A Split&Push Approach to 3D Orthogonal Drawing*

(Extended Abstract)

Giuseppe Di Battista[1], Maurizio Patrignani[1], and Francesco Vargiu[2]

[1] Dipartimento di Informatica e Automazione, Università di Roma Tre
via della Vasca Navale 79, 00146 Roma, Italy. {gdb,patrigna}@dia.uniroma3.it
[2] AIPA, via Po 14, 00198 Roma Italy. vargiu@aipa.it

Abstract. We present a method for constructing orthogonal drawings of graphs of maximum degree six in three dimensions. Such a method is based on generating the final drawing through a sequence of steps, starting from a "degenerate" drawing. At each step the drawing "splits" into two pieces and finds a structure more similar to its final version. Also, we test the effectiveness of our approach by performing an experimental comparison with several existing algorithms.

1 Introduction

Both for its theoretical appeal and for the high number of potential applications, research in 3D graph drawing is attracting an increasing attention. The interest of the researchers has been mainly devoted to straight-line drawings and orthogonal drawings.

Concerning straight-line drawings, many different approaches can be found in the literature. For example, the method in [7] is based on carefully exploiting the "momentum curve" to guarantee no edge crossings and a volume $4n^3$, where n is the number of vertices of the graph to be drawn. The same paper presents another example of algorithm which constructs drawings without edge crossings of planar graphs with degree at most 4. It is based on folding a 2-dimensional orthogonal grid drawing of area $h \times v$ into a straight-line drawing with volume $h \times v$. Force directed approaches have been exploited to devise the algorithms in [5, 8, 10, 18, 25, 14, 20]. Also, the research on this type of drawings stimulated the investigation on theoretical bounds. Examples of bounds on the volume of a straight-line drawing can be found in [7, 6, 21]. Further, special types of straight-line drawings have been studied in [3, 13, 1, 15] (visibility representations) and in [17] (proximity drawings).

Concerning orthogonal drawings, all the algorithms guarantee no intersection between edges and most of the results apply mainly to graphs with maximum vertex degree six. Biedl [2] shows a linear time algorithm (in what follows we call it `Slice`) that draws in $O(n^2)$ volume with at most 14 bends per edge. Eades,

* Research supported in part by the ESPRIT LTR Project no. 20244 - ALCOM-IT and by the CNR Project "Geometria Computazionale Robusta con Applicazioni alla Grafica ed al CAD."

S.H. Whitesides (Ed.): GD'98, LNCS 1547, pp. 87–101, 1998.

Stirk, and Whitesides [11] propose a $O(n^{3/2})$-time algorithm based on augmenting the graph to an Eulerian graph and on applying a variation of an algorithm by Kolmogorov and Bardzin [16]. The algorithm (we call it `Kolmogorov`) draws in $O(n^{3/2})$ volume with at most 16 bends per edge. Eades, Symvonis, and Whitesides [12] propose two algorithms. Both work in $O(n^{3/2})$ time and are based on augmenting the graph to a 6-regular graph and on a coloring technique. A first algorithm (we call it `Compact`) draws in $O(n^{3/2})$ volume with at most 7 bends per edge while a second algorithm (we call it `Three-Bends`) draws in at most $27n^3$ volume with at most 3 bends per edge. Papakostas and Tollis [22] present a linear time algorithm (we call it `Interactive`) that draws in at most $4.66n^3$ volume with at most 3 bends per edge. The algorithm can be extended to draw graphs with vertices of arbitrary degree. Finally, Wood [26] presents an algorithm for maximum degree 5 graphs that draws in $O(n^3)$ volume with at most 2 bends per edge. Although the results presented in the above papers are interesting and deep, the research in this field suffers, in our opinion, the lack of general methodologies.

In this paper we deal with the problem of constructing orthogonal drawings in three dimensions. We experiment several existing algorithms to test their practical applicability. Further, we propose new techniques that have a good average behaviour. Our main target are graphs with number of vertices in the range 10–100 that are crucial in several applications. The results presented in this paper can be summarized as follows. We present a new method for constructing orthogonal drawings of graphs of maximum degree six in three dimensions without intersections between edges. It can be considered more as a general strategy rather than as a specific algorithm. The approach is based on generating the final drawing through a sequence of steps, starting from a "degenerate" drawing; at each step the drawing "splits" into two pieces and finds a structure more similar to its final version. We devise an example of algorithm developed according to the above method, called `Reduce-Forks`. We perform an experimental comparison of `Compact`, `Interactive`, `Kolmogorov`, `Reduce-Forks`, `Slice`, and `Three-Bends` against a large test suite of graphs with at most 100 vertices. We measure the computation time and three important readability parameters: volume, average edge length, and average number of bends along edges. Our experiments show that no algorithm can be considered "the best" with respect to all the parameters. Concerning `Reduce-Forks`, we can say that it has a good effectiveness for graphs in the range 5–30 and, among the algorithms that have a reasonable number of bends along the edges (`Interactive`, `Reduce-Forks`, and `Three-Bends`), `Reduce-Forks` is the one that has the best behaviour in terms of edge length and volume. This is obtained at the expenses of an efficiency that is much worse than the other algorithms. However, the CPU time do not seem to be a critical issue for the size of graphs in the interval.

The paper is organized as follows. In Section 2 we present our approach and in Section 3 we show its feasibility. In Section 4 we describe Algorithm `Reduce-Forks`. In Section 5 we present the results of the experimental comparison.

