# JASPER: Just A new Space-filling and Pixel-oriented layout for large graph ovERview

Jason Vallet, Guy Melançon, Bruno Pinaud

▶ **To cite this version:**

Jason Vallet, Guy Melançon, Bruno Pinaud. JASPER: Just A new Space-filling and Pixel-oriented layout for large graph ovERview. Conference on Visualization and Data Analysis (VDA 2016), Feb 2016, San-Francisco, CA, United States. pp.1–10, 10.2352/ISSN.2470-1173.2016.1.VDA-484 . hal-01302430

**HAL Id: hal-01302430**
**https://hal.science/hal-01302430v1**

Submitted on 14 Apr 2016

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# JASPER: Just A new Space-filling and Pixel-oriented layout for large graph ovERview

***Jason Vallet, Guy Melançon and Bruno Pinaud,***
***Univ. Bordeaux, LaBRI, UMR 5800, F-33400 Talence, France***

## Abstract

*When analysing data and handling a visualisation, users mainly spend their cognitive resources making sense of the graphical representation and mapping it back to the data and domain. This task becomes even more critical when dealing with larger data sets. Therefore, a valuable visualisation design strategy is to couple graphical representations and user tasks to better support the sense making process. This paper focuses on a particular task where users must make sense of state changes occurring on nodes of a graph. To this end, we propose JASPER, a new layout algorithm focusing on the visualisation of nodes inspired from pixel-oriented layouts, relying on node clustering to identify and represent existing connections through spatial adjacency.*

*JASPER can layout moderate size graphs in real-time and is able to tackle large graphs with up to 2 million nodes and 5 million edges in reasonable time (about half a minute). Furthermore, although JASPER has been designed around a specific application, the underlying methodology can be employed to draw quick overviews of any type of graphs. The paper lays down the underlying principles of JASPER, and reports it performances (execution times) on increasingly large graphs. JASPER is then used and showcased to visualise network propagation phenomenon in large graphs.*

## Introduction

In recent years, the graph drawing community has focused on increasingly larger graphs, evolving along the multiplication of the ever expanding social networks and their popularisation. Used to study social behaviours or consumer habits, social networks are nonetheless complex to draw and analyse due to their sheer size, thus giving them additional interest for the research community. Roughly speaking, a social network represents a set of users possibly connected to one another and, when depicted as a graph, users are represented by nodes while a connection between two users is symbolised by an edge. In general, the huge graphs resulting from the mining of such networks force us to face several hundreds of thousands [26], if not millions [25], of nodes at once, thus pushing the existing representations, commonly used to display graphs of more moderate sizes, to their limits.

A lot of different techniques exist nowadays to draw graphs, and vary according to structural specificities (*i.e.* tree, directed/undirected or compound graphs). Most techniques aim at producing a human-readable and easily interpretable map of the network structure [37]. Popular approaches to display graphs are node-link diagrams, matrices and hybrid visualisations mixing these two representations (*e.g.* [17] and [32]). However, all of those visualisation techniques have limitations when facing large graphs, especially node-link representations [12, 17]. The difficulty of displaying millions of elements in a comprehensible fashion is obvious and, as a result, existing layout algorithms are not always efficient in representing such graphs. This is mainly due to the genericity of these methods whereas effective visual analysis can only be performed through appropriate visual representations [37]. Depending, among others, of the analytical tasks we wish to perform, a good visualisation should thus allow to instantly identify the information that really matter.

In a recent work [36], we proposed to use visual analytics to compare information propagation in social networks. However, when facing larger graphs (*e.g.* networks with thousands of users), we have been unable to find an existing layout which allows to understand at a glance how propagation operates on the network. Thus, we have decided to develop a new layout algorithm adequate for our needs: JASPER. It has been designed to compute compact layouts using pixel-oriented techniques so the resulting representations will give an overview of the graph. Propagation can be visually tracked as nodes change states using a simple color mapping.

Though JASPER is adapted to our particular problem, we believe it can be useful in a number of different situations as a complementary visualisation for evaluating at a glance some information on different graphs. Basically, our method adheres to the following observations:

1. Nodes are the most important elements and thus have to be clearly displayed,
2. Edges are important too but can impair the visualisation quality in dense graphs [12],
3. Whether we use a node-link diagram, a matrix or some hybrid visualisation, large graphs will still be too big for the viewer to process/visualise all the embedded information at once; all we can do is to give hints and directions.

The first point relates to the task JASPER is designed to support, with a clear focus on node states. The second point is addressed with authority, since JASPER makes edges non visible. This design choice however is not that radical given that edges carry no other information or attributes other than indicating that nodes are connected. The third observation underlines the fact that the occlusion problem raised by edges is a general one and applies to whatever visualisation we may consider when dealing with large graphs. To compensate this problem, a convenient visual metaphor can be used to suggest the presence of edges. To this end, we position nodes so that adjacent nodes (*i.e.* connected by an edge) remain as close as possible to each other, thus giving a hint of an existing connection between those elements to the viewer. We have to keep in mind that such approximation is necessary as representing every single detail in a legible way is not

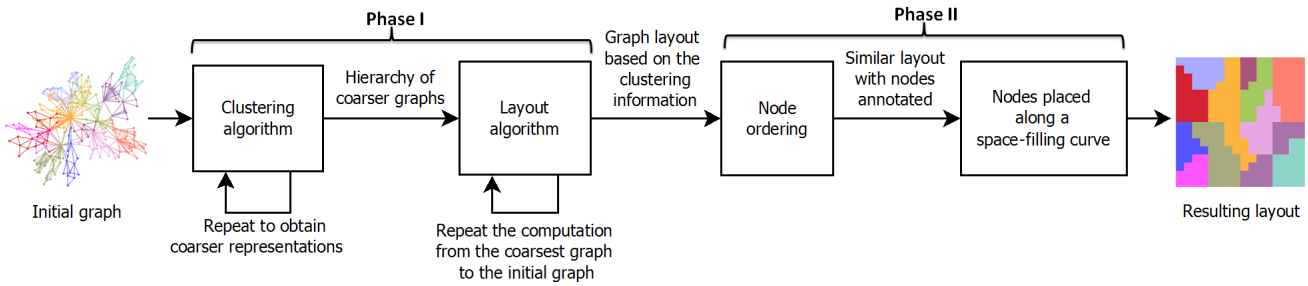



Figure 1: Workflow showing how operations are pipelined in JASPER, starting from a graph with an unspecified layout and returning the same graph with a node arrangement similar to a pixel-oriented layout.

reasonable considering the number of elements we are handling.

Following those observations and using our representation method is therefore not without drawbacks: there are limitations for users trying to visualise dynamic graphs with evolving topologies. This operation is beyond the scope of our current application as we are using snapshots of networks in a fixed configuration. It will nonetheless become relevant in the future as the structure of a social network is constantly evolving. As for now, we limit our visualisation on large static graphs and wish to solely visualise state modifications happening to different nodes.

In the following section, we will present some existing work related to our current solution and motivate in which aspects they were not appropriate for our application. This will be followed by the presentation of our method completed with some design and implementation details. We then produce some execution times and present resulting visualisations, extended afterwards as we introduce a use-case applying JASPER to the visualisation of a propagation phenomena. The final section allows us to start a discussion tackling existing problems, possible solutions and suggesting future work directions.

