Karlsruhe Institute of Technology

Algorithms for crossing minimisation in book drawings

Master Thesis
of

Jonathan Klawitter

At the Department of Informatics
Institute of Theoretical Informatics
Chair of Algorithmics I

Reviewer:	Prof. Dr. Dorothea Wagner
Second reviewer:	Priv.-Doz. Dr. Martin Nöllenburg
Advisor:	Dr. Tamara Mchedlidze

Duration:: 23. October 2015 – 22. April 2016

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 20th April 2016

. . . .

(**Jonathan Klawitter**)

Abstract

A *book* with k pages consists of a line (the *spine*) and k half-planes (the *pages*), each with the spine as boundary. In a *k-page book drawing* of a graph the vertices lie on the spine, and each edge is drawn as arc in one page. The minimal number of edge crossings in a k-page book drawing of a graph is called its *k-page crossing number*, which, in general, is $\mathcal{NP}$-hard to determine [BE14]. A book drawing can be described by an order of the vertices on the spine and a distribution of the edges to pages. To compute book drawings, we combine vertex order heuristics with edge distribution heuristics and create heuristics that compute both simultaneously. We experimentally evaluate the performance of these heuristics on a new test suite that is centred on different graph classes. It turns out that our new heuristics produce drawings with fewer crossings for most of the tested graphs. We further investigate the optimisation of book drawings with force-based approaches, greedy and evolutionary algorithms as well as the successful combinations of the latter two.

Zusammenfassung

Ein *Buch* (*book*) mit k Seiten besteht aus einer Geraden (dem *Bund* (*spine*)) und k Halbebenen (den *Seiten* (*pages*)), die jeweils den Bund als Grenze haben. In einer *k-seitigen Buchzeichnung* (*k-page book drawing*) eines Graphen wird jeder Knoten auf den Bund und jede Kante als Kreisbogen auf eine Seite gezeichnet. Die minimale Anzahl von Kantenkreuzungen in einer k-seitigen Buchzeichnung eines Graphen nennt man seine *k-Seiten Kreuzungszahl* (*k-page crossing number*), welche im Allgemeinen $\mathcal{NP}$-schwer zu bestimmen ist [BE14]. Eine Buchzeichnung kann durch die Ordnung der Knoten auf dem Bund und die Verteilung der Kanten auf die Seiten beschrieben werden. Um eine Buchzeichnung zu erzeugen, kombinieren wir Heuristiken zum Bestimmen von Knotenordnungen und Kantenverteilungen sowie neue Heuristiken, die beides simultan berechnen. Wir evaluieren die Performanz dieser Heuristiken anhand einer neuen Testsammlung, welches auf verschiedenen Graphklassen basiert. Wir konnten zeigen, dass unsere neuen Heuristiken für die meisten der getesteten Graphen Zeichnungen mit weniger Kreuzungen erzeugen. Des Weiteren betrachten wir die Optimierung von Graphzeichnungen mit kräftebasierten und evolutionären Ansätzen, sowie Greedyalgorithmen und vielversprechende Kombinationen selbiger.

Contents

1. Introduction

In this thesis we evaluate heuristics and optimisation algorithms for the NP-hard problem of crossing minimisation in *book drawings*. We start with basic definitions and then proceed with our research question and related work. At the end of the chapter we give an overview of this thesis.

1.1. Book drawings and book embeddings

We draw undirected graphs into topological structures fittingly called *books*. They can be defined as follows.

Definition 1.1. *For $k \geq 1$, a k-**page book**, or a book with k pages, consists of a line in three dimensional space (the spine) and k half-planes (the pages) bounded by this line.*

Less formal, a book consist solely of a spine with attached pages. A graph is drawn into a book by representing its vertices as distinct points on the spine and drawing each edge as a circular arc in exactly one page. In Figure 1.1 a graph with eight vertices is drawn into a book with four pages.

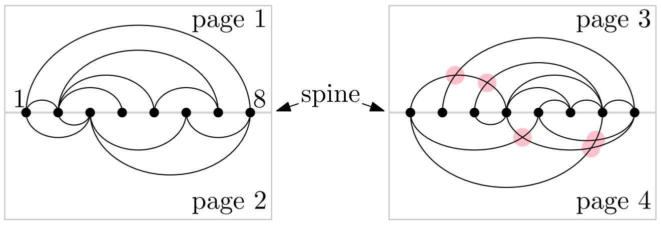


Figure 1.1.: A 4-page book drawing of a graph, where vertices (from 1 to 8) are placed on the spine (shown twice) and edges are distributed to pages. Pages 1 and 2 are crossing-free, whereas the edges in page 3 and 4 produce several crossings.

We note that we can turn around the book, meaning reverse the order of the vertices, and still have a proper book drawing. Furthermore, the numbering of the pages has no further relevance. The drawing stays combinatorially and topologically equivalent. Another feature of book drawings becomes apparent, if we transform them into circular drawings by bending the spine into a cycle, as shown in Figure 1.2. Here, vertices are placed on a circle and edges are drawn as chords. If we cut the circle at a different position than where we joined the ends of the spine, we get a new drawing with rotated order of the vertices. This is also illustrated in Figure 1.2. To represent pages in circular drawings the edges are assigned to layers or are drawn in different colours. Figure 1.3 depicts the 4-page book from Figure 1.1 as circular drawing. Throughout this thesis, we will call the order (respectively circular order) of the vertices, *vertex order*, and the assignment of edges to pages, *edge distribution*. Joining these insights, we can now define the book drawings from the combinatorial point of view.

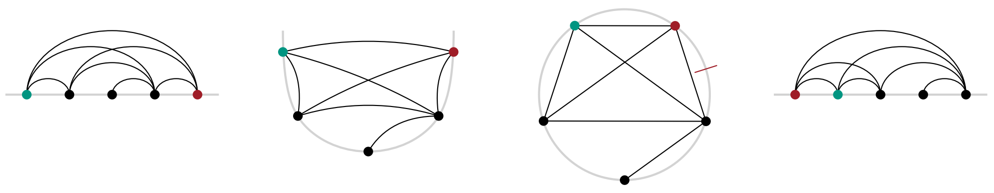


Figure 1.2.: The 1-page book drawing on the left is converted into a circular drawing. The latter is then cut between two other vertices to obtain a book drawing with rotated spine.

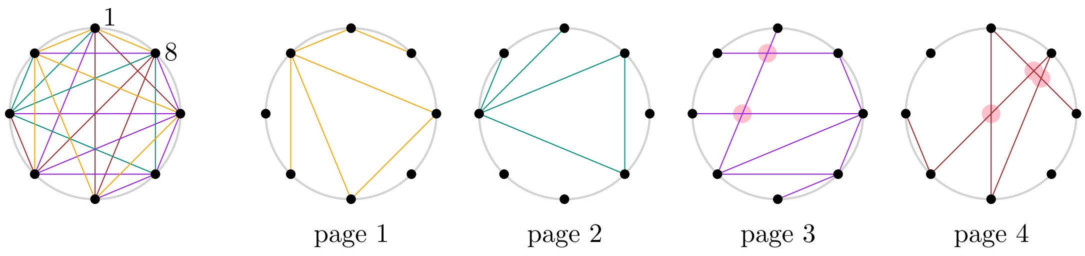


Figure 1.3.: A circular drawing of the book drawing in Figure 1.1, where the pages are represented by the colours of the edges.

Definition 1.2. *A* k*-**page book drawing** of a graph* $G = (V, E)$ *is defined by a vertex order, a circular order of its vertices* V*, and an edge distribution, a map of its edges* E *to* k *pages.*

Similar to planar drawings, there can also be crossings of edges in book drawings. We say two edges uv and xy cross if their arcs (respectively chords) cross. See for example again Figures 1.1 and 1.3 where pages 3 and 4 contain crossings. These are in fact the same crossings, since the transformation described above preserves them. We further observe that a crossing is only possible between non-incident edges and if their end vertices alternate. Hence, we get the following definition for crossings in book drawings.

Definition 1.3. *In a book drawing of a graph* $G = (V, E)$*, two of its edges* $uv, xy \in E$ ***cross** if they are on the same page and their endpoints alternate in the vertex order.*

A planar embedding of a graph G is defined as a planar drawing of G without crossings. A *book embedding* of a graph is defined in the same way in terms of its book drawing.

Definition 1.4. *A* k*-**page book embedding** of a graph* G *is a* k*-page book drawing of* G *without crossings.*

So from now on and throughout this thesis let k be the number of pages. Furthermore, let $G = (V, E)$ be a graph with $n = |V|$ vertices V and $m = |E|$ edges E.

Planar graphs are by definition the only graphs that have planar embeddings. For any other graph G the minimal number of crossings possible in a planar drawing of G is called the *crossing number* $\mathrm{cr}(G)$ of G. In book drawings, on the other hand, we can increase the number of pages and place each edge on its own page to remove any crossings. However, this takes the concept of pages ad absurdum. We are more interested in the minimal number of pages required to find a book embedding of G.

Definition 1.5. *The* ***pagenumber*** *$pn(G)$ (or book thickness or stack number) of a graph G is the smallest k such that G has a k-page book embedding.*

We will use the term *pagenumber of G* for $\mathrm{pn}(G)$. The term *book thickness* is a generalisation of the concept of the graph invariant thickness, which denotes the minimal number of planar graphs into which the edges of a graph can be partitioned. The term *stack number* originates from the analogy between a k-page book embedding and a realisation of a permutation with k stacks.

Finally, we define, similar to the crossing number $\mathrm{cr}(G)$, the minimal number of crossings possible with a fixed number of pages.

Definition 1.6. *The* ***k-page crossing number*** *$cr_k(G)$ of a graph G is the smallest number of crossings possible in a k-page book drawing of G.*

Rephrasing the definition of pagenumber with regard to the k-page crossing number, a graph G has pagenumber k if this is the smallest k for which $\mathrm{cr}_k(G) = 0$.

The pagenumber and the k-page crossing number yield the problems of determining these numbers for any given graph as well as finding one of the best possible book drawings.

Definition 1.7. *The* ***k-page crossing minimisation problem*** *is the problem of finding a book drawing of a given graph G with $cr_k(G)$ many crossings.*

Definition 1.8. *The* ***book embedding problem*** *is the problem of finding a k-page book embedding of a given graph G with $k = pn(G)$.*

We note that these problems can also be defined as decision problems. However, both problems are $\mathcal{NP}$-hard in general, and thus, we will be more interested in heuristic approaches to minimise the number of crossings. We will discuss the computational complexity more detailed in Section 2.2.

1.2. Motivation

The main goal of this thesis is to establish fundamental knowledge about the k-page crossing minimisation problem, in particular to design and evaluate algorithms for it. In the following we discuss the motivation for that as well as for the different aspects we will consider.

In the computational challenge of the 23th International Symposium on Graph Drawing & Network Visualization, in 2015, the task was to minimise the number of crossings in k-page book drawings of ten graphs within one hour. In a practical course at the Karlsruhe Institute of Technology in summer semester 2015, preceding this thesis, we investigated and implemented algorithms for the k-page crossing minimisation problem in order to participate in this challenge. In doing so, we noticed that book embeddings received way more attention in the literature than book drawings. Hence, for this thesis, we got motivated to summarize known results about algorithms for book drawings. We are in particular interested in knowledge necessary for the design and evaluation of new heuristics, i.e. algorithms that compute a new book drawing, as well as optimisation algorithms.

When comparing heuristics for a particular problem, there is the need for a clear evaluation process. However, we observed the lack of systematic approaches in the literature to evaluate the performance of heuristics for book drawings. Hence, one of our goals with this thesis is to create a test suite for book drawing heuristics. We observed that in the literature the choice of graph classes used for testing was neither motivated nor did it seem particular reasonable. It is therefore one of our interest to investigate which graph classes and graph properties are essential for the evaluation of heuristics of book drawings. Furthermore, the k-page crossing minimisation problem was mostly considered only for one or two pages. We are therefore also interested in the performance of heuristics for different number of pages.

After the computation of a book drawing using a heuristic, the problem of optimising it in terms of crossings arises. For this purpose general approaches like greedy optimisation [BB05], evolutionary algorithms [HNS05, HSM07, BSV$^+$08, SSS13] and neural networks [HSM06, LRMCOdLLGM07, Wan08] have been proposed. We are therefore interested in how much they were tailored to book drawings and how sophisticated they are. Furthermore, the question arises in what direction future work concerning optimisation should be directed.

Furthermore, we observed that nearly all algorithms, both heuristics and optimisation algorithms, proposed in the literature considered the vertex order and edge distribution separately. Hence, we were motivated to find out whether it is possible to design algorithms that consider both simultaneously and whether this would lead to better performances.

Before we outline how this thesis is structured and how we tackled this questions, we present related work.

1.3. Related work

The first influential paper on book embeddings is by Bernhart and Kainen from 1979 [BK79]. They gave some first bounds for pagenumbers, described the graphs classes that can be embedded in one and two pages, namely outerplanar and subhamiltonian planar graphs, and determined the page number for complete graphs as well as some complete bipartite graphs. In the following finding pagenumbers for different graph classes was given a lot of attention. For example, the upper bound on the pagenumber of planar graphs was gradually improved to nine, seven and finally to four [BS84, Hea84, Yan86, Yan89]. Determining whether the pagenumber of planar graphs is three or four is still ongoing research [BKZ15]. This is also the case for other graph classes like, for example, 1-planar graphs [BBKR15, ABK15], where the current best lower and upper bounds are four and sixteen. Other considered graph classes are for example de Brujin and shuffle-exchange graphs [Obr93], Cayley graphs [TS06], generalize Petersen graphs [Mah13], hypercubes and cube-like graphs [KHT89, Has09, TS10] and k-trees [GH01, DW07, DW09, VWY09]. There exist several overviews [DW04, Alh05] for these results. Bounds on the pagenumber have been given with respect to n [BK79, McC12], to m [Mal94b, BS14, BS15] and to the graph genus [HI92, Mal94a]. It was also considered to embed graphs in books with small pagewidth, i.e. minimising the maximal number of edges cut by a line perpendicular to the spine [Hea85, Hea87, CLR87, Stö88].

Circular drawings, defined like our circular drawings but with only one page, are also related to book embeddings. Several algorithms have been proposed [Mä88, HS04, BB05, ST06] for the objectives to reduce the number of crossings or the total edge length in these drawings. In fact, most heuristics that we describe in Section 3.1 to compute a vertex order are originally for these circular drawings. There are also algorithms which concern readability and aesthetics of circular drawings [ST06, GK07, DNEH13].

For books with two pages the crossing number is of particular interest, since it gives an upper bound to the planar crossing number cr. There are several heuristics proposed for the problem with fixed spine [Cim02, Cim06, CM07], called the fixed linear crossing number problem, as well as for the general case [HSV05, HSSV06, SSS13]. Genetic and evolutionary algorithms as well as simulated annealing have been considered for crossing minimisation in book drawings with one page [HNS05], two pages [HSM07, PMH07, BSV$^+$08] and any number of pages [SSS13]. Evolutionary algorithms have also been used to determine pagenumbers [KRSZ02, SSG11]. Furthermore, neural networks have been proposed and tested for crossing minimisation in books [HSM06, LRMCOdLLGM07, Wan08]. Extensive evaluation of several algorithms has been done by Satsangi et al. [SSS13] and He et al. [HSMV15]. Aside from finding good algorithms for the general case, there has also been made effort to compute crossing numbers for special graphs and k's [HSSV06, HSM10, AAFM$^+$12, dKP12, dKPS13, dKPS14].

Along with the complexity of the two problems, we will also discuss the result by Bannister and Eppstein [BE14] concerning fixed parameter tractable algorithms for the crossing minimisation problem in the next chapter.

The kind of book embeddings we consider are not the only known in the literature. One variation for example are the so called *topological book embeddings*, where edges are allowed to cross the spine [EM99, MS09, MS10, DGGL11, GLM$^+$15]. Topological book embeddings have already been considered by Bernhart and Kainen [BK79]. Another generalization of book embeddings is due to Overbay [Ove98], who considered books with only one page but a tree as spine instead of a single line.

k-page book embeddings are also known as k-stack layouts and the pagenumber as stack number. This is due to the fact that book embeddings can be used to sort permutations with stacks. Analogously, *k-queue layouts* (respectively *k-arch layouts*) can be defined as drawings with vertices placed on a spine and non-nested (respectively non-disjoint) edges. Dujmović and Wood [DW04] gave a survey on these layouts. Schaefer [Sch13a] gave a broader survey on different crossings numbers.

Book drawings and especially book embeddings find multiple applications in different fields. For example, Clote et al. [CDD$^+$12] considered a connection between book embeddings and RNA folding. Another use of book embeddings is due to Jacobson [Jac89]. He described succinct data structures for graphs with bounded pagenumber based on their book embeddings. Dujmović and Wood [DW04] gave a long list of references to applications like fault tolerant VLSI design, complexity theory, graph drawing, sorting permutations and compact routing tables. McClintock [McC12] gave nice explanations for most of these applications, including in addition optimised VLSI design. Another application is due to Kainen [Kai90], who described the analogy of book embeddings and traffic control at a controlled intersection. More precisely, while the vertex order is given by the circular order in which the lanes for cars and passengers arrive at the intersection, the phases of the traffic signals that prevent crossings lanes from interfering resemble the pages.

1.4. Outline

We will start in Chapter 2 with an overview of the k-page crossing minimisation problem. This will include some basic result, the computational complexity of the problem as well as ways to solve it exactly. Furthermore, we will address the problem of counting crossings in book drawings and propose a new fast algorithm for it.

In Chapter 3 we will look at heuristic approaches to compute book drawings. These heuristics can be categorized as algorithms that compute only a vertex order, an edge distribution or a full book drawing. Especially the latter group has not really been considered in the literature yet, instead mostly combinations of vertex order and edge distribution heuristics have been used. We will discuss this problem and also propose new algorithms for each of these groups. An evaluation of all these heuristics will be done in Chapter 4 based on results from experiments on random graphs of different densities as well as on particular graph classes. In doing so, we will also introduce a test suite for the evaluation of heuristics.

In Chapter 5 we will discuss optimisation algorithms for the k-page crossing minimisation problem. These are in particular force based and greedy approaches, evolutionary algorithms as well as their combinations. Furthermore, experimental evaluation of these algorithms will be presented. Beyond that, we discuss directions for future work concerning optimisation algorithms.

In Chapter 6 we will summarize our findings, draw conclusions and give an outlook.

2. The problem

In this chapter we investigate the complexity of the book embedding and k-page crossing minimisation problem. We will start with cases of these problems, where they are easy to solve or their complexity can be reduced. In Section 2.1.4 we will show that when the vertex order is fixed, these problems are equivalent to other well known problems. However, as already mentioned, the problems are NP-hard in general. This will be addressed in Section 2.2. After that, we investigate whether and when one may still try to solve the problems exactly. We close the chapter with a problem that in turn is easy so solve. More precisely, in Section 2.4 we consider algorithms to count the number of crossings of a book drawing in a fast manner.

2.1. Basic results

It is sufficient to only consider simple graphs, i.e. graphs without loops and without multi-edges. Loops can be embedded directly next to a vertex without interfering any other edges. Multi-edges can be drawn parallel on the same page and thus would only multiply the number of crossing of the underlying edge. Furthermore, it is easy to see that if a graph consists of multiple connected components, these can be embedded in books separately. Hence, throughout this thesis we only consider simple and connected graphs. Beyond that, there are also graph classes that are easy to embed in books with only one page.

2.1.1. Trees

The graphs that have zero crossings in all of their book drawings are the stars. This is due to the fact that two edges can only cross if they are not incident. Another class of graphs that is easy to embed is the class of trees, as illustrated in Figure 2.1.

Theorem 2.1. *Every tree has pagenumber 1.*

To find an embedding one can simply use a depth-first search on the tree and order the vertices as they are visited. This strategy is actually used for arbitrary graphs by some vertex order heuristics presented in Section 3.1. However, not all heuristics presented there are able to embed trees in a 1-page book.

Moreover, consider a graph G with an attached tree T. By attached tree, we mean an induced subgraph that is a tree and connected with the rest of the graph only via one cut

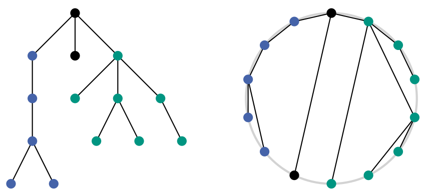


Figure 2.1.: A tree on the left and a circular drawing of it on the right.

vertex v. We can now draw G without T (except of course v) in a book and then in a second step add an embedding of T starting at v, as depicted in Figure 2.2. Clearly T does not produce any crossings. Moreover, this can be done for any attached tree.

Observation 2.2. *Cutting of attached trees of a graph does not influence its minimal number of crossings achievable in a k-page book drawing.*

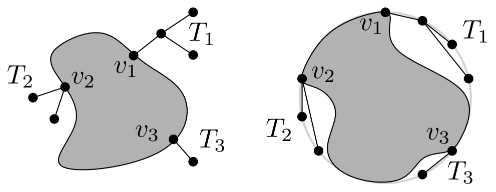


Figure 2.2.: On the left, a graph with three attached trees T_i and a book drawing of it on the right, showing that trees do not influence the rest if placed next to their cut vertices v_i.

As mentioned above, not all algorithms embed trees without crossings and thus also not attached trees. Furthermore, crossings produced by trees can not always be resolved by a greedy optimisation algorithm that takes vertices one by one and places them at the position where the number of crossings gets reduced the most. This can indeed resolve some crossings, however, in general the drawing can be of such kind that no improvement is possible. In fact, the following theorem holds.

Theorem 2.3. *A greedily optimised 1-page book drawing of a tree can still have $\mathcal{O}(n^2)$ crossing.*

Proof. We prove this in two steps. First, we show that a drawing can contain crossings unresolvable by greedy optimisation. Second, we give a construction to extend a tree, such that all of its crossings are of this kind, enabling us to devise an example.

Consider the tree of Figure 2.3 where the edges uv and e cross each other. To resolve this crossing, without loss of generality, either u or v has to be moved to the other side of e. However, this clearly introduces two new crossings and thus greedy optimisation would move neither of them. Moving any other vertex either increases the number of crossings or does not alter the basic structure. Hence, greedy optimisation can not resolve this crossing.

In general, consider a vertex v of degree d. If we attach $d + 1$ new leaves to v and place them next to v on the spine, we bind v to its position. Doing this with all vertices of the tree yields that no vertex can be moved without increasing the number of crossings. We note that this construction only increases the number of edges by a constant factor. Hence, applying this algorithm to an embedded tree with quadratically many crossings yields an example. One such example is shown in Figure 2.4 □

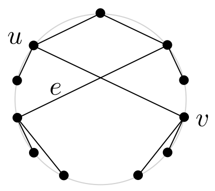


Figure 2.3.: A tree with a crossing that can not be resolved by greedy optimisation. Neither u nor v can be moved over e, nor does moving any other vertex have any helping effect.

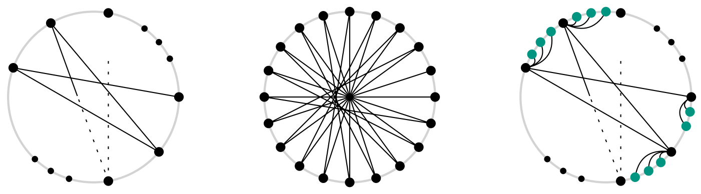


Figure 2.4.: The figure on the left indicates a construction to draw a path into a book with quadratically many crossings. The path alternates across the first edge and thus every edge crosses all except other edges except its incident ones. For a path with $n = 20$ this yields the book drawing in the middle. How to add the additional leaves to make the crossings unresolvable is shown on the right.

In the following experiment we tested how successful greedy optimisation can reduce the number of crossings in drawings produced by one of the heuristics from Section 3.1. More precisely, we first ran the vertex order heuristic `conCro` (which ranks good in the experiments in Chapter 3) on 1000 random trees with 10 to 200 vertices and afterwards greedy optimisation on the resulting book drawings. The diagram in Figure 2.5 shows the average number of crossings before and after the optimisation as well as the percentage of crossings resolved on average. We can observe that greedy optimisation could indeed not resolve all crossings, in fact for trees with 60 vertices not even 50%.

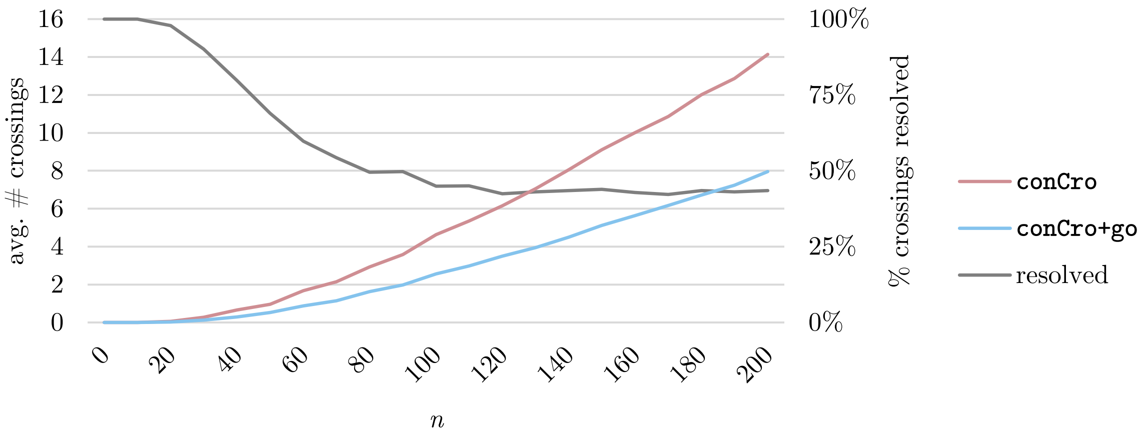


Figure 2.5.: The average crossings on random trees produced by the vertex order heuristic `conCro` before and after greedy optimisation. Also shown the percentage of resolved crossings.

Hence, based on this experiment and the fact that reducing the problem size is in general a good idea, we can conclude that when drawing graphs in books one should handle attached trees separately.

2.1.2. Biconnected components

The idea to implement parts of a graph separately can be extended even further to biconnected components. This has already been shown by Bernhart and Kainen [BK79] in their early work.

Theorem 2.4 ([BK79])**.** *The pagenumber of a graph is the maximal pagenumber of its biconnected components.*

Hence, when embedding a graph in a book one may first identify the graphs biconnected components, embed each separately and then reassemble the resulting embeddings. As Heath [Hea85] showed, disassembling and reassembling can be done in linear time. Extracting the biconnected components can be done with the depth-first search based algorithm by Hopcroft and Tarjan [HT73]. Combining the book embeddings of the biconnected components works straightforward and much like combining a graph and its attached trees. Groneman [Gro15] described this in detail using a block-cut tree of the graph, which is also illustrated in Figure 2.6.

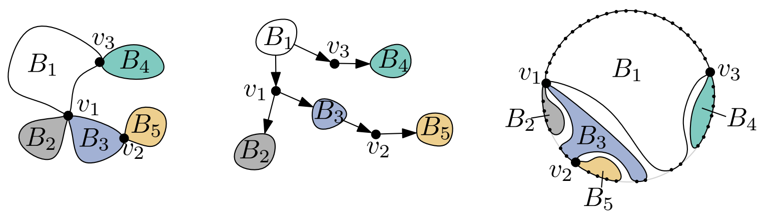


Figure 2.6.: Connected graph with three cut vertices, a corresponding block-cut tree and resulting book embedding after reassembling the embeddings of the biconnected components (as in [Gro15]).

Considering biconnected components separately is obviously not only beneficial for book embeddings but also book drawings. In fact, only one of the vertex order heuristics presented later produces no crossings between biconnected components (if drawn in 1-page books). Nevertheless, testing whether edges of different components can cross is clearly overhead. Hence again, in order to reduce the size of the problems and to reduce the number of crossings in book drawings produced by heuristics, it is advisable to consider biconnected components separately.

2.1.3. Outerplanar graphs

A graph is *outerplanar* if it has an embedding in the plane such that all vertices are on the same face, normally and without loss of generality on the unbounded face. It is called *maximal outerplanar* if no edge can be added while preserving outerplanarity. All maximal outerplanar graphs have exactly $2n - 3$ edges. Outerplanar graphs take an important role in book embeddings due to the following theorem.

Theorem 2.5 ([BK79])**.** *A graph has pagenumber 1 if and only if it is outerplanar.*

As illustrated in Figure 2.7, a proof is straightforward. With out loss of generality, we can assume the graph is maximal outerplanar. Otherwise, we simply make it maximal, embed it and finally remove previously added edges again. The maximality induces that no vertex is incident to the outer face more than once. Hence, given a planar embedding of the graph, we can derive a vertex order from the order of the vertices on the outer face. Obviously, this yields a crossing-free circular drawing.

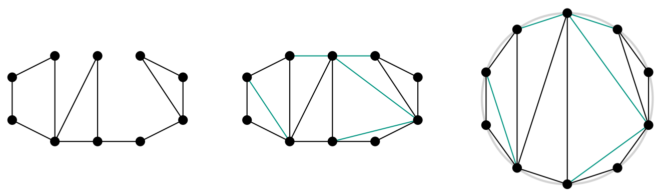


Figure 2.7.: An outerplanar graph, the same made maximal and the induced circular drawing.

Actually, this is the only way to achieve zero crossings for a maximal outerplanar graph. This follows directly from the fact that every biconnected outerplanar graph has a unique planar embedding (up to combinatorial equivalence) [Pat13].

Conclusion 2.6. *A biconnected outerplanar graph is embedded with zero crossings in a 1-page book if and only if the vertex order is the order on the outer face of its planar embedding.*

Rephrasing Theorem 2.5 we get that a crossing-free page of a book embedding can contain at most one outerplanar graph and $2n-3$ edges. This directly yields that any graph needs at least $\frac{m}{2n-3}$ pages. This lower bound was improved by Bernhart and Kainen [BK79], as follows. We note that the edges of the enclosing cycle of the outerplanar graph can each be in at most one page, which without loss of generality can be page 1. Consequently all other pages can contain at most the inner edges of an outerplanar graph on n vertices and thus at most $n-3$ edges. Hence, we get the following result.

Conclusion 2.7 ([BK79])**.** *The pagenumber of a graph with n vertices and m edges is at least $\frac{m-n}{n-3}$.*

This bound is tight, for example, for complete graphs (see Theorem 4.11). Furthermore, Bernhart and Kainen [BK79] derived from this observation bounds on the average degree and the chromatic number of a graph with respect to its pagenumber.

2.1.4. Fixed vertex order

We now consider the k-page crossing minimisation problem when the vertex order is already fixed. This might be due to separate computation of vertex order and edge distribution, from the context of an application or because there is only one vertex order, which is the case for complete graphs.

Consider a circular drawing of a graph $G = (V, E)$ with fixed vertex order, where the edges are drawn as chords. A *circle graph* is the intersection graph of chords in a circular drawing, i.e. the vertices correspond to the chords and two vertices are adjacent if their chords intersect. So for every graph with fixed vertex order we get a corresponding circle graph. Then again, if two chords cross, it is possible that the corresponding edges of E in the book drawing cross. Therefore, we say these edges are in *conflict* and call the circle graph from now on *edge conflict graph*. An example is shown in Figure 2.8.

Definition 2.8. *The* ***edge conflict graph*** *G_c of a graph $G = (V, E)$ and given vertex order is the circle graph of a circular drawing of G with the given vertex order.*

We can now observe two things. The edges of G can be distributed to k pages without crossings only if the edge conflict graph can be coloured properly with k colours. Furthermore, distributing the edges of G to k pages is equivalent to partitioning the vertices of

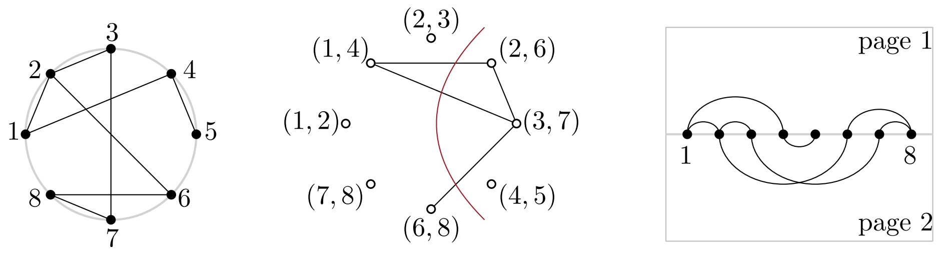


Figure 2.8.: An ordered graph and its corresponding edge conflict graph.

G_c into k partitions. Avoiding crossing in the book drawing is then equivalent to placing adjacent vertices of G_c in different partitions or, in other words, cut as many edges as possible. Latter problem is known as Max k-Cut. Figure 2.8 shows an example with $k = 2$. As Chung et al [CLR87] and de Klerk et al. [dKPS13] have observed, these problems are in fact equivalent:

Observation 2.9. *For a graph G with fixed vertex order and the corresponding edge conflict graph G_c holds:*

1. *G admits a k-page book embedding if and only if G_c is k colourable [CLR87].*
2. *The k-page crossing number problem of G is equivalent to the Max k-Cut problem of G_c [dKPS13].*

From the point of view of algorithm engineering it is worth mentioning that the edges between vertices consecutive in the vertex order can not be part of any crossing. These are in Figure 2.8 the edges $(1, 2), (2, 3), (4, 5)$ and $(7, 8)$, which are all singletons in the edge conflict graph. Consequently they can be placed on any page and ignored by the algorithm when distributing the other edges.

To sum up, we have seen that graphs considered in the book embedding and the k-page crossing number problem can be reduced to simple, biconnected graphs. Furthermore, in a book embedding one page contains at most one outerplanar graphs or its inner edges, respectively. Moreover, when the vertex order is fixed, the problems are equivalent to colouring circle graphs and Max k-Cut on circle graphs, which we call edge conflict graphs. We proceed now with results concerning the computational complexity.

2.2. NP-hardness

As mentioned before, the book embedding and the k-page crossing number problem are both NP-hard in general. In this section we state the respective results.

2.2.1. Book embedding

We have seen that a graph has a 1-page book embedding if and only if it is outerplanar. An embedding of a graph in a 2-page book is equivalent to a planar embedding with the restriction that all vertices are collinear. Hence, a graph that has a 2-page book embedding has to be planar. As Bernhart and Kainen [BK79] observed, this is however not sufficient. The graph also has to be Hamiltonian or *subhamiltonian*, which is defined as being subgraph of a Hamiltonian planar graph. We will see this graph class again in Section 4.4. Figure 2.9 illustrates these conditions and why the following theorem holds.

Theorem 2.10 ([BK79])**.** *A graph has pagenumber at most 2 if and only if it is planar and subhamiltonian.*

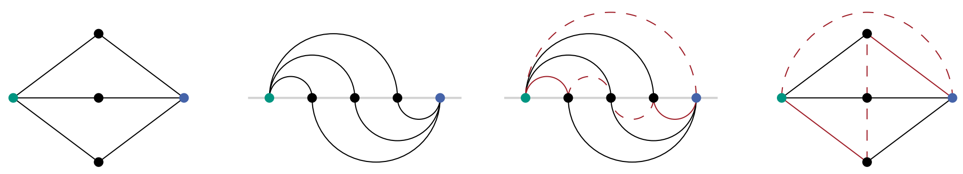


Figure 2.9.: The (non-outerplanar) planar subhamiltonian graph $K_{2,3}$, a two-page embedding and then both embeddings with added Hamiltonian path.

