Implementing a Partitioned 2-Page Book Embedding Testing Algorithm*

Patrizio Angelini[1], Marco Di Bartolomeo[1,2], and Giuseppe Di Battista[1]

[1] Dip. di Informatica e Automazione, Roma Tre University, Italy
[2] Italian Inter-University Computing Consortium CASPUR, Italy
{angelini,gdb}@dia.uniroma3.it, m.dibartolomeo@caspur.it

Abstract. In a book embedding the vertices of a graph are placed on the "spine" of a "book" and the edges are assigned to "pages" so that edges on the same page do not cross. In the PARTITIONED 2-PAGE BOOK EMBEDDING problem egdes are partitioned into two sets E_1 and E_2, the pages are two, the edges of E_1 are assigned to page 1, and the edges of E_2 are assigned to page 2. The problem consists of checking if an ordering of the vertices exists along the spine so that the edges of each page do not cross. Hong and Nagamochi [13] give an interesting and complex linear time algorithm for tackling PARTITIONED 2-PAGE BOOK EMBEDDING based on SPQR-trees. We show an efficient implementation of this algorithm and show its effectiveness by performing a number of experimental tests. Because of the relationships [13] between PARTITIONED 2-PAGE BOOK EMBEDDING and clustered planarity we yield as a side effect an implementation of a clustered planarity testing where the graph has exactly two clusters.

1 Introduction

In a *book embedding* [14] of a graph the vertices are placed on the "spine" of a "book" and the edges are assigned to "pages" so that edges on the same page do not cross. A rich body of literature witnesses the interest of the scientific community for book embeddings. See, e.g., [4,16].

Several constrained variations of book embeddings have been studied. In [15] the problem is tackled when in each page the number of edges incident to a vertex is bounded. In [11] the graph is directed upward planar and the order of the vertices on the spine must be consistent with the orientation of the edges. Hong and Nagamochi [13] provide a linear time algorithm for a problem called PARTITIONED 2-PAGE BOOK EMBEDDING (P2BE). In the P2BE problem the egdes of an input graph $G(V, E_1, E_2)$ are partitioned into two sets E_1 and E_2, the pages are just two, the edges of E_1 are assigned to page 1, and the edges of E_2 are assigned to page 2. The problem consists of checking if an ordering of the vertices exists along the spine so that the edges of each page do not cross.

In [13] the P2BE problem is characterized in terms of the existence of an embedding of G allowing to build a variation of the dual graph containing a particular Eulerian tour.

* This work was partially supported by the ESF project 10-EuroGIGA-OP-003 GraDR "Graph Drawings and Representations", by the MIUR of Italy, under project AlgoDEEP, prot. 2008TFBWL4, and by the italian inter-university computing Consortium CASPUR.

W. Didimo and M. Patrignani (Eds.): GD 2012, LNCS 7704, pp. 79–89, 2013.

The existence of such an embedding is tested exploiting SPQR-trees [9] for biconnected components and BC-trees for connected ones.

In this paper we discuss an implementation of the algorithm in [13]. To efficiently implement the algorithm we faced the following problems: (i) One of the key steps of the algorithm requires the enumeration and the analysis of all the permutations of a set of objects. Even if the cardinality of the set is bounded by a constant this may lead to very long execution times. We restated that step of the algorithm avoiding such enumerations. (ii) Some steps of the algorithm are described in [13] at a high abstraction level. We found how to efficiently implement all of them. (iii) The algorithm builds several embeddings that are tested for the required properties only at the end of the computation. Our implementation considers only one embedding that is greedily built to have the properties. We performed experiments over a large set of suitably randomized graphs. The experiments show quite reasonable linear execution times.

The algorithm in [13] is interesting in itself, as book embedding problems are ubiquitous in Graph Drawing. However, it is even more appealing since it yields [13] almost immediately a linear time algorithm for the following special case of *clustered planarity* (*c-planarity*) testing. A planar graph $G(V_1, V_2, E)$ whose vertices are partitioned into two sets (clusters) V_1 and V_2 is given. Is it possible to find a planar drawing of G such that: (i) each of V_1 and V_2 is drawn inside a simple region, (ii) the two regions are disjoint, and (iii) each edge of E crosses the boundary of a region at most once? In the terminology of Clustered Planarity, this is a c-planarity testing for a flat clustered graph with exactly two clusters. References on c-planarity can be found, e.g., in [10,6]. Hence, as a side effect, we have an implementation of this problem. An alternative algorithm for the same c-planarity problem has been proposed in [3,2].

