SF-LG: Space-Filling Line Graphs for Visualizing Interrelated Time-series Data on Smartwatches

Ali Neshati
neshatia@myumanitoba.ca
University of Manitoba
Winnipeg, MB, Canada

Fouad Alallah
University of Manitoba
Winnipeg, MB, Canada

Bradley Rey
University of Manitoba
Winnipeg, MB, Canada

Yumiko Sakamoto
University of Manitoba
Winnipeg, MB, Canada

Marcos Serrano
University of Toulouse, IRIT - Elipse
Toulouse, France

Pourang Irani
University of Manitoba
Winnipeg, MB, Canada

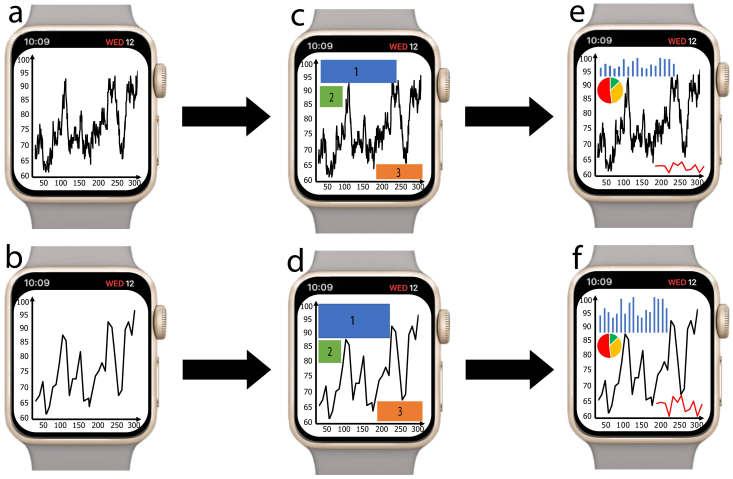

Figure 1: (a): A line graph representing heart rate over six minutes and including 300 data points; (b): we simplify the line graph in (a) using the Space-Filling Line Graph (SF-LG) technique; (c) & (d): the available space created around the line graph can be used to fill auxiliary, interlinked information. In our approach, we can provide the algorithm a priority to position the information. The labels "1", "2", "3" represent the priority assigned to those components which are given the maximal space; (e) & (f): embedding auxiliary and interrelated graphs that augment the primary line graph becomes possible based on the reorganization of the available space

ABSTRACT

Multiple embedded sensors enable smartwatch apps to amass large amounts of interrelated time-series data simultaneously, such as heart rate, oxygen levels or steps walked. Visualizing multiple interlinked datasets is possible on smartphones but remains challenging on small smartwatch displays. We propose a new technique, the Space-Filling Line Graph (SF-LG), that preserves the key visual properties of time-series graphs while making available space on the display to augment such graphs with additional information. Results from our first study (N=30) suggest that, while SF-LG makes available additional space on the small display, it also enables effective (i.e. quick and accurate) comprehension of key line graph tasks. We next implement a greedy algorithm to embed auxiliary information in the most suitable regions on the display. In a second study (N=27), we find that participants are efficient at locating and linking interrelated content using SF-LG in comparison to two baselines approaches. We conclude with guidelines for smartwatch space maximization for visual displays.

MobileHCI '21, September 27-October 1, 2021, Toulouse & Virtual, France

ACM ISBN 978-1-4503-8328-8/21/09…$15.00
https://doi.org/10.1145/3447526.3472040

CCS CONCEPTS

• **Human-centered computing** → **Visualization**.

KEYWORDS

Smartwatches; data visualization; line graph; graph simplification; time-series data

ACM Reference Format:
Ali Neshati, Fouad Alallah, Bradley Rey, Yumiko Sakamoto, Marcos Serrano, and Pourang Irani. 2021. SF-LG: Space-Filling Line Graphs for Visualizing

Interrelated Time-series Data on Smartwatches. In *23rd International Conference on Mobile Human-Computer Interaction (MobileHCI '21), September 27-October 1, 2021, Toulouse & Virtual, France.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3447526.3472040

1 INTRODUCTION

Embedded sensors on smartwatches provide users with a variety of personal time-series data, such as heart rate, breathing rate, oxygen levels and electrodermal activities. Smartwatch users are becoming increasingly dependent on such personal data to adjust their activity levels and behavior [11], also referred to as in-situ analytics [7] . However, unlike smartphones which can present multiple interlinked datasets, the small smartwatch displays make relating different data points difficult, a feature needed for performing tasks such as *"am I running faster than I did yesterday?"* or *"how did the change in my walking speed affect my heart rate"*, among others. The need for such in-situ queries is becoming more relevant [3] as users consider adjusting their actions levels in the midst of activity. However, presenting multiple, interrelated datasets on a small smartwatch display while maintaining the visualization's effectiveness is challenging. Novel visual representations are needed to maximize the available screen real-estate, not only to show key visual properties but to also include data from complementary data sources. Ideally, such systems can aid in complex, yet common, on-the-go queries.

Recent work suggests that not all visualizations on small displays are equally effective [5], and thus need to be designed carefully. Furthermore, it has been reported previously [3] that visualization could be a tool to respond to many of smartwatch users' queries on-the-go, which is not currently supported by smartwatches. Fitness enthusiasts or athletes for example, require such information to either preserve or modify their activity levels according to the set goals. One approach could consist of compressing visual characteristics on the small smartwatch display [1, 24, 31]. However, such an approach can still make it difficult to draw insight from multiple interlinked datasets that can be very easily collected on consumer grade smartwatches.

In this work, we explore means to effectively use a small display for presenting interlinked content. Unlike visual compression methods [24], we consider how best to segment and fill content around a line graph in order to facilitate users' complex time-series data exploration [35]. We introduce the Space-Filling Line Graph (SF-LG), which is designed first to simplify a line graph to make available additional screen space (Figure 1.b), and second to use the created space to place additional visuals to aid with complex queries (Figure 1.d, 1.f). In this initial exploration, we restrict our design of SF-LG to only rectilinear smartwatches as these are known to provide an optimal arrangement of content [23]. Through two studies, we find that the SF-LG method enhances users' ability to identify key line graph features. Furthermore, reorganizing space to embed additional charts enables complex visual queries directly on the smartwatch.

Our contributions include: (i) an algorithm that simplifies line graphs to maximize display space around it; (ii) a method to embed auxiliary, interlinked information in the newly available spaces; (iii) a validation confirming such an approach enables complex visual queries.

2 RELATED WORK

We present previous research regarding smartwatch visualization techniques, present common end-users' line graph queries, and finally review key line graph simplification techniques which inspired our work.

2.1 Smartwatch data visualization

For smartwatches, techniques for accurately collecting data have outpaced those for visualizing said data. Recent studies have suggested how visuals on such small displays could assist with complex tasks [11, 15, 42]. Through focus group studies, Amini et al. [3] identified some of the common queries end-users would desire to accomplish on smartwatches. They concluded that many of the tasks users prefer accessing while on-the-go are not supported by current platforms. With the aid of visual designers, they propose a range of visualizations, many of which have yet to be designed.

In an elaborate evaluation, Blascheck et al. [5] highlight the need to carefully choose visualizations on smartwatches. Their investigation compared the three most commonly used data charts (bar, donut, and radial bar charts), with a task requiring participants to quickly compare two data point values. Interestingly, their results suggest that the radial bar chart, one of the most common visualizations on smartwatches, is the least efficient for quickly viewing data on smartwatches. Furthermore, we note that line graphs can also be compressed to show more content. G-Sparks [24], a method inspired by Sparklines [36], compresses a line graph to include one data sample per pixel which can minimize the number of flicks required to access lengthy time-series data on smartwatches. These prior works point at the need for renewed guidelines to address the growing interest towards data visualizations on small form-factor devices.

