Vizster: Visualizing Online Social Networks

Jeffrey Heer
Computer Science Division
University of California, Berkeley

danah boyd
School of Information Management and Systems
University of California, Berkeley

ABSTRACT

Recent years have witnessed the dramatic popularity of online social networking services, in which millions of members publicly articulate mutual "friendship" relations. Guided by ethnographic research of these online communities, we have designed and implemented a visualization system for playful end-user exploration and navigation of large-scale online social networks. Our design builds upon familiar node-link network layouts to contribute customized techniques for exploring connectivity in large graph structures, supporting visual search and analysis, and automatically identifying and visualizing community structures. Both public installation and controlled studies of the system provide evidence of the system's usability, capacity for facilitating discovery, and potential for fun and engaged social activity.

CR Categories and Subject Descriptors: H.5 [User Interfaces] Graphical User Interfaces, I.3 [Methodology and Techniques] Interaction Techniques, K.8 [Personal Computing]
Keywords: social networks, visualization, graphs, community, data mining, exploration, play

1 INTRODUCTION

The advent of the Internet has given rise to many forms of online sociality, including e-mail, Usenet, instant messaging, blogging, and online dating services. In 2003, another form of online community acquired stunning popularity: online social networking services. In addition to descriptive personal profiles, members of such communities publicly articulate mutual "friendship" links with other members, creating a browseable network of social relations. Although pre-dated by other services such as sixdegrees.com in 1997, social networking services emerged as a veritable phenomenon with the dramatic rise of friendster.com, which rapidly amassed millions of users beginning in the spring of 2003. Friendster's success was closely followed by a number of other popular services, including Tribe.net and orkut.com, quickly earning these services the acronym "YASNS" or "Yet Another Social Networking Service."

Though the users of these systems have constructed massive graph structures of social connectivity, typical web interfaces to these social networks remain relatively impoverished, showing only the network connections of single individuals in a linear list on a web page. Articulated connections between one's own "friends" in these systems are obscured, and can be unearthed only by paging through each friend's profile page. Higher level patterns of community can be even harder to discern. This has problematic implications for members' ability to explore their online community and gauge both the scale and the individuals to which their self-reported personal information is exposed.

In this paper we present the design of Vizster, a visualization system for exploring such online social networks. Vizster builds on ethnographic research of online social networking services and previous work in social network visualization to provide a system by which members of such online communities can explore their articulated social network in the playful manner that they desire. Our system is motivated by both social and technological concerns. On the social side, we attempted to better facilitate the discovery of people, connections, and communities to promote increased awareness of community structure and information exposure, while preserving (and hopefully enhancing) a fun and engaging online space. From a design perspective, this case study explores the mutually informing use of ethnographic techniques and visualization design to craft a domain-specific visualization system in a context as much characterized by play as by analysis.

E-Mail: jheer@cs.berkeley.edu, dmb@sims.berkeley.edu

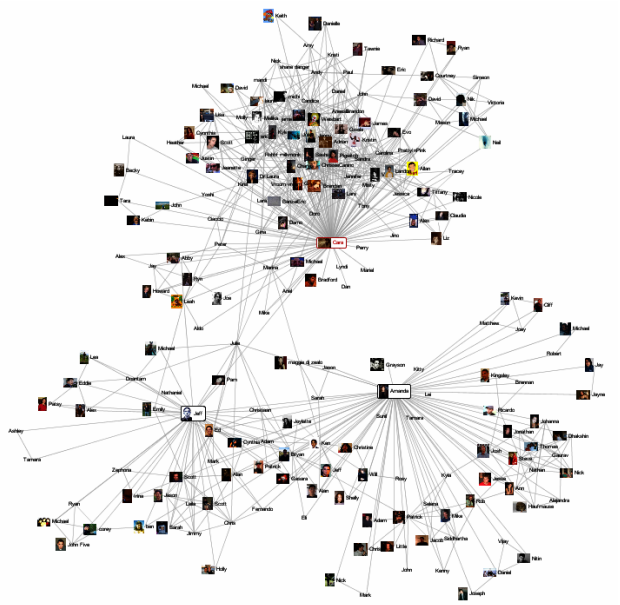


Figure 1: Vizster depicting three intersecting social networks

Though the resulting system leverages existing techniques, these requirements resulted in a number of novel customizations and non-standard design decisions. Our design builds upon familiar node-link network layouts to provide novel techniques for visualizing graph inter-connectivity, supporting visual search and analysis of both network and profile data, customizing spring-embedded layouts, and automatically identifying and visualizing community structures. We have used the resulting system to visualize a 1.5 million member crawl of the popular Friendster service, collected in Winter 2003.

We first summarize related work in social network visualization and share relevant findings from an ethnographic investigation of the Friendster service. We then describe the design of the Vizster visualization in detail, followed by observations from a public installation and informal studies of the system's usage.

2 RELATED WORK

Visualization of social networks has a rich history, particularly within the social sciences, where node-link depictions of social

relations have been employed as an analytical tool since at least the 1930s. Linton Freeman documents the history of social network visualization within sociological research, providing examples of the ways in which spatial position, color, size, and shape can all be used to encode information [8]. For example, networks can be arranged on a map to represent the geographic distribution of a population. Alternatively, algorithmically generated layouts have useful spatial properties: a force-directed layout can be quite effective for spatially grouping connected communities, while a radial layout intuitively portrays network distances from a central actor. Color, size, and shape have been used to encode both topological and non-topological properties such as centrality, categorization, and gender.

In recent years, such approaches have been effectively used in the analysis of domains such as e-mail communication [7], early online social networks [1], and co-authorship networks in scientific publications [16]. There are a number of systems for generating such social network visualizations and performing statistical analyses for the purpose of sociological research, such as UCINet [20], JUNG [12], and GUESS [2].

In addition to sociological research, there have been numerous social visualization projects intended for end-users rather than outside researchers. Though often utilizing techniques seen in analytical domains, these systems present users with visualizations of their own online social world; they are non-anonymous and often perceived with a sense of social place [10] different from more detached analytical environments. Nardi *et al.*'s ContactMap [15] uses spatial grouping and color to redundantly code community groups within a visualization of a user's e-mail contacts. TouchGraph (http://touchgraph.com) uses a force-directed layout to present a network visualization of users of the LiveJournal online community, allowing personal networks to be expanded or contracted by user interaction. BuddyZoo (http://buddyzoo.com) analyzes users' instant messaging (IM) buddy lists to present a static network visualization of their IM contacts. Mutton's PieSpy [14] provides real-time, dynamic visualization of inferred social networks of Internet Relay Chat participants. Viégas et al.'s PostHistory and Social Network Fragments systems [21] visualize personal e-mail archives in both calendar and network views, including rich support for temporal filtering. They found these systems—especially their use in tandem—to be particularly well suited as a memory aid for past contexts and contacts.

Despite this wealth of social network visualization, we believe there is still a need for new designs and techniques, especially as articulated social networks become increasingly common in web services for signifying various kinds of relationship. Visualization of profile attributes unique to online social networks is needed, and techniques for incorporating analytical tools within the simplified domain of end-user visualization may prove useful.

3 FRIENDSTER DESCRIPTION

To better navigate the myriad design decisions we faced, we turned to an ethnographic study of the Friendster service. The following is the result of a 9-month participant-observation during 2003, including interviews, qualitative surveys and focus groups with over 200 Friendster early adopters [4, 6].

Friendster was designed to be an online dating site, complete with profiles, demographic and interest driven search, and a private messaging system. What made Friendster unique was its articulated social networking component and testimonial feature. Users were asked to declare "friends" on the system whose pictures would also appear on the profile when the friends confirmed the relationship. Friends could write testimonials that would also appear on the profile. Both the friends and testimonials were intended to signal additional information about the person's character for those interested in dating the person. Yet, when the early adopters began to use the service, they did not view it as a dating service, but a site where they could gather and communicate with their friends, surf for entertaining profiles and explore public displays of identity and relationships [6].

Early adopters fell primarily into one of three identity groups: gay men, techies/bloggers, and Burners (people who attended the Burning Man art festival in Nevada every September). Each group had a different presentation aesthetic and their profiles conveyed the aspects of their identity most associated with the communities through which they joined the service. For example, Burners usually used their "Playa names" (the nickname they used at the festival), uploaded pictures of themselves at the festival and talked about their music tastes. Many early adopters had densely connected social groups, which aided Friendster's spread. As Friendster became more popular, the user population diversified. That said, the vast majority of users throughout 2003 were in their 20s, college-educated, politically liberal, and living in urban centers. Teenagers did join the service, reporting ages of 69 or 61 and 71 (16 and 17 inverted) because the site forbade users under 18. By marking their age this way, teen users were able to easily find each other. While the service began in the San Francisco Bay Area, its popularity quickly spread to Europe and to Singapore, Malaysia and the Philippines where it took off dramatically. New users primarily came through invitations from active users; thus, the user base represents the extended networks of early adopters.

When users joined the site, they were required to develop a profile that asked for demographic information (age, location, hometown) and a description of tastes (dating preferences, music, books, movies). Users typically created the profile based on the norms of the friends who invited them. As such, the network structure was also apparent in the profile norms of different groups. People regularly updated their profiles, adding new photos, changing tastes dependent on their friends and writing testimonials for their friends in order to encourage reciprocity.

Through ethnographic analysis, boyd [4] uncovered common social practices endemic to Friendster amongst early adopters. While dating did occur, most joined the service because all of their friends joined. Users began using the service by searching for friends and old friends, adding them to their network where appropriate. Because Friendster limited the visibility of profiles to four degrees (friends' friends' friends' friends), users were often motivated to connect to as many people as possible to expand their visibility. Friendster's design decision to list the most highly connected people on the front page further activated the competitive tendency to collect as many people as possible.

In addition to personal profiles, early adopters began creating fake profiles – "Fakesters" – to represent anything from famous media figures like Angelina Jolie to cities like New York, concepts like Love to everyday objects like Salt. Fakesters like Brown University helped connect people who attended that school while Fakesters like Homer Simpson helped connect fans of the TV character. The Fakesters increased the density of the network because they both helped friends find each other and connected new people through shared interests. The profiles created for Fakesters were often quite creative and motivated many users to surf for interesting profiles in a treasure-hunt fashion.

While most users surfed the network by clicking on the friends of other profiles, there was also a search tool available, which helped in finding friends by name and aided in finding potential dates or hookups. Users could also click on interests and find anyone with that interest marked.

While the majority of people used Friendster for social exploration and play, questionable practices were also present. Some users used Friendster to distribute drugs while others used the service to construct fraudulent profiles for public roasting through testimonials. Another problem that emerged was the colliding of networks normally kept apart – not everyone was prepared to expose their networks to both friends and colleagues.

4 Visualization Design

Our goal with Vizster was to build a visualization system that end-users of social networking services could use to facilitate discovery and increased awareness of their online community. We wanted to support the exploratory and playful aspects of Friendster while also giving users easier access to search and group patterns. While users regularly explored the network on Friendster, the linear format limited such explorations. This led us to develop richer network views and exploratory tools, while maintaining a local orientation. We also learned that the use of imagery was indispensable for identifying people and establishing a presentation of self, and so must play a central role in the visualization. In addition to helping support the current practices, we wanted to make sure that Vizster did not eliminate the data that helped users get a sense of people through their profiles. One example is the use of re-appropriated profile fields (*e.g.*, inverting ages to identify teenagers) for coded communication within a sub-population. For this reason, we realized that we must make searchable profile data very present and accessible in the visualization. These goals position Vizster differently from traditional social network visualizations used as analysis tools by social science researchers. The following description includes the implications this approach has had for our design decisions, both in terms of presentation and the level of technical sophistication exposed by the visualization.

Vizster presents social networks using a familiar node-link representation, where nodes represent members of the system and links represent the articulated "friendship" links between them (Figures 1,2). In this view, network members are presented using both their self-provided name and, if available, a representative photograph or image. The networks are presented as egocentric networks: networks consisting of an individual and their immediate friends. Users can expand the display by selecting nodes to make visible others' immediate friends as well. To the right of the network display is a panel presenting a person's profile. As discussed later, the profile panel also provides direct manipulation searches over profile text.

In pursuing this design, we chose to violate Shneiderman's mantra of "overview first, zoom and filter, then details-on-demand" [19], instead opting for a philosophy of "start with what you know, then grow." An overview of the full network is inappropriate in this personal context, as the sheer scale obscures useful landmarks. Users of this system are familiar with their friends, some friends of friends, and various "celebrities." Given a lack of *a priori* knowledge of the user's familiarity with their extended network, starting from an egocentric perspective not only carries less perceptual and computational burden, but guarantees the presence of readily identifiable landmarks for

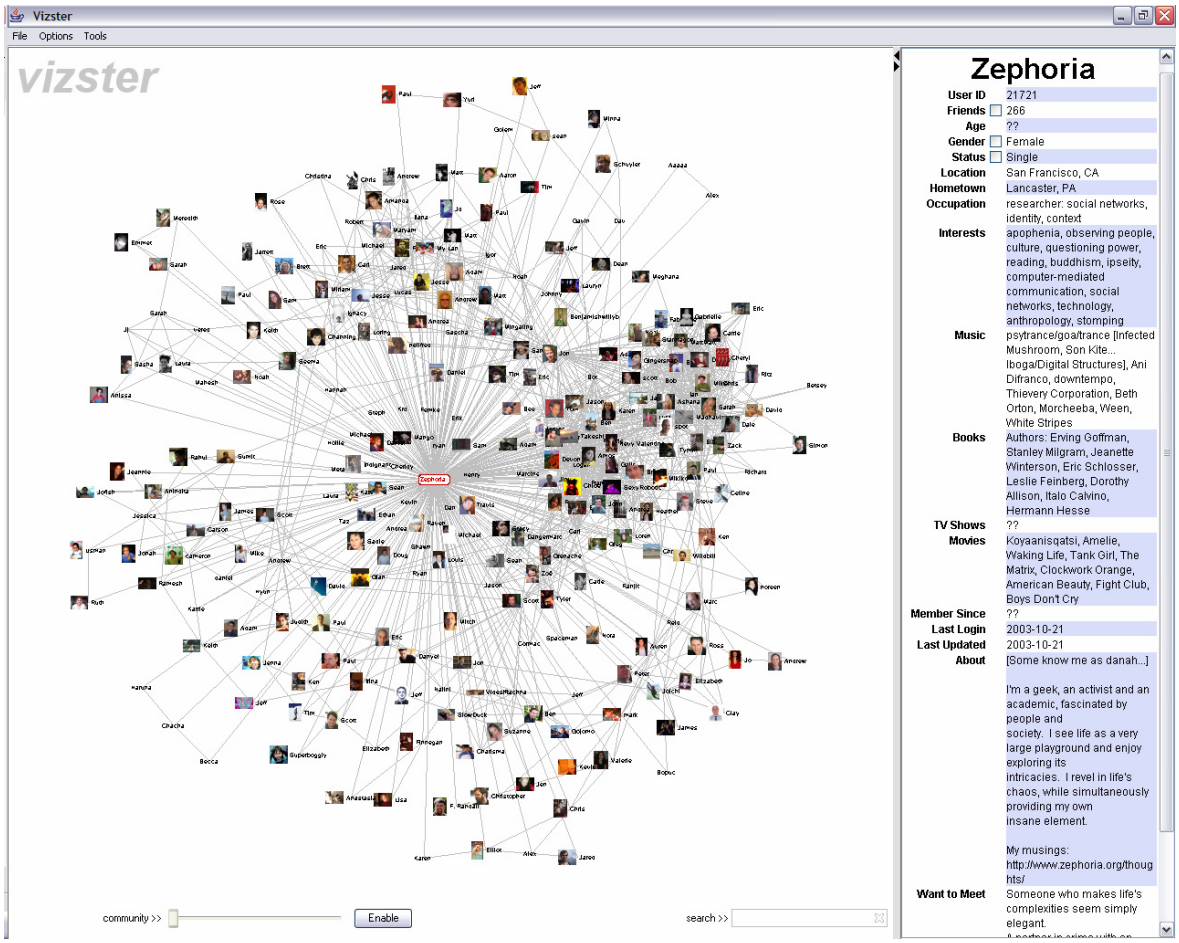


Figure 2: Screen shot of the Vizster visualization system. The left side presents a network display with controls for community analysis and keyword search. The right side consists of a panel displaying a selected member's profile information. Words in the profile panel that occur in more than one profile will highlight on mouse-over; clicking these words will initiate searches for those terms. The checkboxes in the profile panel will initiate an "X-ray" view of that particular profile dimension (see Figures 7-9).

orienting the user: one's self and their immediate friends. From this base, users can selectively expand the network to explore their greater community.

4.1 Layout

Network layout is computed using a spring-embedding (force-directed) algorithm, in which nodes repel each other and edges act as springs. This particular layout has the advantage of grouping users into identifiable communities based upon increased connectivity. The average connectivity of these social networks, with users commonly having fifty to a hundred articulated friendships, renders radial views illegible for anything beyond 1st order networks. Though adjacency matrix-based views have shown promise for visualizing larger networks (at least among very specialized, technical users [9]), their unfamiliarity to our prototypical users and their single degree of freedom for spatially organizing members (thus limiting community groupings) make them less attractive in this particular context.

The layout is computed in real-time, providing an animated layout akin to the Visual Thesaurus (http://visualthesaurus.com) and TouchGraph. The Barnes-Hut algorithm [3] is used to efficiently compute *n*-body (repulsion) forces and numerical integration routines are used to smoothly update screen positions. This avoids interactive delays induced by layout computation and allows the user to participate in the layout process by interactively dragging nodes to tease apart communities. Although items eventually settle into permanent locations, we allow the force simulation to continue running, causing items to maintain some subtle movement. In our observations of usage, this movement has not sparked undue distraction or interference with user actions, but gives the visualization a living or "breathing" feel, connoting social energy and playfulness.

To optimize the layout, we parameterize the tension of the individual spring-edges by node connectivity. Nodes with lower connectivity are given higher tension, causing singly-connected nodes to remain close to expanded nodes and connected communities to assume higher "orbits." This additionally causes nodes with lower connectivity to more closely "tag-along" with their friends, reinforcing their limited connections. We currently assign spring tension proportionally to the inverse logarithm of the degree of the edge's minimally connected node. As discussed later, we also change spring tensions to improve the layout of inferred community groupings.

4.2 Basic Interaction

Basic interaction is done with simple mouse operations. Clicking a node causes the corresponding member profile to appear in the profile panel to the right of the network view. Dragging a node moves it around in the space; upon release it is again subject to the ongoing force-directed layout. Double-clicking a node causes it to either expand or contract, depending on its current state. Upon expansion, the egocentric network for that member is added to the visualization. The expanded node assumes a fixed position (i.e., it is freed from the layout), subject to re-positioning by the user. Double-clicking an expanded node causes that member's egocentric network to disappear and the node again becomes subject to the layout. This mechanism allows users to selectively grow or hide aspects of the network, increasing the scope of the visualization while maintaining the context of already expanded members. The force-directed layout adjusts to these changes, with friends shared between expanded members moving to an intermediate position.

4.3 Exploration: Connectivity Highlighting

To aid network understanding, Vizster highlights nodes based on connectivity in the larger network context. When the mouse hovers over a node, it causes that person, that person's friends, and visible friends-of-friends to highlight. A graded color scale moving from red-orange to orange to yellow—redundantly coding through both intensity and hue—is used to visualize network distance. All other members in the graph are correspondingly desaturated, with images reverting to grayscale, to provide a figure-ground separation between highlighted and non-highlighted nodes (Figure 3). Moving the mouse off the node causes the highlighting to disappear, but with a half-second hysteresis applied to prevent annoying visual "bouncing" as consecutive nodes are visited by the mouse pointer. Clicking and holding a node for 1 second will make the current highlighting persistent (indicated by a brief flashing of the highlighted nodes), after which clicking the background or another node will resume the normal highlighting interaction.

Though readily understandable, egocentric views suffer from a lack of greater network context. To provide cues to the greater network topology, connectivity is computed using the complete backing graph rather than the visualized graph. As a result, people connected to a selected person through intermediate friends will highlight even if those intermediate nodes are not currently visible (Figure 3). These connectivity queries, however, do not pass through expanded nodes (e.g., "Amanda" in Figure 3), as they are obviously intermediate to currently visible nodes, and the goal is to help provide cues to currently obscured connections.

4.4 Exploration: Linkage Views

As expanding a member's egocentric network can significantly change the display, in response to user feedback we have also incorporated linkage views that enable the user to visualize intermediate nodes in a lightweight manner. A user can click a node to select it, mouse over another node of interest, and then tap the space bar to visualize all connections between the selected node and the current mouse target. This causes any intermediate nodes not in the currently expanded networks to also be displayed, providing a complete view of the two-hop connectivity between the nodes (Figure 4).

4.5 Navigation

The network visualization employs a mix of both manual and automated panning and zooming for navigating the space. Panning is performed by dragging the background of the display with the left mouse button down. The display automatically pans when a new node is expanded, centering on the newly expanded network. Manual zooming is performed by holding down the right mouse button and moving the mouse up or down. Simply clicking the right mouse button causes the display to automatically pan and zoom such that the entire visualized network fits within the display. The rendering components update to draw higher resolution photos when zoomed-in to double the normal scale.

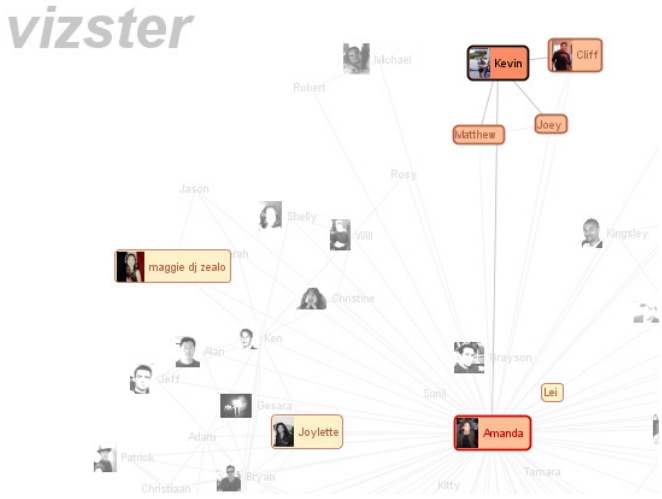


Figure 3: Connectivity Highlighting. The highlighted person "Kevin" is connected to "maggie dj zealo", "Joylette", and "Lei" (all shown in yellow) by friends not currently visible.

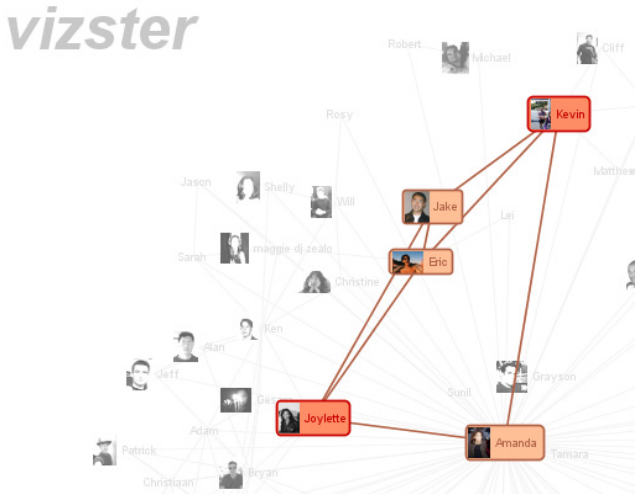


Figure 4: Linkage Views. Clicking on "Kevin" and then tapping the space bar while hovering over "Joylette" (or vice versa) causes all intermediate nodes to be visualized.

In some cases it is desirable to see a particular region while still maintaining the context of a zoomed-out network. Although there exists a rich literature on the use of visual distortion to achieve focus+content views (see [5, 13]), we wanted to introduce such views while minimizing changes to the visual design and dynamics of the application. Our solution was to use geometric distortion within the context of the force-directed layout, "inflating" focal nodes to a larger size, while proportionately increasing their mass values in the layout algorithm to reposition nodes and reduce occlusion (Figure 5). Users initiate such views by holding down the left mouse button on a node for a second, after which the currently highlighted subset of nodes will inflate.

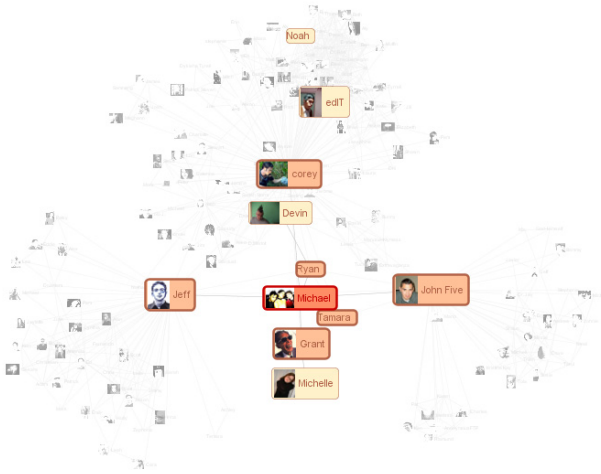


Figure 5: Focus+Context view by inflating highlighted nodes. To reduce occlusion, node mass values increase proportionally to changes in size.

4.6 Search

Vizster also supports keyword search of the visualized network. As items become visible, their profile attributes are added to a single in-memory search index. A search box allows users to type in search queries. Search hits are indicated by "auras" around matching nodes (Figure 6), with non-matching nodes again de-emphasized through desaturation. The search auras are presented in a dark purplish color, intended to leverage blue-yellow color opposition [18] to balance against the hues of the highlighting.

As users mouse over the network with search auras present, highlighting is performed as usual, desaturating the text and images of non-highlighted search hits. The search auras themselves remain visible, however, to balance competing search and connectivity queries. Search auras for highlighted nodes also grow slightly larger to maintain visibility.

Additionally, users can issue search queries from the profile panel in a direct manipulation fashion. As the mouse pointer moves over specific words, the word will highlight if there are at least 2 matching search hits for that term. Clicking the word will then execute a search, with the text of the query appearing in the editable search box. Double clicking a word will initiate a search for the surrounding phrase, where, following Friendster's convention, a "phrase" is any comma-delimited set of words. Manually selecting text by mouse drag similarly executes a search for the selected text.

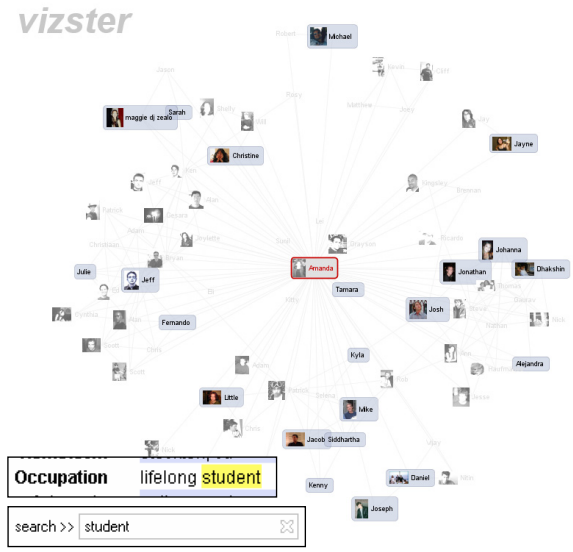


Figure 6: Visualized search results for the query "student". Searches are executed by clicking text in the profile panel or typing in the search box.

4.7 Attribute Visualization using "X-Ray" mode

In addition to generating search queries, the profile panel also includes controls to further visualize the individual profile

attributes of members. By clicking on a check box next to an attribute within the profile panel, Vizster will enter an "X-ray" mode visualizing attribute values such as age, number of friends, gender, relationship status, or time since last login. The background of the display turns black to clearly indicate the modal state, member photos are removed, and node color is used to visualize the currently selected attribute. Additionally, a color legend appears in the upper-right corner of the display. In this state, all the other operations discussed still function as normal. Highlighting and search still desaturate non-matching items, though node colors no longer change under highlighting.

Figure 7 shows Vizster in X-ray mode visualizing the number of friends each person has in the service. Figure 8 shows the use of X-ray mode to visualize each member's gender. Figure 9 displays the results of including a mouse-over highlight, and search results for the query "student." Community groupings (discussed below) are shown, but consistent with the X-ray metaphor are reduced to simple outlines.

4.8 Visualizing Community Structure

Although the force-directed layout does an admirable job of clustering the network into communities based on linkage, Vizster also supports explicit visualization of community structures. As manual creation of large communities can prove somewhat tedious, Vizster includes tools for the automatic determination of community structure. For this task, we have used Newman's community identification algorithm [17], which identifies group structures based solely on link analysis. We chose this particular algorithm as it provides useful topology-based groupings fast enough to support real-time interaction. The algorithm is a variant of hierarchical agglomerative clustering, first placing each node in its own community, then greedily merging groups based upon a metric that attempts to maximize within cluster linkage while minimizing between cluster linkage. The clustering assigns members to unique communities. In future work, it may be useful to generalize the underlying algorithm to support soft or "fuzzy" clustering techniques permitting multiple membership in communities as well as users to manually refine clustering results.