## Related work

Information visualisation and graph drawing have been the subject of extensive research through the years [37]. Among those works, three techniques are commonly used: node-link diagrams, matrices and hybrid visualisations. Node-link diagrams are quite familiar and natural representations for any graphs. Nodes and edges are both displayed and can thus give visual information (size, colour, shape, highlight, etc.). Though this type of visualisation can show the global structure of a graph, node-link diagram limitations are well-known as they give poor results when dealing with larger graphs and more particularly when edge density increases, quickly impairing their overall readability [12, 17]. Matrices, on the other hand, are simple to display and can be useful to appreciate the whole graph structure in one glance. Nodes and edges can still give visual information but the representation is nonetheless less natural and large matrices are difficult to work with [33]. Finally, hybrid visualisations (such as Node-trix [17] or Mat-tree [32]) can offer significant advantages from both previous solution but still require some learning efforts for users to get used to them.

Among all these possible representations, node-link diagrams, though proved to come with flaws, are often preferred to represent social networks [2]. They are mostly displayed in small samples using force-directed layouts which can give rather good results on any kind of graph. However, such algorithms can not be computed on large graphs in a small amount of time [16]. To find a solution to our large-graph visualisation problem, we thus have to look at works tackling visualisation of large-scale datasets (with several hundreds of thousands or millions of elements). Two examples of such work can be found in [4] and [34] where the authors propose multiscale visualisations of social networks. However, the resulting representations never show the graph in its entirety thus forbidding the kind of global overview we are seeking.

These two solutions nonetheless raise an interesting point in using the existing communities within the social networks to group nodes. More commonly called *clusters* when studied outside social networks, such node grouping can be used to simplify the layout computation in considering each cluster as a whole entity instead of trying to find an appropriate position for each node separately. This method has been used by Huang *et al.* [19] and Didimo *et al.* [8] to visualise up to fifteen thousand nodes and forty thousand edges. Huang *et al.* and Didimo *et al.* both describe their representations as space-efficient and space-filling layouts. Such quality is obviously important when dealing with large graphs so any visual information carried by the displayed elements can be noticed without focusing on each one specifically. Additional algorithms using space-filling curves as a base for their resulting layout exist. A few years ago, Muelder *et al.* [29] have proposed such a method. Its computation is quick, it can be applied on large graphs (1M nodes and 2M edges) and the produced layouts seem clearer than those obtained with force-directed algorithms, nonetheless, edges impair node visibility in the densest parts of the graph. A somewhat similar solution has been later proposed by Auber *et al.* [5] only using hierarchical data, thus making it inappropriate for general network visualisation. In this case however, edges are no longer needed as the hierarchy is indicated using coloured nested regions and borders, dividing the layout like a nested treemap. For more information and extended bibliographies on space-filling curves, we refer the reader to the two aforementioned papers ([5, 29]).

The idea of using space-filling curves is elegant and allows to efficiently use space. This can however be pushed even further with a pixel-oriented visualisation as initially introduced by Keim [21]. This method is used to analyse records and visualise the possible correlation between two measures, one being displayed using the position of the elements along a space-filling curve while the second is depicted using a simple colour mapping. Pixel-oriented visualisations have been successfully used to visualise very large datasets containing up to a billion of elements [35] and to perform social network analysis [13]. The capacity to display numerous elements is a necessity for our visualisation case,

however, edges –carrying information on the connectedness of each node in the network– cannot be explicitly represented using a pixel-oriented method. Duarte *et al.* [9] have recently encountered a similar problem and addressed it quite effectively. Their "Nmap" layout is introduced as being a neighbourhood preservation space-filling algorithm able to display connected nodes close to each other. As mentioned in the introduction, considering the number of elements to visualise, we have to represent the information simply and give hints on important facts as the user is not able to process that much information all at once. To this end, a layout algorithm offering to keep each node close to its neighbours, even if only approximately, seems a good solution. Though, it can not be applied on a smaller scale to obtain an exact representation, it presents sufficient qualities to be used to compute overviews on large graphs.

## Presentation of our solution

Our solution is divided in two main phases (see Fig. 1). We first compute a layout based on a coarse representation of the initial graph. This step allows to improve the computation time as we limit the number of elements, thus simplifying the graph to produce. Then, we reorganise all the nodes along a space-filling curve. The order on the curve for each node is based on its spatial position in the previous layout. The resulting layout is a space-filling representation with similitudes to pixel-oriented layout (only the nodes are displayed and existing edges are invisible by default). We end up with a compact visualisation where nodes belonging to the same connected group are set to be displayed spatially close to each other.

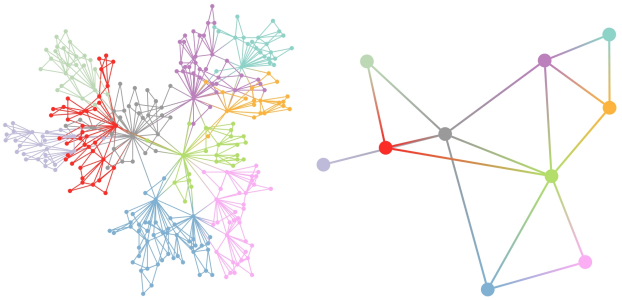
### *Phase I: layout using coarser graphs*

Sometimes mentioned as a coarser graph or a skeleton graph, this method allows to describe an initial graph by a much simpler representation. Similar process can be found in different algorithms and techniques such as Auber *et al.* [4], Frishman *et al.* [11], Itoh *et al.* [20] and Didimo *et al.* [8].

#### *Grouping nodes*

The starting point of our method is the application of a clustering algorithm on the initial graph. Typically, clustering algorithms are used to detect clusters in a graph, describing a subgroup of nodes particularly connected to each other. In graphs representing interactions between persons such as social networks, those groups are called communities. The subject is further developed by Fortunato [10] where an overview of different existing clustering algorithms is presented and several of those techniques are compared. According to the author, the Louvain algorithm proposed by Blondel *et al.* [7] and Infomap of Rosvalt *et al.* [31] provide the best results. Finally, the execution time on large graphs provided by Blondel *et al.* in their paper, completed with additional support from Didimo *et al.* [8] showing the prominence of Louvain's clustering technique have convinced us to opt for this solution in our method.

Once we apply the clustering algorithm on a graph, each node will be put in the same subgroup as their closer neighbours. As we wish for each node to be as close as possible to its neighbours when displayed, we entirely rely on the clusters quality – thus on the clustering algorithm efficiency– to appropriately regroup each node with its proper relatives.



(a) Initial graph $G_0$; each cluster $C_0$ is coloured differently for identification purpose.

(b) Coarser representation $G_1$; each coarse representative is displayed using the same colour as its cluster.

Figure 2: Example of a coarsening process on a simple graph divided in 10 clusters.

#### *Building on clusters*

Once each node is set in the adequate cluster with its closest neighbours, we have information on how to group users together. We start working on the initial graph to identify the different clusters and create a map of their relations. We thus obtain a first coarse graph. Repeating those operations several times produces coarser and coarser representations. Figure 2 shows an example of such treatment on a small graph. Starting with an initial graph (Fig. 2a), a computation of a coarser representation (Fig. 2b) gives us the connections between the clusters and results in a simpler graph. More generally, upon each computation of a coarser graph, we first group close nodes together then communities linked to one another. We use the groupings herein created to decide how close two users should be displayed.

