

Does Cubicity Help to Solve Problems?

T. Calamoneri

*Department of Computer Science
University of Rome "La Sapienza"
Via Salaria 113, 00198 Rome, Italy
e-mail: {calamo}@di.uniroma1.it*

Preface

The aim of this thesis is to give a survey of the main results obtained during my Ph.D. course on Computer Science at the University of Rome “La Sapienza”.

The original parts of Chapter 3 have been carried out with Prof. Rossella Petreschi (Sections 3.2 and 3.3) and Andrea Sterbini (Section ??) and they have been published in the Proceedings of *Cocoon '95* [28], of *ICTCS '95* [29] and of *GD '96* [33], respectively. In addition, an extended version of the work concerning drawing in parallel has been submitted to *IEEE Transactions on Parallel and Distributed Systems*; a journal version of the work dealing with 3D drawing has been accepted for publication on *Information Processing Letters*. Some results described in this chapter have not been published yet because they are very recent (Theorem 4 and Subsection 3.2.4).

Concerning Section 4.2 of Chapter 4, the results have been obtained in collaboration with Aythan Avior, Prof. Shimon Even, Prof. Ami Litman and Prof. Arnold L. Rosenberg. They have been published in the Proceedings of *SPAA '96* [8] and submitted to *Theory of Computing Systems (Math. Systems Theory)*. I studied this topic while visiting the Technion –Israel Institute of Technology– in Haifa. The work presented in Section 4.3 has been done with Prof. Rossella Petreschi; it has been presented at *ISCIS-XI* [31], and some of these results have been accepted for publication on *Information Processing Letters* [32].

Finally, the results in Chapter 5 (joint work with Paola Alimonti) appeared in the Proceedings of *WG '96* [5] and has been submitted to *Theoretical Computer Science*.

Acknowledgments

It is a pleasure to thank the people who supported me during the preparation of this thesis. Especially, I want to thank my supervisor and coauthor Prof. Rossella Petreschi for having introduced me to the fascinating world of research and having guided me with her precious experience throughout the last three years.

I would also like to thank all the other people I worked with during these years, my coauthors: Andrea, Aythan, Paola, and particularly Ami, Arny and Shimon, for the very interesting discussions we had and the suggestions they gave me; all Ph.D. students at the Computer Science Department, for making this place not only a stimulating environment for researching but also a pleasant place to work.

Moreover, my acknowledgment goes to the several people who commented on previous versions of this thesis: members of the internal thesis committee Prof. Giuseppe Italiano and Prof. Janos Körner, external reviewers Prof. Raymond Greenlaw and Prof. Yannis Manoussakis, and all the anonymous referees of my conference and journal submissions.

Finally, I would like to thank my husband Alessandro, for having lovely shared joys and delusions with me in realizing this work; my parents, for having given me the possibility to study, I dedicate this thesis to. But above all, I like to thank God, for giving me happiness, health, intelligence, and whatever I have.

Chapter 1

Introduction

Although algorithmic graph theory has been around since Euler's work [47], if not before, its development has been dramatic and revolutionary in the last thirty years. In fact, it is possible to model a lot of practical situations by means of graphs: typically, every system consisting of discrete states or sites and connections between. For this reason, graphs and their properties have been widely used for solving more and more problems of the most disparate subjects.

For instance, the psychologist Lewin proposed that the 'life space' of a person can be modeled by a planar graph, in which the faces represent the different environments [93].

In probability, a Markov chain is a graph in which events are vertices, a positive probability of direct succession of two events is an edge connecting the corresponding vertices [73].

Military problems like mining operations or destruction of targets may be traced back to the maximum weight closure problem [2].

Different processes such as manufacturing, currency exchanges, and translation of human resources into job requirements have as natural models networks, i.e. directed weighted graphs [48].

This interpretation is also applied to financial networks, in which vertices represent various equities such as stock, current deposits, certificates of deposit and so on, and (directed) edges represent various investment alternatives converting one type of equity into another one.

The previous applications are only a few examples of the search for the solution of practical problems by means of graph theory since the theory has been around so long and widely applied in vastly different fields. The

existence of numerous types of graphs and many basic notions that capture aspects of the structure of graphs is justified also as a consequence of the wide range of uses. Actually, very often a practical problem –when transformed into a graph problem– does not generate a general structure of graph, but a graph belonging to an opportunely restricted class. On the one hand, this is the reason why so many classes of graphs have been created and defined. On the other hand, the interest for the peculiar properties of each class is justified for an analogous but opposite reason. If the graph captures the peculiarities characterizing the starting structure, just these particular properties may be useful to compute the solution of the problem more efficiently than in the general case.

For example, consider a certain graph problem solvable by means of an algorithm, when a general graph is given in input. If the input graph is not general, but it is known to have some characterizing properties, it may happen that a more efficient algorithm –strongly utilizing the input’s properties– exists to solve the same problem. Obviously, this possibility would be very useful for the original practical problem, and this is why a great deal of research has been done in order to solve problems restricted to particular classes of graphs.

1.1 Motivation

In this thesis we focus on a particular class of graphs: bounded degree graphs. In particular, graphs having degree exactly 3 (*cubic graphs*) and bounded degree 3 (*at most cubic graphs*) are our main focus.

What we have just said about the practical interest for restricted classes of graphs is even more so for cubic graphs, because they are a natural model for a large number of real systems. For the sake of brevity, among them, only three practical applications, in which the graph model is cubic, are presented here. Our examples have been chosen in view of the fact that they cover different fields and are different from those examples treated in the next chapters of this thesis. For a survey, see [30].

Interaction of charged particles

Let \mathcal{P}_1 and \mathcal{P}_2 be two quantistic charged particles of high energy and let $P(x_1, y_1, t_1; x_2, y_2, t_2)$ be the probability of having the particles in x_1, y_1 at time t_1 and in x_2, y_2 at time t_2 , respectively. This probability can be expanded into an infinite series of events. The simplest event has the following structure: let $x(t)$ and $y(t)$ be the points of the trajectories reached by \mathcal{P}_1

and \mathcal{P}_2 at time t . The particles start from their initial state, in $x(t)$ and $y(t)$ they have an electromagnetic interaction, then they reach their final state. This event is represented by connecting x_1 and x_2 to x , y_1 and y_2 to y , and x to y , as shown in Fig. 1.1. Graph theoretical and physical considerations allow us to show that all possible events contributing to $P(x_1, y_1, t_1; x_2, y_2, t_2)$ can be modeled by general cubic graphs called Feynmann diagrams [24].

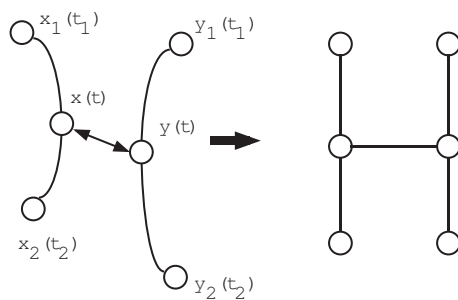


Figure 1.1: Interaction between two particles.

Mosaic problem

The Mosaic problem is related to biology, chemistry and graphics in general. It consists of covering the plane with copies of the same shaped polygon. The only regular polygons that can be used in a mosaic covering of the plane are hexagons, squares and triangles [106]. It is easy to see that hexagons induce a graph that is cubic except along the border of the external face (see Fig. 1.2). Therefore, an efficient drawing of cubic graphs can be equivalent to a perfect tiling of the plane (for a deep insight, see [64]).

Flowgraph model

One of the main goals of software engineering is to assess the quality of the developed software and somehow to measure it. A software metric, assigning a number to each piece of program text, will generally attempt to address one or more quality attributes in its assessment. These might include software reliability, testability, maintainability, and, in general, a kind of complexity of the software in some specific and restricted sense. The field of software metrication can be divided into those program measurement techniques that are largely textual and those that are more structural.

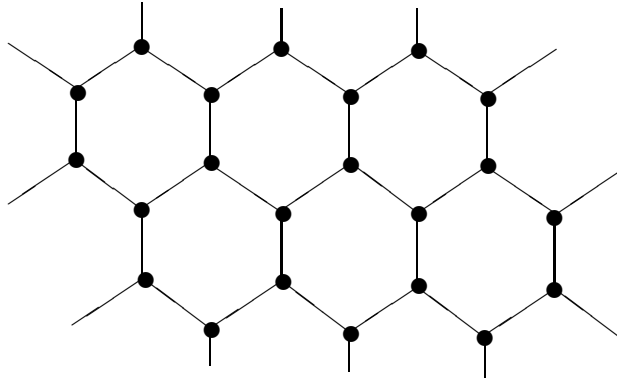


Figure 1.2: Hexagonal mosaic covering.