The paper is organized as follows. In Section 2 we give preliminaries. In Section 3 we outline the algorithm. Section 4 discusses how to search an embedding with the desired features and Section 5 gives further implementation details on the search. In Section 6 we describe our experiments. Section 7 gives concluding remarks. An extended version of the paper appears in [1].

2 Preliminaries

A *planar drawing* of a graph is a mapping of each vertex to a distinct point of the plane and of each edge to a simple Jordan curve connecting its endpoints such that the curves representing the edges do not cross but, possibly, at common endpoints. A graph is *planar* if it admits a planar drawing. Two drawings of a graph are *equivalent* if they determine the same circular ordering around each vertex. An *embedding* is an equivalence class of drawings. A planar drawing partitions the plane into topologically connected regions, called *faces*. The unbounded face is the *outer face*.

A graph is *connected* if every two vertices are joined by a path. A graph G is *biconnected* (*triconnected*) if removing any vertex (any two vertices) leaves G connected. To handle the decomposition of a biconnected graph into its triconnected components, we use *SPQR-trees* (see [8,9,12]). Here we give some notation. Given a biconnected graph G and its SPQR-tree $\mathcal{T}$, the *skeleton* skel(μ) of a node $\mu \in \mathcal{T}$ is a graph representing the arrangement of the triconnected components of μ. Edges of skel(μ) are *virtual edges*.

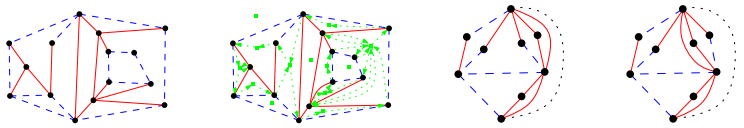

Fig. 1. (a) A disjunctive and splitter-free embedding of a graph. (b) The corresponding green graph. (c) An r-rimmed embedding of a graph G. (d) An embedding of G that is not r-rimmed.

For each virtual edge e_i of skel(μ) corresponding to a child μ_i, recursively replace e_i with skel(μ_i). The resulting subgraph of G is the *pertinent graph* pert(μ) of μ. Graph skel(μ) contains a *parent* virtual edge between the *poles* representing the rest of the graph.

Let G be a directed planar embedded graph. A directed cycle of G is a *Eulerian tour* if it traverses each edge exactly once. Consider a vertex v of G and let (v_1, v), (v, v_2), (v, v_3), and (v_4, v) be four edges incident to v appearing in this order around v in the given embedding. If a Eulerian tour contains egdes (v_1, v), (v, v_3), (v_4, v), and (v, v_2) in this order then it is *self-intersecting*.

3 A Partitioned 2-Page Book-Embedding Testing Algorithm

In this section we describe an algorithm that, given an instance of P2BE, decides whether it is positive and, in case it is, constructs a book embedding of the input graph such that each edge is drawn on the page it is assigned to. The algorithm is the one proposed in [13]. However, substantial modifications have been applied to implement it. Part of them aim at simplifying the algorithm, while others at decreasing the value of some constant factors spoiling the efficiency. Further, some steps that are described at high level in [13] are here detailed. The main differences with [13] are highlighted throughout the paper.

Let $G(V, E_1, E_2)$ be an instance of problem P2BE. We say that the edges of E_1 (of E_2) are *red* (*blue*) edges. As pointed out in [13], G admits a P2BE if and only if all the biconnected components of G admit a solution. Hence, we limit the description to the case in which G is biconnected. Moreover, we assume that both E_1 and E_2 are not empty, since a graph with only red (blue) edges is a positive instance if and only if it is outerplanar, which is testable in linear time.