2.2 Space-efficient data presentation

Space-efficient visualization techniques take up a minimal footprint [24, 26, 31, 36] and can be referred to as micro- or small-scale visualizations [5, 25]. The Horizon Graph [26, 31], for example, colour codes line graph components making it possible to compress these along the y-axis. By conducting a user experiment, Javed et al [18], showed how using the Horizon graph can be used for some specific visual exploration tasks. However, the colour scheme requires interpretation which can be challenging to do on a small display. Space-filling techniques are also efficient, and they are designed to maximize the available display space [19, 33, 37, 40]. As a result, space-filling techniques reduce chart junk [18] and rely on highlighting salient content [39]. The Treemap [19, 33, 39] is a classic example of a space-filling technique for hierarchical data. We build on such concepts to maximize the space around a line graph on a small display.

2.3 Line graph simplification methods

Line graph simplification, or smoothing methods, focus on decreasing the complexity of graphs by reducing the number of data points. Many such techniques exist [30, 34]. In a research paper, Rosen et al. [29] evaluated 12 different smoothing techniques and proposed a taxonomy of these techniques. However, these smoothing methods

are complex and hard to implement on smartwatches with limited processing resources. In this study, we focus on three existing line graph simplification methods that inspired our design. These three simplification algorithms were chosen because they do not require powerful computational resources, making them suitable for smartwatches with limited processing power.

Sampling: Sampling is one of the most commonly used simplification techniques [4] whereby fixed-size intervals are used to select data points (e.g., every minute). The larger interval sizes reduce the complexity of data by having fewer data points (e.g., every hour) and can be beneficial for trend and pattern recognition tasks. In contrast, smaller fixed-intervals generate more data points (e.g., every 5 seconds) and can be beneficial for identifying incidents. Sampling does not prioritize the data in the way our visual system does, and thus we can potentially lose salient information (e.g., maximum and minimum) unless they coincide with the selected intervals.

Piecewise Aggregate Approximation: The Piecewise Aggregate Approximation (PAA) technique simplifies line graphs using a fixed-size window. In PAA, the mean of the data points in each window is calculated, and simplification replaces the original data points with the mean of each window [20]. Thus, similar to Sampling, simplified graphs with PAA often do not contain the actual critical data points (e.g., maximum and minimum). Furthermore, the effectiveness of this method hinges on the size and the number of windows. For example, the simplified graph might not hold the shape of the original graph when the window size is large.

Perceptual Important Points: Chung et al. [6] introduced the Perceptual Important Points (PIP) technique which inspired SF-LG. To compute the PIP graph, first, the furthest data point on the line connecting the first and the last points in the graph is detected. The process is then repeated for each one of the two components separately (e.g., selected data point with the first data point and selected data point with the last data point). While PIP requires intensive resources compared to Sampling or PAA, it can preserve the maximum and minimum data points. PIP was explicitly designed to reduce the complexity of line graphs with a large number of data points. It has many applications in areas such as health and stock market forecasting [13]. It is also one of the only simplification techniques that considers the visual aspect of important features in the graph during graph simplification.

2.4 Perceptual tasks on line graphs

Several studies exploring users' graphical perception with varying graph types [8–10, 18, 27] identified three essential perceptual tasks. These include (i) maximum/minimum point detection [2, 12, 18, 21, 22, 26], which requires finding the highest or the lowest point in a graph (e.g., peak daily heart rate); (ii) value detection, which requires reading an exact data point [16, 18, 21] (e.g., comparing peak heart rate on different days); (iii) value comparison [5, 24], which requires comparing two data points and identifying which one is higher, or how much is the difference between two highlighted data points in a chart; (iv) trend detection [14, 41], which can give users an overview of the entire data. Thus, from the findings in these works we incorporate these perceptual tasks in our evaluations.

3 EXPLORATORY REVIEW OF SPACE UTILIZATION ON SMARTWATCH APPS

To inform our design, we first examine how current smartwatch applications utilize space when presenting data visualizations. Our analysis included highly ranked fitness and health-related smartwatch apps from Google Play and the App Store (12 watchOS, 2 Android Wear, and 8 Android Wear & watchOS apps in total). To do this, one of the authors collected the top 40 ranked apps in the fitness tracking category, which include capturing sleep quality, heart rate, workout duration, and water consumption, from both Google Play and the Apple App Store (top 20 ranking apps on each platform). These apps were specifically designed for smartwatches. We installed and tried all 40 apps on two smartwatches, an Apple Watch series 4 with a 44mm display, and an Android Macwear M7. From these 40 apps, only 22 of them used data visualization techniques to represent the data and the rest only used text or icon to represent fitness information. This result confirms the findings reported by Islam et al. [17] that using charts is not currently a common way to represent data on smartwatch applications and needs more investigation. We then extracted all visualization techniques from 22 selected apps by using the built-in screen capture feature on both smartwatches, and measured how much of the entire space was occupied by the visualization techniques. We also measured the empty space used throughout these screens. Using Inkscape [1] we measured the exact proportion of the occupied space over the empty space by calculating the rectangular space an element occupies on the screen (e.g., Fig 2). The empty space is equal to the area of the entire screen minus the area occupied by all elements.

Our review focused solely on apps available for rectilinear smartwatches as this was our target platform in this initial study. The choice of a rectilinear watch was to even out our comparison between Apple and Android apps. In total, we reviewed 38 interfaces from 22 smartwatch apps (some apps had more than one screen). We then explored each app by examining (i) space usage and (ii) varieties of chart types used.

3.1 Space utilization

Based on the selected apps, we explored space utilization on rectilinear smartwatch displays to understand how designers organize visual data. In particular, we aimed to understand the interplay between the space allotted to the primary visualizations and auxiliary information, such as labels, icons, or other charts. To this end, we first computed the entire area of the interface and the area allotted to visual elements on screen. We further categorized on-screen content into three common visual elements we observed on such interfaces: data visualizations/charts, text, and icons. The space allotted to each element, the free space without visual elements, and the occupied ratio (percent of the interface area with visual elements on it) were manually computed based on screenshots we captured. Figure 2 shows these measures on four selected smartwatch visualization interfaces.

[1] https://inkscape.org/

We observed a high degree of variability in the amount of free space for the 38 interfaces (Figure 3.left). In total, more than half (55%) of the interfaces we examined had between 40% to 70% of free space. Further, no app had free space that was larger than 70%. We note that free space is important for improving the legibility of content but we were equally surprised to see such a large amount devoted to no visual content.

There was also a large variability in space occupied by charts (Figure 3.right). In the majority of the apps (60%) charts occupied less than 50% of the available space, whereas only 26% of the apps used more than 70% of the available space. 89% of the interfaces used icons that occupy between 0-10% of the interface space, whereas only 10% of the interfaces had icons that occupied between 10–20% of the space. Interestingly, the analysis of text showed text elements occupied between 0% to 30% of the total space across all interfaces. About half (47%) of the apps used text that occupied 10–20% of the screen. While specific to the apps we surveyed, these results provide insightful context on how designers distribute balance visual content and free space on the small smartwatch screen.

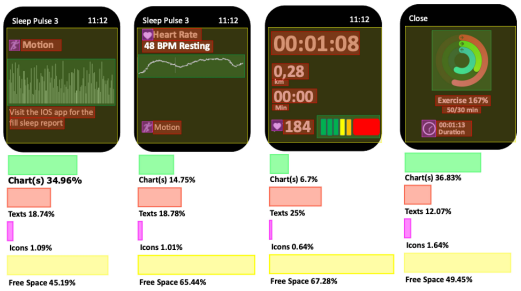

Figure 2: Space usage analysis on four illustrative visualization interfaces across two applications [2]indicating the percentage of free space and of space occupied by each visual element: charts, texts and icons.

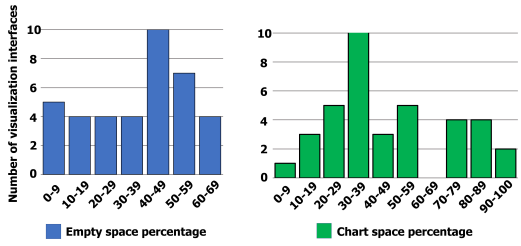

Figure 3: (left) Distribution of free space and (right) Distribution of space occupied by charts in the reviewed apps (N = 38).

