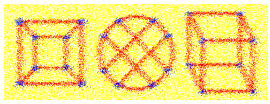


Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 16, no. 1, pp. 109–126 (2012)

Drawing Graphs on a Smartphone

Giordano Da Lozzo[1] *Giuseppe Di Battista*[1] *Francesco Ingrassia*[1]

[1]Dipartimento di Informatica e Automazione,
Università Roma Tre, Italy

Abstract

We present a system for the visualization of information modeled in terms of a graph on a smartphone. First, we show the adopted visualization paradigm, that allows the user to navigate the graph using a focus-based approach. Second, we tackle the algorithmic challenges posed by the new visualization paradigm, introducing and experimenting effective heuristics. Finally, we show several customizations of the system aimed at exploring and at visualizing popular Web contents like social networks and Wikipedia. Current implementations include the iPhone and the Google Android platforms.

Submitted:	Reviewed:	Revised:	Accepted:
January 2011	July 2011	November 2011	November 2011
	Final: November 2011	Published: January 2012	
Article type: Regular paper		Communicated by: U. Brandes and S. Cornelsen	

Research supported in part by MIUR (Italy), Projects AlgoDEEP no. 2008TFBWL4 and FIRB "Advanced tracking system in intermodal freight transportation", no. RBIP06BZW8.
E-mail addresses: dalozzo@dia.uniroma3.it (Giordano Da Lozzo) gdb@dia.uniroma3.it (Giuseppe Di Battista) fra.ingrassia@gmail.com (Francesco Ingrassia)

1 Introduction

Millions of people in the world have in their pocket a smartphone and such a number is rapidly increasing [3]. Such a widely used device is exploited to quickly access, from almost everywhere, different types of data on different subjects and a large amount of such data is relational information. For example, smartphones are used to access social networks like Facebook or Twitter, ontologies like the Wikipedia network of concepts, or technical information related to the job of the smartphone's owner like the connections of a computer network or the delivery routes of a product distribution system.

Graph Drawing can play an important role in supporting information visualization on the smartphones, provided that the methodologies and the tools that are typical of this research area are recast to meet the needs of such a challenging device. Indeed, different information contexts have already changed their visualization methods in this direction. For instance, on-line newspapers have special visualization formats that are designed for the smartphone. However, as far as we know, the only previous attempt to draw graphs on smartphones [8] uses traditional Graph Drawing techniques.

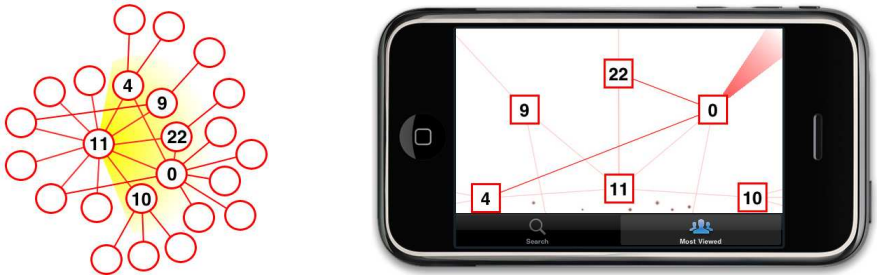
Dealing with smartphones, the main challenge that visualization applications have to face is, of course, the small screen size. On the other hand, such a strong limitation comes together with new technological opportunities that can be exploited to support the interaction. They are the multi-touch screen that is able to capture commonly used gestures like pinch, flick, and slide, sensors like the accelerometer and the compass, and sounds and vibrations.

Since any graph is too large for the little screen of the smartphones, a pivotal reference point for designing interfaces and algorithms for drawing graphs on such a device is the literature on drawing very large graphs. One, for example, could use the fish-eye approach [9] where the details of the drawing decrease according to the distance that separates them from a point chosen by the user. However, using Shneiderman's information visualization mantra [10] (overview first, zoom and filter, then details-on-demand) in this context seems to be unfeasible. In fact, it is unclear how to provide, on such a small screen, a suitable overview of the information.

In this paper we present a system for the visualization and navigation of relational information on the smartphones. Section 2 illustrates the visualization and interaction paradigm we devised, that (i) is based on showing to the user only a small subgraph defined by a focus vertex and its neighborhood and (ii) exploits smartphone-specific interaction primitives to explore the graph. The paradigm is inspired by the navigation approach of [2]. Section 3 shows how even such a simple visualization paradigm can originate interesting algorithmic issues. Section 4 presents the Graph Drawing algorithms that we adopted and implemented into the system. Section 5 discusses experimental results that put in evidence the effectiveness of such algorithms. Section 6 gives technical details on the implementation and presents challenging case studies.

2 A Visualization and Interaction Paradigm

The paradigm is based on a navigation approach. The user selects a *focus vertex* v of the graph of interest. This is done with some selection criteria that depends on the application context. Let $N(v)$ be the set of neighbors of v, that is assumed circularly ordered, the drawing contains vertices and edges as follows (Fig. 1 shows an example of visualization).



(a) Focus vertex 11 with lobe $\{4, 9, 22, 0, 10\}$.

(b) Visualization of the focus vertex and of a lobe on a smartphone.

Figure 1: An example of visualization.

Vertices: the drawing contains v and a subset ω_i^L, called *lobe*, of $N(v)$, where L (*lobe size*), is the size of the lobe and the elements of ω_i^L have consecutive positions $i, \ldots, (i + L - 1) \mod |N(v)|$ in $N(v)$.