Using a more formal approach, coarser representations can be described as follows. For a graph $G_n = (V_n, E_n)$ identified by a set of nodes $V_n$ and a set of edges $E_n \subseteq V_n \times V_n$, where $n \geq 0$; we define $G_0$ as the initial graph we want to visualise and $G_1, G_2, \ldots$ as its gradually coarser representations. By applying a clustering algorithm on $G_k$, where $k \geq 0$, a set of clusters $C_k$ is created such as each node $v \in V_k$ belongs to one and only one cluster $c \in C_k$ (described as $cluster(v) = c$). Those groupings are then translated to a coarser representation of the graph, *i.e.*, for each cluster $c$, a node $w$ is created in a new graph $G_{k+1}$ such as $w \in V_{k+1}$ and, if an edge $e \in E_k$ exists between $v$ and $v' \in V_k$, we update $G_{k+1}$ following those two cases:

*a)* $v$ and $v'$ are in distinct clusters ($cluster(v) \neq cluster(v')$), then, an edge $\varepsilon \in E_{k+1}$ is set between the two nodes $w$ and $w' \in V_{k+1}$ –respectively representing $cluster(v)$ and $cluster(v')$. When an edge linking $w$ and $w'$ already exists, a weight can be set and updated to indicate the multiple connections between the two clusters.

*b)* $v$ and $v'$ are in the same cluster $c$ ($cluster(v) = cluster(v') = c$), then the edge $e$ is not considered in graph $G_{k+1}$.

#### *Layout computation*

The layout is obtained through successive steps on the coarse representations. Starting with the coarsest graph $G_p$, we gradually use the layout computed on the clusters in $G_k$ ($p \geq k > 0$) to place the subgroups in $G_{k-1}$. The operation is repeated until $G_0$ is reached and treated.

To perform the first layout computation, we have to choose an efficient layout algorithm offering legible drawings in a small amount of time. In its survey on spring embedders and force-directed layouts, Kobourov [23] mentions some possible candidates for such application. Ultimately, we have selected the Fast Multipole Multilevel Method ($FM^3$) introduced by Hachul and Jünger [15]. The choice has been motivated based on the results showed by Hachul *et al.* [16] and Archambault *et al.* [1] as this layout method proposes a good balance between execution time and drawing quality with minimal edge-crossings and overlapping edges. Those two points are essentials as we wish to keep a readable layout of the clusters to easily distinguish one from another. An additional perk of the layout resides in its weighted variant offering advanced control if needed.

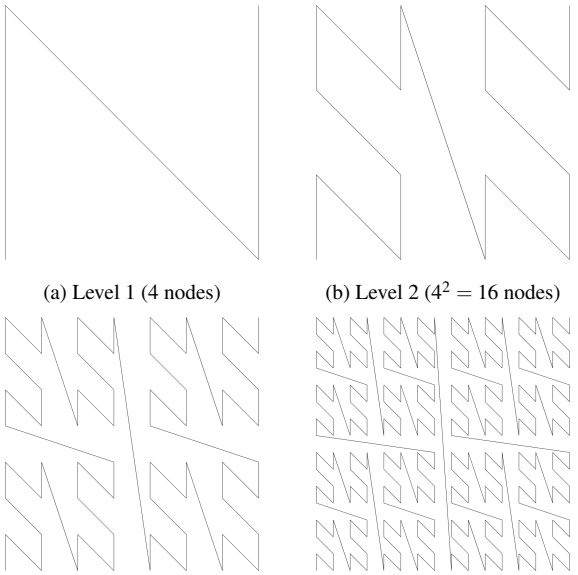
Computing the layout of the coarsest graph $G_p$ is straightforward as it simply requires to apply the algorithm layout on $G_p$. The layout of any of the following coarser graphs $G_k$, where $k < p$, needs a few additional steps. First, a local layout is computed in each cluster $c \in C_k$. Then, freshly computed coordinates of each node $v \in V_k$ belonging to the cluster $c$ are transformed through translation and scaling so that all nodes in $c$ are displayed in an area around $w \in V_{k+1}$, representing $c$ in the coarser graph $G_{k+1}$. Each node coordinate in a coarser graph $G_{k+1}$ is thus used as an anchoring point to layout the nodes of $G_k$. Those two steps are repeated until we reach $G_0$ and have computed the coordinates of each initial node.

Classic spring-embedded or force-directed layouts are time-consuming to compute when facing thousands of elements [1]. This nested method allows to compute several small layouts quickly. Furthermore, those operations and the geometrical transformations affecting node coordinates can be done independently from one another and thus be easily distributed. As a possible performance improvement evaluated on the testing graphs analysed in the following section, computing the inner layout of each cluster is not always the best choice as connections with nodes from outside the subgroup will not be accurately considered. In most cases where nodes within clusters are densely connected, a simple random layout gives acceptable results and allows to speed up the application by averting the nested layouts computations.

On a more general note concerning the whole Phase I, the rapidly decreasing number of nodes for each coarser representation makes the construction of $G_{k+1}$ and each application of the clustering and layout algorithms on it quicker. This is not surprising considering the graphs obtained from the clusters are getting simpler, however, based on our observations on the testing graphs, repeated applications to obtain several levels of coarse graphs do not necessarily improve the resulting layouts. As a result, in our applications, only one coarser representation ($G_1$) has been computed and used for each graph. We offer nonetheless the general method as such, considering structured graphs with established hierarchical communities will benefit from it.

### Phase II: pixel-oriented representation

Pixel-oriented visualisations have been presented by Keim [21] as a representation allowing to display important quantities of data in a minimal space. To do so, elements one wishes to display are ordered and placed along a space-filling curve. Our method uses a similar process with nodes being ordered according to their spatial position (computed by the end of Phase I). Each node is then laid on the space-filling curve at its given position. Finally, edges are hidden to obtain the maximum legibility.



(a) Level 1 (4 nodes) (b) Level 2 ($4^2 = 16$ nodes)

(c) Level 3 ($4^3 = 64$ nodes) (d) Level 4 ($4^4 = 256$ nodes)

Figure 3: Successive developments of the N-order space-filling curve introduced by Morton [27]; the layout is similar to a Z-order curve but with a 90 degrees rotation.

#### Space division and node ordering

Node ordering can be computed fairly easily. We use a technique similar to the one employed for $k$-d tree construction [6]. A $k$-d tree is a data structure used for the storage of $k$-dimensional data, however, in our case, the only dimensions considered are those specified with the nodes layout position ($x$ and $y$ coordinates). We first divide the graph in two parts with a similar number of nodes in each halves using a pivot value on the first dimension. The two resulting halves are then divided again in two equal parts but using a pivot value on the second dimension. This process is repeated while alternating dimensions until each node is isolated and thus is given an order. We can compute from the start how many divisions will be necessary to isolate all the nodes, however, because we wish to conserve a regular shape for the layout, the number of division have to be equal for each halves. This means regions with fewer nodes have to incorporate "holes", or white-spaces, which are given an order too. Moreover, because the number of nodes is equally distributed in each halves during the division, the "holes" are evenly spread. Finally, the resulting shape of the layout will be either a square or a rectangle depending of the number divisions and nodes to display.