The algorithm is based on a characterization proved in [13] stating that an instance admits a solution if and only if G admits a *disjunctive* and *splitter-free* planar embedding (see Fig. 1(a)). An embedding is *disjunctive* if for each vertex $v \in V$ all the red (blue) edges incident to v appear consecutively around v. A *splitter* is a cycle C composed of red (blue) edges such that both the open regions of the plane determined by C contain either a vertex or a blue (red) edge. An embedding is *splitter-free* if it has no splitter. The first part of the algorithm, that is based on the SPQR-tree decomposition of G and whose details are in Sections 4 and 5, concerns the construction of an embedding of G satisfying these requirements, if it exists. Otherwise, G does not admit any solution.

Once a disjunctive and splitter-free embedding Γ of G has been computed, an auxiliary graph G^*, called *green graph*, is constructed starting from Γ. Then, as proved in [13], a non-self-intersecting Eulerian tour on G^* gives the ordering in which placing the vertices of V on the spine of a P2BE of G.

Graph G^* is a directed graph whose vertices are the vertices of V plus a vertex for each face of Γ. See Fig. 1(b). For each vertex v of G incident to at least one red edge and one blue edge, consider each face f incident to v such that v is between a red edge e_1 and a blue edge e_2 on f. If e_1 immediately precedes e_2 in the clockwise ordering of the edges around v, then add to G^* an oriented edge (v, f), otherwise add an oriented edge (f, v). For each vertex w of G^* incident only to red (blue) edges, consider a face f' incident to w that contains at least one blue (red) edge. Since Γ is splitter-free, such face exists. Then, add directed edges (w, f') and (f', w). Note that, by construction, G^* is a bipartite plane digraph, every vertex v of V has degree 2 in G^*, namely v is incident to exactly one entering and one exiting edge, and each vertex f corresponding to a face of Γ has even degree, namely the number of edges entering f equals the number of edges exiting f, and such edges alternate around f. From this and from the fact that the underlying graph of G^* is connected, as pointed out in [13], it follows that G^* contains a Eulerian tour. The alternation of entering and exiting edges around each vertex ensures the existence of a non-self-intersecting Eulerian tour. The algorithm we implemented to find such a tour is based on this fact.

From the above discussion it follows the claimed statement that the described algorithm computes a P2BE of (V, E_1, E_2), if any such a P2BE exists.

4 Computing a Disjunctive and Splitter-Free Embedding

Let $G(V, E_1, E_2)$ be a biconnected planar graph. We describe an algorithm to compute a disjunctive and splitter-free embedding of G, if any such an embedding exists, consisting of two preprocessing traversals of the SPQR-tree $\mathcal{T}$ of G and of a final bottom-up traversal to compute the required embedding.

Let μ be a node of $\mathcal{T}$. According to [13], a virtual edge e of skel(μ) is an *r-edge* (a *b-edge*) if there exists a path in pert(μ) between the poles of μ composed of red edges (of blue edges). If e is both an r-edge and a b-edge, it is a *br-edge*.

Consider a cycle $C = e_1, \ldots, e_q$ in skel(μ) composed of edges of the same color, say r-edges. If C is a splitter in every embedding of skel(μ), then a splitter is unavoidable. However, even if there exists an embedding of skel(μ) such that C is not a splitter, then a cycle in pert(μ) passing through the pertinent graphs of $e_1, \ldots, e_q$ could still be a splitter (since $e_1, \ldots, e_q$ are r-edges, there exists at least one red cycle C' in pert(μ)). Consider any node ν corresponding to a virtual edge e_i and the path $p_\nu(C')$ between the poles of ν that is part of C'. Intuitively, in order for C' not to be a splitter, we should construct an embedding of pert(ν) in which $p_\nu(C')$ is on the outer face. Actually, not all the vertices of $p_\nu(C')$ have to be on the outer face, since red chords might exist in $p_\nu(C')$ (that is, red edges connecting vertices not consecutive in $p_\nu(C')$), separating some vertex of $p_\nu(C')$ from the outer face, as in this case such chords would be internal to C', and this does not make it a splitter. In analogy with [13], where the same concept was described with a slightly different definition, we say that an embedding of pert(ν) in which each