The interested reader will find at our Web site a cgi program that allows to use the experimented algorithms, and the test suite used in the experiments (`www.dia.uniroma3.it/~patrigna/3dcube`).

2 A Strategy for Constructing 3D Orthogonal Drawings

An *orthogonal drawing* of a graph is such that all the edges are chains of segments parallel to the axes. A *grid* drawing is such that all vertices and bends have integer coordinates. A *01-drawing* is an orthogonal grid drawing such that each edge has either length 0 or length 1 and vertices may overlap. A *0-drawing* is a trivial 01-drawing such that each edge has length 0 and all vertices have the same coordinates. A *1-drawing* is a 01-drawing such that all edges have length 1 and vertices have distinct coordinates. (See Fig. 1.) Observe that while all graphs have a 01-drawing, only some admit a 1-drawing.

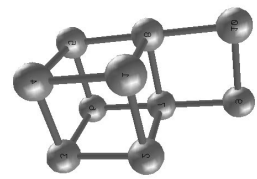


Fig. 1. A 1-drawing of a graph with ten vertices.

Let G be a graph. A *subdivision* G_1 of G is a graph obtained from G by replacing some edge of G with a path. A subdivision G_2 of G_1 is a subdivision of G. There always exists a subdivision of G that admits a 1-drawing. We partition the vertices of G_1 into vertices that belong also to G (*original vertices of* G) and vertices that belong only to G_1 (*dummy vertices*).

A *dummy path* of G_1 is a path consisting only of dummy vertices but, possibly, at the endpoints (that can be original vertices). A *planar path* of an orthogonal drawing of G_1 is a maximal path whose vertices are on the same plane. A planar dummy path is *self-intersecting* if it has two distinct vertices with the same coordinates.

We propose a method for constructing orthogonal grid drawings with all vertices at distinct coordinates and without intersections between edges (except at the common endpoints). The drawing process consists of a sequence of steps such that each step maps a 01-drawing of a graph G into a 01-drawing of a subdivision of G. We start with a 0-drawing of G and at the last step we get a 1-drawing of a subdivision G_1 of G. Hence, an orthogonal grid drawing of G is obtained by replacing each path u, v of G_1, corresponding to an edge (u, v) of G, with an orthogonal polygonal line connecting u and v.

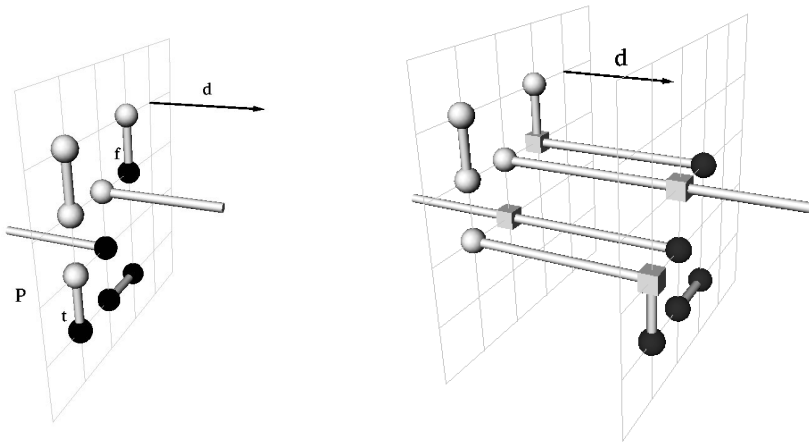
The general strategy is as follows. Let G_0 be a graph, we consider several subsequent subdivisions of G_0. In each subdivision new dummy vertices are

introduced. In each subdivision we call *original* the original vertices of G_0 and *dummy* all the other vertices. We construct an orthogonal grid drawing Γ of G_0 in four steps. **Vertex Scattering:** Construct a 01-drawing of a subdivision G_1 of G_0 such that all the original vertices have different coordinates and all planar dummy paths are not self-intersecting (we call it *scattered 01-drawing*). After this step dummy vertices may still overlap both with dummy and with original vertices. **Direction Distribution:** Construct a scattered 01-drawing of a subdivision G_2 of G_1 such that, for each vertex v, v and all its adjacent vertices have different coordinates (we call it *direction-consistent 01-drawing*). In other words, after this step the edges incident on v "leave" v with different directions. Observe that this is true both in the case v is original and in the case v is dummy. **Vertex-Edge Overlap Removal:** Construct a direction-consistent 01-drawing of a subdivision G_3 of G_2 such that for each original vertex v, no dummy vertex has the same coordinates of v (we call it *vertex-edge-consistent 01-drawing*). After this step the original vertices do not "collide" with other vertices. Observe that dummy vertices having the same coordinates may still exist. **Crossing Removal:** Construct a 1-drawing of a subdivision G_4 of G_3 (all the vertices, both original and dummy, have different coordinates). Observe that Γ is easily obtained from the drawing of G_4.

Each step is performed by repeatedly applying the same simple primitive operation called *split*. Informally, such operation "cuts" the entire graph with a plane perpendicular to one of the axes. The vertices laying on the "cutting" plane are split into two subsets that are "pushed" into two adjacent planes.