Inferred community groupings of two or more nodes are visibly represented as "blobs" surrounding community members, taking advantage of low spatial frequencies to make community structures apparent (Figure 10). To further improve the layout, extra-community edges are given weaker spring forces, promoting the spatial separation of inferred communities. The smooth contours of the blobs are computed by taking the bounding coordinates of constituent members, calculating the convex hull of the resulting set of points, and then interpolating the hull boundary using a cardinal spline. As members within a community are moved about, the blob boundaries adjust smoothly for a dynamic, playful character. Blobs are translucently rendered, with colors determined by uniformly sampling the hue dimension of HSB color space [18]. This semi-transparency will allow the blobs to act as Venn diagrams once the application supports assignment of nodes to multiple community groupings.

Users can instigate automated community analysis by clicking the community analysis button at the bottom-left of the display. The community structure algorithm is then run on the adjacency matrix of the currently visible network. Currently expanded nodes are omitted from the input matrix to avoid unduly collapsing communities—the goal is to show community relations *surrounding* the expanded nodes. The results of the community analysis are then visualized using communities identified at the optimal value of the clustering metric.

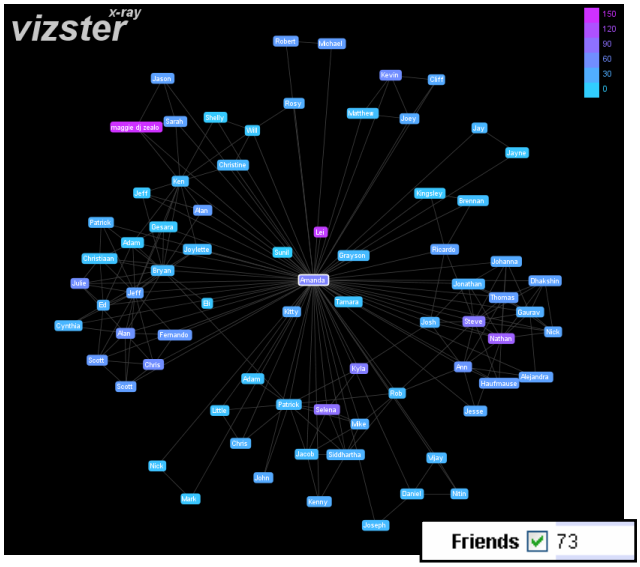


Figure 7: X-ray mode visualizing the number of friends.

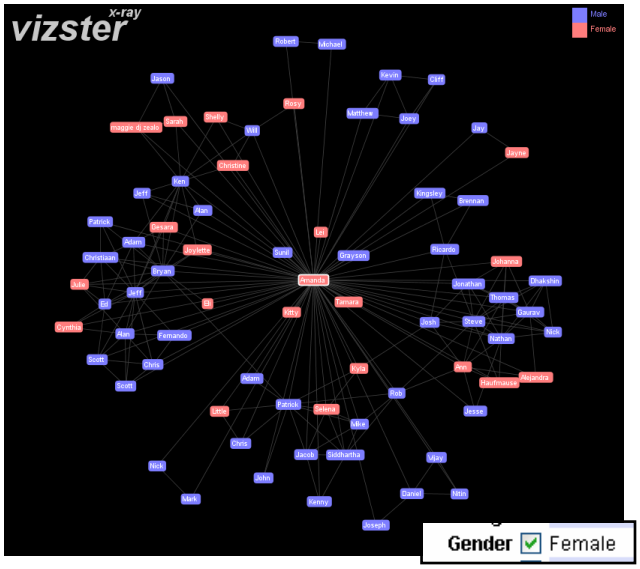


Figure 8: X-ray mode visualizing gender.

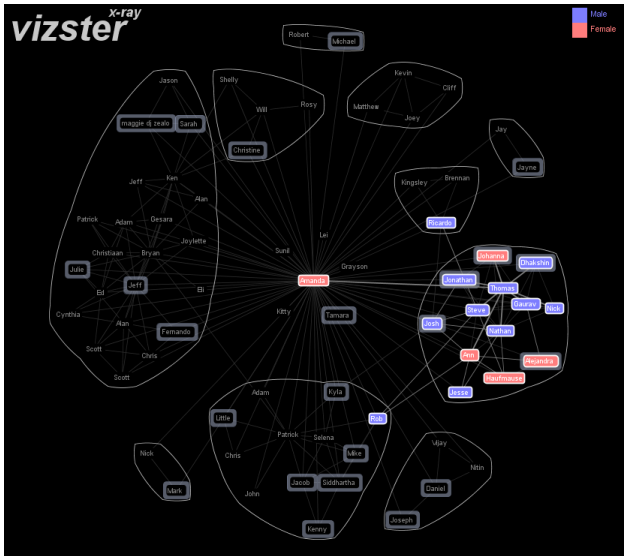


Figure 9: X-ray Mode visualizing genders, search hits, mouse-over highlight and community structures.

Our design hides the complexity of the algorithm from users, letting them explore these data mining results through simple widgets and visual analysis. Like most data mining methods, the community analysis is imperfect, and may identify communities at higher or lower granularities than those desired by the user or make assignments otherwise perceived as problematic. To help combat this, a community slider is provided to explore the various states of the clustering (Figures 10 and 11). Moving the slider to the far left reverts the display to the initial state of the clustering, while moving the slider progressively to the right reveals each merge performed by the algorithm. Thus the slider allows the user to interactively explore clustering states by moving through progressive slices of the computed cluster tree. In practice, we have observed users employ the slider until the communities "look right" to them. The current community visualization persists when additional nodes are expanded or contracted; when the visible network changes, an update button is added to the control panel, allowing users to re-run the community analysis.

4.9 Summary

In summary, the Vizster design constitutes a visual environment for the exploration and analysis of online social networks, including both topological and profile data. The scale of displayed information and layout were chosen to support observed behavior and capabilities, and allow users to expand visualized networks while maintaining landmarks. Interactive highlighting is used to explore friendship relations and unearth "hidden" connections in the larger network structure. Panning, zooming, and distortion techniques are provided to help users navigate visualized networks. Interactive search and attribute visualization ("X-ray" mode) enable visual exploration of member profile data. Finally, visual community analysis is provided to help users construct and explore higher-level structures of their online communities.

5 Implementation Notes

Vizster was written in Java using the prefuse visualization toolkit [11], leveraging the toolkit's filtering, layout, rendering, and image management support. We also wrote extensions for database connectivity and to perform connectivity highlighting and linkage views. To support keyword search, we also integrated the Lucene search engine (http://lucene.apache.org) into the prefuse framework. Network and profile data for the visualization were collected using a custom web crawler and stored in a backing MySQL database. To support highlighting connectivity queries and faster expansion response times, 2nd order networks are loaded from the database upon expansion of nodes, though only the 1st order networks are immediately visualized. This brings increased memory requirements, but have not proved limiting. Source code (but not collected data) for the application is available from http://prefuse.sourceforge.net.

6 Usage Observation

To evaluate and further guide the design of our visualization, we observed usage in two environments: a public installation at a large party and an informal laboratory setting. While we were certainly interested in gauging Vizster's utility and usability, we were also interested in larger patterns of discovery—finding unknown people, connections, communities of relevance—and in people's social and affective reactions to the visualization.

Our first observation of usage was conducted around an interactive installation at a 500-person all-night event in San Francisco. Many of the party-goers included early adopters of the Friendster system, especially those affiliated with the Burning

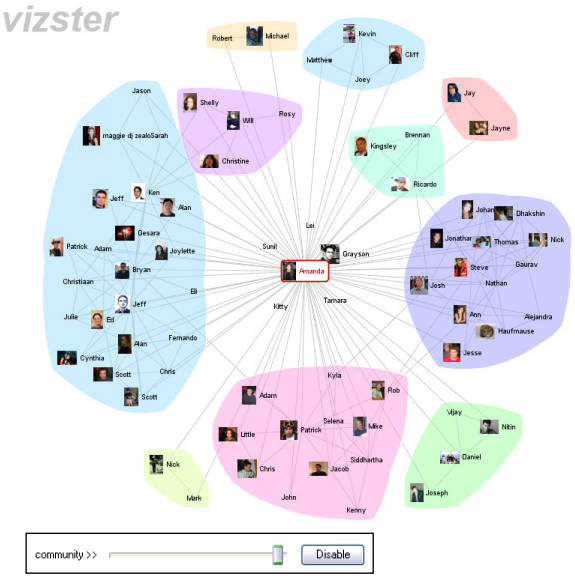


Figure 10: Community structure visualization using algorithmically determined optimum.

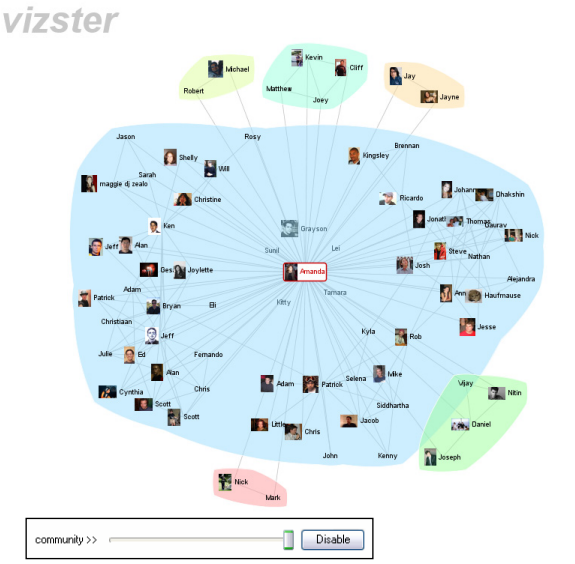


Figure 11: Community structure visualization after the community slider has been dragged to the right.

Man festival and tech culture. The installation consisted of an interactive kiosk and a projection of the visualization onto a large screen. Throughout the night, we observed usage of the system by over a hundred users, noting the reactions of users and onlookers.

We also observed usage in an informal laboratory setting. Participants consisted of 5 males and 1 female, all Friendster members in their early-to-late twenties. To maintain ecological validity, we did not provide users with any tutorials or pre-specified tasks. Instead, we simply asked them to play with the system, explore it as they saw fit, and talk-aloud about their experiences as they did so. After 15 minutes of exploration, we then provided users with a one page reference to the various controls, to ensure that we could gauge user reaction to otherwise undiscovered features. After 10 more minutes of observation, we interviewed participants about their experience and opinions.

Usage, especially within the party setting, was routinely coupled with some form of social play—for example, games to

see if one could hunt down specific people they knew. Onlookers tended to participate as well, engendering a shared experience around the visualization. Most users and observers began retelling stories about people, their relationships, and shared experiences. Users would also play with the various interactive features, often without pursuing a specific task. In particular, users enjoyed zooming navigation and experimenting with the real-time layout, whether dragging items around to tease clusters apart or dragging items to build up spring tensions and then gleefully watch the recoil of the network upon release.

Through experimentation, most party-goers and study participants quickly made sense of the connectivity highlighting. Often times, a lone node would highlight far away from the current locus of interaction, eliciting comments such as "Why did that light up?" This would then trigger exploration, with users further expanding the network to discover the intermediary friends that were not yet visible. Users (or others also watching the visualization) would then answer their own question: "Oh, it's telling you that they are connected in some way."

Similarly, users in both observation contexts enjoyed experimenting with the community analysis. Users would play with the slider to get a feel for how the feature worked, and then settle with a configuration that "looked right" to them. Nearly everyone then began retelling stories about the various communities and their members. Prototypical comments included "Look, it found all the people from Harvard" and "It was smart enough to know not to group them together, they hate each other." In general, people tended to impart a higher level of sophistication to the community analysis than is implemented; some remain convinced that it compares members' profile data. Though they often parameterized the analysis using the slider, users seemed to discount their own input into this semi-automatic process.

Users easily discovered the X-ray mode, often commenting on the results of their ad-hoc analyses (*e.g.*, "What!? She's not single!"). It proved particularly powerful when coupled with searches, as users naturally specified conjunctive queries by running search queries on top of the X-ray state. As we expected, however, users in both contexts showed a strong preference for imagery for navigating the networks, for example saying "bring the pictures back" when they wanted to hunt for someone else.

The lab context also provided valuable feedback on various features. In particular, linkage views were introduced in response to early user requests for a means of quickly unearthing connections without expanding another network. However, we found that both linkage views and the ability to inflate nodes were not discovered through experimentation, though were appreciated after consulting the reference. One participant suggested adding mechanisms for opportunistically suggest features to users—an approach often successful in video games.

In all, we found that the visualization was used by users to both explore and play with their networks—expanding the network to quite large depths, performing visual analyses, and exploring community structures while simultaneously engaging in social narratives. Although more formal comparative and longitudinal studies would be required to more fully assess the utility of the individual features introduced, the engaged yet playful exploration and discovery we observed helps validate a number of our design decisions. As one study participant put it, "Friendster gives you your two hours of fun, and this doubles it."

7 Conclusion

This paper presents a case study of the design of Vizster, a visualization system for end-user exploration of online social networks. Vizster is the result of our own experiment in applying the results of ethnographic inquiry to the design of a domain-specific interactive visualization. The rich understanding of user behaviors, motivations, and aptitudes provided by ethnographic techniques guided our subsequent process of designing and customizing visualization and interaction techniques within an end-user context more characteristic of play than analysis. Resulting techniques include connectivity highlighting and linkage views for viewing network context, X-ray mode and profile search for exploring member profile data, and visualization of inferred community structures. Observations of usage indicate that these features proved both useful and conducive to social engagement. In future work, we are interested in applying these techniques in a more traditional analytical context and conducting more rigorous evaluations. As forms of online sociality continue to spread and diversify, we also anticipate the need for additional design approaches balancing utility and holistic experience, enabling information visualization technologies to take on an increasingly important role in both mediating and making sense of our shared social landscape.

Referernces

[1] Adamic, A., O. Buyukkokten, E. Adar. A Social Network Caught in the Web. *First Monday*, 8(6), 2003.

[2] Adar, E. GUESS: The Graph Exploration System. http://www.hpl.hp.col/research/idl/projects/graphs

[3] Barnes, J., P. Hut. A Hierarchical O(N log N) Force Calculation Algorithm. *Nature*, 324, 4 December 1986.

[4] boyd, d. Friendster and Publicly Articulated Social Networks. *CHI 2004*, Vienna, Austria, 1279-1282.

[5] Carpendale, M.S.T., and C. Montagnese. A Framework for Unifying Presentation Space. *UIST 2001*, Orlando, FL, 61-70

[6] Donath, J., d. boyd. Public displays of connection. *BT Technology Journal*, 22(4), October 2004, 71-82.

[7] Fisher, D., P. Dourish. Social and temporal structures in everyday collaboration. *CHI 2004*, Vienna, Austria, 551-558.

[8] Freeman, L. Visualizing Social Networks. *Journal of Social Structure*, 1, 2000.

[9] Ghoniem, M., J.-D. Fekete, P. Castagliola. A Comparison of the Readability of Graphs Using Node-Link and Matrix-Based Representations. *InfoVis 2004*, Austin, TX, 17-24.

[10] Harrison, S., P. Dourish. Re-place-ing Space: The Roles of Place and Space in Collaborative Systems. *CSCW 1996*, Boston, MA, 67-76.

[11] Heer, J., S.K. Card, J.A. Landay. prefuse: A Toolkit for Interactive Information Visualization. *CHI 2005*, Portland, OR, 421-430.

[12] JUNG: Java Universal Network/Graph Framework. http://jung.sf.net/

[13] Leung, Y.K., M.D. Apperley. A Review and Taxonomy of Distortion-Orineted Presentation Techniques. *ACM TOCHI*, 1(2), 1994.

[14] Mutton, P. Inferring and Visualizing Social Networks on Internet Relay Chat. *InfoVis 2004*, Austin, TX, 35-43.

[15] Nardi, B., S. Whittaker, E. Isaacs, M. Creech, J. Johnson, J. Hainsworth. ContactMap: Integrating Communication and Information Through Visualizing Personal Social Networks. *Communications of the ACM*, 45(4), April 2002, 89-95.

[16] Newman, M.E.J. Co-authorship networks and patterns of scientific collaboration. *Proc. Natl. Acad. Sciences*, 101, 2004, 5200-5205.

[17] Newman, M.E.J. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69, 2004, 066133.

[18] Palmer, S. *Vision Science: Photons to Phenomenology*. MIT Press. Cambridge, MA. 1999.

[19] Shneiderman, B. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. *IEEE Visual Languages*, 1996.

[20] UCINET: Social Network Analysis Software. http://analytictech.com/

[21] Viégas, F., d. boyd, D. Nguyen, J. Potter, and J. Donath. Digital Artifacts for Remembering and Storytelling: PostHistory and Social Network Fragments. *HICSS-37*, Big Island, HI, 2002.

Graph Drawing Tutorial

Isabel F. Cruz

Worcester Polytechnic Institute

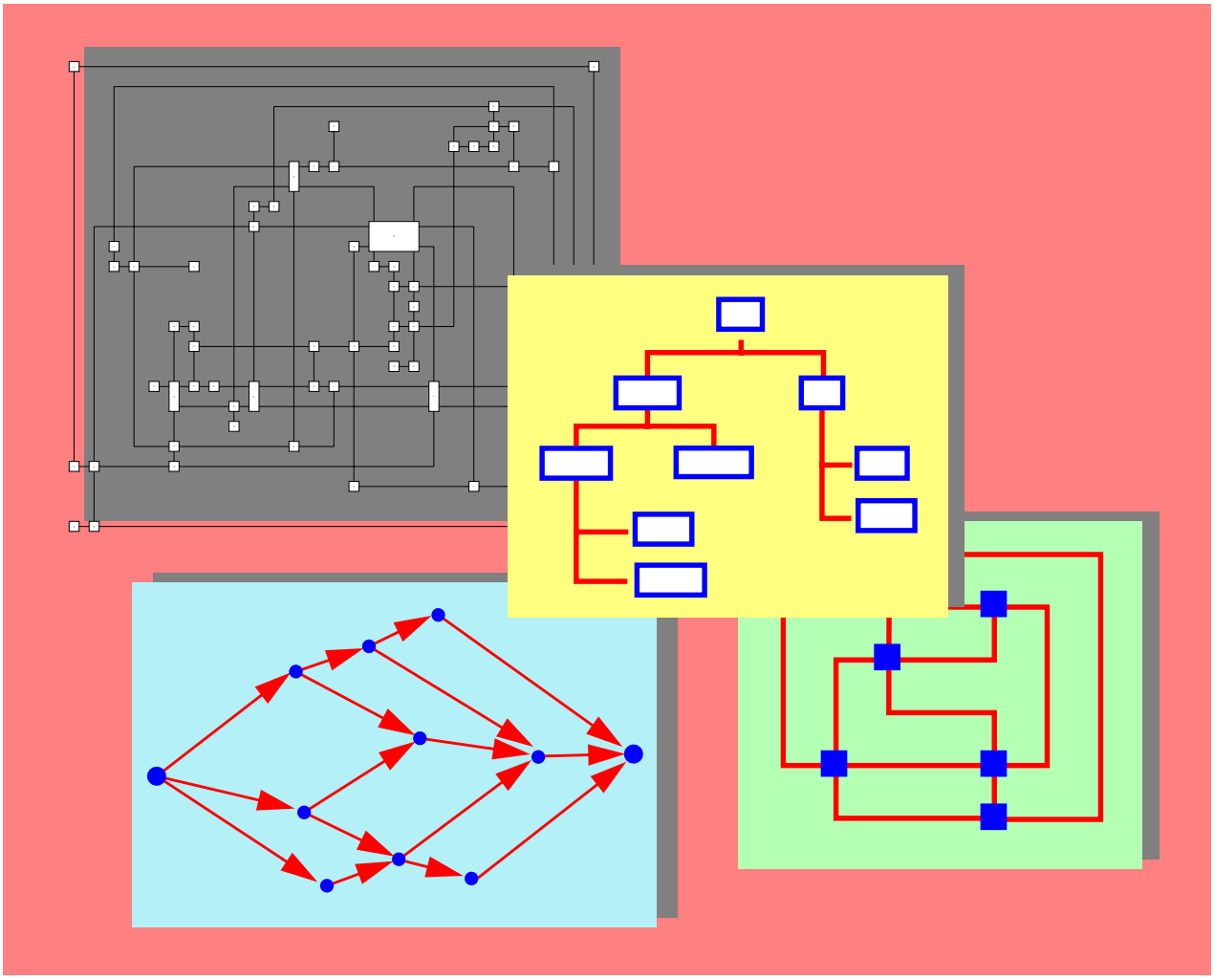
Roberto Tamassia

Brown University

Introduction

Graph Drawing

- models, algorithms, and systems for the visualization of ***graphs*** and ***networks***

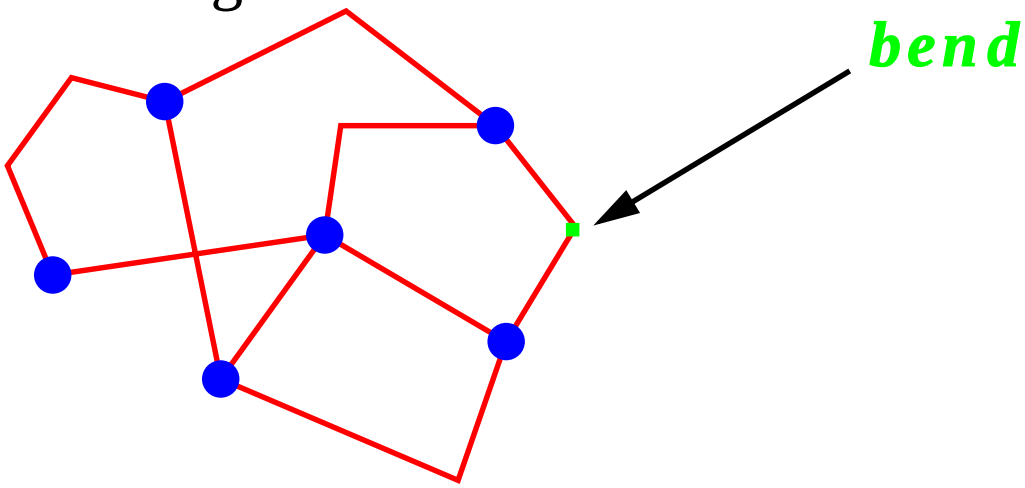


- applications to ***software engineering*** (class hierarchies), ***database systems*** (ER-diagrams), ***project management*** (PERT diagrams), ***knowledge representation*** (isa hierarchies), ***telecommunications*** (ring covers), ***WWW*** (browsing history) ...

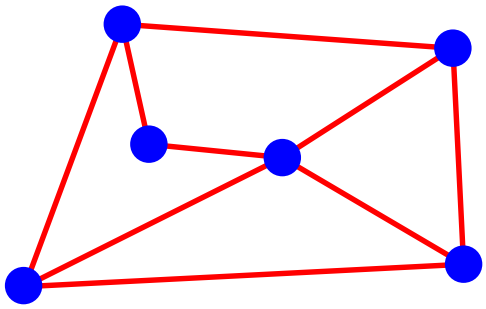
Drawing Conventions

- ***general constraints*** on the geometric representation of vertices and edges

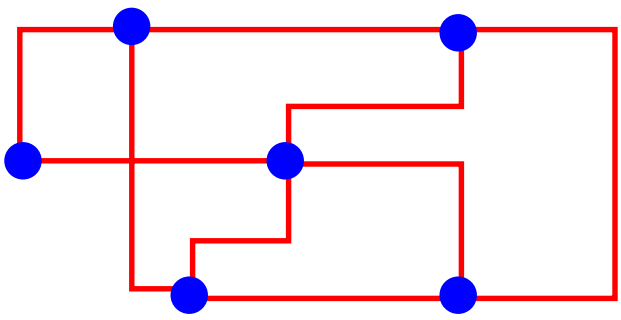
polyline drawing



planar straight-line drawing

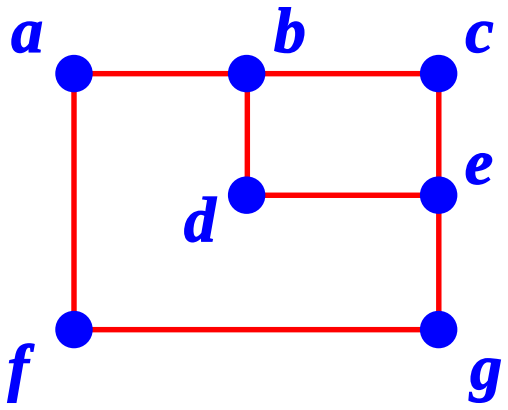


orthogonal drawing

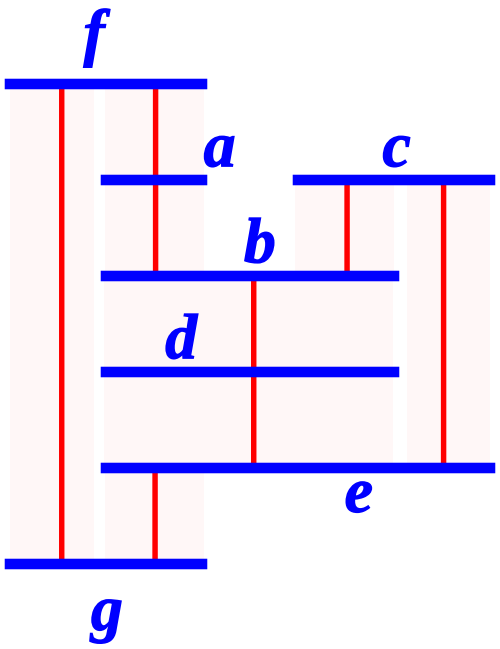


Drawing Conventions

planar othogonal straight-line drawing

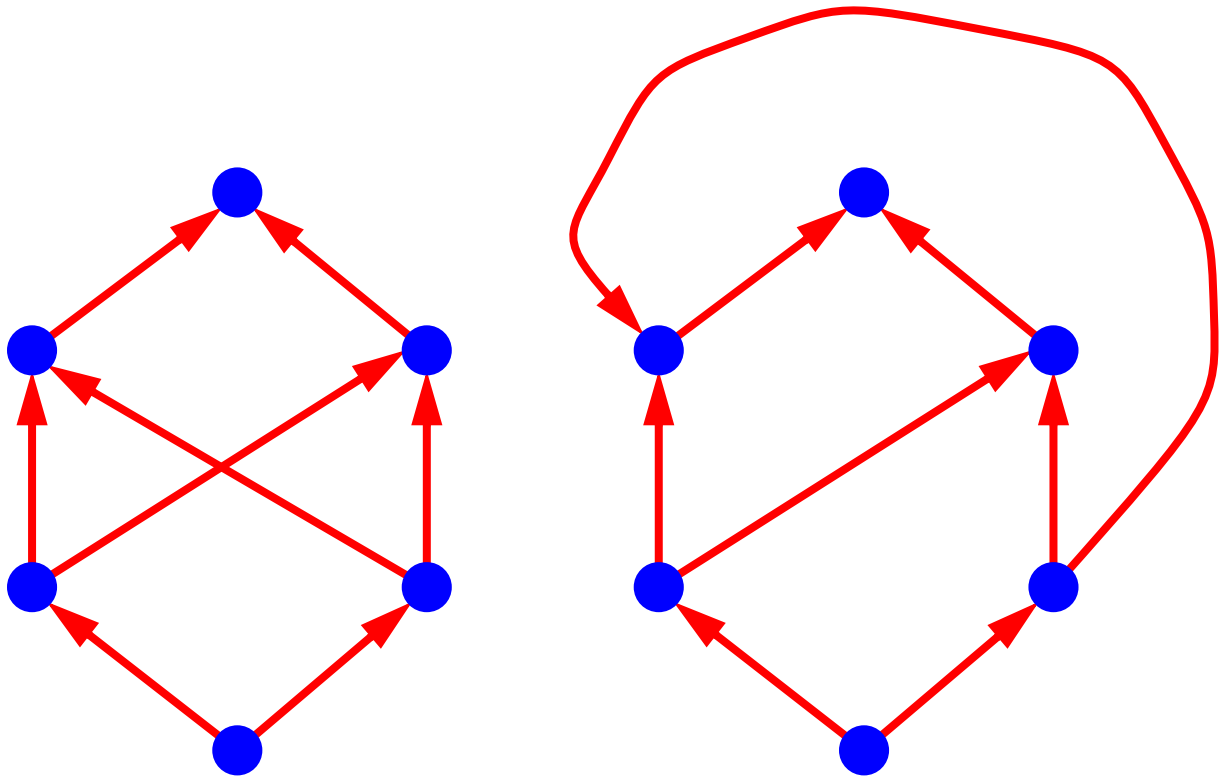


strong visibility representation



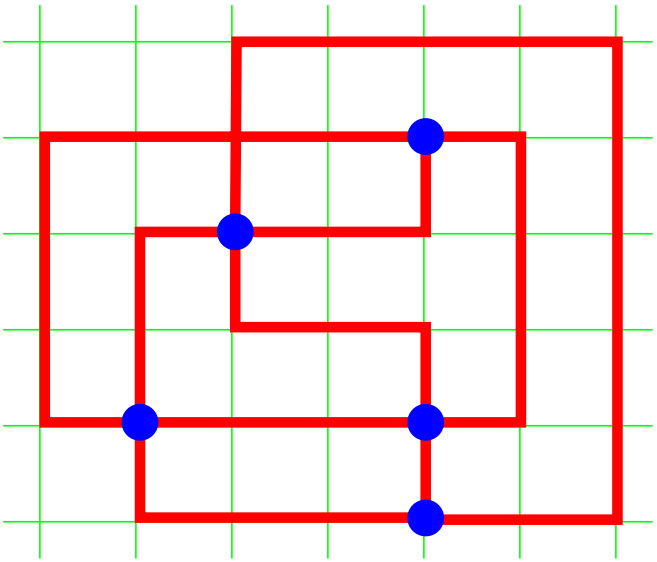
Drawing Conventions

- directed acyclic graphs are usually drawn in such a way that all edges “flow” in the same direction, e.g., from left to right, or from bottom to top
- such ***upward drawings*** effectively visualize hierarchical relationships, such as covering digraphs of ordered sets
- not every planar acyclic digraph admits a planar upward drawing



Resolution

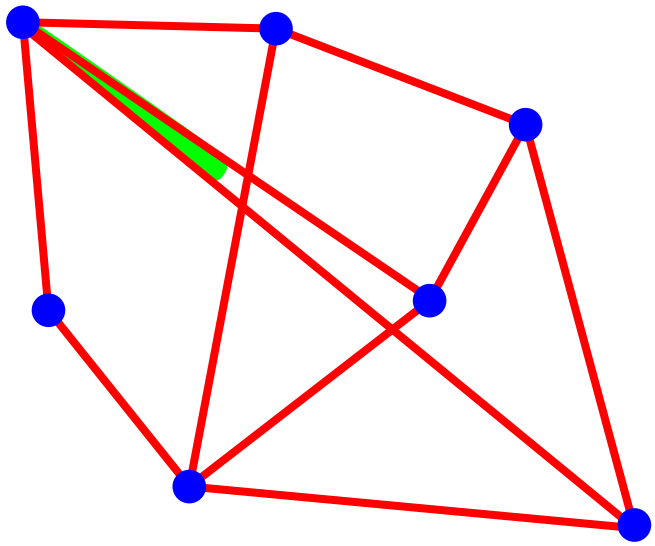
- display devices and the human eye have finite resolution
- examples of ***resolution rules***:
  - integer coordinates for vertices and bends (***grid*** drawings)



  - prescribed minimum distance between vertices
  - prescribed minimum distance between vertices and nonincident edges
  - prescribed minimum angle formed by consecutive incident edges (***angular resolution***)

Angular Resolution

- The ***angular resolution*** ρ of a straight-line drawing is the smallest angle formed by two edges incident on the same vertex



- ***High angular resolution*** is desirable in ***visualization*** applications and in the design of ***optical communication*** networks.
- A ***trivial upper bound*** on the angular resolution is

$$\rho \leq \frac{2\pi}{d}$$

where d is the maximum vertex degree.