In concentrating on the latter, those metrics based on the hierarchical decomposition structure of the underlying flowgraphs can be emphasized. A *flowgraph* \mathcal{F} is a directed graph consisting of ‘process nodes,’ ‘decision nodes’ and *begin* and *end* nodes, satisfying the *flowgraph property*: every node lies on a path from *begin* to *end*. In [115] the further property that the underlying graph must be connected is introduced; removing the process nodes, and ignoring the edge orientation, each such flowgraph is associated with a unique underlying cubic graph (see Fig. 1.3).

This cubic flowgraph model, recently introduced, is shown to be useful in exploring a whole range of software metrics that are said to be ‘hierarchical’. In particular, the cubic flowgraph model leads to the discovery of certain inequalities among existing metrics, serves to provide a new characterization of the important class of prime flowgraphs (i.e. flowgraphs having no proper subflowgraphs), and points the way to an effective method for counting and enumerating the primes.

1.2 Thesis Overview and Statement of Results

The purpose of this thesis is to study how much and in which cases the strong properties of cubic (at most cubic) graphs can help to achieve better results than in the general case, from an algorithmic point of view.

In the following, we would like to describe the state of the art of this

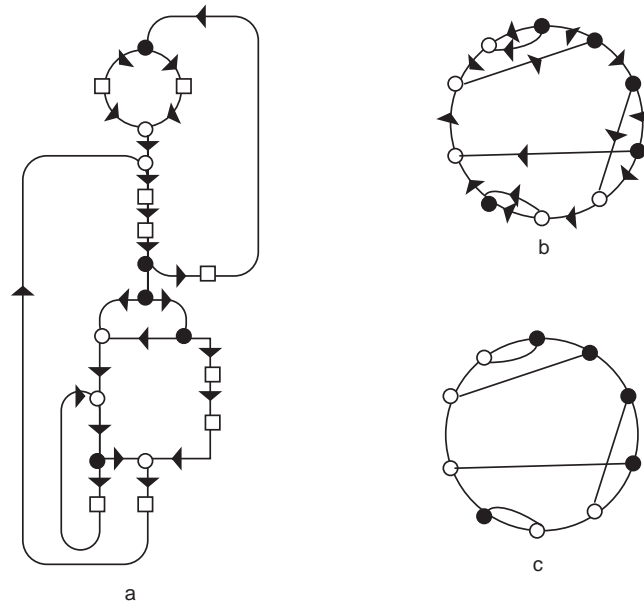


Figure 1.3: a. a flowgraph; b. cubic flowgraph after disregarding process nodes (\square); c. underlying (undirected) cubic graph.

general problem and to show some new results. In particular, Chapter 2 is devoted to a survey on cubic graphs: the main definitions and some properties are pointed out so that it is clear how extensive the research on this fascinating class of graphs is. The majority portion of the cited works deal with theoretical results on cubic graphs, but a lot of papers concerning algorithms running on cubic graphs are cited in the introducing sections of the other chapters. Indeed, in each of the further chapters, a different problem on cubic graphs is studied. Every chapter has a first section introducing the problem while the successive sections are devoted to showing some new results. Namely, Chapter 3 deals with some algorithms for drawing graphs on a grid. The problem is studied through many points of view: first, in Sections 3.2 and 3.3 the problem of drawing on 2-dimensional grid (at most) cubic graphs is solved by means of two algorithms. The first one is sequential and determines a compact drawing with few bends in optimal time. The second one runs on a PRAM and, though it is not computationally optimal, nevertheless it is quite important because it is one of the first parallel al-

gorithms solving the drawing problem. Perhaps it can serve as a starting point for the development of an optimal parallel algorithm. In Section ?? we have two extensions of the same problem: first of all, the grid is now three-dimensional, and then the drawn graphs are not only cubic but are also 2- and 3-colorable graphs. The same technique is applied to 4-colorable graphs, as well.

In Chapter 4 some fixed degree interconnection networks are considered and in Section 4.1 the importance of such networks in practice is pointed out. Section 4.2 is dedicated to butterfly networks and we solve the optimal layout problem: our first theorem states that at least $N^2 + o(N^2)$ area must be occupied by any layout of a butterfly network with N inputs and N outputs, and another one says that a layout of the same network fits in area $N^2 + o(N^2)$. These matching lower and upper bounds allow us to close (to within constant factor) the problem of finding the optimal layout of butterfly networks. In Section 4.3 Trivalent Cayley interconnection networks are introduced and it is shown how to solve some classical networks' problems on them. In particular, we again consider the layout problem and we prove that lower and upper bounds are of the same order of magnitude. Then, the optimal routing problem is studied; we solve it in optimal computational time.

In Chapter 5 some \mathcal{NP} Optimization (\mathcal{NPO}) problems are considered. In Section 5.1 a small survey about \mathcal{NPO} problems on cubic graphs is given. We have considered some \mathcal{NPO} problems and divided them according to the advantage they have from the point of view of cubicity: problems that are \mathcal{NP} -complete in general but that are polynomially solved for cubic graphs; \mathcal{NP} -complete optimization problems that are polynomial-time approximable for general graphs but for which better approximation ratios have been achieved for graphs of low degree; problems that for general graphs cannot be approximated within any constant approximation ratio in polynomial time but that have been shown to be in \mathcal{APX} (i.e. approximable within some constant factor in polynomial time) for bounded degree graphs.

In Section 5.2 the Minimum Independent Dominating Set Problem is studied in the special case when the input graph is cubic and, more generally, when it has bounded degree d . In both situations the best previously known results are improved.

Finally, some conclusions and open problems are drawn in Chapter 6. For the extent of the general subject, we have restricted our attention only to the problems considered in the previous three chapters.

Chapter 2

Cubic Graphs

2.1 Introduction

Let us begin with the most basic definitions to this thesis.

Definition 1 *A graph $\mathcal{G} = (V, E)$ is said to be regular of degree d or d -regular, if every vertex of \mathcal{G} has degree exactly equal to d . A graph \mathcal{G} is called cubic if it is regular of degree 3. When the degree of the vertices is less or equal to 3, \mathcal{G} is said to be an at most cubic graph.*

Since the number of vertices having odd degree must be even in every graph, a cubic graph $\mathcal{G} = (V, E)$ always has $|V| = n$ even, while no restriction for the parity of n is given if the graph is at most cubic. Also, it is easy to see that $|E| = m = \frac{3}{2}n$ if \mathcal{G} is cubic and trivially $m \leq \frac{3}{2}n$ if \mathcal{G} is at most cubic.

The first time cubic graphs appeared in the literature was in an informal manner in [138] and later more formally in [112] dealing with factorization of graphs and related coloring problems.

As emphasized in [62], besides their historical importance and their wide use as models of practical problems (cf. Chapter 1), there are at least two other fundamental facts justifying the study of cubic graphs:

- There are some transformation algorithms allowing us to obtain from a general graph \mathcal{G} a cubic graph \mathcal{G}' preserving certain structural properties of \mathcal{G} (cf. Section 2.2). By means of these transformations, it is possible to reduce problems on general graphs to problems on cubic

graphs in such a manner that the properties of cubic graphs can be utilized. An example is the coloration problem, for which the importance of cubic graphs descends from the following theorem:

Theorem 1 [106] *The face coloration of a graph in $k \geq 3$ colors can be reduced to the case of cubic graphs.*

- Cubic graphs seem to be a ‘boundary’ class of graphs in the sense that they are often the smallest class for which a problem is as difficult as it is in the general case. Indeed, it is evident that regular graphs of degree 0, 1 and 2 are quite simple and uninteresting for the most part. However, cubic graphs already constitute a complex class of graphs. On the other hand, there are cases in which restricting a problem to cubic graphs allows one to find a better solution than in the general case. Namely, given a graph problem having a cubic or at most cubic graph as input, its solution may be found more efficiently and more easily when the input graph is exactly cubic, while in other cases the cubicity does not give any further help to the computation of the solution (at least not so far). For example, consider the Chromatic Index Problem (CIP) and the Minimum Maximal Matching Problem (MMMP).

CIP: “Given a graph $\mathcal{G} = (V, E)$ and an integer K , can E be partitioned into disjoint sets $E_1, \dots, E_{\chi'}$ with $\chi' \leq K$ such that for $1 \leq i \leq \chi'$, no two edges in E_i share a common endpoint in \mathcal{G} ?”

MMMP: “Given a graph $\mathcal{G} = (V, E)$ and an integer K , does there exist a subset $E' \subseteq E$ with $|E'| \leq K$ s.t. E' is a maximal matching of \mathcal{G} ?”