Given a direction d we denote by $-d$ its opposite direction. A *split parameter* is a 4-tuple (d, P, ϕ, ρ), where d is a direction and P is a plane perpendicular to d. Function ϕ maps each vertex laying on P to a boolean. Function ρ maps each edge (u, v) laying on P such that $\phi(u) \neq \phi(v)$ and such that u and v have different coordinates to a boolean. Given a split parameter (d, P, ϕ, ρ), a $split(d, P, \phi, \rho)$ performs as follows (see Fig. 2). (1) We move of one unit in the d direction all vertices in the open half-space determined by P and d. Such vertices are "pushed" towards d. (2) We move of one unit in the d direction each vertex u on P with $\phi(u) = true$. (3) For each edge (u, v) that after the above steps has length greater than one, we substitute (u, v) with the new edges (u, w) and (w, v), where w is a new dummy vertex. Vertex w is placed as follows. (3.a) If the function $\rho(u, v)$ is not defined, then vertex w is simply put in the middle point of the segment u, v. (3.b) If the function $\rho(u, v)$ is defined (suppose, wlog, that $\phi(u) = true$ and $\phi(v) = false$), then two cases are possible. If $\rho(u, v) = true$, then w is put at distance 1 in the d direction from u. If $\rho(u, v) = false$, then w is put at distance 1 in the $-d$ direction from v.

Observe that a *split* operation applied to a 01-drawing of a graph G constructs a 01-drawing of a subdivision of G. Also, although *split* is a simple primitive, it has several degrees of freedom that can lead to very different drawing algorithms. Further, by applying *split* in a "random" way there is no guarantee that the process converges to a 1-drawing.



(a) (b)

Fig. 2. An example of split: (a) before the split and (b) after the split. Vertices with $\phi = true$ ($\phi = false$) are black (light grey). Edges with $\rho = true$ ($\rho = false$) are labelled t (f). The little cubes are dummy vertices inserted by the *split*.

3 Feasibility of the Approach

Property 1. Let Γ be a scattered 01-drawing of a graph. Each edge of Γ incident to a dummy vertex has length 1.

Property 2. Let Γ be a 01-drawing of a graph obtained by a sequence of split operations from a 0-drawing of another graph. Each edge of Γ incident to a dummy vertex has length 1.

Proof. Dummy vertices are created by *split* operations. In such operations they are placed at distance 1 from their neighbors.

Property 3. Let Γ_0 be a 0-drawing of a graph G_0. There exists a finite sequence of split operations that, starting from Γ_0, constructs a scattered 01-drawing of a subdivision of G_0.

Proof. Trivial. All the splits can be performed with planes perpendicular to the same axis. Each split separates one original vertex from the others. Note that all the obtained dummy paths are drawn as straight lines and hence are not self-intersecting and that all vertices lie on the same line.

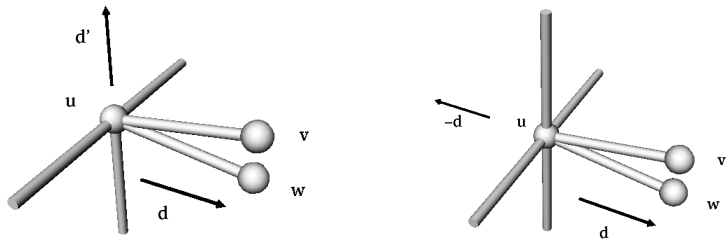
Let u be a vertex. We call the six directions isothetic wrt the axes around u *access directions* of u. The access direction of u determined by traversing edge

(u, v) from u to v is *used* by (u, v). An access direction of u that is not used by any of its incident edges is a *free direction* of u.

Given a direction d isothetic to one of the axes and a vertex v, we denote $P_{d,v}$ the plane through v perpendicular to d.

Theorem 1. *Let Γ_1 be a scattered 01-drawing of a graph G_1, subdivision of a graph G_0. There exists a finite sequence of split operations that, starting from Γ_1 constructs a direction-consistent 01-drawing of a subdivision of G_1.*

Proof. We consider one by one each vertex u with edges (u, v) and (u, w) that use the same access direction d of u. Since Γ_1 is a scattered 01-drawing, at least one of v and w (say v) is dummy. Also, by Property 1 we have that all edges incident to u use a direction of u. Two cases are possible. Case 1: at least one direction d' of the free directions of u is orthogonal to d; see Fig. 3.a. Case 2: direction $-d$ is the only free direction of u; see Fig. 3.b.



(a) (b)

Fig. 3. Cases in the proof of Theorem 1.

Case 1: We perform $split(d', P_{d',u}, \phi, \rho)$ as follows. We set $\phi(v) = true$, all the other vertices of $P_{d',u}$ have $\phi = false$. Also, $\rho(u, v) = true$, all the other edges in the domain of ρ have $\rho = false$. After *split*, edge (u, v) uses direction d' of u. The usage of the other access directions of u is unchanged. Also, all the other vertices still use the same access directions as before the split with the exception, possibly, of v (that is dummy).

Case 2: Let d'' be a non-free direction of u different from d. We perform the same split operation as the one of Case 1, using direction d'' instead of d'. After *split*, edge (u, v) uses direction d'' of u. At this point, since at least one direction of the free directions of u is orthogonal to d'', we can apply the same *split* strategy of Case 1.

Finally, we observe that the above split operations preserve the properties of the scattered 01-drawings.

We define a simplier version of $split(d, P, \phi, \rho)$, called $trivialsplit(d, P)$, where ϕ is identically $false$ for all vertices of the cutting plane, and, as a consequence, the domain of ρ is empty.

Theorem 2. *Let Γ_2 be a direction-consistent 01-drawing of a graph G_2, subdivision of a graph G_0. There exists a finite sequence of split operations that, starting from Γ_2, constructs a vertex-edge-consistent 01-drawing of a subdivision of G_2.*

Proof. Consider one by one each original vertex u of G_0 such that there exists a dummy vertex v with the same coordinates of u. Let (v', v) and (v, v'') be the incident edges of v. By Property 1, v, v' and v'' have different coordinates.

