signal reflection on existing buildings, other obstacles in the path of the signal and even atmospheric differences. Our algorithm permits the use of radiuses that can be varied from sensor to sensor. In the absence of accurate information we use a radius of 100m, consistent with usual WiFi ranges. The set of edges $E_H = \{e_{ij} = (v_i, v_j) \in V_H^2 | i < j; \{v_k..v_l\} \in V_H, i, j \notin \{k..l\}, C_t \textit{ circle of radius } r_t \textit{ around center } c_t, \{C_k..C_l\} \textit{ interupt most paths from } v_i \textit{ to } v_j\}$.

IV. Correlation with Inferred Graphs

To compare the different types of graphs we mentioned in the previous section we needed a real-life application. We used deployments of WiFi-packet sensors used to measure the density and movements of crowds. We deployed a system in the city of Arnhem, The Netherlands. The system gathered data from five sensors placed at reasonable distances (between 70m and 300m) from each other.

We also used a distinct data set that uses the same type of sensors like the ones we deployed. This second data set consisted of data from 27 sensors placed in the city of Assen, also The Netherlands. In this case two sensors were placed further away than the rest, the latter which were grouped near the center of the city.

(a) RNG (b) GG (c) DT

(d) SIG (e) Our Solution (f) Full Mesh

Fig. 2: All Proximity Graphs for the Arnhem Sensors

Both data sets were gathered in two similar cities during popular events that brought more than 100,000 visitors to each city. The data sets vary in size, the Arnhem set spans over one day, while the Assen one spans over 3 days.

Fig. 2 shows the results of running all six graph-generation algorithms described in the previous section over the Arnhem data set. The numbers represent the ID of the sensor as it was set by our system. It is easy to see the relation between the common proximity graph algorithms, that of $RNG \supseteq GG \supseteq DTG$. We argue that our solution is the most useful one. For instance, the edge 2-3 is missing in the Delaunay Triangulation. Considering a radius of our sensors of 100m (the distance between 3 and 5 is of about 80m), one can easily identify paths that an individual can take from sensor 2 to sensor 3 without being detected by sensor 5. This is made even more obvious in the Sphere of Influence Graph between the sensors 2 and 10. The only difference between our solution and a full mesh for this scenario is given by the 4-10 edge. We argue that there are no practical paths to move from sensor 4 to sensor 10 without being detected by sensors 2, 3 or 5. This is confirmed by Table I, where the edge 4-10 is the one with the lowest value. There we can see the number of unique devices moving between all the hotspots.

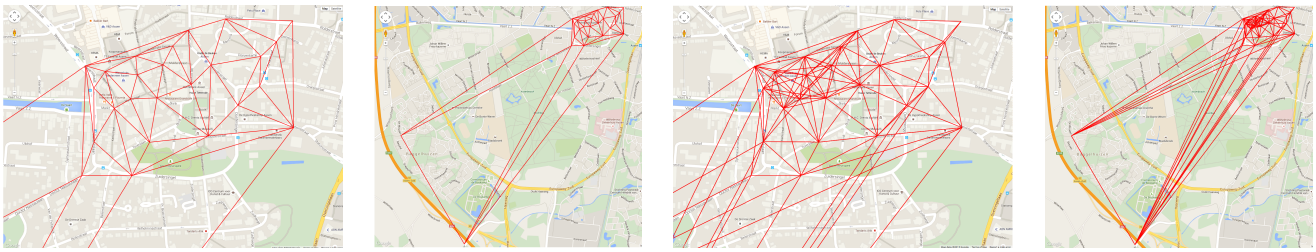
We define a movement to be a set of two detections of the same device at two different sensors, at two distinct moments in time, with no detection of the same device between these two moments (in other words, they are successive detections). The other generated proximity graphs have an even smaller number of edges, minimizing the possible paths that individuals can take to move from one hotspot to the other. This is not favorable.