The first problem is polynomially solvable for cubic graphs [80] while it is an \mathcal{NPO} problem for at most cubic graphs and more general graphs [58]. Furthermore, it is polynomial-time approximable within $4/3$ [155] and is not approximable in polynomial time within $4/3 - \epsilon$ for any $\epsilon > 0$ unless $\mathcal{P} = \mathcal{NP}$ [74].

The second problem is proved to be \mathcal{NP} -complete by a transformation from the Vertex Cover Problem for cubic graphs and it remains NP-complete for at most cubic planar graphs and for at most cubic bipartite graphs [58].

Therefore, while in the former case the cubicity helps to solve the problem, in the latter one there is no known difference between cubic, at most cubic and more general graphs.

In the following, a deep insight regarding cubic graphs can be found. In particular, in Section 2.2 we present some ways to construct cubic and at most cubic graphs. We show both a method to obtain a cubic graph having n vertices starting from a cubic graph having either $n + 2$ or $n - 2$ vertices, and two transformations to obtain an at most cubic graph starting from a general one. In Section 2.3 some properties and known results about different problems are enumerated.

2.2 Construction of Cubic Graphs

2.2.1 H-reduction and H-expansion

For any even value of n , there exists a regular graph having all vertices of degree d when $1 \leq d \leq n - 1$. This is a consequence of the fact that any complete graph having n vertices, for any even value of n , is the edge direct sum of $n - 1$ graphs of degree 1 [106]. In particular, for any even $n \geq 4$, there exist some cubic graphs having n vertices. If $n = 4$ there is only one such cubic graph: the tetrahedron K_4 (see Fig. 2.1.a). If $n = 6$ there are two of them: $K_{3,3}$ and the prism (Fig. 2.1.b and .c). When n increases, the number of all possible cubic graphs having n vertices also increases.

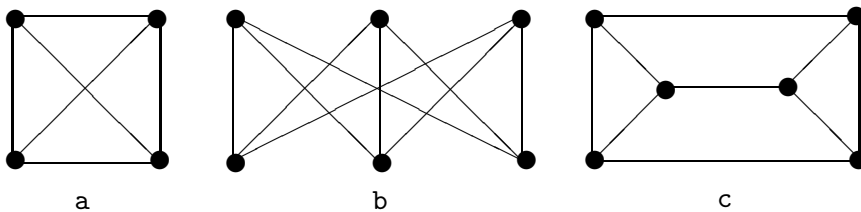


Figure 2.1: Three cubic graphs having 4 (a.) and 6 (b. and c.) vertices.

There exists a successive construction of cubic graphs due to E.L. Johnson [81] and based on a special graph consisting of 6 vertices and 5 edges as

shown in Fig. 2.2; this graph is called an *H-graph*. If we add the condition that edges $\{v_3, v_5\}$ and $\{v_2, v_4\}$ must not be present, then the H-graph is called *restricted*.

The following transformation makes it possible to generate any cubic graph having $n+2$ vertices from a cubic graph having n vertices (*H-expansion*) and vice-versa (*H-reduction*).

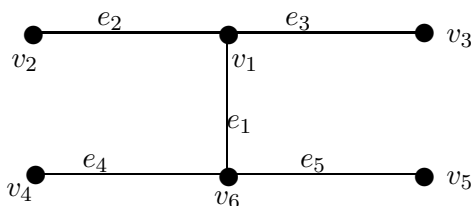


Figure 2.2: An H-graph.

Lemma 1 [106] *Any connected cubic graph \mathcal{G} having $n \geq 8$ vertices contains a restricted H-graph.*

To obtain a graph \mathcal{G}' with n vertices from \mathcal{G} with $n+2$ vertices by an *H-reduction* consists in finding a restricted H-graph in \mathcal{G} (cf. Lemma 1), in eliminating vertices v_1 and v_6 and their edges from \mathcal{G} and in replacing them either by the pair of edges $\{v_2, v_4\}, \{v_3, v_5\}$ or by $\{v_2, v_5\}, \{v_3, v_4\}$; in view of the topology of the restricted H-graph, one of these pairs is not in \mathcal{G} . The resulting graph is called \mathcal{G}' .

To this H-reduction, there exists an inverse operation, the *H-expansion*. Let $\mathcal{G} = (V, E)$ be a cubic graph on n vertices and let $e_1 = \{v_2, v_4\}$ and $e_2 = \{v_3, v_5\}$ be two arbitrary edges in \mathcal{G} where all endpoints are distinct. The *H-expansion* of \mathcal{G} with respect to e_1 and e_2 is obtained by eliminating e_1 and e_2 , and adding two vertices v_1 and v_6 with edges $\{v_6, v_1\}, \{v_6, v_2\}, \{v_6, v_5\}, \{v_1, v_3\}$ and $\{v_1, v_4\}$ or $\{v_6, v_1\}, \{v_6, v_2\}, \{v_6, v_3\}, \{v_1, v_4\}$ and $\{v_1, v_5\}$.

Theorem 2 [106] *For $n \geq 6$, every connected cubic graph having $n+2$ vertices is an H-expansion of a connected cubic graph having n vertices.*

H-expansion allows us to generate all cubic graphs starting from the tetrahedron.

In Fig. 2.3 we show the 8-vertex cubic graphs derived from a 6-vertex one, when edges $\{v_2, v_3\}$ and $\{v_4, v_5\}$ are removed.

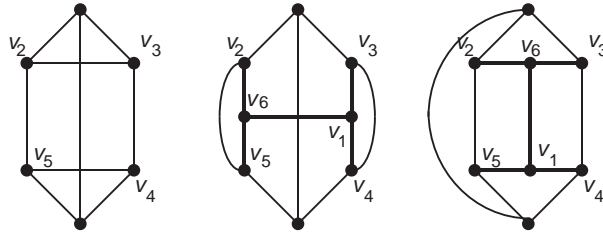


Figure 2.3: An example of H-expansion.

2.2.2 Transforming a Cubic Graph into Another

Given any cubic graph \mathcal{G} , it is possible to get a new cubic graph \mathcal{G}' by replacing any vertex of \mathcal{G} by a triangle and vice versa (Fig. 2.4.a). Similarly, a pair of multiple edges can be inserted into any edge or contracted to an edge (Fig. 2.4.b). If multiple edges are not allowed, instead of inserting a couple of multiple edges, we can insert a diamond (Fig. 2.4.d). Finally, a quadrilateral may be replaced by two edges or inserted along any two non-adjacent edges (Fig. 2.4.c). Observe that, if \mathcal{G} cannot be 3-edge colored, then replacing any vertex by a triangle, the resulting cubic graph still cannot be 3-edge colored. Similar reasonings holds if a pair of multiple edges are inserted into an edge. Analogously, neither contracting a triangle to a vertex nor contracting a pair of multiple edges affects 3-edge colorability. On the other hand, inserting a quadrilateral along two edges can affect the colorability. These considerations help to investigate the minimal 4-edge colorings of cubic graphs (cf. Subsection 2.3.1) [158].

2.2.3 Transformation of a General Graph into a Cubic Graph

In Section 2.1 the importance of transforming a general graph into a cubic (at most cubic) one was discussed. Now, we present two different transformations and sketch an application for each of them.

In both transformations, a general graph $\mathcal{G} = (V, E)$ having n vertices v_1, v_2, \dots, v_n of degree d_1, d_2, \dots, d_n is considered.

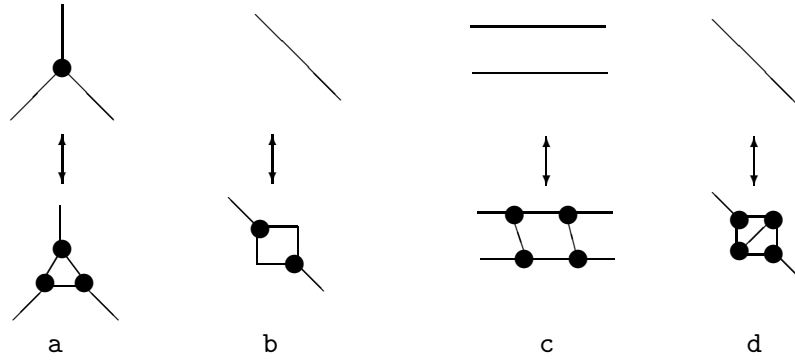


Figure 2.4: Transforming a cubic graph into another one.

Transformation One

This transformation generates an at most cubic graph if in \mathcal{G} there exists some vertices v_i of degree $d_i \leq 2$, a cubic graph otherwise.