Aesthetic Criteria

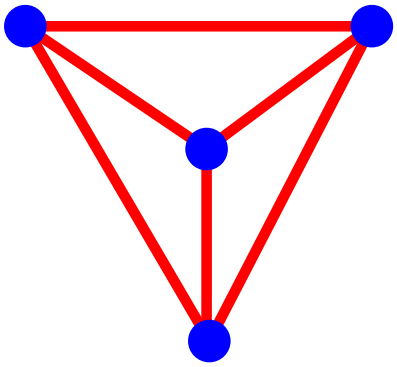
- some drawings are better than others in conveying information on the graph
- ***aesthetic criteria*** attempt to characterize readability by means of general ***optimization*** goals

Examples

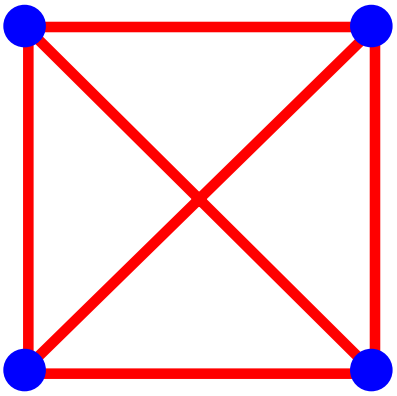
- minimize ***crossings***
- minimize ***area***
- minimize ***bends*** (in orthogonal drawings)
- minimize ***slopes*** (in polyline drawings)
- maximize ***smallest angle***
- maximize display of ***symmetries***

Trade-Offs

- in general, one cannot simultaneously optimize two aesthetic criteria



min # crossings

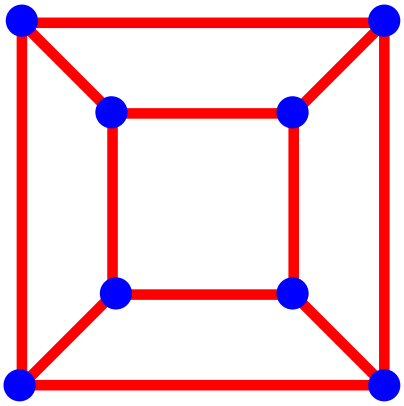


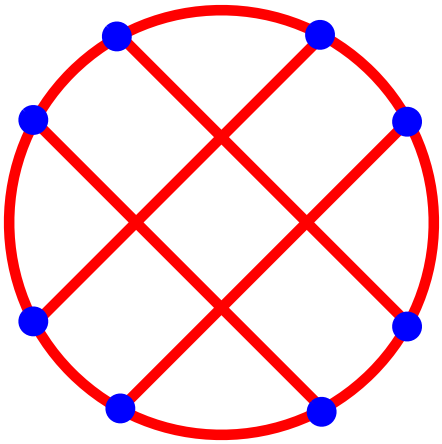
max symmetries

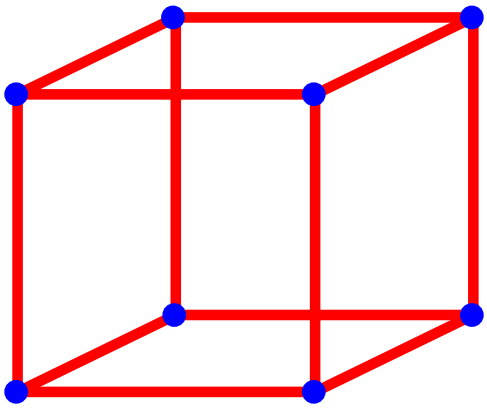
Complexity Issues

- testing planarity takes linear time
- testing upward planarity is NP-hard
- minimizing crossings is NP-hard
- minimizing bends in planar orthogonal drawing:
  - NP-hard in general
  - polynomial time for a fixed embedding

Beyond Aesthetic Criteria







Constraints

- some readability aspects require knowledge about the ***semantics*** of the specific graph (e.g., place “most important” vertex in the middle)
- ***constraints*** are provided as additional input to a graph drawing algorithm

Examples

- place a given vertex in the “middle” of the drawing
- place a given vertex on the external boundary of the drawing
- draw a subgraph with a prescribed “shape”
- keep a group of vertices “close” together

Algorithmic Approach

- Layout of the graph generated according to a prespecified set of aesthetic criteria
- Aesthetic criteria embodied in an algorithm as optimization goals. E.g.
  - minimization of crossings
  - minimization of area

Advantages

- Computational ***efficiency***

Disadvantages

- User-defined ***constraints*** are not naturally supported

Extensions

- A limited constraint-satisfaction capability is attainable within the algorithmic approach

  E.g., [Tamassia Di Battista Batini 87]

Declarative Approach

- Layout of the graph specified by a ***user-defined*** set of ***constraints***
- Layout generated by the ***solution*** of a ***system*** of constraints

Advantages

- ***Expressive power***

Disadvantages

- Some natural aesthetics (e.g., planarity) need ***complicated*** constraints to be expressed
- General constraint-solving systems are computationally ***inefficient***
- Lack of a powerful language for the specification of constraints (currently done with a detailed enumeration of facts, or with a set notation)

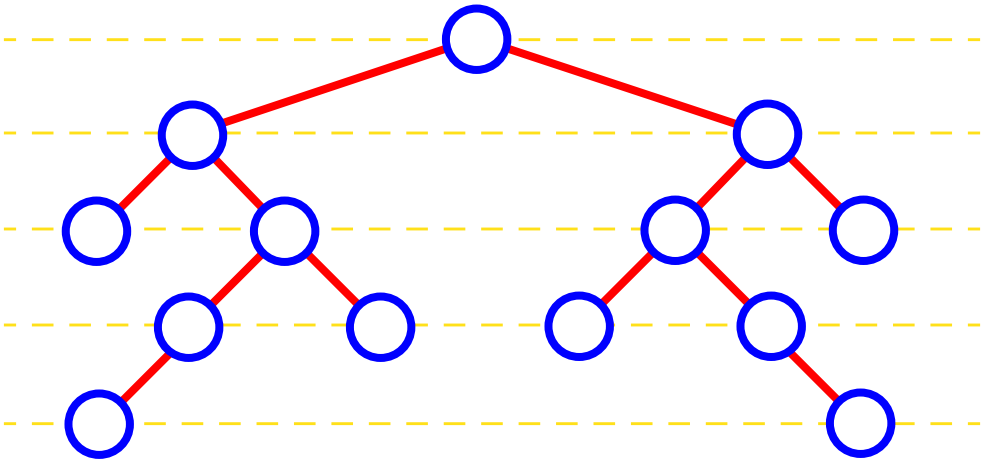
Getting Started with Graph Drawing

- Book on Graph Drawing by G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, ISBN 0-13-301615-3, *Prentice Hall*, (available in August 1998).
- Roberto Tamassia's WWW page
  http://www.cs.brown.edu/people/rt/
- Tutorial on Graph Drawing by Isabel Cruz and Roberto Tamassia (about 100 pages)
- Annotated Bibliography on Graph Drawing (more than 300 entries, up to 1993) by Di Battista, Eades, Tamassia, and Tollis. *Computational Geometry: Theory and Applications*, 4(5), 235-282 (1994).
- Computational Geometry Bibliography
  www.cs.duke.edu/~jeffe/compgeom/biblios.html
  (about 8,000 BibTeX entries, including most papers on graph drawing, updated quarterly)
- Proceedings of the Graph Drawing Symposium (Springer-Verlag, LNCS)
- Graph Drawing Chapters in:
  CRC Handbook of Discrete and Computational Geometry
  Elsevier Manual of Computational Geometry

Trees

Drawings of Rooted Trees

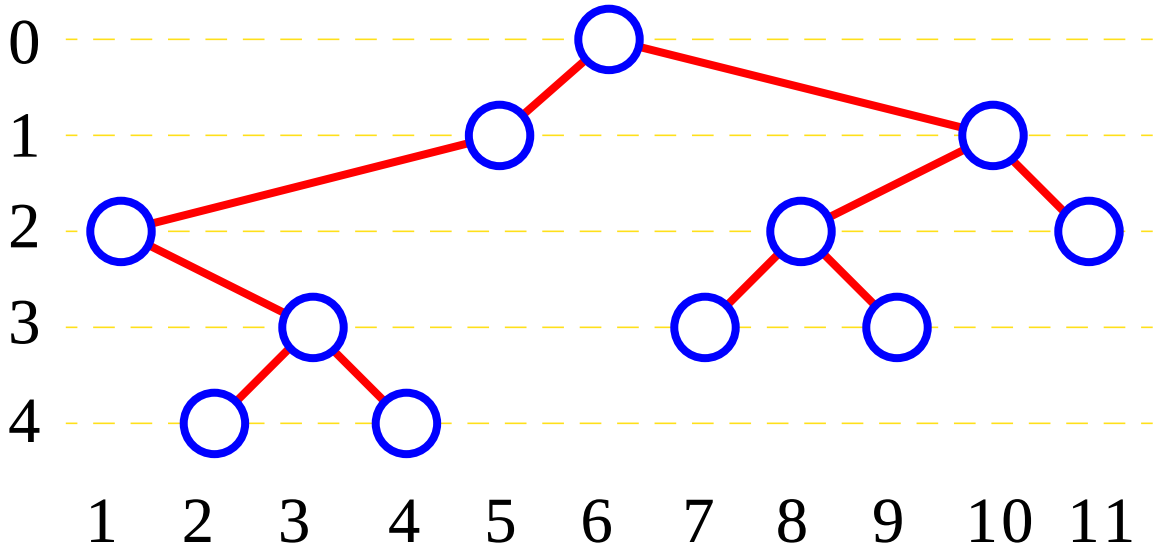
- the usual drawings of rooted trees are ***planar***, ***straight-line***, and ***upward*** (parents above children)
- it is desirable to minimize the ***area*** and to display ***symmetries*** and ***isomorphic subtrees***
- ***level drawing***: nodes at the same distance from the root are horizontally aligned



- level drawings may require $\Omega(n^2)$ ***area***

A Simple Level Drawing Algorithm for Binary Trees

- y(v) = distance from root
- x(v) = inorder rank

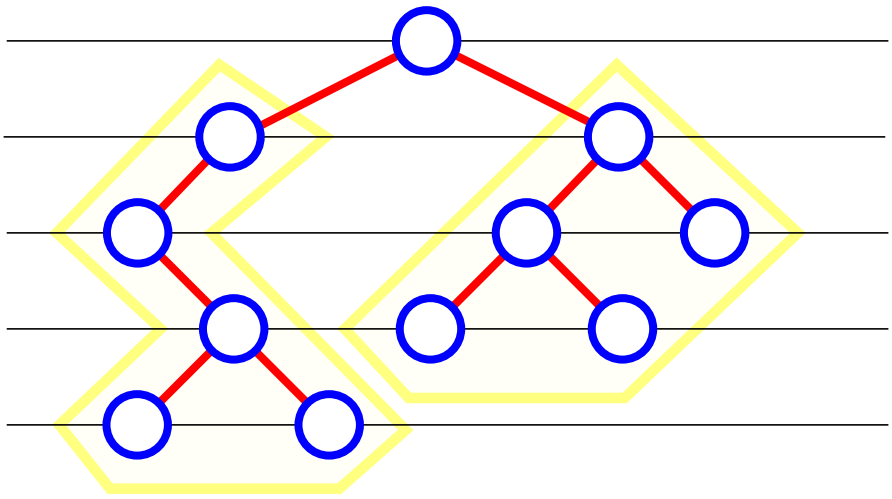


- level grid drawing
- display of symmetries and of isomorphic subtrees
- parent in between left and right child
- parents not always centered on children
- width = n – 1

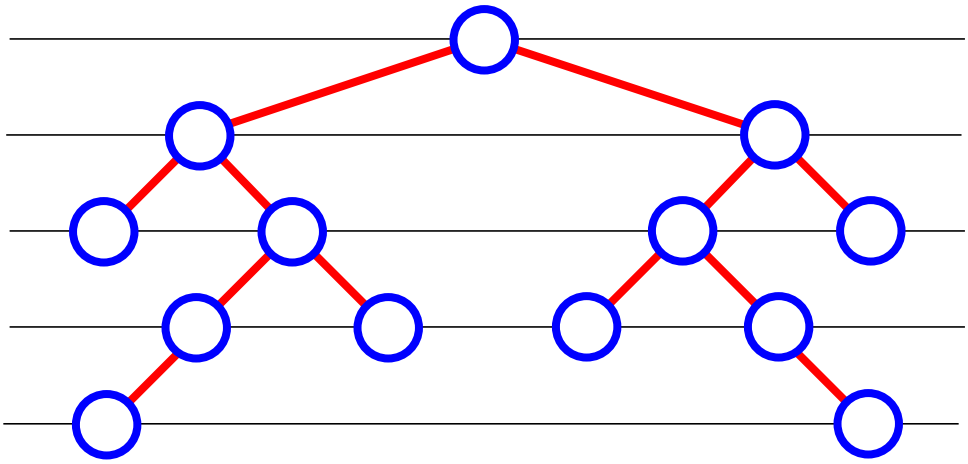
A Recursive Level Drawing Algorithm for Binary Trees

[Reingold Tilford 1983]

- draw the left subtree
- draw the right subtree
- place the drawings of the subtrees at horizontal distance 2
- place the root one level above and half-way between the children
- if there is only one child, place the root at horizontal distance 1 from the child

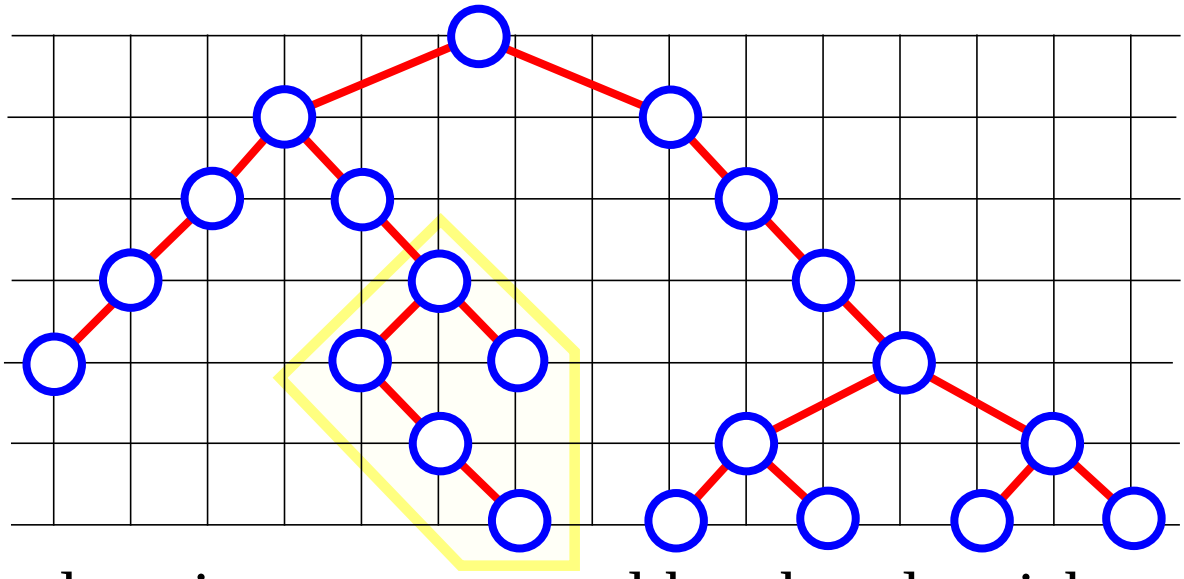


Properties of Recursive Level Drawing Algorithm for Binary Trees

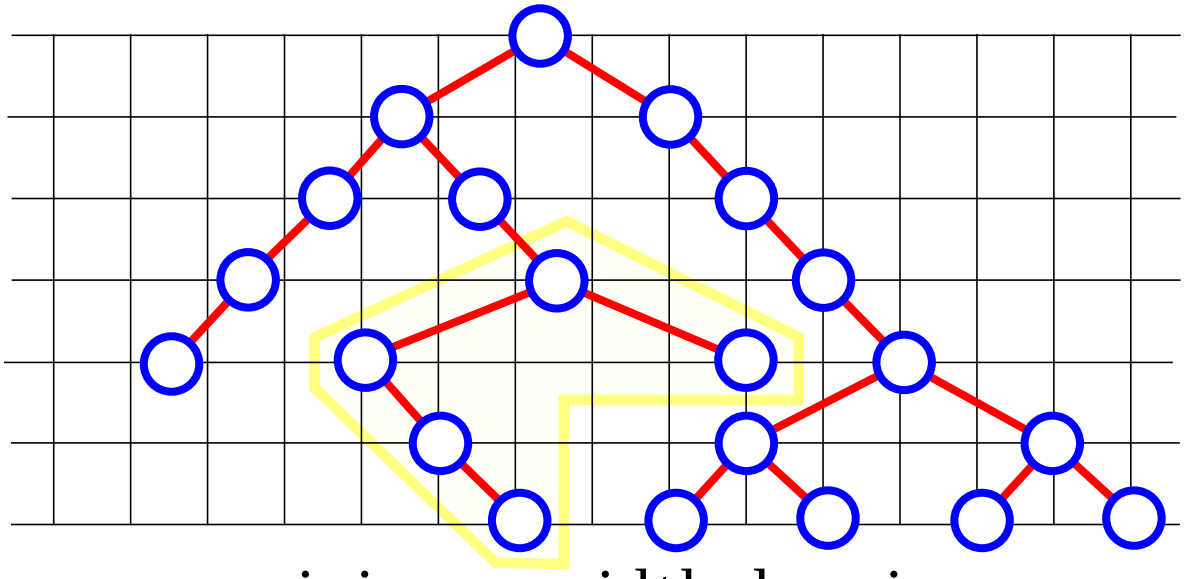


- ***centered*** level drawing
- "small" width
- display of symmetries and of isomorphic subtrees
- can be implemented to run in O(n) time
- can be extended to draw general rooted trees (e.g., root is placed at the average x-coordinate of its children)

Non Optimality of Recursive Tree Drawing Algorithm



drawing constructed by the algorithm

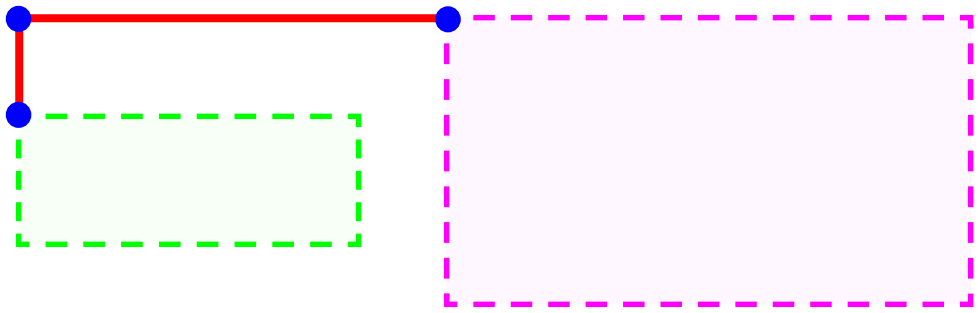


minimum width drawing

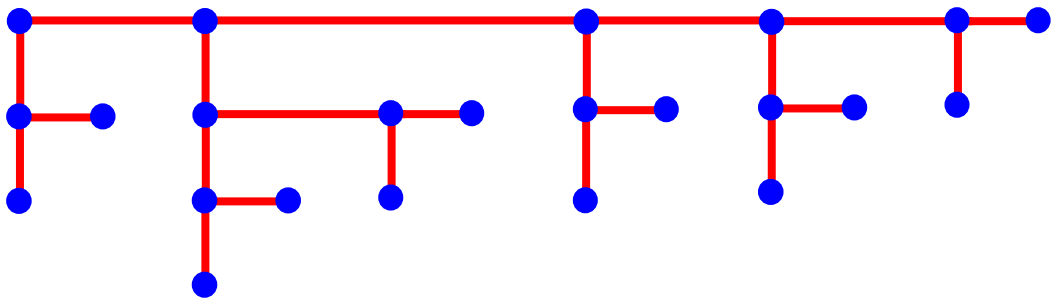
- minimizing the width is NP-hard if integer coordinates are required

Area-Efficient Drawings of Trees

- planar straight-line orthogonal upward grid drawing of a binary tree with ***O(n log n) area***, ***O(n) width***, and ***O(log n) height***
  [Crescenzi Di Battista Piperno 92]
  [Shiloach 76]
- draw the ***largest subtree*** "to the right" and the ***smallest subtree*** "below"

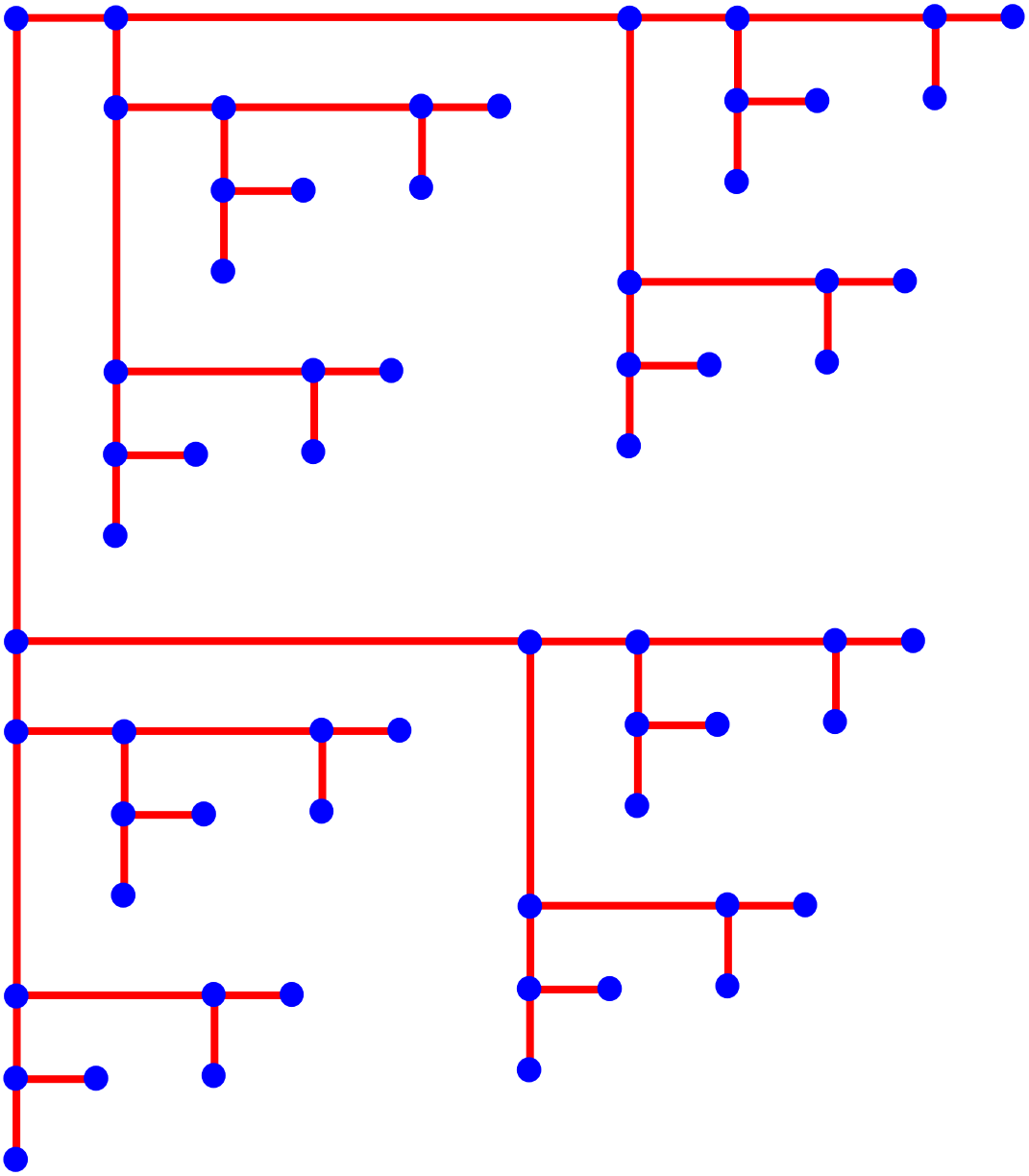


- Example:



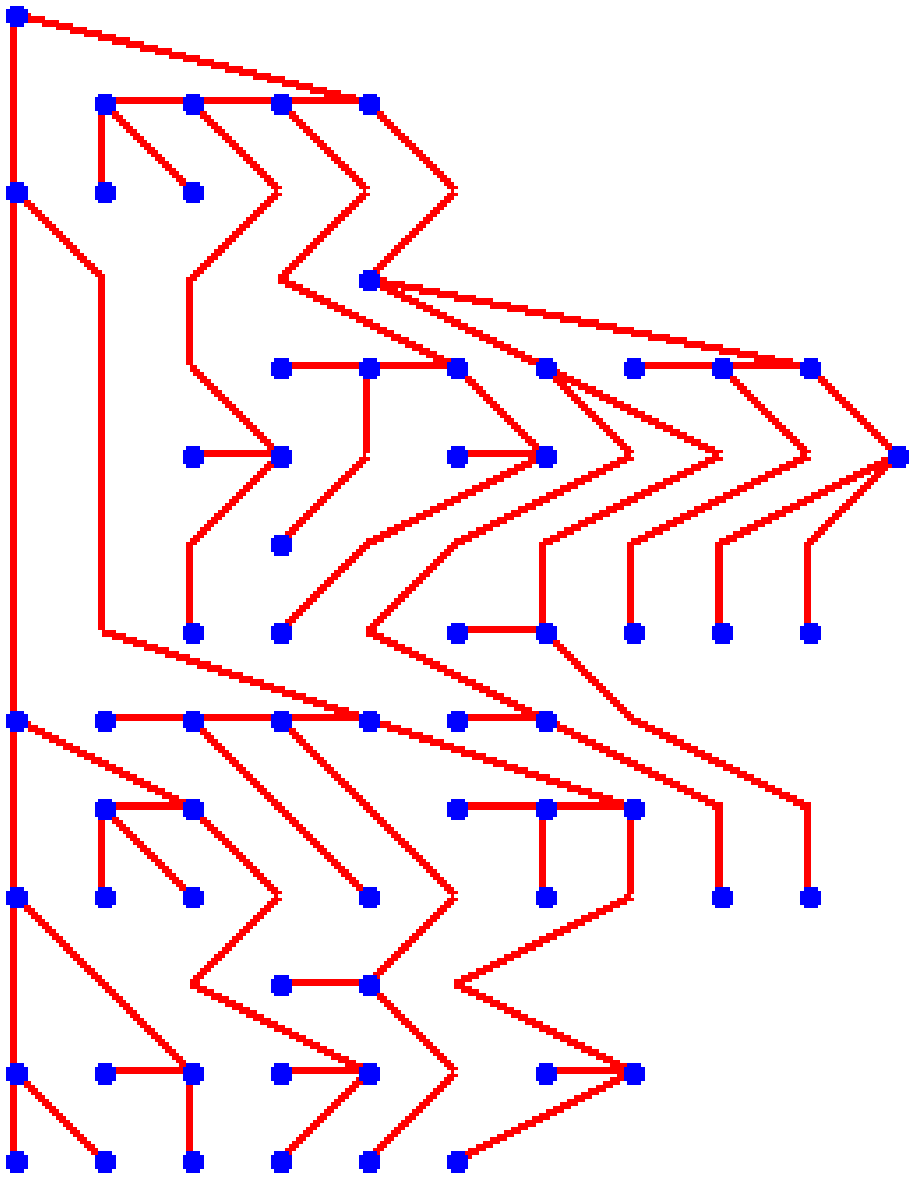
Area-Efficient Drawings of Trees

- planar straight-line upward grid drawings of ***AVL trees*** with ***O(n) area***
  [Crescenzi Di Battista Piperno 92]
  [Crescenzi Penna Piperno 95]