3.2 Chart types

The reviewed apps most commonly employ Bar charts, used in 50% of the apps, followed by Donut (26%), Line (24%), and Scatter (5%) charts. We also looked at the number of charts used in each interface. The analysis showed that 95% of the visual interfaces used one graph compared to only 5% that used up to four charts. This also confirms the results reported by Islam et al. [17] that the average number of chart representations per watch face was nearly one across a wide range of smartwatch applications. Our result shows that most existing smartwatch applications deploy separate visual displays to represent charts which means each visual display represents a single chart. For instance, a line chart can be used to describe heart rate data, and in the following display, a radial bar graph to show the proportion of different activities the user did in a day.

Based on our analysis, we summarize the following key observations (OB), which are closely tied to the review of apps we examined:

- **OB1:** The majority of the apps use little space for their key charts, and do not take advantage of the entire display.
- **OB2:** Most interfaces have a considerable amount of free space. This potentially implies that the overall space utilization could be further optimized to include auxiliary content to the primary data chart.
- **OB3:** Most apps that include more than one visualization from multiple sources (e.g., a radial graph for steps walked and a bar graph of burnt calories) require users to flick between these visual displays. Interlinked charts are rarely presented on a single screen.
- **OB4:** Most apps reviewed used Donut charts rather than Line charts to represent time-series data. However, earlier work [28] has shown Bar and Line charts are best suited to visualize trends and numerical values over time.
- **OB5:** We found that many of the icons augment textual descriptions (see Figure 2; heart rate icon and label). These are important in describing the charts and could be further optimized (but we leave this for future work).

These observations suggest that chart interfaces can be further optimized to use the available space. Either by (i) enlarging the chart or (ii) augmenting the chart with either additional information/graphs. Furthermore, we believe additional space can be made available if the area devoted to the chart is optimized. For this reason, we next seek to expand the available space by (a) simplifying the chart and then (b) augmenting the freed space.

4 SPACE FILLING LINE GRAPH (SF-LG)

We introduce a Space-Filling Line Graph technique that has two elements, (i) a graph simplification algorithm and (ii) efficient embedding of auxiliary data. We first present our simplification approach which frees screen space and then describe how we embed supplemental charts compared with non-simplified graphs (Figure 1). Similar to PIP, we also focus on data points that have "significant meanings", such as peaks, maximum/minimum(s), and outliers. In SF-LG, we look for data points with the highest average distance to all other data points. This provides us with a subset of data points in the line graph that have significant meaning (i.e., anomalies).

We calculate and store the average of the Euclidean distance of each point p_i such that ($i \in [0, N]$, where N is the sample size) for the entire set of points (P) in the graph. A point with higher average distance indicates that this data point is relatively far away from the

[2]Runtastic Run, Mileage Tracker from Apple App Store

rest of the dataset. This causes a challenge because the right-most (and left-most) points in the graph have no neighbours to its right (or left). Furthermore, a large difference in x-values of the right- and left-most data points in the graph makes the average distance of these points artificially larger. This implies that data points with very high and very low x-values will have a higher average distance compared to data points at the graph's center (i.e., data points with neither very high nor very low x-values). To resolve this, akin to the PAA approach, we define windows. With the windowing technique, we can select points with the highest average distance to all other data points, compared to all of its neighbours, for that same window (Figure 4). As such, we can calculate and select the essential points in each graph window. Algorithm 1 shows the SF-LG pseudocode and Figure 4 shows how SF-LG works.

Algorithm 1: Simplification algorithm.

```
Input: A set P, of points in the graph with x and y
       coordinates and window size
Output: A subset of P, representing data points with the
        highest average distance to all other data points
for each data point p_i in P do
    Sum = 0;
    for each data point p_j in P (i != j) do
        EuclideanDistance = calculate Euclidean distance
         from p_i to p_j;
        Sum = Sum + EuclideanDistance;
    AVGDistanceArray[i] = (Sum / number of data points -
     1);
Divide the set of data points into subsets (windows);
for eachwindow do
    mark the point with the highest AVGDistanceArray[i];
return marked points
```

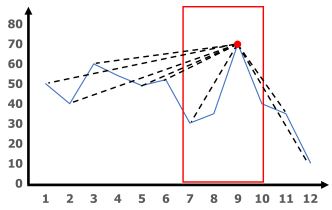

Figure 4: SF-LG calculating the average distance of the red data point in the red rectangle to all other data points in the graph leads to the highest average distance compared to the remaining two data points in the same window. This process is performed for all data points in all windows to choose the data point with the highest average distance.

4.1 Comparing simplification techniques

The aforementioned Sampling technique selects data points with fixed and specific intervals and does not consider the importance of data points. SF-LG solves this by focusing on data points that are more important in the dataset (e.g., anomalies and extreme points), similar to the PIP method. Furthermore, unlike Sampling, as we are using the windowing technique with actual data points in SF-LG, many visual features of the line graphs such as trends, patterns, extreme points, and the general shape of the graph are preserved.

Similar to PAA, increasing the window size (i.e., fewer windows) will reduce data points in the graph, which can affect the representation accuracy of the original graph. Because PAA uses the average of each window and not real data points, the PAA-simplified version of the graph could significantly differ from its original. However, preserving the original extreme data is often crucial, especially for biometrics (e.g., extremely high heart rate). In comparison, because SF-LG uses actual data points, there will be no distortion in the simplified graph.

SF-LG deploys the same concept used in PIP but with improvements aimed at representing and simplifying line graphs on smartwatches. In different windows, data points are selected that have close values (data points that are further away from the rest of the data) which can prevent extreme fluctuations in the graph. This has the effect of creating additional space (Figure 5) which can be used to present augmented information such as pictures and graph details (Figure 1.f). Furthermore, smartwatch hardware is very limited and for this reason we chose to compare PIP with SF-LG. Limited processing capabilities of smartwatches prevent application designers to deploy complex algorithms to visualize or analyze the data. Using non-intensive algorithms such as PIP, and our SF-LG method, smartwatches can take advantage of these simplification techniques. To confirm this, we compared the processing time it takes for both SF-LG and PIP algorithms to simplify 30 data sets with 300 data points. To ensure everything is the same for both conditions, we used the same data sets for both algorithms. On average, it took 12.76 ms (min=11 ms, and max=22) for SF-LG and 17.25 ms for PIP (min=15 ms, and max=22) to simplify these graphs. This result illustrates how fast these algorithms are on smartwatches with limited processing capabilities to simplify large data.

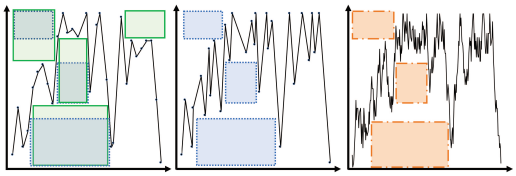

Figure 5: Visually comparing simplification techniques for line graphs: left) SF-LG, middle) PIP method, right) non-simplified graph. Green spaces in the left image represent the spaces created by the SF-LG method, overlapping with blue spaces generated by the PIP technique.

5 STUDY 1: SF-LG GRAPH SIMPLIFICATION

In Study 1 we compared the effect of simplification with SF-LG against two baselines, the Non-Simplified and PIP techniques. We chose PIP as one of our baselines for the following reasons: 1) it is one of the simplification methods that does not require computational power which makes it suitable to be implemented on smartwatches with limited computational capability; 2) it has been commonly used to simplify health data, which is one of the main

reasons smartwatches are being used, 3) it preserves many of the important data points of the original graph (e.g., maximum and minimum), and 4) considering the visual aspects of the graph to simplify the graph.

5.1 Conditions

Our evaluation included three visual conditions: SF-LG, Non-Simplified, and PIP. PIP was carefully selected for three important reasons: (i) PIP is one of the most widely applied simplification techniques; (ii) PIP is the only major technique that does not compromise crucial data points such as maximums and minimums; (iii) PIP is a simple algorithm which can be deployed by smartwatch applications due to limited processing and memory capabilities of smartwatches. Thus, having these conditions will allow us to investigate a) whether SF-LG can improve users' performance (i.e. accuracy and response time) compared with the Non-Simplified graph; b) whether SF-LG achieves at least equal performance compared with PIP, a popular simplification method. Other simplification techniques are also available but given the popularity of the PIP approach, and its ability to preserve salient data points, we use this in our exploration.