Enclose each vertex v_i of degree $d_i \geq 4$ by a circle \mathcal{C}_{v_i} , small enough not to intersect any other circle \mathcal{C}_{v_j} , $i \neq j$. Select \mathcal{C}_{v_i} such that it intersects each of the edges e_k incident to v_i exactly once; denote this intersection by a_k , $k = 1, \dots, d_i$ (see Fig. 2.5). From \mathcal{G} and all the circles \mathcal{C}_{v_i} , we construct an at most cubic graph \mathcal{G}' by omitting all vertices v_i and those parts of the edges lying inside \mathcal{C}_{v_i} . We introduce as new vertices of \mathcal{G}' the points a_k and as new edges the arcs between a_k and a_{k+1} , if $k < d_i$, and between a_{d_i} and a_1 .

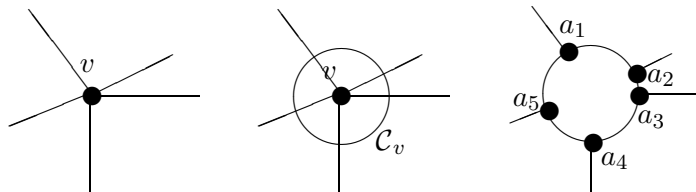


Figure 2.5: Scheme of Transformation 1.

Sometimes, this transformation is useful to solve problems on general

graphs by utilizing the properties of cubic graphs. An example is the case of the Four Color Problem: the conjecture that each map (i.e. planar graph) is 4-colorable is true if it is true for any planar cubic graph [106]. If cubic graph \mathcal{G}' is 4-colorable, then \mathcal{G} is also 4-colorable, and a coloration is obtained simply by shrinking each circle \mathcal{C}_{v_i} back to a single vertex v_i .

This transformation is also used to maintain a minimum spanning tree on-line under the operation of updating the cost of some edge in the graph in the classical work of Frederickson [54], later improved by the general technique described in [45]. In fact, this transformation is used very often when dealing with dynamic graphs, to solve the problems of edge connectivity, vertex connectivity and many others (as an example, see [55, 56, 120]). This proves the importance of finding efficiently optimal solutions of problems when restricted to cubic graphs.

Transformation Two

If graph \mathcal{G} is planar, it makes sense to consider its faces and the previous transformation increases the number of faces when passing from \mathcal{G} to \mathcal{G}' . It is possible to generate an at most cubic graph \mathcal{G}' from a general one \mathcal{G} without increasing the number of faces.

For each vertex v_i of degree $d_i \geq 4$, split it with respect to two non-adjointing faces f and f' having a corner at v_i . Each splitting increases the number of vertices by one, leaves the number of edges unchanged, and joins f and f' into a single face (see Fig 2.6).

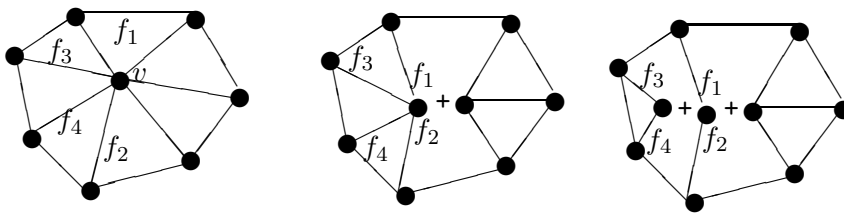


Figure 2.6: Scheme of Transformation 2.

This transformation is also related to the Four Color Problem. If the Four Color conjecture is not true, then there exists a graph with a minimal number of faces which cannot be four-colored. Such a graph is called *irreducible*. It

can easily be proved that an irreducible graph \mathcal{G} must have degree $d_i \geq 3$ for each vertex v_i . This transformation can be used to prove that an irreducible graph must be cubic, if it exists [106]. Indeed, if \mathcal{G} is irreducible and not cubic, then the transformation described can be applied. A face coloring of the split graph \mathcal{G}' leads to a coloring of the original graph \mathcal{G} . It follows that the irreducible graph is cubic.

2.3 Classical Graph Theory Results

Now we describe several general results about cubic graphs. The aim of this section is not to be an exhaustive list of results but only a selection of problems that seem interesting or useful to underline the importance of cubic graphs and the large amount of research that has been done in relation to them.

We have divided the material into different subsections in order to group the cited arguments according to some specific subjects.

2.3.1 Coloring Problems

The earliest equivalent formulation of the Four Color conjecture was proposed by Tait in 1878 [137] and states that the edges of a cubic, planar, bridgeless graph can be 3-colored. None of these hypotheses can be weakened. The very simple graph depicted in Fig. 2.7.a shows that having a bridgeless graph is necessary. The Petersen graph (see Fig. 2.7.b) is an example to show the necessity of planarity. Subdividing one edge of K_4 yields a graph with maximum degree 3 that cannot be 3-edge-colored. This shows that planarity is not renounceable. On the positive side it is straightforward to see that any graph with maximum degree 3 can be either 3- or 4-edge-colored. There is a whole class of cubic graphs that cannot be 3-edge-colored: those with bridges and certain non-planar graphs.

Non-trivial cubic graphs that are 4- but not 3-edge colored captured the fancy of Gardner and led him to introduce the term *snark* [57]. If we take any snark and apply to it the transformation described in Subsection 2.2.2, replacing any vertex by a triangle and inserting a pair of multiple edges into any edges or vice-versa, then the resulting cubic graph still cannot be 3-edge-colored. Thus we may assume that snarks have girth 4 or more. A quadrilateral in a snark may be replaced by two edges producing a smaller snark, but inserting a quadrilateral along two edges of a snark can effect

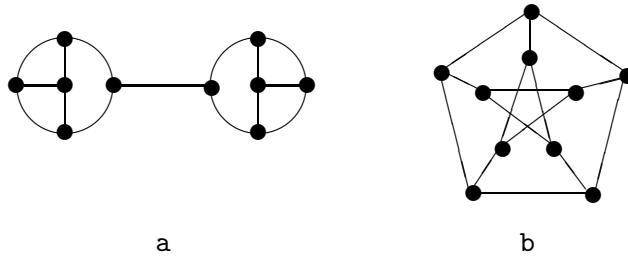


Figure 2.7: Two snarks.

the colorability. For example, a quadrilateral inserted along any two non-adjacent edges of the Petersen graph yields a 3-edge-colorable graph. The status of the quadrilaterals is thus somewhat different from that of multiple edges and triangles. Nonetheless, it has been usual to exclude quadrilaterals and require that snarks have girth 5 or more. It is also typical to require a snark to be cyclically 4-edge-connected; that is, deleting fewer than four edges does not disconnect the graph into two components each containing a cycle. In [158], the authors specify a collection of ‘prime’ snarks, and a list of basic constructions with which every snark can be built up from prime snarks.

The minimum number of colors needed to color the edges of $\mathcal{G} = (V, E)$ so that incident edges receive different colors is called *chromatic index* and denoted by $\chi'(\mathcal{G})$. It is a familiar result of Vizing [155] that $d \leq \chi'(\mathcal{G}) \leq d+1$, where d is the maximum degree of \mathcal{G} .

If $\chi'(\mathcal{G}) = d$, \mathcal{G} is said to be *class 1*, otherwise \mathcal{G} is *class 2*. In [4] the maximal number of edges in a class 1 subgraph of a cubic graph \mathcal{G} is studied. In other words, the authors show that one can 3-color at least 13/15 of the edges of an arbitrary cubic graph. For convenience let $c(\mathcal{G}) = \max\{|E(\mathcal{H})| : \mathcal{H} \text{ is a subgraph of } \mathcal{G} \text{ and } \mathcal{H} \text{ is class 1}\}$ and $\gamma(\mathcal{G}) = \frac{|c(\mathcal{G})|}{|E|}$. The Four Color conjecture is equivalent to: If \mathcal{G} is cubic, planar, bridgeless graph, then $\gamma(\mathcal{G}) = 1$. In [4] it is proved that if \mathcal{G} is cubic, then $\gamma(\mathcal{G}) \geq 13/15$ while, if \mathcal{G} is at most cubic, then $\gamma(\mathcal{G}) \geq 26/31$. Some other bounds are presented if \mathcal{G} is bridgeless and planar and if \mathcal{G} is 4-regular.

For other coloring problems see [52].

2.3.2 Matching Problems

In this subsection we survey some results about matching when the underlying graphs are cubic.

The deep link between the concepts of matching and alternating chain (i.e. a chain whose edges alternatively belong to the current matching) is well known. However, it is not widely known that the concept of alternating chain was introduced in 1891 by Petersen to prove that, in some cubic graphs, any linear factor can be modified in order to use a given edge of the graph [14]. In 1891 paper on the factorization of regular graphs [112], Petersen proved the following famous result: If \mathcal{G} is a cubic graph with at most 2 cut edges then \mathcal{G} has a perfect matching. Given an even subset T of vertices of a graph $\mathcal{G} = (V, E)$, a T -join is a set A of edges such that T is exactly the set of vertices of odd degree in the graph (V, A) . In [61] Petersen's classical result is generalized by showing that any cubic graph $\mathcal{G} = (V, E)$ with at most 1 cut edge has a T -join of cardinality less than or equal to $n/2$ for every even subset T of vertices.