However, Wigderson [Wig82] showed that deciding whether a maximal planar graph is Hamiltonian is $\mathcal{NP}$-complete. Thus, as a consequence we can draw the following conclusion.

Conclusion 2.11 ([CLR87]). *Deciding whether a graph admits a 2-page book embedding is $\mathcal{NP}$-complete.*

If, however, a vertex order is predetermined or forced by a gadget graph (as proposed by Schaefer [Sch13a]), the problem of finding a book embedding is, as mentioned above, equivalent to colouring the corresponding edge conflict graph. This problem is also $\mathcal{NP}$-complete [GJMP80], at least for $k \geq 4$ [Ung88]. Unger [Ung92] claimed that it is solvable for $k = 3$ in polynomial time.

Conclusion 2.12 ([CLR87]). *Determining whether a graph with fixed vertex order admits a k-page book embedding for $k \geq 4$ is NP-complete.*

Consequently, the same holds when the vertex order is not known.

For $k = 2$ the problem can be reduced to planarity testing by adding edges between consecutive vertices in the vertex order. Hence, it can be solved in linear time [Pat13]. Also the contrary problem, where the edge distribution is fixed, can be solved in linear time for $k = 2$ [HN09].

2.2.2. Crossing number

Masuda et al. [MNKF90] proved that the 1-page crossing number problem is $\mathcal{NP}$-complete. Since the k-page book embedding problem is $\mathcal{NP}$-complete for $k > 1$, the less restricted k-page crossing number problem is also $\mathcal{NP}$-complete.

Conclusion 2.13 ([BE14]). *Determining the k-page crossing number $cr_k(G)$ of a graph G is $\mathcal{NP}$-complete.*

Bannister and Eppstein [BE14] have shown that cr_1 and cr_2 are fixed parameter tractable with respect to the number of crossings respectively to the sum of the number of crossings and the treewidth. However, they mention that their algorithms depend highly on the parameters and are therefore impractical. Nevertheless, this shows that treewidth and thus K-trees, which we will discuss in Section 4.6, are tightly coupled with book drawings.

In the next section we consider the question whether one may try to solve the embedding and crossing minimisation problems even though they are $\mathcal{NP}$-hard.

2.3. Exact solution

Several approaches to solve book embedding and k-page crossing minimisation problems exactly can be found in the literature [dKP12, dKPS13, BKZ15]. We describe them in more detail in the following.

A graph can be drawn in a k-page book with $\frac{(n-1)!}{2}$ different spines. This is due to the fact that there are $n!$ many permutations of the n vertices, for which we identify those n yielding the same circular order and then the pairs with mutually reversed order. With $k = 2$ there are already 2^{m-1} different edge distributions, (Stirling number of the second kind $s_k^{II}(m)$ different for any $k > 0$, if we request that no page is empty). So brute force becomes impossible for $n > 20$ [Sed77] or, if the vertex order is fixed, for large m. Hence, one has to make use of the fact that some solutions are obviously not the best. This has been done for example by de Klerk and Pasechnik [dKP12] to compute cr_2 for some complete graphs with a branch and bound algorithm. However, for $n \in \{19, 21, 22, 23\}$ or $n > 25$ they had stopped their computations unsuccessfully after 60 hours. De Klerk et al. [dKP12, dKPS13] also used known bounds on complete and complete bipartite graphs in combination with new bounds computed by semidefinite programming to establish further results for some small values of n.

Another way to solve the book embedding problem and the k-page crossing number problem is by formulating them as satisfiability problem (SAT) and weighted maximum satisfiability problem (*Weighted MAX-SAT*), respectively. In latter the aim is to maximise the weight of the satisfied clauses. In the remainder of this section we first outline the transformation to SAT and then the extension to *Weighted MAX-SAT*.

2.3.1. SAT

We consider the book embedding problem as decision problem asking whether a graph G can be embedded without crossings in a k-page book, i.e. whether $\mathrm{pn}(G) \leq k$. To transform this problem into a satisfiability problem we have to express that the vertices are in a proper order and that edges are distributed to pages and do not cross. We note that to describe the vertex order it is enough to describe a linear order of the vertices instead of cyclic order. Hence, we use again the concept of a spine, say a horizontal spine, where the vertices are ordered from left to right. Thus $v_i < v_j$ means that vertex v_i is left of v_j. In the following we will first establish the variables and then the clauses for each of the necessary conditions using the notations of Bekos et al. [BKZ15].

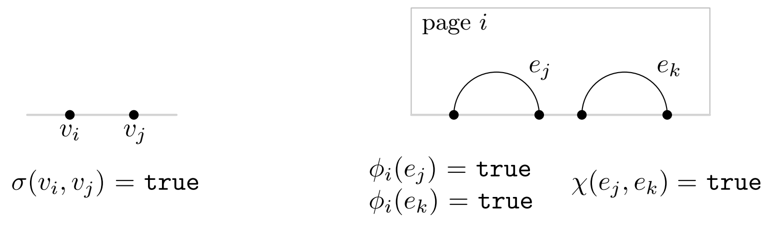
We have variables $\sigma(v_i, v_j)$ to describe the vertex order, $\phi_i(e_j)$ for the edge distribution and $\chi(e_i, e_j)$ for possible crossings, as illustrated in Figure 2.10.

$$\begin{aligned} \forall v_i, v_j \in V, i < j: &\quad \sigma(v_i, v_j) \\ \forall e_j \in E, i \in \{1, \dots, k\}: &\quad \phi_i(e_j) \\ \forall e_j, e_k \in E, j \neq k: &\quad \chi(e_j, e_k) \end{aligned} \tag{2.14}$$

Here, $\sigma(v_i, v_j) =$ `true` if vertex v_i is left of v_j, $\phi_i(e_j) =$ `true` if edge e_j is on page i, and $\chi(e_j, e_k) =$ `true` if the edges e_j and e_k are on the same page.

To define a proper ordering, σ has to be antisymmetric and transitive. We want to note that we define $\sigma(v_i, v_j)$ only for $i < j$. To state that vertex v_j is left of v_i, we simply take $\neg\sigma(v_i, v_j)$. Conveniently, this also yields the antisymmetry. Transitivity on the other hand is ensured by the following rule.

$$\sigma(v_i, v_j) \wedge \sigma(v_j, v_k) \rightarrow \sigma(v_i, v_k) \tag{2.15}$$



v_i v_j
$\sigma(v_i, v_j) = \texttt{true}$
page i
e_j
e_k
$\phi_i(e_j) = \texttt{true}$
$\phi_i(e_k) = \texttt{true}$
$\chi(e_j, e_k) = \texttt{true}$

Figure 2.10.: Illustration of variables $\sigma(v_i, v_j)$ to describe the vertex order, $\phi_i(e_j)$ for the edge distribution and $\chi(e_i, e_j)$ for possible crossings.

Since a book embedding actually has a circular order we can chose one vertex, say v_1, to be at the first position in the non-circular order. Furthermore, we know that a spine is also valid if reversed. By fixing the order of two further vertices, say v_2 and v_3, we fix the orientation of the spine as well. This is encoded by the rules 2.16.

$$\begin{aligned}&\sigma(v_1, v_i) \;\; \forall v_i \in V, i \in \{2, \ldots, n\}\\ &\sigma(v_2, v_3)\end{aligned} \tag{2.16}$$

In a proper edge distribution every edge is assigned to exactly one page. However, it is enough to ensure with rule 2.17 that an edge is assigned to at least one page. Furthermore, we can again reduce the search space by fixing one edge to one page, e.g., e_1 to page 1, as achieved by 2.18.

$$\phi_1(e_i) \lor \phi_2(e_i) \lor \ldots \lor \phi_k(e_i) \;\; \forall e_i \in E \tag{2.17}$$

$$\phi_1(e_1) \tag{2.18}$$

Next, we have to connect possible crossings with the vertex order and the edge distribution. $\chi(e_i, e_j)$ has to be `true` if and only if e_i and e_j are on the same page. In this case, however, to avoid a crossing, they either have to share a vertex or the order of their vertices can not alternate. This is accomplished by the following rules.

$$((\phi_1(e_i) \land \phi_1(e_j)) \lor \ldots \lor (\phi_k(e_i) \land \phi_k(e_j))) \rightarrow \chi(e_i, e_j) \;\; \forall e_i, e_j \in E \tag{2.19}$$

$$\begin{aligned}&\forall (v_i, v_j), (v_k, v_l) \in E, v_i, v_j, v_k, v_l \text{ pairwise different :}\\ &\chi((v_i, v_j), (v_k, v_l)) \rightarrow\\ &\neg(\sigma(v_i, v_k) \land \sigma(v_k, v_j) \land \sigma(v_j, v_l)) \land \ldots \land \neg(\sigma(v_l, v_j) \land \sigma(v_j, v_k) \land \sigma(v_k, v_i))\end{aligned} \tag{2.20}$$

Finally, these rules can be converted in CNF clauses straightforwardly. Summing up we get $\mathcal{O}(n^2 + m^2 + km)$ variables and $\mathcal{O}(n^3 + m^2)$ clauses.

Bekos et al. [BKZ15] investigated whether planar graphs have pagenumber three or four. They used a SAT solver to check critical instances and whether solutions have certain properties. Of interest for us are reports concerning the running time. We want to note here that planar graphs have at most $3n-6$ edges. For graphs up to 100 vertices a solution was mostly found within 3 seconds. However, for graphs with n in the range 500 and 700 they have reported running times of "few hours to a couple of days".

2.3.2. Weighted MAX-SAT

The k-page crossing minimisation problem is, in contrast to the book embedding problem, not that easy to convert into a decision problem. One approach could be to add an allowed number of crossings into the SAT formulation above and use binary search to determine the optimum. Another approach was used by de Klerk et al. [dKPS13], namely *Weighted MAX-SAT*, to solve the k-page crossing minimisation problem with fixed vertex order as optimisation problem. We extended this approach to the unrestricted case. In a MAX-SAT problem one seeks to maximise the number of simultaneously satisfiable clauses. In a *Weighted MAX-SAT* problem not the number but the weight of the satisfied clauses gets maximised.

For the formulation of the k-page crossing minimisation problem as *Weighted MAX-SAT*, we can reuse the SAT formulation from above. However, in addition, we have to ensure that in all solutions the vertex order σ and the edge distribution ϕ is valid, i.e. the respective clauses are always fulfilled. Only the clauses ensuring that there are no crossings should get object of optimisation. If the vertex order is fixed (as in the case of the work by de Klerk et al. [dKPS13]) this can be done straight forward by setting the weights, such that it is always better to first fulfill the needed constraints than to avoid any crossing. When the vertex order is not fixed it is more complex.

Rule 2.20 ensured that if two edges are on the same page, the involved vertices are in a correct order. These rules get converted to multiple clauses in CNF. Hence, if only some of these clauses are fulfilled, it does not follow that there is no crossing. They have to be satisfied simultaneously and as the clauses for the vertex order and edge distribution should not be subject to the maximisation process. Consequently we introduce new variables for avoided crossings $\beta((v_i, v_j), (v_k, v_l))$ for all edge pairs. Let $\chi_{i,j,k,l}$ represent the rule 2.20. Now we add the following rule, which says that $\beta((v_i, v_j), (v_k, v_l))$ is equal to whether $\chi_{i,j,k,l}$ is fulfilled.

$$\begin{aligned} &\forall (v_i, v_j), (v_k, v_l) \in E, v_i, v_j, v_k, v_l \text{ pairwise different :} \\ &\beta((v_i, v_j), (v_k, v_l)) \leftrightarrow \chi_{i,j,k,l} \end{aligned} \tag{2.21}$$

Making the clauses resulting from a transformation of 2.21 to CNF mandatory by choosing appropriate weights, we can bundle whether a crossing occurs or not. To optimise the number of avoided crossing we simple add the clauses $(\beta((v_i, v_j), (v_k, v_l)))$ for all edge pairs. By giving these clauses the weight 1 and all other clauses appropriate high enough weight, we achieve that only the number of avoided crossings gets maximised and thus the number of crossings minimised.

Weighted MAX-SAT problems are obviously harder to solve than MAX-SAT and SAT problems. Hence, the maximal number of vertices and edges for which the running time is still practicable is even more constrained for crossing minimisation in book drawings. de Klerk et al. [dKPS13] used *Weighted MAX-SAT* on complete graphs (hence, with fixed vertex order) only for up to 13 vertices. They report that for $n > 13$ no result was found within 48 hours.

We conclude that even for small instances of the k-page crossing minimisation problem as well as for the book embedding problem computing exact solutions is impractical. Here, small instances means only just double-digit number of vertices, when minimising the number of crossing, and up to at most some hundred vertices, when computing a full book embedding of sparse graphs. Consequently, in the remainder of this thesis we examine heuristics to compute book embeddings as well as optimisation algorithms. However, before that, we examine the problem of counting crossings in book drawings.

2.4. Counting crossings

We are interested in algorithms that minimise the number of crossings in book drawings and thus have to count these crossings. It is not only natural to ask for a fast algorithm, but also relevant for the performance of the experiments on these algorithms. Furthermore, one approach to optimise book drawings is by using evolutionary algorithms (see Section 5.2). There, the currently computed book drawings of one generation are evaluated and selected for recombination and survival based on their number of crossings. Hence, counting the crossings in a fast manner gets essential for this approach.

We will now look at three different approaches to count crossings. At first we consider a simple and naive algorithm. Second, we look at an approach with running time depending on the number of crossings. Last but not least, we show how to transform the problem to counting crossings in two-layered drawings. At the end of this section an experimental evaluation of these algorithms is presented.

2.4.1. Counting and reporting

The straightforward way to count the number of crossings is to check all pairs of edges whether they cross. This can be done easily by checking the mutual positions of their end vertices. Since a graph with m edges has $\frac{m(m-1)}{2}$ edge pairs, the algorithm runs in $\mathcal{O}(m^2)$ asymptotic time. Nonetheless, this algorithm can not only count, but also report the crossings and since there can be up to $\mathcal{O}(m^2)$ crossings in the worst case, there can be no asymptotically better algorithm. However, we are mostly not interested in a list of all crossings, rather only their count.

2.4.2. Open edges sweep

Six and Tollis [ST06] describe a method to count crossings in circular drawings with one page. This approach can easily be adopted to work with with more pages. For ease of exposition, we describe the algorithm for the case $k = 1$ first and then how to extend it. We consider again the vertex order as horizontal spine, i.e. linear order with vertices x left of y denoted by $x < y$. We start with a basic observation.

Observation 2.22. *Two disjoint edges uv and xy placed on the same page can only cross if xy starts between uv, i.e. $u < x < v$, or vice verse, i.e. $x < u < y$.*

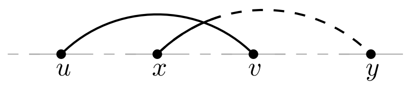


Figure 2.11.: Case one of Observation 2.22, with xy starting between uv.

The algorithm (see Algorithm 1) sweeps along the vertices on the spine, visiting incident edges as they occur in some order. We call an edge uv *open*, if it has been processed at its start vertex u, but not at its end vertex v. If we visit an edge during the sweep at its start vertex, we add it to a list of open edges (Algorithm 1, lines 13 - 14). The counting of crossings happens, when we visit the end vertex v of an open edge uv (lines 6 - 12). Then we know by Observation 2.22 that all open edges currently in the list after uv can cross uv, see Figure 2.12 (a) for an illustration. Thus, we process the list backwards, counting the visited and thus open edges, until we arrive at u. The edge uv is now closed and thus removed from the list of open edges.

We should not count edges also starting at u or ending at v, since they do not produce crossings with uv. This can easily be avoided if we process the edges at a vertex in a particular order, more precisely, the one shown in Figure 2.12 (b).

The count of visited edges above equals the number of crossings of uv with edges xy, where $u < x$. A crossing of uv with an edge $x'y'$ and $x' < u$ is counted when $x'y'$ gets closed. Therefore, we can conclude that sweeping over the whole spine counts all crossings. To adopt the algorithm for $k > 1$ one simply takes for each page a separate list for the open edges.

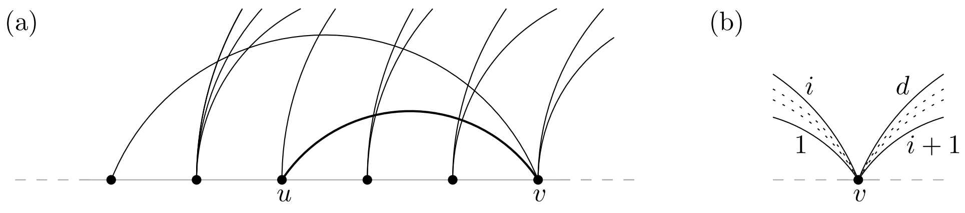


Figure 2.12.: Example of open edges, which can and can not cross uv (a), and the order in which to process d incident edges at v (b).

Algorithm: CountCrossingsByOpenEdges

Data: book drawing given by spine and distribution

Result: the number of crossings in the book drawing

```
1  $L \leftarrow$ new lists
2  count $\leftarrow 0$
3  **foreach** *vertex $v$ on spine from position 1 to $n$* **do**
4      sort edges at $v$
5      **foreach** *edge $e$ at $v$ in sorted order* **do**
6          **if** *$e$ ends at $v$* **then**
7              **foreach** *edge $e'$ backwards through $L$* **do**
8                  **if** $e = e'$ **then**
9                      remove $e$ from $L$
10                     stop going through $L$
11                 **else**
12                     count++
13         **else**
14             append $e$ to $L$
15 **return** *count*
```

Algorithm 1: Algorithm to count crossings by counting open edges during a sweep.

Sorting the edges (line 5) to process them in the right order can be done with an adopted radix sort algorithm in $\mathcal{O}(\deg(v))$, which in total takes $\mathcal{O}(\sum_{i=1}^{n} \deg(v_i)) = \mathcal{O}(m)$ time. Going through the open edges lists sums up to the number of crossings C. Thus, in total the algorithm runs in $\mathcal{O}(m + C)$ time and $\mathcal{O}(m)$ space.

The upside of this algorithm is that its asymptotic running time is linear in the number of edges and depends on the number of crossings C. However, as mentioned before, this can also be its downside as C can be quadratic in m. The next approach does not depend on C and gives rise to a whole set of algorithms.

2.4.3. Transformation to two-layer cross counting

We transform the problem of counting crossings in a book drawing to counting crossing in a two-layer drawing for which several algorithms have been proposed in the literature [BJM02, NY04]. As before, we explain the algorithm for one page. With more pages we count the number of crossings for each page separately and sum them up.

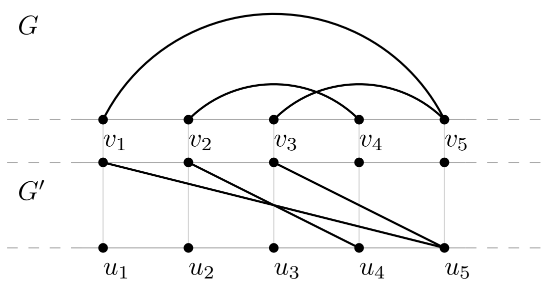


Figure 2.13.: An embedded graph G and its corresponding two-layer graph G', illustrating Observation 2.23. The edges v_1v_5 and v_2v_4 are nested in G and v_1u_5 and v_2u_4 are crossing in G'.

The reduction works as follows and is, like the algorithm above, based on Observation 2.22. For each edge we count the number of edges starting between its two endpoints. As shown in Algorithm 2 (line 4 - 12), this can be done in linear time. However, this way, we overcount the number of crossings by the number of nested edge pairs, i.e. edge pairs xy, uv where $x < u < v < y$ or $u < x < y < v$, respectively.

We define a two-layered drawing $G' = (V, U, E')$ based on G and its book drawing. The vertex sets V equals $V(G)$ and U is a copy of $V(G)$, both with the ordering given by the spine of the drawing. Each edge in $v_iv_j \in E(G)$ gives rise to an edge v_iu_j in E' with $v_i \in V, u_j \in U$, formally, $E' := \{v_iu_j | v_iv_j \in E(G)\}$. Figure 2.13 illustrates this reduction and the following observation.

Observation 2.23. *Two edges v_iv_j and v_lv_k of G are nested if and only if the two corresponding edges v_iu_j and v_ku_k of G' cross in the described two-layered drawing.*

Thus, we conclude that if we subtract the number of crossings in G' from our previous count, we get the desired number of crossings in the book drawing of G. For the two-layered cross counting we use the *divide & conquer* algorithm proposed by Bach et al. [BJM02], which counts the crossings in a merge-like fashion and runs in $\mathcal{O}(m \log n)$ time and $\mathcal{O}(m)$ space. Since the first counting and the reduction can be done in linear time, $\mathcal{O}(m \log n)$ is also the asymptotic runtime of our whole algorithm. Among several other algorithms for the two-layer cross count the one by Nagamochi and Yamada [NY04] running in $\mathcal{O}(n^2)$ time would be of interest for dense graphs.

2.4.4. Evaluation

We have seen three different approaches to count crossings, all with different running times. Checking edges pairwise runs in $\mathcal{O}(m^2)$, the open edges sweep runs in $\mathcal{O}(m + C)$ and the divide & conquer approach for two layer cross count runs in $\mathcal{O}(m \log n)$. As already mentioned, the downside of the open edges sweep is that C can be in $\Omega(m^2)$. We could easily evaluate with the following experiment that this does in fact matter.

We tested the three algorithms for planar graphs on one page and complete balanced bipartite graphs on five pages with the number of vertices n growing in steps of ten. For each n we used 100 randomly generated different planar graphs and the balanced complete bipartite graph, respectively, and ran heuristics to compute a vertex order and an edge distribution. Then we calculated the crossings with all three algorithms for each graph 100 times. The results are presented in Figure 2.14 along with the average number of crossings (curve smoothed).

```
Algorithm: CountCrossingsBy2LayerCrossCount
Data: book drawing given by spine and distribution
Result: the number of crossings in the book drawing
1  edgesStartingBefore ← new int[n]
2  edgesEndingAt ← new int[n]
3  count ← 0
4  foreach edge e do
5      edgesStartingBefore[e.start]++
6      edgesEndingAt[e.end]++
7  for i from 2 to n do
8      edgesStartingBefore[i] += edgesStartingBefore[i − 1]
9  foreach edge e do
10     count + = edgesStartingBefore[e.end −1] − edgesStartingBefore[e.start]
11     edgesEndingAt[e.end]−−
12     count − = edgesEndingAt[e.end]
13 count − = number of crossings in G′
14 return count
```

Algorithm 2: Algorithm to count crossings by overcounting open edges and subtracting crossings in the two-layer graph G'.

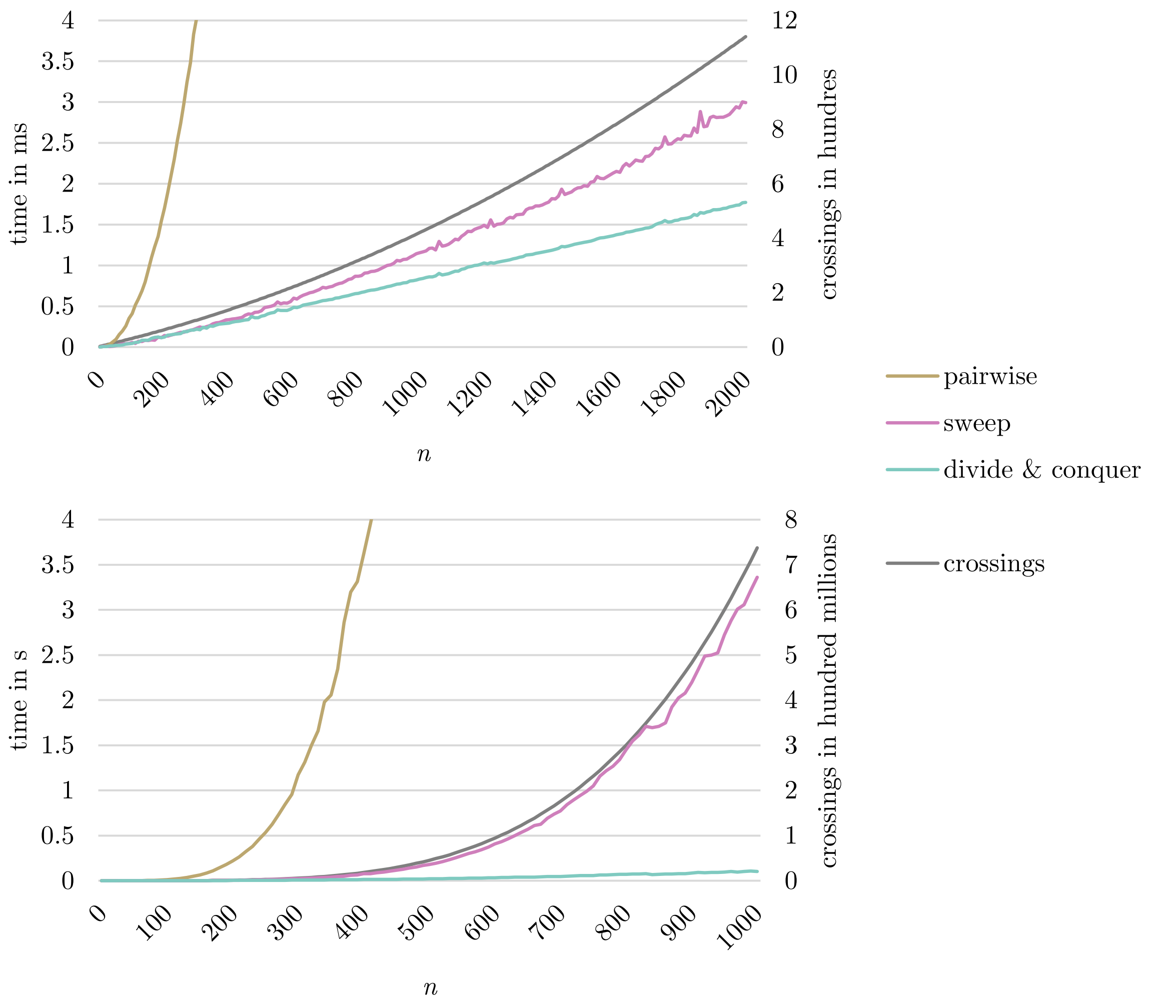


Figure 2.14.: The three cross counting algorithms on 1-page book drawings of planar graphs (top) as well as on 5-page book drawings of complete bipartite graphs (bottom).

We can observe that for planar graphs with few crossings the needed time was marginal and differences could be due to the implementations. Nevertheless, we already see that, not surprisingly, the curve of the pairwise edge comparison grows way faster. However, considering the diagram for complete bipartite graphs with growing n and hundred millions of crossings, also the difference between the open edges sweep and divide & conquer algorithm gets clear. The open edges sweep grew with the number of crossings and at some points needed seconds, while the divide & conquer algorithm stayed in the milliseconds.

Hence, not surprisingly our choice is the divide & conquer algorithm after a reduction to the two-layer cross count. We further want to note that for some algorithms it is possible to compute the number of crossings alongside the drawing. Examples are the vertex order heuristic by Bauer and Brandes [BB05] (Section 3.1.4) and the greedy edge distribution heuristics (Section 3.2.1). However, since this is not always possible, we count the number of crossings in our experiments after the drawings are computed.

3. Heuristic approaches

As we have seen, the k-page crossing minimisation problem is $\mathcal{NP}$-hard in general and solving it exactly has its limitation. Only for some graph classes are exact algorithms known, however, mostly only for k equal to one, two or their pagenumber. Consequently, for arbitrary graphs, *heuristics* are needed to compute book drawings. By heuristics, we mean algorithms that use simple strategies to compute a vertex order or edge distribution only once and do not iterate or optimise previous results. Optimisation algorithms are subject of Chapter 5.

In Section 2.3 we have seen that the SAT based solution approaches transform the problems of finding a vertex order and an edge distribution into a single problem. However, most heuristics proposed in the literature tackle either the vertex order or the edge distribution. Hence, this chapter is also split into several parts. First, in Section 3.1 we discuss vertex order heuristics, second, in Section 3.2 edge distribution heuristics and at the end in Section 3.3 possible combinations. We also introduce new heuristics in each of these categories. Furthermore, for each presented heuristic we discuss whether and why we implemented and tested them (or why not). This is mostly due to the fact that they either performed well in previous experiments or are newly proposed. Afterwards, in Chapter 4, we will evaluate the heuristics on graphs of different densities and graph classes.

3.1. Vertex order heuristics

When computing vertex order and edge distribution separately, the vertex order should be created first. Otherwise it is unclear how to decide, if two edges should be on different pages. In fact, all edge distribution heuristics we have found in the literature and those presented in Section 3.2 make use of a previously computed vertex order, while, in turn, no vertex order heuristic uses an edge distribution.

A vertex order heuristic tries to find a good circular order for the vertices of the given graph. It thereby handles vertex after vertex. This means it first has to select the next vertex to add and then insert it into the current order. However, it is not clear what a good circular order is. One approach is to say that a vertex order is good if the number of crossings is small when all edges lie in the same page. Another approach is placing long chains on the spine. This is encouraged by the fact that book embeddings of maximal outerplanar and Hamiltonian planar graphs use Hamiltonian cycles for the vertex order.

3.1.1. Depth-first search

One approach to compute a vertex order is to simply place the vertices on the spine in the order they are visited during a depth-first search (DFS). In the literature two different strategies are proposed to decide where to start and which neighbour to visit next. Figure 3.1 illustrates both of them.

random DFS (`randDFS`) The start vertex and the order of the neighbours are chosen at random. Hence this algorithm by Bansal et al. [BSV+08] has running time $\mathcal{O}(m+n)$.

smallest degree DFS (`smlDgrDFS`) Aiming for a decrease in total edge length, He and Sýkora [HS04] proposed placing the vertex with smallest degree first and then proceed with the neighbour with smallest degree. Checking the degree of all neighbours at each step sums up to $\mathcal{O}(m)$. Hence this algorithm has also asymptotic running time $\mathcal{O}(m+n)$. This algorithm is often called *AVSDF*.

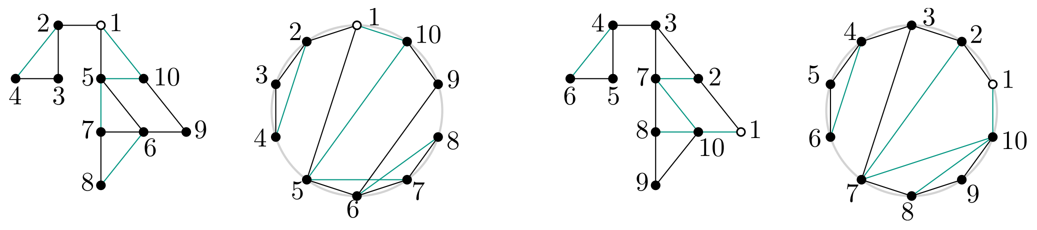


Figure 3.1.: On the left, the `randDFS` algorithm on a graph and the resulting order for the book drawing, and on the right `smlDgrDFS` on the same graph. The white vertex is the chosen start vertex and the black edges are forward edges of the respective DFS.

We want to mention that we have not found a direct comparison of their performance in the literature. However, we want to note that the algorithms are the same on regular graphs.

Furthermore, `randDFS` is also used in several evolutionary approaches to compute the initial population and as *intermediate DFS* [SSS13, BSV+08, SSG11]. There, a part of an existing vertex order gets fixed and the search is only run on the non-fixed vertices.

A third algorithm, which uses DFS, is due to Six and Tolis [ST06]. They propose a DFS algorithm on an reduced edge set to find a long path when constructing 1-page circular drawings. However, since the algorithm was outperformed by `smlDgrDFS` in experiments by He and Sýkora [HS04], and due to its complexity, we exclude it from our experiments.

3.1.2. Breath-first search

Similar to the DFS based heuristics, algorithms can compute a vertex order based on a breath-first search (BFS). While the two DFS algorithms above differ in their search, the two BFS algorithms below create two different orders from the same search. Figure 3.2 illustrates both of them.

random BFS (`randBFS`) As in the DFS algorithms, this algorithm sets the vertex order as the order in which it visits the vertices in the BFS. Similarly, it picks the start vertex and the order in which it processes the neighbours randomly [SSS13].

tree BFS (`treeBFS`) A breath-first search generates a spanning tree on the traversed graph. Our algorithm `treeBFS` embeds this tree crossing-free in a 1-page book (see Section 2.1.1) and then uses the resulting vertex order for the whole graph.

randBFS works similar as randDFS, however, as we can see in Figure 3.2, the algorithms clearly tends to produce many crossings. The edges between each level cross each other in the worst possible way. Hence, we exclude the heuristic from the experiments in the next chapter due its bad performance.

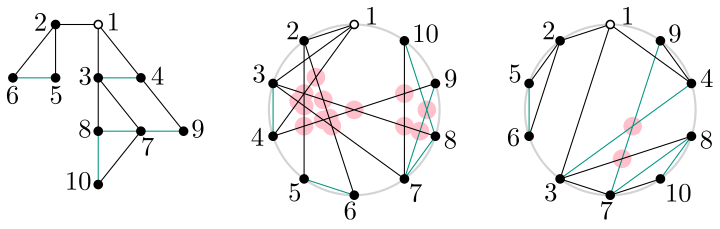


Figure 3.2.: The graph from Figure 3.1 with computed BFS tree in black on the left. The resulting book drawing for randBFS in the middle and for treeBFS on the right. The white vertex indicates the start vertex of the search.

3.1.3. Max-neighboring

The maxNbr algorithm processes groups of vertices. More precisely it picks the vertex with the highest degree, say v, and places it at the end of the spine. Next, it places behind v all neighbours of v in increasing order of their degree. Then it removes both v and its neighbours from the graph and starts again. It stops when all vertices are placed. In other words, the algorithm covers the vertices of the graph with stars and places the stars behind each other on the spine. Satsangi et al. [SSS13] used this algorithm, but sorted the neighbours by decreasing degree. It can be implemented to run in $\mathcal{O}(m+n)$ time using a bucket priority queue. Figure 3.3 illustrates this algorithm. We can observe that this algorithm has the same problems as randDFS. Hence, we also exclude maxNbr from the experiments due to this problem and its bad performance.

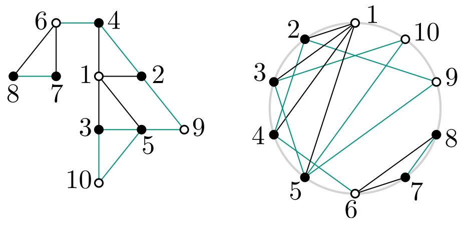


Figure 3.3.: The maxNbr heuristic, where white vertices are those picked by degree. The black vertices are ordered after the white vertex they are adjacent to via a black edge.

Satsangi et al. [SSS13] also proposed to place the vertices in an order found by a vertex cover approximation algorithm, which chooses vertices by highest degree. However, they also show that the resulting book drawings are mostly even worse than drawings with random vertex order or computed by maxNbr.