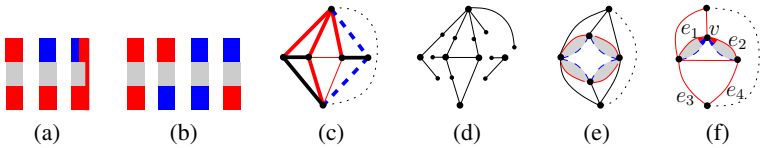

Fig. 2. Parallel virtual edges are sketched with rectangles colored according to their poles. (a) An r-rimmed embedding forces an RBR color-pattern on a pole. (b) A color-pattern BR or RB on a pole forces either an RBR or a BRB on the other pole. (c) An R-node. Virtual edges representing Q-nodes are thin. (d) The corresponding auxiliary graph O_1. (e) A splitter that is not a rigid-splitter. (f) Disjunctiveness constraints on nodes e_1 and e_2 determine a splitter (e_1, e_2, e_3, e_4).

path between the poles composed of red edges (of blue edges) has only red edges (blue edges) on one of its sides is *r-rimmed* (is *b-rimmed*). Figs. 1(c) and (d) show an r-rimmed and a non-r-rimmed embedding, respectively. Note that an embedding could be at the same time both r- and b-rimmed, with the red and the blue paths on different sides of the outer face.

The existence of an r-rimmed (b-rimmed) embedding is necessary only for each node μ such that there exists a cycle C of red (blue) edges traversing both μ and its parent. However, the existence of C is not known when processing μ during a bottom-up visit of $\mathcal{T}$. Thus, we perform a preprocessing phase to decide for each node μ whether any such cycle C exists. In this case, μ is *r-joined* (*b-joined*). Hence, when processing μ, we know whether it is r-joined (b-joined) and, in case, we inductively compute an r-rimmed (b-rimmed) embedding.

Concerning disjunctiveness, for each vertex w of skel(μ) we have to check whether the ordering of the edges around w determined by the embedded pertinent graphs of the child nodes incident to w makes it disjunctive. In order to classify the possible orderings of edges around the poles of a node we define, in analogy with [13], the *color-pattern* of a node μ on a vertex v as the sequence of colors of the edges of pert(μ) incident to v. Namely, the color-pattern of μ on v is one of R, B, RB, BR, RBR, BRB. Note that, if the color-pattern is either R or B, then it is the same in any embedding. Otherwise, it depends on the chosen embedding. Hence, it might be influenced by the fact that the embedding needs to be r- or b-rimmed (see Fig 2(a)) and by the need of a particular color-pattern on the other pole (see Fig 2(b)). Thus, a color-pattern either RBR or BRB could be forced on a pole u of μ although an RB or a BR pattern would be possible as well. Another factor influencing the color-pattern on u is the presence of red or blue edges incident to u in the pertinent of the parent ν of μ. In fact, if u has color-pattern RBR (BRB) and there is a blue (red) edge in pert(ν) incident to u, then u is not disjunctive. Thus, in the preprocessing phase we also determine two flags for each pole u of μ, stating whether ν contains at least one red (blue) edge incident to u. Hence, when processing μ, we know whether it is admissible to have an RBR (a BRB) color-pattern on its poles.

The two information obtained in the preprocessing can be properly combined when processing a node to decide whether an embedding satisfying all the constraints exists, as described is Section 5. If it is not the case, we state that the instance is negative, while in the case that at least one of such embeddings exists, we can arbitrarily choose one

of them, without the need of carrying on a multiplicity of embeddings. This is one of the most crucial differences between our implementation and [13]. In fact, even if they perform a preprocessing to determine whether a node is r-joined (b-joined), they do not exploit it for disjunctiveness, and have to consider at each step all the possible embeddings determining different color-patterns on the two poles. Of course, as the number of color-patterns is bounded by a constant, this does not affect the asymptotic complexity, but our solution noticeably improves on the execution times. Also, they deal with constraints given by the r-joinedness (b-joinedness) and by the disjunctiveness in two different steps. In our case, instead, instances that are negative due to disjunctiveness are recognized much earlier. The preprocessing consists of a bottom-up and a top-down traversal of $\mathcal{T}$.