Let the Reals be extended to include ∞ with $\infty > r$ for every Real number r . Given an extended Real number r , a property $\mathcal{P}(r)$ of graphs is *super-hereditary* if, whenever graph \mathcal{G} has property $\mathcal{P}(r)$ and \mathcal{H} is a subgraph of \mathcal{G} , then \mathcal{H} has property $\mathcal{P}(s)$ with $s \geq r$. In [72] it is proved that, given a super-hereditary property \mathcal{P} of graphs, if g is the smallest cardinality of the set of vertices of a graph with property \mathcal{P} and having one vertex of degree 2 and all others of degree 3, and if \mathcal{G} is a cubic graph with n vertices and property \mathcal{P} , then \mathcal{G} contains a matching with at least $\frac{1}{2}n(3g - 1)/(3g + 1)$ edges. This result has the corollary that every cubic graph with n vertices possesses a matching containing at least $7/16n$ edges.

An *induced matching* \mathcal{M} in a graph $\mathcal{G} = (V, E)$ is a matching such that no two edges of \mathcal{M} are joined by an edge of \mathcal{G} . The edge set of a cubic graph can always be partitioned into at most 10 subsets, each of which induces a matching in the graph [76]. This result is a special case of a general conjecture made by Erdős and Nešetřil [46]: for each $d \geq 3$, the edge set of a graph of maximum degree d can always be partitioned into at most $\lfloor 5d^2/4 \rfloor$ subsets each of which induces a matching.

Although the maximum matching problem in graphs dates back to 30's, with König's work, it was not earlier than 1965 when the first polynomial algorithm for general graphs was found by Edmonds [44]. Subsequently,

numerous more and more efficient algorithms were obtained, and today the fastest known algorithm is due to Micali and Vazirani [102, 154], who hold this record since 1980.

Dahlhaus and Karpinski [39] introduced a new interesting approach to explore the complexity behaviour of the perfect matching problem. In fact they proved that the existence and the construction problems for a perfect matching in general graphs are as difficult as (in fact, \mathcal{AC}^0 -equivalent to) the same problems in cubic graphs. In [9], the authors extend the results of [39] to the case of planar graphs: they prove that the perfect matching problem for planar graphs is \mathcal{NC} -equivalent to the same problem in planar cubic graphs. Moreover, they prove that the perfect matching problem for bipartite graphs is as difficult as the maximum weight perfect matching in cubic bipartite graphs.

The parallel complexity of the maximum matching problem is a famous open question. It is well known that there is an \mathcal{NC} reduction of the maximum matching problem to the perfect matching one [118], but even deciding whether a graph has any perfect matching, is also an open problem and it is unknown whether it belongs to \mathcal{NC} or not. If we restrict our attention to the maximal matching problem for cubic graphs, there is an algorithm [62] generating a maximal matching of a cubic graph in parallel. It runs in $O(\log^2 n)$ time using $\mathcal{M}(n)$ processors on a CREW PRAM, where $\mathcal{M}(n)$ denotes the number of processors required to multiply two $n \times n$ matrices in $O(\log n)$ time on a CREW PRAM.

It is also possible to use a new technique not considering alternating paths [129] that finds a perfect matching in a bipartite cubic graph in $O(\log^2 n)$ time using $O(n\alpha(n)/\log n)$ processors in the arbitrary CRCW PRAM model, where $\alpha(n)$ is the inverse Ackermann function. This result improves the processor bound of [92] that gave an \mathcal{NC} algorithm to find a perfect matching in bipartite d -regular graphs. Even though only a special case is solved, this method appears to be a new attractive way to solve the parallel maximum matching problem, since in general even the problem of finding a single augmenting path is not known to be in \mathcal{NC} . It is interesting to notice that this new technique appears naturally applicable to cubic graphs.

2.3.3 Counting Problems

Now, let us turn our attention to some results about cubic graphs regarding counting.

Read [122] proved that the number \mathcal{M} of labeled cubic graphs on n vertices is asymptotically

$$\mathcal{M} \sim \frac{(3n)!}{e^2(3/2n)!3^n 2^{5/2n}}.$$

If we consider the quotient between the number of all cubic graphs and the number of all Hamiltonian cubic graphs, it tends towards 1 as the number of vertices n tends towards ∞ . This means that almost all cubic graphs are Hamiltonian [124].

A *dominating set* of a graph $\mathcal{G} = (V, E)$, is a subset $S \subseteq V$, such that each $v \in V - S$ is adjacent to some vertex in S . The *dominating number* of \mathcal{G} , $D(\mathcal{G})$, is the cardinality of the smallest dominating set of \mathcal{G} . Trivially, if \mathcal{G} is cubic, then $D(\mathcal{G}) \geq n/4$, as each element of S is adjacent to at most 3 vertices of $V - S$. A conjecture of Koch and Perles that $D(\mathcal{G}) \leq 3n/8$ is proved in [121]. Both of these bound are tight. In [103] the dominating number of a random cubic graph is considered and it is shown that $D(\mathcal{G})$ almost surely satisfies $.263n \leq D(\mathcal{G}) \leq .312n$.

In [161] the problem of counting the labeled cubic graphs with given numbers of cycles of given fixed lengths is solved: the probability that a graph chosen at random from the cubic graphs with n vertices contains precisely t cycles of length l is asymptotic to $\frac{(2^l/2l)^t e^{-2^l/2l}}{t!}$ as $n \rightarrow \infty$ with t fixed. Furthermore, the probability that a random cubic graph with n vertices has girth at least $j \geq 4$ is asymptotic to $\exp(-\sum_{i=3}^{j-1} 2^i/2i)$ as $n \rightarrow \infty$ with j fixed. Finally, the expected number of cycles of length $j \geq 3$ in a random cubic graph with n vertices is asymptotic to $\frac{2^j}{2j}$ as $n \rightarrow \infty$.

2.3.4 Cycles Problems

Here we address some issues involving cubic graphs and cycles.

The problem of finding bounds on the length of a longest cycle in cubic graphs was first raised by Tait in [137], where he conjectured that all planar 3-connected cubic graphs are Hamiltonian. This conjecture was disproved by Tutte [148], by exposing the counterexample shown in Fig. 2.8.

Later, different authors constructed infinite families of planar 3-connected cubic graphs \mathcal{G} with n vertices, such that the longest cycle in \mathcal{G} is at most n^t , for various constants $t < 1$ [63, 65, 156, 157]. As a lower bound, in [10] it is shown that every planar 3-connected cubic graph \mathcal{G} has a cycle of length

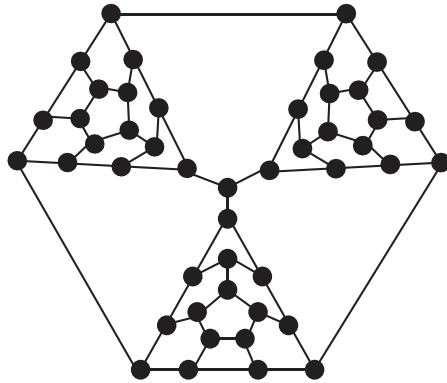


Figure 2.8: Tutte Graph disproving the conjecture of Tait.

at least $3 \log_2 n - 10$. For cubic graphs, which are not necessarily planar, the previous result can be improved by showing that every 2-connected cubic graph \mathcal{G} has a cycle of length at least $4 \log_2 n - 4 \log_2 \log_2 n - 20$ [26]. An example due to Lang and Walther [89] shows that this result is the best possible for the class of 2-connected cubic graphs. In [79] it is shown that, given a 3-connected cubic graph $\mathcal{G} = (V, E)$ with n vertices, for any $e_1, e_2 \in E$, e_1, e_2 are contained in a cycle of \mathcal{G} of length at least $n^t + 1$ for $t = \log_2(1 + \sqrt{5}) - 1 (\simeq 0.69)$.

An *even polyhedral decomposition* of a cubic graph \mathcal{G} is a set D of elementary cycles of even length such that every edge of \mathcal{G} belongs to exactly two cycles of D . Szekeres [136] remarked that if $\mathcal{G} = (V, E)$ is a cubic graph of chromatic index 3, then \mathcal{G} has an even polyhedral decomposition. Indeed, given any particular 3-edge-coloring, it is easy to check that the set of the 2-colored elementary cycles is an even polyhedral decomposition. This property does not characterize cubic graphs of chromatic index 3, as shown in [116]: there exists an infinite family of snarks (called *flower-snarks*) all having an even polyhedral decomposition.