The variable form of the representation has been introduced to limit the number of "holes" in the resulting layout. Our ideal visualisation ratio is set to be 1:1, thus a square is appropriate. However, the successive divisions and the rearrangement along a space-filling curve –explained in the next subsection– do not give us the possibility to freely rearrange nodes. The appearance of "holes" is noticeable on representations with a limited number of nodes (see Fig. 4b) but becomes less visible when facing graphs with more nodes (see Fig. 5b). A variable layout shape allows us to avoid incorporating too many "holes" and to use most of

Table 1: Complexity of Phase I and II of JASPER applied on a graph $G$ using one level of coarsening. $|V_0|$ and $|E_0|$ define respectively the number of nodes and edges of the initial graph $G_0$ while $|V_1|$ expresses the number of clusters computed from $G_0$ (equivalent to the number of nodes of $G_1$) and $|E_1|$ designates the edges connecting those clusters (similar to the number of edges of $G_1$).

| | Phase I: coarser graphs | | | Phase II: pixel-oriented representation | |
|---|---|---|---|---|---|
| Step | Clustering | Coarse graph construction | Layout computation | Node ordering | Placement on the curve |
| Complexity | $O(\|E_0\|)$ | $O(\|E_0\|)$ | $O(\|V_1\|log\|V_1\|+\|E_1\|)$ | $O(\|V_0\|log^2\|V_0\|)$ | $O(\|V_0\|log\|V_0\|)$ |

the space available by following a simple rule: let $n \in \mathbb{N}$ and $|V|$ the number of nodes in the graph to display, the resulting shape will be a square if $2 \times 4^{n-1} \leq |V| < 4^n$ and a rectangle otherwise (when $4^n < |V| \leq 2 \times 4^n$). We test the number of nodes during the first space division (on the $x$ coordinate) to either spread the data on a square or on half that surface, *i.e.*, a rectangle.

#### *Placement along a space-filling curve*

Following the order computed during the space division operation, we can sort the nodes. We use the space-filling curve popularised by Morton [27]. Sometimes called Z-order curve, we will prefer the original layout using an *N* shape (see Fig. 3). For different developments of the curve, the number of nodes we are able to place along it evolves.

One can see how the successive space divisions give an appropriate order to each node to be displayed along this specific curve shape. Similar results can be obtained when using different curves designs or orientations –such as the Hilbert [18] or the Z-order curves– as long as the node ordering during the division respects the path followed by the curve. This last point is crucial to conserve adjacency in the final layout, otherwise, nodes which should be close end up being separated if the space division and reconstruction do not follow the same pattern.

## Complexity and results

After having introduced the algorithm and motivated each implementation choice, we now present the layout obtained when using our method. A specific use case will be introduced after this section but we wish to apply the algorithm to well known graphs beforehand for the reader to judge the resulting layouts and the computing times. First, let us speak a few words about the complexity of our method.

### *Complexity*

We stay in the previous context by keeping the algorithm separated in two phases. Following the notation developed earlier, let $|V_0|$ and $|E_0|$ be respectively the number of nodes and edges of the initial graph $G_0$, $|V_1|$ the number of clusters computed from $G_0$ (equal to the number of nodes from $G_1$) and $|E_1|$ the number of edges connecting those clusters (number of edges in $G_1$). We remind the reader we only compute one coarse representation ($G_1$) from the initial graph as we find using more than one level does not significantly improve the final layout. A more complete analysis showing the differences when using more than one coarser graph is left for future work.

The overall worst-case complexity is in $O(|V_0|log^2|V_0| + |E_0|)$. The first phase, during which the coarse graph is built, is performed in $O(|V_1|log|V_1|+|E_1|+|E_0|)$ while the second phase is done in $O(|V_0|log^2|V_0|)$. To simplify the expression, we maximise $|V_1|$ and $|E_1|$ to $|V_0|$ and $|E_0|$ respectively. The complete details for each step and computation phase are given in Table 1. The given values concern non-parallel executions. However, due to the current operating systems abilities and processor architectures, we hardly recommend to use single-core implementation when parallelisation of instructions can be achieve easily with compiler tools like OpenMP.

### *Execution times*

Table 2 presents the results of our algorithm execution. The implementation and running time measurements have been done on a computer using an Intel i7-3840QM processor (2.8GHz, 4 hyper-threaded cores) with 32GB of RAM, a NVidia K2000M, running a Linux distribution (Ubuntu 14.04) and the visualisation software Tulip (`http://tulip.labri.fr`) [3] in its latest version (4.8). The time measurements have been performed with the Boost library (`http://www.boost.org` – version 1.55) and consider the real process time execution (processor time used exclusively for the program execution[1]). The datasets used are freely available online on the *Stanford Network Analysis Project* website (`http://snap.stanford.edu`) and easily fit in random-access memory during computation as less than 8GB are used on the testing machine at all time. The graphic rendering and display times are not included in the measures, thus only concerning the layout computation. As an informal measure, the *wiki-Talk* layout has been displayed in less than 20s; please also note the important need in memory for the full rendering of the *com-LiveJournal* dataset ($\simeq$ 25GB).

The results presented in the table are average values obtained after 10 runs for each dataset. We use the existing implementations of Louvain and $FM^3$ available in Tulip, thus a small overhead due to the inner mechanisms of the framework is added to our workflow. Nonetheless, we end up being able to compute the layout on relatively large graphs (2M nodes and 5M edges) in roughly 35 seconds. The last dataset (*com-LiveJournal*) is presented to show the limits of our solution on a much denser graph. Table 2 shows some details concerning the time needed for intermediate computations as well. We observe the clustering algorithm is responsible on average for 30% of the total computation time on the first datasets while it is accountable for around 75% of the computation time needed for the last graph. Hence, the clustering algorithm impacts significantly on both the quality –when deciding the groupings– and the rapidity of the solution, especially when dealing with large and dense graphs. On the other hand, the layout algorithm ($FM^3$) executed on the coarse graph presents a relatively small workload (15% in the worst case) in comparison (measured as a part of the *Coarsening* time).