3.1.4. Connectivity

The following algorithm, introduced by Baur and Brandes [BB05], greedily chooses an unplaced vertex and places it at one end of the spine, similar to an algorithm of Mäkinen [Mä88]. They proposed several strategies for selecting the next vertex to place and

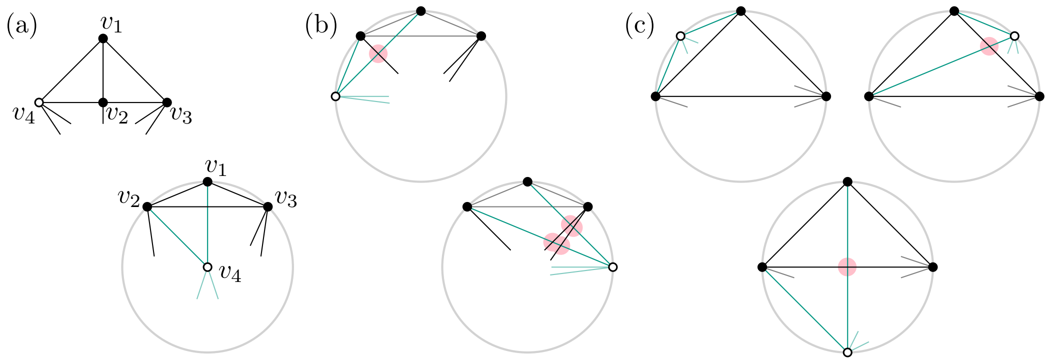


Figure 3.4.: In (a) the connectivity strategy is illustrated, choosing vertices first by most placed neighbours and then fewer unplaced ones. In (b) and (c) the options for `conCro` and `conGreedy` to place the vertices are shown. For both algorithms the upper left option would be chosen.

for choosing the end to place the vertex. However, we only use their most successful combination, which works as follows.

The algorithm selects the vertex with most placed neighbours and breaks ties in favor of vertices with fewer unplaced neighbours. In order words, the chosen vertex closes most open edges and opens fewest at ties. This is illustrated in Figure 3.4 (a). The end to place the vertex is picked in favor of the one producing fewer new crossings, as shown in Figure 3.4 (b). We refer to the algorithm with `conCro`, since the algorithm first selects based on connectivity and then based on crossings. It can be implemented to run in $\mathcal{O}((m+n)\log n)$ time [BB05].

The limitation to place vertices only at the beginning or at the end of spine is good for the running time, but might not be so good for the number of crossings. Hence, we propose Algorithm 3, which uses the idea of choosing vertices to place by connectivity as well. However, it places the vertex greedily between any two other already placed vertices, such that the number of new crossings is minimal. This strategy is illustrated in Figure 3.4 (c). We call the algorithm `conGreedy`, since it picks vertices based on connectivity and places them greedily. Its running time is in $\mathcal{O}(m^2n)$, since it has to compare all new edges with those already closed at each position.

```
Algorithm: conGreedy
Data: graph $G$
Result: vertex order
1 while not all vertices placed do
2     vertex $v \leftarrow$ chose next vertex based on connectivity
3     foreach open edge $uv$, $u$ already placed do
4         foreach closed edge $xy$, $x, y \neq u$ do
5             if $x < u < y$ then
6                 mark positions between $y$ and $x$ as bad for $v$
7             else
8                 mark positions between $x$ and $y$ as bad for $v$
9     place $v$ at position with fewest marks
```

Algorithm 3: Algorithm `conGreedy`.

We note that a new crossing is formed differently in `conCro` and `conGreedy`. In the former a new crossing is produced by a new closed edge and an open edges, whereas in the latter it is produced by a new closed edge and an old closed edge. We recall that an edge is open if only one of its incident vertices is already placed, and accordingly closed when both are already placed.

In total we have six vertex order heuristics that we will consider in the experiments. First, we have the three search based heuristics, namely, `randDFS` and `smlDgrDFS` using DFS and `treeBFS` using BFS. Second, we have the two connectivity based heuristics `conCro` and `conGreedy`. Furthermore, in Section 3.3 we introduce with a variation of `conGreedy`, namely `conGreedy+`, another vertex order heuristic.

3.2. Edge distribution heuristics

The task of an edge distribution heuristic is to place each edge on a single page with the goal of keeping the number of crossings low. The heuristics presented here use a previously computed vertex order, in contrast to the vertex order heuristics above that do not use a given edge distribution.

3.2.1. Greedy

There are several heuristics sharing a general framework. They all first compute an edge order according to some strategy. Then they process the edges in this order and place each of them on the page where the increase in crossings is minimal.

We describe three strategies to find an edge order. The first two were both motivated by the idea to place long edges first. However, only the second, namely `ceilFloor`, does this correctly. The third strategy uses the result that complete graphs can be embedded in $\lceil \frac{n}{2} \rceil$ pages (see Theorem 4.11).

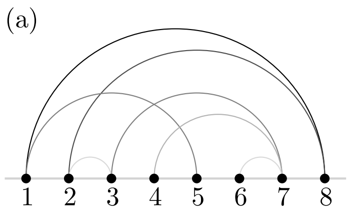
length *(`eLen`)* In this strategy by Cimikowski [Cim02] and Satsangi et al. [SSS13] the edges are ordered non-increasingly by the distance of their end vertices in a linear order (not a circular order). This can be seen as the length of the edge in a linear spine. Thus edge $(1, n)$ is listed first and any edge $(i, i+1)$ last. This is illustrated in Figure 3.5 (a). However, since a book drawing has actually a circular order, the edges with such linear length i and $n-i$ have the same possibilities of producing crossings. This is achieved by the next heuristic.

ceil-floor *(`ceilFloor`)* In this strategy the edges are ordered non-increasingly by their actual length, which is the distance of the end vertices in the vertex order (a circular order), as illustrated in Figure 3.5 (b). Kapoor et al. [KRSZ02] used this strategy when searching for pagenumbers and they achieve what `eLen` aims for, namely placing the edges with highest probability of producing crossings first. We reuse the name `ceilFloor` from Satsangi et al. [SSS13].

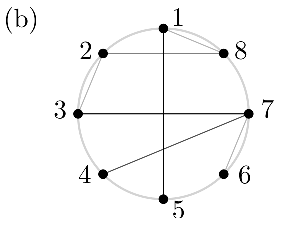
circular *(`circ`)* The vertex order achieving zero crossings for complete graphs on $\lceil \frac{n}{2} \rceil$ pages is to start at vertex i and then go to $i+1, i-1, i+2, \ldots, i+\lceil \frac{n}{2} \rceil$, iterating i from 1 to $\lceil \frac{n}{2} \rceil$. For each i this gives a path that can be placed on the same page. The strategy by Satsangi et al. [SSS13] orders the edges of a graph as they appear in this sequence.

Finding the order takes $\mathcal{O}(m \log m)$ time for `eLen` and `ceilFloor`, but $\mathcal{O}(n^4)$ time for `circ`. The second step greedily distributing the edges takes $\mathcal{O}(m^2)$ time for all the heuristics. Hence their running time in total is $\mathcal{O}(m^2)$ and $\mathcal{O}(n^4)$ for `eLen`, `ceilFloor` and `circ`, respectively.

There are several other greedy approaches by Cimikowski [Cim02], He et al. [HSSV06] and Satsangi et al. [SSS13]. However, since they are either only slight variations or were



`eLen`:
(1, 8)
(2, 8)
(1, 4), (3, 7)
(4, 7)
(2, 3), (6, 7)



`ceilFloor`:
(1, 5), (2, 8)
(4, 7)
(2, 8)
(1, 8), (2, 3), (6, 7)

Figure 3.5.: Illustration of the orders in which `eLen` and `ceilFloor` consider the edges. `eLen` uses the length according to a linear spine, `ceilFloor` according to a circular spine.

outperformed in their experiments, we decide to only use the three greedy strategies from above. Furthermore, Cimikowski also described a dynamic programming and a divide and conquer approach. Again however, both are more complex and were outperformed by the greedy approaches. Hence, we do not consider them.

3.2.2. Slope

The next algorithms makes use of the geometry of circular drawings. Consider two edges with the same slope for an already computed vertex order and the corresponding circular drawing where the vertices are distributed evenly. Then we can observe that their end vertices can not be alternating and hence they can not cross. If however, their slopes differ more and more, it gets also more likely that they cross. He et al. [HSV05] used this fact for two-page book drawings to determine an edge distribution. They mapped edges with a positive slope to page one and edges with negative slope to page two. Our algorithm `slope` extends this straight forward to k pages, as illustrated in Figure 3.6 for $k = 3$.

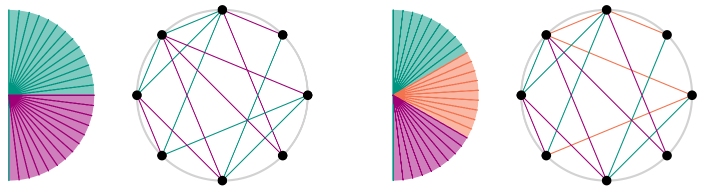


Figure 3.6.: Illustration of the `slope` heuristic for two pages on the left and for three pages on the right. The coloured parts of the half cycle show which slope gets mapped to which colour respectively page.

The slope of an edge can be computed in different ways. One approach is to compute it with trigonometric functions. However, since this approach is prone to floating point errors, we use the second approach, which uses the combinatorics to compute the slope. Considering Figure 3.7 (a), we observe that an edge can have at most n different slopes in a circular drawing with n evenly distributed vertices. This is due to the fact that any edge is either parallel to an edge starting at vertex $0 \equiv n$ or to the edge $(1, n-1)$. We therefore label the different slopes from 0 to $n-1$ according to the sum of the vertices modulo n of these edges $(1, n-1), (0, 1), \dots, (0, n-1)$, i.e. the edge $(0, i)$ defines the slope i. This is illustrated in Figure 3.7 (b).

To compute the slope of an edge (i, j) we use the following lemma.

Lemma 3.1. *Let the slopes be defined as above. The slope of an edge (i, j) in a circular drawing with evenly distributed vertices is $i + j \bmod n$.*

Proof. Without loss of generality let $i < j$. We proof the statement by showing that the edge (i, j) is parallel to the edge $(0, i + j \bmod n)$ or $(1, n - 1)$, respectively.

It is easy to see that two non-incident chords $(i, j), (k, l)$ with $i < j, k < l$ of a cycle are parallel if and only if they do not cross and if the circular arcs between the chords have the same length. This means that if $k < l < i < j$ we have $j - k = (n + l) - i$, and if $k < i < j < l$ we have $i - k = l - j$.

Assuming $i + j < n$, we are in the case $k < i < j < l$ and it follows that the edge (i, j) is parallel to $(0, j + i)$. This is also illustrated in Figure 3.7 (c). Furthermore, if $i + j > n$, which is the case $k < l < i < j$, (i, j) is parallel to $(0, j + i \bmod n)$, since $i - 0 = (j + i) - j$. Last but not least, if $i + j = n$, we see that (i, j) is parallel to $(0 + 1, n - 1) = (1, n - 1)$, since $i - 1 = (n - 1) - j = (i + j - 1) - j$. □

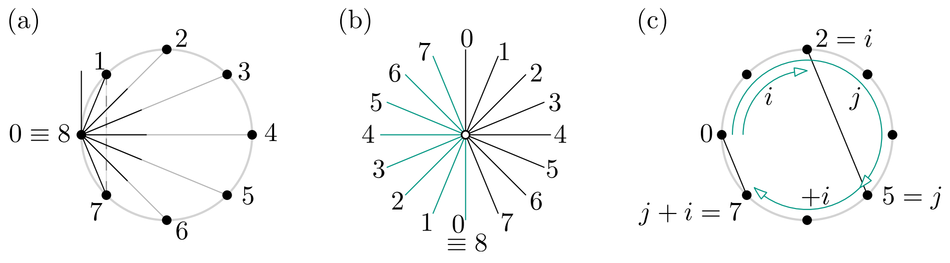


Figure 3.7.: (a) and (b): The different slopes defined by the edges $(0, i)$.
(c): Illustration why (i, j) is parallel to $(0, i + j)$.

Since the slope of each edge can be computed independently in $\mathcal{O}(1)$, the algorithm runs in $\mathcal{O}(m)$ time. Furthermore, we want to note in advance that it is actually conjectured that `slope` produces book drawings with minimal number of crossings of complete graphs for any number of pages [dKPS13]. We will discuss this more detailed in Section 4.8.

3.2.3. Ear decomposition

The algorithm `earDecomp`, which was developed during the practical course preceding this thesis, works on an *ear decomposition* of the edge conflict graph. We recall that the edge conflict graph G_c for a given graph G and vertex order is the graph with edges of G as vertices and adjacency between two conflict edges, i.e. edges that can cross for the given vertex order. An *ear* of a graph is a path where every internal vertex has degree two. An *ear decomposition* partitions the graph's edge set into a sequence of ears, such that, except from the first ear, the endpoints of every ear belong to an ear appearing earlier in the sequence and its internal vertices do not.

The algorithm constructs an ear decomposition on the edge conflict graph and simultaneously distributes the edges to pages. When its finds a new ear, it can place its internal vertices (edges of the original graph) alternating on two or three pages, as Figure 3.8 indicates, avoiding any crossings among the corresponding edges in the original graph. Consequently, `earDecomp` needs two or three pages to do this, depending on whether the start and end vertices are on the same page and the path has even or odd length. If however, an ear consists of only one edge, like edge $e = xy$ in Figure 3.8, and the start and end vertices, x and y, are already on the same page, then there is a crossing in the book drawing. However, the algorithms tries to avoid this by selecting the best page with respect to its already placed neighbours. Thus it might not always be the best to alternate.

`earDecomp` can be implemented to run in $\mathcal{O}(m^2)$ time. Constructing the edge conflict graph needs $\mathcal{O}(m^2)$ time, traversing it to find the ears runs also in $\mathcal{O}(m^2 + m)$ time and distributing the vertices of the ears also needs $\mathcal{O}(m^2)$ time. For each edge the algorithm considers at most all other edges when searching for the best page.

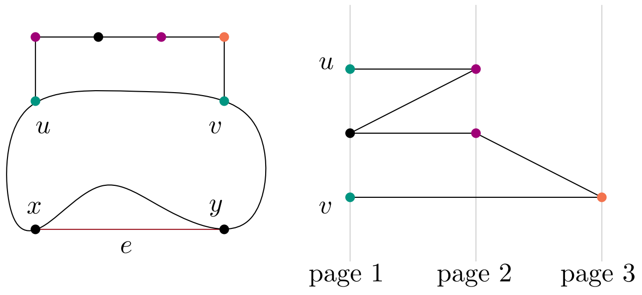


Figure 3.8.: Illustration of the `earDecomp` heuristic with conflict graph on the left and edge distribution on the right. The internal vertices of the ear from u to v are distributed to pages in alternating fashion. The ear consisting of $e = xy$ produces a crossing if x and y are on the same page.

We recall from the last chapter that a crossing-free page can contain at most one maximal outerplanar graph and thus at most $2n-3$ edges. Hence, every additional edge introduces at least one edge in the edge conflict graph, independent of the vertex order and the number of pages. Moreover, with higher density, every additional edge introduces on average more and more edges in the edge conflict graph. For example, the edge conflict graph of the complete graph K_n has $\binom{n}{4}$ edges, since any four distinct vertices introduce a conflict. We note that $\binom{n}{4}$ is quadratic in m. With higher density the average length of the ears becomes shorter and thus the idea to alternate between pages ineffectual. We can report in advance to the experiments in the next chapter that `earDecomp` performed indeed mostly worst of the edge distribution heuristics for graphs with higher density

So in total we have five edge distribution heuristics. We have three different greedy edge distribution heuristics, namely `ceilFloor`, `eLen` and `circ`. Furthermore, we have `slope` using the geometry of book drawings and `earDecomp` using an ear decomposition.

3.3. Full drawing heuristics

Having seen that book drawings can be computed in two steps by using vertex order and edge distribution heuristics, it is natural to ask whether these steps can be combined. The presented edge distribution heuristics make use of a previously computed vertex order and thus have to run after. Strictly speaking, `slope` can be used simultaneously if the vertex order heuristic only appends vertices to the currently computed vertex order like `conCro`. However, then it would not make a difference to running `slope` after the vertex order heuristic has completed its work.

Nevertheless, the concept of distributing the edges greedily to the best pages can be implemented during the computation of a vertex order. He et al. [HSSV06] did exactly this within the `smlDgrDFS` algorithm. When their algorithm visits an edge during the search the second time, it places the edge on the page where it produces fewest crossings. When it visited the edge the first time, the position of the second vertex is not known yet and thus no real decision where to place the edge could be made.

We implemented this approach for `smlDgrDFS`, `randDFS` and `randBFS`. We call these new algorithms, which now have running time $\mathcal{O}(m^2)$, `smlDgrDFS+`, `randDFS+` and `randBFS+` respectively. The second BFS based heuristic, `treeBFS`, can not be combined with concurrent greedy edge distribution, since it computes the vertex order based on the computed spanning tree and thus after the search.

Furthermore, following the same idea, we implemented `conGreedy+` based on the `conGreedy` heuristic. Pseudocode of the algorithm is given in Algorithm 4. When searching for a new position for a vertex, the algorithm also considers the pages of the already placed edges. For this purpose, it marks, like `conGreedy`, the positions where the new edges would produces crossings with the already closed edges. However, this times the marks depend on the pages of the closed edges. The best position for a new vertex is then the position that received fewest marks from the newly closed edges in Line 11. We see that finding the position for one new vertex (Algorithm 4, Line 2 to Line 12) takes $\mathcal{O}(mn\Delta)$ time. This sums up to $\mathcal{O}(m^2n)$ (or $\mathcal{O}(mn^2\Delta)$) for all vertices. After a vertex has been placed on the spine, the algorithm distributes its incident edges greedily to the best page (Lines 13 and 14). This runs in total in $\mathcal{O}(m^2)$. So overall, the asymptotic running is time $\mathcal{O}(m^2n)$.

We observe that, in contrast to the search based algorithms, the immediate edge distribution also effects the computed vertex order. Hence, `conGreedy+` can also be used as vertex order heuristic. A combination of `conCro` with greedy edge distribution would however not alter the vertex order, since `conCro` considers the open edges to find the position for the vertex, which in turn are not yet distributed to pages. Therefore we omit this combination. In Section 5.1.3 we also combine vertex order and edge distribution optimisation in one greedy algorithm.

```
Algorithm: conGreedy+
Data: graph $G$, number of pages $k$
Result: $k$-page book drawing of $G$
1  while not all vertices placed do
2      vertex $v \leftarrow$ chose next vertex based on connectivity
3      foreach open edge $uv$, $u$ already placed do
4          foreach closed edge $xy$, $x, y \neq u$ do
5              $p \leftarrow$ page of $xy$
6              if $x < u < y$ then
7                  mark positions between $y$ and $x$ as bad for $v$ for page $p$
8              else
9                  mark positions between $x$ and $y$ as bad for $v$ for page $p$
10         foreach possible position $a$ do
11             mark $a$ with the minimal number of marks it has on one page
12     place $v$ at position with fewest marks
13     foreach open edge $uv$, $u$ already placed do
14         distribute $uv$ to page where it produces fewest crossings
```

Algorithm 4: Algorithm `conGreedy+`.

4. Evaluation of heuristics

In this chapter we evaluate the heuristics described in the previous chapter. We have introduced three new vertex order heuristics, namely `treeBFS`, `conGreedy` and `conGreedy+`, and one edge distribution heuristic, `earDecomp`. Naturally we are interested how they perform against the heuristics from the literature. We have also extended some of the vertex order heuristics to distribute the edges greedily while computing the vertex order. This raises the question, whether and when this yields better book drawings. Especially the performance of `conGreedy+` is of interest, since the simultaneous edge distribution directly influences the vertex order.

It is also one of our interests to create a test suite for the evaluation of heuristics for book drawings. We therefore discuss also why we used the particular graph classes for testing. Since the heuristics from literature have mostly been proposed only for one or two pages, we are also interested in their performances for larger number of pages. Furthermore, as far as we know, these heuristics have also not been designed for particular graph classes or for graphs with certain properties. Therefore, we used graph classes that represent different graph aspects. We have chosen planar and 1-planar graphs due to their general significance in graph drawing. Cartesian products of cycles, hypercubes and t-ary d-cubes (also known as k-ary n-cubes) represent regular graphs with many automorphisms. Furthermore, we consider K-trees as well as random graphs of different densities. In addition, we also discuss for other graph classes why or why not they should be considered for the evaluation of heuristics for book drawings.

We consider the graph classes in increasing order of their density. We first look at random graphs with a linear number of edges in terms of n (Section 4.1). This is followed by graph classes that likewise have a linear number of edges. More precisely, we consider outerplanar graphs (Section 4.2), Cartesian products of cycles $C_i \times C_j$ (Section 4.3), Hamiltonian and random planar graphs (Section 4.4), 1-planar graphs (Section 4.5) and K-trees (Section 4.6). These classes have in common that their pagenumbers are bounded by constants. After that, in Section 4.7 we look at very regular graphs, more precisely hypercubes and t-ary d-cubes (also known as k-ary n-cubes) . Then, we consider graphs with high density, namely complete graphs (Section 4.8) and complete bipartite graphs (Section 4.9) and again random graphs (Section 4.10). The graph classes of the latter two groups (the cube-like and dense graphs) have in common that their pagenumber grows with respect to n.

We refer to Appendix A for more information regarding graph generation. Moreover, we describe the languages, machine and tools we used for our experiments in Appendix C.

4.1. Linear number of edges

We start our evaluation of the heuristic with experiments on random graphs with linear number of edges. More precisely, we use random graphs generated with the Erdös-Rényi model with the expected number of edges $m = an$, where a ranges from 2 to 10. We start at 2, since graphs with roughly n edges are either not connected or almost trees and thus either graphs we excluded or easy embeddable ones. We are interested whether the best heuristics can be determined knowing only the density of a graph and the number of pages. Hence, we use these random graphs in order to compare the performance of the heuristics on graphs with different densities and graphs of certain graph classes. For example, since 1-planar graphs have roughly $4n$ edges, we compare the results for 1-planar graphs with those from random graphs with $a = 4$. Furthermore, for $a = 2$ we consider outerplanar graphs (Section 4.2) and Cartesian products of cycles (Section 4.3), for $a = 3$ Hamiltonian planar and random planar graphs (Section 4.4) and for a from 5 to 10 we consider K-trees with fixed K (Section 4.6).

4.1.1. Experimental results

The settings of our experiments were the following. We tested all combinations of vertex order and edge distribution heuristics as well as all full drawing heuristics. The heuristic `conGreedy+` was used both as vertex order and full drawing heuristic. We set the number of vertices to 50, 100 and 150. For each factor a, ranging from 2 to 10 in steps of one, we generated 200 random graphs with roughly $m = an$ edges. We then ran each heuristic and combination of heuristics on these graphs with the number of pages k ranging from 1 to 20.

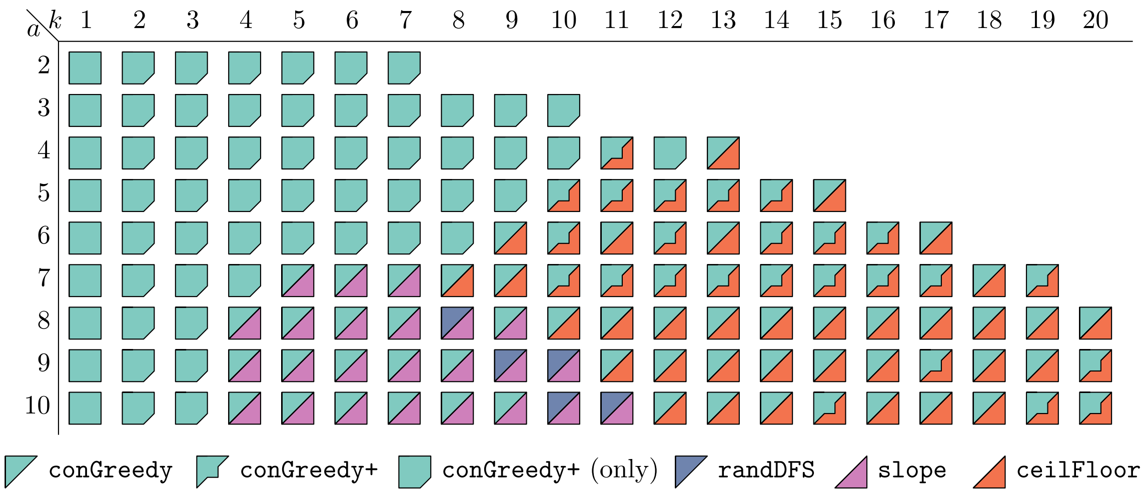


Figure 4.1.: Heuristics on random graphs with linear number of edges and 50 vertices.

Figures 4.1, 4.2 and 4.3 show the winning heuristics for each combination of a and k for 50, 100 and 150 vertices, respectively. To determine the *winning heuristic* for k and a we mark each out of the 200 graphs with the heuristic that performed best and then find which heuristic has been used as a mark most of the times over the 200 graphs. We ended a row at a k, where the average number of crossings for the best heuristic was less than 1 for the first time. The diagrams depict the results as follows. Each factor a is represented by a row and each number of pages k by a column. In each cell the best heuristic or heuristic combination is shown. In each item the upper left corner represents the vertex order heuristic and the lower right corner the edge distribution. If there is only one tile, this is due to one of two reasons. Either $k = 1$ and there is no edge distribution needed and thus only a vertex order is presented, or the tile represents a full drawing heuristic, which in these results is `conGreedy+`. Furthermore, when `conGreedy+` won in combination with a certain edge distribution heuristic, it is represented with its own shape, as depicted in the legends of Figures 4.1 to 4.3.

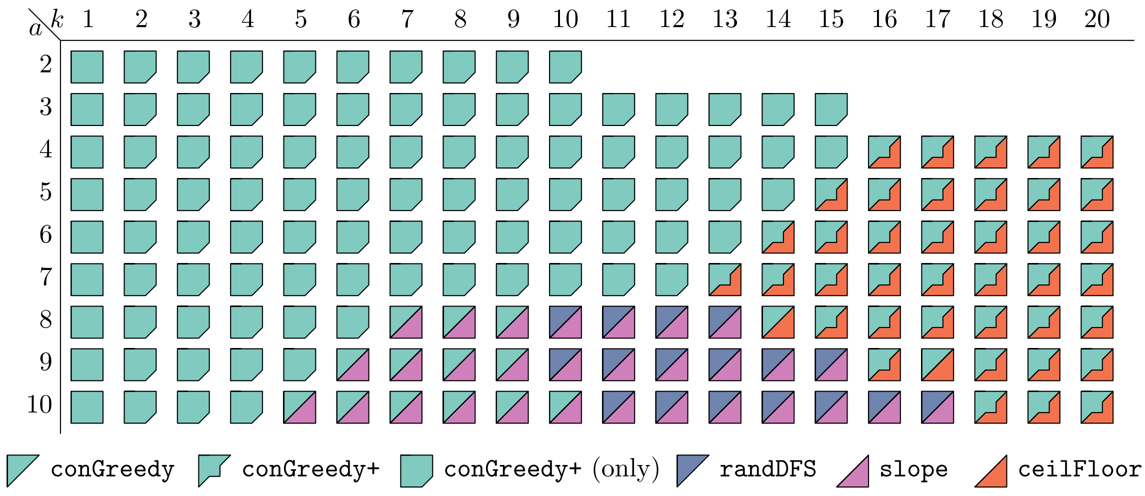


Figure 4.2.: Heuristics on random graphs with linear number of edges and 100 vertices.

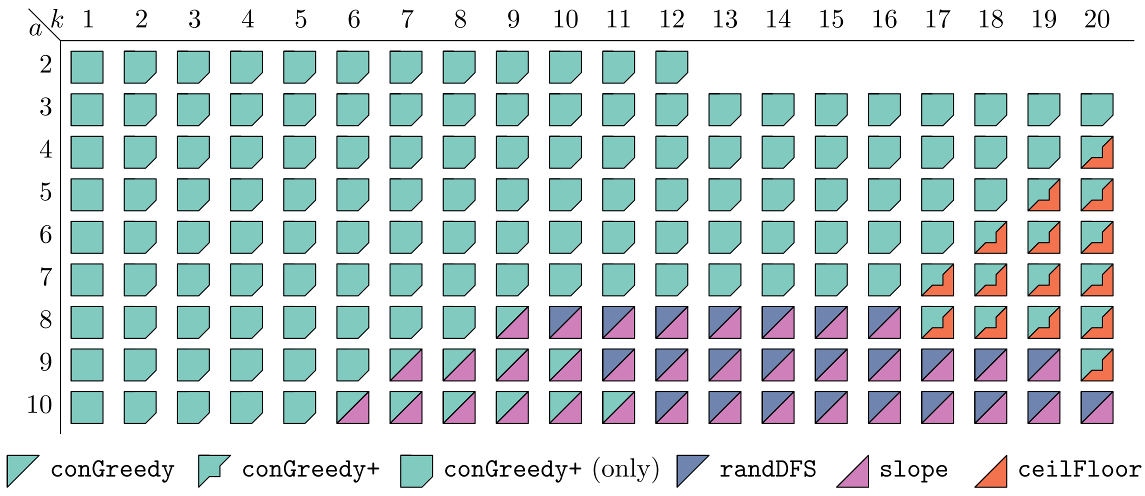


Figure 4.3.: Heuristics on random graphs with linear number of edges and 150 vertices.

The first thing we notice is that all three diagrams consist basically out of the same four regions. The predominant heuristic is `conGreedy+` used as full drawing heuristic. It was the best heuristic for small a and for small k. Furthermore, if used as vertex order heuristic in combination with `ceilFloor`, it was also comparable or better than the simple `conGreedy` vertex order heuristic for large k. We further observe that the region occupied by `conGreedy+`, `conGreedy` and `ceilFloor` shifts to larger k with the growth of the graph size. Also `conGreedy+` completely replaces `conGreedy` for $n = 150$. For larger a, the two regions at the bottom of the diagram show the edge distribution heuristic `slope`, for smaller k in combination with `conGreedy` and for larger k with `randDFS`. These regions occupy more k for larger n.

Based on these observations we can already conclude that the simultaneous computation of vertex order and edge distribution in `conGreedy+` can be an advantage. In other words, the combination of `conGreedy` with greedy edge distribution yields a well performing heuristic, both if used as full drawing heuristic or to only improve the vertex order.

4.2. Outerplanar graphs

In Section 2.1.3 we have seen that outerplanar graphs have at most $2n-3$ edges, pagenumber one and a 1-page book embedding can be derived directly from a planar embedding by using the order on the outer face. Since computing a planar embedding of an outerplanar graphs as well as making the graph maximal outerplanar can be done in linear time [Pat13], we can also construct a 1-page book embedding in linear time. Hence, outerplanar graphs are of less interest for the evaluation of heuristics for book drawings. Nevertheless, as we do in the following, it can been worth considering whether a heuristic is capable of computing 1-page book embeddings of outerplanar graphs in general or of certain subclasses of outerplanar graphs.

In contrast to the embedding of trees (Section 2.1.1), most vertex order heuristics can not guarantee to embed outerplanar graphs crossing-free in one page. In fact, only in the case where the graph is maximal outerplanar, `conGreedy` achieves always zero crossings.

Lemma 4.1. *The vertex order heuristic* `conGreedy` *computes crossing-free circular drawings of maximal outerplanar graphs.*

Proof. We recall that `conGreedy` picks a vertex with most placed neighbours and then finds the best position for it in terms of crossings. We proof inductively that during the computation of the vertex order with `conGreedy` the already placed vertices form a maximal outerplanar graph. For $n \leq 3$ this clearly holds.

Now, if we already have a maximal outerplanar graph with n vertices embedded, the question is which vertex v gets picked next and where it gets placed. We therefore consider Figure 4.4. We see that v has at least two already placed neighbours, since it will be part of a new triangle while other vertices can be adjacent to at most one vertex in the current graph. Furthermore, we see that v has indeed exactly two already placed neighbours x and y, since otherwise we would form a K_4 as minor, which is a forbidden subgraph of outerplanar graphs. Moreover, x and y are obviously consecutive in the vertex order. Hence, v can only be added in the vertex order between x and y without introducing crossings. Furthermore, it is easy to see that the now embedded graph is still maximal. □

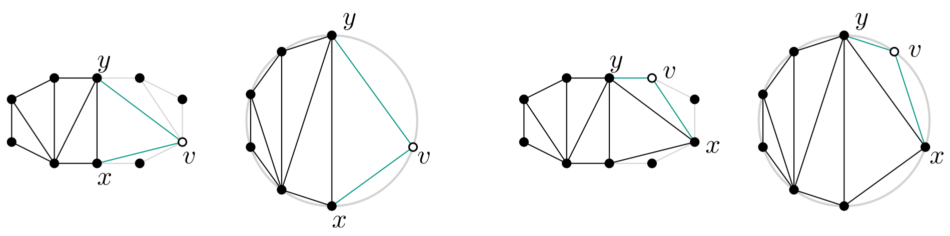


Figure 4.4.: Two steps of `conGreedy` on a maximal outerplanar graph.

The properties used in the proof also follow from the fact that maximal outerplanar graphs are 2-trees (see Section 4.6). However, for non-maximal outerplanar graphs this is no longer the case and `conGreedy` does not always output crossing-free drawings.

Comparing this to the result on random graphs with factor $a = 2$ and $k = 1$, we see that again `conGreedy` wins. Hence, we conclude that in this case the density is enough to decide for the algorithm. However, we recall that `conGreedy` has an asymptotic running time of $\mathcal{O}(m^2 n)$. Consequently, if the running time is an objective, the algorithms of choice for outerplanar graphs is the one mentioned above, which first computes a planar embedding and then derives from this a vertex order, both in linear time.

4.3. Cartesian product of cycles

The *Cartesian product graph* $G \times H$ of two graphs G and H has vertex set $V(G) \times V(H)$ and any two vertices $(u_G, u_H), (v_G, v_H) \in V(G \times H)$ are adjacent if the original vertices are adjacent in G or H, i.e. either $(u_G, v_G) \in E(G)$ or $(u_H, v_H) \in E(H)$. Furthermore, let C_i (P_i) stand for a cycle (path) with i vertices.

In this section we use Cartesian products of two cycles $C_i \times C_j$, which we call *cycle product graphs*, to test the heuristics. These graphs are also known as toroidal mesh. They have exactly $2n$ edges, which follows from the fact that a vertex in a cycle has degree 2 and thus in $C_i \times C_j$ degree 4 and the handshake lemma. He et al. [HSSV06] considered the one-page and two-page crossing numbers of $C_i \times C_j$. They also consider $P_i \times P_j$ and $C_i \times P_j$. These graphs were also subject to experiments of Satsangi et al. [SSS13] and He et al. [HSMV15].

Before we test our heuristics on the graphs $C_i \times C_j$, we proof in a series of steps that they have pagenumber at most 3 and thus settle a conjecture of Satsangi et al. [SSS13]. Before we start with a statement on their subgraphs $C_i \times P_j$, we recall that subhamiltonian is defined as being subgraph of a planar Hamiltonian graph.

Lemma 4.2 ([HSSV06])**.** *The graphs $C_i \times P_j$ are planar and subhamiltonian.*

Proof sketch. If i or $j \leq 2$ the statements clearly holds. Figure 4.5 depicts planar embeddings of $C_4 \times P_5$ and $C_5 \times P_5$ that can be extend to planar embeddings of any $C_i \times C_j$. Hence, the first part of the statement holds.