Let \mathcal{G} be a connected graph which is not a tree; the *odd girth* (*even girth*) of \mathcal{G} denotes the length of a shortest odd (even) cycle in \mathcal{G} . If there is no odd (even) cycle in \mathcal{G} then the odd (even) girth of \mathcal{G} is taken as ∞ . Define the *girth* of \mathcal{G} to be $g = \min \{ \text{odd girth, even girth} \}$. Let $h = \max \{ \text{odd$

girth, even girth $\}$. Then (g, h) is called the *girth pair* of \mathcal{G} . A lot of papers have been written about the girth of cubic graphs. Here, only some results are reviewed.

Given a fixed integer g , it is interesting to determine the smallest cubic graph having girth g (for a survey on minimal regular graphs having girth g see [160]); in particular, if $g = 5$, the (unique) minimal cubic graph is the Petersen graph (see Fig. 2.9).

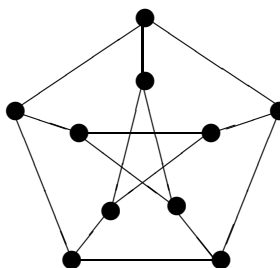


Figure 2.9: Petersen Graph.

Given $m - 1$ arbitrary non-negative integers $A_2, A_3, \dots, A_m, m \geq 2$, one can construct infinitely many mutually non isomorphic connected cubic graphs without loops, bridges and cut vertices such that each graph contains exactly A_l circuits with length l for every $l = 2, 3, \dots, m$; moreover, any two circuits having lengths not exceeding m are disjoint [128].

The smallest graph with given girth pair were found by Harary and Kovács [69]. Several infinite classes of such graphs are constructed in [70].

For other cycles problems, particularly tied to connectivity, see [146].

2.3.5 Subgraphs Problems

The problem of finding the extremal bipartite subgraphs of cubic graphs was resolved years ago. It is easy to see that the Petersen graph (Fig. 2.9) can be reduced to a bipartite graph by removing three of its 15 edges, and that removing only two edges is not enough. Similarly, the dodecahedron may be reduced to a bipartite graph by removing six of its 30 edges, and it is not enough to remove five edges. In both cases described, the starting cubic graphs are triangle-free and the obtained bipartite subgraphs have $4/5$ of the original edges. In [75] it is proved that any cubic triangle-free graph has

bipartite subgraphs with at least $4/5$ of the original edges, and this is best possible in light of the two examples mentioned above. Furthermore, every cubic graph not containing a tetrahedron has a bipartite subgraph with $7/9$ of the original edges, and this inequality is the best possible [132]. In [96] this latter result appears as a particular case of a more general formula dealing with k -colorable subgraphs of vertices with maximum degree d .

More generally, if \mathcal{G} is a cubic graph then \mathcal{G} contains a bipartite subgraph containing at least $7/9$ of the edges of \mathcal{G} . This is a special case of a more general result found by Staton [132] for d -regular graphs and their $(d - 1)$ -colorable subgraphs.

Let $B(\mathcal{G})$ be the edge set of a bipartite subgraph of a graph \mathcal{G} having the maximum possible number of edges. Let $b_k = \inf \{|B(\mathcal{G})|/|E(\mathcal{G})| \text{ such that } \mathcal{G} \text{ is a cubic graph with girth at least } k\}$. In [163] it is proved that $\lim_{k \rightarrow \infty} b_k \geq 6/7$.

For a graph \mathcal{G} let $k(\mathcal{G})$ denote the number of connected components of \mathcal{G} . Then, the *toughness* $\tau(\mathcal{G})$ is the maximum of $|S|/k(\mathcal{G} - S)$ taken over all sets S of vertices such that $k(\mathcal{G} - S) \geq 2$. A set S for which the maximum is achieved is called a *tough* set.

The parameter toughness was introduced by Chvátal [35], who also looked at the toughness of regular graphs and proved that $\frac{\kappa(\mathcal{G})}{d(\mathcal{G})} \leq \tau(\mathcal{G}) \leq \frac{\kappa(\mathcal{G})}{2}$, where $d(\mathcal{G})$ is the maximum degree and $\kappa(\mathcal{G})$ is the vertex connectivity of the graph. In [60] upper bounds on the toughness of a cubic graph are derived in terms of the independence number and coloring parameters.

In [133] Stewart proves that, given any planar graph, deciding whether it contains a cubic graph as a subgraph is \mathcal{NP} -complete.

Given a general graph \mathcal{G} having some costs on its edges, consider the following on-line update problem: a minimum spanning tree is to be maintained for an underlying graph, which is modified repeatedly by having the cost of an edge changed. In [54] a technique to deal with this update problem is presented. The structures used are designed to handle graphs in which no vertex has degree greater than three, therefore Frederickson uses the transformation described in Subsection 2.2.3 on \mathcal{G} . Then, the algorithm handles an at most cubic graph. Finally the solution is transformed into a solution for the general initial graph \mathcal{G} .

The results just described and many other on line algorithms have been widely improved by means of the general sparsification technique [45], but the contribution of this paper remains as a classical example of the transformation of a problem on general graphs to an analogous one on cubic graphs.

In this chapter we gave a look to cubic and at most cubic graphs, presented a description of several transformations from general graphs to cubic graphs, and then surveyed a number of interesting results having to do with cubic graphs. The results involved coloring, matching, counting, cycles, and subgraphs problems. The wide variety of such results illustrate the fundamental importance and basic nature of cubic graphs.

Chapter 3

Orthogonal Drawing

3.1 Introduction

The orthogonal grid drawing (for a complete annotated bibliography on this topic, see [41]) of a graph is a drawing such that edges are polygonal chains consisting of horizontal and vertical segments and vertices have integer coordinates. It follows that only graphs with maximum degree 4 can admit such a drawing. For a single example, see Fig. 3.1.

Generally speaking, an orthogonal graph drawing algorithm requires as input a combinatorial description of a graph and produces as output a feasible drawing of it according to a given graphic standard.

Within a fixed standard, a graph has infinitely many different drawings. However, in almost all data presentation applications, the usefulness of a drawing of a graph depends on its readability, that is the capacity of conveying the meaning of the diagram quickly and clearly. For example, a graph that appears congested with many concentrated line-crossings would not represent a visually appealing layout.

Since the investigation of orthogonal grid drawings came first, and is partially still, motivated by problems in circuit layout, the choice of the graphic standard should take it into account. For this reason, although there are a variety of important considerations in choosing one layout, the best understood, and perhaps the most desirable cost measure to minimize is layout area. The *area* of a layout is most naturally defined as the area of the “bounding-box” around the layout, and equals the product of its two sides.

Minimizing the area of a circuit on a chip is due, in part, to the fact

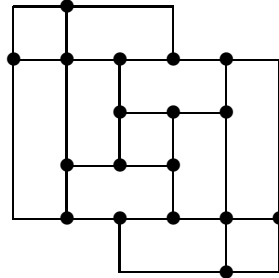


Figure 3.1: An example of orthogonal drawing.

that layouts which consume large amounts of chip area are more expensive to fabricate, less reliable and harder to test than layouts which consume smaller amounts of chip area.

Clearly, the area cannot be less than the number n of vertices of the graph. On the other hand, the area required for an n -vertex graph is no greater than $O(n^2)$ since we can draw vertices at equally spaced intervals along a line, and use a distinct horizontal track for each edge (see Fig. 3.2) [20].

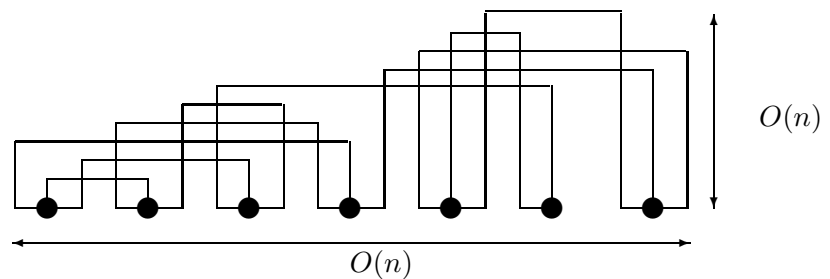


Figure 3.2: Every n vertex graph can be laid out in $O(n^2)$ area.

These bounds are independent of the structure of the graph and hold for all n vertex graphs. In general, however, the minimum area needed to lay a graph out depends on the graph.

Another layout-related issue that has been studied is minimizing propagation delay by decreasing *wire lengths*.