Edges: the drawing contains the edges from v to the vertices of ω_i^L (*radial* edges), the edges between vertices of ω_i^L (*inner* edges), the edges that have one end-vertex in ω_i^L and one end-vertex in $N(v) \setminus \omega_i^L$ (*outer* edges), and the edges that have one end-vertex in ω_i^L and one end-vertex that is not in $N(v)$ (*external* edges).

The focus vertex is placed in the center of the bottom side of the drawing, while the vertices of the lobe lie on an half-ellipse centered at the focus vertex. This pattern, together with other graphical features, suggests to the user that ω_i^L is only a subset of $N(v)$ and that the vertices of $N(v) \setminus \omega_i^L$ (*outer* vertices) are under the bottom side of the screen. This approach both allows to deal with large degree focus vertices and supports visualization of very large graphs. In fact vertices whose distance from the focus vertex is bigger than 3 do not enter the screen. Therefore those vertices and their incident edges can be loaded on demand without the need for the whole graph to be stored in the main memory of the device.

External edges exit the drawing from the left, top, right borders of the screen, giving the impression that the rest of the graph is outside the screen. Their external end-vertices are not represented. Outer edges are directed downward, leading to vertices of $N(v)$ that are not represented and that "are under the bottom side of the screen".

Different types of edges have different thicknesses. Radial edges, that do not give additional information, are thin, while inner edges, that represent relationships between vertices of the lobe, are emphasized. External edges could be too many to be represented explicitly, hence they are drawn with straight-lines only up to a certain number. Over that number they are shown with a shadow exiting the drawing (see vertex 0 of Fig. 1). Essentially, the information given to the user is if they are 0, a few, or a lot. Similar graphic features are used for outer edges.

There are several possibilities to help the user in perceiving the number of outer vertices. In iPhone implementations (see Fig. 1) we did the choice of adding to the bottom of the screen a cloud of small points whose number is a function of $|N(v) \setminus \omega_i^L|$. In Android applications (see Fig. 8(b)) we surrounded the focus vertex with a half ellipsis again with size proportional to $|N(v) \setminus \omega_i^L|$.

The user interacts with the graph according to the following *primitives*. There are *vertex oriented primitives* like focus change, navigation backtrack, and execution of actions associated with vertices (e.g., open a browser or send an sms) and *lobe oriented primitives* like lobe shift, lobe resize, and lobe layout temporary optimization.

The *focus change* primitive substitutes the currently visualized subgraph with the subgraph induced by a new focus vertex and its neighborhood. See Figs. 2(a) and 2(b). The gesture used to change the focus consists of dragging the vertex of the lobe that will be the new focus towards the focus position.

(a) A focus change starts.

(b) Focus change result.

Figure 2: Usage of the focus change primitive.

The *backtrack* primitive shows the previously displayed lobe. The user can navigate the sequence of the explored focus vertices by performing a double fingers flick towards the bottom or top side of the screen. A side feature of our framework is the capability of extending the interaction experience by executing an action related to a selected vertex. For example, if the input graph is a social network, possible actions are: showing more information about the individual, sending an e-mail or an sms, deleting him/her from the graph, etc. The gesture associated with this action is a double tapping or a pressure on the vertex.

A primitive that changes the lobe is *lobe shift*. Let ω_i^L be the visualized lobe, the effect of lobe shifting is to substitute ω_i^L with ω_k^L where $k = i \pm 1 \mod |N(v)|$. Sliding a finger on the screen is the natural way to do this.

However, if $|N(v)|$ is very large this interaction can be unsuitable to reach vertices of $N(v)$ that are far from the current lobe. Hence, the user needs a single gesture to skip a large portion of the subgraph. This is obtained with a flick of finger on the screen. The number of the vertices that are skipped is proportional to the speed of the gesture. Alternatively, the same primitive can be invoked using the accelerometer by changing the slope of the device or using the compass by changing the orientation of the device.

Zooming is a quite common functionality offered by smartphone applications. In our paradigm this function corresponds to a *lobe resize*. Let ω_i^L be the visualized lobe, the effect of this function is to show the lobe ω_k^M where $k = i$ if $(M = L) \vee (M = L + 1)$ or $k = i - 1 \mod |N(v)|$ if $M = L + 2$. The gesture used to resize the current lobe is a multi-touch gesture called pinch, this consists of moving together two fingers on the screen, increasing or decreasing their distance. Although in our implementations (see Par. 6) the size of vertices does not change when the lobe size increases or decreases, it's likely that some application contexts could also get benefit from scalable representations of the vertices.

The current order of the vertices of a lobe could lead to a drawing that is unpleasant. Hence, we provide a *local optimization* feature that temporary reorders the vertices of the lobe in such a way to increase the readability of the drawing. This local optimization is invoked by performing a double tap on the screen.

In order to help the user in maintaining the mental map during the navigation, a mechanism for morphing between successive drawings, supported by a smooth movement of edges and vertices, is provided. Hence, for all primitives, vertices move slowly towards their final positions. In lobe oriented primitives, vertices entering (exiting) the current lobe come in (out) from the bottom side of the screen. Also, inner edges can become external or outer and vice-versa, and their representation features change coherently. The flick function is accompanied by an inertial rotation of the neighborhood of the focus vertex. The morphing features of the focus change primitive are more complex than those described above. Since they depend on the order of the lobes, they are discussed in Section 4. With the purpose of helping the user interaction experience, sounds and vibration events are associated with the primitives. They are selected in such a way to be consistent with the effects of the corresponding primitives.