5.2 Participants and apparatus

Thirty (30) participants, 10 for each condition, (F = 13; M = 17; $Mage$ = 28.53; 27 right-handed), with normal to corrected vision, were recruited via poster advertisement at a local university. Participants completed the Ishihara colour blindness test [3] which confirmed that none were colour blind.

We used an IMACWEAR M7 smartwatch, with a 1.54" and 240×240 resolution display. To avoid any distraction by the title or notification bars on the screen, we used the entire screen for the study. For user input, based on [5], we used a Targus AKP03CA Bluetooth keypad (Figure 6) so we could solely focus on the visualization aspects without introducing possible confounds (e.g., fat-finger concerns) due to small displays. Participants were asked not to touch the display.

We chose not to control the orientation and the distance of the smartwatch, unlike [5] for three main reasons: 1) to preserve the authenticity of users' daily smartwatch usage without any artificial restrictions; 2) even if the distance was fixed at the beginning, the participants could still move their heads; 3) the discomfort that fixed distance could induce may impact the participants' performance.

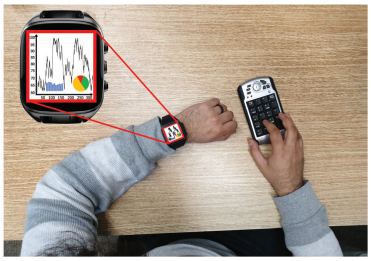

Figure 6: Apparatus and setup used in both studies.

[3]https://enchroma.com/pages/color-blind-test

5.3 Tasks

There were five tasks in total, three of which were adapted based on previous studies (see Perceptual Tasks on Line Graph section 2.4). All tasks are described below. For each task, each participant was asked to process ten unique line graphs. Participants had to finish one task to be able to carry on to the next task. Each graph contained 300 data points. We used a partial latin square design for the study. Furthermore, to boost the generalizability of our results, we used real heart rate data as stimuli instead of synthetic data [13, 24].

Maximum/Minimum Detection: In this task, participants were asked to detect the highest/lowest data points on each graph. Participants were asked to report the x-value of the max/min data point using the keypad, and pressed Enter to move to the next line graph. To avoid any confusion, each graph had only one maximum and one minimum data point.

Value Detection: In each line graph, one data point was marked with a small red circle, and participants were asked to type the exact y-value of this point. This is a commonly performed task on line graphs (e.g., reading the exact heart rate, the number of steps, or the body temperature).

Value Comparison: Two randomly selected data points were marked with small red circles in each graph. The participants entered the difference between them as measured on the y-axis.

Trend Detection: Participants were asked to indicate the overall trend of each line graph (i.e., an increase or decrease) by pressing the UP or DOWN arrow buttons on the keypad. We ensured that each graph contained an unambiguous trend. An ambiguous trend line is a trend line that is close to the flat (horizontal) line. This means that it is not clear if the trend is increasing or decreasing.

5.4 Study design and procedure

The study followed a between-subject design with one factor, Interaction Technique (SF-LG vs. Non-Simplified vs. PIP). After reading and signing the consent form, participants were randomly assigned to one of the three conditions. A between-subject design was selected to avoid potential cognitive fatigue because the entire session was designed to take longer than 30 minutes. For each condition (e.g., SF-LG, PIP and Non-Simplified), we counterbalanced the order of five tasks by using a latin square design. Before each task, a research assistant explained the process and gave detailed instructions. Participants were asked to perform each task as quickly and accurately as possible. First, each participant was allowed practice trials with sample line graphs until they felt confident about performing each task. Participants were informed that all the data points' x- and y-values were integers (i.e., whole numbers), and not fractions (e.g., 56.5). Participants received a $10 gift card upon completion.

To collect post-study user preference, we presented participants with three graphs (SF-LG, Non-Simplified, & PIP) made from the same datasets. Participants were asked to rank the graphs based on their preference. Also, participants' Accuracy and Response Time were recorded as dependent variables for further analysis.

5.5 Results

Shapiro-Wilk tests yielded that certain data collected was normally distributed while some was not normally distributed. Thus, Kruskal-Wallis and Mann-Whitney U tests were conducted for the non-normal data. Otherwise, we conducted a one-way ANOVA test on the normal data. Furthermore, Bonferroni correction was applied to reduce Type I errors (i.e., $p = .05/3$).

Maximum/Minimum Detection: For the accuracy of maximum/minimum detection, we did not find any significant difference across conditions. However, for the response time in both maximum and minimum detection tasks, there was a significant difference between the three conditions (Figure 7). First, the result of the Kruskal-Wallis analysis for maximum detection was significantly different ($\chi^2 = 13.94, p < 0.001, df = 2$). To identify the location of the effect, Mann-Whitney U tests were conducted. For the maximum detection task, there were significant effects between SF-LG and Non-Simplified ($U = 58.00, p < .001$) but no significant effect between PIP and Non-Simplified nor SF-LG and PIP was found (SF-LG; $Mdn = 5599ms$, PIP; $Mdn = 7788ms$, Non-Simplified; $Mdn = 10326ms$). Altogether, in comparison to the Non-Simplified graph, SF-LG was able to improve the response time for maximum detection tasks; such improvement was not found with PIP, however.

Because response time for the minimum detection task was normally distributed, we employed a one-way ANOVA test. A significant effect was found among the three conditions, $F(2, 27) = 12.62, p < .001$. Tukey post-hoc tests revealed that there was a significant difference between SF-LG and Non-Simplified ($p < 0.001$; SF-LG; $M = 5847ms$, PIP; $M = 8254ms$, Non-Simplified; $M = 11096ms$). Thus, parallel to the maximum detection, SF-LG again yielded the improved response time compared to the Non-Simplified.

Value Detection: For the accuracy in value detection we did not find any significant condition effects. However, a Kruskal-Wallis test revealed a significant effect for response time ($\chi 2 = 17.99, p < 0.001, df = 2$). A Mann-Whitney U test yielded a significant effect between SF-LG and the Non-Simplified condition ($U = 56.00, p < .001$), SF-LG and PIP ($U = 71.00, p = .009$), as well as PIP and Non-Simplified ($U = 70.00, p = .007$). The median response time was 4328ms with SF-LG, 6583ms with PIP, and 14106ms with Non-Simplified.

Value Comparison: For both response time and accuracy, we did not find any significant condition effects.

Trend Detection: A Kruskal-Wallis test found a significant effect among the three conditions in response time ($\chi 2 = 10.4, p = 0.005, df = 2$). Furthermore, Mann-Whitney U tests found significant effects between SF-LG and Non-Simplified graphs ($U = 63.00, p < .001$; SF-LG; $Mdn = 4890ms$, Non-Simplified; $Mdn = 12166ms$). No statistically significant effects were found in accuracy.

Participant preference: The majority of the participants (83.33%) preferred simplified graphs over Non-Simplified graphs, and more than half of the participants (53.33 %) ranked SF-LG the highest (Figure 8, Left chart). Many participants noted that the SF-LG graphs

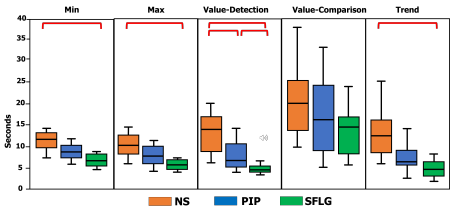

Figure 7: Response time (secs). Sig. differences between pairs are noted by the red lines.

were more simple than the two other techniques and easier to understand. Some of the participants believed that representing many data points on a small smartwatch screen could backfire (e.g., P4: "Although more data point gives me the best data output, it is more uncomfortable and difficult to understand."[sic], P5: "[Non-Simplified] graph is too complex." [sic], P23: pointed out that "too many data points on the graph cause distraction and inaccurate interpretation").