Area-Efficient Drawings of Trees

- planar polyline upward grid drawings with ***O(n) area***
  [Garg Goodrich Tamassia 93]



Area Requirement of Planar Drawings of Trees

upward level	$\Theta(n^2)$ [RT 83]
upward polyline	$\Theta(n)$ [GGT 93]
upward straight-line	$\Omega(n)$ $O(n \log n)$ [CDP 92]
upward orthogonal	$\Theta(n \log \log n)$ [GGT 93]
non-upward orthogonal	$\Theta(n)$ [L80, V91]
non-upward leaves-on-hull orthogonal	$\Theta(n \log n)$ [BK 80]

- ***Open Problem***: determine the area requirement of planar upward straight-line drawings of trees

Size of Planar Drawings of Binary Trees

- the ***size*** of a drawing is the maximum of its ***height*** and ***width***
- known bounds on the size of ***planar*** drawings of binary trees:

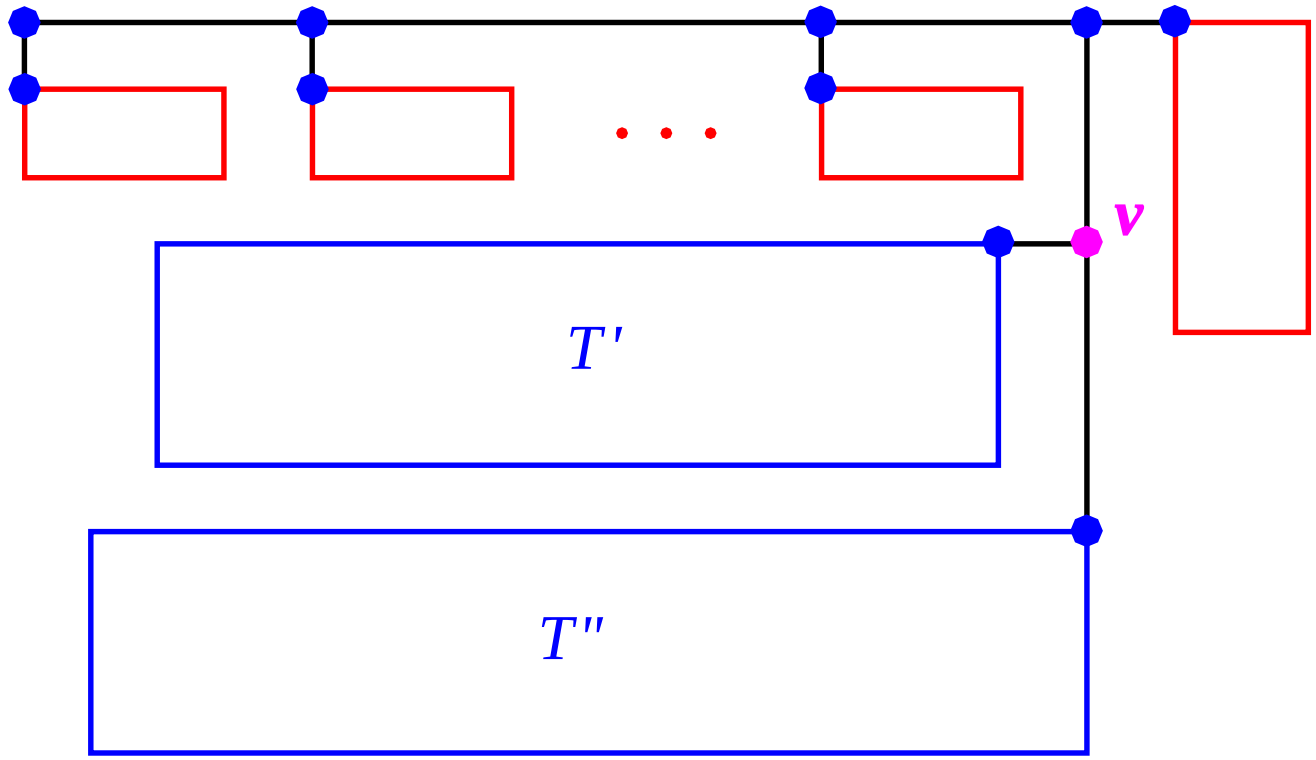
upward, straight-line level	$O(n)$ [RT 83]
upward, polyline	$\Theta(n^{1/2})$ [GGT93]
upward, straight-line orthogonal, ***AVL trees***	$\Theta(n^{1/2})$ [CGKT96]
upward, straight-line orthogonal	$\Theta((n \log n)^{1/2})$ [CGKT96]

- ***Open Problem***: can $\Theta(n^{1/2})$ size be achieved for (nonupward) planar straight-line drawings of binary trees?

Planar Upward Straight-Line Drawings of Binary Trees with Optimal Size

- ***recursive winding*** technique [CGKT96]:
  - let N be number of nodes in the tree, and N(v) be the number of nodes in the subtree rooted at v
  - for each node u, swap children to have N(left(u)) ≤ N(right(u)
  - find the first node ***v*** on the rightmost path such that:
    $$N(right(\boldsymbol{v})) \leq N - (N \log N)^{1/2} < N(\boldsymbol{v})$$
  - draw the left subtrees on the path from the root to ***v*** with linear width (height) and logarithmic height (width)
  - draw recursively the subtrees T' and T'' of ***v***

Recursive Winding Drawing

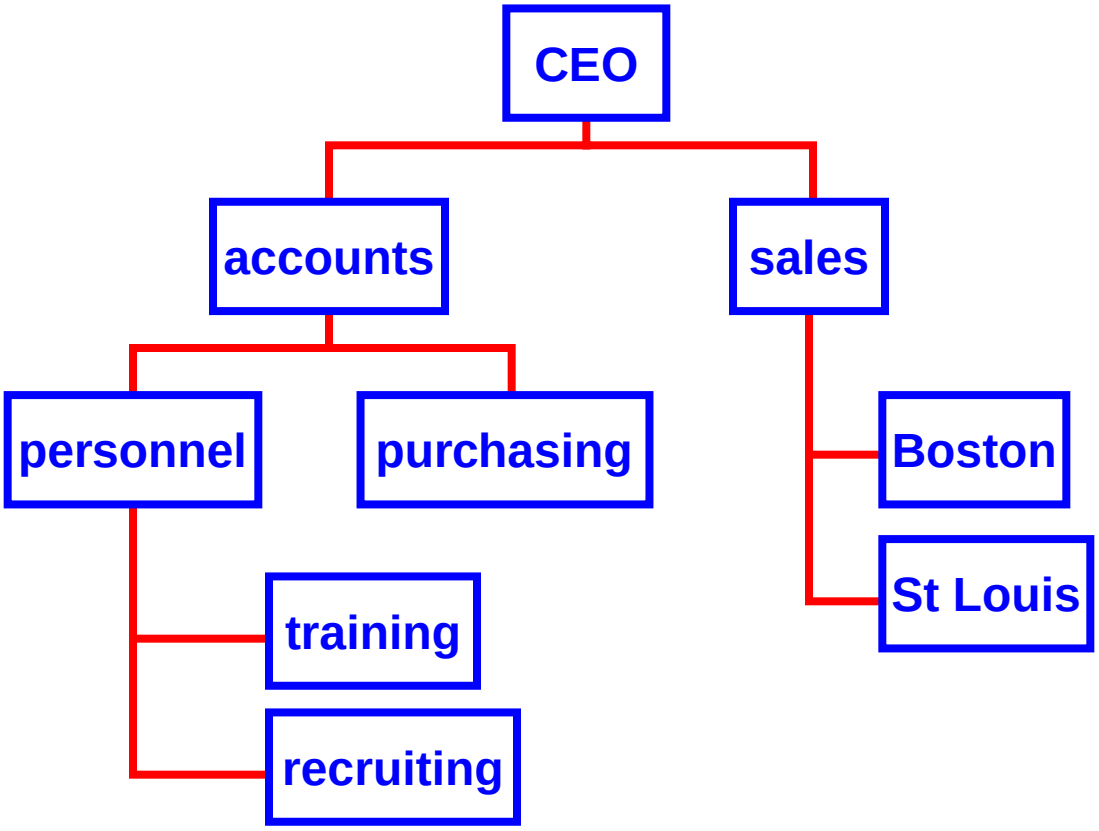


- recurrence relations for the width W(N) and height H(N):
  - W(N) max{W(N'), W(N''), A} + O(log N)
  - H(N) max{H(N') + H(N'') + O(log N), A}

  where:
  - $A = (N \log N)^{1/2}$
  - $\max(N', N'') \leq N - A$
- solution:
  - $W(N)=H(N)= O(N \log N)^{1/2}$

Tip-Over Drawings of Rooted Trees

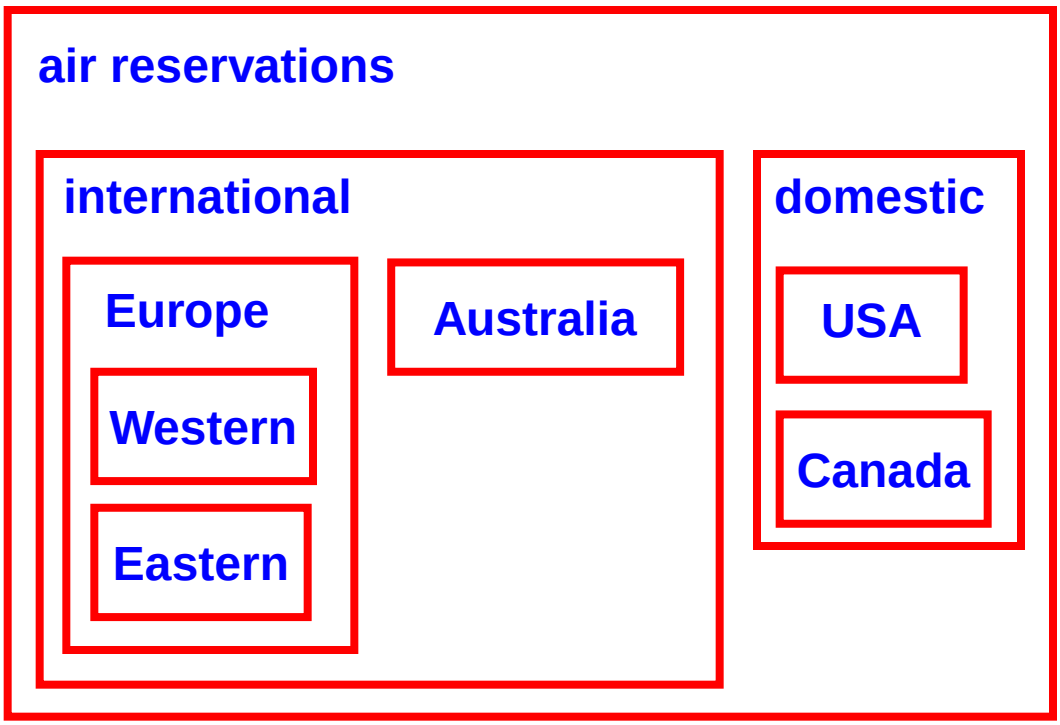
- ***Tip-over drawings*** are upward planar orthogonal drawings such that the children of a node:
  - are arranged either horizontally or vertically
  - share portions of the edges to the parent.



- Widely used in organization charts.
- Allow to better fit the drawing in a prescribed region.

Inclusion Drawings of Rooted Trees

- ***Inclusion drawings*** display the parent-child relationship by the inclusion between isothetic rectangles.



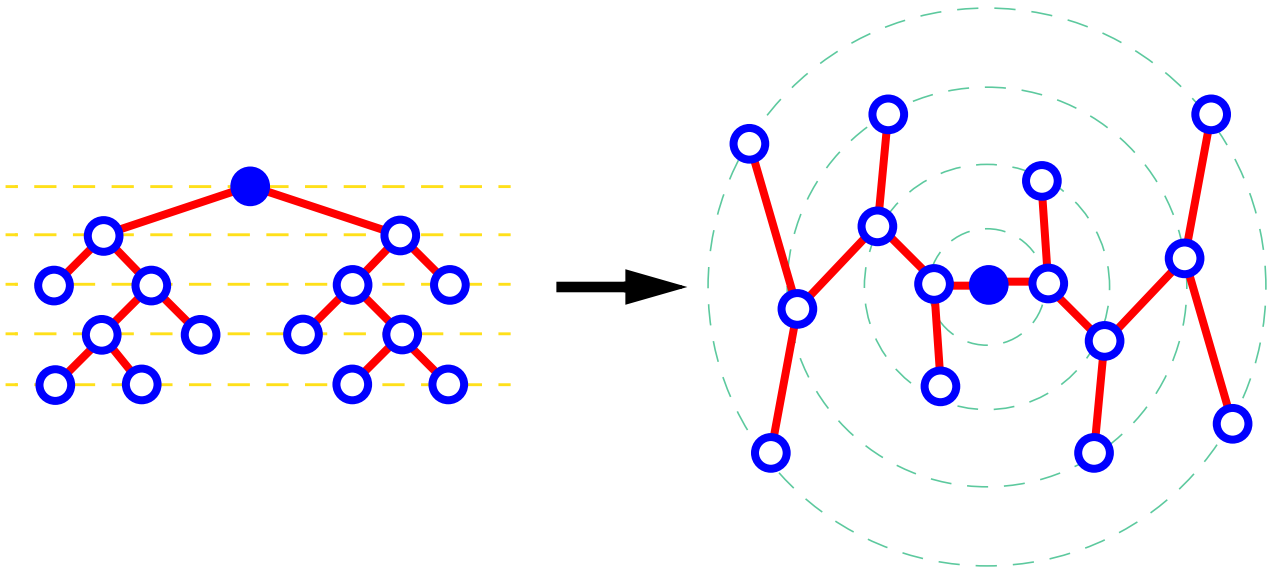
- Closely related to tip-over drawings.
- Used for displaying compound graphs (e.g., the union of a graph and a tree)
- Allow to better fit the drawing in a prescribed region

Area of Tip-Over and Inclusion Drawings

- Eades, Lin and Lin (1992) study of the area requirement of tip-over and inclusion drawings of rooted trees.
- The dimensions of the node labels are given as part of the input.
- ***Minimizing the area*** of the drawing is:
  - ***NP-hard for general trees***
  - computable in ***polynomial time*** for ***balanced trees*** with a ***dynamic programming*** algorithm
- Similar results for the following problems:
  - minimizing the ***perimeter*** of the drawing.
  - minimizing the ***width*** for a given height
  - minimizing the ***height*** for a given width

How to Draw Free Trees

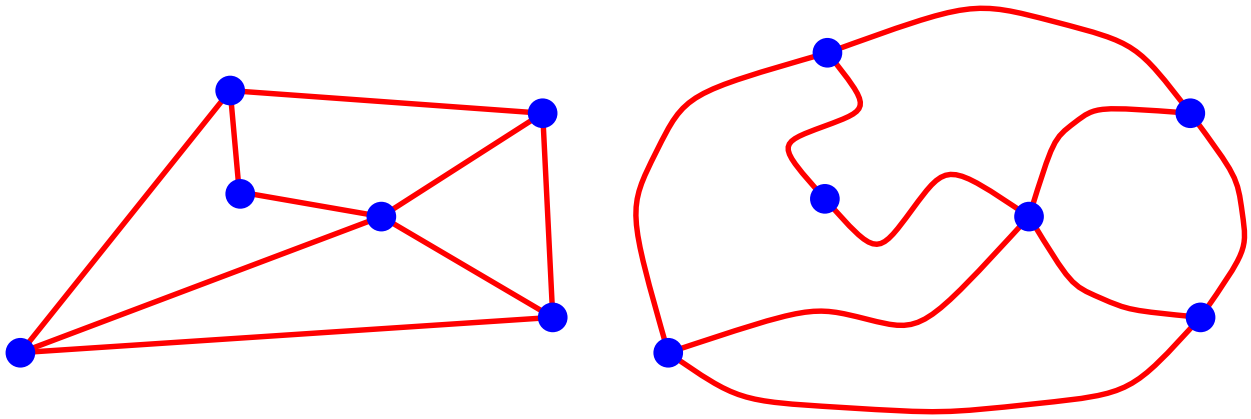
- ***Free trees*** are connected graphs without cycles and do not represent hierarchical relationships (e.g., spanning trees)
- Level drawings of rooted trees yield ***radial drawings*** of free trees:
  - root the free tree T at its ***center*** (node with minmax distance from the leaves), which gives a rooted tree T'
  - construct a level drawing Δ' of T'
  - use a geometric transformation (***cartesian*** → ***polar***) to obtain from Δ' a radial drawing Δ of T



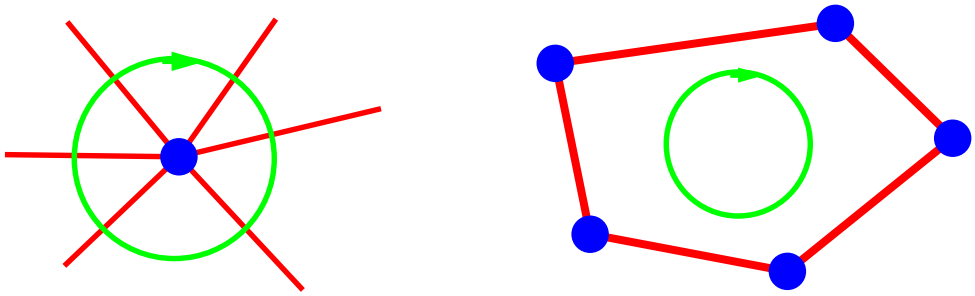
Planar Undirected Graphs

Planar Drawings and Embeddings

- a ***planar embedding*** is a class of topologically equivalent planar drawings



- a planar embedding prescribes
  - the ***star*** of edges around each vertex
  - the ***circuit*** bounding each face



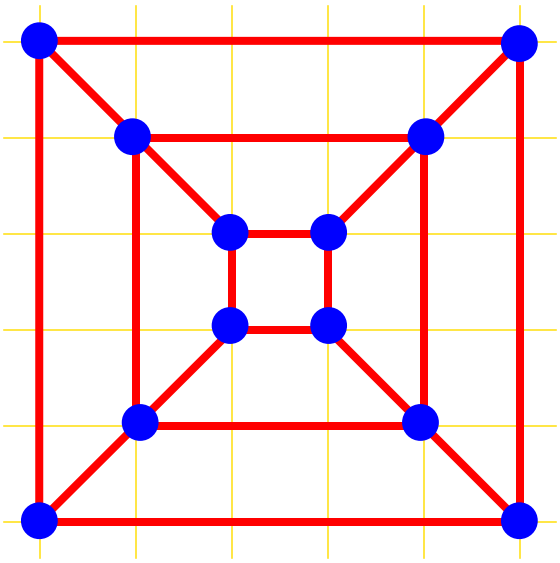
- the number of distinct embeddings is exponential in the worst case
- triconnected planar graphs have a unique embedding

The Complexity of Planarity Testing

- Planarity testing and constructing a planar embedding can be done in ***linear time***:
  - ***depth-first-search***
    [Hopcroft Tarjan 74]
    [de Fraysseix Rosenstiehl 82]
  - ***st-numbering and PQ-trees***
    [Lempel Even Cederbaum 67]
    [Even Tarjan 76]
    [Booth Lueker 76]
    [Chiba Nishizeki Ozawa 85]
- The above methods are ***complicated*** to understand and implement
- ***Open Problem:***
  - devise a ***simple*** and ***efficient*** planarity testing algorithm.

Planar Straight-Line Drawings

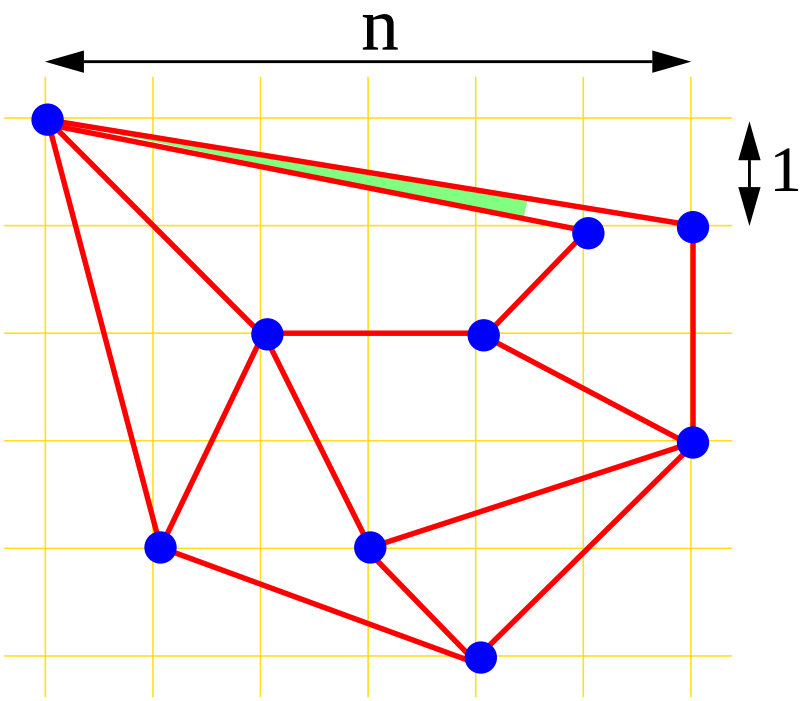
- [Hopcroft Tarjan 74]: planarity testing and constructing a planar embedding can be done in O(n) time
- [Fary 48, Stein 51, Steinitz 34, Wagner 36]: every planar graph admits a planar straight-line drawing



- Planar straight-line drawings may need $\Omega(n^2)$ area
- [de Fraysseix Pach Pollack 88, Schnyder 89, Kant 92]: $O(n^2)$-area planar straight-line grid drawings can be constructed in O(n) time

Planar Straight-Line Drawings: Angular Resolution

- $O(n^2)$-area drawings may have $\rho = O(1/n^2)$



- [Garg Tamassia 94]:
  - ***Upper bound*** on the angular resolution:

$$\rho = O\left(\sqrt{\frac{\log d}{d^3}}\right)$$

  - ***Trade-off*** (area vs. angular resolution):

$$A = \Omega(c^{\rho n})$$

- [Kant 92] Computing the optimal angular resolution is ***NP-hard***.

Planar Straight-Line Drawings: Angular Resolution

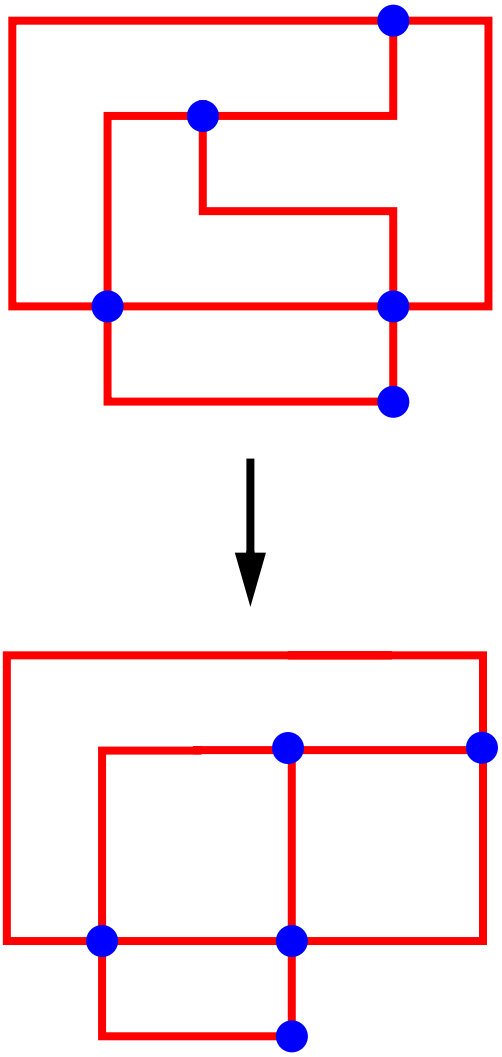
- [Malitz Papakostas 92]: the angular resolution depends on the degree only:

$$\rho = \Omega\left(\frac{1}{7^d}\right)$$

- Good angular resolution can be achieved for special classes of planar graphs:
  - ***outerplanar graphs***, $\rho = O(1/d)$ [Malitz Papakostas 92]
  - ***series-parallel graphs***, $\rho = O(1/d^2)$ [Garg Tamassia 94]
  - ***nested-star graphs***, $\rho = O(1/d^2)$ [Garg Tamassia 94]
- ***Open Problems***:
  - can we achieve $\rho = O(1/d^k)$ (k a small constant) for all planar graphs?
  - can we efficiently compute an ***approximation*** of the optimal angular resolution?