3 Choosing a Lobe Order

A central aspect of the visualization paradigm is the left-to-right order of the vertices of the lobe. The specific choice of this order may depend on the specific application. However, the need of preserving the user's mental map implies that the orderings of contiguous lobes of a focus vertex v are consistent. Since this constraint holds for all the lobes of v, this implies the need of choosing a unique order for all the vertices of $N(v)$.

Trivial choices are possible, like using the alphabetical order, that can be

suitable for some applications. On the other hand, it is also possible to make different choices, according to aesthetics that are related to the selected visualization paradigm. We deepen two alternatives corresponding to two aspects of the paradigm. The first is the one of displaying the relational information as clean as possible, while the second is the one of showing as much relational information as possible. The two choices conflict each other.

The first choice is the one of selecting an order that tries to minimize the visible crossings. Given a focus vertex and a lobe, a *visible crossing* is a crossing between two inner edges. We concentrate only on visible crossings because the only edges that are completely visible for a certain lobe are the inner edges and the radial edges. Also, crossings with radial edges do not compromise readability. More precisely, we try to minimize the average number of visible crossings for all the lobes of a focus vertex.

The problem of minimizing the visible crossings has similarities with a classical Graph Drawing problem, called *circular crossings minimization* problem. In that problem, the vertices lie on a circumference, the edges are straight line segments, and a circular order is searched that minimizes the total number of crossings. See, e.g. [6, 11, 1, 4]. However, as shown in Fig. 3, the two problems are different. Namely, an ordering that minimizes the circular crossing number not necessarily minimizes the number of visible crossings for a given lobe size. The converse is also true.

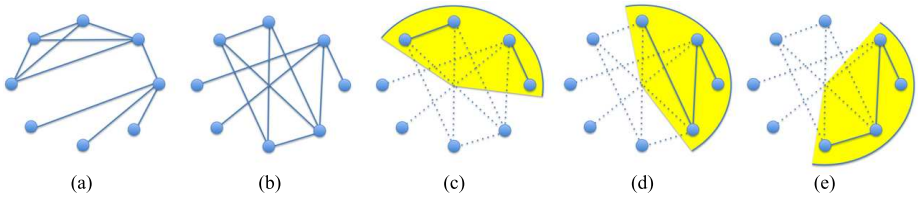


Figure 3: Circular crossings and visible crossings of $N(v)$. (a) Order with minimum total number of crossings. (b) Order with minimum number of visible crossings if the lobe size is equal to 4. (c)–(e) Lobes with 0 visible crossings.

Observe that minimizing the visible crossings with a lobe whose size equals $|N(v)|$ is equivalent to solving the circular crossings minimization problem for the subgraph induced by $N(v)$. Unfortunately, such a problem has been proved to be NP-complete [7]. Hence, solving in practice the minimization of the visible crossings requires the usage of heuristics.

Even if the two problems are different, one may ask whether heuristics for the circular crossings minimization are effective also for our problem. Our experiments show that this is not the case (see Section 5). Hence, we used in our system a special purpose algorithm that is described in Section 4.

A second alternative for choosing an order is to select one that tries to maximize the *visible edges*, that is to minimize the number of edges that are not inner edges in any lobe. This corresponds to minimizing the information that is lost for a certain focus vertex. Let L be the lobe size and let v be the

Algorithm 1 Lobe Ordering Algorithmic Framework

$N \leftarrow |N(v)|$
$S \leftarrow N(v)$
while $S \neq \emptyset$ **do**
  $Nexts \leftarrow$ select a start vertex $s \in S$
  while $Nexts \neq \emptyset$ **do**
    1. extract a vertex $u \in Nexts$ according to the adopted processing policy
    2. $S \leftarrow S \setminus \{u\}$
    3. assign priority $p_u(x)$ to each position $x \in \{0, ..., N-1\}$
    4. place u at position x so that $\min_x p_u(x)$
    5. $Nexts \leftarrow Nexts \cup \{$unplaced vertices of $N(u) \cap N(v)\}$
  end while
end while

focus vertex. Observe that it might not exist an order for $N(v)$ in which every edge with end-vertices in $N(v)$ is an inner edge for at least one lobe of size L. As an example, consider the case when the subgraph induced by $N(v)$ contains the clique K_M with $M \geq 2L$. Of course, this information is not lost. In fact, such edges will be visible selecting one of the end-vertices of the missing edges as focus vertex.

The visible edges problem has been studied (with the name *circular bandwidth problem*) and shown NP-complete in [5]. Hence, even for this second problem the usage of heuristics is required.

4 Algorithmic Framework

The algorithmic framework that we have developed in our system is inspired by the algorithms for the circular crossings minimization problem presented in [6, 1, 4] and allows to tackle both the visible crossings minimization and the visible edges maximization problems. The typical parameters of the algorithms in [6, 1, 4] are:

1. a start vertex s of the graph to be drawn;
2. the policy which selects the next vertex to process, and
3. the position that is chosen for the selected vertex.

We make use of the same general setting. The algorithmic framework we propose is Algorithm 1. Notice that Algorithm 1 refers to a vertex processing policy that is not specified. The main processing policies used in the literature for determining the insertion sequence are: *Random,* vertices are processed in random order; *Maximum degree,* at each step, a vertex with the largest number of neighbors is placed; *Minimum degree,* at each step, a vertex with the least

number of neighbors is placed; and *Connectivity,* at each step, a vertex with the largest number of already placed neighbors and (in case of ties) the least number of unplaced neighbors is selected. In our experiments we test the effectiveness of each of them.