As planned, the results for some steps based on the previously announced complexities are different, the information given

[1] Corresponds to the *process_real_cpu_clock* from the Boost library

Table 2: Presentation of the execution times for JASPER. Each time value is an arithmetic mean measured on 8 runs and expressed in seconds: *Clustering* gives the time for the clustering algorithm execution and *Coarsening* is for the rest of the Phase I operations; *Phase II* corresponds to the whole Phase II application (pixel-oriented layout computation); *Total* indicates the total execution time, sum of the previous values, and $\sigma$ gives the standard deviation of the *Total* value.

| | Graph size | | Time (seconds) | | | | |
|---|---|---|---|---|---|---|---|
| Dataset | Nodes | Edges | Clustering | Coarsening | Phase II | Total | $\sigma$ |
| email-Enron [26] | 36,692 | 367,662 | .23 | .35 | .27 | .85 | .02 |
| soc-Slashdot0922 [26] | 82,168 | 948,464 | .77 | 1.16 | .59 | 2.52 | .04 |
| com-DBLP [26] | 317,080 | 1,049,866 | 3.11 | 1.32 | 2.37 | 6.80 | .16 |
| com-Youtube [39] | 1,134,890 | 2,987,624 | 8.35 | 5.87 | 9.08 | 23.30 | .64 |
| wiki-Talk [25, 24] | 2,394,385 | 5,021,410 | 9.84 | 4.18 | 20.24 | 34.26 | 1.64 |
| com-LiveJournal [39] | 3,997,962 | 34,681,189 | 190.59 | 20.78 | 33.79 | 245.16 | 5.97 |

earlier being only valid for single threaded executions. Thus, space-filling and node ordering computations are quicker than initially expected thanks to their parallel implementations. To this end, we have used OpenMP (with G++ 4.82) to help improve our code efficiency and execute it on 8 simultaneous threads, the maximum possible number on the computer used for measures. We plan to address the running time comparison when using different number of cores in future work.

### *Resulting visualisation*

We illustrate the general results obtained with our technique using two datasets. The first one, presented in Figure 4, is a random network generated following the model of Wang *et al.* [38]. The graph is rather small, only consisting of 30,000 nodes and 59,997 edges. Colours are mapped on the clusters to help differentiate one from another. The number of clusters, while being relatively low (between 70 and 80), is still significant enough so that no existing colour-scale can provide enough discernible variations [14]. Consequently, some cluster colours appear to be quite similar while truly being different. We first show the coarse graph and the final layout obtained when using our algorithm, respectively Figures 4a and 4b. Figure 4c displays the same graph using a classic node-link view with a force-directed layout ($FM^3$).

One can use the colours to identify the correspondences in the clusters between the two layouts, when looking at the coarse graph and resulting layouts (resp. Fig. 4a and 4b). Viewing those two representations side by side also allows to witness how our solution tends to preserve locality as connected clusters ends up next or not far one from another. Likewise, we can notice the nodes placement in Fig. 4b respects the clusters position from Fig 4a (for instance: the two orange clusters on the bottom-left or the "reddish" ones in the top-right area). Similar cross-examination is unfortunately much harder between the classic view (Fig. 4c) and the resulting layout (Fig 4b) due to the random initialisation performed by $FM^3$, used in both drawings.

Altogether, the force-directed layout in Figure 4c provides a clear and understandable view of the graph. However, when facing densely connected or larger networks, force-directed representations are not clear enough and may take a long time to compute. To illustrate this point, we use the DBLP dataset (317,080 nodes and 1,049,866 edges) introduced in [26]. Figure 5 shows the difference between the two rendered representations. The classic visualisation (Fig. 5a) using the $FM^3$ algorithm alone is unreadable and the communities, somewhat still discernible when solely using the force-directed algorithm with the previous example (Fig. 4c), now melt into a single shapeless mass. On the other hand, our resulting layout (Fig. 5b) offers a much clearer overview of the graph. Each cluster appears more distinctively, even with limited colour variations, and, as connected communities are most likely laid out one next to another, additional information and insights concerning the group relations can be gathered by the person visualising the graph. This point is especially true if the used visualisation framework proposes specific interactors for neighbour identification or exploration such as the neighbour highlighter [28]. As shown in Figure 5c, using such tool allows users to examine and study the graph as well as the connections between nodes. The absence of edges improves the visualisation overall legibility but those elements are not deleted and can still be visualised. Moreover, organising and positioning the nodes in clusters gives some structure to the graph, helping users in seeing which groups of nodes are connected.

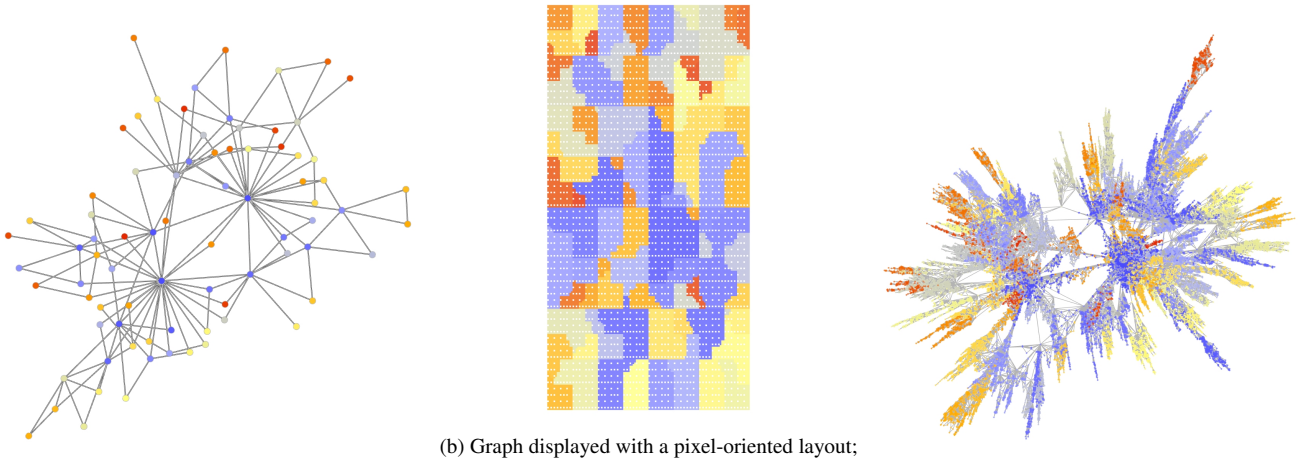
## Use case: overview of a propagation phenomenon

Our work emerged from a problem encountered while working on information propagation over social networks. The main task we were trying to achieve was to assist users in visualising how a propagation evolves, helping them to perform visual comparison between different propagation models. We were then looking for a simple way to obtain readable overviews of large graphs which could visually inform the user of the current state of propagation in the network. In the following, we quickly recall the principle of information propagation in a network and present some of the resulting visualisations we obtained using JASPER.

### *Propagation in a network*

Information propagation is a basic phenomenon observed in social networks. The principle is similar to an infection, initially starting with a few contaminated persons and gradually transferred to their acquaintances and persons who may have been in contact with them. Such infection can start in one or several different locations in a network with some nodes transferring it to their neighbours. Newly contaminated nodes can themselves become infectious and thus able to spread the disease to their own neighbours, repeating the process.