Planar Orthogonal Drawings: Minimization of Bends

- given planar graph of degree ≤ 4, we want to find a planar orthogonal drawing of G with the minimum number of bends

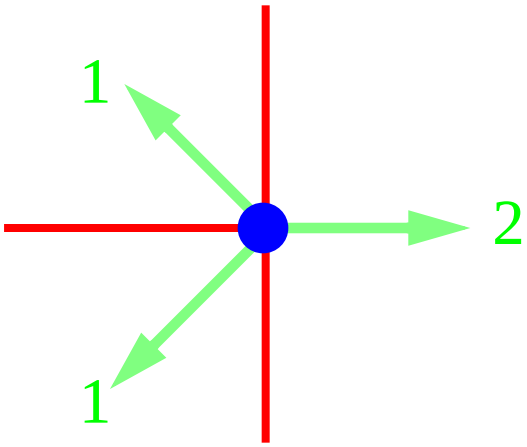


Minimization of Bends in Planar Orthogonal Drawings

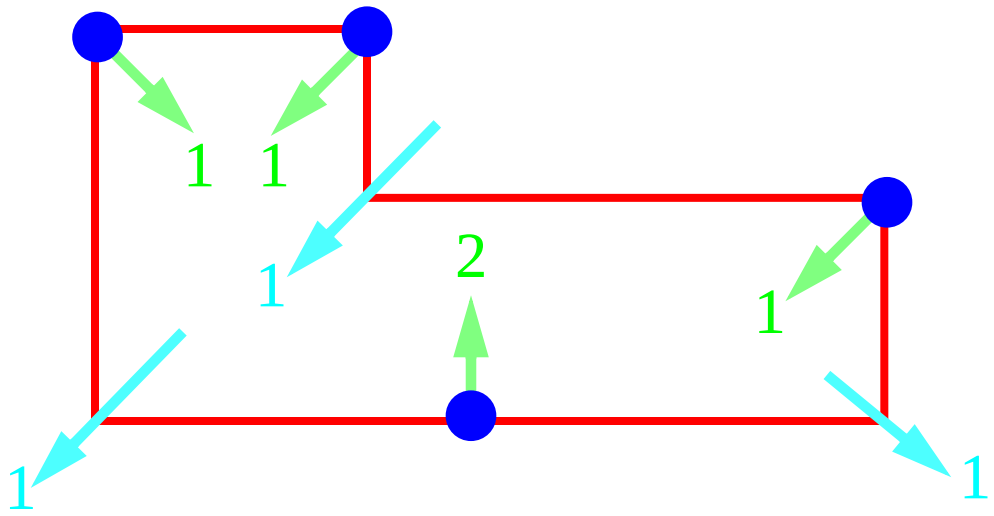
- [Tamassia 87]
  - $O(n^2 \log n)$-time bend minimization for fixed embedding
- [Di Battista Liotta Vargiu 93]
  - polynomial-time bend minimization for degree-3 and series-parallel graphs
- [Tamassia Tollis 89]
  - $O(n)$-time approximation with $O(n)$ bends
- [Garg Tamassia 93]
  - minimization of bends is NP-hard
  - approximation with $O(opt + n^{1-\varepsilon})$ bends is NP-hard
  - rectilinear planarity testing is NP-complete

Network Flow Model

- a unit of flow is a 90° angle
- a vertex (source) produces 4 units



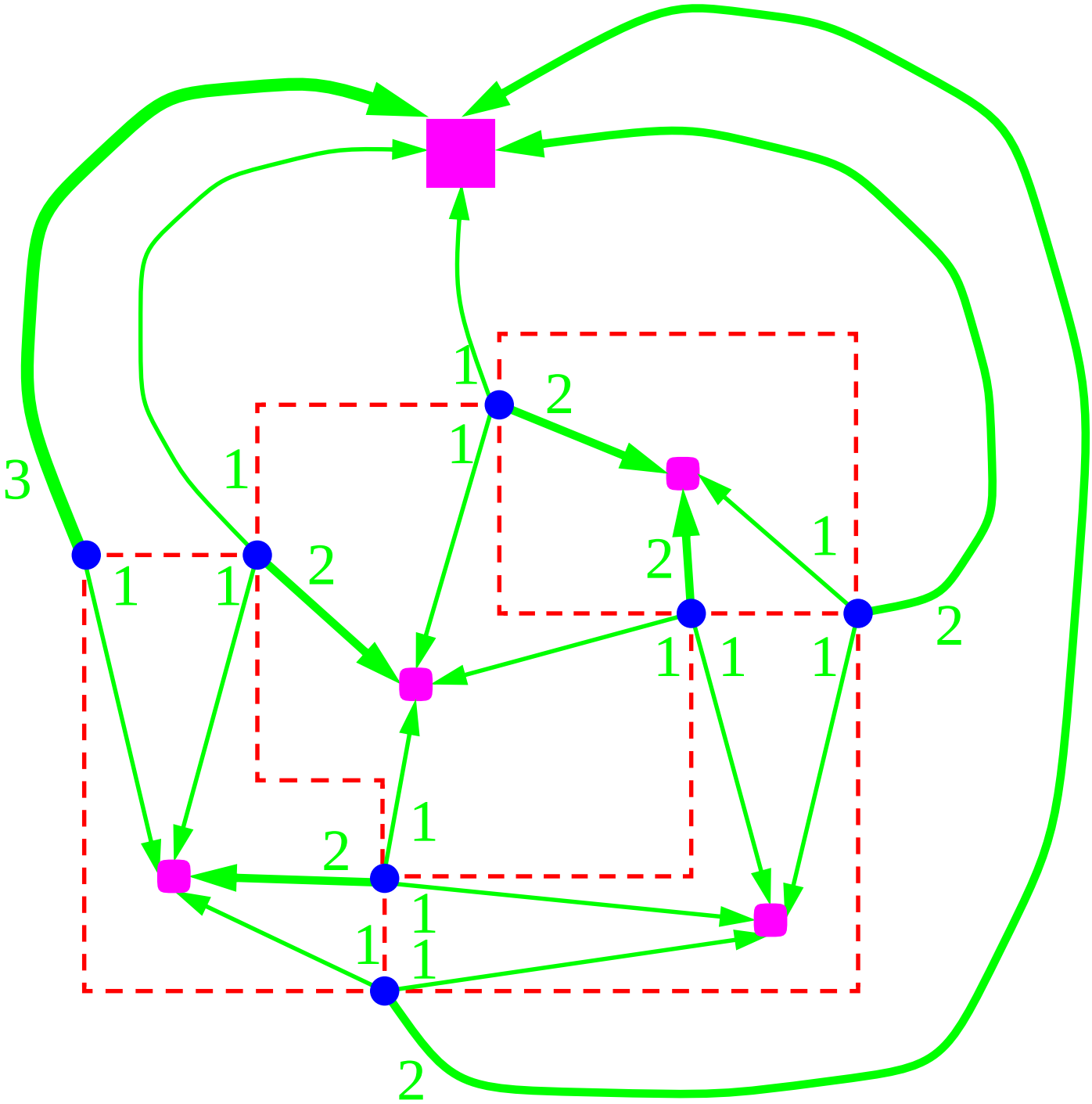
- a face f (sink) consumes 2 deg(f) – 4 units (deg(f) + 4 for the external face)



- Edges transport flow across faces

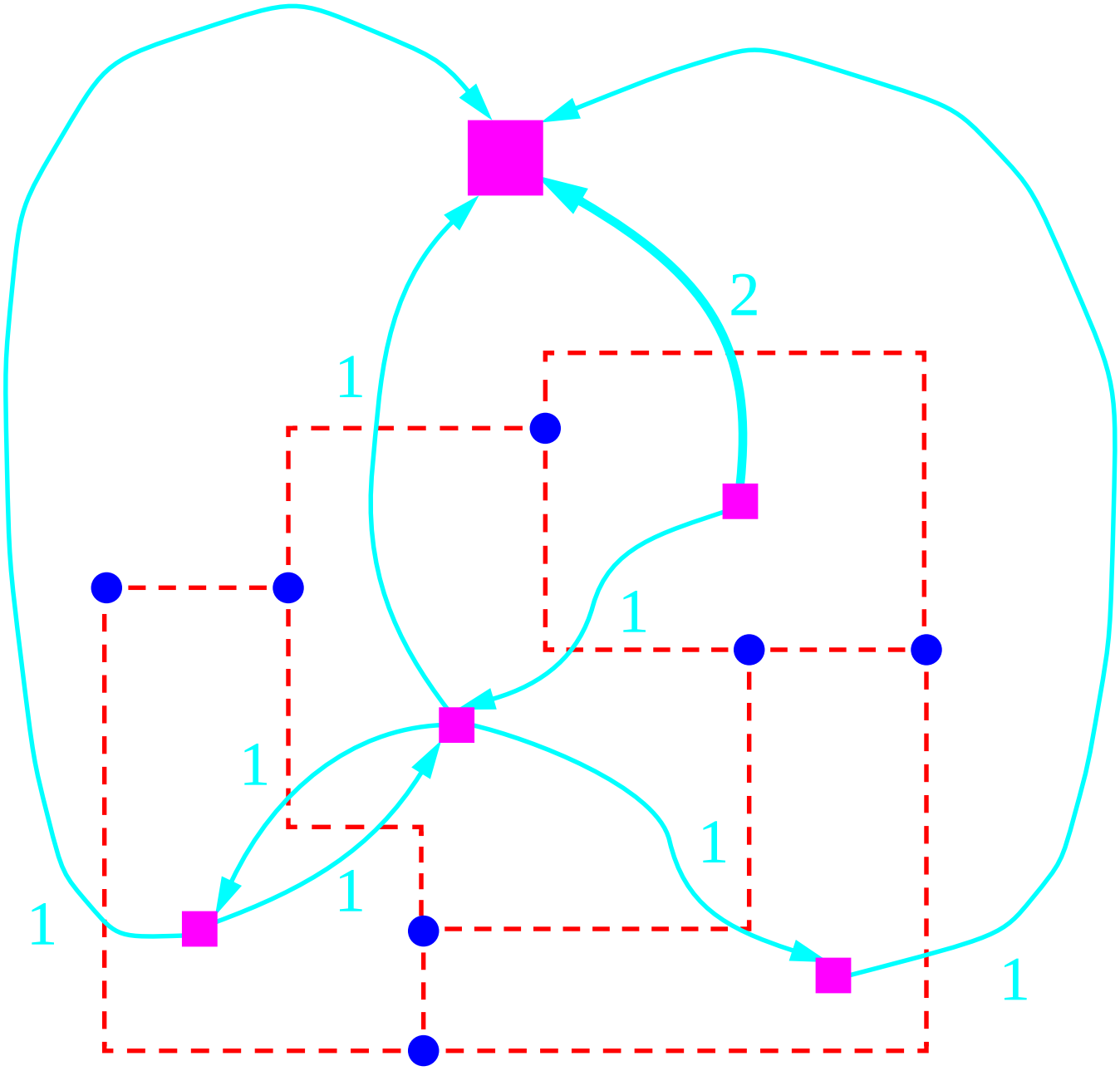
Flow Network

- vertex-face arcs: flow ≥ 1, cost = 0

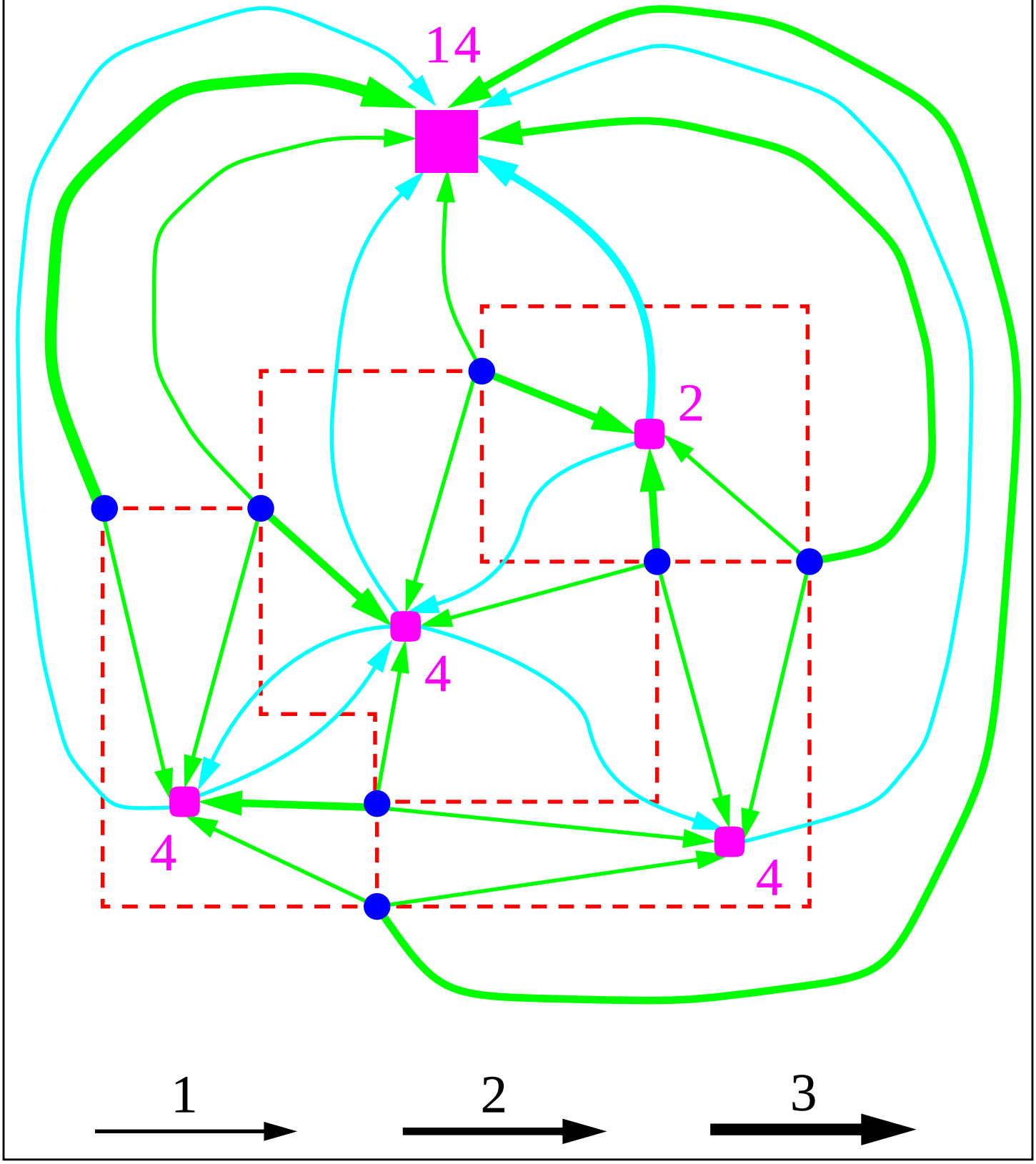


Flow Network

- face-face arcs: flow ≥ 0, cost = 1



Complete Flow Network



Correctness of Flow Model

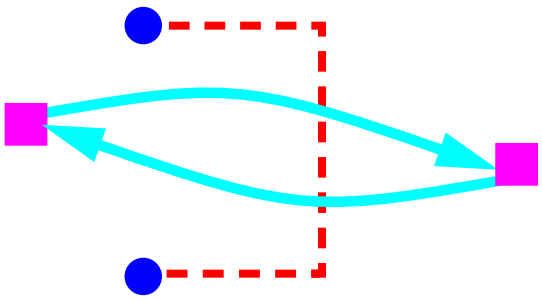
- supply of sources = demand of sinks ↔ Euler's formula
- flow conservation at vertex ↔ ∑ angles around vertex = 360°
- flow conservation at face ↔ (# 90° angles) – (# 270° angles) = 4
- cost of flow ↔ # bends
- flow in N ↔ drawing of G
- minimum cost flow ↔ optimal drawing

Theorem [Tamassia 87] Computing the minimum number of bends for an embedded graph G is equivalent to computing a minimum cost flow in network N, and takes $O(n^2 \log n)$ time

Open Problem: reduce the time complexity of bend minimization.

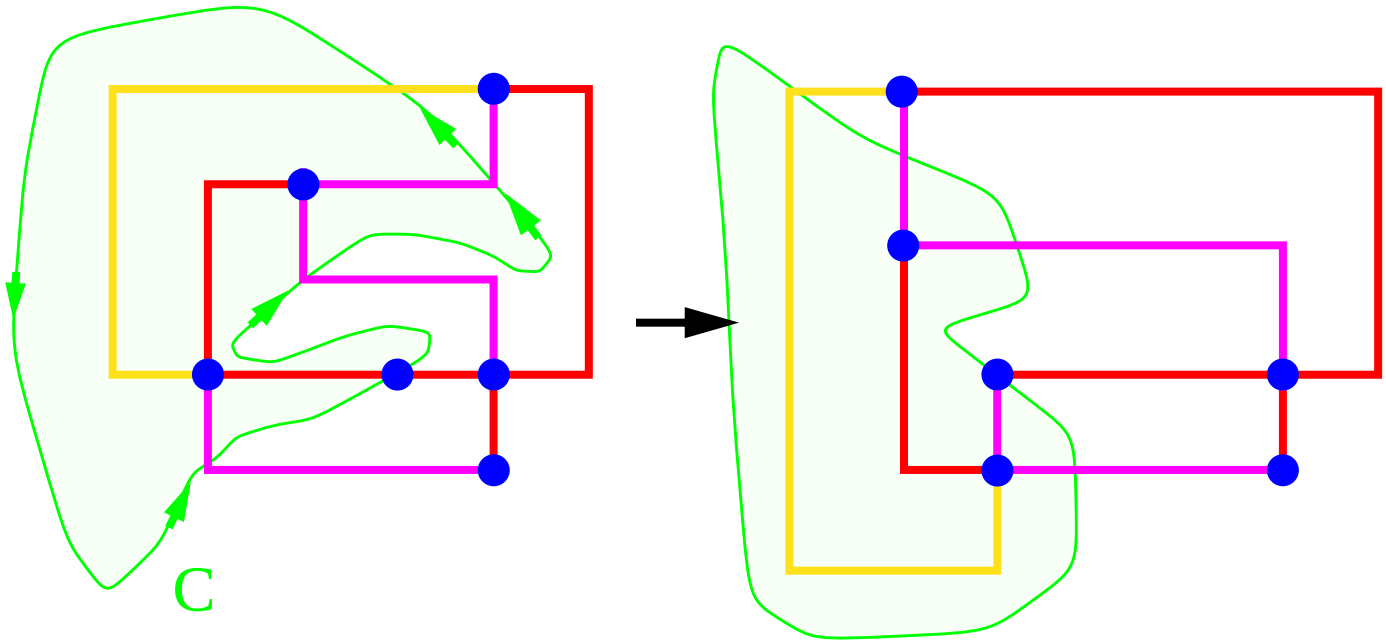
Constrained Bend Minimization

- the network flow model allows us to minimize bends subject to ***shape constraints***
  - prescribed angles around a vertex
  - prescribed bends along an edge
  - upper bound on the number of bends on an edge
- the above ***shape constraints*** on the drawing can be expressed by setting appropriate ***capacity constraints*** on the edges of the network
- E.g., we can prescribe a maximum of 2 bends on a given edge ***e*** by setting equal to 2 the capacity of the ***face-face arcs*** associated with ***e***



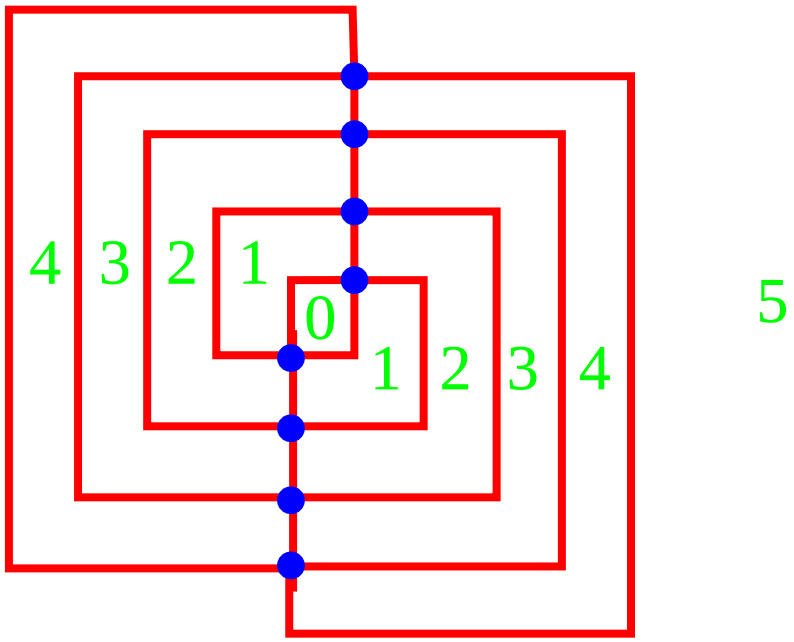
Characterization of Bend-Minimal Drawings

- A drawing has the minimum number of bends if and only if there is no oriented closed curve C such that
  - vertices are intersected by C entering from angles ≥ 180°
  - (# edges crossed by C from 90° or 180°) < (# edges crossed by C from 270°)
- If such a curve exists, “rotating” the portion of the drawing inside C reduces the number of bends



Proving the Optimality of a Drawing

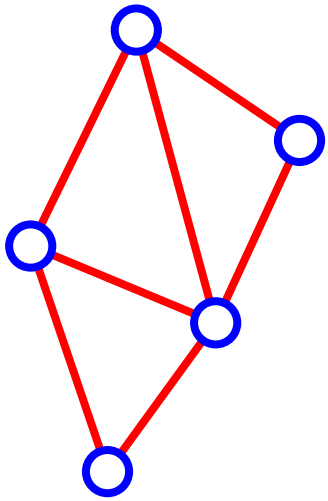
- potential Φ on each face

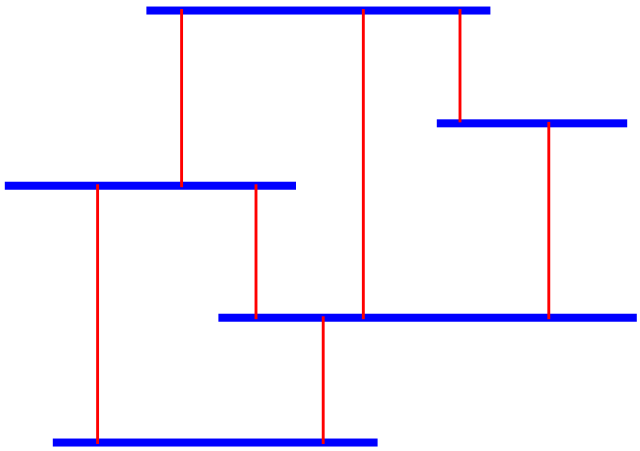


- vertices cannot be traversed by C
- C traverses edge from 270° $\Rightarrow$ $\Delta\Phi_i = -1$
- C traverses edge from 90° $\Rightarrow$ $\Delta\Phi_i = +1$
- bends removed going "inward" and inserted going "outward" $\Delta B_i + \Delta\Phi_i = 0$
- C is a closed curve $\Rightarrow$ $\Sigma i\ \Delta\Phi_i = 0$
- Hence, $\Sigma i\ \Delta B_i = 0$

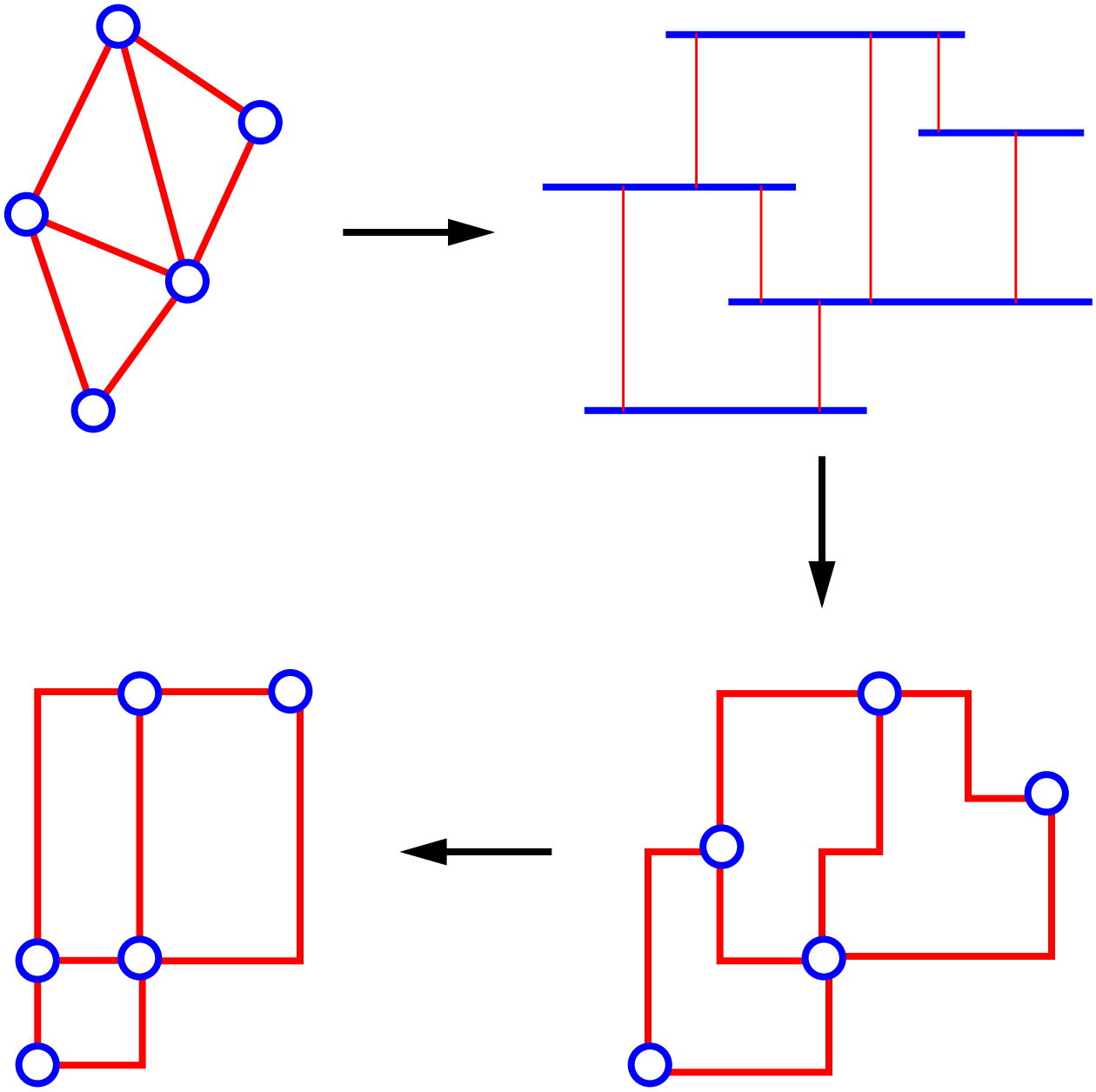
Visibility Representation

- vertices → horizontal segments
- edges → vertical segments
- can be constructed in O(n) time
- preliminary step for drawing algorithms





From Visibility Representations to Orthogonal Drawings



Heuristic Algorithm for Bend Minimization

1. Construct visibility representation
2. Transform visibility representation into a preliminary drawing
3. Apply bend-stretching transformations
4. Compact orthogonal representation

Runs in O(n) time and can be parallelized

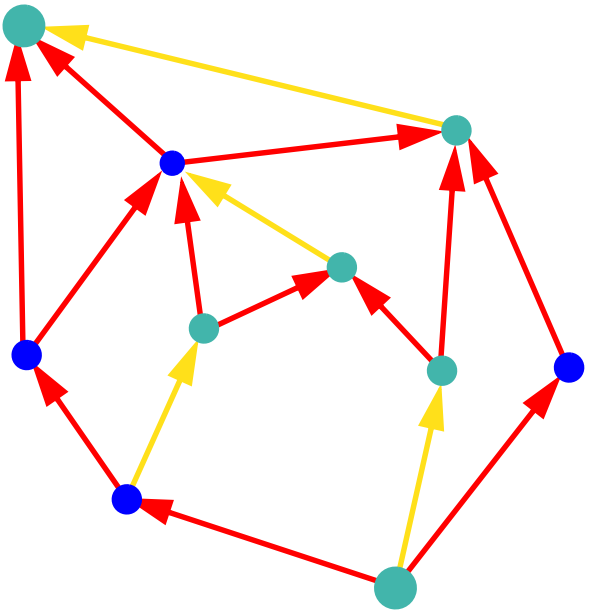
At most 2n + 4 bends if G is biconnected (2.4n + 2 otherwise)

$O(n^2)$ area

Planar Directed Graphs

Upward Planarity Testing

- upward planarity testing for ordered sets has the same complexity as for general digraphs (insert dummy vertices on transitive edges)
- [Kelly 87, Di Battista Tamassia 87]: upward planarity is equivalent to subgraph inclusion in a planar st-digraph (planar acyclic digraph with one source and one sink, both on the external face)



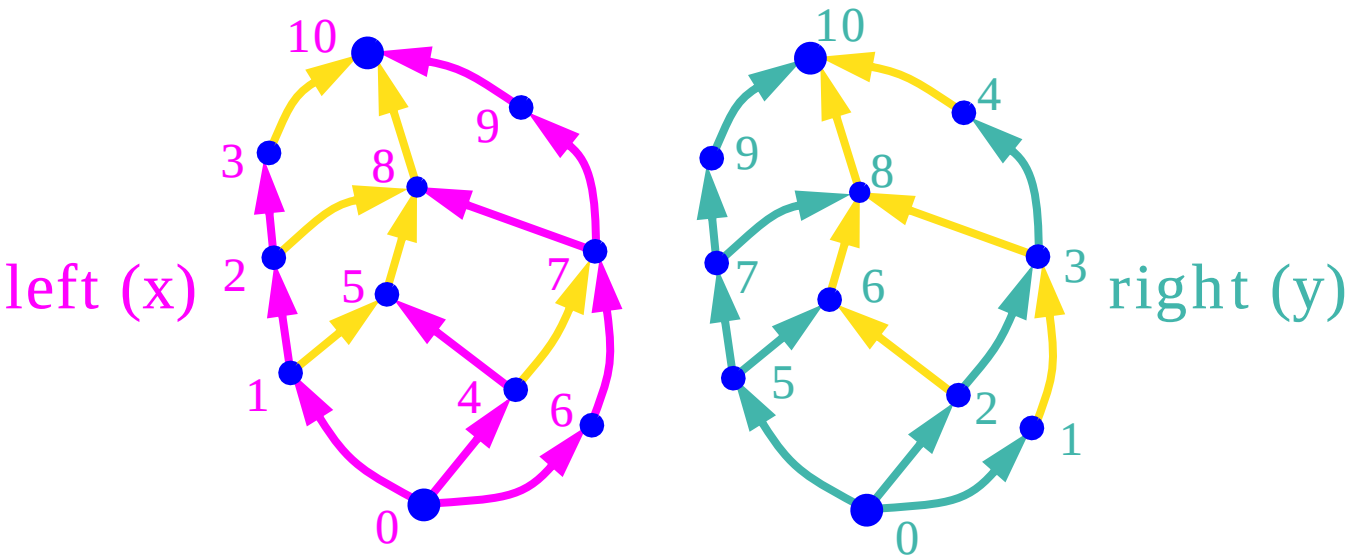
- [Kelly 87, Di Battista Tamassia 87]: upward planarity is equivalent to upward straight-line planarity

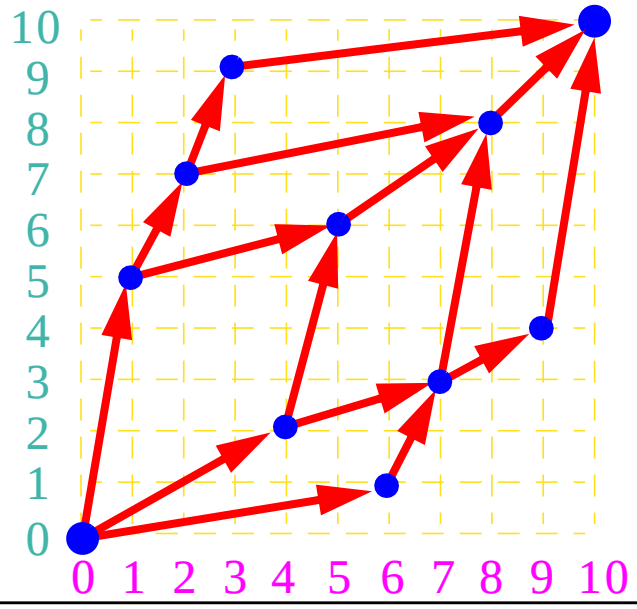
Complexity of Upward Planarity Testing

- [Bertolazzi Di Battista Liotta Mannino 91]
  - $O(n^2)$-time for fixed embedding
- [Hutton Lubiw 91]
  - $O(n^2)$-time for single-source digraphs
- [Bertolazzi Di Battista Mannino Tamassia 93]
  - $O(n)$-time for single-source digraphs
- [Garg Tamassia 93]
  - NP-complete