Observe that the inner cycle of Algorithm 1 involves vertices belonging to the same connected component. Indeed, inside this cycle, a breadth-first scanning of the connected component is performed. The outer cycle is used to iterate the inner one over all the connected components of the subgraph induced by $N(v)$, until all vertices are placed.

In order to evaluate the impact (in terms of the qualities of interest) of the placement of the selected vertex in each available position of the current layout, with respect to a lobe size L, we introduce a real valued *priority function.* The function $p_u(x) : \{0 \dots N-1\} \to \mathbb{R}$ estimates the quality of the position x for vertex u considering the *cost* associated with the edges that link u to its already placed neighborhood $N_p(u)$. This function, in turn, relies on a *cost function*

$$c_k(d, L) : \{1, \dots, \lfloor N/2 \rfloor\} \times \{1, \dots, N\} \to \mathbb{R}$$

whose purpose is to estimate the burden produced by an edge of length d in a layout with lobe size L.

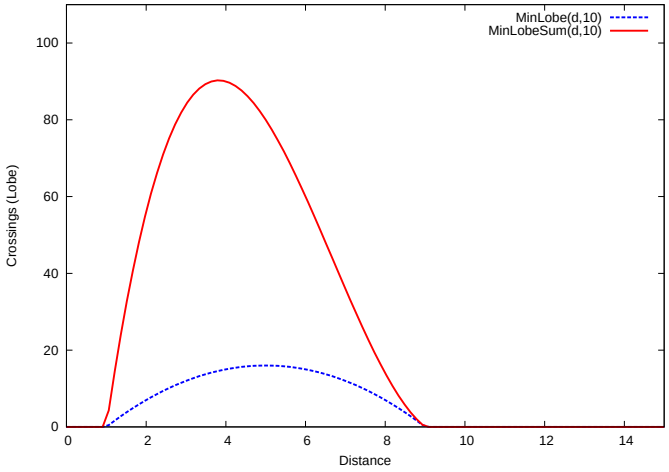
Let s and t be two positions on the circular sequence, function $length(s,t) = min(|s-t|, N-|s-t|)$ computes the distance between s and t. Also, let $\pi(w)$ denote the position of a vertex w. We define:

$$p_u(x) = \begin{cases} \sum_{w \in N_p(u)} c_k(length(x, \pi(w)), L) & \text{if x is available} \\ \\ \infty & otherwise \end{cases}$$

Different cost functions are possible. Fig. 4 shows two possible cost functions aimed at minimizing the number of crossings between inner edges. The former function $c_L(d, L)$ (*MinLobe*) estimates the cost of an edge of length d in a lobe of size L as the maximum number of visible crossings this edge can produce if there were (in the lobe) all the edges that are able to cross it. The latter function $c_{LS}(d, L)$ (*MinLobeSum*) multiplies the cost estimated by the previous function by the number of lobes the edge will appear in. Observe that none of the two functions implements a policy of type "the shorter a new edge the better."

Actually, the current implementation of the algorithm associates with each location a list of priority values that are sequentially evaluated. This feature makes it possible to choose the final position of a vertex using multiple selection criteria (or, equivalently, a *multi-objective function*), whose relative importance is defined by the order they are taken into account.

If we want to maximize the visible edges, a simple cost function whose purpose is to maximize the number of edges that can become inner edges in some lobe is function $c_I(d, L)$ (*MaxVisibleEdge*), defined as follows: $c_I(d, L) = -1$ if $d < L$ and $c_I(d, L) = 0$ if $d \geq L$.



$$c_L(d,L) = \begin{cases} (d-1)(L-d-1) & d \geq 1 \wedge d < L \\ \\ 0 & otherwise \end{cases}$$

$$c_{LS}(d,L) = \begin{cases} c_L(d,L)(L-d) & d \geq 1 \wedge d < L \\ \\ 0 & otherwise \end{cases}$$

Figure 4: Cost functions for minimizing visible crossings.

If we find that two positions have the same priority, we break the tie using a cost function called *MinEdgeLength*, that is $c_D(d,L) = d \; \forall d \leq \lfloor N/2 \rfloor, \forall L \leq \lfloor N/2 \rfloor + 1$. It is used to avoid increasing too much the distance between currently used positions. This choice has two major benefits: (i) it produces fewer circular crossings than a random placement and (ii) it makes it easier to reveal hidden information by means of the zoom function.

Crossings among inner and outer edges are kept low by using as second level priority function the MinEdgeLength cost function for all algorithms.

As observed in Section 2, special attention is needed to perform an effective morphing procedure for the focus change primitive from the current focus v to the new focus u. This has important effects also in the algorithmic framework. Let ω and θ be the visible lobes of v and u, respectively.

It is essential, to preserve the user mental map, to order the vertices in $N(u)$ such that the relative order of the vertices in $N(u) \cap \omega$ remains the same of that of ω. Also, since a lobe has a limited size and since $|N(u)|$ can be much larger than the lobe size, there are degrees of freedom in selecting the vertices that enter θ. From this point of view, it is important to place in θ at least some

vertices that help the user not to loose the context. Let w_1 (w_2) be the first vertex of ω encountered moving in ω from u to its left (right) and belonging to $N(u)$. Observe that one or both w_1 and w_2 might not exist. If they exist, θ must contain at least one of w_1 and w_2.