One of our previous work [36] was proposing to use visual analytics to compare existing propagation models describing such behaviour. We have shown how this task could be achieved on a

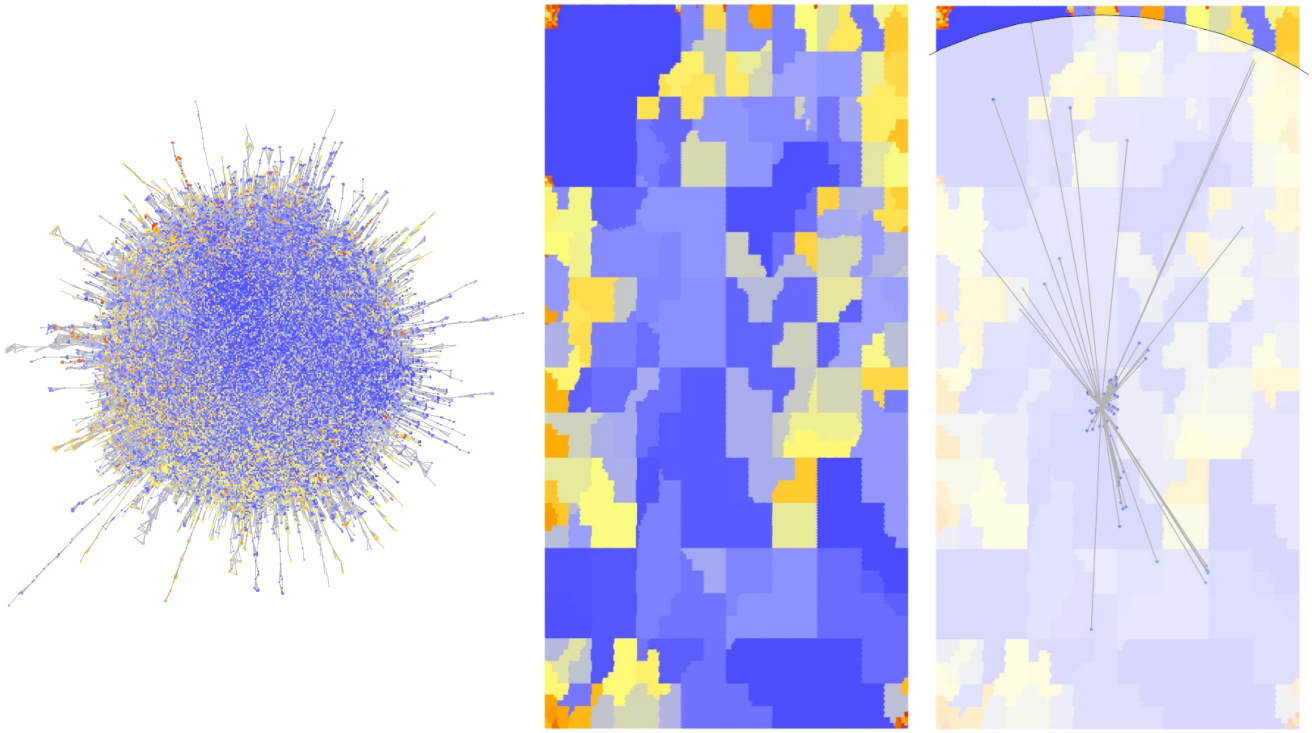


(a) Layout of the coarse graph using $FM^3$ – obtained after Phase I.

(b) Graph displayed with a pixel-oriented layout; mapping based on the coarser graph representation – obtained after Phase II.

(c) Node-link representation of the same graph using the $FM^3$ algorithm.

Figure 4: Application of our solution on a random graph (30k nodes and 60k edges) generated with the model of Wang *et al.* [38]; the first two figures (a, b) show the intermediary and final layout of our algorithm while the third figure (c) shows the same graph displayed with a common force-directed layout.



(a) Node-link representation of the network using the $FM^3$ algorithm; layout computed in 180 seconds using OGDF.

(b) Coarse graph view transformed in a pixel-oriented view.

(c) Use of a neighbour highlighter interactor on a node of the same graph.

Figure 5: Application of our solution on the DBLP graph [26] (317k nodes and 1M edges); the first figure (a) shows the graph displayed with a force-directed layout while the second figure (b) uses a layout computed using JASPER. In the third figure (c), we use the representation (b) previously obtained, select a node and visualise its direct neighbours using a visual filter.

small graph (300 nodes) drawn using a force-directed layout to give a simple and direct example. The propagation evolution was clearly noticeable, each node infection being notified by a changing colour, first, as it is touched –or contaminated– and then, as it becomes *active*, *i.e.*, infectious. Nonetheless, without an algorithm able to compute a layout for a large graph in a small time, this visual method was not scalable. The algorithm presented in this paper aims at achieving such goal with a layout computable in a decent amount of time and presenting a full representation of the graph. The need for a spatial grouping of nodes in clusters becomes evident at this point as a propagation is due to local events, thus affecting first the nodes in clusters containing already infected elements. By gathering the connected nodes in the same area, we want to focus the visual changes induced by the propagation to a specific region of the graph displayed.

Our original work was using the graph rewriting software PORGY introduced in [30], however, facing large graphs similar to those studied in the previous section has proved to be too much for the program. Indeed, some of the performed operations, like the search of matching subgraphs, are very demanding and thus may take an overwhelming amount of time on large graphs. We thus have decided to discard for now the graph rewriting layer and only use the basic Tulip software [3], upon which PORGY is based, to perform and visualise the propagation. To this end, we have implemented the linear threshold model presented by Kempe *et al.* [22] and used the Python interpreter embedded in Tulip to directly apply it to a network.

We have selected the DBLP dataset [26] as a topological base for this use case. Every additional information concerning the graph, such as the influence on a user from another (noted $b_{v,w}$) or the threshold value for each node ($\theta_v$), are randomly generated to compensate for our lack of knowledge. We nonetheless respect the model specificities noted by Kempe *et al.*: let $N(v)$ and $N_A(v)$ be respectively all the neighbours of $v$ and solely its active neighbours, $i$) the joint influence of the neighbours is never higher than 1 ($\sum_{w \in N(v)} b_{v,w} \leq 1$) and $ii$) a node activates only when it is sufficiently influenced (*i.e.* when $\sum_{w \in N_A(v)} b_{v,w} \geq \theta_v$).