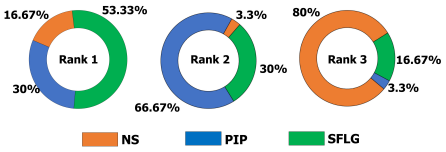

Figure 8: Proportion of each of the three techniques' rankings. SF-LG was ranked 1st by 53.33% (left), and ranked 2nd (in the middle) by 30%. We observe a higher number of 1st and 2nd rankings for SF-LG, than PIP or NS.

6 SPACE MAXIMIZATION AND EMBEDDED GRAPHS

SF-LG frees up space on the smartwatch display that we use to represent auxiliary information useful for queries requiring interlinked datasets. Collating data sources on multiple views is common and also available on commercial software applications. Tableau, for example, does this in the form of dashboards to aid in improved data analysis. To place additional charts on the available empty space on SF-LG, we develop a greedy algorithm consisting of two steps. First, the system calculates the entire space available. Second, we select regions for maximum graph legibility.

6.1 Space calculation

To calculate the available space around the line graph, we divide the graph into the area above and below. We explain our calculation for the lower area as the same method is applied to the upper area. The algorithm starts from the leftmost data point of the graph. If the gradient of the line where this data point is located is positive, the algorithm calculates the horizontal line (parallel to the x-axis) connecting this data point to the first intersection of this horizontal line, and the line graph, or the border of the graph. The rectangular area beneath the calculated horizontal line will be the available space in the line graph which can be used for additional content. To

continue, we just need to add a small value (e.g., one pixel) to the x-value of the starting data point, on the line graph, and repeat the process as for the first data point. For data points with a negative gradient, we have the same calculation in the upper area of the graph.

In another step, we calculate all available space under the line, and start from the bottom of the y-axis and move upward to reach the data point on the y-axis of the graph. Our algorithm does this by adding a small value, such as a pixel, to the y-value of the previous point. In each step we calculate the horizontal line, connecting this point and the intersection of this horizontal line with the graph. The output of this space calculation provides an array of available spaces with different heights, widths and areas. We use these calculated spaces in the next step of our greedy algorithm to allocate auxiliary visualizations. Algorithms 2 and 3 show how the available space beneath the line graph is computed. With few changes, this algorithm can be used to calculate all the available spaces above the line graph as well.

6.2 Space allocation

The space allocation component of the algorithm takes an array, ChartsArray, as one of the input arguments, representing the type of embedded visualization we wish to add to the line graph (e.g., ChartsArray = [Bar, Pie, Pie, Bar, Line]). The output of the previously mentioned algorithm is another input argument containing all possible available spaces to present additional information in the line graph. For instance, ChartsArray = [Bar, Pie] indicates that we give the best available space on the screen to the first item, a bar chart. The pie chart uses the next largest available space. Charts, such as pie and donut are circular while graphs such as bar and line graphs have horizontal rectangular shapes. Since we calculate all available quadrilateral spaces in the line graph, the algorithm can use the available spaces that are horizontal-rectangular for line and bar charts, while using square-shaped spaces for pie and donut charts. As finding the exact square shape space is intractable, we define a square shape as a rectangle that follows one of the conditions in Formula 1. A pilot study on our smartwatch showed that rectangles that satisfy one of the following conditions could be considered as square spaces.

$$\begin{aligned} Height - (1/20)Height < Width < Height + (1/20)Height \\ Width - (1/20)Width < Height < Width + (1/20)Width \end{aligned} \quad (1)$$

The previous steps lead to multiple small rectangular spaces. However, after experimental implementations, we noticed that such spaces are too small to represent many graph types such as line and bar charts. Similarly, available spaces with a very high (or low) ratio of width over height are not suitable for these graph types. Therefore, we implement restrictions on how we define horizontal rectangular space in our algorithm as follows: (i) the minimum height and the width of the space should be 60 pixels; (ii) the ratio of the width over height should be greater than 3/2.

Based on the chart type to display and its order in ChartsArray, the algorithm seeks the largest available area and allocates that space around the central line graph. In the next step, the available spaces need to be recalculated only for those spaces that have a common area with the space occupied by the newly added chart. For instance, for ChartsArray = [Bar, Pie], the algorithm first tries to calculate all available spaces. Then it identifies the largest area with the aforementioned characteristics for horizontal-rectangular spaces to insert the bar graph. Finally, the algorithm recalculates the spaces that have a common area with the occupied space. The algorithm then runs a similar process to insert a pie chart. Changing the assignment priority gives a different layout (Figure 9). Algorithm 4 represents a pseudocode of the space allocation algorithm.

Algorithm 2: SpaceCalculation(P)

```
Input: A set P, of all data points in the line graph with x
       and y coordinates
Output: An array of all available spaces underneath the
        line graph (AllEmptySpacesArray)
YLeftTop = 0;
AllEmptySpacesArray[];
while YLeftTop! = p_0y do
    PointIntersection =
     CalculateIntersectionPointWithGraph(0, YLeftTop);
     AllEmptySpacesArray.add(Rectangular space
     between(0, YLeftTop,
     PointIntersectionX,PointIntersectionY);
     YLeftTop++;
for each data point t on the line segment between all P_i and
 P_i+1 do
    PointIntersection =
     CalculateIntersectionPointWithGraph(tX, tY);
     AllEmptySpacesArray.add(Rectangular space
     between(tX, tY, PointIntersectionX,
     PointIntersectionY));
return AllEmptySpacesArray
```

Algorithm 3: CalculateIntersectionPointWithGraph(X,Y)

```
Input: X and Y coordinates of a point
Output: Horizontal intersection of this point with graph
for all data points of line graph with x-values greater than X
 do
    find the first two subsequent data points on the line
     graph, PNext and PPrevious, such that PNextY <= Y
     and PPreviousY >= Y;
    return the x and y coordinates of the horizontal
     intersection point between (X,Y) and the line segment
     between PPrevious and PNext;
if there are no such PNext and PPrevious then
    return x coordinate of the right border of the line graph
     and Y
```

7 STUDY 2: EMBEDDED GRAPHS

Study 2 explored the capability of the SF-LG technique to efficiently present multiple chart contents on a smartwatch display to facilitate users' complex visual queries.

Algorithm 4: GreedySpaceAllocation(ChartsArray[], AllSpaces[])

Input: An array of all available rectangular spaces sorted from maximum to minimum area (AllSpaces) and an array indicating the priority of representing charts (ChartsArray)

Output: Additional charts allocated to the optimal spaces within the main line graph

```
foreach CurrentChart in ChartArray do
    if CurrentChart = Bar | CurrentChart = Line then
        foreach CurrentSpace in AllSpaces do
            if CurrentSpace = Rectangle then
                Embed the new Line\Bar chart in this space;
                SpaceCalculation(); \\to update AllSpaces
                break;
    else
        foreach CurrentSpace in AllSpaces do
            if CurrentSpace = Square then
                Embed the new Pie\Donut chart in this
                  space;
                SpaceCalculation(); \\to update AllSpaces
                break;
```

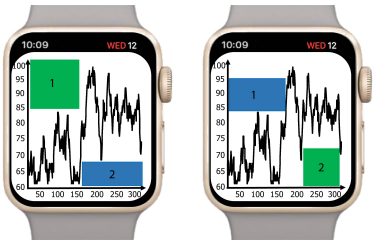

Figure 9: The greedy algorithm is provided with desired charts in a prioritized list of importance to be shown first. Based on the chart properties (rectangular or square) the algorithm identifies the optimal region to insert them. (Left) The priority given to the algorithm is to first find the largest square space and then a rectangular region (for example, a donut chart and bar chart respectively). (Right) The reversed priority (bar chart first, then donut chart) provides a different layout as it seeks to give the rectangular component the larger space first and then allocates space to the donut chart.

7.1 Conditions and stimuli

In Study 2, we evaluated the use of SF-LG against embedded graphs in a Non-Simplified graph. By comparing these two conditions, we will identify if generated spaces and embedded auxiliary information in SF-LG are effective or not compared to Non-Simplified graphs. We also included Flicking in our comparison as it is the most common method for exploring multiple graphs on a smartwatch. Our investigation of the existing smartwatch applications (section 3.2) revealed that more than 90% of these applications use only one visualization in each visual display. A similar result also reported by Islam et al. [17]. Therefore including the Flicking condition is very similar to existing smartwatch applications to represent multiple charts.