Let d' and d'' be the directions of v used by (v', v) and (v, v''), respectively. We perform $trivialsplit(d', P_{d',v})$ and $trivialsplit(d'', P_{d'',v})$. After such operations vertex v is guaranteed to be adjacent to dummy vertices w' and w'' created by the performed splits. Two cases are possible: either $d' = -d''$ or not. In the first case we define d''' as any direction orthogonal to d'; in the second case we define d''' as any direction among d', d'', and the two directions orthogonal to d' and d''. We perform $split(d''', P_{d''',v}, \phi, \rho)$ as follows. We set $\phi(v) = true$, all the other vertices of $P_{d''',v}$ have $\phi = false$. All the edges in the domain of ρ have $\rho = true$. Note that now u and v have different coordinates, that each split preserves the properties of direction-consistent 01-drawings, and that each operation does not generate new vertex-edge overlaps.

Two distinct planar paths p_1 and p_2 on the same plane *intersect* if there exist two vertices one of p_1 and the other of p_2 with the same coordinates.

Theorem 3. *Let Γ_3 be a vertex-edge-consistent 01-drawing of a graph G_3, subdivision of a graph G_0. There exists a finite sequence of split operations that, starting from Γ_3, constructs a 1-drawing of a subdivision of G_3.*

Proof. Since Γ_3 is vertex-edge-consistent, all vertices have distinct coordinates but dummy vertices, that may still overlap. We consider the number χ of the pairs of intersecting planar dummy paths of Γ_3. Fig. 4 shows that χ can be greater than one even with just two vertices with the same coordinates. If $\chi = 0$, then Γ_3 is already a 1-drawing of G_3. Otherwise, we repeatedly select a pair of intersecting planar dummy paths p_1 and p_2 and "remove" their intersection, decreasing the value of χ. Such removal is performed as follows.

Let u_1 and v_1 be the endpoints of p_1. If u_1 (v_1) is an original vertex we perform $trivialsplit(d_1, P_{d_1,u_1})$ ($trivialsplit(d_1, P_{d_1,u_1})$), where d_1 is the direction determined by entering p_1 from u_1 (v_1). The value of χ stays unchanged. We denote by r_1 (s_1) the dummy vertex possibly introduced by the $trivialsplit$.

Let x_1 and x_2 be two vertices, one of p_1 and the other of p_2, with the same coordinates; x_1 and x_2 are dummy. Let d a free direction of x_2 such that $-d$ is a free direction of x_1. Since x_1 and x_2 have both degree 2, direction d always exists. We perform $split(d, P_{d,x_1}, \phi, \rho)$, by setting $\phi(v) = true$ for each $v \in p_1$ and $v \neq r_1, s_1$ ($false$ otherwise). All the edges in the domain of ρ have $\rho = true$. The proof is easily completed by showing the decrease of χ after the split.

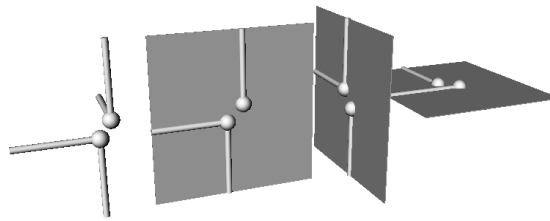


Fig. 4. Two dummy vertices with the same coordinates originating 3 pairs of intersecting planar dummy paths.

We have shown that *split* is a powerful tool in performing the steps of the method presented in Section 2. Namely, each step of Vertex scattering, Direction distribution, Vertex-edge overlap removal, and Crossing removal can be performed by a finite sequence of splits.

4 The Reduce-Forks Algorithm

An edge (u, v) is *cut* by $split(d, P, \phi, \rho)$ if u and v were on the same P-plane before the split and are on different planes parallel to P after the split. A pair of adjacent edges cut by a split is a *fork*.

Algorithm `Reduce-Forks` follows the strategy described in Sections 2 and 3. However, the steps of the approach are refined into a set of heuristics that can be summarized as follows. **Vertex Scattering:** We repeatedly apply the following strategy. We select two original vertices u and v of G_0 with the same coordinates. We consider the set of split operations that separate u from v and perform the one with "a few" forks. The number of forks is kept small since, intuitively, each fork will require at least one bend to be removed in the subsequent Direction distribution step. **Direction Distribution:** For each original vertex u of G_0 with edges (u, v) and (u, w) such that v and w have the same coordinates: (1) We compute all the planar dummy paths containing (u, v) and (u, w). (2) We determine all the split operations that separate such paths and that separate v from w. (3) We weight such splits according to the number of bends they introduce and to the number n_d of vertices that become direction-consistent after the split. We have that $1 \leq n_d \leq 2$. (4) We select and apply the split with minimum weight. **Vertex-Edge Overlap Removal:** For each original vertex u of G_0 such that v has the same coordinates as u: (1) We compute all the planar dummy paths containing v. (2) We determine all the split operations that separate such paths from u. (3) We weight such splits according to the number of bends they introduce and to the number of crossings introduced by the split. (4) We select and apply the split with minimum weight. **Crossing Removal:** For each pair of dummy vertices u and v having the same coordinates: (1) We

compute all the planar dummy paths containing u or v. (2) We determine all the split operations that separate such paths and u from v. (3) We weight such splits according to the number of bends they introduce. (4) We select and apply the split with minimum weight.