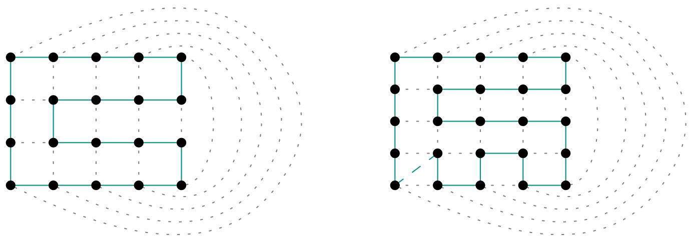


Figure 4.5.: On the left $C_4 \times P_5$ with Hamiltonian cycle and on the right $C_5 \times P_5$ with a Hamiltonian path that can be extended to a Hamiltonian cycle, showing that both graphs are planar and subhamiltonian.

Figure 4.5 also shows a Hamiltonian cycle in $C_4 \times P_5$ and a Hamiltonian path in $C_5 \times P_5$. The latter can be extended to a Hamiltonian cycle by adding the dashed edge. To construct a Hamiltonian cycle of $C_i \times P_j$ if either i or j is even, we use the strategy of $C_4 \times P_5$ in Figure 4.5. From the subgraph $P_i \times P_j$ we use, informally speaking, the first path of P_i on the left and then go in an alternating manner right and left along the paths P_j to visit all other vertices. Since i is even, this works out. If however, both i and j are odd, we also use the strategy from $C_5 \times P_5$ in Figure 4.5 for the two bottom rows. There, we alternate up and down to visit all vertices. This yields a Hamiltonian path that can be extended to Hamiltonian cycle. $\square$

We can also observe in Figure 4.5 that these Hamiltonian cycles yield 2-page book embeddings of $C_i \times P_j$. The first part of the following statement follows directly from the previous lemma, the second part from Figure 4.6.

Lemma 4.3. *The graphs $C_i \times C_j$ are subhamiltonian and if $i, j \geq 3$ non-planar.*

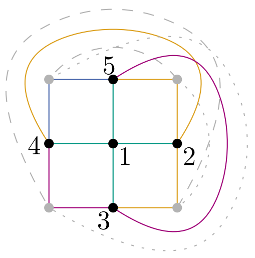


Figure 4.6.: K_5 as Minor in $C_3 \times C_3$.

Using the previous two lemmata we can now consider the pagenumber of cycle product graphs $C_i \times C_j$.

Theorem 4.4. *The cycle product graphs $C_i \times C_j$ have pagenumber at most 3 and this bound is tight if $i, j \geq 3$.*

Proof. The 3-page book embeddings of $C_i \times C_j$ are based on the following 2-page book embeddings of $C_i \times P_j$. We use the Hamiltonian cycles, or paths if both i and j are odd, from Lemma 4.2. This leaves only the edges connecting the ends of the paths P_j in the subgraph to form the cycles C_j for embeddings of $C_i \times C_j$. These edges can be embedded without crossings on page 3, since the ends of the different paths are not alternating on the Hamiltonian cycle (or path) but nested. Hence, we have a 3-page book embedding of $C_i \times C_j$. This is illustrated in Figure 4.7 for $i = 4$ and $j = 5$.

For $i, j \geq 3$ we known from Lemma 4.3 that $C_i \times C_j$ is non-planar and thus requires at least 3 pages. □

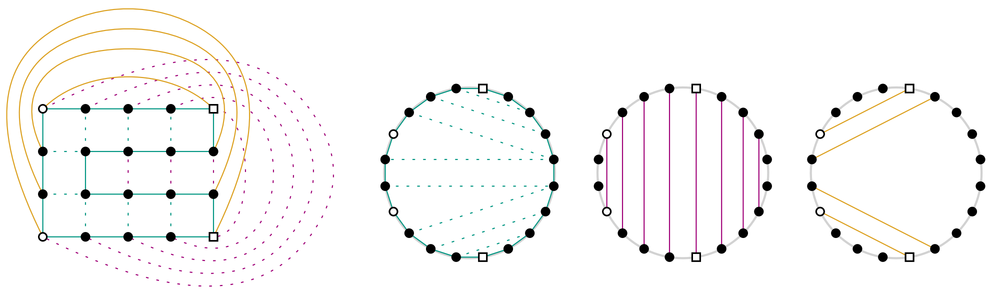


Figure 4.7.: A 3-page book embedding of $C_4 \times C_5$ based on the Hamiltonian cycle from Figure 4.5.

4.3.1. Experimental results

In contrast to random graphs, the graphs $C_i \times C_j$ are 4-regular and have many automorphisms. This raises the question, whether the ranking of the heuristics between random graphs and product cycle graphs differ. Hence, we tested the heuristics on the product cycle graphs to compare the results with those from above on random graphs with linear number of edges, where a is equal to 2.

We recall that we tested on random graphs with $n = 50, 100$ and 150. To get n to roughly 50 we used for i and j the combinations $(7, 7)$, $(6, 8)$, $(5, 10)$, $(4, 12)$, $(4, 13)$ and $(3, 17)$, for 100 we used $(10, 10)$, $(9, 11)$, $(8, 13)$, $(7, 15)$, $(6, 17)$ and $(5, 20)$ and for 150 we used $(12, 12)$, $(11, 13)$, $(10, 15)$, $(9, 17)$, $(8, 19)$, $(7, 21)$, $(6, 25)$ and $(5, 30)$. Furthermore, we ran each heuristic 50 times on each graph.

Table 4.1 shows the three best heuristics in terms of average number of crossings for this experiment and for random graphs from the experiment of Section 4.1 to ease the comparison.

First, we consider the case $k = 1$. `conCro` performed best on $C_i \times C_j$ while `conGreedy` did on random graphs. The third best heuristic on random graphs was `smlDgrDFS`. Since every vertex has degree 4 in a graph $C_i \times C_j$, `smlDgrDFS` behaves just like `randDFS` on these graphs. In addition, we report that `randDFS` was the best search based heuristic when i and j differed more, while `treeBFS` was the best when i and j differed less or were equal.

For $k = 2, 3$ `conGreedy+` was the overall best heuristic on both graph classes. However, beyond that, we observe that again `conCro` performed better than `conGreedy` on cycle product graphs. Concerning the edge distribution `earDecomp` performed best for $k = 2$, while a greedy edge distribution heuristic performed best for $k = 3$.

We conclude that for $k = 1$ it is worth to consider the structure of the graph. Concerning $k = 2, 3$ the best choices coincide between both graph classes and hence the density seems enough to decide. If we consider however the second or third best heuristics, we observe that different vertex order heuristics should be chosen.

k	random	$C_i \times C_j$
1	`conGreedy`	`conCro`
	`conCro`	`conGreedy`
	`smlDgrDFS`	`randDFS`, `treeBFS`
2	`conGreedy+`	`conGreedy+`
	`conGreedy-earDecomp`	`conCro-earDecomp`
	`conGreedy-eLen`	`conCro-ceilFloor`
3	`conGreedy+`	`conGreedy+`
	`conGreedy-eLen`	`conCro-ceilFloor`
	`conGreedy-earDecomp`	`conCro-eLen`

Table 4.1.: Best heuristics on random graphs with m roughly $2n$ in experiment of Section 4.1 and on Cartesian product of cycles.

4.4. Planar graphs

In this section we consider planar graphs. We recall from Theorem 2.10 that a graph has a 2-page book embedding if only if it is a subgraph of a Hamiltonian planar graph. However, not all planar graphs are subhamiltonian and not all maximal planar graphs are Hamiltonian. Figure 4.8 shows the smallest non-Hamiltonian maximal planar graph, which has 11 vertices. Hence we distinguish between Hamiltonian planar graphs and others.

The Hamiltonian planar graphs contain large classes of planar graphs. For example, planar graphs without separating triangles are subhamiltonian and they can be extended to maximal planar graphs keeping this property [KO07]. Is is further known that all bipartite

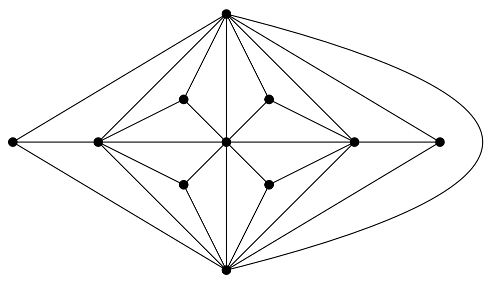


Figure 4.8.: The Goldner Harary graph has 11 vertices and is the smallest non-hamiltonian planar graph.

planar graphs and all triangle-free planar graphs are subhamiltonian [Ove98]. Furthermore, all 4-connected planar graphs [Ove07] and all planar graphs with maximum degree three [Hea85] or even four [BGR14] admit a 2-page book embedding. In the latter case they can be constructed in $\mathcal{O}(n^2)$ time. Moreover, series-parallel graphs, X-trees [CLR87] and Halin graphs [Gan95] are 2-page embeddable.

The maximal pagenumber of planar graphs is still unknown. The best known upper bound is due to Yannakakis.

Theorem 4.5 ([Yan86])**.** *Every planar graph has pagenumber at most 4.*

Yannakakis claimed that this bound is tight. However, neither a proof nor a counterexample, showing that three pages are not enough, is known [BKZ15]. For some subclasses of planar graphs it is known that they are 3-page embeddable, like planar 3-trees (Apollonian networks, stellations of K_3) [Hea84].

We ran several experiments to evaluate the heuristics on maximal planar graphs. In the first two experiments we tested vertex order heuristics and edge distribution heuristics separately on Hamiltonian maximal planar graphs. Then we tested their combinations on both Hamiltonian and random maximal planar graphs.

4.4.1. Finding Hamiltonian cycle

We start with a fact about 2-page book embeddings of Hamiltonian maximal planar graphs.

Lemma 4.6. *The vertex order in a 2-page book embedding of a Hamiltonian maximal planar graph G is always a Hamiltonian cycle of G.*

Proof. Let G be a Hamiltonian maximal planar graph. Since G is maximal, we know that it has a unique planar embedding (up to combinatorial equivalence) [Pat13], and, furthermore, every face of this embedding is a triangle. We now consider a 2-page book embedding of G, i.e. a 2-page book drawing without crossings. Since this book embedding is also a planar embedding, we know that again each face is again a triangle.

We assume now, for the sake of contradiction, that two vertices u, v that are consecutive in the vertex order are not adjacent. Thus, since every face of the 2-page book embedding is a triangle, it follows that there must be an edge e separating u and v in the drawing topologically. However, since no edge is allowed to cross the spine, no such edge e can exist. □

We are therefore interested in how successful the vertex order heuristics can find Hamiltonian cycles. For this purpose, we generated Hamiltonian maximal planar graphs of different sizes and counted how often the vertex order heuristics computed a vertex order that admits a crossing-free edge distribution on two pages, i.e. a Hamiltonian cycle. For each graph size we used 2000 graphs. A description of how we generated the graphs can be found in Appendix A.4. The resulting success rates are shown in Figure 4.9.

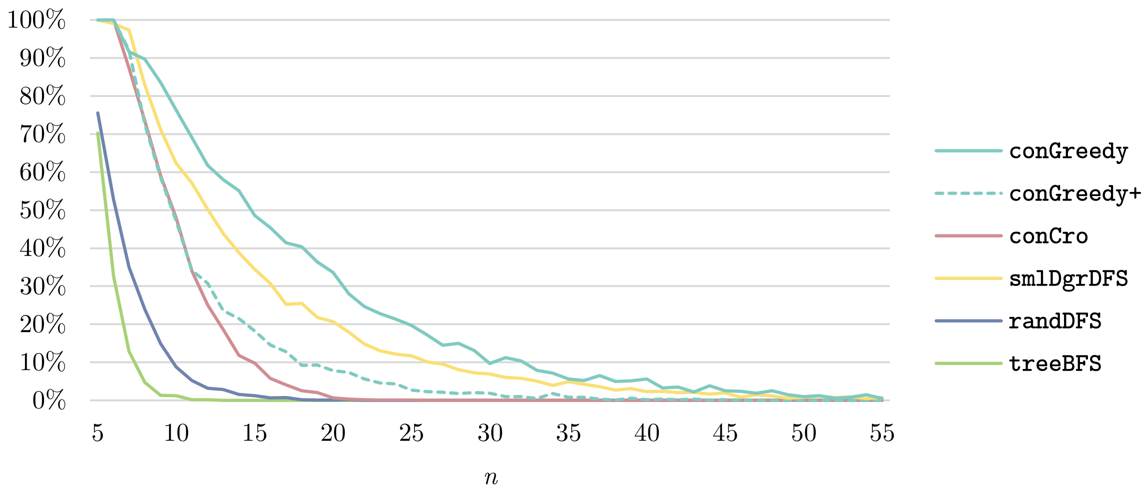


Figure 4.9.: Success rates of the vertex order heuristics to find a vertex order for Hamiltonian maximal planar graphs that admits a crossing-free 2-page book embedding.

Since the heuristics were not designed to find Hamiltonian cycles, it is not surprising to see that they were indeed not very successful. Even the two best performing heuristics `conGreedy` and `smlDgrDFS` found a Hamiltonian cycle for less than 1% of the graphs with $n = 55$.

4.4.2. Distributing perfectly

As counterpart to the previous experiment, we also ran the following one to test the edge distribution heuristics. We used again 2000 Hamiltonian maximal planar graphs with different sizes. This time we set a Hamiltonian cycle as vertex order and then counted how often the heuristics were able to distribute the edges to two pages without crossings. Figure 4.10 shows the resulting success rates.

We observe that, not surprising, `slope` performed worst on this task. The three greedy heuristics `ceilFloor`, `eLen` and `circ` were unsuccessful for graphs with more than 40 vertices. However, `earDecomp` found for more than 10% of the graphs with 50 and more vertices still a perfect distribution. We report that the heuristics success rate dropped below 1% not until n reached 120. Then again, recalling from Section 2.3.1 that we can find complete book embeddings for planar graphs with several hundred vertices in reasonable time using a SAT solver, we can conclude that the heuristics are not very successful when it comes to finding book embeddings of Hamiltonian planar graphs. So in order to find out which combination to use when the planar graphs are to large for the SAT solving approach, we look at the average number of crossings they achieve next.

4.4.3. Average number of crossings

We tested the performance of the heuristics in terms of crossings for Hamiltonian planar graphs and random maximal planar graphs for one, two and three pages. Concerning the case $k = 1$, we report that `conGreedy` performed best for Hamiltonian and random

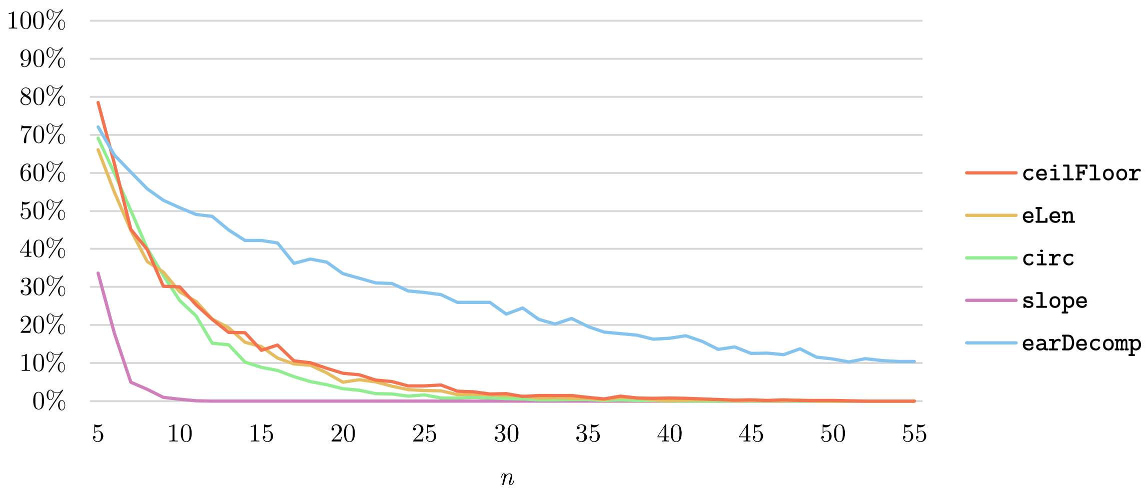


Figure 4.10.: Success rates of the edge distribution heuristics to distribute the edges without crossing on two pages given a maximal planar graph with a Hamiltonian cycle used as vertex order.

maximal planar graphs in our experiment, where we used the same settings as in the random graphs experiment.

For $k = 2, 3$, in comparison to two the experiments above, we used the combinations of the vertex order and edge distribution heuristics as well as the full drawing heuristics. For each n we tested with 200 different graphs.

First, we look at the results on the Hamiltonian maximal planar graphs. Bearing in mind the results from the previous two experiments, we expect to see the combination `conGreedy` and `earDecomp` to perform best. This is indeed the case and their combinations performs equally as good as `conGreedy+` with `earDecomp`. In fact, `earDecomp` was the best edge distribution heuristic for all vertex order heuristics. Figure 4.11 shows the results of `earDecomp` with all vertex order heuristics as well as the two best full drawing heuristics.

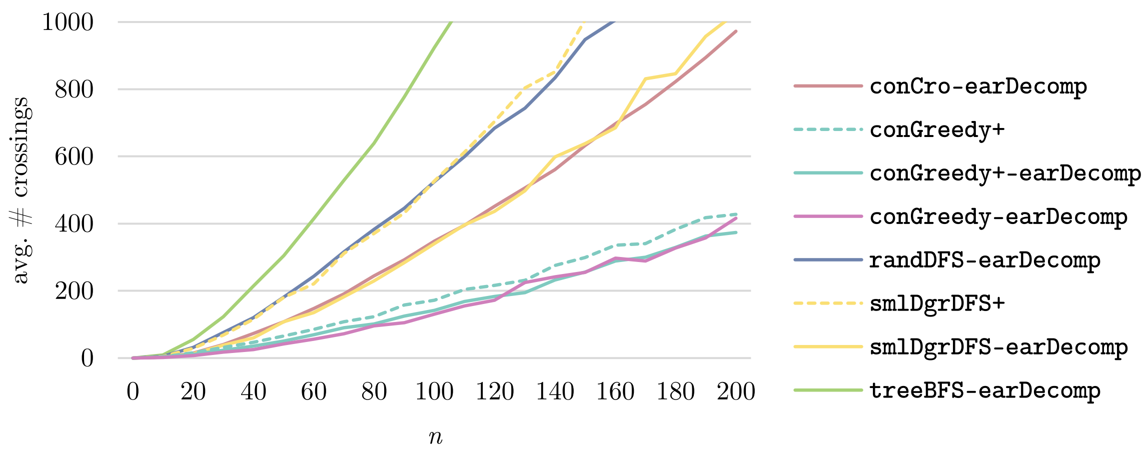


Figure 4.11.: Average number of crossings achieved by selection of heuristic combinations for Hamiltonian maximal planar graphs and two pages.

Figure 4.12 shows the results for $k = 2$ and random maximal planar graphs. More precisely, it shows the best average number of crossings achieved by the different vertex order

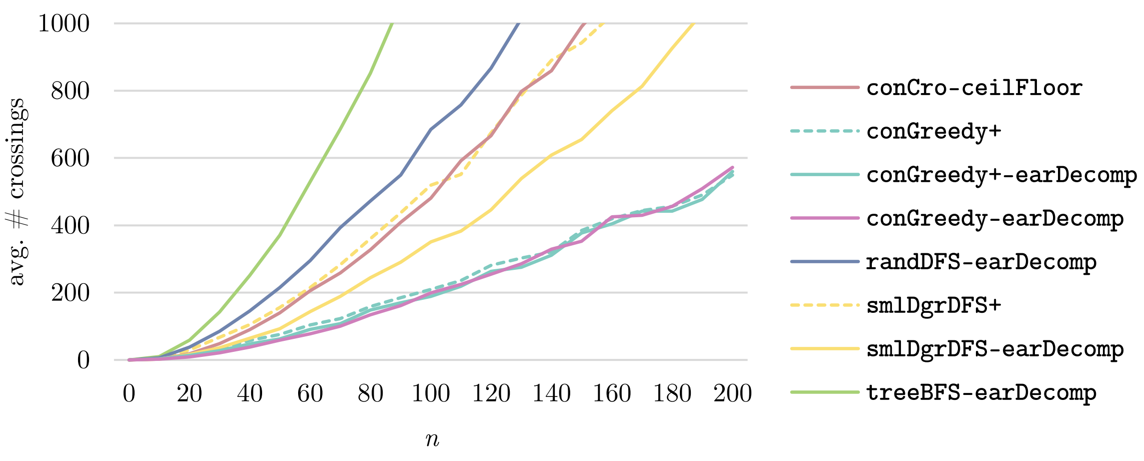


Figure 4.12.: Average number of crossings achieved by selection of heuristic combinations for maximal planar graphs and two pages.

heuristics in combination with an edge distribution heuristic as well as the two best full drawing heuristics. We observe that again `earDecomp` was often the best edge distribution heuristic, however, `conCro` performed better with `ceilFloor`. Furthermore, we observe that the average number of crossings achieved on Hamiltonian maximal planar graphs are lower than on random maximal planar graphs for all shown heuristics.

In addition to the diagrams, we report that `conGreedy` and `conGreedy+` performed also better in combination with the other greedy edge distribution heuristics (and solely) than `conCro` and all searched based vertex order heuristics. Hence, we can conclude that, if `slope` is not used, the choice of the vertex order heuristic is more essential for Hamiltonian maximal planar graphs than the choice of the edge distribution heuristic. We will see this again in the next section on 1-planar graphs.

Next, we look at the the results on random planar graphs and three pages. We do not consider Hamiltonian planar graphs, since they have pagenumber 2. Figure 4.13 shows again for the best combinations of each vertex order heuristic with an edge distribution heuristic. This time, however, we divided the average number of crossings by the number of edges, i.e. by $3n - 6$. We want to remark that the ranking of the performances is the same as if the diagram would depict the total number of crossings. We can observe that similar to the Cartesian product of cycles `earDecomp` is not the best choice for $k = 3$. This time `conGreedy` with `ceilFloor` performed best, followed by `conGreedy+`. In contrast to Hamiltonian planar graphs, `conGreedy+` performed better alone than with an edge distribution heuristic. We can also observe that the average number of crossings per edge grows very slowly for `conGreedy-ceilFloor`. This indicates that the crossings of the heuristics are more local problems than global ones, which is, considering how `conGreedy` works, not surprising but nevertheless pleasing.

We can observe in all three diagrams, Figures 4.11 to 4.13, that `smlDgrDFS` in combination with an edge distribution heuristics worked better than `smlDgrDFS+`. In fact, `smlDgrDFS` performed better with `earDecomp` and all three greedy edge distribution heuristics `ceil-Floor`, `eLen` and `circ`. This was also the case for `randDFS` and `randBFS`. However, we excluded both from the diagrams due to their too bad performance. Nevertheless, we can conclude that the order in which the greedy edge distribution heuristics consider the edges works better than the order in which the search based vertex order heuristics visit the edges, at least on maximal planar graphs.

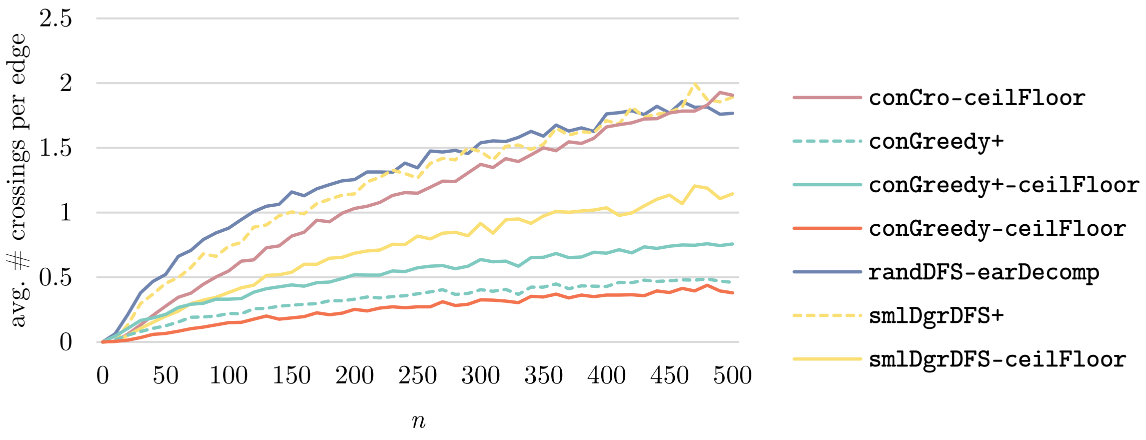


Figure 4.13.: Average number of crossings achieved by selection of heuristic combinations for random maximal planar graphs and three pages.

4.5. 1-planar graphs

Lately, book drawings have also been studied for *1-planar graphs*. A graph is 1-planar if it can be drawn in the plane such that each edge has at most one crossings. Every 1-planar drawing has at most $n - 2$ crossings and since planar graphs have at most $3n - 6$ edges, it follows that 1-planar graphs have at most $4n - 8$ edges [CH13]. However, there exist 1-planar graphs with fewer edges where no edge can be added while preserving 1-planarity. These are called *maximal*. 1-planar graphs that have $4n - 8$ edges are, on the other hand, called *optimal*. Bekos et al. [BBKR15] have shown that 1-planar graphs have pagenumber at most 39. This has been improved further by Alam et al. [ABK15].

Theorem 4.7 ([ABK15]). *Let G be a 1-planar graph. Then holds:*

- *The pagenumber of G is at most 16.*
- *If G is 3-connected, then its pagenumber is at most 12.*
- *If the planar skeleton of G is Hamiltonian, then the pagenumber of G is at most 4.*

Their proofs are constructive and the algorithms run in linear time if provided with an 1-planar embedding of the considered graph. Furthermore, in the former two cases the algorithms are based on the algorithm by Yannakakis [Yan89] for planar graphs. It is open, whether there exist 1-planar graphs that are not embeddable in a 4-page book. In Section 2.3 we mentioned that Bekos et al. [BKZ15] used SAT solving to test for some planar graphs whether they have 3-page book embeddings. They also tested optimal 1-planar graphs with 25 to 155 vertices, which they could embed all in books with 4 pages.

4.5.1. Experimental results

We tested the heuristics on 200 1-planar graphs of sizes 50, 100 and 150 with k from 1 to 16. These are the same setting as in the experiment on random graphs with linear number of edges. The results are easy to report. `conGreedy-ceilFloor` was the best combination for all k. This stands in contrast to the results on random graphs with $m \approx 4n$, where `conGreedy+` won with a clear lead. Furthermore, `conGreedy-ceilFloor` needed at most 8 pages to achieve an average number of crossings below 1. Concerning $k = 2, 3$, further results can be found in Appendix B.1.

We also measured the average number of crossings achieved by the heuristics for $k = 4$ and growing n. A selection of the heuristics is shown in Figure 4.14. `conGreedy+` achieved the

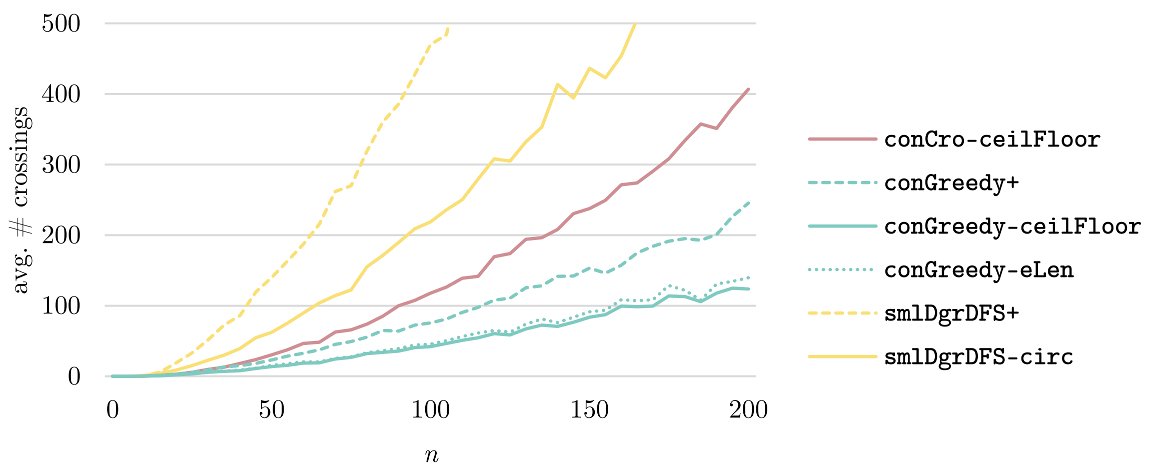


Figure 4.14.: Average number of crossings achieved by a selection of heuristic combinations on 1-planar graphs on four pages.

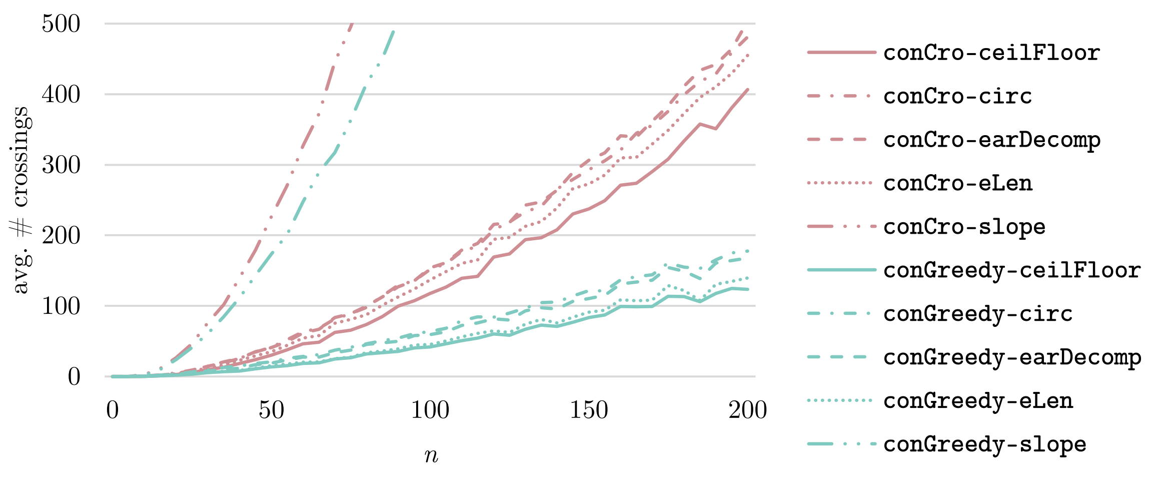


Figure 4.15.: Comparison of the combinations with `conCro` and `conGreedy` in terms of average number of crossings achieved on 1-planar graphs on four pages.

best results when run as full drawing heuristic and not with additional edge distribution heuristic afterwards. However, `smlDgrDFS+` performed again way worse than `smlDgrDFS`. The more interesting observation is that again the choice of the vertex order heuristic is more important than of the edge distribution heuristic, at least if the latter is not `slope`. This is emphasized by Figure 4.15, which shows all edge distribution heuristics in combination with `conCro` and `conGreedy`. Ignoring `slope`, we can see that the differences between edge distribution heuristics with the same vertex order heuristic are far less than between different vertex order heuristics.

4.6. K-trees

A *K-tree for fixed K* can be defined recursively. First, the complete graph with K vertices is a K-tree. Next, if G is a K-tree and C is a K-clique of G, then the graph formed by adding a new vertex to G and making it adjacent to all vertices of C is again a K-tree. Hence, a K-tree has $m = nK - \frac{K-1}{1} - 1$ edges, which for large n is roughly nK. A subgraph of a K-tree is a *partial K-tree* and has *treewidth K*.

1-trees are exactly the trees and thus have pagenumber 1. All 2-trees are maximal series-parallel graphs and thus have pagenumber 2 [RVM95]. Maximal outerplanar graphs are also 2-trees. Ganley and Heath [GH01] have proven constructively that every K-tree has pagenumber at most $K + 1$. This bound is tight for $K \geq 3$ due to Dujmović and Wood [DW07], which gave examples of K-trees that need $K + 1$ pages. Hence, the following theorem holds.

Theorem 4.8 ([GH01, DW07]). *Every K-trees has pagenumber at most $K + 1$, which can be tight for $K \geq 3$.*

This result does not imply that for $K \geq 3$ all K-trees need $K + 1$ pages. On the contrary, the complete graph K_n, which is a n-tree, can be embedded without crossings in $\lceil \frac{n}{2} \rceil$ pages (as shown in Section 4.8). We want to mention that K-trees are normally denoted with lower case k. However, since we use k for the number of pages, we use K instead.

We are interest in K-trees for several reasons. On the one hand, as we have seen in Section 2.2, the problem of determining $\mathrm{cr}_2(G)$, i.e. the minimal number of crossings possible for a graph G in a 2-page book, is fixed parameter tractable in the treewidth of G and $\mathrm{cr}_2(G)$ [BE14]. Furthermore, if we consider the structure of K-trees, we see that a good strategy to draw K-trees in books works similar to the embedding of trees in 1-page books. The branches of a K-tree (of its *width-k tree-decomposition*, respectively) should not intersect. For further details we refer to the constructive proof for $k = K+1$ by Ganley and Heath [GH01]. So, summing up, we see that K-trees are tightly coupled to the k-page crossing minimisation problem. On the other hand, K-trees of different K provide us with graphs that have roughly $m = Kn$ edges.

4.6.1. Experimental results

We tested the heuristics again on 200 graphs for $n = 50, 100, 150$ and k from 1 to $K + 1$, i.e. the maximal pagenumber. The winning heuristics are shown in Figure 4.16.

Our first observation is that the best vertex order heuristic was always `conGreedy`. We recall from the results for random graphs that there the tiles for different factors a (for number of edges $m = an$) and the number of pages were covered by three regions. `con-Greedy+` was best in most of the cases. However, for $a > 6$ and $k > 3$ `slope` in combination with `conGreedy` and `randDFS` was often the best heuristic. Therefore, our second observation is that `conGreedy-ceilFloor` has replaced them and is for K-trees in most cases the best combination, at least for the ranges of K we tested.

The third observation we can make is that the greedy edge distribution heuristic `circ` was the best choice for $k = 2$ as well as for bigger k with growing n. This stands in contrast to the result on random graphs and also the other graph classes tested above.