One potential condition in this experiment could be dividing the screen into segments, for instance, on the y- or x-axis, and use each segment to represent one graph. As we wanted to use different types and shapes of graphs in this study (square and rectangle shapes), representing multiple graphs with different shapes could be highly dependent on the layout of these graphs, which will add more complexity to our study. So, we excluded this condition in this experiment. Thus, we arrive to three conditions; (i) SF-LG, (ii) Non-Simplified Embedding (NSE), and (iii) Flicking.

For the SF-LG and NSE conditions, we applied our greedy algorithm to identify the best locations to place the embedded charts. In the Flicking condition, one graph was presented per screen and users flicked through three screens in total per trial. Participants were allowed to go back and forth between the screens. For embedded charts, three of the most commonly used types on smartwatches were selected: line chart, bar chart, and pie chart [5]. Note, specific values were not represented on embedded charts, as these are often provided to help users get an overview of the central graph instead of presenting specific numerical values (e.g., to identify when the maximum/minimum happened, or to detect a trend).

The main line graph represented real heart rate data (e.g., the black line graphs on Figure 10) which ranged between 60 and 90 bpm, taken over 24 hours. This range was divided into three subranges: low (60-70 bpm), mid (70-80 bpm), and high (80-90bpm). The proportion of these ranges were represented with an embedded bar or pie chart (e.g., Low in Green, Medium in Yellow, and High in Red in the bar chart in Figure 10). The embedded line chart was randomly generated with data points ranging between 0-200 (e.g., the purple line chart in Figure 10), representing the burnt calories associated with the heart rate over 24 hours.

Note, our greedy algorithm selected either a line graph or a bar chart when the conserved space was rectangular. In contrast, a pie chart was selected when the conserved space was a square shape in the main line graph.

7.2 Task

The main goal of this study was to explore the participants' capability to link information between the main line graph and embedded charts. This analytic task consisted of the user understanding the overall tendency on heart rate (represented by the main line graph), and to identify how it was associated with burnt calories (an embedded chart). This query type is commonly of interest to users of in-situ visualizations, such as fitness enthusiasts [3].

Users had to follow these four steps to successfully complete a trial. Step 1: identify the most frequently occurring heart-rate range from the embedded chart (bar or pie). In Figure 10 (left), the highest value in the bar chart consists of heart rates in the high range (80-90bpm). Step 2: identify the corresponding region on the central line graph. This is marked in red in Figure 10 (center). Step 3: based on the region in Step 2, use the second embedded chart (purple line graph in Figure 10) to make a visual estimation of the average burnt calories. This required users to estimate this average based on the selected region in Figure 10 (right). Note, as both line

graphs represented data recorded for 24 hours, this identification process was performed by comparing the relative segments of the line graph with the embedded line chart, on the x-axis. Step 4: indicate whether the mean within this region falls above or below the middle line (dotted in Figure 10 right), by pressing the Up or Down arrow keys.

Figure 10: Illustration of Study 2's task. (Left) Users need to first find the maximum value in the embedded bar chart. (Middle) Using the previously found value, the user next has to find the corresponding data points in the line graph. (Right) Finally, users provide their best visual estimate of the average value of the corresponding data points in the lower auxiliary line graph. This is representative of a task such as wanting to know the maximum burnt calories during peak performance

7.3 Study design and procedure

To prevent possible learning effects and cognitive fatigue, we used a between-subject design with one factor, the Visualization Technique (SF-LG vs. NSE vs. Flicking). First, participants went through a practice session. They performed all the tasks until they felt comfortable. Importantly, participants were asked to perform all the tasks as quickly and as accurately as possible. If the incorrect response was entered, the users repeated that task after completing all trials. They performed 20 trials per visual condition. As in Study 1, participants used a Bluetooth keyboard to prevent any confounds. We measured response time and accuracy. After they completed the graph exploration tasks, we provided sample stimuli from each of the Visual Techniques and asked them to rank the visualization techniques based on their preference. The study took approximately 30 minutes.

7.4 Participants

Twenty-seven (27) new participants volunteered ($F = 15; M = 12$; $Mage = 27.31$; 26 right-handed) from a local university. Participants received a $15 gift card as compensation.

7.5 Results

We applied the same statistical tests as in Study 1.

Response Time: A Kruskal-Wallis test revealed a significant difference among three conditions ($\chi 2 = 20.63, p < 0.001, df = 2$). Further, Mann-Whitney U tests yielded significant effects between all the pairs (SF-LG vs. NSE, $U = 55.00, p = 0.006$; SF-LG vs. Flick, $U = 45.00, p < 0.001$, NSE vs. Flick, $U = 45.00, p < .001$, SF-LG; $Mdn = 9764ms$, NSE; $Mdn = 18903ms$, Flicking; $Mdn = 37106ms$) (Figure 11).

Accuracy: Because accuracy was normally distributed, we applied one-way ANOVA along with Tukey post-hoc tests. An ANOVA test revealed a significant difference among the three conditions, $F(2, 24) = 6.82, p = .005$. Pos-hoc tests revealed only one significant difference: SF-LG was significantly different from Flicking ($p = 0.003$; SF-LG; $M = 57.7\%$, Flicking; $M = 40.3\%$). Altogether, accuracy results indicated an encouraging potential for SF-LG on a smartwatch display. Although overall accuracy rate might not appear sufficiently high, placing multiple graphs on one page yielded a generally higher accuracy than placing them on different pages (Figure 11).

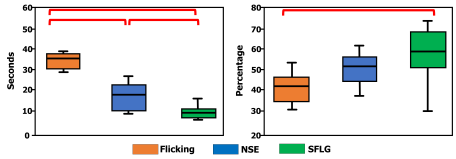

Figure 11: (Left) Participants' response time and (Right) Participants' accuracy in Study 2. significant differences between pairs are noted by the red lines.

Participants' preference: More than 65% of the participants preferred the SF-LG method compared to the NSE or Flicking methods, and more than 60% of participants rated Flicking as the least favorable condition (Figure 12). Participants commented on the simplicity element of the SF-LG, P10: "the graph has less data points so that makes it less dense and easy to understand the trend."[sic]). Also, multiple participants noted that seeing all three graphs at once made the task easier and more accurate to perform in comparison to flicking (e.g., P26: "I don't have to swipe to different screen to compare different conditions"[sic], P13: "I prefer the 'SF-LG' because [...] it is easier to see the relationship between all the graphs" [sic]).

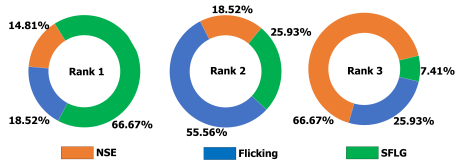

Figure 12: Proportion of each of the three techniques' rankings. SF-LG was ranked 1st by 66.67% of participants.

In summary, results from Study 2 reveal SF-LG benefits users on a smartwatch for tasks requiring viewers to relate information across multiple datasets. While we primarily tested a limited number of datasets, to facilitate three- or four-way comparisons across multiple datasets, additional mechanisms would be needed to include and remove graphs to be compared.

8 DISCUSSION

We briefly present potential SF-LG interaction techniques, use cases, as well as limitations and future work.

8.1 SF-LG interaction techniques

Through a few scenarios, we briefly present interaction techniques with SF-LG that resolve challenges with either the technique itself or with current ways of interacting with line graphs.

Details-on-Demand: A user wants to view their heart rate at a certain point during their workout, which is one of the most basic smartwatch queries. To interact with their heart rate line graph, the user taps an area on the line graph. Additional charts, such as details of the simplification, or related bar or pie charts, fade-in to fill the largest spaces available. This approach can also be used to alleviate fat-finger interactions on a touch display [38], by providing the call-out of the details in a space made available by the SF-LG technique (Figure 13.a-c).

Multi-Level Details: A user receives a weather warning notification, and tries to learn what the warning is about. The notification appears first as an icon in an available space. A tap on the notification icon opens a layer of details, into the largest area. Such multi-level details could be applied to other forms of information, such as receiving an email notification, and then viewing the content of the email either in an overlay or a call-out around the line graph (Figure 13.d-e).