Fig. 3 we see the results of running the Delaunay Triangulation algorithm and our solution over the 27 sensors placed in Assen. Because of the large number of sensors the differences and the required edges are not as clear as they were in the Arnhem case, where the full mesh had only 10 edges. We argue here that even though the graph resulted from the Delaunay Triangulation looks less crowded and all nodes seem to be connected to their neighbors, these connections are not sufficient. Because sensors are placed extremely close together, causing a lot of overlap, the DT removes a lot of edges that contain paths. This is mostly visible in the zoomed out images, the two sensors that are far away have only few edges, in the DT version, to the cluster of nodes in the center of the city. In our version we can see extra edges that go towards nodes that could have detections because there are road paths that could lead directly to these nodes. We have not added here the graphs for the other algorithms in order to preserve space. DT is also the algorithm that compares best to our solution, all other have similar or worse results.

Inferred graphs show the most likely edges that should be considered when building a proximity graph. To further

TABLE I: Arnhem Unique detections for sensor pairs

Vi	2	2	3	4	2	3	3	2	5	4
Vj	3	4	5	5	5	4	10	10	10	10
# Unique Devices	6040	3009	2331	2220	1856	1604	925	237	92	90



DT zoomed in DT zoomed out Ours zoomed in Ours zoomed out

Fig. 3: Proximity graphs for the Assen sensors

validate our solution we compare the graphs that we generated with all possible inferred graphs from the Assen data set. To make the comparison we take all combinations of generated and inferred graphs and build a confusion matrix [14] for them. We consider the inferred graphs to be the ground truth and the generated graphs as the test. After we built the appropriate matrix for each case we calculate the accuracy with which our generated graph matches the inferred one. We consider edges that appear in the graph to be true/positive, while all other possible edges are false/negatives. The accuracy is calculated as $\frac{\sum \text{true positives} + \sum \text{true negatives}}{\sum \text{total population}}$.

The results can be observed in Fig. 4. We note here that all the generating algorithms construct graphs with a fixed number of edges, while the Inferred method can generate graphs with any number of edges between 1 and the number of edges present in the full mesh.

The algorithm we use to make inferred graphs generates a large number of these graphs, equal in size with the number of possible edges, for any given data set. As mentioned before all edges from the full mesh graph are ordered decreasingly with the number of pairs of consecutive detections identified between the nodes that make the edge. A question that still remains unanswered is: how many edges should the inferred graph have and is there such a graph that is optimal? We assume that this number is application-domain dependent. This is why in Fig. 4 we compare with all the possible inferred graphs.

There is also a noticeable effect on the accuracy: generated graphs with small number of edges match the inferred graph with small number of edges more accurately than the ones with large number of edges and the same is true in reverse. This is why at left part of Fig. 4 our method is the least accurate, because it also generates the most edges. We have however identified possible candidates for the optimal inferred graph. These are represented in Fig. 4 with the black vertical lines and are, in ascending order: 100 edges - $\sharp movements/\sharp devices > 1.1$; 112 edges - more than 1000 unique devices; 117 edges - more than 1000 movements; 130 edges - average time

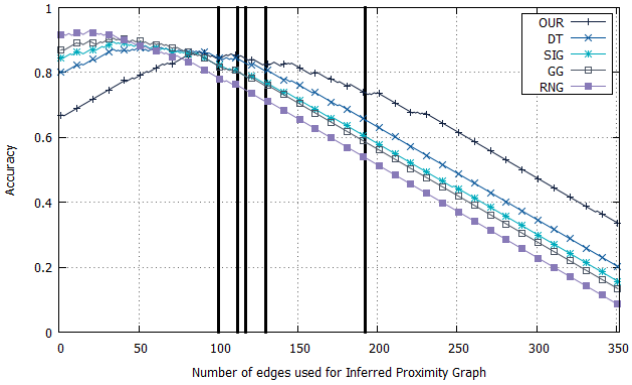


Fig. 4: Comparison of proximity graphs and IG

difference $<$ 5400s (1 and a half hours); 192 edges - that have: $\sharp movements/\sharp devices > 1$ and normalized average time difference between departure from first sensor and arrival at the second $<$ 0.5 and average time difference between departure from first sensor and arrival at the second $<$ average of all these values.