Our last observation is that for $k = K + 1$ the full drawing heuristic `conGreedy+` always performed best. Figure 4.17 shows that `conGreedy+` in fact also got way better than the others for this particular k. We can report that it achieved an average total number of crossings below 10, whereas the results of the other heuristics and especially of `conGreedy-ceilFloor`, which was best for one page less ($k = K$), did not drop this far. For example

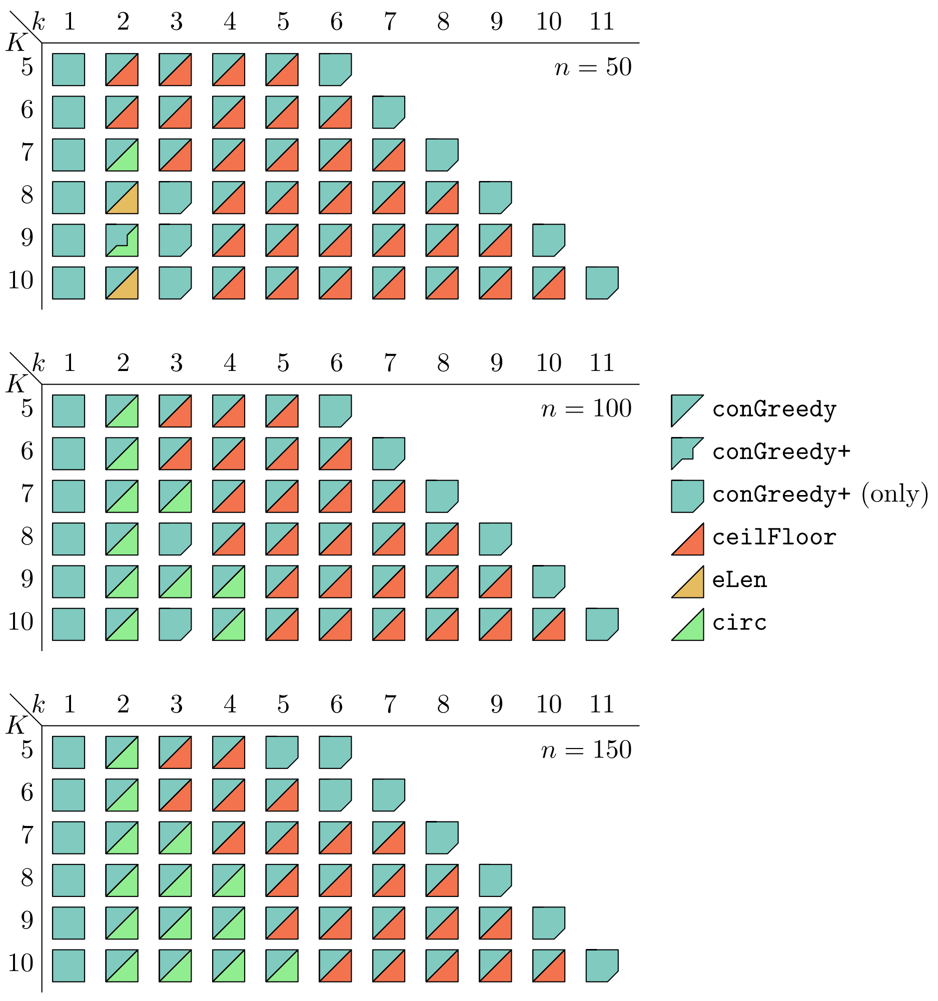


Figure 4.16.: Winning heuristic on K-trees for different number of pages k.

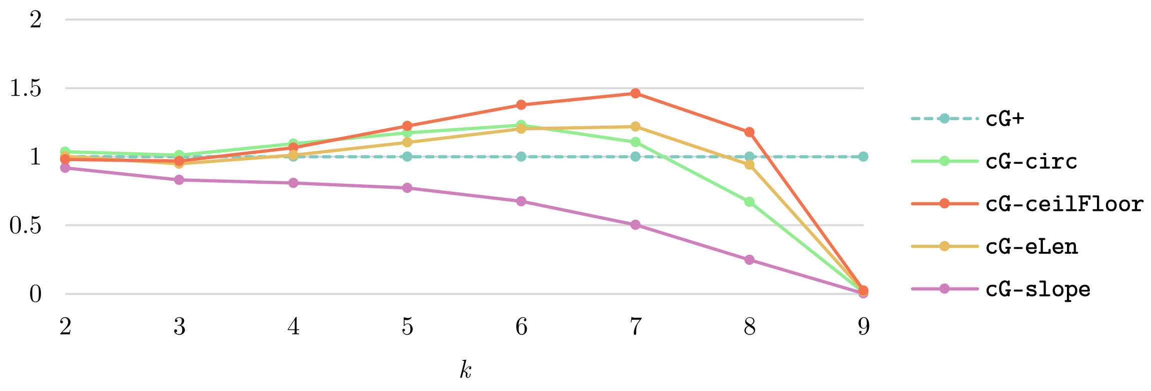


Figure 4.17.: Average number of crossings of `conGreedy` (abbreviated with `cG`) with several edge distribution heuristics relative to `conGreedy+` for 8-trees, $n = 150$ and k from 2 to 9.

for $n = 150$ and $K = 8$ `conGreedy+` improved from an average total number of crossings of over 900 to 8.5, while `conGreedy-ceilFloor` only got from 760 to 340.

In summary, we can say that the results on K-trees differ to the results on random graphs with the same density and also for specific k to the other graph classes tested above. Moreover, `conGreedy+` works well on the maximal pagenumber $k = K + 1$.

4.7. Hypercubes and t-ary d-cubes

In this section we consider *hypercubes* and *t-ary d-cubes*. The latter are known as *k-ary n-cubes*, however we call them t-ary d-cubes to not overload the notation of k and n. The closely related graph classes, incomplete hypercubes and cube-connected cycles, have also been considered in the context of book drawings [KHT89, Cim02, Has09, TS10].

A *hypercube Q_d* of *dimension d* is the Cartesian product of d edges (or C_2). They can also be defined to be the graphs that have all binary strings of length d as vertex set and where two vertices are adjacent if and only if they differ in exactly one bit. The class of hypercubes is the first graph class (not counting random graphs) we consider, where the pagenumber is not bounded by a constant.

Theorem 4.9 ([KHT89])**.** *The hypercube Q_d of dimension $d \geq 2$ has pagenumber $d - 1$.*

2-page book drawings of hypercubes have been considered Faria et al. [FdFRv13].

t-ary d-cubes can be defined in terms of Cartesian products of graphs and strings as well. A t-ary d-cube is the Cartesian product of d cycles C_t. Equivalently, it is also the graph with vertex set all strings of length d over the alphabet $\{0, 1, \dots, t-1\}$, where again two vertices are adjacent if and only if they differ in exactly on position by 1 mod t. Figure 4.18 depicts the first three 3-ary d-cubes. Furthermore is a hypercube Q_d a 2-ary d-cube and a cycle C_t a t-ary 1-cube. The Cartesian product of two cycles, which we considered in Section 4.3, is a t-ary 2-cube if both cycles have length t. Except for these three cases, we are not aware of known pagenumbers.

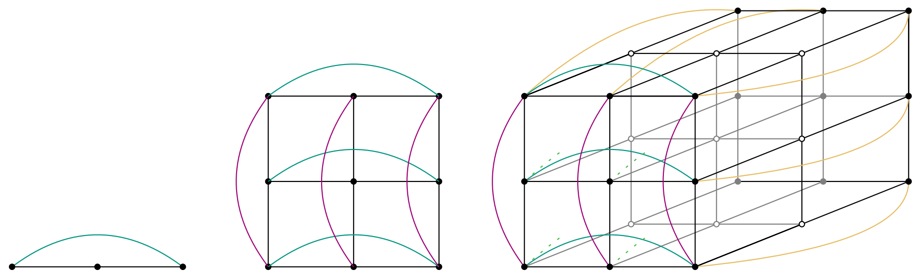


Figure 4.18.: From left to right the 3-ary 1-cube, the 3-ary 2-cube and the 3-ary 3-cube (some edges inside cube omitted).

The hypercubes Q_d has $n = 2^d$ vertices and $m = \frac{1}{2}nd = \frac{1}{2}n \log n$ edges, since every vertex has degree d. A t-ary d-cube has $n = t^d$ vertices, all with degree $2d$, if $d > 2$, and thus $m = n \log_t(n)$ edges. Hence, these graphs do not have a linear but a linearithmic number of edges. Both graph classes are of interest since they have many automorphisms (or symmetries).

4.7.1. Experimental results

We tested the heuristics on different hypercubes Q_d as well as t-ary d-cubes for $k = d - 1$ and for $k = d + 1$, respectively. Results for different k can be found in Appendix B.2. Each heuristic was executed 50 times. Figure 4.19 shows the average number of crossings per edge for a selection of heuristic combinations on hypercubes, which are, except for `conGreedy`, the best combinations per vertex order heuristic. Our main observation is that `conGreedy-ceilFloor` and `conGreedy-eLen` achieved always zero crossings. The third greedy edge distribution heuristic `circ` in combination with `conGreedy` came close behind but did not achieve perfect results for higher dimensions.

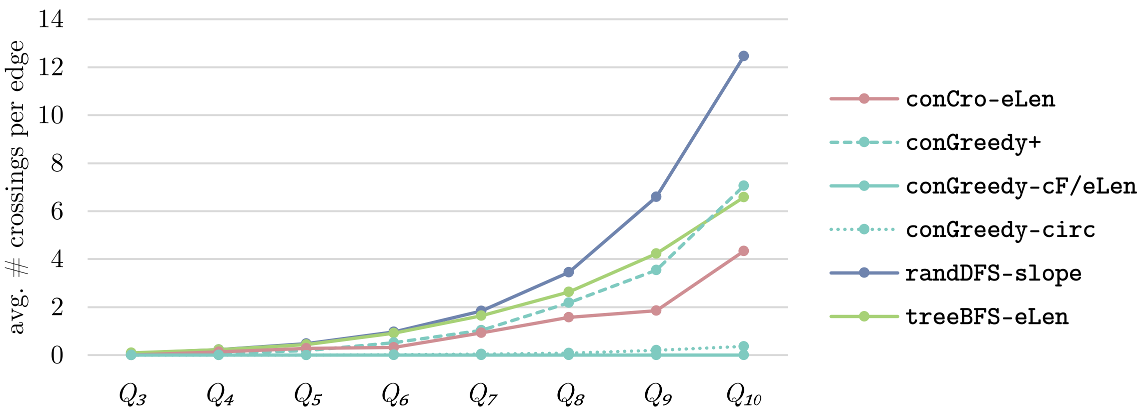


Figure 4.19.: Average number of crossings achieved by a selection of heuristic combinations on hypercubes Q_d and $k = d - 1$. (`ceilFloor` is abbreviated with `cF`).

Figure 4.20 shows the performance on the t-ary d-cubes of combinations of vertex order heuristic with `circ` , since the latter was always the best edge distribution heuristic. However, the vertex order heuristic is still the more important choice, which in this case is `conCro`. We recall that `smlDgrDFS` and `randDFS` work identically on regular graphs. Furthermore, it is noteworthy that both on hypercubes and t-ary d-cubes `treeBFS` performed better than `randDFS` and on t-ary d-cubes also sometimes better than `conGreedy`. We have already seen this on the Cartesian products on cycles. Hence, for regular and very symmetric graphs there seems to be a trend.

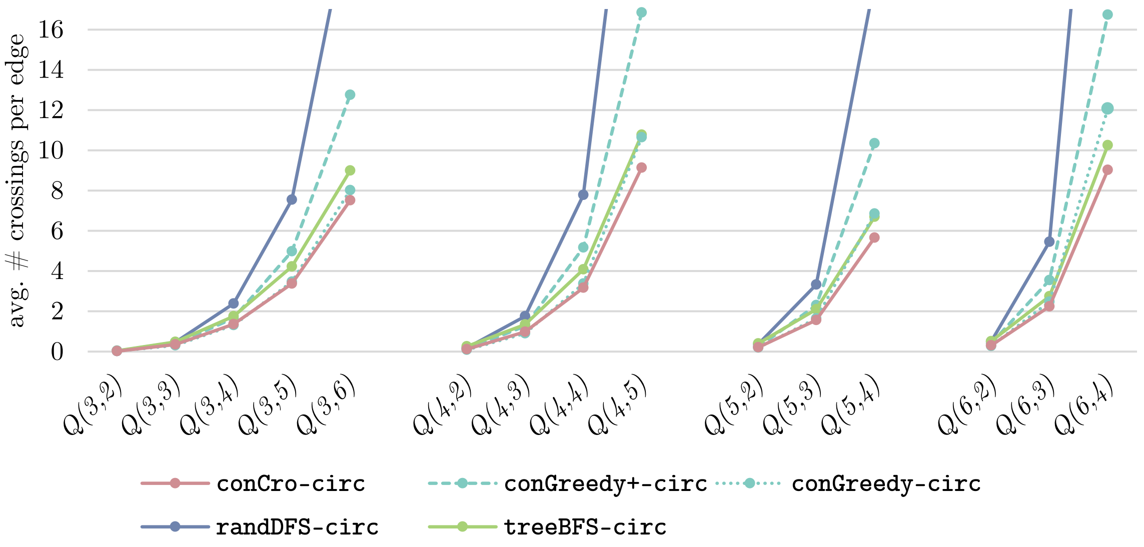


Figure 4.20.: Average number of crossings achieved by vertex order heuristics in combination with `circ` on t-ary d-cubes $Q(t, d)$ and $k = d + 1$.

4.8. Complete graphs

We now consider graphs with higher density. We start with the graphs containing an edge between any two vertices, the *complete graphs*, and then continue with *complete bipartite graphs*. After that we will evaluate the heuristics on random graphs with high density.

The *complete graph* K_n on n vertices contains all $m = \frac{n(n-1)}{2}$ edges. Hence, it has only one vertex order. If we now consider a 1-page book drawing of K_n, we can directly determine the number of crossings. Since any four distinct vertices produce exactly one crossings, $\mathrm{cr}_1(K_n) = \binom{n}{4}$. The minimal number of crossings is also known for $k = 2$ and of course $k = \mathrm{pn}(K_n) = \lceil \frac{n}{2} \rceil$. We first take a look at the latter case.

Theorem 4.10 ([BK79])**.** *The complete graph K_n with $n \geq 4$ has pagenumber $\lceil \frac{n}{2} \rceil$.*

Proof. First, we show that $\lceil \frac{n}{2} \rceil$ is a lower bound for $\mathrm{pn}(K_n)$ and than that we actually can find a $\lceil \frac{n}{2} \rceil$-page book drawing of K_n.

We recall from our observations on outerplanar graphs in Section 2.1.3 that for the pagenumber of a graph a lower bound is given by $\frac{m-n}{n-3}$. This yields for K_n that we need at least $\frac{\frac{n(n-1)}{2}-n}{(n-3)} = \frac{n(n-3)}{2(n-3)}$ pages, or more precisely $\lceil \frac{n}{2} \rceil$. Hence, we have proven the lower bound.

Let now n be even. Then, since K_{n-1} is a subgraph of K_n, the result will hold for n odd as well. We now cover K_n with $\lceil \frac{n}{2} \rceil$ zigzag paths, as illustrated in Figure 4.21 for K_8. We start at vertex i and then go to $i+1, i-1, i+2, \ldots, i+\lceil \frac{n}{2} \rceil$, iterating i from 1 to $\lceil \frac{n}{2} \rceil$. Clearly, each path is crossing free and all edges are covered. Hence, the upper bound holds as well and thus also the statement. $\square$

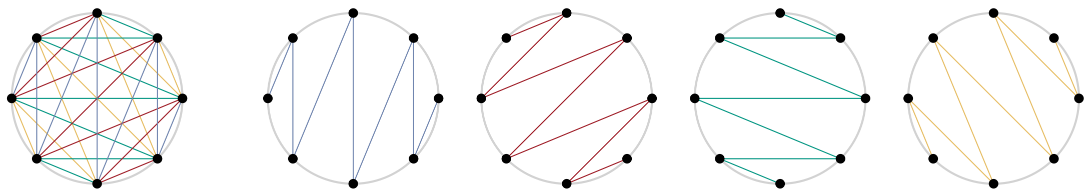


Figure 4.21.: Book embedding of K_8.

We observe that `slope` produces exactly these book drawings of K_n. Moreover, it is known that `slope` creates also the 2-page book drawings of K_n with minimal number of crossings [dKPS13]. Figure 4.22 depicts such a book drawing for K_8. It is conjectured that the exact minimal number of crossings for two pages $Z_2(n)$, which is given below, is also the (planar) crossing number of K_n, i.e. $\mathrm{cr}(K_n) = Z_2(n)$ [dKPS13]. This is known as the Harary-Hill conjecture.

Theorem 4.11 ([AAFM⁺12])**.** *The 2-page crossing number of the complete graph K_n is $Z_2(n) = \frac{1}{4}\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor \lfloor \frac{n-2}{2} \rfloor \lfloor \frac{n-3}{2} \rfloor$.*

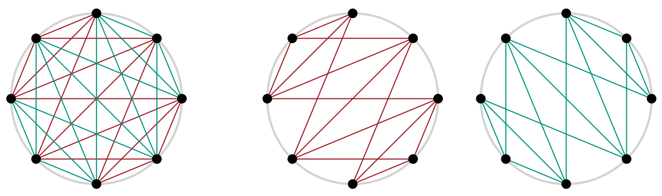


Figure 4.22.: Crossing optimal 2-page book drawing of K_8.

In fact, de Klerk et al. [dKPS13] have conjectured that the book drawings obtained by `slope` are optimal for any k.

Conjecture 4.12 ([dKPS13])**.** *The edge distribution heuristic `slope` computes k-page book drawings of complete graphs K_n with minimal number of crossings for all $2 \leq k \leq \lceil \frac{n}{2} \rceil$.*

Furthermore, they gave the exact number of crossings of these drawings. We want to note that they described different but equivalent approaches to obtain these drawings.

We therefore conclude that complete graphs are not suitable for the evaluation of heuristics. They should only be considered to check whether `slope` does not work perfectly or whether the heuristic under consideration works as good as `slope`.

4.9. Complete bipartite graphs

We now consider *complete bipartite graphs* $K_{s,t}$. For $s \leq t$ we get a book embedding on s pages of $K_{s,t}$, if we embed each of its s different subgraphs $K_{1,t}$ (stars) in a single page. Using the pigeon hole argument, Bernhart and Kainen [BK79] have shown that this is optimal if s is a lot smaller than t, more precisely, if $s^2 - s + 1 \leq t$. However, this does not hold in general. For example, a better bound is known in the balanced case.

Theorem 4.13 ([ENO97])**.** *The pagenumber of $K_{s,s}$ is at most $\lfloor \frac{2s}{3} \rfloor + 1$.*

For further bounds on the pagenumber as well as k-page crossing numbers we refer to Bernhart and Kainen [BK79], Muder et al. [MWW88], Enomoto et. al [ENO97] and de Klerk et al [dKPS14].

4.9.1. Experimental results

We tested the heuristics on different complete bipartite graphs with 50 and 100 vertices of different densities to compare the results with random graphs. We used $K_{25,25}$ ($K_{50,50}$) for density 0.5, $K_{13,37}$ ($K_{28,72}$) for 0.4, $K_{9,41}$ ($K_{18,82}$) for 0.3 and $K_{6,44}$ ($K_{11,89}$) for 0.2. Each heuristic ran 50 times on each graph.

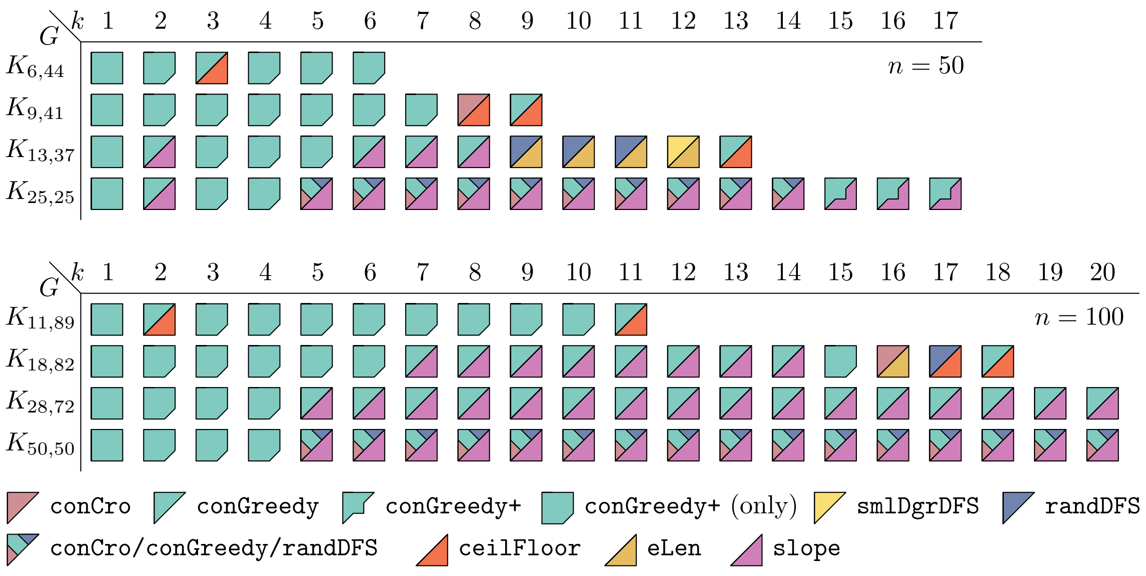


Figure 4.23.: Heuristics on complete bipartite graphs $K_{s,t}$ with densities from 0.2 to 0.5 for 50 (top) and 100 vertices (bottom).

Figure 4.23 shows which heuristics performed best for these graphs. The first thing we observe is that for the balanced graphs `conCro`, `conGreedy` and `randDFS` together with slope were best for k from 5 to 14 and 20, respectively. If we consider how they work, we notice that they all create a vertex order where the vertices alternate between the two partitions. However, we can report that in most cases subsequent greedy optimisation yields a lower number of crossings and thus, these book drawings seem not to be optimal. Next, we observe that `conGreedy+` was successful for small k and `conGreedy-slope`for larger k. We will see this pattern again in the experiment on random graphs with quadratic number of edges in the next section.

Figure 4.24 shows the performance in terms of average number of crossings of some heuristics in comparison to `conGreedy-slope` on $K_{28,72}$. We observe that `slope` is for these cases a better choice than `ceilFloor` (or other greedy edge distribution heuristics). However, we can also observe that the choice of the vertex order heuristics is important as well and that `conGreedy+` becomes for bigger k a worse and worse choice.

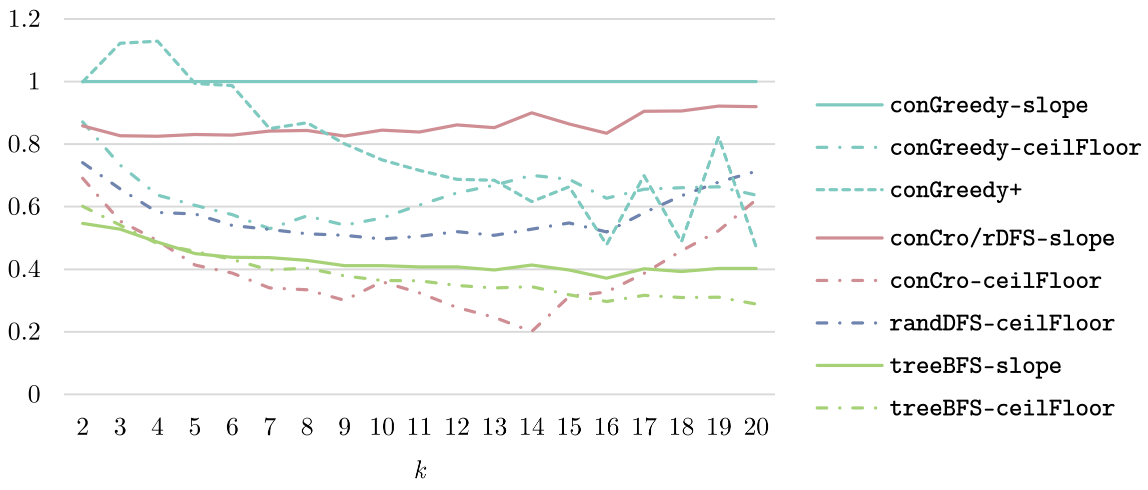


Figure 4.24.: Performance of some heuristics in comparison to `conGreedy-slope` in terms of average number of crossings on the complete bipartite graph $K_{28,72}$ for k from 2 to 20.

4.10. Quadratic number of edges

In this section we consider random graphs with high density, i.e. where the number of edges is quadratic in n, namely, a percentage p of all possible edges. We tested again with 200 graphs for 50, 100 and 150 vertices and k ranging from 1 to 20. This time, however, we tested for densities from 10% to 100% or in other words edge probability p from 0.1 to 1. The results are presented in Figures 4.25, 4.26 and 4.27. We note that for $k = 1$ no edge distribution is needed and that for $p = 1$, the complete graphs, only one vertex order exists. Hence, the last row of each diagram resembles the conjecture from Section 4.8.

Comparing with the results for graphs with linear number of edges (see Figures 4.1, 4.2 and 4.3) we observe the following. On the one hand, we can observe that `conGreedy+` got less dominant for higher density. However, it was still the best heuristic for two and three pages. On the other hand, the pattern from the tests with linear number of edges continues to arise. `slope` is the best and only successful edge distribution heuristic for dense graphs. Either in combination with `conGreedy` or `randDFS`.

The key argument for the good performance of `slope` on random graphs with quadratic density is the fact that, as we have seen, `slope` is conjectured to work perfectly on complete graphs. Random graphs are basically complete graphs with missing edges and no further structure, in contrast to complete bipartite graphs. So due to this resemblance and the missing structure of random graphs, it is not surprising that the geometry of the drawing is more important to the number of crossings and thus `slope` performs best.

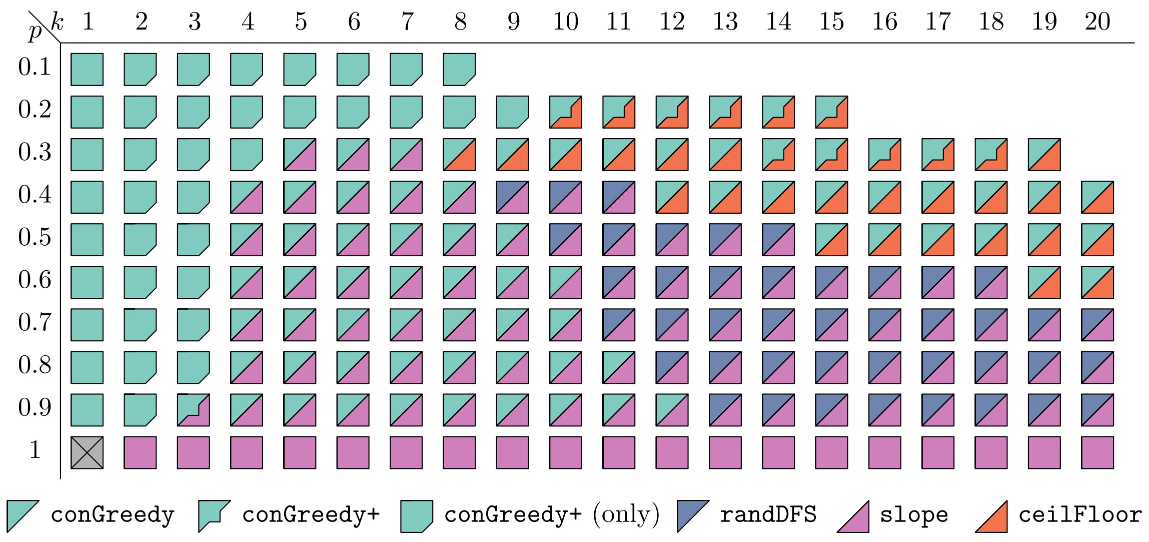


Figure 4.25.: Heuristics on random graphs with different density and 50 vertices.

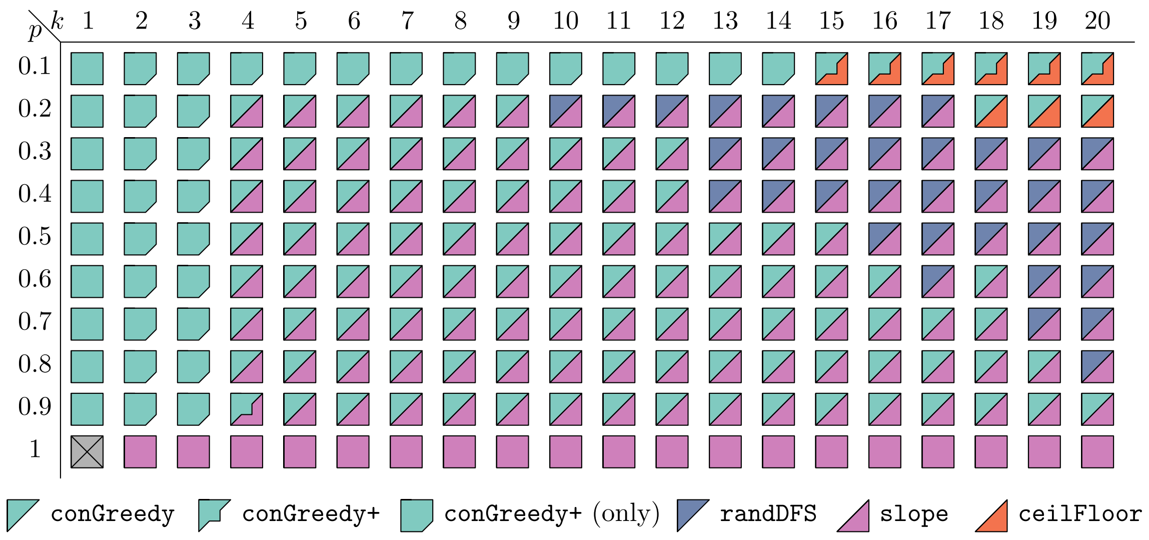


Figure 4.26.: Heuristics on random graphs with different density and 100 vertices.

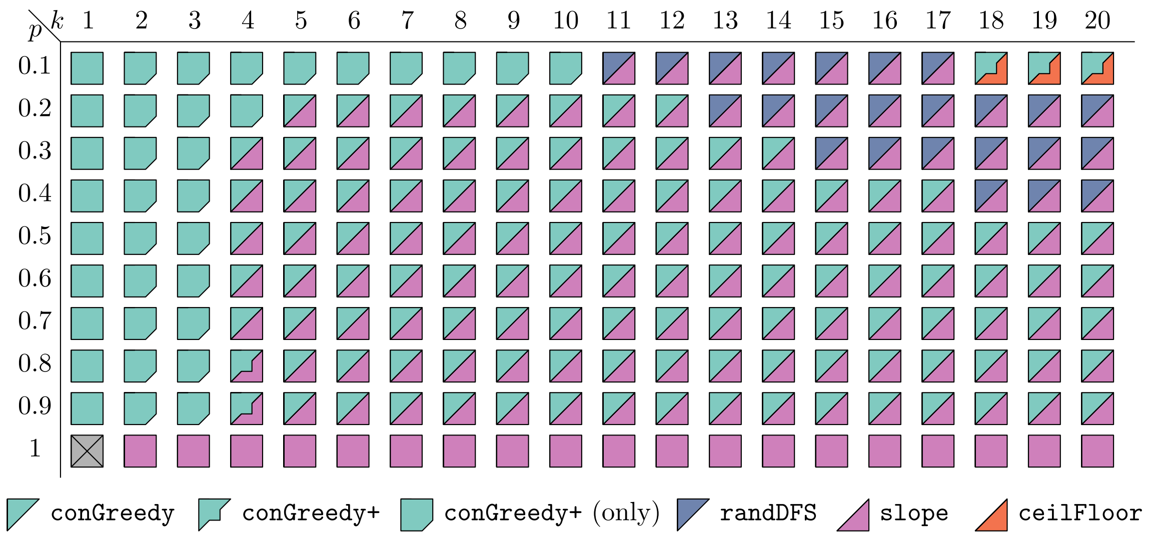


Figure 4.27.: Heuristics on random graphs with different density and 150 vertices.

4.11. Conclusion

In the previous sections we looked at the performance of the heuristics for different graph classes and graph densities. We will now summarize our findings and make some conclusions.

We can record that our new heuristic `conGreedy` is successful as vertex order heuristic as well as its extended version `conGreedy+` as full drawing heuristic. `conGreedy` is mostly the best choice if we have only one page. For small k mostly `conGreedy+` was the best choice and for higher k `conGreedy` in combination with an edge distribution heuristic. However, we have seen that they are clearly worse than `conCro` for $k = 1$ on the Cartesian product of cycles and for $k = d + 1$ on t-ary d-cubes. `conGreedy+` in combination with `ceilFloor` was also more successful than `conGreedy-ceilFloor` for random graphs with linear number of edges and high k. So even when the edge distribution of `conGreedy+` is not the best choice, the vertex order can still benefit from the simultaneous computation of an edge distribution.

The full drawing heuristics `smlDgrDFS+`, `randDFS+` and `randBFS+`, however, perform worse than their corresponding vertex order heuristics in combination with another edge distribution heuristic.

Our other new heuristics, however, performed well in some cases. `treeBFS` works well on regular graphs like $C_i \times C_j$, Q_d, t-ary d-cubes. It was still not the best choice for these graphs, however, it performed often best among the search based heuristics and even better than `conGreedy+` on the t-ary d-cubes and $k = d + 1$. So for regular graphs and if linear running time is required, `treeBFS` is a good candidate. Our new edge distribution heuristic `earDecomp` performs well for $k = 2$ and structured sparse graphs like the product of cycles and planar graphs. However, on random graphs, graphs with higher densities or for more pages it was outperformed by other heuristics.

Regarding the choice of the heuristics, our main conclusion is that for low graph density a version of `conGreedy` is mostly the best choice, while for higher density `slope` becomes most dominant. If the density is low, a good advice is to not take `slope` as edge distribution heuristic. As we have seen, it performs mostly worse than others. Furthermore, the vertex order heuristic is the more important choice. Differences are often higher among the results produced by different vertex order heuristics than by different edge distribution heuristics. On the other hand, if the density gets higher, `slope` gets more and more successful. This is possibly due to the fact that it is conjectured that `slope` computes perfect book drawings of complete graphs. Only for k near the pagenumber we can observe that sometimes other edge distribution heuristics perform better. This might be due to the fact that with many pages the geometry gets less and less important and the structure of the graph becomes more significant. However, for graphs with high density, we also observe that the choice of the vertex order heuristic does still have a large impact on the resulting number of crossings. So in general, computing a vertex order greedily works mostly well and either a greedy edge distribution (low density) or `slope` (high density) is the best choice. However, for particular graph classes and k it can be worth checking which heuristics perform best.

Furthermore, we were interested in the creation of a test suite. Concerning this matter, we have seen that all the different graph classes, different densities and different number of pages yielded new insights. There were differences between drawing-related graphs, like planar and 1-planar graphs, and regular graphs with many automorphisms, like hypercubes. Moreover, the results differed also between planar and 1-planar graphs as well as between hypercubes and t-ary d-cubes. For growing density or number of pages, the results changed also for random graphs. Especially the cases $k = 2, 3$ and $k = \mathrm{pn}(G)$ have shown interesting results. Summarising, we conclude that the graph classes we used provide a diverse test suite for the evaluation of the performance of a heuristic. This hols for vertex order, edge distribution or full drawing heuristic.

We advise to not use combinations of graph classes and number of pages where exact algorithms with feasible running time are known. Examples are outerplanar graphs, planar graphs and $k = 4$ or complete graphs. However, these cases can still be considered in order to find out whether a heuristic finds the best book drawings as well.