Two cases are possible: either $|N(u) \cap \omega| + |N(u) \setminus N(v)| \leq L$ or not.

In the first case all the external edges that enter the drawing can become radial edges (the lobe size is large enough). All their end vertices are placed in θ after w_1 and/or before w_2, while possible free positions in θ are assigned to the remaining vertices in $N(u)$.

In the second case only a subset of the external edges entering the drawing can become radial edges (the lobe cannot host, indeed, all their end vertices). In order to keep at least one of w_1 and w_2, the number of external vertices of u entering θ must not exceed $L - min(2, |N(u) \cap \omega|)$. Again, the selected external vertices of u are placed in θ after w_1 and/or before w_2.

Notice that the above algorithmic approach, since the positions are explicitly represented, easily integrates constraints related to the lobe ordering and composition discussed above.

In order to give an impression of continuity, morphing from ω to θ, vertices move as follows. The selected external vertices come from the top side of the screen during the animation associated to the primitive, towards their final position. Context vertices move towards the lobe borders. The remaining vertices move towards the bottom side of the screen or to their final positions in θ.

5 Experimental Analysis of the Algorithms

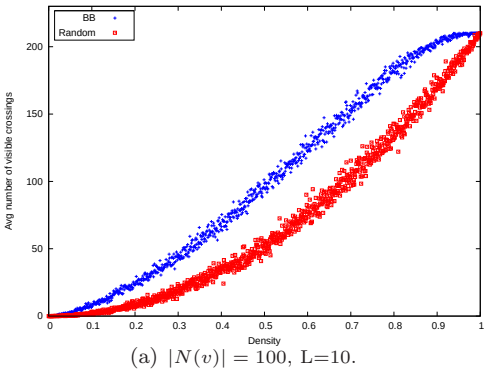
In order to assess the effectiveness of our algorithmic techniques we performed several experimental tests using a suite of randomly generated connected graphs, where each graph represents the subgraph induced by $N(v)$ for some choice of focus vertex v. For each test we generated an average of $5,000$ graphs according to two modalities:

1. fixed number of vertices (100) and variable density (from 0.1% to 100%);
2. fixed density and variable number of vertices (from 10 to 100).

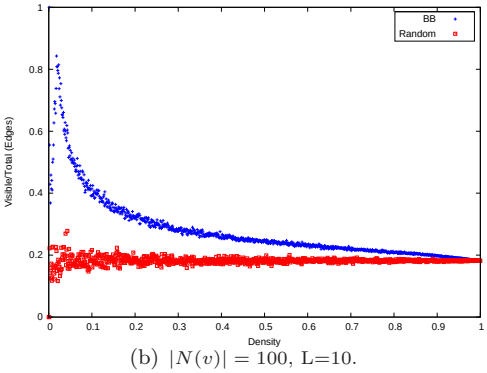
For both types of graphs we used a fixed lobe size $L = 10$. This value represents a good compromise between the need of maximizing the number of vertices of the lobe and the limited space available on the screen of a smartphone.

We evaluated the algorithms according to the following metrics:

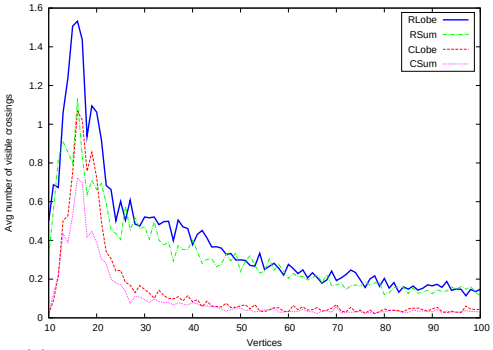
1. number of visible crossings, that is the number of crossings between inner edges;
2. visible edges ratio, that is the ratio between the number of visible edges and the total number of edges of the graph.



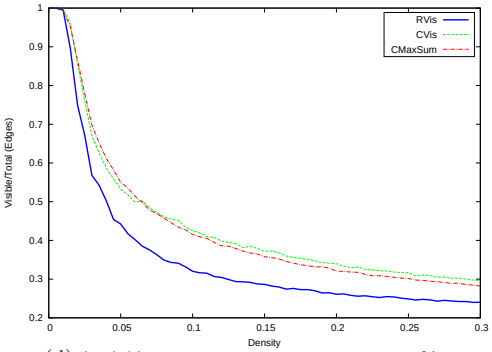
(a) $|N(v)| = 100$, L=10.



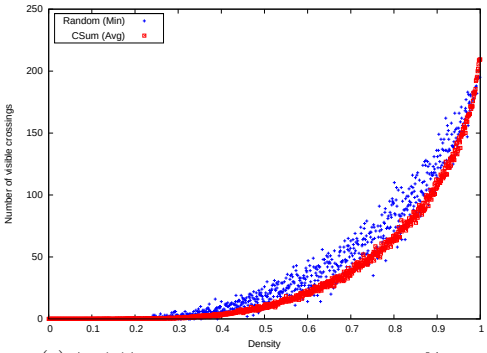
(b) $|N(v)| = 100$, L=10.