Concerning the Vertex scattering step, observe that a split with no forks is a matching cut. Unfortunately, the problem of finding a matching cut in a graph is NP-complete (see [23]), hence a heuristic solution is needed. A simple and efficient heuristic for finding a split with a few forks is described below.

Let G be a graph with adjacent vertices u and v. We color *black* and *red* the vertices of G in the two sides of the split. Each step of the heuristic colors one vertex. At a certain step a vertex can be black, red or *free* (uncolored). At the beginning u is black, v is red, and all the other vertices are free.

Colored vertices adjacent to free vertices are *active vertices*. Black (Red) vertices adjacent to red (black) vertices are *boundary vertices*. See Fig. 5. Each step works as follows. (1) If a boundary active black (red) vertex, say x, exists, then we color black (red) one free vertex y adjacent to x. This is done to prevent a fork between (x, y) and (x, w), where w is a red (black) vertex. (2) Else, if an active black (red) vertex, say x, exists, then we color black (red) one free vertex y adjacent to x. This is done for not cutting edge (x, y). (3) Else, we color black or red (random) a (random) free vertex.

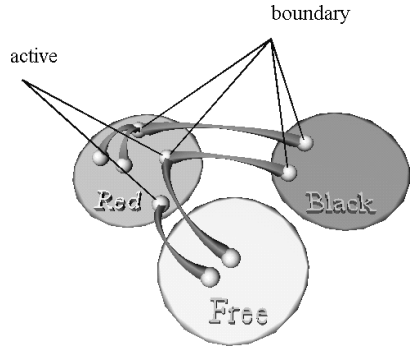


Fig. 5. Red, black, and free vertices in the Vertex scattering heuristic of Algorithm `Reduce-Forks`.

It is easy to implement the above heuristic to run in linear time and space (a graph with at most six edges per vertex has a linear number of edges).

5 Experimental Results

The experiments have been performed on a Sun Sparc station Ultra-1 by using `3DCube` [24]. All the algorithms have been implemented in C++. The test suite

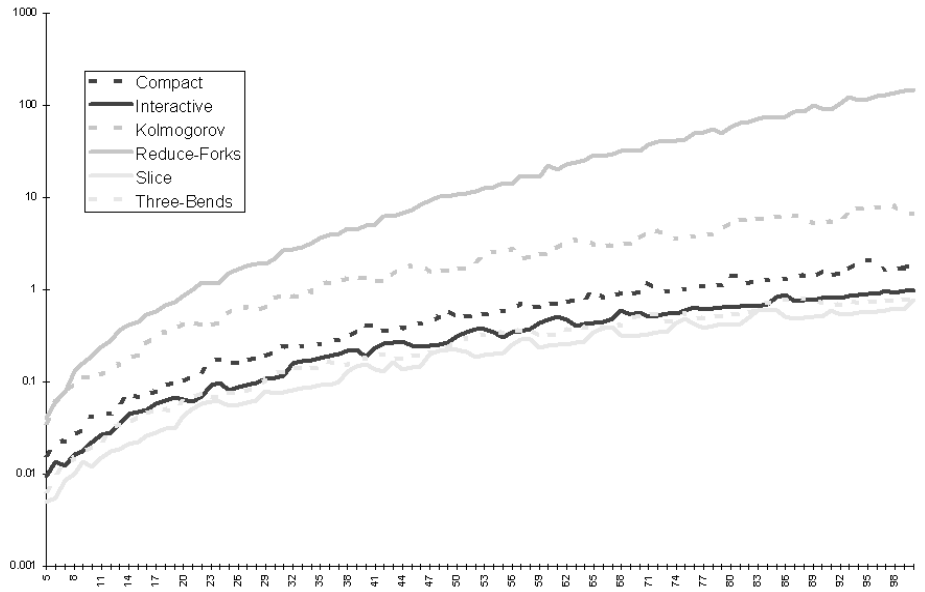
was composed by 1900 randomly generated graphs having from 5 to 100 vertices, 20 graphs for each value of vertex cardinality. All graphs were connected, with maximum vertex degree 6, without multi-edges and self-loops. The density was relatively high: the number of edges was twice the number of vertices.

We considered two families of quality measures. For the efficiency we relied on the time performance (cpu seconds); for the readability we measured the average number of bends along the edges, the average edge length, and the volume of the minimum enclosing box with sides isothetic to the axes.

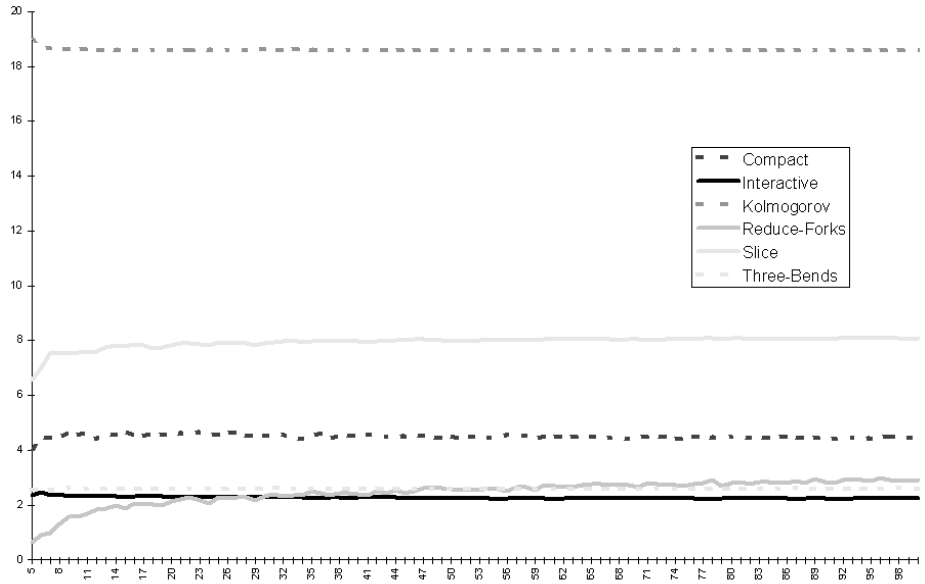
We compared algorithms `Compact`, `Interactive`, `Kolmogorov`, `Reduce-Forks`, `Slice`, and `Three-Bends`. Fig. 6 and 7 illustrate the results of the comparison.