Since signals do not propagate instantaneously across wires, and the longer the wire, the longer the propagation delay, it is very important to draw graphs so that the longest edge is as short as possible. It is easy to see that $O(n)$ length for each edge is guaranteed if both sides of the drawing are $O(n)$ long.

This problem was studied in [111] for complete binary trees, and in [21] for arbitrary trees. Bhatt and Cosmadakis [19] showed that computing such a drawing for a tree is \mathcal{NP} -complete.

Another classical problem is constructing an orthogonal drawing with the minimum *number of bends* along the edges; Storer [135] conjectured the problem was \mathcal{NP} -hard. Nowadays, several algorithms solve this problem with the help of some constraints (cf. for example, [139] if the input graph has a fixed embedding and [42] if either a series-parallel graph or a biconnected planar cubic graph is given in input).

On the other hand, the minimization of the number of bends has several applications. It can be an approach to minimize other measures, such as area and edge-length. Furthermore, with communication by light or microwave, edge-length and area are relatively unimportant, while it costs a separate device each time it is desired to bend a wire.

For all of these reasons, a good graph drawing algorithm should try to balance the cost measures and get a ‘good’ drawing, i.e. with small area, short edges and few bends.

Although the investigation of orthogonal grid drawings arose in relation to circuit layout problems and VLSI applications, at present it has assumed an interest of its own as a theoretical problem. For this reason, we chose not to follow the formal model developed by Thompson [143, 144] that it is based on, and is consistent with, VLSI design rules established by Mead and Conway [100], but give an equivalent definition, turning more to graph theory than to networks.

Definition 2 *Let $\mathcal{G} = (V, E)$ be a graph, $n = |V|$ and $m = |E|$. An orthogonal drawing of \mathcal{G} is a drawing of \mathcal{G} in the plane such that all edges are drawn as sequences of horizontal and vertical segments. The edges are not allowed to overlap for any distance (although a vertical segment may cross*

a horizontal segment). In addition, the edges cannot cross vertices that are not their extremes. This drawing is said to be a drawing in the (rectangular) grid if all vertices are at integer coordinates.

If no crosses exist between any couple of edges, the drawing is called an embedding. Obviously, an embedding of \mathcal{G} exists only if \mathcal{G} is planar.

Definition 3 A point where the drawing of an edge changes its direction is called a bend of this edge. Let k be a non-negative integer. A k -bend drawing of a graph \mathcal{G} is an orthogonal drawing of \mathcal{G} in which every edge contains at most k bends.

A 0-bend drawing (embedding) is called straight-line drawing (embedding).

We say that a drawing (embedding) is *almost straight-line* if no edge contains any bend except one edge, having only one bend.

The four directions on the grid with respect to each vertex are distinguished by labels from the set { *Left, Right, Up, Down* }. We call *free* a direction with respect to a vertex if no edge is present on it.

In the following sections we will deal with graph drawing in some of its different aspects: first, in Sections 3.2 and 3.3 the problem of drawing on 2-dimensional grid (at most) cubic graphs is solved by means of two algorithms. The first one is sequential and determines a compact drawing with few bends in optimal time. The second one runs on a PRAM and, although it is computationally not optimal, nevertheless it is quite important because it is one of the first parallel algorithms solving the drawing problem. In Section ?? two extensions of the same problem are presented: first of all, a three-dimensional approach is introduced, when the graphs to be drawn are not only cubic but all 2- and 3-colorable graphs. The same technique is also applied to 4-colorable graphs.

It is easy to see that every algorithm drawing a cubic graph also works for an at most cubic graph; for this reason, in the next two sections, we will refer to cubic and at most cubic graphs, sinonimously.

3.2 A Sequential Algorithm to Orthogonally Draw Cubic Graphs

3.2.1 Introduction

Several results regarding orthogonal drawings of cubic graphs have appeared in the literature just in the last years, both from a theoretical point of view [42, 94] and from a more algorithmic one [23, 85, 95, 109]. Table 3.1 summarizes some of these results, together with those ones presented in this and in the next section.

In this section we present an algorithm that constructs an orthogonal drawing of a graph \mathcal{G} with degree at most three in which each edge has at most 1-bend, the total number of bends is $\leq n/2 + 1$, and the area is $\leq (n/2)^2$.

The algorithm is divided into three phases: during the first one the input graph is divided into its biconnected components; during the second one an algorithm drawing biconnected graphs (*BiconnCub*) is run on each component; finally, all components, that lie on different pieces of grid, are collected together onto a unique grid (*ConnCub*) and the final drawing is given in output.

So, we match the best known results in the literature; furthermore, we do not require any limitations either for planarity or biconnectivity. To the best of my knowledge the only paper dealing with such general graphs is [23], having upper bounds larger than these. In the next subsections the algorithm is first described in detail and then practically compared with the other ones cited by schematically presenting the performances of the algorithms (area, number of bends and computational time). Before doing that, we will present some definitions and address some considerations about the number of bends of an orthogonal drawing. Namely, we will discuss the lower bound problem.

Let us begin with a well known definition from graph theory.

Definition 4 [51] *Given a biconnected graph $\mathcal{G} = (V, E)$, an st-numbering is a numbering of the vertices $v_1, v_2, \dots, v_n \in V$ such that $\{v_1, v_n\} \in E$ and every vertex $v_i (1 < i < n)$ has edges to two vertices v_k and v_l for some k and l , with $1 \leq k < i < l \leq n$.*

Theorem 3 [51] *It is possible to compute an st-numbering of a graph \mathcal{G} if and only if \mathcal{G} is biconnected.*

From now on, each vertex v will be identified by means of its st-number.

Let $b(\mathcal{C})$ be the lower bound on the number of bends for the orthogonal drawing of a certain class of graphs \mathcal{C} . First, observe that an upper limitation U for the lower bound of a certain class of graphs \mathcal{C} can be obtained by exhibiting a graph belonging to \mathcal{C} needing at least U bends. If the graph belongs to a subclass \mathcal{C}' of \mathcal{C} , then U is an upper limitation for $b(\mathcal{C}')$ and even more so for $b(\mathcal{C})$. In the following we will show an infinite family of planar 2-connected cubic graphs that need at least $n/2$ bends to be laid out in the grid.

The following theorem states this result:

Theorem 4 $b(\text{Cubic Graphs}) = n/2$.

Proof Let \mathcal{G} be the planar 2-connected cubic graph depicted in Fig. 3.3, let k be the number of diamonds of \mathcal{G} and $n = 4k$ be the number of vertices of \mathcal{G} .

It is easy to see [22] that a diamond cannot be drawn on a grid with less than two bends. It follows that \mathcal{G} needs at least $2k = n/2$ bends in any

	Input	Output	Time	Gridsize	Total number of bends	Max num. of bends per edge
LMP [95]	biconn. at most cubic graph	orth. planar drawing if \mathcal{G} is planar, nothing if \mathcal{G} is not planar	$O(n)$ amortized	$O(n^2)$	$n/2 + 1$	1
BK [23]	conn. at most cubic graph and a layout of the graph if it is planar	orth. drawing (planar if \mathcal{G} is planar)	$O(n)$	$(n/2+2) \times (n-2)$ if \mathcal{G} is biconn. $(n-1) \times (n-1)$ otherwise	$n + 2$ if \mathcal{G} biconn. $2n - 1$ otherwise	2
PT [109]	biconn. at most cubic graph	orth. drawing	$O(n)$	$(n/2 + 1) \times n/2$	$n/2 + 3$	1 except one edge bending twice
K [85]	planar at most cubic graph	orth. planar drawing	$O(n)$	$\lceil n/2 \rceil \times \lceil n/2 \rceil$	$\lceil n/2 \rceil + 1$	1
CP1 [28]	conn. at most cubic graph	orth. drawing (not necessarily planar)	$O(n)$	$n/2 \times n/2$	$n/2 + 1$	1
CP2 [29]	conn. at most cubic graph	orth. drawing (not necessarily planar)	par. work $O(n)$	$(n+1) \times (n+1)$	$3n/2$	1
			par. work $O(n)$	$(3n/4 + 1/2) \times (3n/4 + 1/2)$	$n + 3$	2

Table 3.1: Known results.