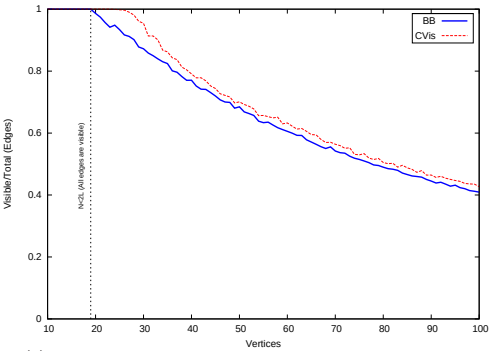
(c) $|N(v)| = \{10, ..., 100\}$, L=10, density=15%.



(d) $|N(v)| = 100$, L=10, density=$0 - 30\%$.



(e) $|N(v)| = 100$, L=10, density=$0 - 100\%$.



(f) $|N(v)| = \{10, ..., 100\}$, L=10, density=15%.

Algorithm	**Processing policy**	**Cost functions (2: MinEdgeLength)**
RSum	Random	1: MinLobeSum
RLobe	Random	1: MinLobe
CSum	Connectivity	1: MinLobeSum
CLobe	Connectivity	1: MinLobe
RVis	Random	1: MaxVisibleEdge
CVis	Connectivity	1: MaxVisibleEdge
CMaxSum	Connectivity	1: MaxSumLobe (i.e. $-c_{LS}(d, L)$)

Figure 5: Experimental results.

A selection of the obtained results is presented below.

In the experiment shown in Fig. 5(a) we compare a simple random order with the BB algorithm [1], chosen as one of the best algorithms for computing a circular layout with few crossings. In our implementation we decided not to consider the biconnected components of $N(v)$ separately, in order for BB to exploit all available positions of the layout, as it happens for the devised algorithms. Each point shows the average number (computed over all lobes) of the visible crossings for a graph whose density is reported on the x-axis. A random order produces fewer visible crossings than the BB algorithm. Such results suggest that the circular approach is not a good choice for the *visible crossings minimization problem*. The reason is that the tested circular algorithm tries to minimize the total length of the edges. However, this produces many short edges that more easily get inner edges, increasing the possibility of generating visible crossings.

In the experiment shown in Fig. 5(c) we compare several variants of our algorithm (see the table in Fig. 5). We changed the processing policy that selects the next vertex to insert (Random or Connectivity) and the cost functions for each level of priority (MinLobe or MinLobeSum). Observe that in all cases we used, for a second priority level, cost function MinEdgeLength. Mixing these features we obtained four different algorithms to evaluate (RLobe, RSum, CLobe, CSum). The graphic shows the comparison between these algorithms with a variable number of vertices and fixed density 15%. It is evident that the choice of a good processing policy has a strong influence on the number of visible crossings. Algorithms that use the connectivity policy perform better than those using the random policy. For a given processing policy, the MinLobeSum cost function is slightly better than the MinLobe.

In the experiment shown in Fig. 5(e) we compare the average number of visible crossings of the CSum algorithm with the minimum number of visible crossings obtained in all lobes using a Random placement. The graphic shows that CSum has a similar trend to Random with slightly better results. Roughly, given an unordered layout and selecting the lobe with the minimum number of visible crossings, the algorithm generates a drawing with the same average number of visible crossings for all the lobes.

In the experiment shown in Fig. 5(b) we compare the visible edges ratio between Random order and BB Algorithm. Although BB is not designed for solving this problem, drawing shorter edges decreases the number of edges hidden by the paradigm and therefore increases the visible edges ratio. In the graphic we observe that BB Algorithm has a better behaviour than the one of a Random order.

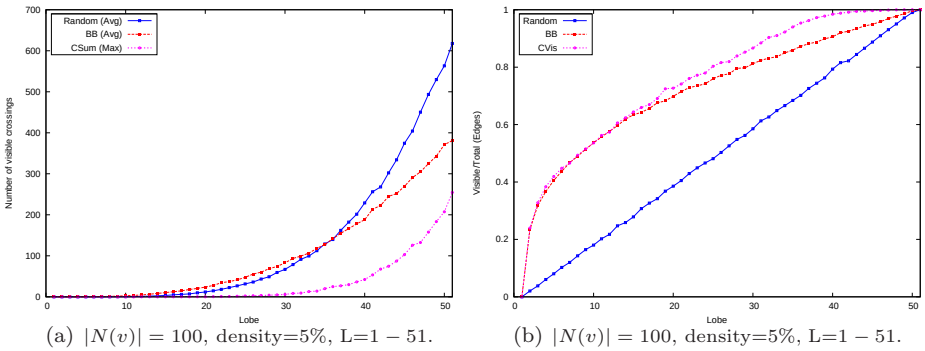
For the problem of maximizing the visible edges we follow two different approaches. The first is to reverse the cost function of CSum in order to increase the number of crossings and consequently the number of visible edges. The second approach is a greedy approach where a vertex is placed in the position that generates the highest number of visible edges. This cost function is associated with a Random and Connectivity processing policy. A Random selection decreases the effectiveness of the results of the MaxVisibleEdge cost function.

With an equal processing policy the MaxVisibleEdge function is slightly better than MaxSum. (See Fig. 5(d).)

In the experiment shown in Fig. 5(f) we compare CVis algorithm with BB algorithm. CVis has better results than BB, especially with a small number of vertices (CVis keeps visible 100% of edges for graphs with more vertices than BB). Increasing the number of vertices both algorithms have the same trend and CVis has generally an improvement of $2-3\%$.