5 SPQR-Tree Algorithm

When considering a node μ of $\mathcal{T}$ with children $\nu_1, \ldots, \nu_k$, exploiting the information resulting from the preprocessing and the information inductively computed for $\nu_1, \ldots, \nu_k$, we check whether μ admits a splitter-free and disjunctive embedding and compute the following: (i) if μ is r-joined (b-joined), an r-rimmed (a b-rimmed) embedding; and (ii) the color-patterns of the poles of μ.

In the base case, μ is a **Q-node**. Suppose that skel(μ) is an r-edge, the other case being analogous. If μ is r-joined, every embedding of skel(μ) is r-rimmed. Further, the color-pattern on the poles is R in any embedding of skel(μ).

Suppose that μ is an **R-node**. Since skel(μ) is triconnected, it has one planar embedding, up to a flip. Hence, if there is a splitter in skel(μ), then it is unavoidable. Hong and Nagamochi call such splitters *rigid-splitters*. In order to test the existence of such splitters, for each set E_i, $i = 1, 2$, we construct an auxiliary graph O_i starting from skel(μ). See Figs. 2(c) and (d). We describe the construction for E_1, the other case being analogous. Initialize O_1 =skel(μ). Subdivide each virtual edge of skel(μ) (including the one representing the parent) with a dummy vertex, except for the r-edges corresponding to Q-nodes. Then, for each dummy vertex subdividing a virtual edge that is not an r-edge, remove one of its incident edges without modifying the embedding. Finally, check whether the obtained embedding of O_1 is an outerplane embedding, that is, all the vertices of O_1 are on the same face. In [13] this step is performed by constructing a variant of the green graph and checking whether it is connected. We find that our approach is easier to implement and slightly more efficient, since O_1 does not need to be constructed, but can be obtained by flagging the edges of skel(μ).

Note that, for each cycle composed of r-edges (b-edges) in skel(μ) that is not a rigid-splitter, all the nodes composing it inductively admit an r-rimmed (b-rimmed) embedding. Hence, it suffices to flip them in such a way that their red (blue) border is turned towards the red (the blue) outerplanar face. However, if each of them has an embedding that is both r-rimmed and b-rimmed, the red and the blue outerplanar faces coincide and it is not possible to flip the nodes properly, which implies that a splitter exists in the embedding. See Fig. 2(e). This type of splitter seems to have gone unnoticed in [13], where flips imposed by cycles of r- and b-edges are considered independently.

We deal with disjunctiveness constraints. We observe some straightforward properties of the color-patterns of the nodes incident to the same vertex w of skel(μ).

(i) At most two nodes have color-pattern different from R and B. (ii) If one node has color-pattern RBR (BRB), then all the other nodes have color-pattern R (B). Hence, since each vertex has degree at least 3 in skel(μ), at least one node ν incident to w exists with color-pattern either R or B. Thus, starting from ν, we consider all the nodes incident to w in clockwise order and greedily decide a flip based on the current color. If more than two changes of color are performed, then G does not admit any disjunctive embedding. If exactly one node ν has color-pattern different from R or B and all the other nodes have color-pattern R (B), then the flip of ν is not decided at this step. Also, the flip is not decided for the nodes having color-pattern R or B.

Disjunctiveness and splitter-free constraints might be in contrast. See Fig. 2(f). We can efficiently determine such contrasts by flagging the nodes that need to be flipped and, in case such contrasts exist, state that the instance is negative. This check is not described in [13], where possible contrasts between disjunctive and splitter-free constraints are noticed for P-nodes but not for R-nodes.

The color-patterns of the poles and, if needed, an r-rimmed (a b-rimmed) embedding of pert(μ) are computed by considering the information on the parent node, the color-patterns of the virtual edges incident to the poles, and the r-rimmed (b-rimmed) embedding of the children.

Suppose that μ is an **S-node**. Since skel(μ) is a cycle containing all the virtual edges, even if such a cycle is composed of edges of the same color, then it is not a splitter. Namely, even if there exist both a red and a blue cycle passing through all the children of μ, such nodes can be flipped so that the red and the blue borders are turned towards the two faces of skel(μ).