5. Optimisation

In this chapter we consider approaches to optimise given book drawings in terms of crossings. The presented optimisation algorithms are, in contrast to the heuristics discussed above, not restricted to compute a drawing in one run but are allowed to iterate and to use multiple drawings. In Section 5.1 we look at different greedy optimisation algorithms and also briefly evaluate them. After that, in Section 5.2, we look at evolutionary algorithms. This includes those presented in the literature as well as our implementation and a combination with greedy optimisation. We compare the algorithms of these two approaches in Section 5.3. We then discuss further possible optimisation approaches, including force-based ones, in Section 5.4.

5.1. Greedy optimisation

We consider several greedy optimisation algorithms. First, those that only alter the vertex order, then, those that only optimise the edge distribution, and finally algorithms that improve both.

5.1.1. Greedy vertex order optimisation

Greedy vertex order optimisation works mostly like the two heuristics `conCro` and `conGreedy` from Section 3.1.4, which compute a vertex order greedily. The algorithm picks a vertex and places it at the position where the number of crossings gets reduced the most. If it does not find a better position, the vertex is not moved. However, in contrast to the heuristics, this algorithm also considers the edge distribution (if $k > 1$) of the book drawing. The algorithm then repeats moving a vertex to the best position with all vertices. We call this one *round* of greedy vertex order optimisation.

An exhaustive greedy optimisation can run several rounds. We consider the following situation. The algorithm already moved vertex u to its current best position. Next, after it moved another vertex v, it is possible that u could be moved again to an even better position. Hence, if the algorithm moves at least one vertex in the current round, it should run another round afterwards. If, however, the algorithm only moves the first vertex of a round or no vertex at all, than it stops and the greedy vertex order optimisation reached a local minimum.

The algorithm can be implemented to run one round in $\mathcal{O}(mn)$ time. When searching for the best position the crossing number is not computed anew at each position. More cleverly, the considered vertex is swapped through the whole spine and with each swap the number of crossings is updated. Thus, only the edges of the two swapped vertices are involved and in total for all positions and all vertices each edge is only considered $\mathcal{O}(n)$ times.

Baur and Brandes [BB05] proposed this algorithm as post-processing for circular drawings with 1 page. They called it *sifting*, however, we refer to it with `greedyVOO`.

Local adjusting

He and Sýkora [HS04] and Six and Tollis [ST06] proposed a similar algorithm, which they called *local adjusting*. This algorithm also searches for the best position to place a vertex. However, it only considers the current position of a vertex and those next to its neighbours. He and Sýkora also gave an order in which to consider the vertices, namely with descending number of crossings on incident edges. Again, one round of the algorithm takes $\mathcal{O}(nm)$ time, however, finding the number of crossings for each vertex can take $\mathcal{O}(m^2)$ time in total. Hence, in our evaluation we will stick to the first described algorithm, `greedyVOO`, which uses a random order.

5.1.2. Greedy edge distribution optimisation

Greedy edge distribution optimisation takes an edge and moves it to the page, where it produces fewest crossings. It repeats this with all edges, which again is one round of optimisation. The algorithms can run one such round in $\mathcal{O}(m^2)$. The order in which the edges are considered is random in our implementation. However, one could also use the order used by `ceilFloor`, `eLen` or `circ`. As in `greedyVOO`, if an improvement was made in one round, the algorithm might reduce the number of crossings further in another round. We refer to this algorithm with `greedyEDO`.

We recall that the k-page crossing minimisation problem is equivalent to Max k-Cut if the vertex order is fixed (see Section 2.1.4). The equivalent of the greedy edge distribution optimisation is known as the *local search* algorithm for Max k-Cut. It has an approximation factor of $1 - \frac{1}{k}$ [GKK08]. Hence, greedy edge distribution optimisation avoids also $1 - \frac{1}{k}$ of all possible crossings. We note, however, that this does not directly yield a bound on the number of crossings, since the minimal number of crossings is not directly linked to the number of edges in the edge conflict graph. Consider for example the complete graph K_n and a non-complete graph G on a book with $\lceil \frac{n}{2} \rceil$ pages and an arbitrary vertex order. The edge conflict graph for K_n has $\binom{n}{4}$ edges, while the one for G has fewer. However, both graphs zero crossings are possible.

5.1.3. Greedy book drawing optimisation

We propose several algorithms for *greedy book drawing optimisation*, i.e. optimising a whole book drawing greedily. Above we have seen `greedyVOO` for greedy vertex order optimisation and `greedyEDO` for greedy edge distribution optimisation. These two algorithms can be combined in several ways. The first approach is to run them in an alternating manner, either one round of optimisation or exhaustively until they make no further improvement (*exhaustive execution*). We call these algorithms `greedyAlt` and describe the variants by different suffixe.

`greedyAltRR` Alternate one **r**ound of `greedyVOO` with one **r**ound of `greedyEDO`.

`greedyAltRE` Alternate one **r**ound of `greedyVOO` with an **e**xhaustive execution of `greedyEDO`.

`greedyAltER` Alternate an **e**xhaustive execution of `greedyVOO` with one **r**ound of `greedyEDO`.

`greedyAltEE` Alternate an **e**xhaustive execution of `greedyVOO` with an **e**xhaustive execution of `greedyEDO`.

Our second approach for greedy book drawing optimisation works like the full drawing heuristic `conGreedy+`. The algorithm takes the vertices in some order (not based on connectivity) and finds the best position for them on the spine, just like `greedyVOO`. However, at each considered position the incident edges of the vertex are placed on the best page as in `greedyEDO`. We note that, since pairwise incident edges can never cross each other, the order in which incident edges are redistributed is non-relevant and does not effect the result. One round of repositioning the vertices and distributing its incident edges can be implemented to run in $\mathcal{O}(m^2n)$ time. At each of the n positions, each edge is considered together with all other edges. We refer to this algorithm with `greedyCombi`.

For further work, more sophisticated greedy algorithms could be developed. One idea could be to modify `greedyCombi`, such that when it searches for the best position of a vertex, it not only considers the edges incident to this vertex but also the edges for which new crossings get introduced. Another approach could move sets of edges in `greedyEDO` instead of single edges, similar to an approach for graph partitioning [DMP95].

5.1.4. Evaluation

We want to compare the five greedy book drawing optimisation algorithms with each other. Therefore we test which algorithm optimises to the smallest number of crossings We tested the algorithms on multiple book drawings of planar graphs, 1-planar graphs and K-trees. Each initial drawing was created with `conGreedy+`. Figure 5.3 shows the obtained results. We also ran tests with other graph classes and parameters, but they indicated the some observations and hence we omitted the details of the full experiments.

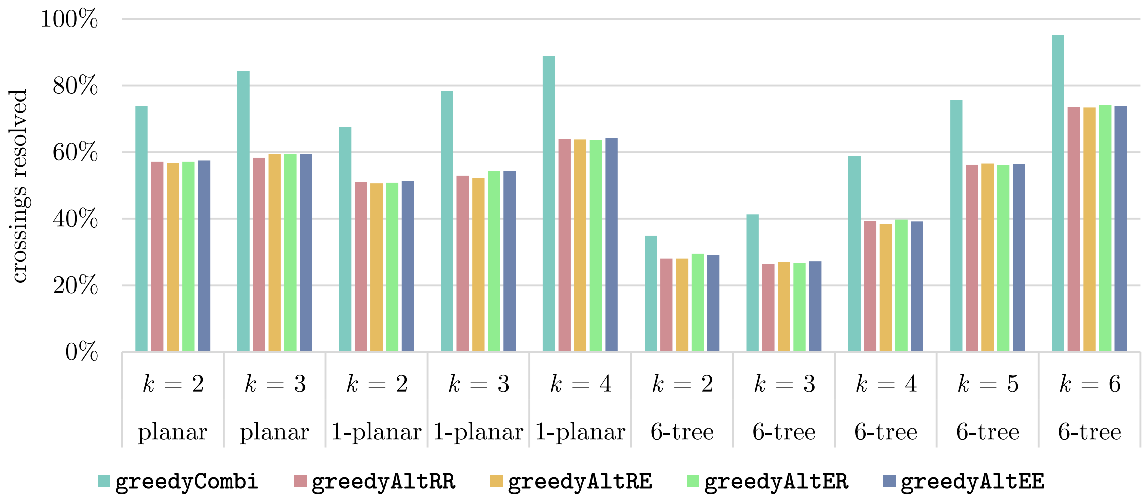


Figure 5.1.: Comparison of the greedy book drawing optimisers on planar and 1-planar graphs with 150 vertices and 6-trees with 100 vertices.

Our first result is that there was no notable difference between the four `greedyAlt` algorithms. Hence, we stick from now on two `greedyAltRR` and omit the suffix. Second, we observe that `greedyCombi` achieved far better results. However, it is not surprising that we have to report that it was also way slower. While the `greedyAlt` algorithms needed less than a second, `greedyCombi` needed between 10 and 17 seconds. Last but not least, we observe that with growing number of pages the performance of all algorithms got better in terms of resolved crossings. We can observe the same pattern in the results shown in Figure 5.2 of an experiment with the hypercube Q_8. The fact that `greedyEDO`, which is a subalgorithm of both `greedyAlt` and `greedyCombi`, gets better in resolving crossings for higher k supports the conclusion that our greedy book drawing optimisation algorithms get better in resolving crossings for higher k as well. However, we have to be aware that the results depend also on the varying performance with respect to k of the heuristic `conGreedy+`, which we used to compute the initial drawings. Hence, the results can neither be generalized to other graph classes nor do they directly proof an improved performance of the optimisation algorithms for higher k.

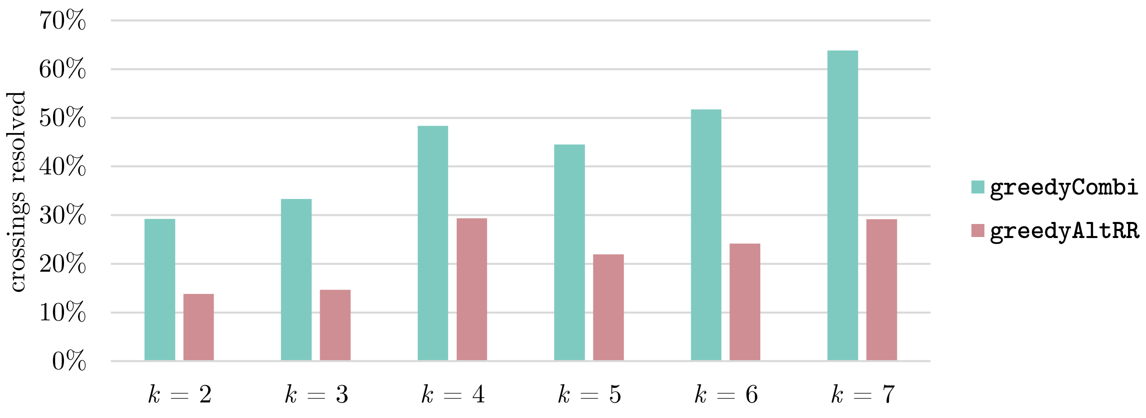


Figure 5.2.: Percentage of resolved crossings by `greedyCombi` and `greedyAlt` for different k on the hypercubes Q_8.

We also wanted to scratch the surface of the question whether different initial drawings have an effect on the outcome of the greedy optimisation. We therefore ran an experiment where we first created drawings with several heuristics of 1-planar graphs with 150 vertices and 3 pages and then used the greedy optimisation algorithms. The results are presented in Figure 5.3. We can observe that both the initial number of crossings as well as the particular heuristics effect the results. For more detailed answers one should test again with different graph classes, densities, number of pages and number of vertices, as we did for the evaluation of the heuristics.

5.2. Evolutionary algorithms

In this section we consider evolutionary algorithms for the k-page crossing minimisation problem. We start with an outline of evolutionary algorithms and then describe the variations we used in the experiments of the next section.

5.2.1. Mimicing evolution in nature

An *evolutionary algorithm* is an optimisation algorithm inspired by the biological evolution in nature. The goal is to find a good solution based on some fitness function for a given problem. The algorithm thereby mimics the evolution of nature in a series of steps. At first, it generates an initial *population* of solutions (the *individuals*) to the problem, the first *generation*. In the case of the k-page crossing minimisation problem, this is a set of k-page book drawings of the graph. The population then evolves over multiple generations and, hopefully, brings forth better solutions. Generating a new generation is implemented in multiple steps. First, the algorithm selects a subset of the current population as *parents*

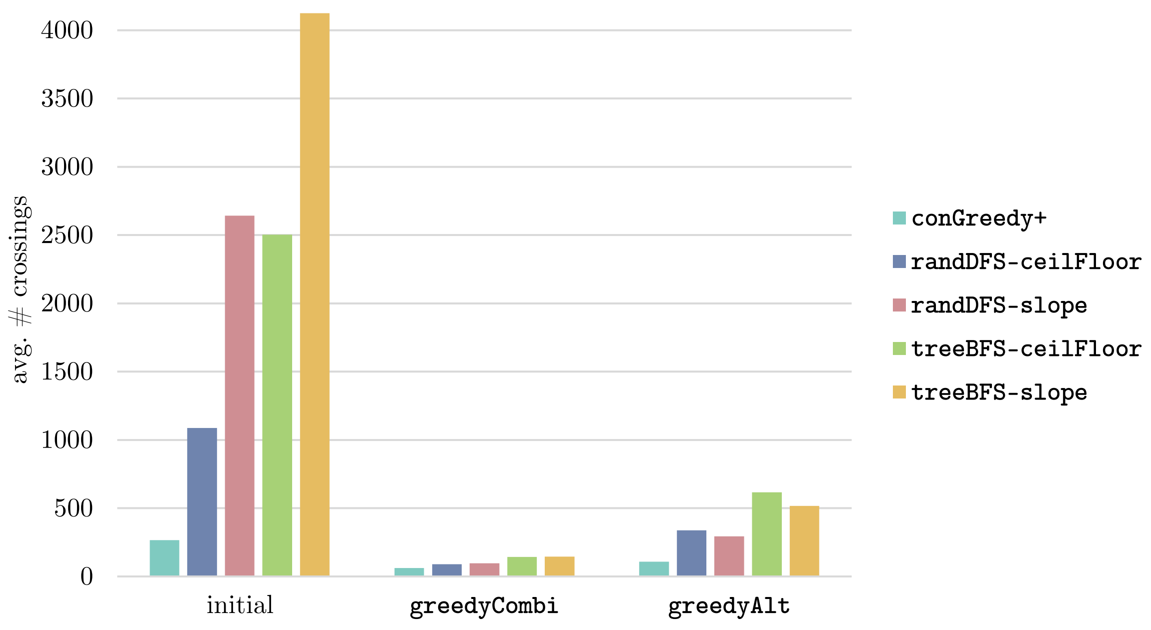


Figure 5.3.: Comparison of the performance `greedyCombi` and `greedyAlt` for different initial book drawings from different heuristics on 1-planar graphs with 150 vertices and $k = 3$.

for the next generation. Second, it combines these parents (*mating, recombination*) to get children. The idea is to create solutions that fuse good properties of their parents and, therefore, improve the population. Next, some of the children undergo a *mutation*, just like in nature. These mutations do not target an improvement of the quality of the children, but rather an exploitation of the search space. Finally, the algorithms selects individuals from the old population and the children to form the next generation. After a fixed number of iterations, elapsed time or when no further progress is made or possible, the algorithm stops. This iterative process is illustrated in Figure 5.4. We discuss how to implement this concept after a short remark.

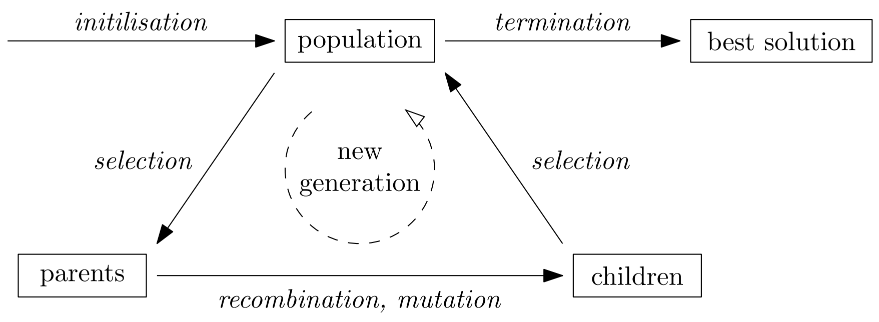


Figure 5.4.: Concept of an evolutionary algorithm. The population evolves by recombining a selection of parents and applying mutations to form the next generation.

The use of evolutionary algorithms in graph theory is not uncommon. For example, Sanders and Schulz [SS12] applied the concept to graph partitioning and Branke et al. [BBS96] to graph drawing. Evolutionary algorithms have also been used for book drawings and book embeddings. Kapoor et al. [KRSZ02] and Satsangi et al. [SSG11] used evolutionary algorithms to find the pagenumber of graphs. Moreover, evolutionary algorithms have been used for crossing minimisation in book drawings with one page [HNS05], two pages [HSM07, BSV$^+$08] and k pages [SSS13].

5.2.2. Implementation

We now discuss how to implement an evolutionary algorithm, in general but also, in particular, for the k-page crossing minimisation problem. The three main components are the population, the fitness function and the *operators*.

The population consists of individuals that are a representation of a solution. Depending on the problem at hand, the representation can, for example, be a bit vector, an array of integers or coordinates. To represent a book drawing, we simply use its vertex order and its edge distribution.

The fitness function summarises in a single figure of merit how good an individual is. Hence, it should reflect the underlying problem. However, it can include further properties of an individual. In our case, an individual with less crossings is better. If a higher fitness should be represented by a bigger number, the functions $\frac{1}{(C+1)}$ or $\frac{1}{(C^2+1)}$, with C the number of crossing, by He et al. [HSM07] could be used.

The operators process or create a set of individuals. All steps in Algorithm 5, which represents the framework for our evolutionary algorithms, are executed by such an operator. This includes the task of creating an initial population, selecting parents or survivors for the next generation, recombining parents and mutating children. Moreover, we add the step of developing individuals. This can be seen as a growing up or getting older. We propose that a book drawing develops by simple greedy optimisation. In the following, we describe the methods we use in each operator and how they work with regard to book drawings.

```
Algorithm: EvolutionaryAlgorithm
Data: k-page crossing minimisation problem
Result: best solution found
1  create first generation of population
2  compute number of crossings of individuals (fitness)
3  while termination criteria not fulfilled do
4      develop population
5      select parents
6      create children (recombine parents or recompute)
7      mutate children
8      develop children
9      compute number of crossings of children (fitness)
10     select survivor for next generation
11 return best solution
```

Algorithm 5: Framework of our evolutionary algorithms.

Initialisation

Bansal et al. [BSV$^+$08] and Satsangi et al. [SSS13] used `randDFS` and `eLen` to generate the initial population. However, we want more diversity in the initial population. Hence, we use `randDFS`, `smlDgrDFS`, `conCro` and `conGreedy` to compute initial vertex order. Taking into account the running time of the two connectivity based heuristics, we use the latter two only for a smaller proportion of the initial population. We then use `ceilFloor`, `eLen` and `slope` to compute the initial edge distributions. Having regard to our heuristics evaluation, we use `slope` on a larger portion for graphs with higher density.

Parent selection

There exist several strategies to select the parents, like selecting at random with probability proportional to their fitness (*fitness proportionate* or *roulette wheel*, *stochastic universal sampling*) or taking the best $\frac{1}{p}$ of the population p times (*truncation*). In one algorithm we simply use all parents and in another we use the *tournament* strategy. Here, two randomly picked individuals are compared (play a tournament) and the one with fewer crossings is selected. This is repeated till enough parents are found.

Recombination & recomputation

The representation of an individual is sometimes also called its genome. *Recombination* takes the genomes of two or more parents and combines them to generate two or more children. This is called *crossover.* We use the so called *two-point crossover* technique and only combine two parents to get two new children. This works differently for the spine and the edge distribution.

crossover spines Keep an interval of the spine (vertex order) in both of the parents. Then fill the rest of the spine with unplaced vertices in the order they have in the other parent.

crossover edge distribution Swap an interval of the edge distribution between two parents, i.e. swap the page assignment of some edges.

The described crossover on spines has also been used by He et al. [HNS05].

Bansal et al. [BSV$^+$08] and Satsangi et al. [SSS13] used an asexual operator to create children, called *recomputation.* This operator keeps a part of the vertex order, computes the order for the other vertices anew with `randDFS` and then also computes a new edge distribution. We use both, recombination and recomputation, in different variants of our evolutionary algorithms.

Spine alignment

The idea behind recombination is to fuse good properties of the parents. However, Branke et al. [BBS96] described the *competing conventions problem* in evolutionary algorithms for graph drawing. Two graphs drawings might be actually the same or nearly the same but are rotated, shifted or inverted. Hence, recombination might miss its initial objective and create inferior children. This might also occur in the crossover of spines. Therefore, we tested whether rotating and reversing spines to increase similarity before running crossover recombination improves the performance. However, we could not observe significant differences in our experiment.

Mutation

Mutations alter a small number of genes in the genome of an individual. The main purpose is to maintain genetic diversity and a broader exploitation of the search space. For each new child it is randomly decided whether it undergoes mutation. We decrease (increase) the chance to mutate, the *mutation rate*, in one of our algorithms when the population does (does not) improve over one generation. This is called *adaptive mutation rate.*

Mutations concerning the vertex order are for example moving vertices, swapping the positions of vertices and rotating or reversing a part of the order. We use the swapping of positions of two randomly chosen vertices. A mutation of the edge distribution moves an edge to a randomly chosen page.

Developing

We introduce an additional step in the normal evolutionary algorithm. After a child has been generated it is given the chance to *develop* (or *grow up*). Furthermore, the whole population gets older each iteration and thus also develops. Developing means that we optimise the individuals with one of our greedy optimisation algorithms. We thereby use two different strategies. In the first strategy we only optimise the position of one vertex and the assigned page of one edge. In the second strategy, we run one round of greedy vertex order optimisation `greedyVOO` and one round of greedy edge distribution optimisation `greedyEDO`, so in other words one round of `greedyAlt`. A third strategy could be to use one round of `greedyCombi`. However, we did not implement this strategy due to the slow running time of one round in `greedyCombi` in comparison to the other approaches.

Survivor selection

After generating a set of children, we have to decide which individuals come into the new population. We use a simple strategy and select the best individuals out of each parent and its child. One variation of this strategy is to select a child with some chance even if it is worse than its parent. Further possible strategies are to select only the children or to select the best out of the last generation and all the children.

5.2.3. Algorithms

We decided to test several different evolutionary algorithms, which differ in the amount of greedy optimisation they use for developing. Our choices of the operators and parameters are based on other publications, preliminary test runs and our intuition.

The first group of algorithms uses crossover for recombination. Two of them, `evo` and `evoSngl`, differ only in the developing operator. The former does not have development, while the latter optimises the position of a single vertex and the assigned page of a single edge. They use a population size of $\min\{\frac{3}{2}n, 100\}$. The size of $\frac{3}{2}n$ has also been used by Satsangi et al. [SSS13]. Both algorithms select parents with the tournament operator and survivors by comparing each parent with its child. The chance that a child mutates is set to 20%. The third algorithm, `evoRnd`, varies only in two points, namely, it has a population size of only 30 and uses one round of `greedyAlt` (`greedyVOO` and `greedyEDO`) as development.

The algorithm `evoRec` does not use crossover but instead uses recomputation with the asexual operator like Satsangi et al. [SSS13]. So from a parent a part of the vertex order is kept, while the second part is generated anew with `randDFS`. For the resulting vertex order a new edge distribution is computed with `ceilFloor`. The rest of the algorithm's settings are the same as those of `evoSngl`.

The algorithm `evoGreedy` is centered around greedy optimisation with `greedyAlt`. It does not generate new children, but reuses the whole population. This means that each individual gets optimised and mutates in an alternating manner, and that neither parent nor survivor selection is needed. The algorithm uses adaptive mutation rates, i.e. the chance that an individual mutates depends on the progress of the population. If greedy optimisation improves the population, the probability to mutate is decreased by factor 4, otherwise it gets set to 100%. The population size set to 5.

We want to stress that these algorithms are not designed to be the best of all possible evolutionary algorithms for the k-page crossing minimisation problem. The input of such a problem has many parameters, like n, m, k and the graph class. It is likely that for different configurations of these parameters different settings of the evolutionary algorithms

name	population size	child creation	mutation rate	developing
`evo`	$\min\left\{\frac{3}{2}n, 100\right\}$	recombination	20%	none
`evoSngl`	$\min\left\{\frac{3}{2}n, 100\right\}$	recombination	20%	single greedy opt.
`evoRnd`	30	recombination	20%	one round `greedyAlt`
`evoRec`	$\min\left\{\frac{3}{2}n, 100\right\}$	recomputation	20%	single greedy opt.
`evoGreedy`	5	none	adaptive	one round `greedyAlt`

Table 5.1.: Settings of the different evolutionary algorithms.

achieve better results. This has for example been show by He et al. [HSM07] for 2-page book drawings. However, this is also not the point we want to make. Our main goal is to show that the classical crossover operator does not work well and that, however, the combination of evolutionary algorithm with greedy optimisation in `evoGreedy` can achieve better results than plain greedy optimisation.

5.3. Evaluation of optimisation algorithms

We compare the evolutionary algorithms `evo`, `evoSngl`, `evoRnd` , `evoRec` and `evoGreedy` with the greedy book drawing optimisation `greedyCombi`.

Our first result concerns the evolutionary algorithms that use crossover for recombination. We have to report that we could not find settings for which the algorithms worked effectively. Figure 5.5 shows the progress of the three algorithms `evo`, `evoSngl` and `evoRnd` for 1-planar graphs with 250 vertices and $k = 3$ (top) as well as for the hypercube Q_8 and $k = 7 = \mathrm{pn}(Q_8)$ (bottom). More precisely, it depicts the number of crossings of the best individual in the population for the first 100 generations. We want to mention that in order to prove our point, we could have used any other graph class, and that, furthermore, the results did also not improve after several thousand iterations. We observe that the best solution was actually found in the first generation or at least very early for `evoRnd`. In fact, not even the heavy greedy optimisation of `evoRnd` was able to prevent the population from worsening after a few generations. We can observe, however, that `evoSngl` was able to slowly improve the population after the bad start. However, the improvement happened so slow and the population was after several thousand generations still way closer to the worst result than to best at the start. Furthermore, the difference between `evo` and both `evoSngl` and `evoRnd` on 1-planar graphs shows that greedy optimisation was here more successful against crossover and the mutations.

The diagrams in Figure 5.5 also show the performance of `evoRec`, which uses recomputation to generate children. We have to note that we used slightly different settings than Satsangi et al. [SSS13]. Nevertheless, even though Satsangi et al. could show good results for small graphs, we question the efficiency of this approach. We believe that the good results are actually based on the heuristics as well as on the choice of tested graphs and less on the evolutionary approach. Improvements to the algorithm could possibly be made, if, instead of only a version of `randDFS`, the algorithm would also use other heuristics and also other edge distribution heuristics like `slope`, especially for dense graphs. We tested the latter approach for random graphs with edge probability 0.6, $n = 150$ and $k = 6$. The result in Figure 5.6 shows an improvement of `evoRec` with the use of `slope` in comparison to `ceilFloor`. However and nevertheless, with regard to the evaluation of `evoGreedy` and `greedyCombi`, which comes next, we have to question whether variations of `evoRec` are actually reasonable.

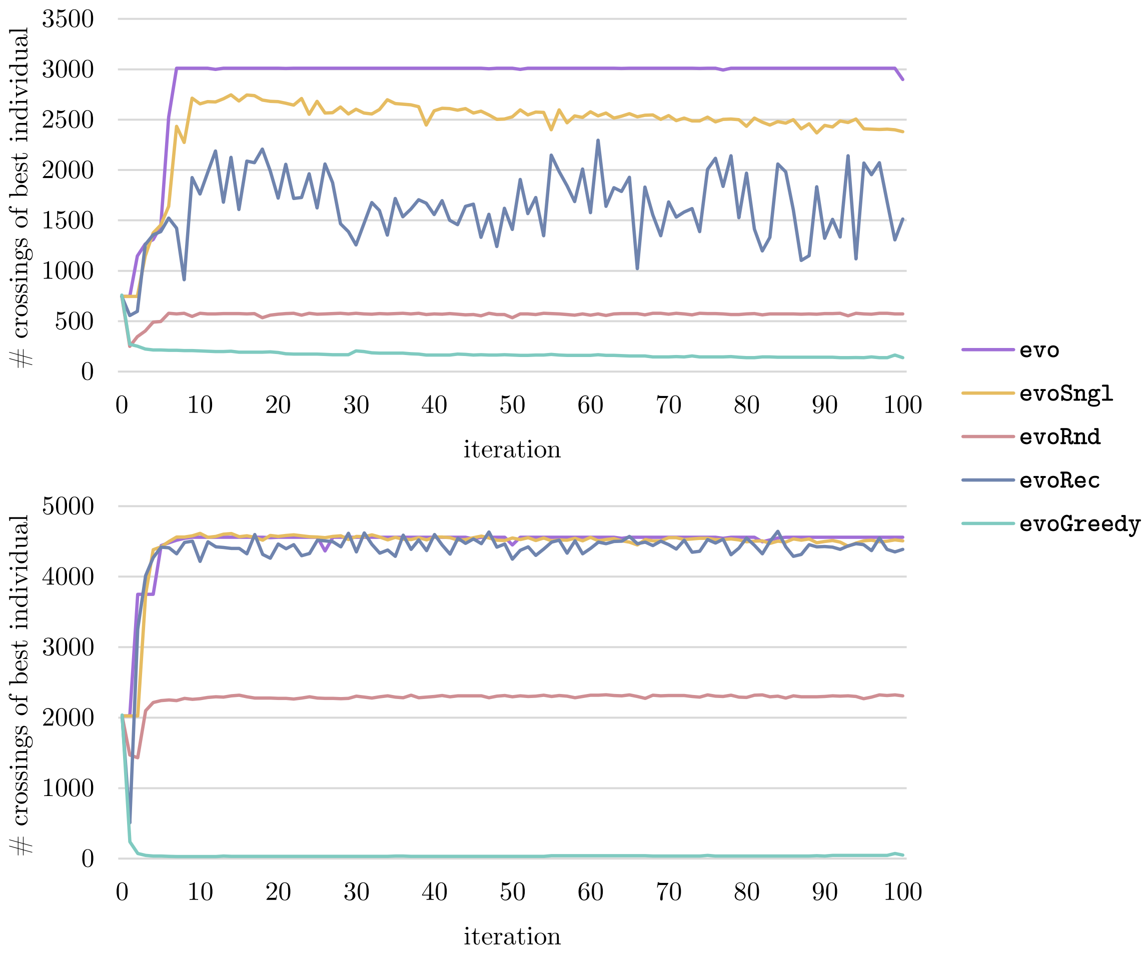


Figure 5.5.: The number of crossings of the best individual in the population in the first 100 iterations of the evolutionary algorithms on on a 1-planar graph with $n = 250$ and $k = 3$ on top and Q_8 with $k = 7$ at the bottom.

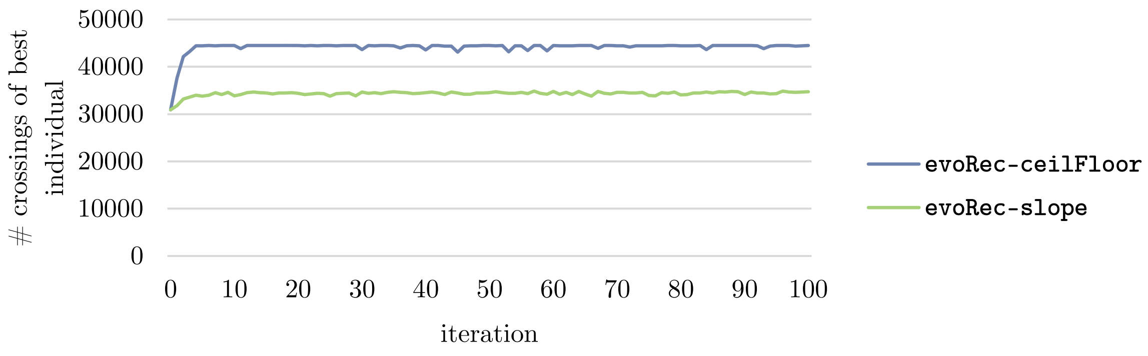


Figure 5.6.: The number of crossings of the best individual in the population in the first 100 iterations of `evoRec` with `ceilFloor` and `slope` on a random graph with $n = 250$, edge probability $p = 0.2$ and $k = 6$.

We now compare the two algorithms that by construction always optimise, `evoGreedy` and `greedyCombi`. The diagrams in Figure 5.5 also show that `evoGreedy` performed way better than the other evolutionary algorithms. We recall that `evoGreedy` does not use a recombination or recomputation operator but only uses greedy optimisation, `greedyVOO` and greedyEDO, and adapting mutation rates. In Section 5.1.4 we have seen that greedyCombi achieved better results than greedyAlt, which is now kind of a subalgorithm of evoGreedy. We are therefore interested whether the mutations and especially the adapting mutations rates can help to overcome this performance gap. Hence, we tested with several graph classes, namely with planar graphs with $n = 500$, K-trees with $K = 6$ and $n = 250$, the complete bipartite graph $K_{50,50}$ and the hypercube Q_7. We stopped evoGreedy when it did not find a better book drawing during 1000 iterations. To overcome the bias of the initial drawing, which, as we have seen above, can affect the greedy optimisation, we ensured that all three algorithms used the same heuristics for the initial drawings. The results are presented in Figure 5.7.

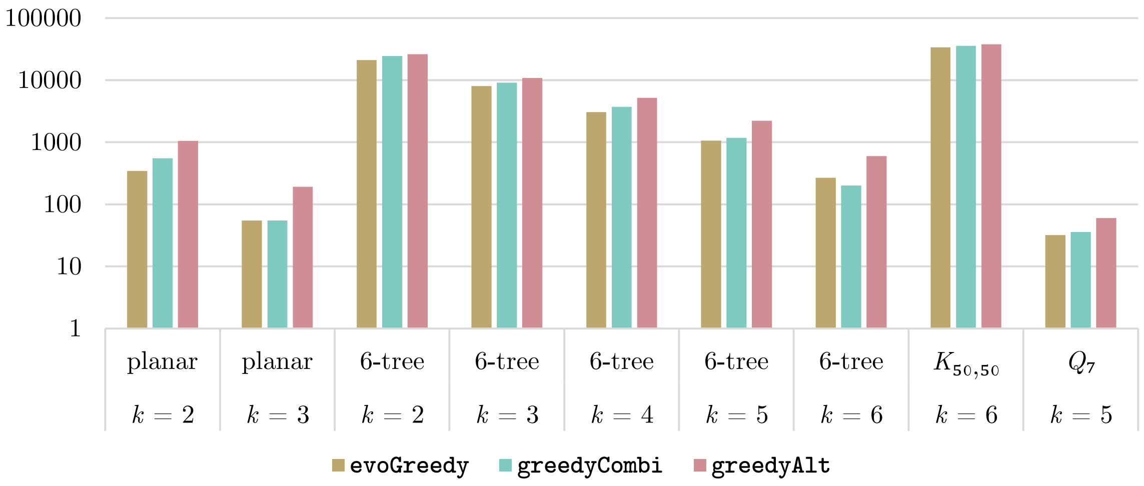


Figure 5.7.: Comparison of evoGreedy, greedyCombi and greedyAlt on different graphs.