5.1 Zoom function evaluation

Since resizing the lobe by means of the zoom function represents one of the most powerful primitives offered by the paradigm, tests have been performed in order to show the effectiveness of the proposed algorithms as the lobe size changes. The experimental analysis has been performed letting L changing within the range $[1, \lfloor N/2 \rfloor + 1]$ because when L reaches the value $\lfloor N/2 \rfloor + 1$ the problem of maximizing the visible edges becomes trivial and such a range certainly covers the range that is useful in practical applications for the visible crossing minimization problem.



(a) $|N(v)| = 100$, density=5%, L=1 − 51. (b) $|N(v)| = 100$, density=5%, L=1 − 51.

Figure 6: Zoom function evaluation.

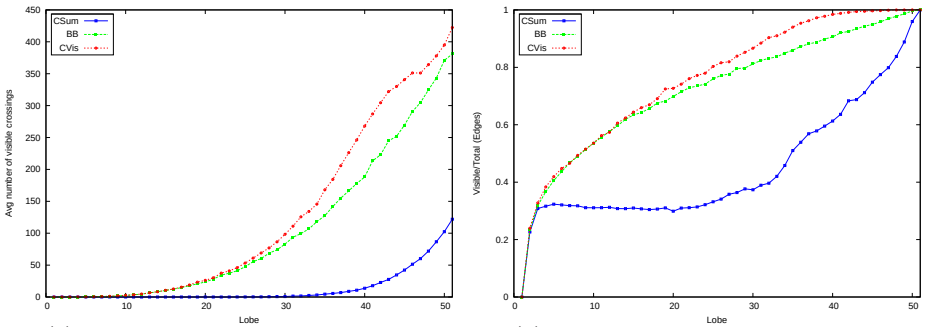
Figure 6(a) show a comparison between Random, BB, and CVis algorithms in terms of visible crossing as the lobe size changes. Notice that the behaviour of BB is such that for a large range of lobe sizes it produces an higher number of visible crossing than a Random placement of the vertices (see also Fig. 5(a)). CSum results in a maximum number of visible crossings per lobe that is always bounded above by the average number of visible crossings per lobe of both the two shown algorithms.

In Fig. 6(b) we compare the growth of the number of visible edges among Random, BB, and CVis algorithms. The uniform distribution of the edges' length causes the Random's curve to grow linearly in the lobe size (i.e. the number of edges of length k is statistically independent from k), whereas CVis follows an ideal concave behaviour. BB instead, by applying a shortest edge

length strategy, has good results for small lobe sizes but its performance decreases rapidly for larger ones.

5.2 Selecting an algorithm

An experimental comparison in terms of both visible crossings and visible edges between the main discussed algorithms is provided. Figures 7(a) and 7(b) clearly identify the CVis algorithm as the best choice for many practical contexts. In fact while showing more information than BB does, it also keeps the number of visible crossings low for those lobe sizes that the user will reasonably select during the navigation of the graph in the smartphone's screen.



(a) $|N(v)| = 100$, density=5%, L=1 − 51. (b) $|N(v)| = 100$, density=5%, L=1 − 51.

Figure 7: Visible crossings vs visible edges.

6 Implementation and Case Studies

The project has been fully driven by the experiments performed on the devices. This led to two software libraries, one for the iPhone OS 3.1.3 (Objective-C language) and the other for the Google Android 2.1 (Java language) platforms. Both the libraries were designed to have fully customizable graphical and behavioural components and to allow simple usage for the software developers.

Although the two platforms are very different, we developed the software so that both prototypes have the following common features:

1. use vector graphics;
2. build a new abstract layer over the platform to manage animations and gestures; and
3. optimize the number of operations and edge drawings needed for each display refresh.

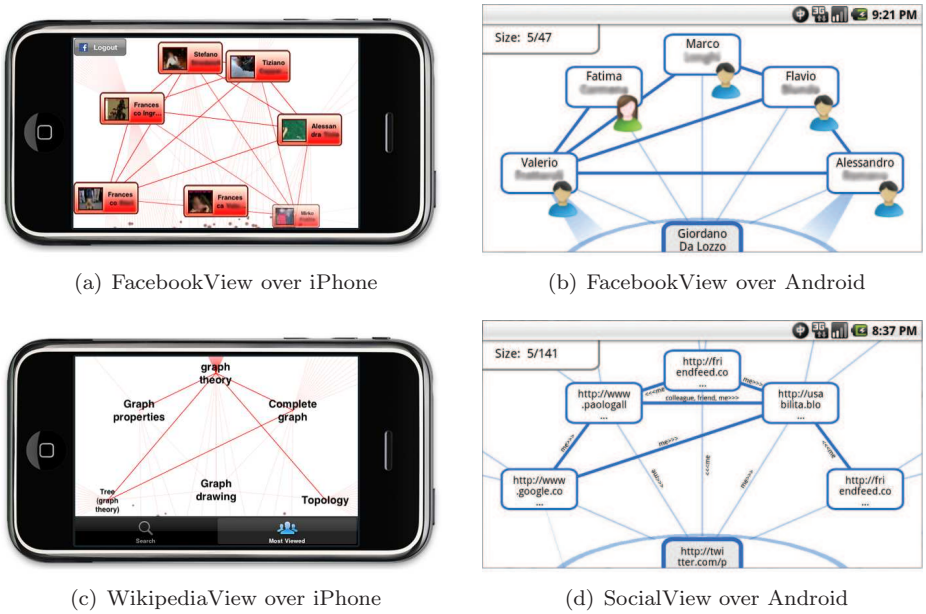
Because of the limitations of the platform, the Android release required also to develop platform-specific functions to minimize the number of refreshes.