How to Construct Upward Planar Drawings

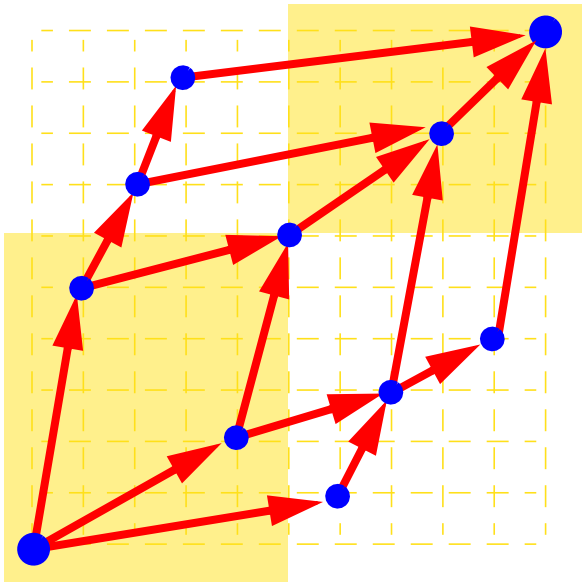
- Since an upward planar digraph is a subgraph of a ***planar st-digraph***, we only need to know how to draw planar st-digraphs
- If G is a planar st-digraph without transitive edges, we can use the ***left***/***right*** numbering method to obtain a ***dominance drawing***:



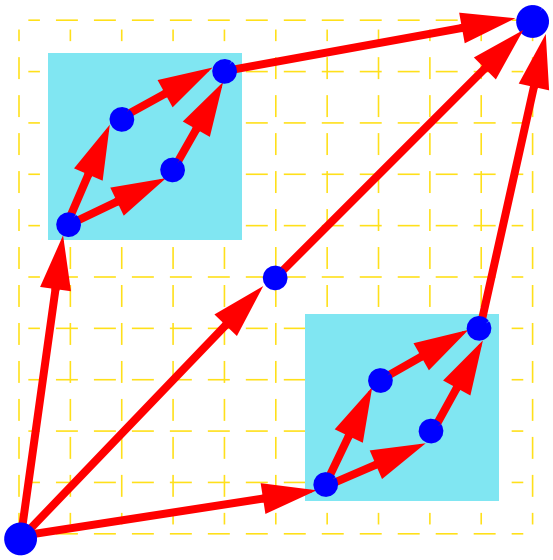


Properties of Dominance Drawings

- ***Upward, planar, straight-line, $O(n^2)$ area***
- The ***transitive closure*** is visualized by the geometric dominance relation

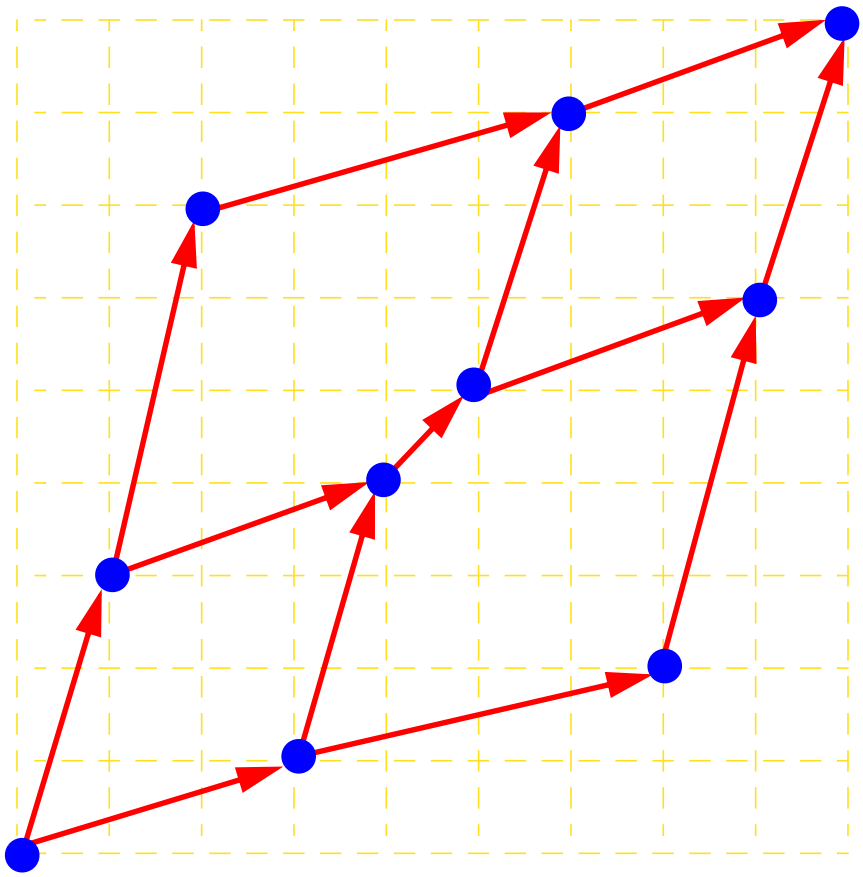


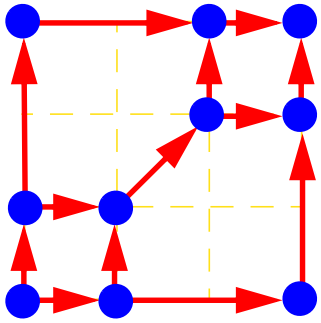
- ***Symmetries*** and ***isomorphisms*** of ***st-components*** are displayed



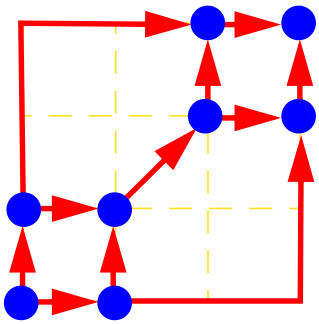
More on Dominance Drawings

- A variation of the left/right numbering yields dominance drawings with ***optimal area***



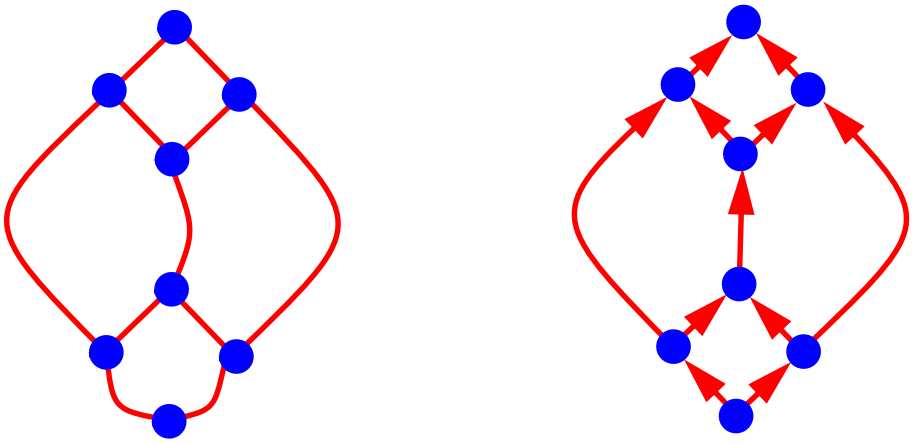


- Dummy vertices are inserted on transitive edges and are displayed as bends (upward planar polyline drawings)

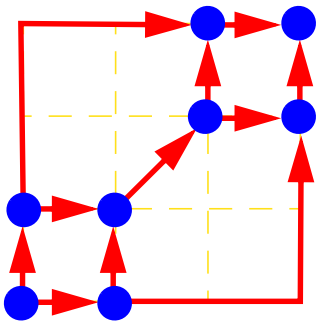


Planar Drawings of Graphs and Digraphs

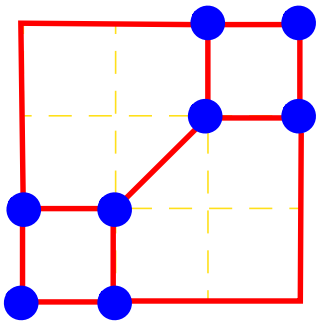
- We can use the techniques for dominance drawings also for undirected planar graphs:
  - orient G into a planar st-digraph G'



  - construct a dominance drawing of G'



  - erase arrows ...



General Undirected Graphs

Algorithmic Strategies for Drawing General Undirected Graphs

- ***Planarization method***
  - if the graph is nonplanar, ***make it planar!*** (by placing dummy vertices at the crossings)
  - use one of the drawing algorithms for planar graphs

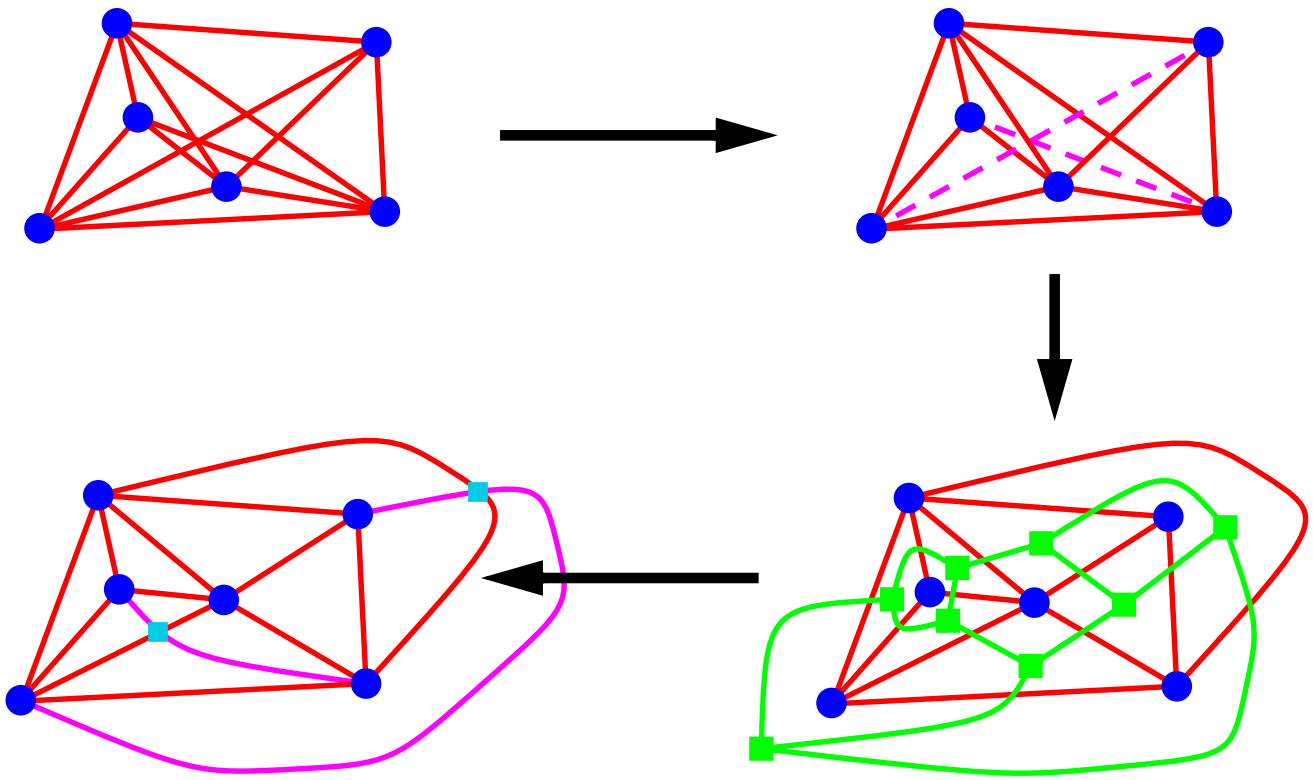
  e.g., GIOTTO [Tamassia Batini Di Battista 87]
- ***Orientation method***
  - ***orient*** the graph into a digraph
  - use one the drawing algorithms for digraphs
- ***Force-Directed method***
  - define a ***system of forces*** acting on the vertices and edges
  - find a ***minimum energy state*** (solve differential equations or simulate the evolution of the system)

  e.g., Spring Embedder [Eades 84]

A Simple Planarization Method

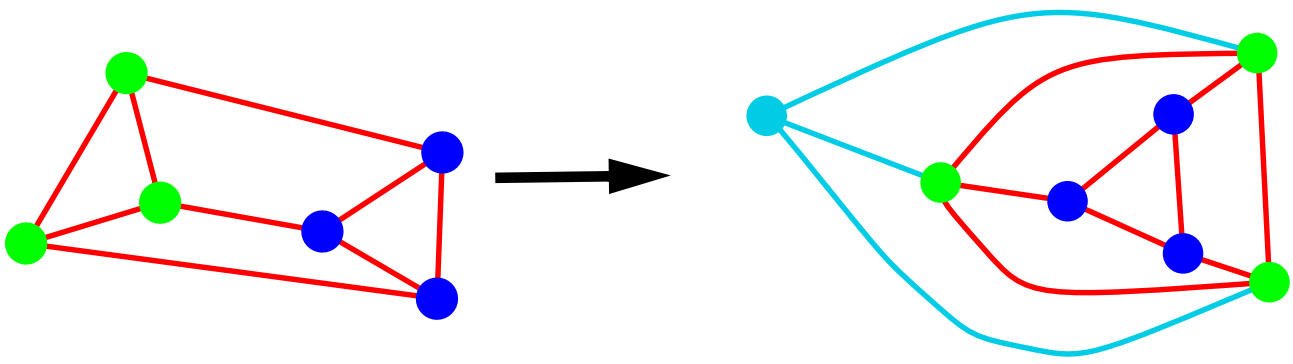
use an ***on-line planarity testing*** algorithm

1. try adding the edges one at a time, and divide them into "***planar***" (accepted) and "***nonplanar***" (rejected)
2. construct a planar embedding of the subgraph of the planar edges
3. add the nonplanar edges, one at a time, to the embedding, minimizing each time the number of ***crossings*** (shortest path in ***dual graph***)



Topological Constraints in the Planarization Method

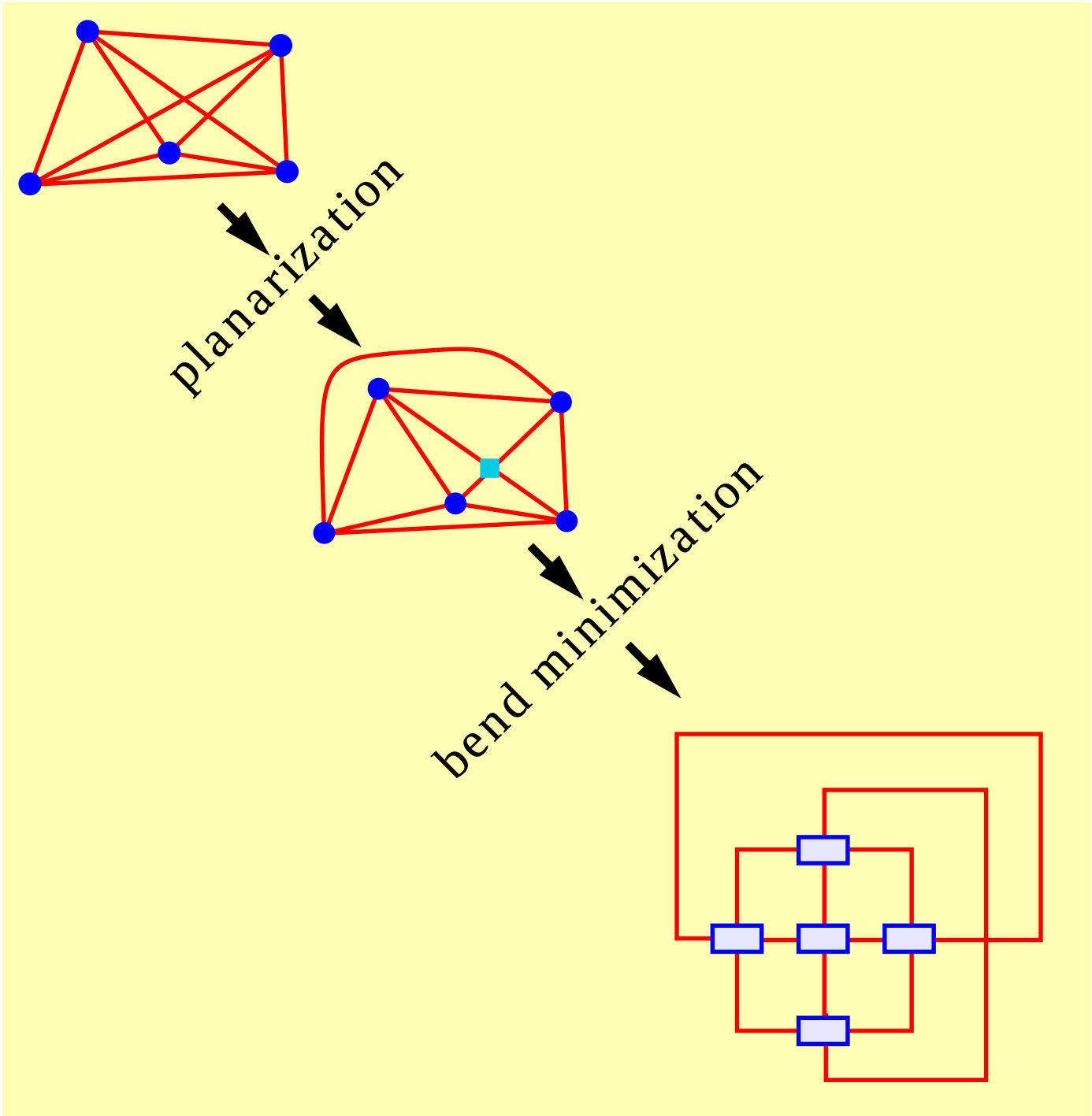
- a limited constraint satisfaction capability exists within the planarization methods
- ***Example:*** draw the graph such that the edges in a given set ***A*** have ***no crossings***
  - in Step 1, try adding first the edges in ***A***
  - in Step 3, put a large "crossing cost" on the planar edges in ***A***, and add first the nonplanar edges in ***A*** (if any)
- ***Example:*** draw the graph such the vertices of ***subset U*** are on the ***external boundary***
  - add a ***fictitious vertex v*** and edges from v to all the vertices in ***U***
  - let ***A*** be the set of edges (u,v), with u in ***U***
  - impose the above constraint



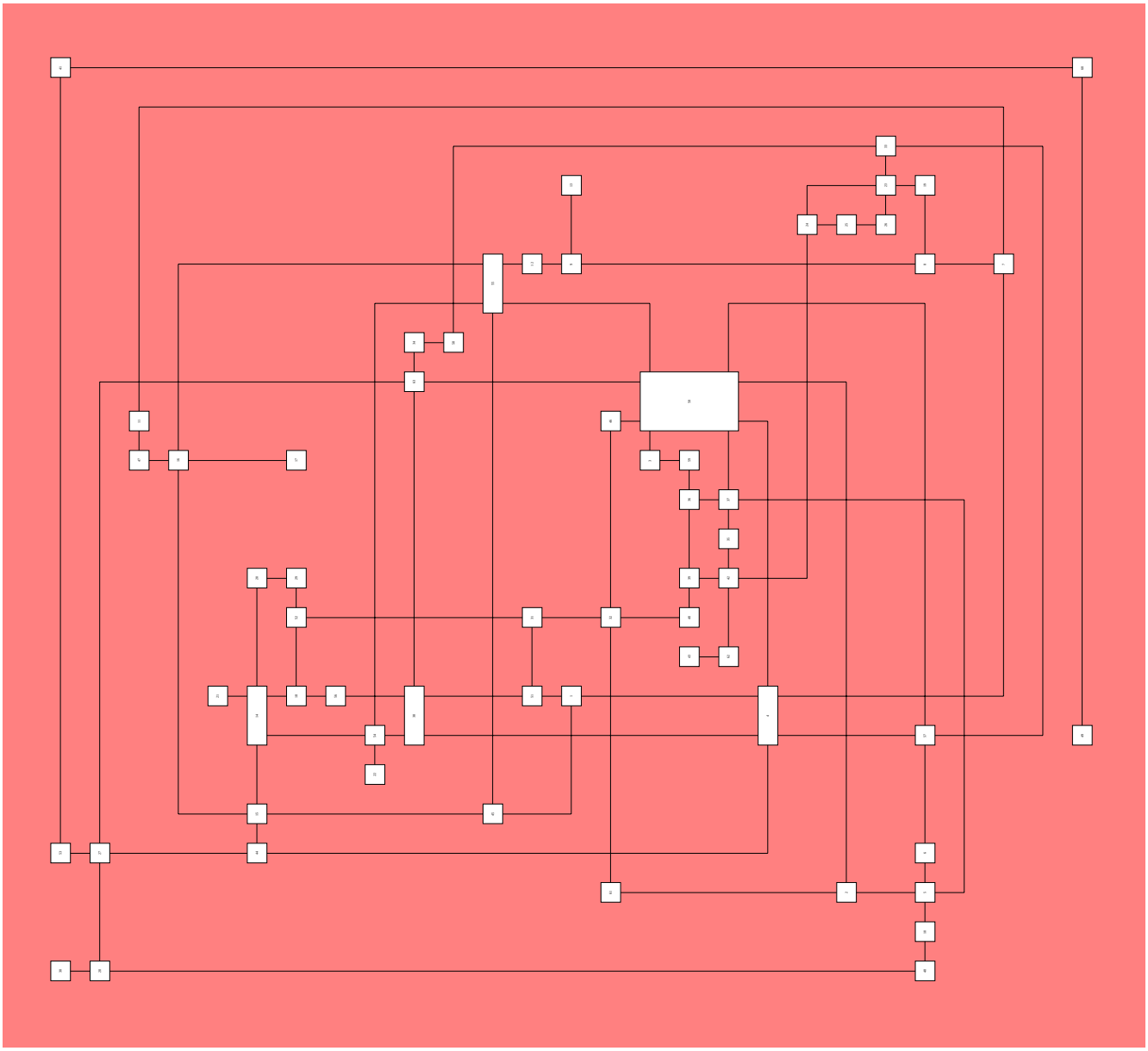
GIOTTO

[Tamassia Di Battista Batini 88]

- time complexity: $O((N+C)^2 \log N)$



Example



Constraint Satisfaction in GIOTTO

- ***topological constraints***
  - vertices on external face
  - edges without crossings
  - grouping of vertices
- ***shape constraints***
  - subgraphs with prescribed orthogonal shape
  - edges without bends
- topological contraints have ***priority*** over shape contraints because the algorithm assigns first the topology and then the orthogonal shape
- ***grouping is only topological***
- ***no position constraints***
- ***no length contraints***

Advantages and Disadvantages of Planarization Techniques

Pro:

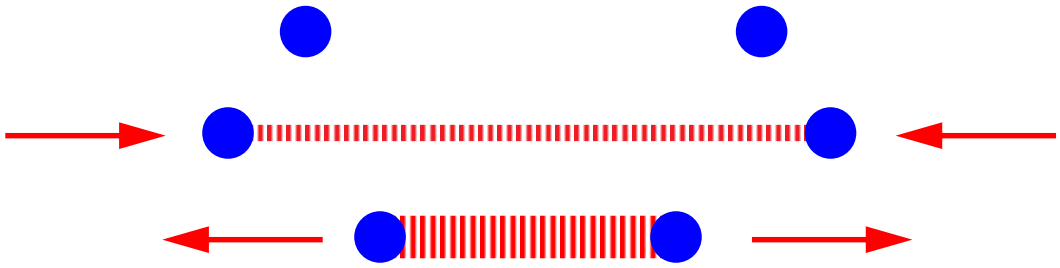
- ***fast*** running time
- ***applicable*** to straight-line, orthogonal and polyline drawings
- supported by ***theoretical results*** on planar drawings
- ***works well*** in practice, ***also for large graphs***
- limted ***constraint satisfaction*** capability

Con:

- relatively ***complex*** to implement
- ***topological transformations*** may alter the user's mental map
- ***difficult to extend to 3D***
- ***limted constraint satisfaction*** capability

The Spring Embedder
[Eades 1984]

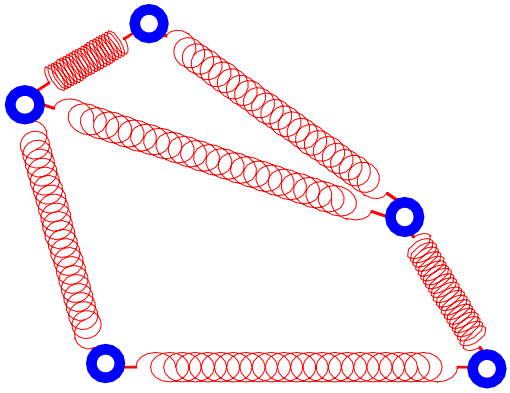
- replace the edges by ***springs*** with unit natural length
- connect nonadjacent vertices with additional springs with infinite natural length
- recall that the springs attract the endpoints when stretched, and repel the endpoints when compressed

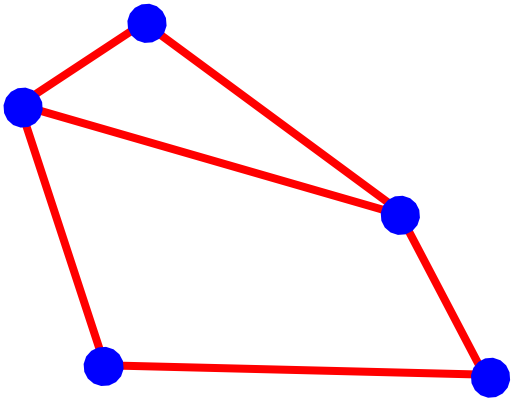


- start with an initial random placement of the vertices
- let the system go ... (assume there is ***friction*** so that a stable minimum energy state is eventually reached)

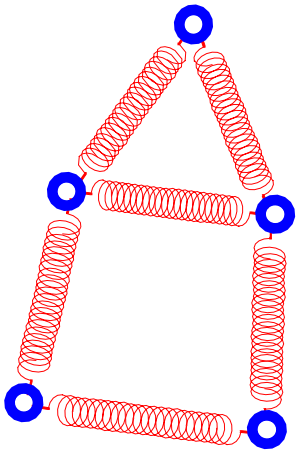
Example

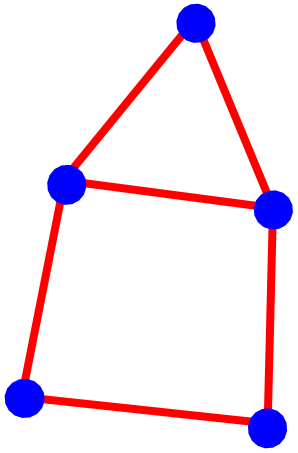
- initial configuration





- final configuration



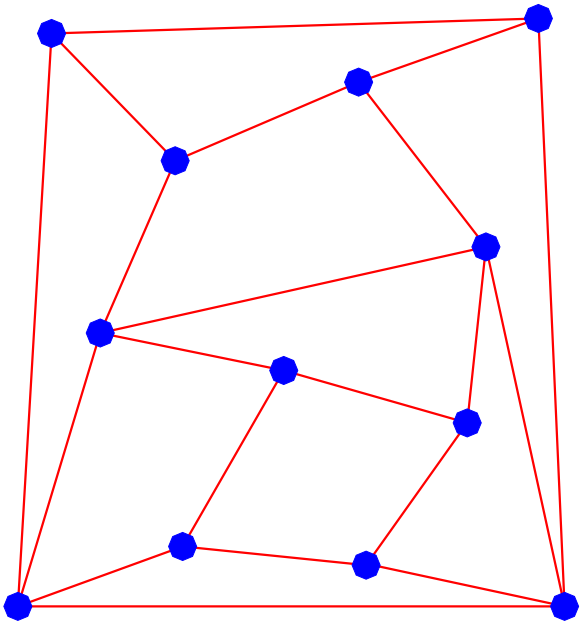


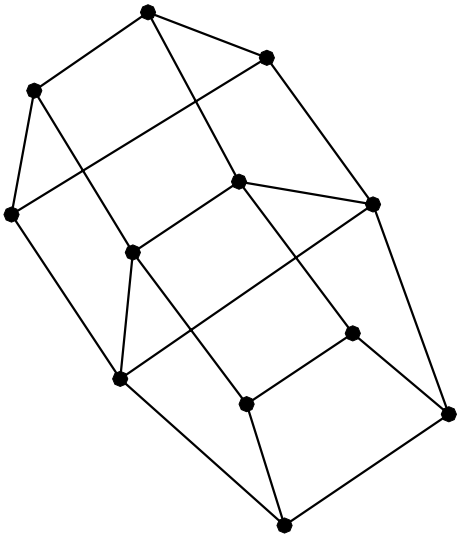
Other Force-Directed Techniques

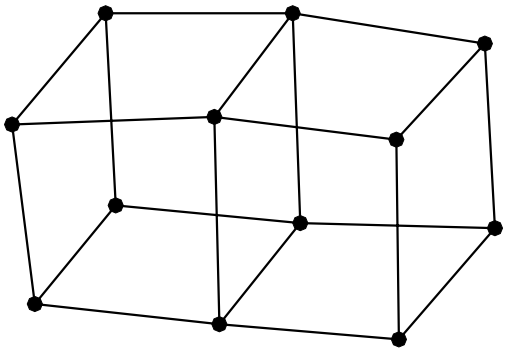
- [Kamada Kawai 89]
  - the forces try to place vertices so that their ***geometric distance*** in the drawing is equal to their ***graph-theoretic distance***
  - for each pair of vertices (u,v) use a ***spring*** with natural length dist(u,v)
- [Fruchterman Reingold 90]
  - system of forces similar to that of ***subatomic particles*** and celestial bodies
  - given drawing region acts as wall
  - ***n-body simulation***
- [Davidson Harel 89]
  - ***energy function*** takes into account vertex distribution, edge-lengths, and edge-crossings
  - given drawing region acts as wall
  - ***simulated annealing***