The comparison shows that no algorithm can be considered "the best". Namely, some algorithms are more effective in the average number of bends (`Interactive`, `Reduce-Forks`, and `Three-Bends`) while other algorithms perform better with respect to the volume (`Compact` and `Slice`) or to the edge length (`Compact`, `Interactive`, and `Reduce-Forks`). In particular: (i) The average number of bends (see Fig. 6-b) is comparable for `Interactive`, `Reduce-Forks`, and `Three-Bends`, since it remains for all of them under the value of 3 bends per edge, while it is higher for `Compact` and `Slice`, and it definitely too much high for `Kolmogorov`. Furthermore, `Reduce-Forks` performs better than the others for graphs with number of vertices in the range 5–30. `Interactive` performs better in the range 30–100. Another issue concerns the results of the experiments vs. the theoretical analysis. About `Kolmogorov` the literature shows an upper bound of 16 bends per edge [11] while our experiments obtain about 19. This inconsistency might show a little "flaw" in the theoretical analysis (or, of course, in our implementation). Further, about `Compact` the experiments show that the average case is much better than the worst case [12]. (ii) Concerning the average edge length (see Fig. 7-a), `Reduce-Forks` performs better for graphs up to 50 vertices, whereas `Compact` is better from 50 to 100; `Interactive` is a bit worse, while the other algorithms form a separate group with a much lower level of performance. (iii) The values of volume occupation (see Fig. 7-b) show that `Compact` and `Slice` have the best performance for graphs bigger than 30 vertices, while `Reduce-Forks` performs better for smaller graphs. Examples of the drawings constructed by the algorithms are shown in Fig. 8.

As overall considerations we can say that `Reduce-Forks` is the most effective algorithm for graphs in the range 5–30. Also, among the algorithms that have a reasonable number of bends along the edges (`Interactive`, `Reduce-Forks`, and `Three-Bends`), `Reduce-Forks` is the one that has the best behaviour in terms of edge length and volume. This is obtained at the expenses of an efficiency that is much worse than the other algorithms. However, the CPU time do not seem to be a critical issue for the size of graphs in the interval. In fact, even for `Reduce-Forks` the CPU time never overcomes the sole of 150 seconds, that is still a reasonable time for most of the applications.

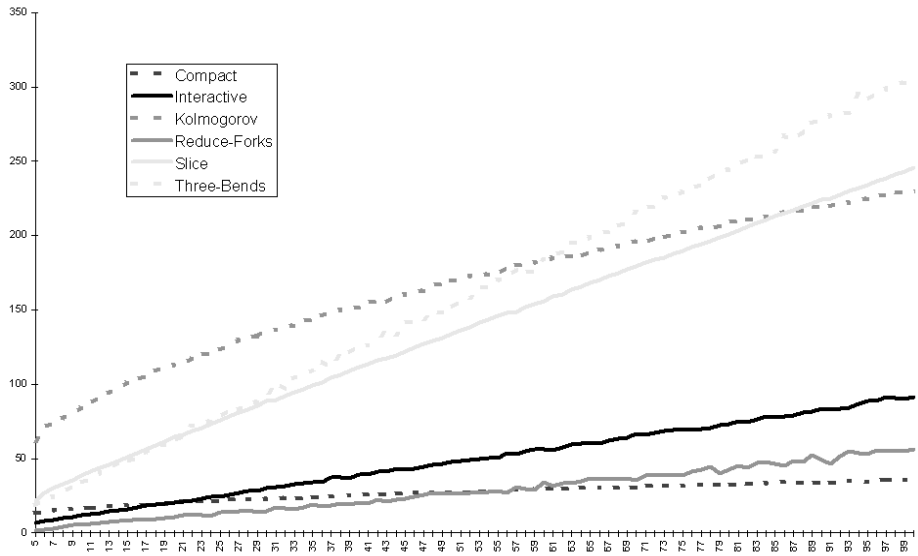


(a)