We implemented several case studies. Two of them, that refer to the context of social networks, are especially interesting. Notice that, in this context, the effectiveness of the devised algorithms is clearly visible and appreciable in the developed applications. In fact, because of the average density of the social networks, the placement of vertices according to a random or alphabetical order, typically shows a minimal amount of inner edges. With respect to the relational information represented, even the MinLobeSum cost function (that aims at minimizing visible crossings) performs better of those orderings.

The first case study uses the *Facebook API* to determine the graph of the friendships of a Facebook user (see Fig. 8(a) and 8(b)). These APIs allow to read (and write) objects and social connections of the Facebook Graph. The objects, that are the vertices of the graph, have a unique identifier (ID) and their associated data can be retrieved with a simple fetch of the URL `https://graph.facebook.com/ID`. All objects are linked together through relationships of different types for different objects. We retrieve the connections using URLs of the form: `https://graph.facebook.com/ID/CONNECTION_TYPE`. Obviously, in order for the queries to succeed they must not violate the privacy restrictions set by the users. The user can provide his/her personal credentials (username and password) to our *FacebookView Application* through an input window. The system automatically generates the necessary requests to determine the subgraph of the Facebook Graph induced by the set of vertices consisting of the facebook user and his/her friends. The initial focus vertex corresponds to the user that is owner of the account. Actually, in the current implementation the set of vertices (and relationships between them) is extended at run-time with queries that refer to resources (events, photos, links, videos, etc.) liked or tagged by other users.

The second case study shows the public relations exposed and available on the Web from the *Google Social Graph API* (see Fig. 8(d)). This information is declared within public profiles via XFN (XHTML Friends Network), FOAF (Friend Of A Friend), and other declared public connections. For example, XFN provides a simple way to define human relationships through Web links using the *rel* attribute of the <*a*> tag (e.g. Pino). The user can provide to our *SocialView Application* the URL of a public account through an input window. The system automatically generates a query for acquiring as much relational information as possible. The initial focus vertex even in this case corresponds to the user that is owner of the account.

We also implemented a case study to explore Wikipedia (see Fig. 8(c)). A user selects a word and the smartphone shows the related concepts. For example if the user selects Graph Drawing the device shows the Graph Theory, Topology, Geometry, etc. The initial focus vertex corresponds to the selected starting word. When the user finds an interesting concept, he/she can expand the vertex and visualize the Wikipedia page.



(a) FacebookView over iPhone

(b) FacebookView over Android

(c) WikipediaView over iPhone

(d) SocialView over Android

Figure 8: Application samples. The surnames of the involved people have been erased for privacy reasons.

7 Conclusions and Future Work

We have presented a system for the visualization of graphs on a smartphone. First, we have introduced a visualization paradigm that allows the user to navigate the graph using a focus-based approach. Second, we tackled the algorithmic challenges posed by the new visualization paradigm, introducing and experimenting effective heuristics. Finally, we have shown several customizations of the system aimed at exploring and at visualizing popular Web contents like social networks and Wikipedia. Current implementations include the iPhone and the Google Android platforms.

Several problems related to our paradigm are left open.

1. How to apply variations of our paradigm for graph visualization on handheld devices, like the iPad, whose screen has a size larger that a smartphone but smaller than a usual computer screen? Variations of the paradigms could include: the visualization of more than one focus vertex, the placement of lobe vertices on several concentric layers, or the usage of all the sides of the screen.

2. A typical way to visualize large graphs it to use clustering techniques. Is it possible to combine our paradigm with existing techniques for the visualization of clustered graphs?

3. Can the heuristics that we have presented for selecting a lobe be improved both from the point of view of the performance and from the point of view of the effectiveness?

References

[1] Michael Baur and Ulrik Brandes. Crossing reduction in circular layouts. In J. Hromkovič, M. Nagl, and B. Westfechtel, editors, *Proc. of WG'2004*, volume 3353 of *LNCS*, pages 332–343. Springer-Verlag, 2004.

[2] Peter Eades, Robert F. Cohen, and Mao Lin Huang. Online animated graph drawing for web navigation. In G. Di Battista, editor, *Proc. 5th Int. Symp. Graph Drawing, GD*, number 1353 in LNCS, pages 330–335. Springer-Verlag, 1997.

[3] Gartner. Press releases. http://www.gartner.com/, 2010.

[4] Hongmei He and Ondrej Sykora. New circular drawing algorithms. *Proc. ITAT'O4*, 2004.

[5] Joseph Y-T. Leung, Oliver Vornberger, and James D. Witthoff. On some variants of the bandwidth minimization problem. *SIAM J. Comput.*, 13:650–667, July 1984.

[6] Erkki Makinen. On circular layouts. *Internal Journal of Computer Mathematics*, pages 29–37, 1988.

[7] Sumio Masuda, Toshinobu Kashiwabara, Kazuo Nakajima, and Toshio Fujisawa. An NP-hard crossing minimization problem for computer network layout. Technical report, 1986.

[8] Pixelglow. Instaviz. http://instaviz.com/, 2008.

[9] Manojit Sarkar and Marc H. Brown. Graphical fisheye views of graphs. In *Proceedings of the Conference on Human Factors in Computing Systems CHI'92*, 1992.

[10] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proc. of the IEEE Symp. on Visual Lang.*, pages 336–343, 1996.

[11] Janet M. Six and Ioannis G. Tollis. Circular drawings of biconnected graphs. In *Proc. ALENEX*, 1999.