Examples

- drawings of the same graph constructed with the technique of [Davidson Harel 89] using three different energy functions







Advantages and Disadvantages of Force-Directed Techniques

Pro:

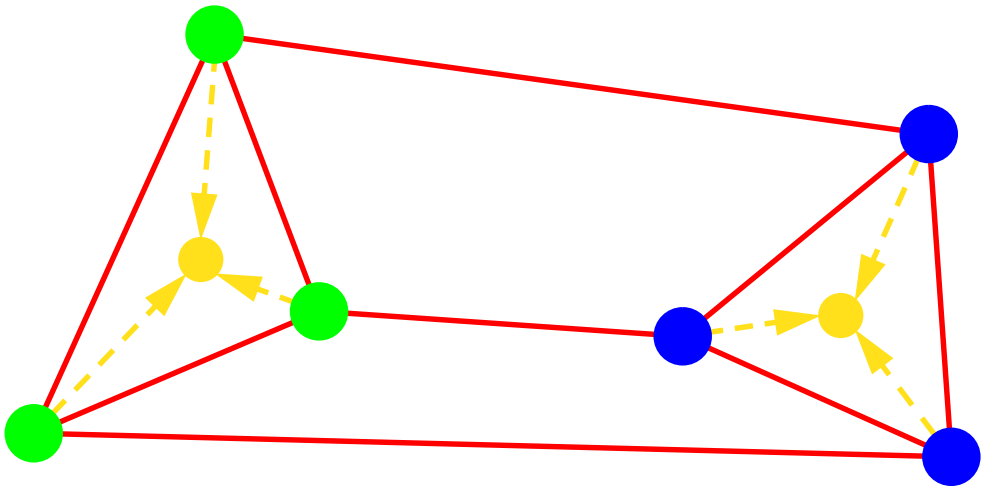
- relatively ***simple*** to implement
- ***heuristic improvements*** easily added
- ***smooth evolution*** of the drawing into the final configuration helps preserving the user's mental map
- ***can be extended to 3D***
- often able to detect and display ***symmetries***
- ***works well*** in practice ***for small graphs*** with regular structure
- limted ***constraint satisfaction*** capability

Con:

- ***slow*** running time
- ***few theoretical results*** on the quality of the drawings produced
- ***diffcult to extend*** to orthogonal and polyline drawings
- ***limited constraint satisfaction*** capability

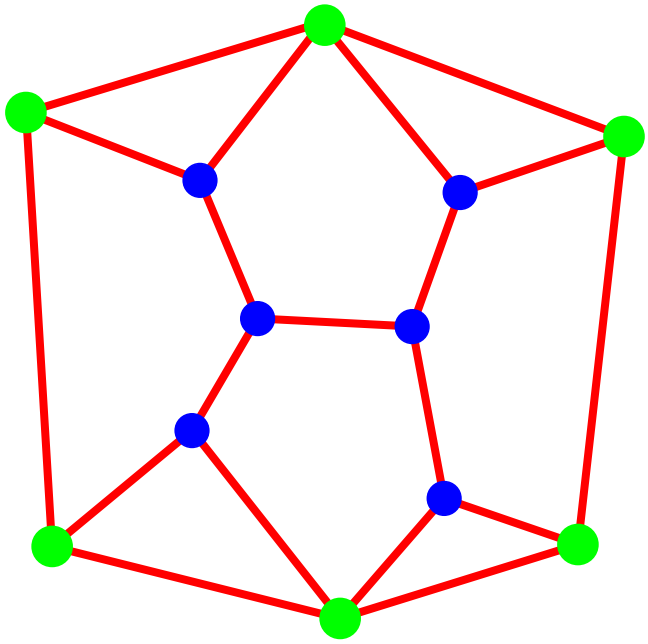
Constraints in Force-Directed Techniques

- ***position constraints*** can be easily imposed
  - we can constrain each vertex to remain in a prescribed region
- other ***constraints*** can be satisfied provided they can be ***expressed by means of forces***, e.g,
  - "***magnetic field***" to impose orientation constraints [Sugiyama Misue 84]
  - dummy "***attractor***" vertex to enforce grouping



Springs for Planar Graphs

- use springs with natural length 0, and attractive force proportional to the length
- pin down the vertices of the ***external face*** to form a given ***convex polygon*** (position constraints)
- let the system go ...



- the final configuration is a state of minimum energy: $\min \sum_{e} [\text{length}(e)]^2$
- equivalent to the ***barycentric mapping*** [Tutte 60]:

$$\mathbf{p}(v) = 1/\deg(v) \sum_{(v,w)} \mathbf{p}(w)$$

General Directed Graphs

Layering Method for Drawing General Directed Graphs

- ***Layer assignment:*** assign vertices to layers trying to minimize
  - ***edge dilation***
  - ***feedback edges***
- ***Placement:*** arrange vertices on each layer trying to minimize
  - ***crossings***
- ***Routing:*** route edges trying to minimize
  - ***bends***
- ***Fine tuning:*** improve the drawing with local modifications

[Carpano 80]

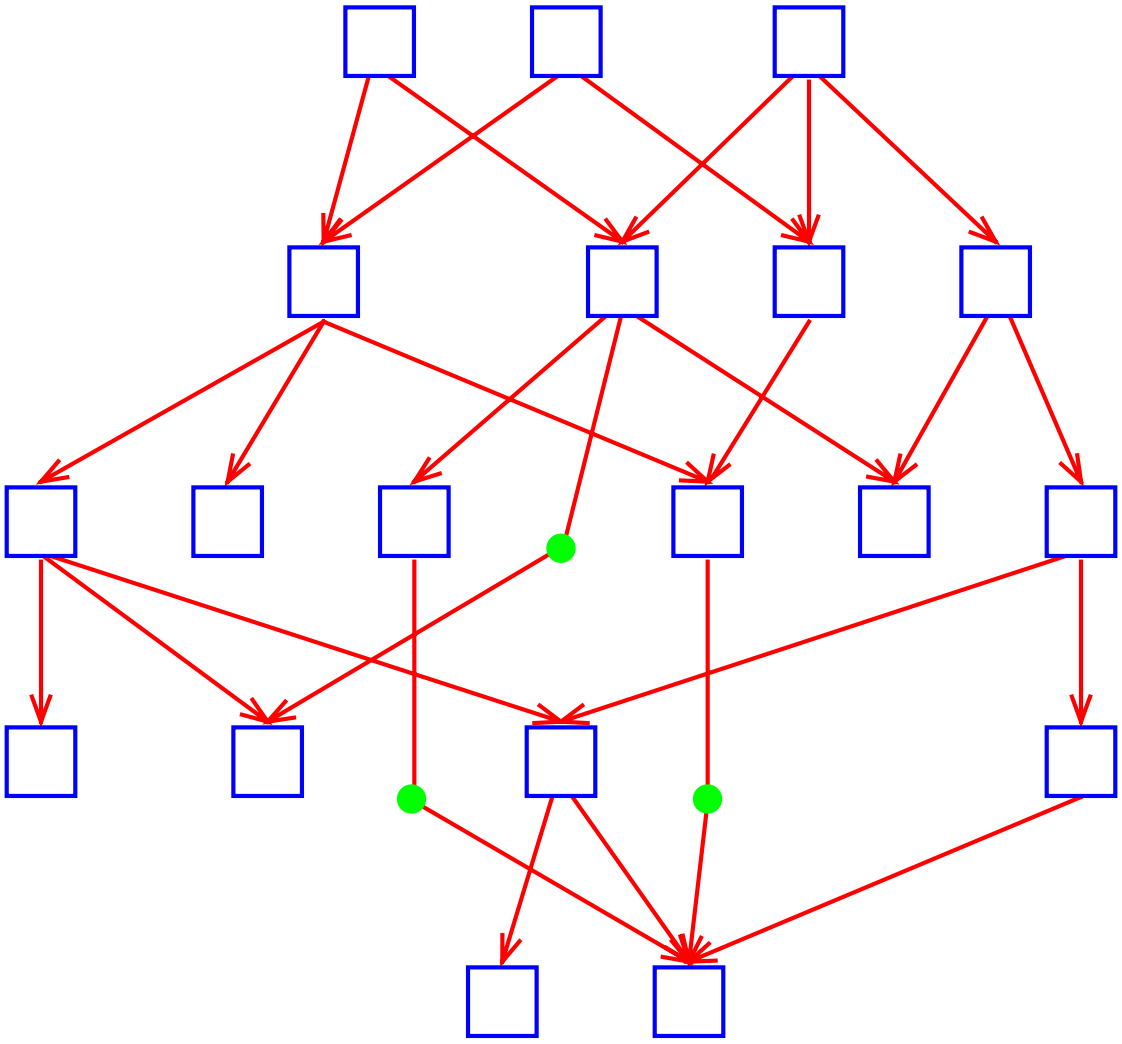
[Sugiyama Tagawa Toda 81]

[Rowe Messinger et al. 87]

[Gansner North 88]

Example

- [Sugiyama Tagawa Toda 81]



Declarative Approaches

Declarative Approach

- **These approaches cover a broad range of possibilities:**
  - **Tightly-coupled**: specification and algorithms cannot be separated from each other.
  - **Loosely coupled**: the specification language is a separate module from the algorithms module.
  - Most of the approaches are somewhere in between ...

Tightly-coupled approaches

Advantages:

- The algorithms can be optimized for the particular specification.
- The problem is well-defined.

Disadvantages:

- Takes an expert to modify the code (difficult extensibility).
- User has less flexibility.

Loosely-coupled approaches

Advantages:

- Flexible: the user specifies the drawing using constraints, and the graph drawing module executes it.
- Extensible: progressive changes can be made to the specification module and to the algorithms module.

Disadvantages:

- Potential “impedance mismatch” between the two modules.
- Efficiency: more difficult to guarantee.

Languages for Specifying Constraints

- Languages for display specification
  - ThingLab [Borning 81]
  - IDEAL [Van Wyk 82]
  - Trip [Kamada 89]
  - GVL [Graham & Cordy 90]
- Grammars
  - Visual Grammars [Lakin 87]
  - Picture Grammars [Golin and Reiss 90]
  - Attribute Grammars [Zinßmeister 93]
  - Layout Graph Grammars [Brandenburg94] [Hickl94]
  - Relational Grammars [Weitzman &Wittenburg 94]
- Visual Constraints
  - U-term language [Cruz 93]
  - Sketching [Gleicher 93] [Gross94]

Visual

Used in GD

Used in GD and Visual

ThingLab [Borning 81]

- Graphical objects are defined by example, and have a ***typical*** part and a ***default*** part.
- Constraints are associated with the classes (methods specify constraint satisfaction).
- Object-oriented (message passing, inheritance).
- Visual programming language.

Ideal [Van Wyk 82]

- Textual specification of constraints.
- Graphical objects are obtained by instantiating
  abstract data types, and adding constraints.
- Uses complex numbers to specify coordinates.

GVL [Graham & Cordy 90]

- Visual language to specify the display of program data structures.
- Pictures can be specified ***recursively*** (the display of a linked list is the display of the first element of the list, followed by the display of the rest of the list.

Layout Graph Grammars

[Brandenburg 94] [Hickl 94]

- grammatical (rule-based method) for drawing graphs
- extension of a ***context-free string grammar***
  - underlying context-free graph grammar
  - layout specification for its productions
- by repeated applications of its productions, a graph grammar generates labeled graphs, which define its graph language
- class of layout graph grammars for which optimal graph drawings can be constructed in polynomial time:
  - H-tree layouts of complete binary trees
  - hv-drawings of binary trees
  - series-parallel graphs
  - NFA state transition diagrams from regular expressions

Picture Grammars

[Golin & Reiss 90, Golin 91]

- **Production rules use constraints.**
- **Terminals are:**
  - *shapes* (e.g., rectangle, circle, text)
  - *lines* (e.g., arrow)
- **spatial relationships between objects are *operators* in the grammar (e.g., over, left_of)**

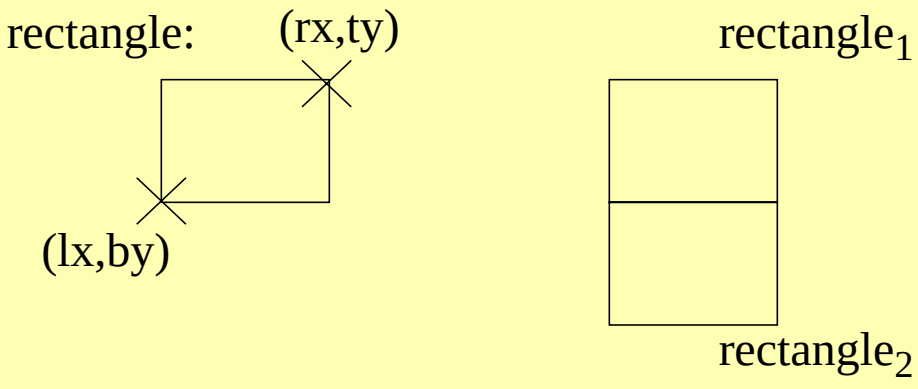
FIGURE → over (rectangle$_1$, rectangle$_2$)

Where

rectangle$_1$.lx == rectangle$_2$.lx

rectangle$_1$.rx == rectangle$_2$.rx

rectangle$_1$.by == rectangle$_2$.ty



- **More expressive relationships : *tiling*.**
- **Complexity of parsing has been studied.**

Relational Grammars

[Weitzman & Wittenburg 93, 94]

- **Generalization of attribute string grammars that allow for the specification of geometric positions in 2D and 3D, topological connectivity, arbitrary semantic relations holding among information objects.**

Article → Text Text Text Number Image

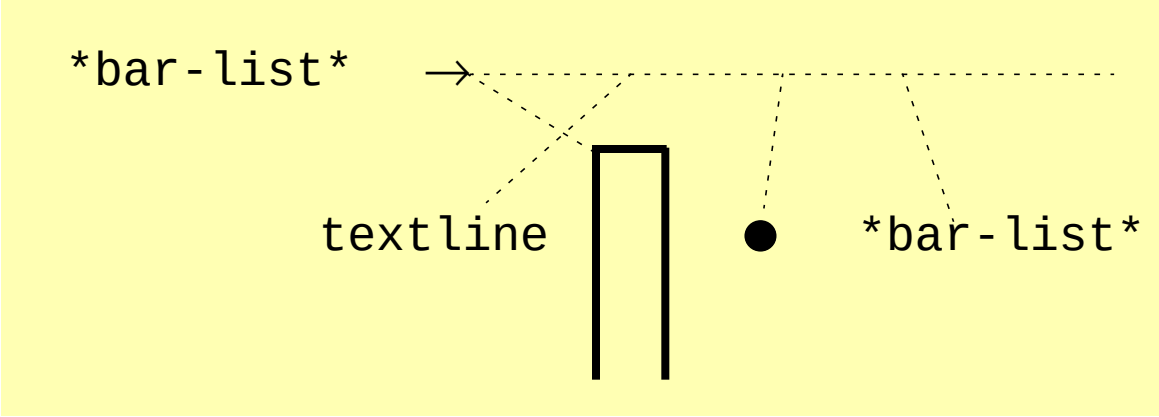
```
(Defrule (Make-Article The-Grammar)
         (0 Article)
         (1 Text)
          (2 Text (Author-Of   2 1))
           . . .

         :OUT
         (
            . . .

         (spaced-below 2 1)
         (spaced-below 3 1)
         (set-font 1 10pt :bold)
         (set-font 1  8pt :italic)

          . . .
                                  ))
```

- **Constraints are solved with DeltaBlue (U. of Washington) for non-cyclic constraints.**

Visual Grammars

[Lakin 87]

- **Contex-free grammar.**
- **Symbols are visual, and are visually annotated.**



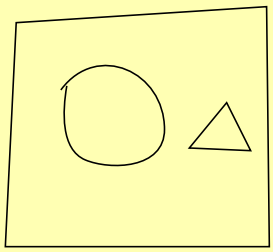
- **The interpretation of the visual symbols is left to the implementation.**

Expressing Constraints by Sketching

- **Briar [Gleicher 93]**

  Constraint-based drawing program:
  - Direct manipulation drawing techniques.
  - Makes relationships between graphical objects persistent
  - Performance concerns in solving constraints.

- **Spatial Relation Predicates [Gross 94]**



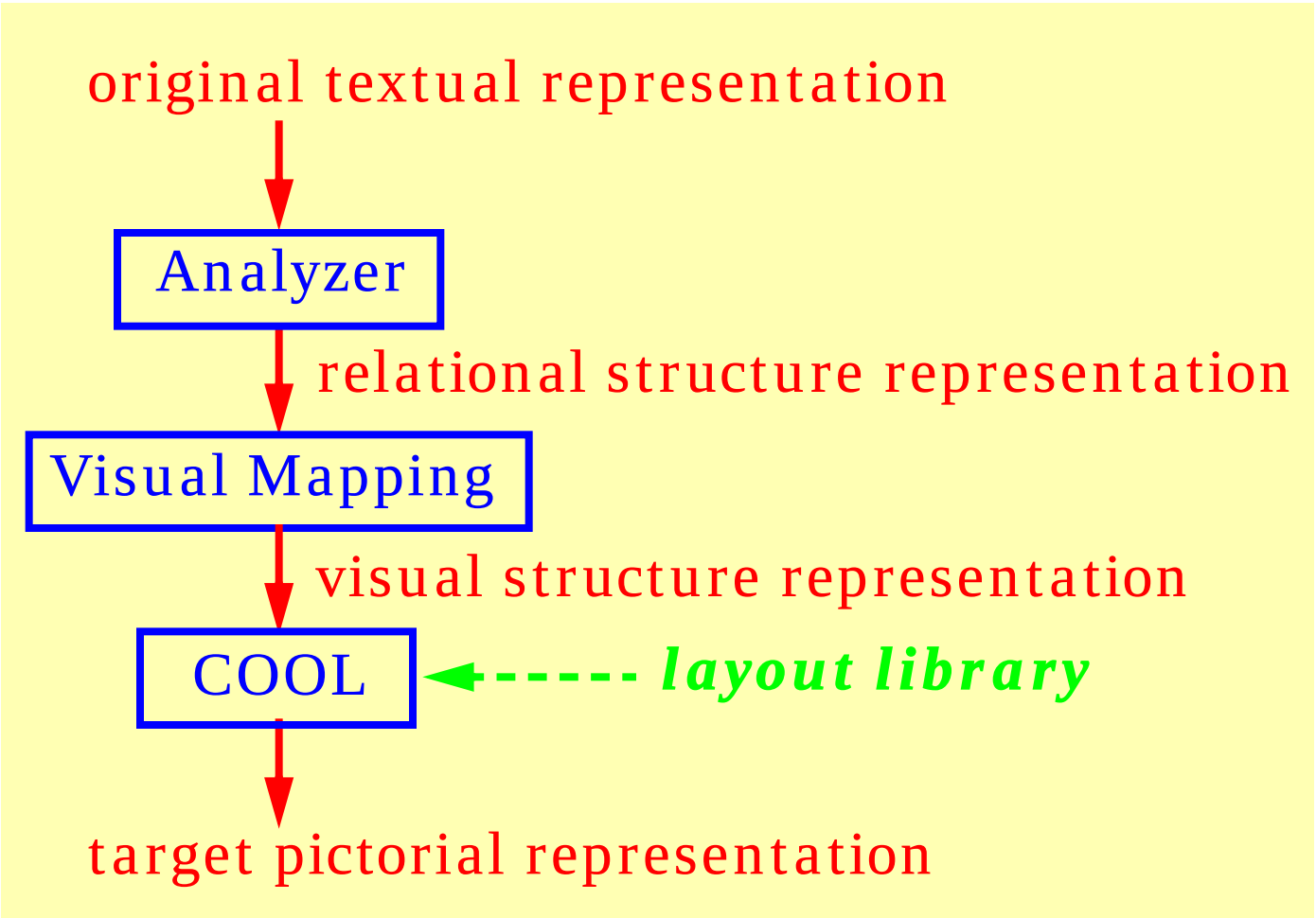
(CONTAINS BOX CIRCLE)
(CONTAINS BOX TRIANGLE)
(IMMEDIATELY-RIGHT-OF CIRCLE TRIANGLE)
(SAME-SIZE CIRCLE TRIANGLE)

  - Applications include retrieval of buildings from an architecture database.

COOL

[Kamada 89]

- framework for visualizing abstract objects and relations.
- constraint-based object layout system
  - ***rigid*** constraints
  - ***pliable*** constraints
  - conflicting constraints can be solved approximately



ANDD

[Marks et al]

- layout-aesthetic concerns subordinated to perceptual-organizational concerns
- notation for describing the visual organization of a network diagram
  - alignment, zoning, symmetry, T-shape, hub shape
- layout task as a constrained optimization problem:
  - constraints derived from a visual-organization specification
  - optimality criteria derived from layout-aesthetic considerations
- two heuristic algorithms:
  - rule-based strategy
  - massive parallel genetic algorithm

Visual Graph Drawing

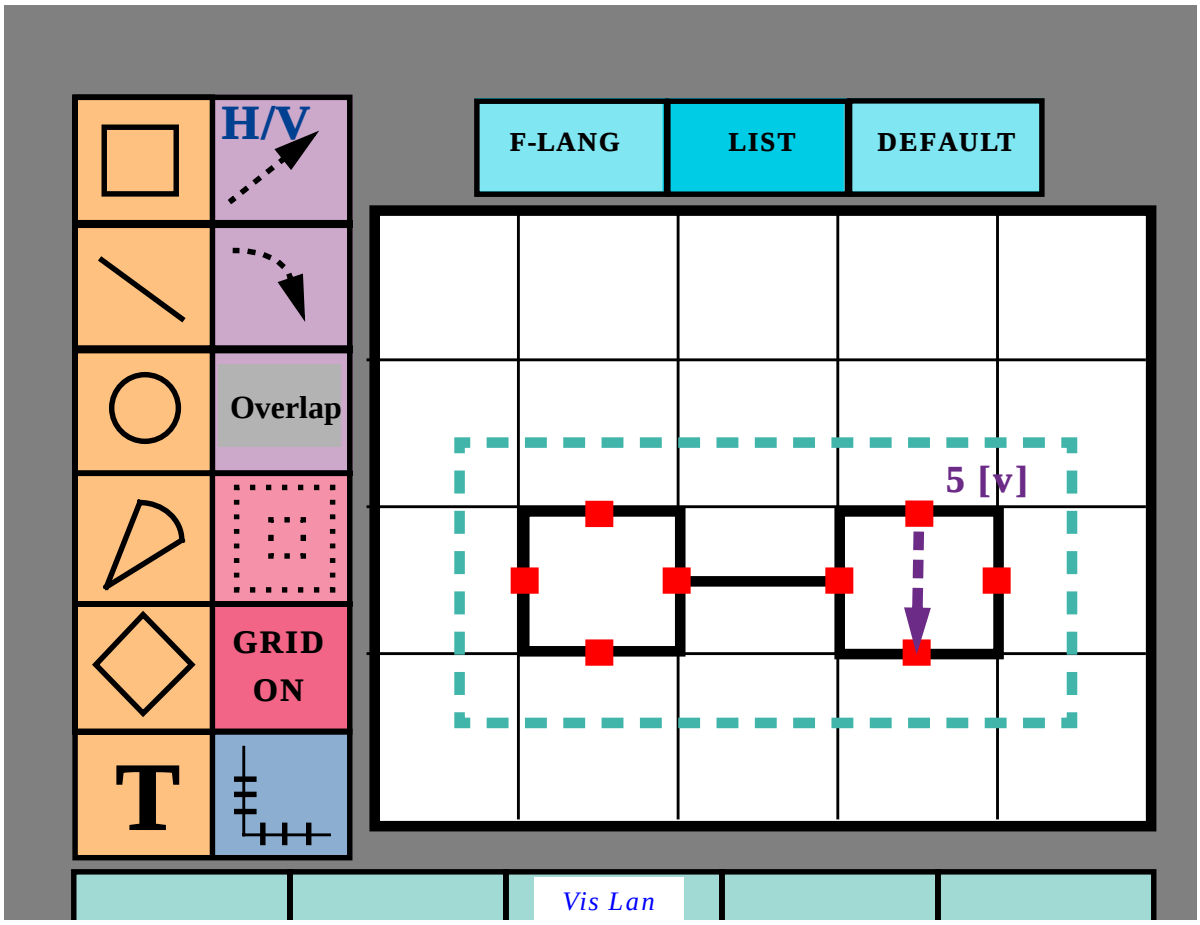
[Cruz, Tamassia Van Hentenryck 93]

- a ***visual*** approach to graph drawing can reconcile ***expressiveness*** with ***efficiency***
- **Goals**
  - ***Visual*** specification of layout ***constraints***: the user should not have to type a long list of textual specifications
  - ***Visual*** specification of aesthetic criteria associated with ***optimization*** problems
  - ***Extensibility***: the user should not be limited to a prespecified set of visual representations.
  - ***Flexibility***: the user should not have to give precise geometric specifications.