### *Visualising the resulting evolution*

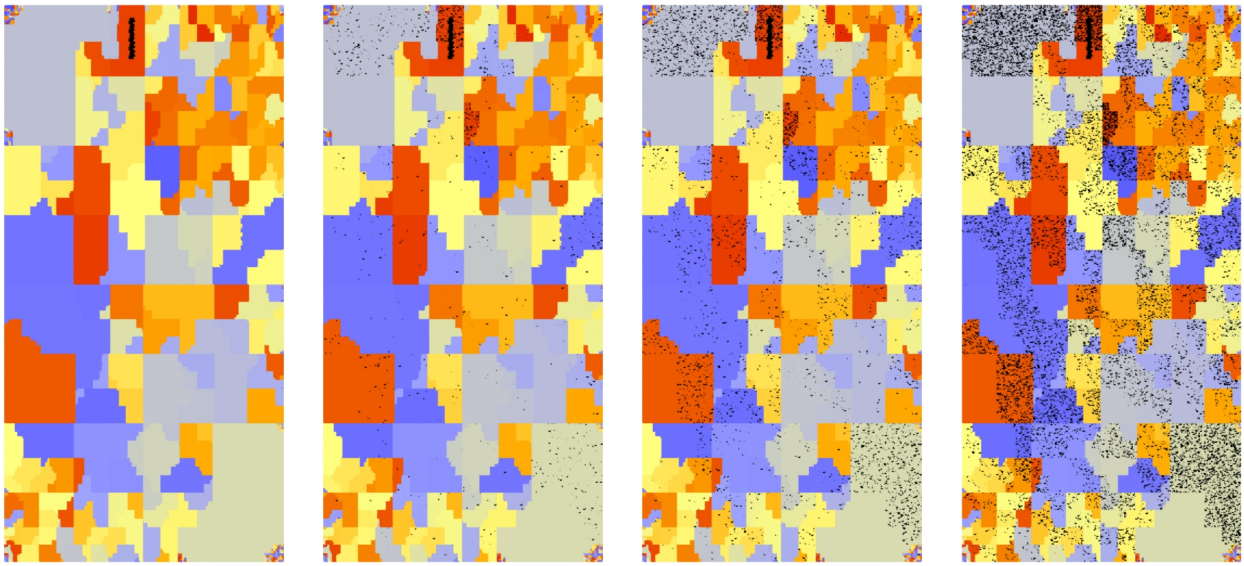
Figure 6 presents the visual overviews, obtained with JASPER, of a single propagation simulation. Just like before, we colour in a similar fashion the nodes belonging to the same clusters to allow their visual identification. The initial set of nodes, selected to be the propagation starting point, are coloured in black and can be identified as the dark rectangle in the central top part of Figure 6a in the red cluster. From this point and using the linear threshold model, we generate a propagation phenomenon. To visually depict this ongoing process, the colour of nodes relaying the propagation is changed to black. Fig. 6b shows the graph evolution after several propagation steps. Some nodes within the same cluster as the starting set have been contaminated and the infection is slowly spreading to other clusters. Fig. 6c and 6d show additional overviews with an ever increasing number of infected nodes. As you may notice, the resulting layout is not quite identical to the one showed in Fig. 5b. It has been explained earlier that, as JASPER uses the $FM^3$ algorithm as a basic layout, the final representation is not deterministic. Additional changes may also be attributed to the local modularity optimisation performed by Louvain, the clustering algorithm used in our method.

In this use case, we have computed an additional layer of clustering compared to the results presented in the previous section. It allows us to group connected nodes more precisely. This underlying structure can be clearly seen in Fig. 6d. Two of the biggest clusters (top-left in light-blue and bottom-right in sand-yellow) show a clean separation in the repartition of active and inactive nodes within them. This is not an artefact and can be explained, for instance, by either a high threshold value for one or several nodes between those two subclusters, or an insufficient influence of those central nodes on their inactive neighbours. If we look carefully, we can see that similar events actually happen in most of the clusters. When we continue the propagation process, some of those subclusters end up been infected while others keep their nodes completely inactive.

## Discussion and future work

The use case previously presented exposes perfectly our main intention when initially designing this layout algorithm: to propose a legible overview for large graphs. We acknowledge the limitations of our method as we emphasize the *overview* characteristic of the resulting visualisation. Performing analysis or in-depth studies on a graph can not be achieved using our method alone. Obviously, if one tries to discover all there is to know about a graph using a single approach, it will not be sufficient to understand every singular detail; attaining this outcome will indeed require different methods and points of view. Only then, may it ultimately result in a more complete picture. This is true for our method as, due to our focus on node representation, we have mostly hidden the edges, thus disregarding some information. Even though we use node placement and colours to give hints of detected communities and existing connections, such metaphors are far from perfect and the visualisation produced is not entirely sound. Nevertheless, with the assistance of supplementary interactors and visual representations, additional insights on the structure of the graph and connections established through the edges can be easily accentuated; the neighbourhood highlighter available in Tulip or a matrix/node-link drawing focusing on specific sub-graphs are such tools. Furthermore, this lack of explicit information is not solely encountered when using our method as large graphs typically contain more information and details than one can possibly display simultaneously. Even other solutions similar to JASPER, using pixel-oriented layout to maximise the amount of information displayed in limited space, can only give an approximation [35]. For instance, considering the wiki-Talk dataset ($2,394,385$ nodes), we realise that simply using one pixel to display each node is impossible as the sheer number of elements to display is bigger than the number of pixels available on nowadays common display interfaces (a "full HD" resolution is only able to display $1920 \times 1080 = 2,073,600$ pixels). Multi-level solutions like [4] and [34] can overcome such restrictions but only up to a certain point as their resulting layouts solely provide an approximated or partial representation of the graph.

The visualisation obtained with JASPER is very different from a typical node-link representation. Most notably, laying out the elements on a grid limits the number of direct spatial neighbours in the representation whereas a regular force-directed layout will not face such restriction. Any layout usually produces some distortion as it is not a perfect representation of the existing graph topology (different edges length, various radial distribution for



(a) (b) (c) (d)

Figure 6: Visualisation of a propagation phenomenon on the DBLP graph [26] (317k nodes and 1M edges) using JASPER overviews; the four layouts represent nonconsecutive steps of a single propagation simulation on the same graph, each node turning black (for visual impact) when becoming *active*, or infected, during the propagation.

each neighbour, ...). Altogether, due to their particularly compact node arrangement, we can expect the distortion induced by pixel-oriented layouts to be bigger than any force-directed solutions. Further studies and analyses of the distortions introduced by JASPER is left for future work.

The topic of dynamic data has been briefly mentioned earlier but not developed further. Obviously, the quick execution of our method allows to recompute a whole layout whenever an element is added or deleted, however, the non-deterministic characteristic of the drawings, imputed mostly to $FM^3$, implies a complete transformation of the graph, thus disrupting the user mental map. Entirely recomputing the layout may in some cases be a mandatory step if added or deleted elements are key nodes or edges. Indeed, deleting the central node of a cluster may result in a division of the said cluster while inserting a new edge can result in two different clusters merging. Yet, we believe it is better to keep a certain continuity in the position of the elements, thus we plan to adapt JASPER to display dynamic graphs with minimal changes in the final layout and, consequently, on the mental map.

## Conclusion

Initially created to be an efficient method to compute overviews of large graphs, we believe JASPER can be useful in all the situations where a quick visualisation to assess the state of a graph is required. When used with the appropriate colour-mapping, the resulting graph can visually communicate relevant and accurate information as all the nodes are displayed at the same time, thus avoiding approximations or incomplete information one can encounter in other visualisations (for instance, when using multi-scale views).

The resulting layout is compact and present similarities with pixel-oriented layouts, allowing an efficient use of the available space. We believe this characteristic to be very important and, because the size of the data sets we wish to visualise is continuously increasing with time, methods able to represent such large graphs will always be needed.

## References


[1] Daniel Archambault, Tamara Munzner, and David Auber. Topolayout: Multilevel graph layout by topological features. *IEEE Trans. on Visualization and Computer Graphics*, 13(2):305–317, 2007.

[2] Daniel Archambault and Helen C. Purchase. On the effective visualisation of dynamic attribute cascades. *Information Visualization*, 2015.

[3] David Auber, Daniel Archambault, Romain Bourqui, Maylis Delest, Jonathan Dubois, Bruno Pinaud, Antoine Lambert, Patrick Mary, Morgan Mathiaut, and Guy Melançon. Tulip III. In *Encyclopedia of Social Network Analysis and Mining*, pages 2216–2240. Springer New York, 2014.

[4] David Auber, Yves Chiricota, Fabien Jourdan, and Guy Melançon. Multiscale visualization of small world networks. In *Proc. of the 9th Annual IEEE Conf. on Information Visualization*, pages 75–81. IEEE Computer Society, 2003.