Concerning disjunctiveness constraints, if two children both incide on a vertex u of skel(μ) with color-pattern either BR or RB, then they have to be flipped in such a way that the red edges (and hence the blue edges) are consecutive around u. In all the other cases, the relative flip of the two children incident to u is not fixed by their color-patterns. If there exists at least a vertex u with this property, we say that μ admits two different *semi-flips*. Intuitively, this means that the color-pattern of a pole is independent of the one on the other pole, since they depend on flips performed on two different subsets of children of μ.

Note that in an S-node no contrast between splitter-free and disjunctiveness constraints are possible, since flipping the r-rimmed embeddings towards the same face implies placing the red edges consecutive around u. Hence, no negative answer can be given during the processing of an S-node.

The color-pattern on each pole is the color-pattern of the unique node incident to it, while an r-rimmed (b-rimmed) embedding is obtained by concatenating the r-rimmed (b-rimmed) embeddings of the children.

Suppose that μ is a **P-node**. In order for a splitter-free embedding to exist, the following must hold: (i) There exist at most 3 r-edges (b-edges); if they are 3 then one is a Q-node. (ii) There exist at most 2 virtual edges that are both r-edges and b-edges; if they are 2 then there exists only another virtual edge and it is a Q-node.

On the other hand, in order for a disjunctive embedding to exist, the following must hold: (i) if there exists a virtual edge with RBR (BRB) color-pattern on a pole, then

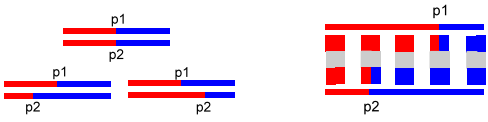

Fig. 3. (a) All possible alignments of a pair of RB color-patterns. (b) Correspondence between an alignment of a pair of color-patterns and a sequence of virtual edges of a P-node.

all the other edges have color-pattern R (B) on that pole; (ii) there exist at most two virtual edges with color-pattern RB or BR on a pole.

Consider a child node ν_1 having color-pattern either R or B on both poles, say R on pole u and B on pole v, and consider another child node ν_2 having color-pattern R on u and B on v. Nodes ν_1 and ν_2 can be considered as a single node ν^* with color-patterns R and B on the two poles. When the permutation of the P-node has been computed, ν^* is replaced by ν_1 and ν_2. This operation reduces the number of virtual edges to at most 8, namely at most 4 groups of nodes having either R or B on both poles plus at most 2 nodes with color-pattern different from R and B on a pole and at most 2 nodes with color-pattern different from R and B on the other pole. Note that the parent cannot be grouped, since its color-patterns are unknown at this stage. In [13] this fact is exploited to search an embedding with the desired properties by exhaustively checking all permutations, i.e., with a brute-force approach. However, even if the time complexity is aympthotically linear, this yields a huge number of cases, namely $8! * 2^8$ combinations, i.e., all permutations of 8 edges multiplied by all flip choices.

Hence, our implementation uses a different approach in order to search into a much smaller space. Namely, consider any color-pattern, say RBR, and map it to a linear segment of fixed length, partitioned into three parts R, B, R, by two points that represent the two changes of color $R - B$ and $B - R$. Such points are identified by a unidimensional coordinate p along the segment. Given two color-patterns, their segments, and a separating point for each of them, with coordinates p_1 and p_2, respectively, any of the following conditions can hold: (i) $p_1 < p_2$; (ii) $p_1 = p_2$; (iii) $p_1 > p_2$. See Figure 3(a). We call *alignment* of two color-patterns each combinatorial possibility obtained by exhaustively making conditions (i)-(iii) hold for all pairs of separating points of their segments. An alignment of two color-patterns P_1, P_2 uniquely corresponds to a sequence of virtual edges whose color-patterns compose P_1 and P_2 on the two poles. Such sequence makes both poles disjunctive by construction. See Fig. 3(b). Our approach exploits this fact by exhaustively enumerating all alignments of all pairs of color-patterns. The result is the set L containing all and only the disjunctive edge permutations of a generic P-node. L contains exactly 180 elements.

Since the virtual edges of a P-node have a disjunctive permutation if and only if they can be disposed in the same sequence as an element in L, a disjunctive embedding can be found, if it exists, by a brute force search across the 180 elements of L, an impressive improvement with respect to the algorithm in [13].