We can observe that evoGreedy did not always achieve the best results. For example, for the 6-trees and $k = 6$ greedyCombi performed better. However, concerning most of the other cases, we can see that evoGreedy was the best algorithm. Since the diagram shows also greedyAlt, we know that the advantage of evoGreedy was obtained by the use of the mutations in the evolutionary framework. Regarding this, Figure 5.8 shows for several planar graphs that the adapting mutation rate indeed helped evoGreedy to come out of local minima. However, we have to state that the algorithm is more a random search, which converges fast to a local minimum, than a target-oriented search towards a global minimum.

5.4. Force-based optimisation

We now take a look at algorithms that use the concept of physical forces to optimise book drawings. They work in a similar way to force-directed graph drawing in the plane [Kob12]. The algorithms consider the whole drawing as physical systems and move the edges (vertices in the edge conflict graph) in the direction of some forces to improve the drawings. We first consider an approach to optimise the vertex order and then discuss an idea for force-based optimisation of the edge distribution.

5.4.1. Mean & median iteration vertex order

The following two algorithms are only in the broader sense force-based algorithms. Gasner and Koren [GK07] proposed them for circular drawings with one page to improve readability by decreasing the total edge length. We want to find out whether they can also improve a book drawings with more than one page in terms of crossings. Hence, after we

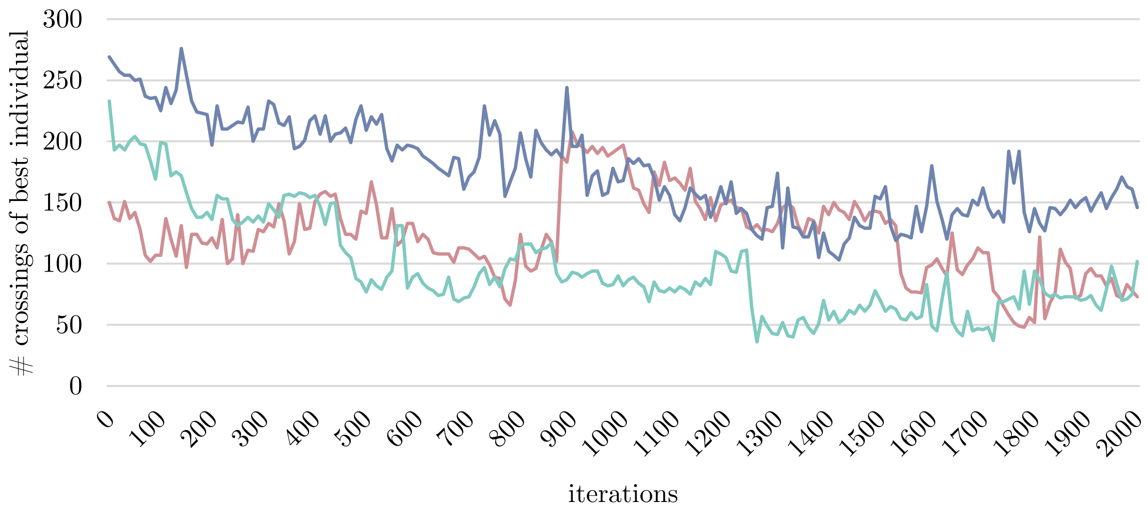


Figure 5.8.: Illustration of **evoGreedy** on several planar graphs with $n = 500$ and $k = 3$. The lines represent the number of crossings in the best individual of the population, measured after every 10 iterations.

describe how the algorithms work, we present the results of an experiment tackling this question.

The main idea of the two algorithms is to iteratively place the vertices at the mean or median position of their neighbours. So they need an initial vertex order. This vertex order is interpreted as uniformly distributed positions on the circle, just like in our previous illustrations of circular drawings (like Figure 4.21 or Figure 4.22). Now, for one vertex, the length of its incident edges is smallest, when it is at the barycenter of its neighbours. However, the vertices have to be placed on the circle. Thus, in an iterative procedure, one by one, the vertices are first placed on the barycenter of their neighbours and then projected back to the circle. Since the barycenter corresponds to the mean of the angular coordinates, this algorithm is called *mean iteration* (`meanIter`). To prevent the vertices from accumulating at one point and minimising the total edge length to zero, the vertices are again distributed uniformly as in the order of their angular position after ten iterations. Furthermore, in each iteration three randomly chosen vertices are not repositioned to function as anchors.

In a variation of this algorithm, the vertices are not placed at the mean but at the median of their neighbours independently for the x and y coordinate and then projected back to the circle. This algorithm is called *median iteration* (`medianIter`). Both algorithms run one iteration in $\mathcal{O}(n+m)$ time and Gasner and Koren suggested running $\mathcal{O}(n)$ iterations. We note that these two algorithms do not consider an edge distribution.

We tested the algorithms with planar graphs, hypercubes and random graphs with edge probability $p = 0.2$. For the initial book drawings we used different vertex order heuristics, namely random, `treeBFS`, `conCro` and `conGreedy`, and then `ceilFloor` to compute the edge distribution. We then ran the two iteration algorithms and after that created new edge distributions for the resulting book drawings, again with `ceilFloor`. The results are shown in Figure 5.9.

We report that the use of different initial vertex order heuristics did not have a significant effect on the book drawings created by `meanIter` and `medianIter`. Hence, we should see these two algorithms as two further vertex order heuristics and not as optimisation algorithms. Furthermore, the resulting book drawings of `meanIter` had mostly most crossings, apart from those with a random vertex order. `medianIter`did perform well on some graphs, however, still much worse than the best vertex order heuristic. Therefore, we conclude that the two heuristics are not a good choices for crossing minimisation in book drawings.

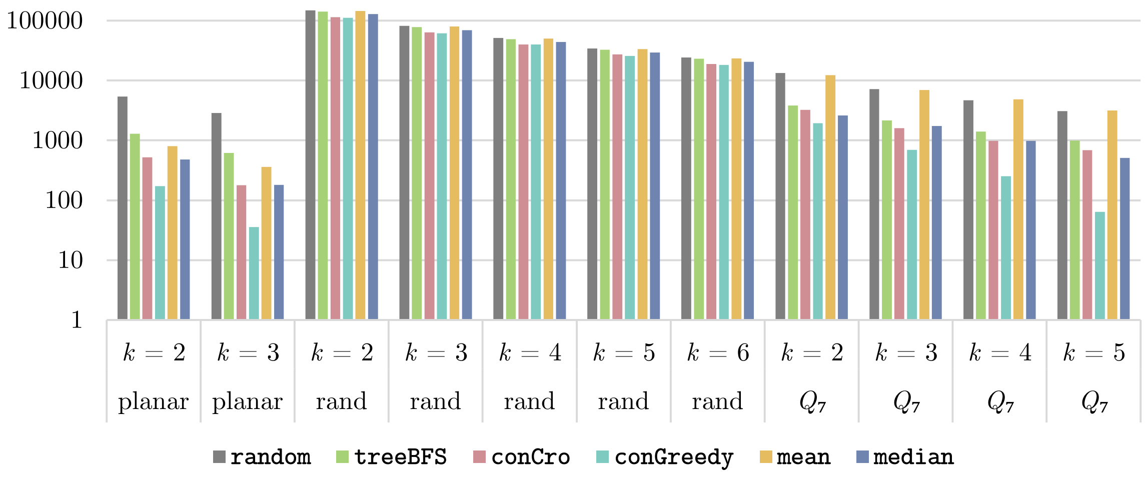


Figure 5.9.: Comparison of different vertex order heuristics in combination with `ceilFloor` and the two force-based vertex order optimisation algorithms on different graphs. *rand* stands for random graphs.

5.4.2. Force-based edge distribution

In our force-based edge distribution approach we model the edge distribution problem as physical system and then let the system reduce its energy potential itself. The original idea for the system is shown in Figure 5.10. Two edges that can cross for a given spine do not want to be on the same page. Hence, to reduce the energy of the system, they repel each other. Two edges that can not produce a crossing, on the other hand, may attract each other. So in our system we (conceptually) remove the pages and let the edges distribute themselves around the spine. When the system has converged, we partition them again based on their positions into pages. The hope is that conflicting edges distribute themselves far enough apart such that we do not place them on the same page in the last step.

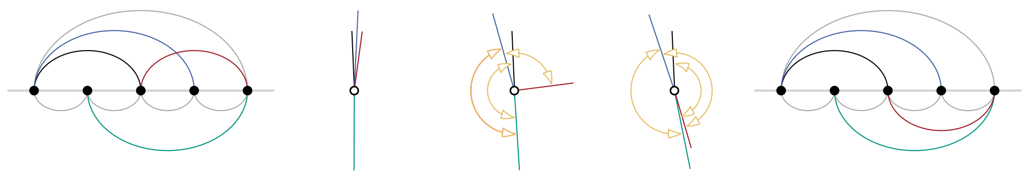


Figure 5.10.: In the middle the spine viewed from above, first with the initial distribution on the left and then two iterations of repelling forces. The resulting edge distribution is shown on the right.

Algorithm 6 describes the basic concept. The algorithms works iteratively and on the complement of the edge conflict graph G_c^c. Two vertices of G_c^c (original edges) are adjacent if they are not in conflict, i.e. if the corresponding edges of G can not produce a crossing for the given spine. Now, the algorithms works mostly like a normal force-directed algorithm. In each iteration two adjacent vertices attract each other with forces related to their distance (Algorithm 6, Lines 5 to 7). These forces are weak if the vertices are already close. If they are further apart also the forces gets stronger to group them. However, we set the forces to zero if the two vertices are very far apart, more precisely if they are distance is larger than the expected size of a page when we collect them again in the end. The idea behind this is that we are not interested in grouping all non-conflicting edges, only those that already close together. Besides the adjacent vertices, there are also repelling forces between non-adjacent vertex pairs (conflicting edge pairs, Lines 8 to 10). Such a repelling force is stronger if the two vertices are close, since we do not want to have them on the same page. With greater distance it gets less likely that the edges get placed on the same page and hence the repelling force gets weaker. After some iterations the system should converge to a local minimum of its energy. We then cluster the vertices (original edges of G) with a k-means algorithm (Line 13) and distribute each of the obtained clusters to one of the k pages (Line 14).

```
Algorithm: forceBasedEdgeDistribution
Data: graph $G$, fixed vertex order, initial edge distribution
Result: new edge distribution
1  $G_c \leftarrow$ edge conflict graph for $G$ and given vertex order
2  $G_c^c \leftarrow$ complement of edge conflict graph
3  position vertices of $G_c^c$ to $k$ uniformly distributed positions based on initial edge
   distribution
4  while system has not converged do
5      foreach edge $uv$ of $G_c^c$ do
6          compute attracting forces between $u$ and $v$ based on their positions
7          store forces for both
8      foreach pair $u, v$, $uv \notin E(G_c^c)$ do
9          compute repelling forces between $u$ and $v$ based on their positions
10         store forces for both
11     foreach vertex $u$ do
12         compute new position based on computed forces
13 cluster vertices based on their positions with $k$-means
14 distribute edges of $G$ to pages based on clustering
```

Algorithm 6: Algorithm concept for force-based edge distribution optimisation.

We use a normal k-means algorithm [Llo82], but set the initial means to specific positions. The goal of k-means is to partition the vertices into k clusters such that all vertices belong to the cluster with the nearest mean. The algorithms approximates this with an iterative refinement technique. First, k points are set as initial means. We use the initial positions of our pages for these points. Next, each vertex is assigned to the nearest point and thus k clusters created. Now, for each cluster the mean is computed and set as new reference point. So in the next round the vertices are now assigned to clustered with respect to these new points. This improves the clustering iteratively. The algorithm stops when the position of the means has converged, i.e. when they did not change by at least some ϵ. We observe that this solves our problem of collecting the vertices in our force-based system. If the forces created a good geometrical clustering, k-means may find a good clustering and thus also a good edge distribution.

In Figure 5.10, motivated by our illustrations of book drawings, the space where the vertices of G_c^c (the edges of G) can move is a cycle. However, in concept, this algorithm could use also another metric space. In fact, using a cycle has some drawbacks. For example, consider the case where $k = 3$ and already three groups have formed. Now, if a vertex lies between two groups which both repel it, but it would be better for it (in terms of minimal energy or avoided crossings) to be in the third group, then the forces may capture it between the two groups. This problem can occur between any two groups.

Hence, with higher k there are more of these bad zones on the cycle. Consequently, it is preferable to use a space of higher dimension, where more groups are needed to catch a vertex.

We made several attempts to successfully implement this algorithm. However, we faced the problem as we did with the force-based vertex order algorithms above. Even though our implementations worked in general, they were still often not better than the simple edge distribution heuristics. One big challenge in the implementation is the configuration of the parameters and force functions. We also tested a similar force-based clustering algorithm by Noah [Noa07, Noa08]. However, the produced book drawings using the algorithms standard settings were even worse than those of the other algorithms. Nevertheless, with regards to the matters discussed in the next section, we still think that the force-based edge distribution optimisation approach, if sophisticatedly implemented, has the potential to work successfully. Beyond this, it is also of interested whether vertex order and edge distribution optimisation can be combined in one force-based algorithm.

5.5. Further approaches

In this section we consider two further approaches for the edge distribution problem with fixed spine. We recall that this problem is equivalent to the Max k-Cut problem. Hence, any algorithm for Max k-Cut would work for the edge distribution problem as well. However, we will discuss below to further approaches, namely using graph partitioning and maximal independent sets. We also want to mention again that neural networks have been considered for $k = 2$ [CS96, HSM06, Wan08] as well as for the general k-page crossing minimisation problem [LRMCOdLLGM07] including the problem of finding a vertex order.

5.5.1. Graph partitioning & clustering

Both graph partitioning and graph clustering are related to the edge distribution problem. In the *k-way graph partitioning problem* for a graph G the problem is to partition G into k components with specific properties. A partitioning is defined as good if the number of edges running between different components, the *inter-edges*, is small. Furthermore, the problem also contains a balancing constraint on the sizes of the k partitions. On the other hand, in the *graph clustering problem* of a graph G the problem is to partition the graph into clusters such that a specific function gets optimised. The goal is again to have few inter-edges and also many *intra-edges*, i.e. edges within a cluster. However, the trivial clustering into one cluster is forbidden. There is also no balancing constraint. If we consider graph partitioning or clustering on the complement of the edge conflict graph G_c^c and regard a partition as one page, we can observe some similarities to the edge distribution problem. An edge in G_c^c indicates that two edges of G are not in conflict and, thus, can be distributed to the same page. Hence, if we have a high density in the partitions and thus many intra-edges, we get many edges (vertices of G_c^c) on the same page that are not in conflict. Furthermore, if we have less inter-edges, then there are more edges of G_c between partitions and we thereby avoid more conflicts. Thus, the goal of graph partitioning and clustering to have few inter-edges aligns with the goal of the edge distribution problem to avoid many crossings.

There are, however, also several differences between the problems. First, if we partition G_c^c with regard to a good edge distribution, we may find that there can be actually quite many inter-edges. Distributing two non-conflicting edges to different pages is not negative. If, on the other hand, two vertices in a partition are not adjacent, then we have a crossing in the corresponding book drawing. Hence, the actual goal of the edge distribution problem is to partition G_c^c such that the partitions are as dense as possible. We can thus state that problems have similar goals, however, not the same. Second, graph partitioning has a balancing constraint that, in contrast, the edge distribution problem has not. One page might contain many edges while another contains only few or even only one. A big difference to graph clustering is that graph clustering has no fixed number of clusters while the number of pages of the edge distribution problem is fixed.

For graph partitioning well crafted and tested algorithms exists [Sch13b]. Hence, one approach to tackle the edge distribution problem is to compute the complement of the conflict graph G_c^c and apply an existing graph partitioning algorithm with a weak balancing constraint. The problem that the edge distribution problem has by definition no balancing constraint does not yield that the pages of good book drawings actually are very unbalanced. In fact, it seems reasonable that, at least for some graphs and k, the pages are rather balanced. We therefore suggest that before this approach gets tested, it should be investigated how balanced or unbalanced the pages of good book drawings are. However, even if they are balanced there is still the discrepancy between the goals of the two problems.

The problem that graph clustering does not have a fixed number of clusters prevents a similar approach using a graph clustering algorithm instead of a graph partitioning algorithm. We want to note here that this can also be a problem in the force-based edge distribution approach above, since the described physical system does not have to goal to converge to k clusters. Hence, this should be introduced with additional forces into the system, for example with k anchors (or "magnets").

5.5.2. Maximal independent set

Another idea to compute an edge distribution is by using maximal independent sets. If we consider a page of a book drawing without crossings, we can observe that the corresponding vertices in the edge conflict graph G_c of the edges in this page form an independent set of G_c. A maximal independent set of G_c, in return, represents a page to which no further edge of G can be added without introducing any crossings.

In order to compute an edge distribution, the idea is to compute several maximal independent sets, merge those that contain few edges between each other and at the end obtain k partitions of G_c. The number of maximal independent sets should be at least k and the union of all the sets has to contain all vertices of G_c. Otherwise some edges of G would not be distributed to pages. The maximal independent sets can be computed with simple heuristics. Computing maximum independent sets might yield better result, it is however also $\mathcal{NP}$-hard. After the sets have been computed, merging those that span few edges between each other introduces few crossings in the resulting book drawing. Once only k sets remain, they can be used to obtain an edge distribution. One set yields one page and an edge corresponding to a vertex that is contained is several sets should be placed on the page where it produces fewest crossings.

To describe a full algorithm we have to solve several problems first. It is unclear how many maximal independents sets should be computed. Moreover, it has to be ensured that the maximal independent sets cover all vertices of G_c. One approach could be to start every new maximal independent set with an uncovered vertex. Furthermore, we also need an algorithm to merge the sets efficiently. If we considering the discussion about the graph partitioning based approach above, we observe that this approach with independent sets does not necessarily yield balanced pages. However, it might be reasonable that the pages are balanced and thus the sets should be maximal but also at most of size $\frac{n}{k}(1 + \epsilon)$, with $\epsilon > 0$ small. However, we might then end with an approach that is only another graph partitioning approach.

6. Summary and outlook

In this thesis we considered book drawings and book embeddings and in particular algorithms for the k-page crossing minimisation problem. In Chapter 2 we saw that book embeddings can be computed easily for trees and outerplanar graphs and that, however, it is $\mathcal{NP}$-hard to determine whether a planar graph has a 2-page book embedding. We outlined that therefore also the k-page crossing minimisation problem is $\mathcal{NP}$-hard. We then referred to results showing that book embeddings can be computed exactly for planar graphs with up to about 500 vertices in reasonable time and that, however, the limit for the k-page crossing minimisation problem can be reached already with 13 vertices. Furthermore, since counting crossings of book drawings is an important part of experiments and also of the evolutionary algorithms, we considered algorithms to solve this problem. We investigated two existing algorithms that run in $\Theta(m^2)$ time in the worst case. We therefore introduced a transformation of the problem to counting crossings in two-layered graphs and thus obtained a divide & conquer algorithm that runs in $\mathcal{O}(m \log m)$ time.

In Chapter 3 we presented heuristics that compute the vertex order of a book drawing and heuristics that compute an edge distribution of a book drawing as well as heuristics that compute both simultaneously. We proposed two new vertex order heuristics. `treeBFS` computes the vertex order based on the spanning tree found by a BFS search. `conGreedy` chooses vertices to place on the spine based on connectivity and then finds the best position greedily. We also introduced a new edge distribution heuristic, namely `earDecomp`, that uses an ear decomposition of the edge conflict graph. Furthermore, we combined the vertex order heuristics `conGreedy`, `smlDgrDFS`(also known as AVSDF), `randDFS` and `randBFS`with simultaneous greedy edge distribution. In particular the resulting full drawing heuristic `conGreedy+` is of interest, since the immediate edge distribution also effects the computed vertex order.

In Chapter 4 we investigated the performance of the heuristics, especially of our new heuristics. The lack of systematic approaches in the literature to evaluate the performance of heuristics motivated us to create an extensive test suite for the evaluation. We were in particular interested in differences of the heuristics' performance for different graph classes and graph densities. Therefore, we ran experiments with the heuristics and heuristic combinations on different graph classes, graph sizes and different number of pages. We evaluated them based on the achieved number of crossings. With our experiments we found out that `treeBFS` performs well on regular graphs with many automorphisms like Cartesian product of cycles and t-ary d-cubes. It is the best choice among vertex order heuristics with linear running time for these graph classes. However, on other graph classes it did not perform that well and ranked behind `randDFS` and `smlDgrDFS`. The edge distribution heuristic `earDecomp` is successful for sparse graphs, like planar graphs, and $k = 2$, but less for denser graphs or more pages. The full drawing heuristics `smlDgrDFS+`, `randDFS+` and `randBFS+` do not perform well. In our experiments they always produced more crossings than their corresponding vertex order heuristics in combination with a greedy edge distribution heuristic like `ceilFloor`. Our second new vertex order heuristic `conGreedy` as well as its full drawing heuristic `conGreedy+` are in most of the cases the best choice. In fact, only for t-ary d-cubes (also known as k-ary n-cubes) and sometimes for a large number of pages non of them ranked first. `conGreedy+` was very successful, especially on random graphs and a small number of pages as well as on K-trees and $k = K+1$. It also computes book embeddings of maximal outerplanar graphs. On random graphs and large k `conGreedy+`, used as vertex order heuristic in combination with `ceilFloor`, performed best. We can conclude that simultaneous distribution of the edges while computing a vertex order can produce better book drawings or at least improve the vertex order.

Concerning all heuristics, we saw that for graphs with low density the choice of the vertex order heuristic is more important than the choice of the edge distribution heuristic, at least if it is not `slope`. For example on 1-planar graphs, `conGreedy` with any edge distribution heuristic (except `slope`) is the better choice than `conCro`, which likewise is better than `smlDgrDFS`. For higher densities `slope` is mostly the best choice. However, similar to low densities, the choice of the vertex order heuristic is still important, as we observed on complete bipartite graphs. Furthermore, we observed that different heuristics performed well when comparing random graphs and structured graphs of the same densities. The heuristic rankings do also often change between different graph classes, especially for $k = 2$ or $k = 3$ and then also with growing k.

For future work on this topic, in order to evaluate a new heuristic, we suggest the created test suite around random graphs, graph classes like planar and 1-planar and graph classes with regular graphs and many automorphisms. K-trees are of interest since they are tightly coupled to book drawings via treewidth. However, complete graphs should only be considered with the conjecture in mind that `slope` always performs optimally. Graph classes like trees or outerplanar graphs, for which a book embedding can be computed in linear time, are not of interest. Furthermore, however, it is worth considering the heuristics' performances for different k, especially small ones, the pagenumber and the overall tendency. Considering a vertex order heuristic more emphasis should be put on graphs with lower density. For an edge distribution heuristic any density is of interest, however, in particular for dense graphs the challenge is to perform better than `slope`.

In Chapter 5, we considered optimisation algorithms, since the heuristics are often far from optimal. We tested for several approaches whether they really improve the results. We then compared them against each other. Greedy optimisation works well, but runs in local minima. The optimisation algorithm `greedyCombi` , which optimises vertex order and edge distribution simultaneously, achieves better results than `greedyAlt` , which alternates between the optimisation of vertex order and edge distribution. However, it also has higher running time, namely $\mathcal{O}(m^2n)$ for one round, in contrast to $\mathcal{O}(m^2)$ of `greedyAlt`. We also tested evolutionary algorithms, which try to come out of local minima. We found out that the classical crossover operator for reproduction does not perform well. Even simultaneous greedy optimisation can not always prevent the population from worsening. Furthermore, the approach using recomputation with an intermediate version of `randDFS` instead of recombination did not work well either. In both approaches it is also hard to configure the parameters and to choose the best operators. However, the concept of mutation and in particular adapting mutation rates of the evolutionary algorithm together with greedy optimisation worked well. Using the evolutionary framework with adapting mutation rates, greedy optimisation was able to climb out of local minima and achieve better results than standalone greedy optimisation. However, we suggest to search for further new optimisation algorithms. For example, we considered force-based approaches for optimisation. Further approaches could make use of the similarity between well known problems and the edge distribution problem. Hence, we suggested to consider graph partitioning algorithms and an approach based on maximal independent sets. However, these approaches need further investigation of the properties of good book drawings and then also sophisticated implementations.

Another goal of this thesis was to find out if we can combine the computation or optimisation of the vertex order and edge distribution efficiently. With `conGreedy+` and `greedyCombi` we found such algorithms. Both perform well and often better than algorithms that consider vertex order and edge distribution separately. However, both also have higher asymptotic running time. On the other hand, with `smlDgrDFS+`, `randDFS+` and `randBFS+` we saw that only computing vertex order and edge distribution simultaneously but still more or less independently does not yield better book drawings. We suggest to further investigate the idea of computing or optimising full book drawings and not only either vertex order or edge distribution. Furthermore, heuristics that compute full book drawings at once might be in particular of interest for certain graph classes, since algorithms that compute book embeddings of graphs often also compute vertex order and edge distribution simultaneously [BK79, Yan89, ABK15].

Bibliography

[AAFM$^+$12] B. Ábrego, O. Aichholzer, S. Fernández-Merchant, P. Ramos, and G. Salazar, "The 2-page crossing number of kn," in *Proceedings of the Twenty-eighth Annual Symposium on Computational Geometry*, ser. SoCG '12. New York, NY, USA: ACM, 2012, pp. 397–404. [Online]. Available: http://doi.acm.org/10.1145/2261250.2261310

[ABK15] M. J. Alam, F. J. Brandenburg, and S. G. Kobourov, "On the book thickness of 1-planar graphs," *CoRR*, vol. abs/1510.05891, 2015. [Online]. Available: http://arxiv.org/abs/1510.05891

[Alh05] M. Alhashem, "The book embedding of ordered sets," 2005.

[BB05] M. Baur and U. Brandes, "Crossing reduction in circular layouts," in *Graph-Theoretic Concepts in Computer Science*. Springer, 2005, pp. 332–343.

[BBKR15] M. A. Bekos, T. Bruckdorfer, M. Kaufmann, and C. Raftopoulou, "1-planar graphs have constant book thickness," in *Algorithms-ESA 2015*. Springer, 2015, pp. 130–141.

[BBS96] J. Branke, F. Bucher, and H. Schmeck, "Using genetic algorithms for drawing undirected graphs," in *The Third Nordic Workshop on Genetic Algorithms and their Applications*, 1996, pp. 193–206.

[BE14] M. J. Bannister and D. Eppstein, "Crossing minimization for 1-page and 2-page drawings of graphs with bounded treewidth," *CoRR*, vol. abs/1408.6321, 2014. [Online]. Available: http://arxiv.org/abs/1408.6321

[BGR14] M. A. Bekos, M. Gronemann, and C. N. Raftopoulou, "Two-Page Book Embeddings of 4-Planar Graphs," in *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), E. W. Mayr and N. Portier, Eds., vol. 25. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014, pp. 137–148. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2014/4453

[BJM02] W. Barth, M. Jünger, and P. Mutzel, "Simple and efficient bilayer cross counting," in *Graph Drawing*, ser. Lecture Notes in Computer Science, M. Goodrich and S. Kobourov, Eds. Springer Berlin Heidelberg, 2002, vol. 2528, pp. 130–141. [Online]. Available: http://dx.doi.org/10.1007/3-540-36151-0_13

[BK79] F. Bernhart and P. C. Kainen, "The book thickness of a graph," *Journal of Combinatorial Theory, Series B*, vol. 27, no. 3, pp. 320 – 331, 1979. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0095895679900212

[BKZ15] M. A. Bekos, M. Kaufmann, and C. Zielke, "The book embedding problem from a sat-solving perspective," *23rd International Symposium on Graph Drawing and Network Visualization*, 2015.

[BS84] J. F. Buss and P. W. Shor, "On the pagenumber of planar graphs," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing.* ACM, 1984, pp. 98–100.

[BS14] J. Balogh and G. Salazar, "Decompositions of permutations and book embeddings," 2014.

[BS15] ——, "Book embeddings of regular graphs," *SIAM Journal on Discrete Mathematics*, vol. 29, no. 2, pp. 811–822, 2015. [Online]. Available: http://dx.doi.org/10.1137/140961183

[BSV$^+$08] R. Bansal, K. Srivastava, K. Varshney, N. Sharma *et al.*, "An evolutionary algorithm for the 2-page crossing number problem," in *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on.* IEEE, 2008, pp. 1095–1102.

[CDD$^+$12] P. Clote, S. Dobrev, I. Dotu, E. Kranakis, D. Krizanc, and J. Urrutia, "On the page number of rna secondary structures with pseudoknots," *Journal of mathematical biology*, vol. 65, no. 6-7, pp. 1337–1357, 2012.

[CH13] J. Czap and D. Hudák, "On drawings and decompositions of 1-planar graphs," *the electronic journal of combinatorics*, vol. 20, no. 2, p. P54, 2013.

[Cim02] R. Cimikowski, "Algorithms for the fixed linear crossing number problem," *Discrete Applied Mathematics*, vol. 122, no. 1, pp. 93–115, 2002.

[Cim06] ——, "An analysis of some linear graph layout heuristics," *Journal of Heuristics*, vol. 12, no. 3, pp. 143–153, 2006. [Online]. Available: http://dx.doi.org/10.1007/s10732-006-4294-9

[CLR87] F. R. K. Chung, F. T. Leighton, and A. L. Rosenberg, "Embedding graphs in books: A layout problem with applications to vlsi design," *SIAM Journal on Algebraic Discrete Methods*, vol. 8, no. 1, pp. 33–58, 1987. [Online]. Available: http://dx.doi.org/10.1137/0608002

[CM07] R. Cimikowski and B. Mumey, "Approximating the fixed linear crossing number," *Discrete Applied Mathematics*, vol. 155, no. 17, pp. 2202 – 2210, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X07001485

[CS96] A. Cimikowski and P. Shope, "A neural-network algorithm for a graph layout problem," *Neural Networks, IEEE Transactions on*, vol. 7, no. 2, pp. 341–345, Mar 1996.

[DGGL11] E. Di Giacomo, F. Giordano, and G. Liotta, "Upward topological book embeddings of dags," *SIAM Journal on Discrete Mathematics*, vol. 25, no. 2, pp. 479–489, 2011.

[dKP12] E. de Klerk and D. V. Pasechnik, *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 581–595, 2012.

[dKPS13] E. de Klerk, D. V. Pasechnik, and G. Salazar, “Improved lower bounds on book crossing numbers of complete graphs,” *SIAM Journal on Discrete Mathematics*, vol. 27, no. 2, pp. 619–633, 2013.

[dKPS14] ——, “Book drawings of complete bipartite graphs,” *Discrete Applied Mathematics*, vol. 167, pp. 80 – 93, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X13004824

[DMP95] R. Diekmann, B. Monien, and R. Preis, “Using helpful sets to improve graph bisections,” *Interconnection networks and mapping and scheduling parallel computations*, vol. 21, pp. 57–73, 1995.

[DNEH13] H. R. Dehkordi, Q. Nguyen, P. Eades, and S.-H. Hong, *Circular graph drawings with large crossing angles.* Springer, 2013.

[DW04] V. Dujmović and D. R. Wood, “On linear layouts of graphs.” *Discrete Mathematics & Theoretical Computer Science*, vol. 6, no. 2, pp. 339–358, 2004.

[DW07] ——, “Graph treewidth and geometric thickness parameters,” *Discrete & Computational Geometry*, vol. 37, no. 4, pp. 641–670, 2007. [Online]. Available: http://dx.doi.org/10.1007/s00454-007-1318-7

[DW09] ——, “On the book thickness of k-trees,” *arXiv preprint arXiv:0911.4162*, 2009.

[EM99] H. Enomoto and M. S. Miyauchi, “Embedding graphs into a three page book with o(m log n) crossings of edges over the spine,” *SIAM J. Discret. Math.*, vol. 12, no. 3, pp. 337–341, Sep. 1999. [Online]. Available: http://dx.doi.org/10.1137/S0895480195280319

[ENO97] H. Enomoto, T. Nakamigawa, and K. Ota, “On the pagenumber of complete bipartite graphs,” *Journal of Combinatorial Theory, Series B*, vol. 71, no. 1, pp. 111 – 120, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0095895697917731

[FdFRv13] L. Faria, C. de Figueiredo, R. Richter, and I. vrt’o, “The same upper bound for both: The 2-page and the rectilinear crossing numbers of the n-cube,” in *Graph-Theoretic Concepts in Computer Science*, ser. Lecture Notes in Computer Science, A. Brandstädt, K. Jansen, and R. Reischuk, Eds. Springer Berlin Heidelberg, 2013, vol. 8165, pp. 249–260. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-45043-3_22

[Gan95] J. L. Ganley, “Stack and queue layouts of halin graphs,” 1995.

[GH01] J. L. Ganley and L. S. Heath, “The pagenumber of k-trees is $O(k)$,” *Discrete Applied Mathematics*, vol. 109, no. 3, pp. 215 – 221, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X00001785

[Gil59] E. N. Gilbert, “Random graphs,” *Ann. Math. Statist.*, vol. 30, no. 4, pp. 1141–1144, 12 1959. [Online]. Available: http://dx.doi.org/10.1214/aoms/1177706098

[GJMP80] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou, “The complexity of coloring circular arcs and chords,” *SIAM Journal on Algebraic Discrete Methods*, vol. 1, no. 2, pp. 216–227, 1980. [Online]. Available: http://dx.doi.org/10.1137/0601025

[GK07] E. Gansner and Y. Koren, “Improved circular layouts,” in *Graph Drawing*, ser. Lecture Notes in Computer Science, M. Kaufmann and D. Wagner, Eds. Springer Berlin Heidelberg, 2007, vol. 4372, pp. 386–398. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-70904-6_37

[GKK08] D. R. Gaur, R. Krishnamurti, and R. Kohli, “The capacitated max k-cut problem,” *Mathematical Programming*, vol. 115, no. 1, pp. 65–72, 2008.

[GLM$^+$15] F. Giordano, G. Liotta, T. Mchedlidze, A. Symvonis, and S. Whitesides, “Computing upward topological book embeddings of upward planar digraphs,” *Journal of Discrete Algorithms*, vol. 30, pp. 45 – 69, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570866714000914

[Gro15] M. Gronemann, “Algorithms for incremental planar graph drawing and two-page book embeddings,” Ph.D. dissertation, Universität zu Köln, 2015.

[Has09] T. Hasunuma, “Improved book-embeddings of incomplete hypercubes,” *Discrete Applied Mathematics*, vol. 157, no. 7, pp. 1423 – 1431, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X08004599

[Hea84] L. S. Heath, “Embedding planar graphs in seven pages,” in *Foundations of Computer Science, 1984. 25th Annual Symposium on*, Oct 1984, pp. 74–83.

[Hea85] ——, “Algorithms for embedding graphs in books,” 1985.

[Hea87] ——, “Embedding outerplanar graphs in small books,” *SIAM Journal on Algebraic Discrete Methods*, vol. 8, no. 2, pp. 198–218, 1987. [Online]. Available: http://dx.doi.org/10.1137/0608018

[HI92] L. S. Heath and S. Istrail, “The pagenumber of genus g graphs is o(g),” *J. ACM*, vol. 39, pp. 479–501, Jul. 1992. [Online]. Available: http://doi.acm.org/10.1145/146637.146643

[HN09] S.-H. Hong and H. Nagamochi, “Two-page book embedding and clustered graph planarity,” Technical Report 2009-004, Department of Applied Mathematics & Physics, Kyoto University, Tech. Rep., 2009.