Comparative Analysis: A user wakes up and views their sleep quality data. They notice a spike in the data at a certain time, and would like to know what caused the spike. We implemented an interaction method that allows a user to compare two data points using the embedded data. First, a user taps on the spike to view additional data (e.g., body temperature at the time of the spike). An auxiliary graph will fill in the largest available space. Next, they tap on another point in the sleep quality graph to view their body temperature at that time. A second embedded graph fills the second largest available space. This allows them to compare data at two points (Figure 13.f).

Managing Screen Updates: An inherent challenge with the SF-LG is having to deal with changes in the view, as when a user flicks the line graph or when viewing streaming data. The challenge involves dealing with rapidly changing screen real-estate from one view to another. We addressed this concern in two steps.

In the first step, the embedded chart scales down but stays in the same position, while the central line graph is shifting out of view. This allows the embedded charts to remain in the same location within the line graph until such a point that the space is too small to display any content. As the embedded chart is shrinking in size, we only move it to the next available space if the point of interest on the line graph is still in view (Figure 13.g-i). If the selected point is no longer in view, the embedded chart disappears altogether.

8.2 Takeaway lessons

From our study results we propose several key takeaways:

- Multilinked time-series data, collected and visualized on smartwatches, could be simplified using techniques that highlight a line graph's salient features.
- Simplifying a line graph frees up space, which importantly can be used to embed additional information.

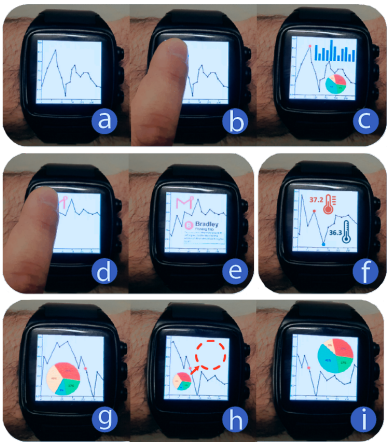

Figure 13: Scenario 1: (a) To obtain details-on-demand, (b) the user clicks a point in SF-LG. (c) This opens a callout in the most available region with details relevant to that point. Scenario 2: (d) A notification icon appears in the second most available region, and clicking it (e) opens details. Scenario 3: (f) Comparative analysis between two points on the line graph is possible. Scenario 4: (g) With streaming data, an embedded chart first (h) shrinks to the smallest size, before (i) moving to a region with sufficient space. This animated movement maintains visual consistency during the operation.

- To present multiple charts that are linked to a line graph, having them on one view (via efficient space-filling) could yield better accuracy and response times than representing them on separate screens.

8.3 Limitations

Here, we present the limitations of the SF-LG approach and of our studies. First, we did not vary the simplification levels. We simplified PIP and SF-LG to a fixed number of points. As one potential technique to find the optimum window size, a preprocessing step could be added to SF-LG to calculate the number of high fluctuations for different window sizes. In a simplified line graph, if the difference of y-values of two consecutive data points is above a threshold (proportional to the range of all possible y-values) this can be considered as a height fluctuation. The window size with the minimum number of high fluctuations could be picked. We could have included such a factor in a larger study, which was beyond the focus of this initial work. Another limitation of our studies was the use of only a rectangular screen. While we believe SF-LG can be adapted to other display form factors (i.e. circular), we leave this for future work.

One of the advantages of SF-LG over PIP is that we can use it as a real-time algorithm for streaming data. However, we did not evaluate this element of SF-LG in practice. There are potential limitations of using SF-LG for streaming data. Depending on the shape of the simplified line graph, representing the streaming data, the scale and the location of auxiliary information could change continuously. We however demonstrate (described in SF-LG Interaction

Techniques) how we can handle the continuously changing space available to embed graphs. With smooth animations and transitions, we believe we can alleviate these side effects of the space-filling method. Also, to prevent continuous quick location changing of the additional content, which could add cognitive load, a time threshold could be added to maintain the location of auxiliary information for a specific minimum period of time.

Preventing extreme fluctuations in the line graph provides us with more suitable available spaces to add information to the line graph. Given that this is one of the main goals of SF-LG, for datasets that have periodic increasing and decreasing patterns in the data, if the size of windows in SF-LG is equal to periodic behavior of the data, this will cause a jagged simplified graph which is not suitable to present additional information, similar to PIP. This issue can be fixed by changing the size of the windows.

Another limitation of our technique is that we did not provide labels on the axis of the auxiliary charts (e.g., bar and line chart) embedded in the main line graph. Although adding this information could help specific tasks, this may not be beneficial when the size of the supplemental chart is too small. This makes it hard for smartwatch users to read the labels and may add additional cognitive load. To solve this issue, the labels could be dynamically added to the auxiliary chart if its size is larger than a specific threshold to make sure labels are visible to the smartwatch users. Another essential factor in adding information to supplemental charts is the density and the number of data points in these charts. For instance, if a pie chart has only a few segments (e.g., two segments), depending on the size of the auxiliary pie chart, we can add values to the segments of this additional chart.

Choosing a suitable task, as we did for Study 2, is challenging. Our primary aim was to find a complex visual query, that users often reported needing based on prior work [3, 32]. However, our choice of task resulted in a relatively low accuracy rate in Study 2. This is not surprising given that the task (i) required users to make a complex decision; (ii) involved viewing and comparing content across multiple small views; (iii) involved limited familiarity with these queries. However, simplifying the query could have resulted in floor effects, where we may not have observed differences among conditions. A suitable balance between task complexity and ecological relevance will need consideration for future tasks.

In future work, we intend on extending the Space-Filling techniques for glancing at more than one graph. Techniques such as Braided Charts [18] offer inspiration on how to handle multiple time-series graphs in one view. We plan to explore the effect of simplifying charts that include multiple time-series data. In such cases the simplification could be applied to one, multiple, or even all of the data sets in the graph based on the details needed by the users.

9 CONCLUSION

Line graphs, common on smartwatch applications, are one of the most frequently used visualizations to represent large time-series data. We presented SF-LG, a two-step technique, to make effective use of the smartwatch display space to assist users with in-situ visual queries. In the first step, SF-LG simplified a line graph, by focusing on salient features. We found that users can efficiently perceive simplified graphs to address basic queries. Furthermore, the simplification frees up space, allowing us in a second step to embed additional content around the line graph. In a second study, we found that embedding content around a simplified line graph can facilitate visual queries involving interlinked datasets. Finally, we introduced interaction techniques that benefit from the SF-LG approach and proposed methods to improve in-situ analytics, directly on a smartwatch display.

ACKNOWLEDGMENTS

We would like to thank all of our participants. We acknowledge funding from the CRC program and the NSERC Discovery Grant to the last author. The authors in this paper are partially supported by the following grants: ANR JCJC PERFIN grant (ANR-18-CE33-0009) and PICS-CNRS (PICS07635).

REFERENCES

[1] Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, and Christian Tominski. 2011. Survey of visualization techniques. In *Visualization of Time-Oriented Data*. Springer, 147–254.

[2] Reem Albaghli and Kenneth M Anderson. 2016. A vision for heart rate health through wearables. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*. 1101–1105.

[3] Fereshteh Amini, Khalad Hasan, Andrea Bunt, and Pourang Irani. 2017. Data representations for in-situ exploration of health and fitness data. In *Proceedings of the 11th EAI International Conference on Pervasive Computing Technologies for Healthcare*. 163–172.

[4] Karl Johan Åström. 1969. On the choice of sampling rates in parametric identification of time series. *Information Sciences* 1, 3 (1969), 273–278.

[5] Tanja Blascheck, Lonni Besançon, Anastasia Bezerianos, Bongshin Lee, and Petra Isenberg. 2018. Glanceable visualization: Studies of data comparison performance on smartwatches. *IEEE transactions on visualization and computer graphics* 25, 1 (2018), 630–640.

[6] Fu-Lai Chung, Tak-Chung Fu, Robert Luk, Vincent Ng, et al. 2001. Flexible time series pattern matching based on perceptually important points. (2001).