As the parent node could not be grouped with other nodes, it could impose some additional constraints on the permutation to find that forbid permutations having color-patterns RBR or BRB and that require any r-rimmed (b-rimmed) node to be either the first or the last, if the P-node is r-joined (b-joined).

The whole P-node algorithm must be repeated for every possible choice of semi-flip for the virtual edges admitting it. However, at most two such virtual edges can exist, since they have color-patterns RB or BR on both poles. Hence, the algorithm must be repeated up to 4 times.

6 Experimental Results

In this section we describe the experimental tests performed to check correctness and efficiency of our implementation. When performing experiments a crucial aspect is to have at disposal a representative set of negative and positive instances. Negative instances have the main role of checking correctness, while positive instances are both used to check correctness and to test the performance in a complete execution, without being influenced by early recognition of negative instances. We constructed the former set using ad-hoc examples, conceived to stress each step of the algorithm. On the other hand, to obtain a suitable set of positive instances, we used random generation. To the best of our knowledge, no tool is available to uniformly create graphs with a P2BE. Hence, we devised and implemented a graph generator, whose inputs are a number n of vertices and a number $m \leq 3n-6$ of edges. The output instance is selected uniformly at random among the positive instances of P2BE with n vertices and m edges.

We generated three test suites, Suite 1, 2, and 3, with $m = 2n$, $m = 2.5n$, and $m = 3n - 6$, respectively. For each Suite, we constructed ten buckets of instances, ranging from $n = 10,000$ to $n = 100,000$ with an increment of $10,000$ from one bucket to the other. For each bucket we constructed five instances with the same parameters n and m. The choice of diversifying the edge density is motivated by the wish of testing the performance of the algorithm on a wide variety of SPQR-trees, with Suite 3 being a limit case.

The algorithm was implemented in C++ with GDToolkit [7]. The OGDF library [5] was used to construct the SPQR-trees. We used GDToolkit because of its versatile and easy-to-use data structures and OGDF to construct SPQR-trees in linear time. Among the technical issues, the P-node case required the analysis of a set of cases that is so large to create correctness problems to any, even skilled, programmer. Hence, we devised a code generator that, starting from a formal specification of the constraints, wrote automatically the required C++ code. For performing our experiments, we used an environment with the following features: (i) CPU Intel Dual Xeon X5355 Quad Core (since the algorithm is sequential we used just one Core) 2.66GHz 2x4MB 1333MHz FSB. (ii) RAM 16GB 667MHz. (iii) Gentoo GNU/Linux (2.6.23). (iv) g++ 4.4.5.

Fig. 4(a)–(c) show the total execution times for the three suites. These measurements include the time necessary to decompose the graphs in their connected, biconnected, and triconnected components. The algorithm clearly shows linear running times, with very little differences among the three suites. Figs. 5(a)–(c) show the execution times of the main algorithmic steps for the three suites, namely (i) the total time spent to process biconnected components, (ii) the time spent to deal with the SPQR-trees (excluding the time to create them), and (iii) the time spent to create the green graphs and to find the Eulerian tours. Beside remarking the linear running time, these charts show how the time spent on biconnected components is distributed among the two main algorithmic steps.

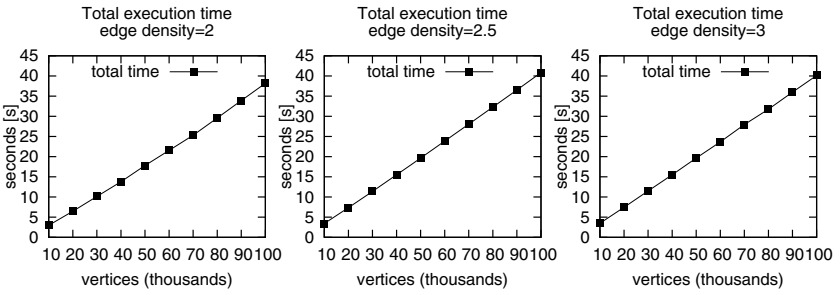

Fig. 4. Total execution time of the algorithm on the three test suites. The y-axis represents the average execution time of the algorithm on the instances in the bucket