[5] David Auber, Charles Huet, Antoine Lambert, Benjamin Renoust, Arnaud Sallaberry, and Agnes Saulnier. Gospermap: Using a gosper curve for laying out hierarchical data. *IEEE Trans. on Visualization and Computer Graphics*, 19(11):1820–1832, 2013.

[6] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[7] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *J. of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[8] Walter Didimo and Fabrizio Montecchiani. Fast layout computation of hierarchically clustered networks: Algorithmic advances and experimental analysis. In *16th Int. Conf. on Information Visualisation*,

pages 18–23, 2012.

[9] Felip S.L.G. Duarte, Fabio Sikansi, Francisco M. Fatore, Samuel G. Fadel, and Fernando V. Paulovich. Nmap: A novel neighborhood preservation space-filling algorithm. *IEEE Trans. on Visualization and Computer Graphics*, 20(12):2063–2071, 2014.

[10] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(35):75 – 174, 2010.

[11] Yaniv Frishman and Ayellet Tal. Online dynamic graph drawing. In *Proc. of the 9th Joint Eurographics / IEEE VGTC Conf. on Visualization*, pages 75–82. Eurographics Association, 2007.

[12] Mohammad Ghoniem, Jean-Daniel Fekete, and Philippe Castagliola. On the readability of graphs using node-link and matrix-based representations: A controlled experiment and statistical analysis. *Information Visualization*, 4(2):114–135, 2005.

[13] Robert Gove, Nick Gramsky, Rose Kirby, Emre Sefer, Awalin Sopan, Cody Dunne, Ben Shneiderman, and Meirav Taieb-Maimon. Netvisia: Heat map and matrix visualization of dynamic social network statistics and content. In *Privacy, Security, Risk and Trust and IEEE 3rd Int. Conf. on Social Computing*, pages 19–26, 2011.

[14] Stephen J. Guastello. *Human Factors Engineering and Ergonomics: A Systems Approach, Second Edition*. Taylor & Francis, 2013.

[15] Stefan Hachul and Michael Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In *Graph Drawing*, LNCS, pages 285–295. Springer Berlin Heidelberg, 2005.

[16] Stefan Hachul and Michael Jünger. Large-graph layout algorithms at work: An experimental study. *J. of Graph Algorithms and Applications*, 11(2):345–369, 2007.

[17] Nathalie Henry, Jean-Daniel Fekete, and Michael J. McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE Trans. on Visualization and Computer Graphics*, 13(6):1302–1309, 2007.

[18] David Hilbert. Ueber die stetige Abbildung einer Linie auf ein Flächenstück. *Mathematische Annalen*, 38(3):459–460, 1891.

[19] Mao Lin Huang and Quang Vinh Nguyen. A space efficient clustered visualization of large graphs. In *4th Int. Conf. on Image and Graphics*, pages 920–927, 2007.

[20] Takayuki Itoh, Chris Muelder, Kwan-Liu Ma, and Jun Sese. A hybrid space-filling and force-directed layout method for visualizing multiple-category graphs. In *IEEE Pacific Visualization Symposium*, pages 121–128, 2009.

[21] Daniel A. Keim. Designing pixel-oriented visualization techniques: theory and applications. *IEEE Trans. on Visualization and Computer Graphics*, 6(1):59–78, 2000.

[22] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proc. of the 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 137–146. ACM, 2003.

[23] Stephen G. Kobourov. Spring embedders and force directed graph drawing algorithms. *CoRR*, abs/1201.3011, 2012.

[24] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Predicting positive and negative links in online social networks. In *Proc. of the 19th Int. Conf. on World Wide Web*, pages 641–650. ACM, 2010.

[25] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed networks in social media. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*, pages 1361–1370. ACM, 2010.

[26] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *CoRR*, abs/0810.1355, 2008.

[27] G. M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, IBM Ltd., 1966.

[28] Tomer Moscovich, Fanny Chevalier, Nathalie Henry, Emmanuel Pietriga, and Jean-Daniel Fekete. Topology-aware navigation in large networks. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*, pages 2319–2328. ACM, 2009.

[29] Chris Muelder and Kwan-Liu Ma. Rapid graph layout using space filling curves. *IEEE Trans. on Visualization and Computer Graphics*, 14(6):1301–1308, 2008.

[30] Bruno Pinaud, Guy Melançon, and Jonathan Dubois. Porgy: A visual graph rewriting environment for complex systems. *Computer Graphics Forum*, 31(3):1265–1274, 2012.

[31] Martin Rosvall and Carl T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proc. of the National Academy of Sciences*, 105(4):1118–1123, 2008.

[32] Sébastien Rufiange, Michael J. McGuffin, and Christopher P. Fuhrman. Treematrix: A hybrid visualization of compound graphs. *Computer Graphics Forum*, 31(1):89–101, 2012.

[33] Joris Sansen, Romain Bourqui, Bruno Pinaud, and Helen Purchase. Edge visual encodings in matrix-based diagrams. In *Proc. of the 19th Int. Conf. on Information Visualisation*, 2015.

[34] Lei Shi, Nan Cao, Shixia Liu, Weihong Qian, Li Tan, Guodong Wang, Jimeng Sun, and Ching-Yung Lin. Himap: Adaptive visualization of large-scale online social networks. In *IEEE Pacific Visualization Symposium*, pages 41–48, 2009.

[35] Ben Shneiderman. Extreme visualization: Squeezing a billion records into a million pixels. In *Proc. of the 2008 ACM SIGMOD Int. Conf. on Management of Data*, pages 3–12. ACM, 2008.

[36] Jason Vallet, Hélène Kirchner, Bruno Pinaud, and Guy Melançon. A visual analytics approach to compare propagation models in social networks. In *Proc. of Graphs as Models*, volume 181 of *Electronic Proc. in Theoretical Computer Science*, pages 65–79. Open Publishing Association, 2015.

[37] Tatiana von Landesberger, Arjan Kuijper, Tobias Schreck, Jörn Kohlhammer, Jarke J. van Wijk, Jean-Daniel Fekete, and Dieter W. Fellner. Visual analysis of large graphs: State-of-the-art and future research challenges. *Computer Graphics Forum*, 30(6):1719–1749, 2011.

[38] Lei Wang, F. Du, H. P. Dai, and Y. X. Sun. Random pseudofractal scale-free networks with small-world effect. *The European Physical Journal B - Condensed Matter and Complex Systems*, 53(3):361–366, 2006.

[39] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *CoRR*, abs/1205.6233, 2012.

## Author Biography

*Jason Vallet received his MSc in computer science from "Université du Québec à Chicoutimi" (2013) and is currently a PhD student in computer science at "Université de Bordeaux". His thesis subject is focused on information visualisation, graph rewriting, and their joint applications.*

*Guy Melançon received his PhD in combinatorial mathematics from "Université du Québec à Montréal" in 1991. Since 2007, he is a full professor in computer science at "Université de Bordeaux". His work has focused on information visualisation and network analysis.*

*Bruno Pinaud received his PhD in computer science from "Université de Nantes" in 2006. Since 2008, he is an associate professor in computer science at "Université de Bordeaux". His work has focused on information visualisation, graph rewrite modelling and experimental evaluation.*