[7] Sunny Consolvo, Beverly Harrison, Ian Smith, Mike Y Chen, Katherine Everitt, Jon Froehlich, and James A Landay. 2007. Conducting in situ evaluations for and with ubiquitous computing technologies. *International Journal of Human-Computer Interaction* 22, 1-2 (2007), 103–118.

[8] Frederick E Croxton and Harold Stein. 1932. Graphic comparisons by bars, squares, circles, and cubes. *J. Amer. Statist. Assoc.* 27, 177 (1932), 54–60.

[9] Frederick E Croxton and Roy E Stryker. 1927. Bar charts versus circle diagrams. *J. Amer. Statist. Assoc.* 22, 160 (1927), 473–482.

[10] Walter Crosby Eells. 1926. The relative merits of circles and bars for representing component parts. *J. Amer. Statist. Assoc.* 21, 154 (1926), 119–132.

[11] Andrey Esakia, D Scott McCrickard, Samantha Harden, and Michael Horning. 2018. FitAware: Promoting Group Fitness Awareness ThroughGlanceable Smartwatches. In *Proceedings of the 2018 ACM Conference on Supporting Groupwork*. 178–183.

[12] Paolo Federico, Stephan Hoffmann, Alexander Rind, Wolfgang Aigner, and Silvia Miksch. 2014. Qualizon graphs: Space-efficient time-series visualization with qualitative abstractions. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces*. 273–280.

[13] Tak-chung Fu, Fu-lai Chung, Ka-yan Kwok, and Chak-man Ng. 2008. Stock time series visualization based on data point importance. *Engineering Applications of Artificial Intelligence* 21, 8 (2008), 1217–1232.

[14] Johannes Fuchs, Fabian Fischer, Florian Mansmann, Enrico Bertini, and Petra Isenberg. 2013. Evaluation of alternative glyph designs for time series data in a small multiple setting. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 3237–3246.

[15] Xiaonan Guo, Jian Liu, and Yingying Chen. 2017. FitCoach: Virtual fitness coach empowered by wearable mobile devices. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 1–9.

[16] Jeffrey Heer and Michael Bostock. 2010. Crowdsourcing graphical perception: using mechanical turk to assess visualization design. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 203–212.

[17] Alaul Islam, Anastasia Bezerianos, Bongshin Lee, Tanja Blascheck, and Petra Isenberg. 2020. Visualizing information on watch faces: A survey with smartwatch users. *arXiv preprint arXiv:2009.00750* (2020).

[18] Waqas Javed, Bryan McDonnel, and Niklas Elmqvist. 2010. Graphical perception of multiple time series. *IEEE transactions on visualization and computer graphics* 16, 6 (2010), 927–934.
[19] Brian Johnson and Ben Shneiderman. 1999. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. *Readings in Information Visualization: Using Vision to Think* (1999), 152–159.
[20] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. 2001. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems* 3, 3 (2001), 263–286.
[21] Bongshin Lee, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete, and Nathalie Henry. 2006. Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization*. 1–5.
[22] Mark A Livingston, Jonathan W Decker, and Zhuming Ai. 2012. Evaluation of multivariate visualization on a multivariate task. *IEEE transactions on visualization and computer graphics* 18, 12 (2012), 2114–2121.
[23] Ben Lovejoy. 2006. Stats and Analysis. Retrieved June 7, 2006 from http://www.poker-edge.com/stats.php
[24] Ali Neshati, Yumiko Sakamoto, Launa Leboe-Mcgowan, Jason Leboe-McGowan, Marcos Serrano, and Pourang Irani. 2019. G-sparks: Glanceable sparklines on smartwatches. In *45th Conference on Graphics Interface (GI 2019)*. 1–9.
[25] Jonas Parnow and Marian Dörk. 2015. Micro visualizations. *Diss. Master's thesis, Potsdam Univ. Appl. Sci., Potsdam, Germany* (2015).
[26] Charles Perin, Frédéric Vernier, and Jean-Daniel Fekete. 2013. Interactive horizon graphs: Improving the compact visualization of multiple time series. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 3217–3226.
[27] Lewis V Peterson and Wilbur Schramm. 1954. How accurately are different kinds of graphs read? *Audio Visual Communication Review* (1954), 178–189.
[28] George Robertson, Roland Fernandez, Danyel Fisher, Bongshin Lee, and John Stasko. 2008. Effectiveness of animation in trend visualization. *IEEE transactions on visualization and computer graphics* 14, 6 (2008), 1325–1332.
[29] Paul Rosen and Ghulam Jilani Quadri. 2020. LineSmooth: An Analytical Framework for Evaluating the Effectiveness of Smoothing Techniques on Line Charts. *IEEE Transactions on Visualization and Computer Graphics* (2020).
[30] Gabriel Ryan, Abigail Mosca, Remco Chang, and Eugene Wu. 2018. At a glance: Pixel approximate entropy as a measure of line chart complexity. *IEEE transactions on visualization and computer graphics* 25, 1 (2018), 872–881.
[31] Takafumi Saito, Hiroko Nakamura Miyamura, Mitsuyoshi Yamamoto, Hiroki Saito, Yuka Hoshiya, and Takumi Kaseda. 2005. Two-tone pseudo coloring: Compact visualization for one-dimensional data. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.* IEEE, 173–180.
[32] Ben Shneiderman. 1994. Dynamic queries for visual information seeking. *IEEE software* 11, 6 (1994), 70–77.
[33] John Stasko, Richard Catrambone, Mark Guzdial, and Kevin McDonald. 2000. An evaluation of space-filling information visualizations for depicting hierarchical structures. *International journal of human-computer studies* 53, 5 (2000), 663–694.
[34] Ashley Suh, Christopher Salgado, Mustafa Hajij, and Paul Rosen. 2019. TopoLines: Topological Smoothing for Line Charts. *arXiv preprint arXiv:1906.09457* (2019).
[35] Saiganesh Swaminathan, Raymond Fok, Fanglin Chen, Ting-Hao Huang, Irene Lin, Rohan Jadvani, Walter S Lasecki, and Jeffrey P Bigham. 2017. Wearmail: On-the-go access to information in your email with a privacy-preserving human computation workflow. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. 807–815.
[36] Edward Tufte. 2006. R.(2006) Beautiful evidence.
[37] Jarke J Van Wijk and Huub Van de Wetering. 1999. Cushion treemaps: Visualization of hierarchical information. In *Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis' 99)*. IEEE, 73–78.
[38] Daniel Vogel and Patrick Baudisch. 2007. Shift: a technique for operating pen-based interfaces using touch. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 657–666.
[39] Martin Wattenberg. 1999. Visualizing the stock market. In *CHI'99 extended abstracts on Human factors in computing systems*. 188–189.
[40] Martin Wattenberg. 2005. A note on space-filling visualizations and space-filling curves. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.* IEEE, 181–186.
[41] Marc Weber, Marc Alexa, and Wolfgang Müller. 2001. Visualizing time-series on spirals.. In *Infovis*, Vol. 1. 7–14.
[42] Qinge Wu, Kelli Sum, and Dan Nathan-Roberts. 2016. How fitness trackers facilitate health behavior change. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 60. SAGE Publications Sage CA: Los Angeles, CA, 1068–1072.

A APPENDIX: LIMITATION OF SF-LG

The limitation of SF-LG has been illustrated in Figure 14. If the data set has very high and very low values periodically, and if this periodic behaviour is equal to windows size, this will result in a jagged simplified line graph.

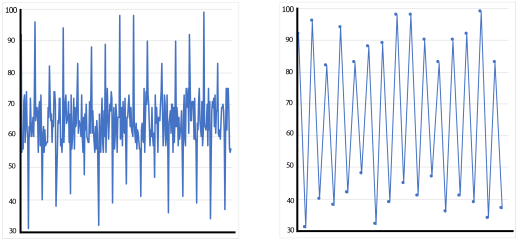

Figure 14: (Left) A non-simplified line graph with 300 data points with periodic high and low values in the data. (Right) Simplified line graph using SF-LG technique result in numerous frequent high fluctuations preventing app designers for adding auxiliary information.