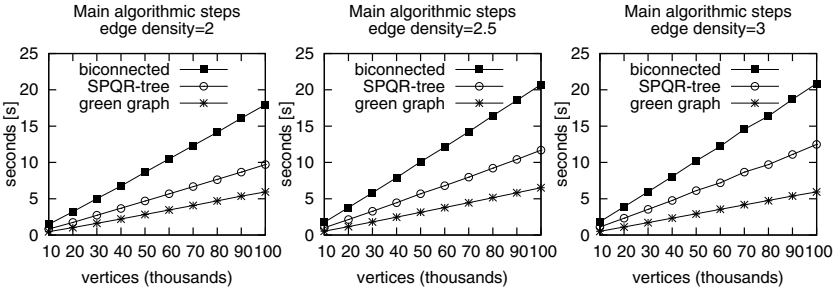

Fig. 5. Execution times of the algorithm for the biconnected components of the three test suites

7 Conclusions

We described an implementation of a constrained version of the 2-page book embedding problem in which the edges are assigned to the two pages and the goal is to find an ordering of the vertices on the spine that generates no crossing on each page. The implemented linear time algorithm is the one given in [13], with several variations aimed at simplifying it and at improving its performance.

We performed a large set of experimental tests on randomly generated instances. From these experiments we conclude that the original algorithm, together with our variations, correctly solves the given problem, and that its performance are pretty good on graphs of medium-large size.

References

1. Angelini, P., Di Bartolomeo, M., Di Battista, G.: Implementing a partitioned 2-page book embedding testing algorithm. CoRR, arXiv:1209.0598v1 (2012)
2. Biedl, T.: Drawing planar partitions III: Two Constrained Embedding Problems. Tech. Report RRR 13-98, Rutcor Research Report (1998)

3. Biedl, T.C., Kaufmann, M., Mutzel, P.: Drawing Planar Partitions II: HH-Drawings. In: Hromkovič, J., Sýkora, O. (eds.) WG 1998. LNCS, vol. 1517, pp. 124–136. Springer, Heidelberg (1998)
4. Buss, J.F., Shor, P.W.: On the pagenumber of planar graphs. In: STOC 1984, pp. 98–100. ACM (1984)
5. Chimani, M., Gutwenger, C., Jünger, M., Klau, G., Klein, K., Mutzel, P.: Handbook of Graph Drawing and Visualization: The Open Graph Drawing Framework. CRC-Press (2012)
6. Cortese, P.F., Di Battista, G.: Clustered planarity. In: SoCG 2005, pp. 32–34 (2005)
7. Di Battista, G., Didimo, W.: Handbook of Graph Drawing and Visualization: GDToolkit. CRC-Press (2012)
8. Di Battista, G., Tamassia, R.: On-line maintenance of triconnected components with SPQR-trees. Algorithmica 15(4), 302–318 (1996)
9. Di Battista, G., Tamassia, R.: On-line planarity testing. SIAM J. Comput. 25, 956–997 (1996)
10. Feng, Q.W., Cohen, R.F., Eades, P.: Planarity for Clustered Graphs. In: Spirakis, P.G. (ed.) ESA 1995. LNCS, vol. 979, pp. 213–226. Springer, Heidelberg (1995)
11. Frati, F., Fulek, R., Ruiz-Vargas, A.J.: On the Page Number of Upward Planar Directed Acyclic Graphs. In: van Kreveld, M., Speckmann, B. (eds.) GD 2011. LNCS, vol. 7034, pp. 391–402. Springer, Heidelberg (2012)
12. Gutwenger, C., Mutzel, P.: A Linear Time Implementation of SPQR-Trees. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 77–90. Springer, Heidelberg (2001)
13. Hong, S., Nagamochi, H.: Two-page book embedding and clustered graph planarity. TR [2009-004], Dept. of Applied Mathematics and Physics, University of Kyoto, Japan (2009)
14. Ollmann, L.T.: On the book thicknesses of various graphs. Cong. Num, vol. VIII, p. 459 (1973)
15. Wood, D.R.: Degree constrained book embeddings. J. of Algorithms 45(2), 144–154 (2002)
16. Yannakakis, M.: Embedding planar graphs in four pages. JCSS 38(1), 36–67 (1989)