In all these cases our solution is the one with the highest accuracy compared to the other methods to generate proximity graphs. This shows how adding the functioning radius of the sensor to the generation algorithm can improve the result. Furthermore we checked the edges visually, by identifying which match roads and walkways that people can take, to confirm that our solution indeed fits best.

Because of the high correlation between the generated graphs using our solution and the inferred graphs, given by the real-life data set, with an accuracy of over 80% we can assume there is a high correlation between the proximity graphs generated using sensor location and radius, and the graph generated using crowd movement data. This implies that a possible application for the proximity graphs is the validation of sensor data for crowd movements, a large number of consecutive detections at sensors not connected in the proximity graph would raise alarms.

V. Conclusion

Considering static sensor networks and more accurately, sensors of WiFi packet used for detecting crowd movements we show how multiple proximity graph types can be applied to the system and how it compares to a proximity graph inferred from crowd behavior. The inferred proximity graphs were obtained from traces from two distinct deployments of WiFi-packet sensor networks.

We presented a solution to construct a proximity graph considering not only the location of all sensors but also their radius. We show, through direct comparison and through extensive analysis, how our solution manages to outperform the other proximity graph types, for this scenario. The work in identifying the best proximity graph type for this application needs to be continued, in this paper we do not claim to have obtained a perfect solution and there might be room for improvement. One possible direction for this research is the use of the maps in the calculation of the proximity graph.

Acknowledgment

The work has been funded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Ministry of European Funds through the Financial Agreement POSDRU/187/1.5/S/155420, and national project MobiWay, Project PN-II-PT-PCCA-2013-4-0321. We thank Roel Schiphorst from BlueMark Innovations for providing us with the Assen data set and to Spyros Voulgaris and Claudio Martella for their support and suggestions during this project.

References

[1] A. Ruiz-Ruiz, H. Blunck, T. Prentow, A. Stisen, M. Kjaergaard: Analysis methods for extracting knowledge from large-scale WiFi monitoring to inform building facility planning, PerCom (2014).

[2] M. Sonia: Distributed interpolation schemes for field estimation by mobile sensor networks, Control Systems Technology, 18(2):491–500, 2010.

[3] C. Jean, D. Simplot-Ryl: Energy-efficient area monitoring for sensor networks, Computer 2 (2004):40–46.

[4] H.-W. Bernhard, H. Lichtenegger, J. Collins: Global positioning system: theory and practice, Springer, 2013.

[5] G. Craig, Y. Koren: Distributed graph layout for sensor networks, Journal of Graph Algorithms and Applications, 9(3):327–346, 2005.

[6] S. Nattapong, P. Krishnamurthy: Location fingerprint analyses toward efficient indoor positioning, PerCom (2008).

[7] O.-S. Reza, P. Jalalkamali: Collaborative target tracking using distributed Kalman filtering on mobile sensor networks, American Control Conference (2011).

[8] S. Ricardo, R. Morla, A. Maio, J. Coelho: Analysis of the Logical Proximity between 802.11 Access Points, Conferłncia sobre Redes de Computadores (2013).

[9] P.-P. Carlos, A. Moreira: Analyzing the quality of crowd sensed WiFi data, PerCom Workshops (2014).

[10] G. Toussaint: The relative neighborhood graph of a finite planar set, Pattern Recognition, 12(4):261-268, 1980.

[11] K. Gabriel, R. Sokal: A new statistical approach to geographic variation analysis, Systematic Zoology, 18(3):259-270, 1969.

[12] D. Boris: Sur la sphère vide. Bulletin de l'Académie des Sciences de l'URSS, 6:793-800, 1934.

[13] T. Michael, T. Quint: Sphere of influence graphs in general metric spaces, Mathematical and Computer Modelling, 29(7):45–53, 1999.

[14] S. Stehman: Selecting and interpreting measures of thematic classification accuracy, Remote Sensing of Environment, 62(1):77-89, 1997.