[HNS05] H. He, M. Newton, and O. Sýkora, “Genetic algorithms for bipartite and outerplanar graph drawings are best!” *Communications of the 31st Conference on Current Trends in Theory and Practice of Informatics*, 2005.

[HS04] H. He and O. Sýkora, “New circular drawing algorithms,” *Conference Papers and Presentations (Computer Science)*, 2004.

[HSM06] H. He, O. Sýkora, and E. Makinen, “An improved neural network model for the two-page crossing number problem,” *Neural Networks, IEEE Transactions on*, vol. 17, no. 6, pp. 1642–1646, Nov 2006.

[HSM07] H. He, O. Sýkora, and E. Mäkinen, "Genetic algorithms for the 2-page book drawing problem of graphs," *Journal of heuristics*, vol. 13, no. 1, pp. 77–93, 2007.

[HSM10] H. He, A. Sălăgean, and E. Mäkinen, "One- and two-page crossing numbers for some types of graphs," *International Journal of Computer Mathematics*, vol. 87, no. 8, pp. 1667–1679, 2010. [Online]. Available: http://dx.doi.org/10.1080/00207160802524747

[HSMV15] H. He, A. Sălăgean, E. Mäkinen, and I. Vrt'o, "Various heuristic algorithms to minimise the two-page crossing numbers of graphs," *Open Computer Science*, vol. 5, no. 1, August 2015.

[HSSV06] H. He, O. Sýkora, A. Salagean, and I. Vrt'o, "Heuristic crossing minimisation algorithms for the two-page drawing problem," 2006.

[HSV05] H. He, O. Sýkora, and I. Vrt'o, "Crossing minimisation heuristics for 2-page drawings," *Electronic Notes in Discrete Mathematics*, vol. 22, pp. 527 – 534, 2005, 7th International Colloquium on Graph Theory. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1571065305052637

[HT73] J. Hopcroft and R. Tarjan, "Algorithm 447: Efficient algorithms for graph manipulation," *Commun. ACM*, vol. 16, no. 6, pp. 372–378, Jun. 1973. [Online]. Available: http://doi.acm.org/10.1145/362248.362272

[Jac89] G. Jacobson, "Space-efficient static trees and graphs," in *Foundations of Computer Science, 1989., 30th Annual Symposium on*, Oct 1989, pp. 549–554.

[Kai90] P. C. Kainen, "The book thickness of a graph. ii," *Congressus Numerantium*, vol. 71, pp. 121–132, 1990.

[KHT89] M. Konoe, K. Hagihara, and N. Tokura, "Page-number of hypercubes and cube-connected cycles," *Systems and Computers in Japan*, vol. 20, no. 4, pp. 34–47, 1989. [Online]. Available: http://dx.doi.org/10.1002/scj.4690200404

[KO07] P. C. Kainen and S. Overbay, "Extension of a theorem of whitney," *Applied Mathematics Letters*, vol. 20, no. 7, pp. 835 – 837, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893965906002783

[Kob12] S. G. Kobourov, "Force-directed drawing algorithms," in *Handbook of Graph Drawing and Visualization*, R. Tamasia, Ed. CRC Press, 2012, ch. 12.

[KRSZ02] N. Kapoor, M. Russell, I. Stojmenovic, and A. Y. Zomaya, "A genetic algorithm for finding the pagenumber of interconnection networks," *Journal of Parallel and Distributed Computing*, vol. 62, no. 2, pp. 267 – 283, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731501917897

[Llo82] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, Mar 1982.

[LRMCOdLLGM07] D. López-Rodríguez, E. Mérida-Casermeiro, J. Ortíz-de Lazcano-Lobato, and G. Galán-Marín, "K-pages graph drawing with multivalued neural networks," in *Artificial Neural Networks – ICANN 2007*, ser. Lecture Notes in Computer Science, J. de Sá, L. Alexandre, W. Duch, and D. Mandic, Eds. Springer Berlin Heidelberg, 2007, vol. 4669, pp. 816–825. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74695-9_84

[Mä88] E. Mäkinen, "On circular layouts," *International Journal of Computer Mathematics*, vol. 24, no. 1, pp. 29–37, 1988. [Online]. Available: http://dx.doi.org/10.1080/00207168808803629

[Mah13] B. Mahavir, "Optimal book embedding of the generalized petersen graph p (n, 2)," in *International Conference on Mathematical Computer Engineering-ICMCE*, 2013, p. 921.

[Mal94a] S. Malitz, "Genus g graphs have pagenumber $O(\sqrt{g})$," *Journal of Algorithms*, vol. 17, no. 1, pp. 85 – 109, 1994. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0196677484710285

[Mal94b] ——, "Graphs with E edges have pagenumber $O(\sqrt{E})$," *Journal of Algorithms*, vol. 17, no. 1, pp. 71 – 84, 1994. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0196677484710273

[McC12] J. McClintock, "Extremal graph theory for book-embeddings," Ph.D. dissertation, M. Sc. Thesis, University of Melbourne Department of Mathematics and Statistics, 2012.

[MNKF90] S. Masuda, K. Nakajima, T. Kashiwabara, and T. Fujisawa, "Crossing minimization in linear embeddings of graphs," *Computers, IEEE Transactions on*, vol. 39, no. 1, pp. 124–127, Jan 1990.

[MS09] T. Mchedlidze and A. Symvonis, "Crossing-optimal acyclic hamiltonian path completion and its application to upward topological book embeddings," in *WALCOM: Algorithms and Computation.* Springer, 2009, pp. 250–261.

[MS10] ——, "On ρ-constrained upward topological book embeddings," in *Graph Drawing: 17th International Symposium, GD 2009, Chicago, IL, USA, September 22-25, 2009. Revised Papers*, D. Eppstein and E. R. Gansner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 411–412. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-11805-0_40

[MWW88] D. J. Muder, M. L. Weaver, and D. B. West, "Pagenumber of complete bipartite graphs," *Journal of Graph Theory*, vol. 12, no. 4, pp. 469–489, 1988. [Online]. Available: http://dx.doi.org/10.1002/jgt.3190120403

[Noa07] A. Noack, "Energy models for graph clustering," *Journal of Graph Algorithms and Applications*, vol. 11, no. 2, pp. 453–480, 2007.

[Noa08] ——, "Modularity clustering is force-directed layout," *CoRR*, vol. abs/0807.4052, 2008. [Online]. Available: http://arxiv.org/abs/0807.4052

[NY04] H. Nagamochi and N. Yamada, "Counting edge crossings in a 2-layered drawing," *Information Processing Letters*, vol. 91, no. 5, pp. 221 – 225, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020019004001516

[Obr93] B. Obrenic, "Embedding de bruijn and shuffle-exchange graphs in five pages," *SIAM Journal on Discrete Mathematics*, vol. 6, no. 4, pp. 642–654, 1993.

[Ove98] S. B. Overbay, "Generalized book embeddings," Ph.D. dissertation, Colorado State University, 1998.

[Ove07] S. Overbay, "Graphs with small book thickness," *Missouri Journal of Mathematical Sciences*, vol. 19, no. 2, pp. 121–130, 2007.

[Pat13] M. Patrignani, *Planarity testing and embedding.* CRC press, 2013, ch. 1, pp. 1–42.

[PMH07] T. Poranen, E. Mäkinen, and H. He, "A simulated annealing algorithm for the 2-page crossing number problem," in *Proceedings of International Network Optimization Conference (INOC)*, 2007.

[Prü18] H. Prüfer, "Neuer Beweis eines Satzes über Permutationen," *Arch. Math. Phys*, vol. 27, no. 1918, pp. 742–744, 1918.

[RVM95] S. Rengarajan and C. E. Veni Madhavan, *Computing and Combinatorics: First Annual International Conference, COCOON '95 Xi'an, China, August 24–26, 1995 Proceedings.* Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, ch. Stack and queue number of 2-trees, pp. 203–212. [Online]. Available: http://dx.doi.org/10.1007/BFb0030834

[Sch13a] M. Schaefer, "The graph crossing number and its variants: a survey," *The electronic journal of combinatorics*, vol. 1000, pp. DS21–May, 2013.

[Sch13b] C. Schulz, "High quality graph partitioning," 2013.

[Sed77] R. Sedgewick, "Permutation generation methods," *ACM Comput. Surv.*, vol. 9, no. 2, pp. 137–164, Jun. 1977. [Online]. Available: http://doi.acm.org/10.1145/356689.356692

[SS12] P. Sanders and C. Schulz, "Distributed evolutionary graph partitioning." in *ALENEX*. SIAM, 2012, pp. 16–29.

[SSG11] D. Satsangi, K. Srivastava, and Gursaran, "A hybrid evolutionary algorithm for the page number minimization problem," in *Trends in Computer Science, Engineering and Information Technology*, ser. Communications in Computer and Information Science, D. Nagamalai, E. Renault, and M. Dhanuskodi, Eds. Springer Berlin Heidelberg, 2011, vol. 204, pp. 463–475. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24043-0_47

[SSS13] D. Satsangi, K. Srivastava, and G. Srivastava, "K-page crossing number minimization problem: An evaluation of heuristics and its solution using gesakp," *Memetic Computing*, vol. 5, no. 4, pp. 255–274, 2013. [Online]. Available: http://dx.doi.org/10.1007/s12293-013-0115-5

[ST06] J. M. Six and I. G. Tollis, "A framework and algorithms for circular drawings of graphs," *Journal of Discrete Algorithms*, vol. 4, no. 1, pp. 25 – 50, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570866705000031

[Stö88] E. Stöhr, "A trade-off between page number and page width of book embeddings of graphs," *Information and Computation*, vol. 79, no. 2, pp. 155 – 162, 1988. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0890540188900363

[TS06] Y. Tanaka and Y. Shibata, "On the pagenumber of trivalent cayley graphs," *Discrete Applied Mathematics*, vol. 154, no. 8, pp. 1279 – 1292, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X06000023

[TS10] ——, "On the pagenumber of the cube-connected cycles," *Mathematics in Computer Science*, vol. 3, no. 1, pp. 109–117, 2010. [Online]. Available: http://dx.doi.org/10.1007/s11786-009-0012-y

[Ung88] W. Unger, "On the k-colouring of circle-graphs," in *STACS 88: 5th Annual Symposium on Theoretical Aspects of Computer Science Bordeaux, France, February 11–13, 1988 Proceedings*, R. Cori and M. Wirsing, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 61–72. [Online]. Available: http://dx.doi.org/10.1007/BFb0035832

[Ung92] ——, *STACS 92: 9th Annual Symposium on Theoretical Aspects of Computer Science Cachan, France, February 13–15, 1992 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, ch. The complexity of colouring circle graphs, pp. 389–400. [Online]. Available: http://dx.doi.org/10.1007/3-540-55210-3_199

[VWY09] J. Vandenbussche, D. B. West, and G. Yu, "On the pagenumber of k-trees," *SIAM Journal on Discrete Mathematics*, vol. 23, no. 3, pp. 1455–1464, 2009. [Online]. Available: http://dx.doi.org/10.1137/080714208

[Wan08] J. Wang, "Hopfield neural network based on estimation of distribution for two-page crossing number problem," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 55, no. 8, pp. 797–801, Aug 2008.

[Wig82] A. Wigderson, "The complexity of the hamiltonian circuit problem for maximal planar graphs," Tech. Rep. EECS 198, Princeton University, USA, Tech. Rep., 1982.

[Wik16] Wikipedia. (2016) Algorithm to convert a Prüfer sequence into a tree. [Online]. Available: https://en.wikipedia.org/wiki/Pr{ü}fer_sequence#Algorithm_to_convert_a_Pr.C3.BCfer_sequence_into_a_tree

[Yan86] M. Yannakakis, "Four pages are necessary and sufficient for planar graphs," in *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '86. New York, NY, USA: ACM, 1986, pp. 104–108. [Online]. Available: http://doi.acm.org/10.1145/12130.12141

[Yan89] ——, "Embedding planar graphs in four pages," *Journal of Computer and System Sciences*, vol. 38, no. 1, pp. 36 – 67, 1989. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022000089900329

List of Figures

List of Algorithms

Appendix

A. Graph generation

In this section we briefly describe how we created the graphs for our experiments. The graphs of some graph class can be generated straight forward. These graph classes are complete graphs, complete bipartite graphs, Cartesian products of cycles, hypercubes and t-ary d-cubes. Graphs of the other graph classes can be generated in many different ways and thus caution is required. For examples, when generating trees we did not want to end up with trees that only have few vertices with very high degree and many leaves. Another aspect of graph generation is that, if the graphs are not shuffled, the algorithms could become biased to the graph generation algorithm if they consider vertices or their adjacent edges in the order they appear in the data structure. Hence, we also discuss how we shuffled the graphs before the experiments at the end of this section.

A.1. Trees

We generate trees using Prüfer sequences [Prü18]. Prüfer sequence can be used to describe labeled trees uniquely. To generate a random tree we generate a random labeled tree and then remove the labels. For a labeled tree with n vertices the Prüfer sequence a has length $n-2$ and each value $a[i]$ lies between 1 and n. Hence, we can start with generating such a sequence randomly and then convert it into a labeled tree. This is possible in linear time. A good description of the algorithm can be found at Wikipedia [Wik16].

A.2. Outerplanar graphs

To generate an outerplanar graph we start with a cycle C_n and then add edges in the inner face recursively, which works as follows. We take two adjacent vertices x, y of the cycle and randomly chose a third one z. We then add the edge to form the triangle xyz. If z is adjacent to either x or y on the cycle, then we have now only one new smaller cycles, which is not triangulated. Otherwise we have two such cycles. We then proceed recursively with these smaller cycles.

A.3. Maximal planar graphs

When generating maximal planar graphs, we hope to have a non-Hamiltonian graph and thus aim for many separating triangles. Therefore, as described in Algorithm 7, we start with a random *Apollonian network*, since it has by definition many separating triangles, and then apply a random number of *edge flips*. An Apollonian network is generated by a process of recursive subdivision of a triangle into three smaller triangles. We generate an Apollonian networks by randomly choosing the triangle of the current graph that gets subdivided next. An edge xy in a maximal planar graph is always on the boundary of two triangles xyu, xyz. An edge flip in a maximal planar graph takes an edge xy, removes it and in return adds the edge uz, connecting the two other vertices u and z of the two former triangles. We apply a random number of edge flips on randomly chosen edges of the Apollonian network to get a maximal planar graph.

```
Algorithm: generateMaxPlanarGraph
Data: number of vertices $n$
Result: maximal planar graph with $n$ vertices
1 $G \leftarrow$ create Apollonian network recursively and randomly
2 $r \leftarrow$ random number between 0 and $3n - 6$
3 for $i$ from 0 to $r$ do
4 |   flip random edge of $G$
5 return $G$
```

Algorithm 7: Algorithm to generate a random maximal planar graph.

A.4. Hamiltonian planar graphs

To generate Hamiltonian maximal planar graph, we take two maximal outerplanar graphs, identify their spines and then flip those edges that exist twice. Edges of the outer cycle of the second outerplanar graph are removed. The whole process is illustrated in Figure A.1. In (a), the dotted edge in the right outerplanar graph also exists in the left. Hence, as shown in Figure A.1 (b), we take the edge obtained by a flip of the dotted edge instead.

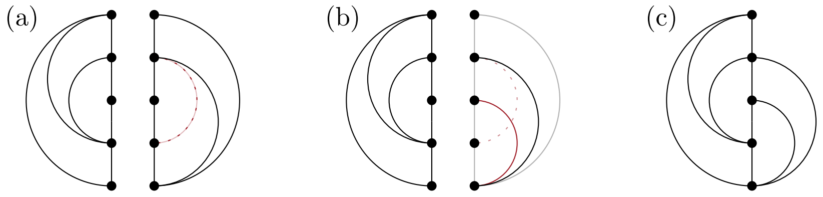


Figure A.1.: Starting with two maximal outerplanar graphs in (a), we remove the outer cycle of the second graph and also flip its edges that already appear in the first, shown in (b), and then merge the two graphs in (c).

A.5. 1-planar graphs

We generate 1-planar graphs using planar graphs. We process the vertices of a random maximal planar graph in a random order and check whether we can add an edge in a quadrangle, formed by two triangles. Figure A.2 shows a maximal planar graph, where the edge xz can not be added, since it already exists. However, the edge xy can be added and the edges uv and xy produce one crossing.

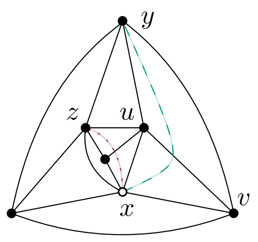


Figure A.2.: At x the edge xz can not be added, since it already exists, however, the edge xy can be added to get two edges crossing only each other.

Bekos et al. [BKZ15] produced 1-planar graphs by starting with planar triconnected quadrangulation and then augmented each face with two crossing edges.

A.6. K-trees

To generate a $K - tree$ with n vertices, we start with a complete graph with K vertices and then iteratively choose a random clique C of size K, add a vertex and connect it to all vertices of C. We stop once we have n vertices.

A.7. Random graphs

We use the Erdös-Rényi model to generate random graphs or more precisely the variant of Gilbert [Gil59]. To get a random graph $G(n, p)$ with n vertices, we take each edge of the complete graph K_n into the graph with probability p.

A.8. Shuffling

We shuffle each graph after the generation in order to prevent a bias of the algorithms towards the generation process. We represent graphs in our implementation with an array of vertices, where each vertex contains an array of its incident edges. To shuffle a graph we permute the vertex array and the edge array of each vertex.

B. Further experiment results

In this section we provide additional results to those presented in Chapter 4.

B.1. 1-planar graphs

In addition to Section 4.5, we report further results on 1-planar graphs. Figure B.3 shows the winning heuristic combination for 1-planar graphs with 200 graphs of sizes $n = 50, 100, 150$. As already discussed, `conGreedy-ceilFloor` performed (nearly) always best.

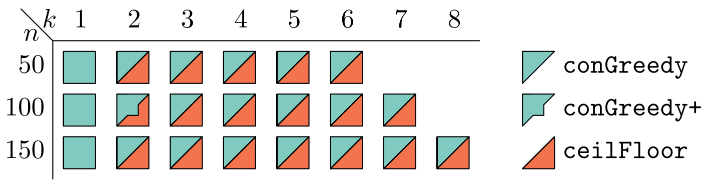


Figure B.3.: Winning heuristic on 1-planar graphs. Rows stop at k where average number of crossings was below 1 for the first time for the best heuristic.

In order to compare the performances on 1-planar graphs to the performances on planar graphs, we also tested the heuristics more extensively for $k = 2, 3$. Figures B.4 and B.5 show the best performances of all vertex order heuristics in combination with an edge distribution heuristic as well as the two best full drawing heuristics for $k = 2$ and 3, respectively.

Concerning $k = 2$ and in contrast to the results on planar graphs (see Section 4.4), we observe that `earDecomp` is no longer always the best choice. The connectivity based heuristics performed best with `ceilFloor`, and `randDFS` and `treeBFS` performed best with `circ`. Furthermore, we observe that for larger n `conGreedy+-ceilFloor` performs slightly better than `conGreedy-ceilFloor`.

Concerning $k = 3$, the main difference to the results on planar graphs is that on 1-planar graphs `conCro` performed way better than `smlDgrDFS`, while it was the opposite on maximal planar graphs. Furthermore, in contrast to $k = 2$, we see that again `conGreedy+` performs better alone than in combination with an edge distribution heuristic.

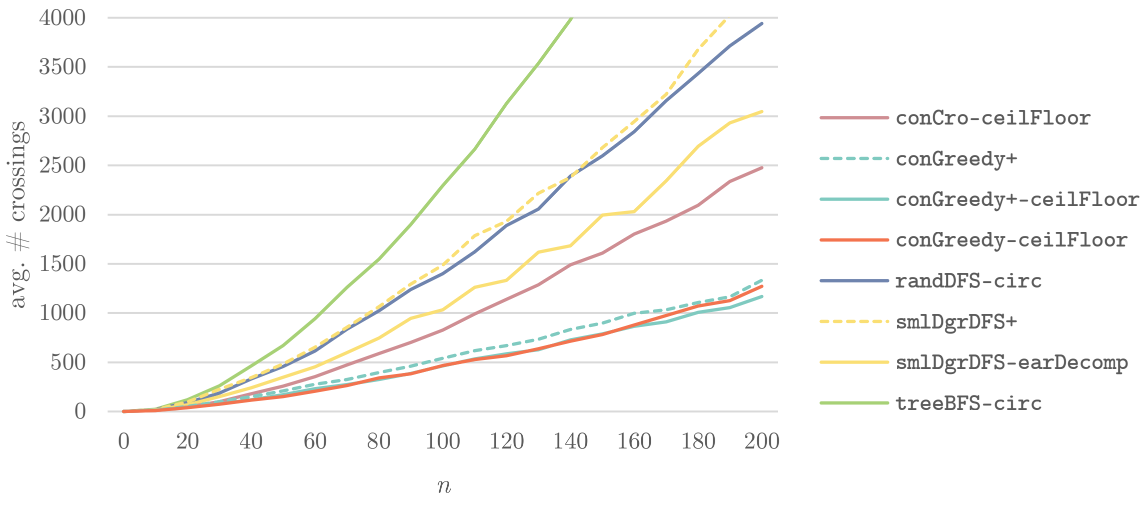


Figure B.4.: Average number of crossings achieved by selection of heuristic combinations for 1-planar graphs and two pages.

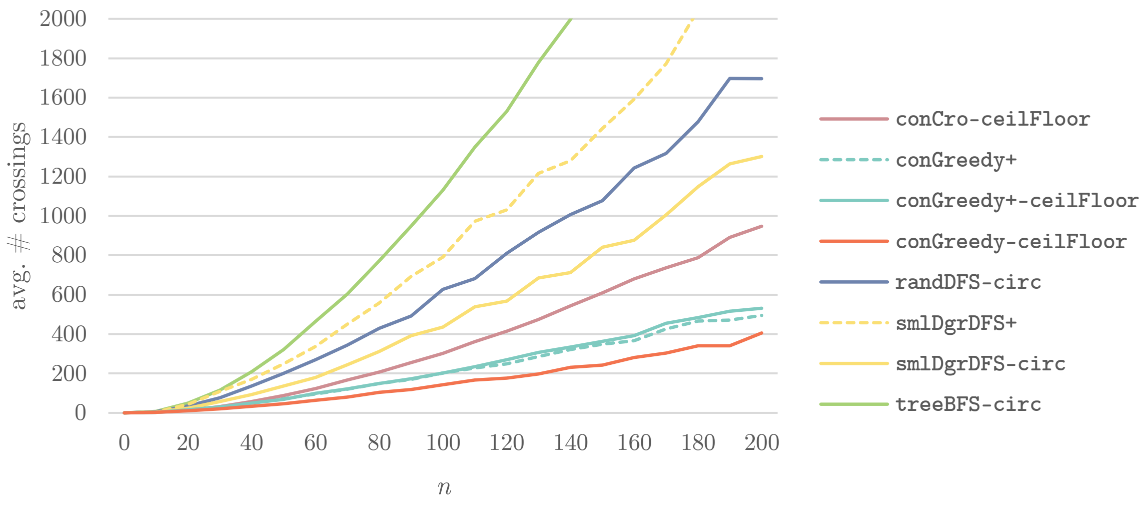


Figure B.5.: Average number of crossings achieved by selection of heuristic combinations for 1-planar graphs and three pages.

B.2. Hypercubes and t-ary d-cubes

Figure B.6 shows the winning heuristic combination on hypercubes for $k = 1$ to $d - 1 = \text{pn}(Q_d)$. We observe that `conGreedy` was always the best vertex order heuristic in combination with one or two of the greedy edge distribution heuristics. For $k = d - 2$ and $d - 1$, `ceilFloor` and `eLen` achieved the same results. This aligns with the observations in Section 4.7.

Figure B.7 shows the winning heuristic combination on t-ary d-cubes $Q(t, d)$ for $k = 1$ to 10. In Section 4.7 we have seen that `conCro` was often the best heuristic for $k = d + 1$. We observe now that it sometimes also for other k the best choice.

B.3. Random graphs

Figures B.8 and B.9 show the best performing heuristic combination in terms of average crossings on random graphs with linear and quadratic number of edges in terms of n, if

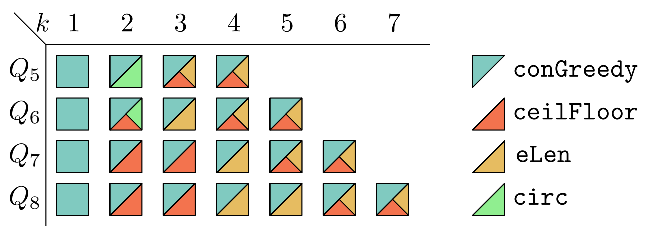


Figure B.6.: Winning heuristic on hypercubes Q_d. A split corner represents that two heuristics performed equally good.

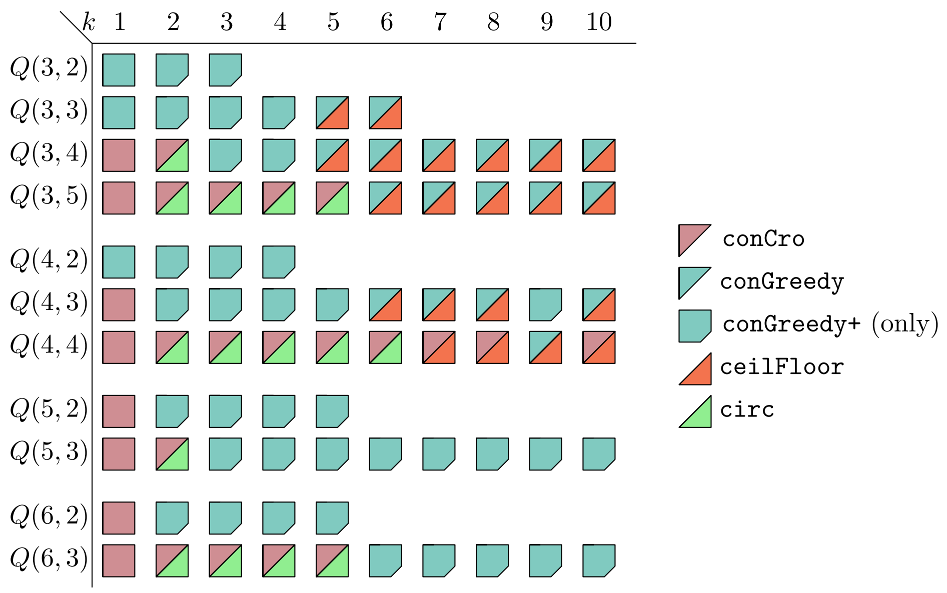


Figure B.7.: Winning heuristic on t-ary d-cubes $Q(t,d)$.

`conGreedy` and `conGreedy+` are excluded. Settings for the experiment were again 200 graphs for each k and a or p.

C. Tools and machines

We implemented our graphs, book drawings, algorithms and experiments in an object-oriented manner in Java, version 8. Our experiments were run on a private computer with Windows 7 (64bit), an Intel Core i5-4670 (quad-core, 3,4Ghz) and 8GB RAM. We generated our diagrams with Microsoft Excel 2013 and Inkscape. We used Ipe to create our figures.

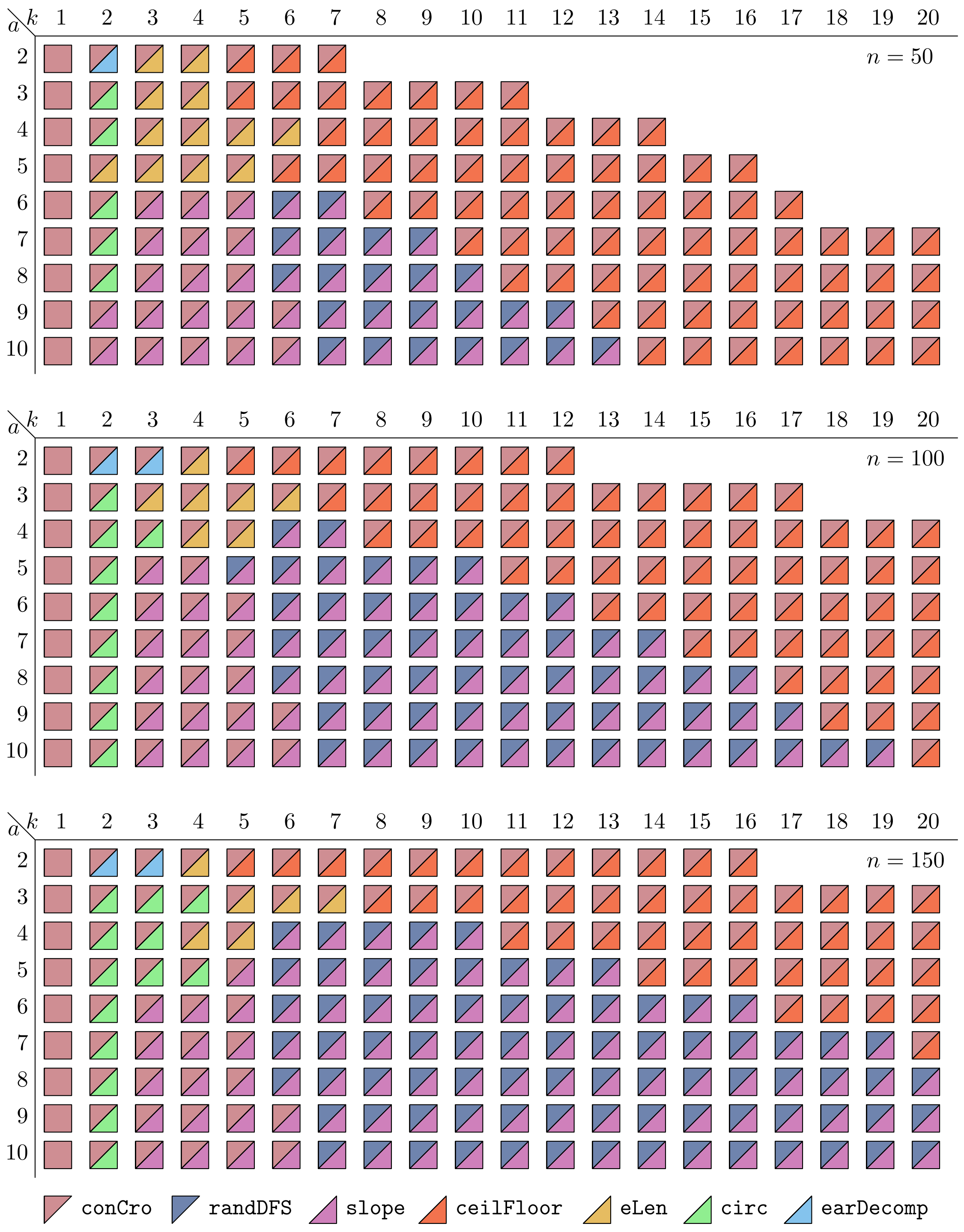


Figure B.8.: Winning heuristics in terms of average number of crossings on random graphs with linear number of edges and with `conGreedy` and `conGreedy+` excluded.

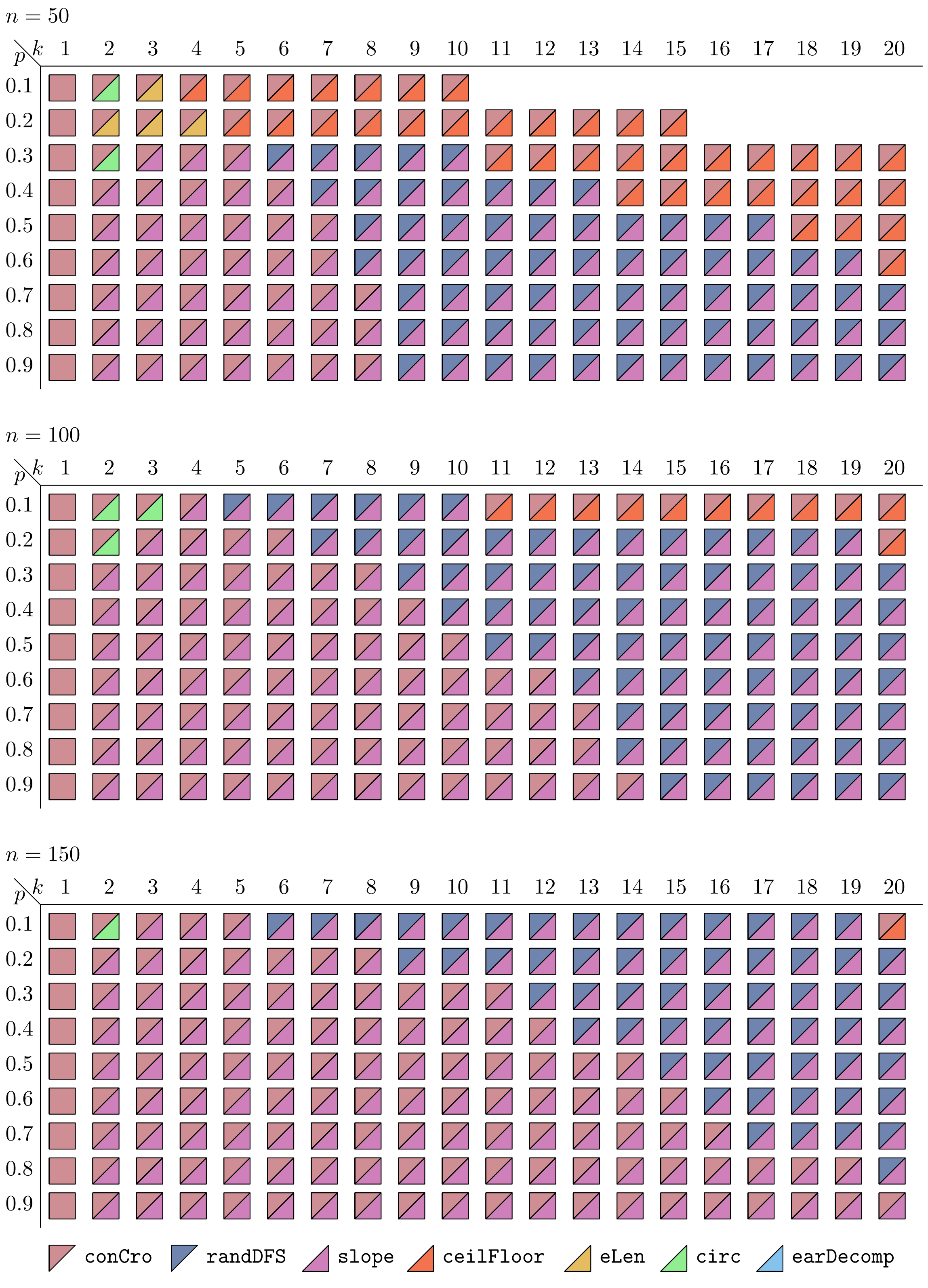


Figure B.9.: Winning heuristics in terms of average number of crossings on random graphs with quadratic number of edges and with `conGreedy` and `conGreedy+` excluded.