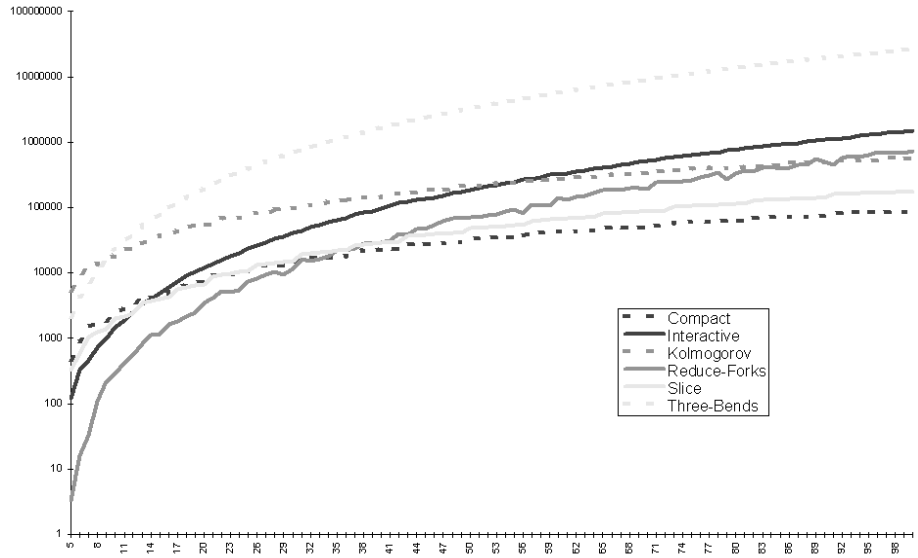


(b)

Fig. 6. Comparison of Algorithms `Compact`, `Interactive`, `Kolmogorov`, `Reduce-Forks`, `Slice`, and `Three-Bends` with respect to time performance (a) and average number of bends along edges (b).



(a)



(b)

Fig. 7. Comparison of Algorithms `Compact`, `Interactive`, `Kolmogorov`, `Reduce-Forks`, `Slice`, and `Three-Bends` with respect to average edge length (a) and volume occupation (b).

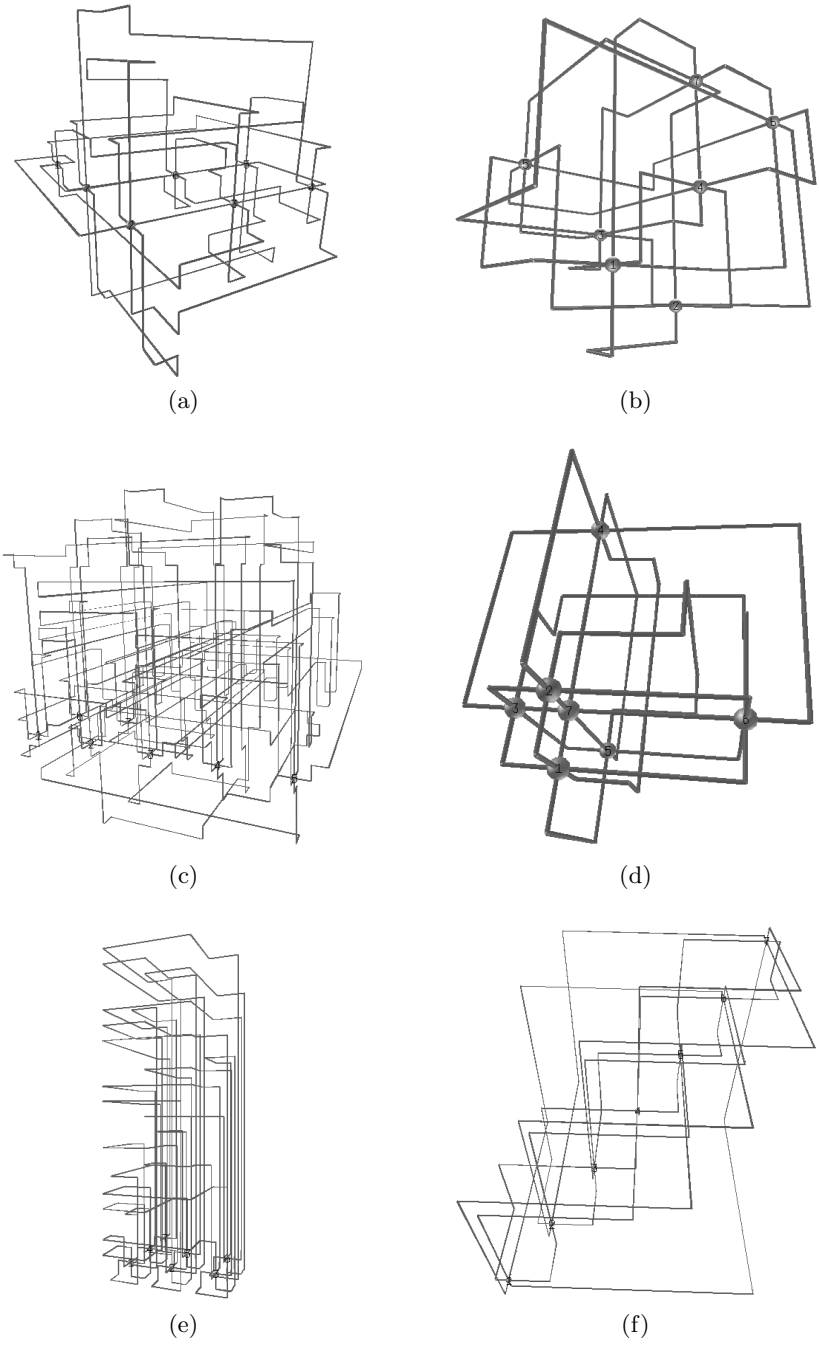


Fig. 8. Three-dimensional orthogonal drawings of a K_7 as yielded by `Compact` (a), `Interactive` (b), `Kolmogorov` (c), `Reduce-Forks` (d), `Slice` (e), and `Three-Bends` (f).