U-term Language

[Cruz 93, 94]

- **Visual constraints.**
- **Simplicity and genericity of the basic constructs.**
- **Ability to specify a variety of displays: graphs, higraphs, bar charts, pie charts, plot charts, . . .**
- **Compatibility with the framework of an object-oriented database language, DOODLE.**
- **Recursive visual specification.**

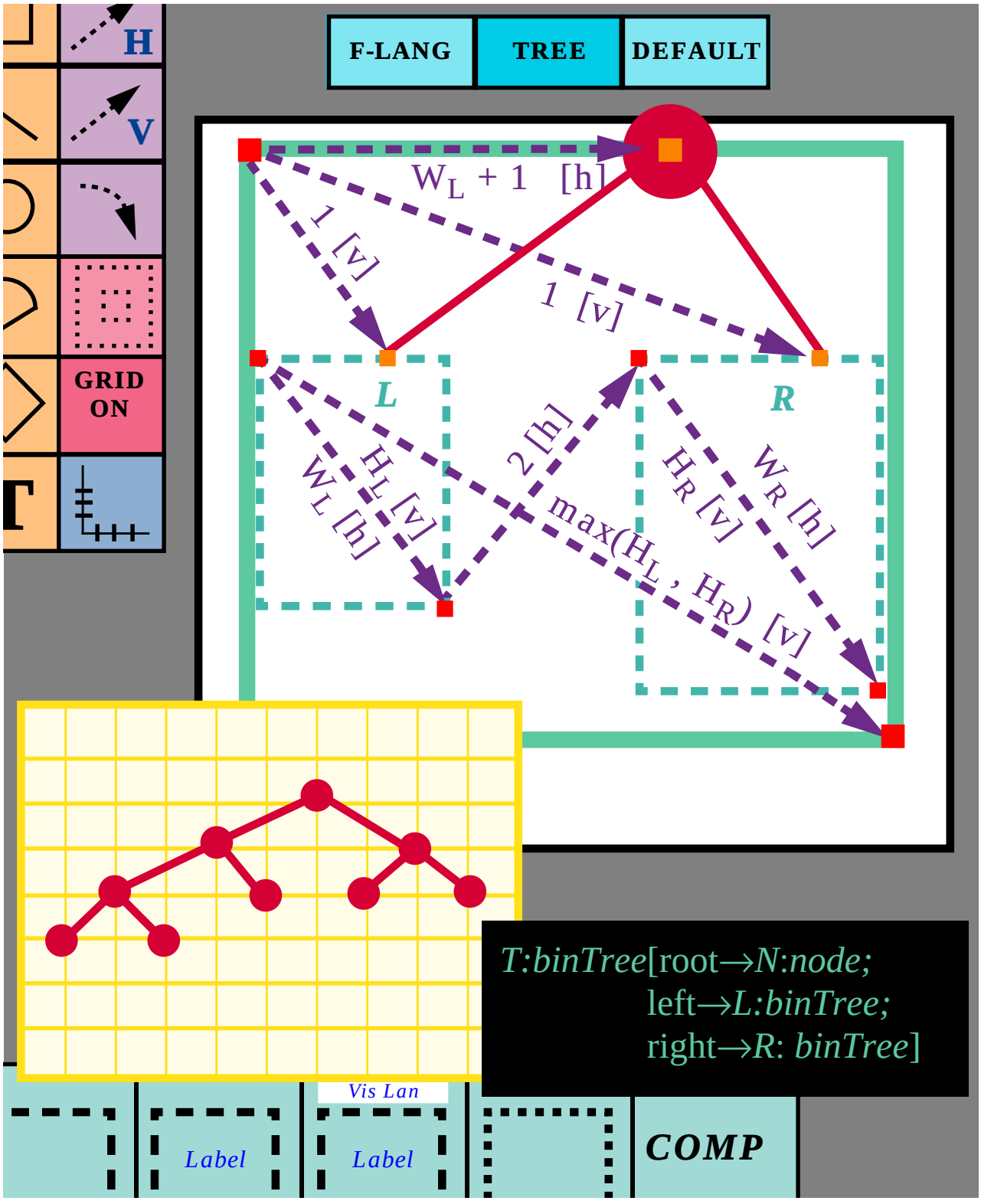


Efficient Visual Graph Drawing

[Cruz Garg 94] [Cruz Garg Tamassia 95]

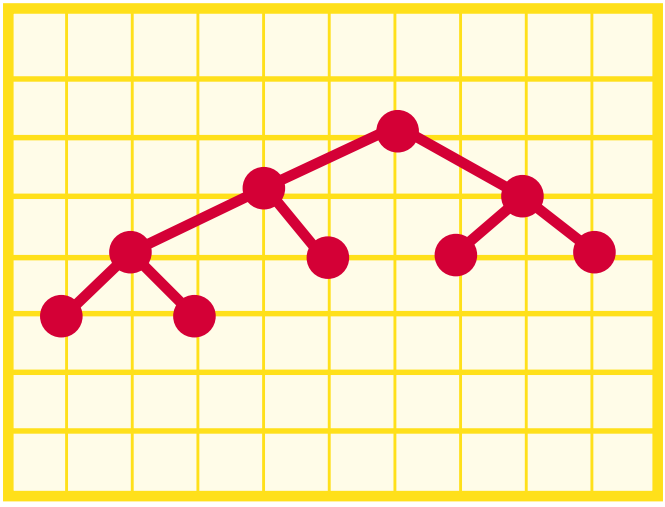
- graph stored in an object-oriented database
- drawing defined "***by picture***" using recursive visual rules of the language DOODLE [Cruz 92]
- a set of ***constraints*** is generated by the application of the visual rules to the input graph
- various types of drawings can be visually expressed in such a way that the resulting set of constraints can be solved in ***linear time***, e.g.,
  - drawings of trees (upward drawings, box inclusion drawings)
  - drawings of series-parallel digraphs (delta drawings)
  - drawings of planar acyclic digraphs (visibility drawings, upward planar polyline drawings)

Tree Layout

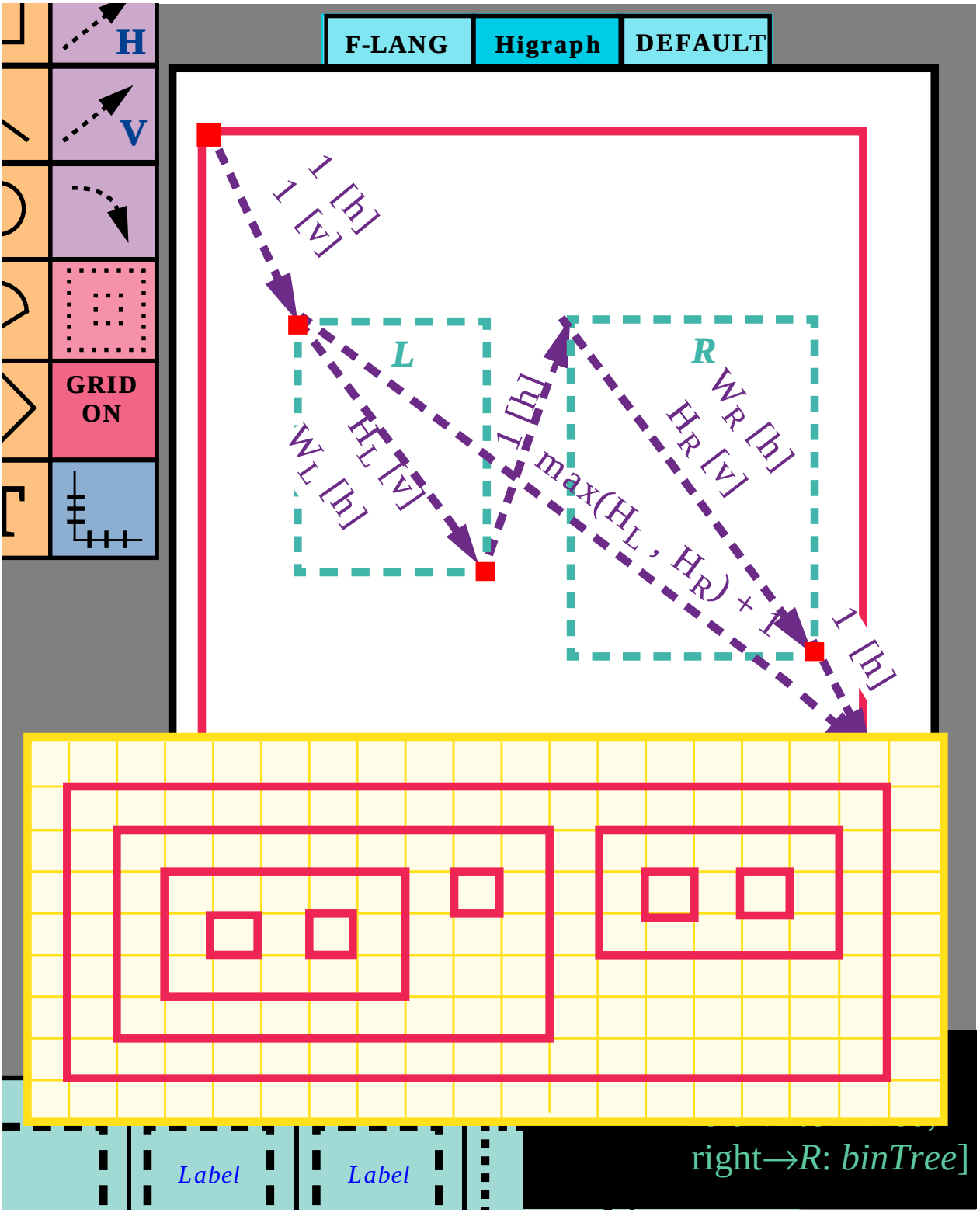

H
V
GRID ON
F-LANG
TREE
DEFAULT
$W_L + 1$ [h]
1 [v]
1 [v]
L
R
H_L [v]
W_L [h]
2 [h]
W_R [h]
H_R [v]
max(H_L, H_R) [v]
T:binTree[root→N:node;
left→L:binTree;
right→R: binTree]
Vis Lan
Label
Label
COMP

Characteristics of the Previous Tree Drawings

- Level Drawings
  - Upward
  - Planar
  - Nodes at the same distance from the root are horizontally aligned.
- Display of symmetries.
- Display of isomorphic subtrees.



Change a few things . . .



Efficient Visual Graph Drawing

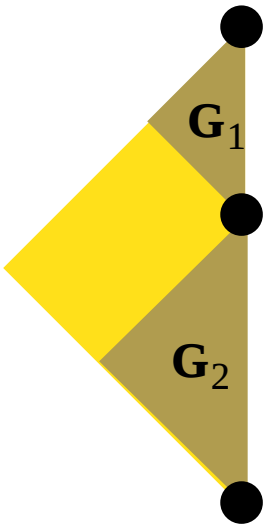
[Cruz & Garg 94]

- **Recognize classes of graphs and drawings that can be expressed with DOODLE and evaluated efficiently.**
- **Devise algorithms and data structures for performing drawings in linear time (optimal time):**
  - Trees (upward drawing, box inclusion drawing).
  - Series-parallel digraphs (delta drawing).
  - Planar acyclic digraphs (visibility drawing, upward planar polyline drawing).
- **Next:**
  - Extend above results to other classes of graphs and drawings.
  - Constraint viewpoint: framework for evaluating constraints efficiently.
  - Incorporate these algorithms into a declarative graph drawing system that uses **DOODLE**.

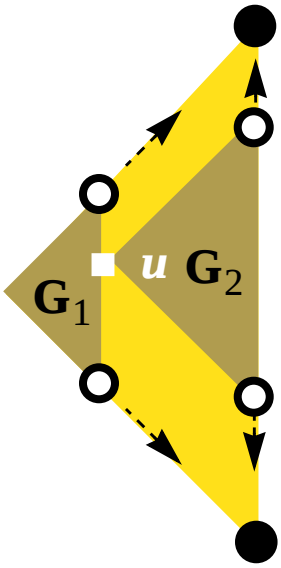
More examples

- Series-parallel graphs / delta-drawings [Bertolazzi, Cohen, Di Battista, Tamassia & Tollis, 92]

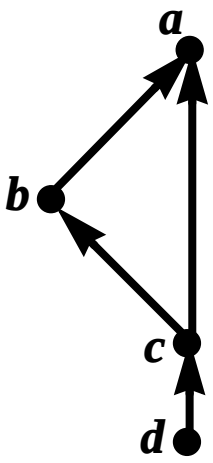
Base case



Series composition

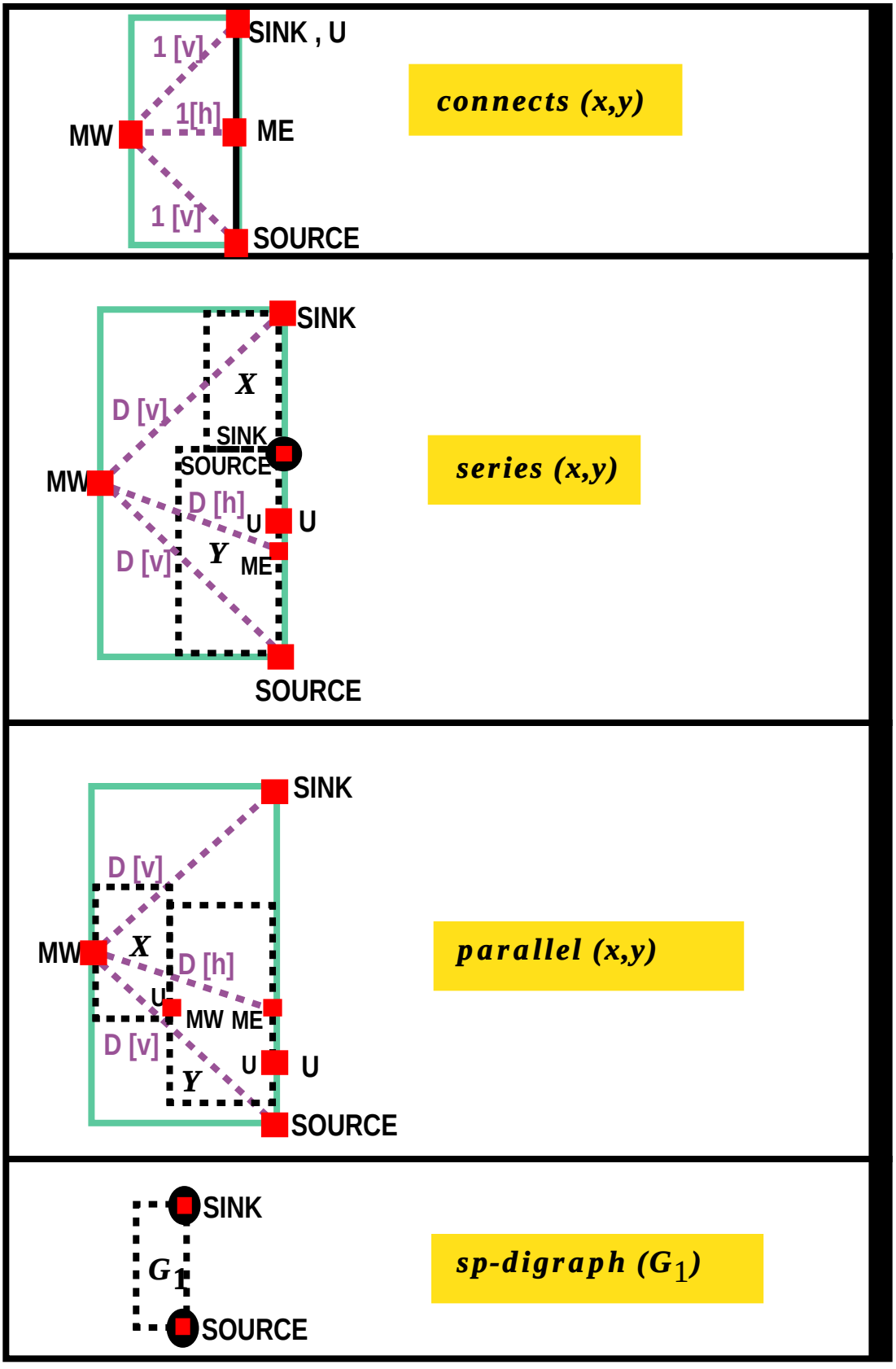


Parallel composition



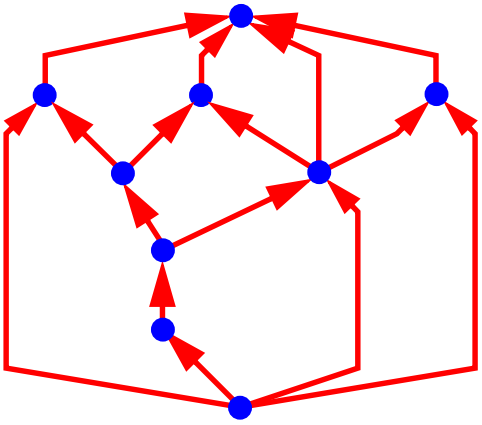
Example

deltaGraph

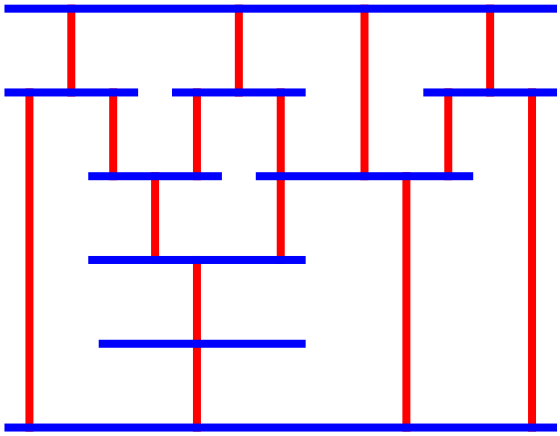

SINK , U
1 [v]
1[h]
MW
ME
1 [v]
SOURCE
connects (x,y)
SINK
X
D [v]
SINK
SOURCE
MW
D [h]
U
U
Y
D [v]
ME
SOURCE
series (x,y)
SINK
D [v]
MW
X
D [h]
U
MW ME
D [v]
U
U
Y
SOURCE
parallel (x,y)
SINK
G1
SOURCE
sp-digraph (G1)

Drawings of Planar DAGs

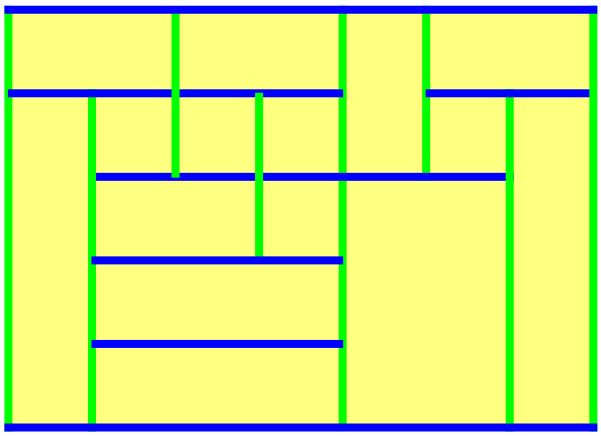
- planar upward drawing



- visibility drawing



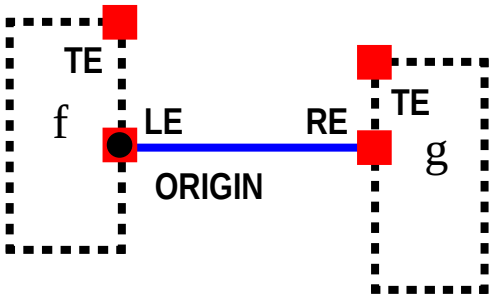
- tessellation drawing



Tessellation Drawing

TessellationDrawing

v: sourceVertex

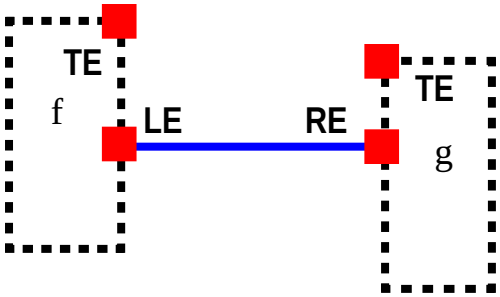


F-Language

v: sourceVertex [leftFace → f : face ;
rightFace → g: face]

TessellationDrawing

v: vertex

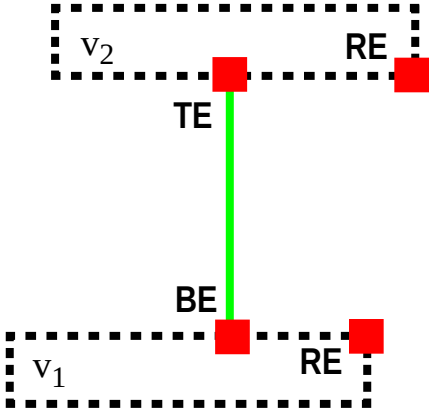


F-Language

v: vertex [leftFace → f : face ;
rightFace → g: face]

TessellationDrawing

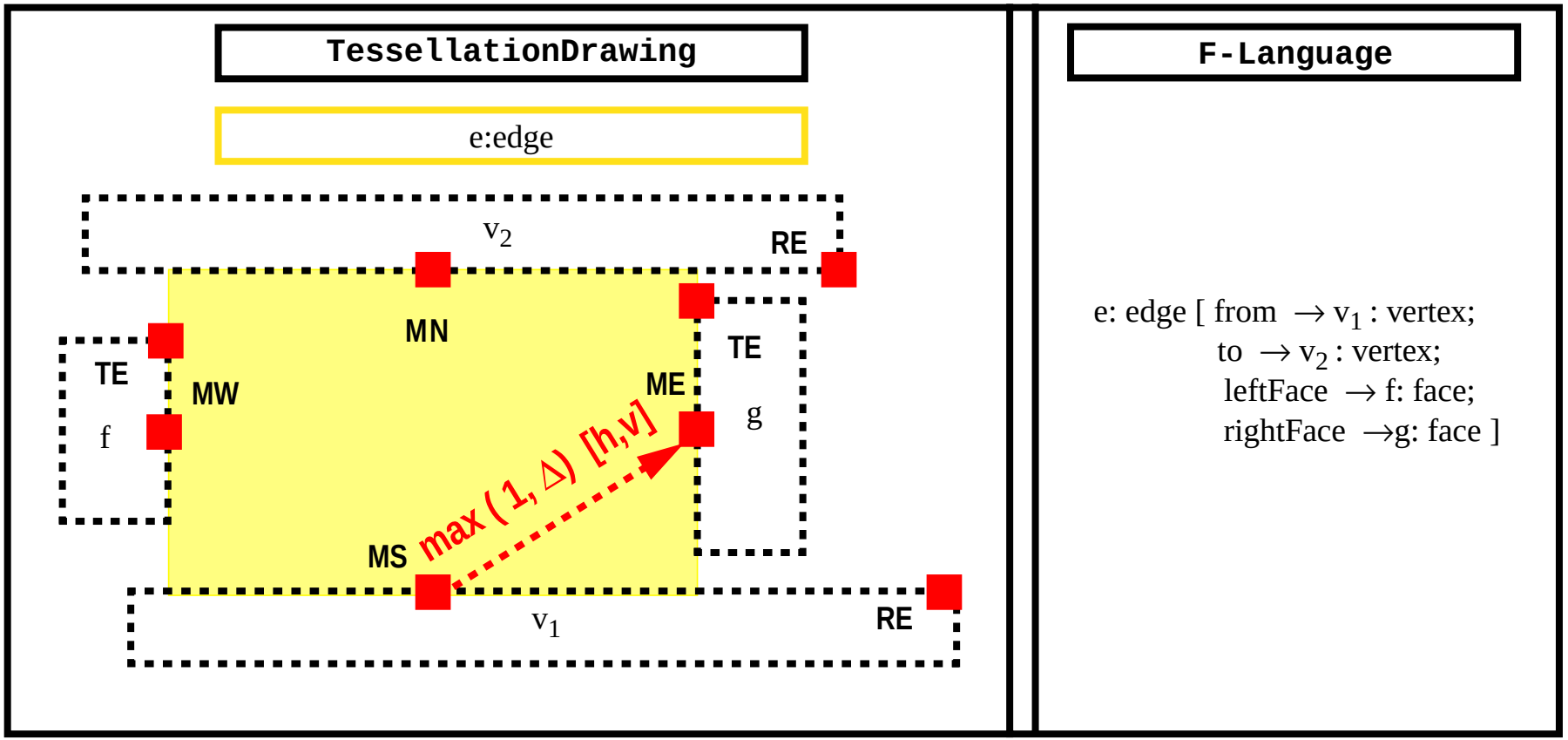
f: face



F-Language

f: face [α→ v_2: vertex ;
bottomVertex → v_1: vertex]

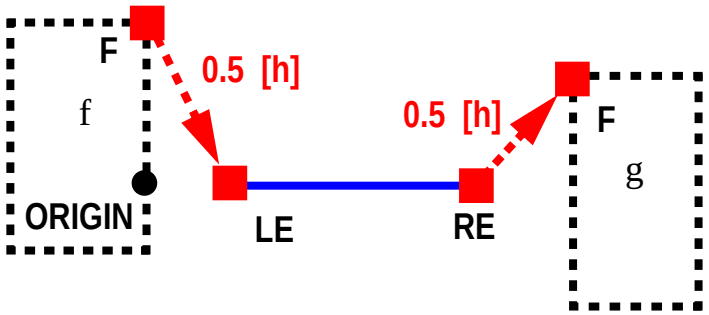
Tessellation Drawing



Visibility Drawing

VisibilityDrawing

v: sourceVertex

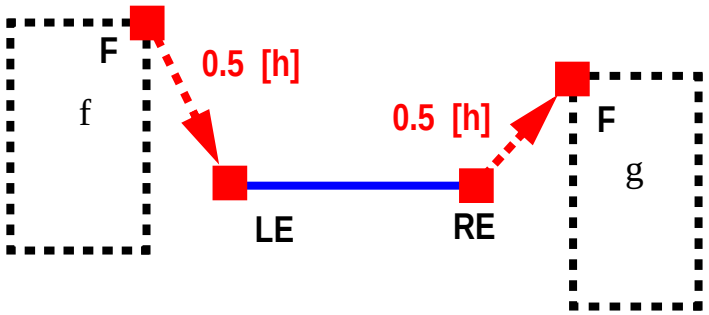


F-Language

v: sourceVertex [leftFace → f : face ;
rightFace → g: face]

VisibilityDrawing

v: vertex



F-Language

v: vertex [leftFace → f : face ;
rightFace → g: face]

Visibility Drawing

VisibilityDrawing

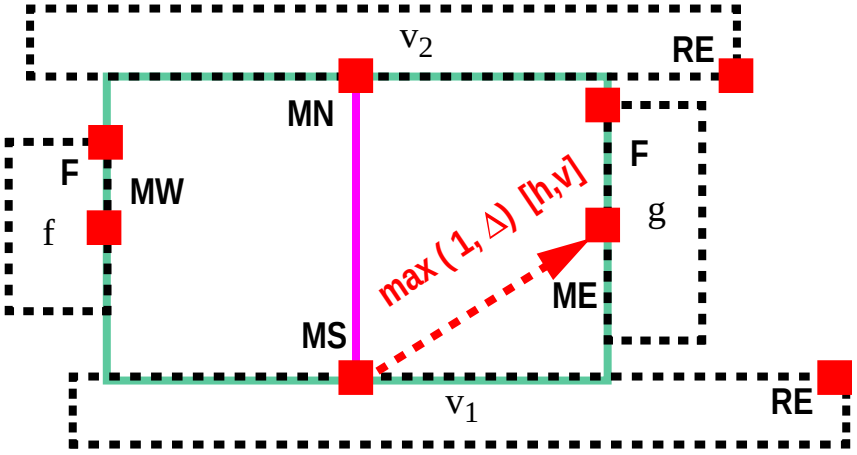
f: face

F

f: face

VisibilityDrawing

e:edge



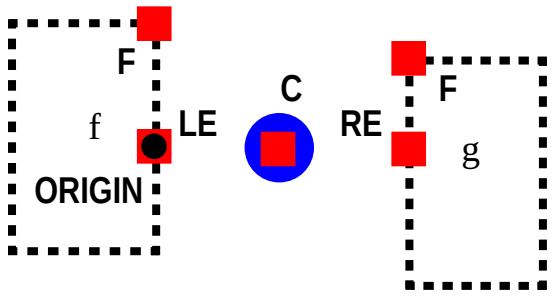
F-Language

e: edge [from → v_1 : vertex;
to → v_2 : vertex;
leftFace → f: face;
rightFace →g: face]

Upward Polyline Drawing

PolylineDrawing

v: sourceVertex

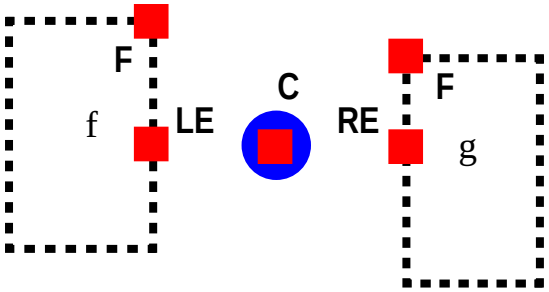


F-Language

v: sourceVertex [leftFace → f : face ;
rightFace → g: face]

PolylineDrawing

v: vertex



F-Language

v: vertex [leftFace → f : face ;
rightFace → g: face]

Upward Polyline Drawing

PolylineDrawing

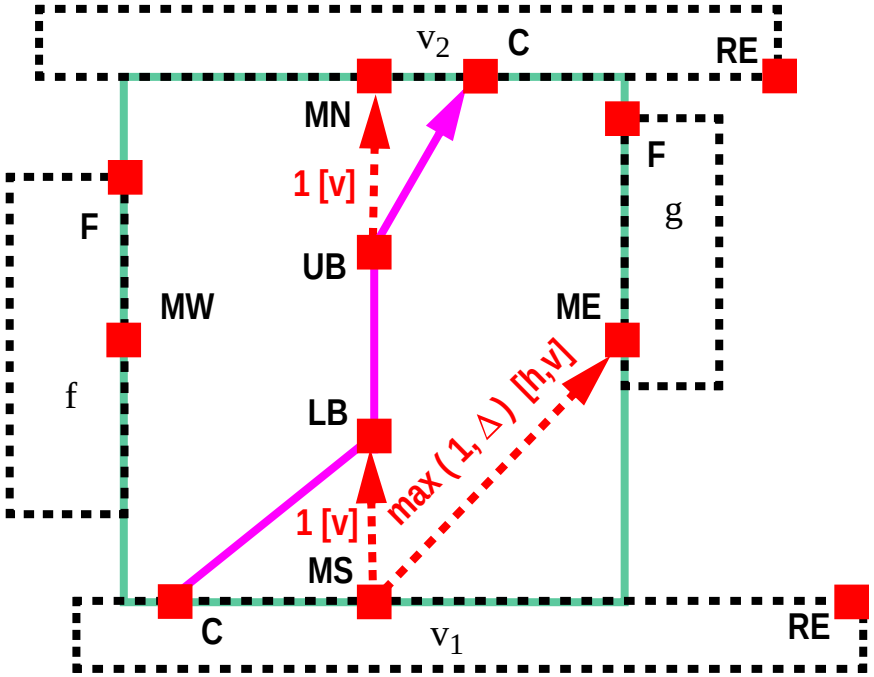
f: face

F-Language

f: face

PolylineDrawing

e:edge



F-Language

e: edge [from $\rightarrow v_1$: vertex;
to $\rightarrow v_2$: vertex;
leftFace $\rightarrow$ f: face;
rightFace $\rightarrow$g: face]

Challenges and Open Problems (Declarative Approach):

- New approach, therefore much left to explore, in particular:
  - New specification languages.
  - Reducing the "impedance mismatch."
  - Design of user interfaces, and evaluation in different environments/ applications.
  - Identification of levels of complexity in drawing graphs (e.g., with graph grammars, constraint languages).
  - Expressiveness of the specification languages, in particular of declarative and visual languages.
  - Refinement of the *diagram server* hierarchy, so that we can have a true "tool box" for the declarative, loosely-coupled approach.

Systems

Some Graph Drawing Systems

- ***Graph Drawing Server***
  (Brown University, USA)
  - `loki.cs.brown.edu:8081/graphserver/`
  - Roberto Tamassia (`rt@cs.brown.edu`)
- ***GDToolkit***
  (University of Rome III)
  - `www.dia.uniroma3.it/people/gdb/wp12/GDT.html`
  - Giuseppe Di Battista
    (`dibattista@iasi.rm.cnr.it`)
- ***Graphlet***
  (University of Passau, Germany)
  - `www.fmi.uni-passau.de/Graphlet/`
  - Michael Himsolt
    (`himsolt@fmi.uni-passau.de`)
- ***GraphViz***
  (AT&T Research)
  - `www.research.att.com/sw/tools/graphviz/`
  - Sthephen North (`north@research.att.com`)