6 Conclusions and Open Problems

In this paper we presented a new method for constructing orthogonal drawings in three dimensions of graphs of maximum degree 6, and tested the effectiveness of our approach by performing an experimental comparison with several existing algorithms. The presented techniques are easily extensible to obtain drawings of graphs of arbitrary degree with the following strategy. The Vertex scattering step remains unchanged. In the Direction distribution step for vertices of degree greater than six, we first saturate the six available directions and then we evenly distribute the remaining edges. The Vertex-edge overlap removal step remains unchanged. In the Crossing removal step we distinguish between crossings that are "needed" because of the overlay between edges that is unavoidable because of the high degree and the crossings that can be removed. For the latter type of crossings we apply the techniques presented in Section 3.

Several problems are opened by this work. (1) Devise new algorithms and heuristics (alternative to `Reduce-Forks`) within the described paradigm. (2) Explore the trade-off, put in evidence by the experiments, between number of bends and volume. (3) Measure the impact of bend-stretching (or possibly other post-processing techniques) on the performances of the different algorithms. (4) Devise new quality parameters to better study the human perception of "nice drawing" in three dimensions.

References

[1] H. Alt, M. Godau, and S. Whitesides. Universal 3-dimensional visibility representations for graphs, in [4], pp. 8–19.

[2] T. C. Biedl. Heuristics for *3d*-orthogonal graph drawings. In *Proc. 4th Twente Workshop on Graphs and Combinatorial Optimization*, pp. 41–44, 1995.

[3] P. Bose, H. Everett, S. P. Fekete, A. Lubiw, H. Meijer, K. Romanik, T. Shermer, and S. Whitesides. On a visibility representation for graphs in three dimensions. In D. Avis and P. Bose, eds., *Snapshots in Computational and Discrete Geometry, Vol. III*, pp. 2–25. McGill Univ., July 1994. McGill tech. rep. SOCS-94.50.

[4] F. J. Brandenburg, editor: Proceedings of Graph Drawing '95, Vol. 1027 of *LNCS*, Springer-Verlag, 1996.

[5] I. Bruß and A. Frick. Fast interactive 3-D graph visualization, in [4], pp. 99–110.

[6] T. Calamoneri and A. Sterbini. Drawing 2-, 3-, and 4-colorable graphs in $O(n^2)$ volume, in [19], pp. 53–62.

[7] R. F. Cohen, P. Eades, T. Lin, and F. Ruskey. Three-dimensional graph drawing. *Algorithmica*, 17(2):199–208, 1996.

[8] I. F. Cruz and J. P. Twarog. 3d graph drawing with simulated annealing, in [4], pp. 162–165.

[9] G. Di Battista, editor: Proceedings of Graph Drawing '97, Vol. 1353 of *LNCS*, Springer-Verlag, 1998.

[10] D. Dodson. COMAIDE: Information visualization using cooperative 3D diagram layout, in [4], pp. 190–201.

[11] P. Eades, C. Stirk, and S. Whitesides. The techniques of Kolmogorov and Bardzin for three dimensional orthogonal graph drawings. *I. P. L.*, 60(2):97–103, 1987.

[12] P. Eades, A. Symvonis, and S. Whitesides. Two algorithms for three dimensional orthogonal graph drawing, in [19], pp. 139–154.

[13] S. P. Fekete, M. E. Houle, and S. Whitesides. New results on a visibility representation of graphs in 3-d, in [4], pp. 234–241.

[14] A. Frick, C. Keskin, and V. Vogelmann. Integration of declarative approaches, in [19], pp. 184–192.

[15] A. Garg, R. Tamassia, and P. Vocca. Drawing with colors. In *Proc. 4th Annu. Europ. Sympos. Algorithms*, vol. 1136 of *LNCS*, pp. 12–26. Springer-Verlag, 1996.

[16] A. N. Kolmogorov and Y. M. Bardzin. About realization of sets in 3-dimensional space. *Problems in Cybernetics*, pp. 261–268, 1967.

[17] G. Liotta and G. Di Battista. Computing proximity drawings of trees in the 3-dimensional space. In *Proc. 4th Workshop Algorithms Data Struct.*, volume 955 of *LNCS*, pp. 239–250. Springer-Verlag, 1995.

[18] B. Monien, F. Ramme, and H. Salmen. A parallel simulated annealing algorithm for generating 3D layouts of undirected graphs, in [4], pp. 396–408.

[19] S. North, editor: Proceedings of Graph Drawing '96, Vol. 1190 of *LNCS*, Springer-Verlag, 1997.

[20] D. I. Ostry. *Some Three-Dimensional Graph Drawing Algorithms.* M.Sc. thesis, Dept. Comput. Sci. and Soft. Eng., Univ. Newcastle, Oct. 1996.

[21] J. Pach, T. Thiele, and G. Tóth. Three-dimensional grid drawings of graphs, in [9], pp. 47–51.

[22] A. Papakostas and I. G. Tollis. Incremental orthogonal graph drawing in three dimensions, in [9], pp. 52–63.

[23] M. Patrignani and M. Pizzonia The complexity of the matching-cut problem. Tech. Rep. RT-DIA-35-1998, Dept. of Computer Sci., Univ. di Roma Tre, 1998.

[24] M. Patrignani and F. Vargiu. 3DCube: A tool for three dimensional graph drawing, in [9], pp. 284–290.

[25] R. Webber and A. Scott. GOVE: Grammar-Oriented Visualisation Environment, in [4], pp. 516–519.

[26] D. R. Wood. Two-bend three-dimensional orthogonal grid drawing of maximum degree five graphs. Technical report, School of Computer Science and Software Engineering, Monash University, 1998.