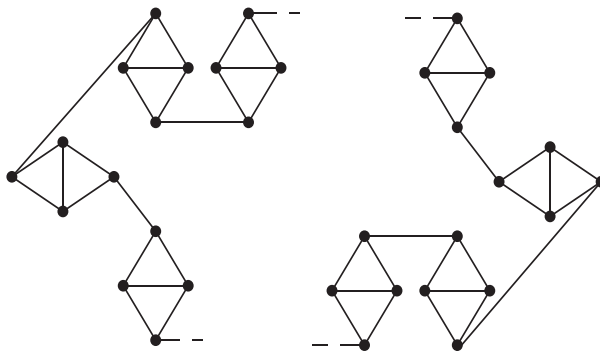


Figure 3.3: Planar 2-connected graph \mathcal{G} used in the proof of Theorem 4.

drawing and therefore $b(\mathcal{C}) \geq n/2$. Also, no cubic graph can have more than $n/2$ triangles; and therefore, $n/2$ is a tight lower bound for the whole class of cubic graphs.

In view of Theorem 4, the bound of $n/2 + 1$ bends achieved by our algorithm is nearly optimal.

3.2.2 Drawing Biconnected Graphs

The first algorithm we describe is called *BiconnCub*. It obtains a single bend drawing for biconnected graphs of maximum degree three. We begin with a high level presentation of the algorithm and then proceed to an indepth examination where a deeper understanding of the algorithm can be achieved.

Without loss of generality, we assume that the input graph is cubic since a biconnected graph cannot have any vertex of degree one and a vertex of degree two is a dot on a single edge.

The basic idea of *BiconnCub* consists in adding to the drawing, one at a time, all vertices of the graph, ordered according to an st-numbering.

Let $\mathcal{G}_k = (V_k, E_k)$ be the subgraph induced by the first k vertices in the st-numbering ($V_k = \{1, 2, \dots, k\}$) and \mathcal{D}_k be a drawing for \mathcal{G}_k . During the k -th step, vertex k and edges $\{i, k\}$, with $i < k$, are added to \mathcal{D}_{k-1} . So, if $1 < k < n$, at each step k , in view of st-numbering's properties, it is possible to draw at least one and at most two edges, together with vertex k . The drawing is produced in such a way that at least one of the two possible edges is connected in a directed way (i.e. it does not introduce any bend).

Only vertex n has three adjacent vertices in \mathcal{D}_{n-1} and it is drawn as a *comb graph*, shown in Fig. 3.4.

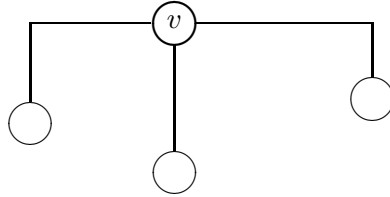


Figure 3.4: A comb graph.

Observation 1 *Given an st -numbered biconnected cubic graph \mathcal{G} , the number of vertices v having two adjacent vertices i and j such that $i, j < v$, is exactly $n/2 - 1$.*

Before describing the algorithm in detail, it is necessary to introduce two operations that will be useful in the following.

In Fig. 3.5 a *rotation* is shown. The advantage of this operation is the change of the free directions with respect to the rotated vertex, despite the fact there is an increase in the number of bends.

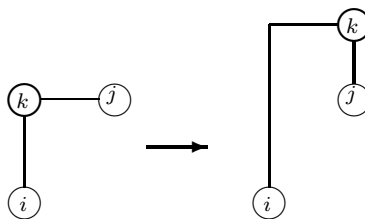
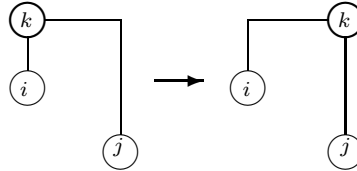
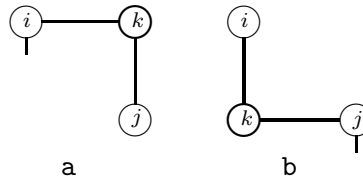


Figure 3.5: Rotation of vertex k .

In Fig. 3.6 a *bend's movement* is represented. In this case the change of the labels does not require any increase of the number of bends.

Figure 3.6: Movement of vertex k on a bend.Figure 3.7: The insertion of k does not introduce any bend.

The input of *BiconnCub* is an st -numbered cubic graph. Let (x_k, y_k) be the coordinates of vertex k on the grid and $(0, 1), (0, 2)$ the coordinates in \mathcal{D}_2 of 1 and 2 respectively. After \mathcal{D}_2 is constructed, vertices from 3 to $n - 1$ are added one at a time. Two cases have to be distinguished according to the fact that k 's adjacent vertices in \mathcal{D}_{k-1} are 1 (call it i) or 2 (i and j).

In the first case, edge $\{i, k\}$ is added to \mathcal{D}_{k-1} as a straight-line. One of k 's coordinate coincides with the corresponding one of i 's while the other one needs to be incremented such that vertex k is positioned either on the first new row or on the first new column of the current gridsize.

A bit more difficult is the case in which $\{i, k\}$ and $\{j, k\}$ are to be added to \mathcal{D}_{k-1} . In order to limit the number of bends, *BiconnCub* tries to put in a straight-line for both the edges, so that neither new rows, nor new columns are added. This is possible only in the situation of Fig. 3.7.

In all the other situations, *BiconnCub* introduces only one new bend: two cases have to be distinguished according to the existence of a common free direction for i and j .

In Fig. 3.8 and in Fig. 3.9 we show the two different cases in which either

a new row or a new column is added to the drawing: in the case of Fig. 3.8 the graph is extended from the border of the grid while in the case of Fig. 3.9 the expansion is from the inner part of the drawing.

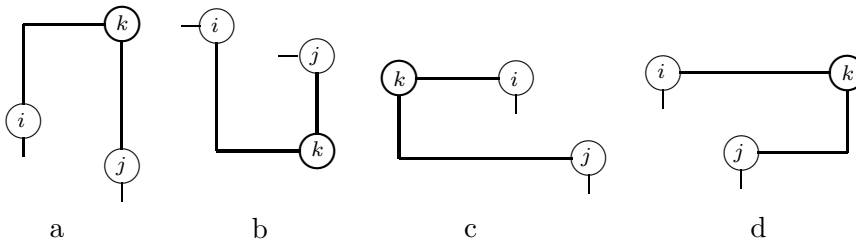


Figure 3.8: The two vertices adjacent to k have a common free direction.

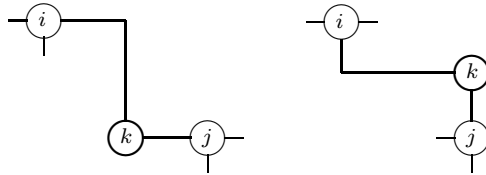


Figure 3.9: The two vertices adjacent to k have different free directions.

The analysis of all cases of *BiconnCub* is done either on the current drawing or on the drawing modified by some bend's movements. The operation of rotating a vertex is done only if no one of the previous cases is verified.

The last step of the algorithm puts on the grid vertex n according to the comb graph configuration, after a possible operation of movement of a bend and/or a rotation.

Theorem 5 *Given a biconnected at most cubic graph \mathcal{G} , BiconnCub runs in linear time and space to draw \mathcal{G} on an $n/2 \times n/2$ grid with at most $n/2 + 1$ bends. Furthermore, every edge bends at most once.*

Proof *BiconnCub works correctly: it is easy to see that BiconnCub analyzes all the possible combinations of vertices' free directions, either directly*

or after a rotation and/or a bend's movement.

Moreover, it is always possible to draw an edge without overlapping a vertex: every new vertex put on the grid lies either on a new row or on a new column, in such a way that lines passing through its free directions do not cross any vertex. The only case in which neither a new row nor a new column is introduced is the case of Fig. 3.7, but in that case the free directions are not hindered by further vertices because they were the free directions of vertices i and j .

The limitation of the total number of bends is a consequence of the following four facts:

- a. at most $n/2 - 1$ vertices have two adjacent vertices (cf. Obs. 1), then at most $n/2 - 1$ bends are introduced during steps 3 to $n - 1$;
- b. exactly two bends are introduced by a comb graph;
- c. no new bends are introduced by the bend's movement operation;
- d. the new bend introduced by the rotation of a vertex k has already been computed in a. Indeed k has two adjacent vertices in \mathcal{D}_{k-1} and its first assignment in \mathcal{D}_k is the same type as in Fig. 3.7.

To prove that the gridsize is bounded by $n/2 \times n/2$, consider that vertices 1 and 2 do not introduce any area but they add one row. An insertion of a new row or a new column, but not both, is required by each vertex from step 3 to $n - 1$. If exactly h rows are introduced, then at most $n - 3 - h$ columns are introduced. Vertex n introduces at most a row and a column. So, $A \leq (h + 2) \times (n - 2 - h)$ and this is bounded by $n/2 \times n/2$.

To complete the proof, we simply observe that the algorithm runs in linear time and uses linear space.

Now we give a more detailed description of the algorithm we have just sketched. Observe that when a condition is required (i.e. that some directions must be free) we mean that the condition can be verified either directly or after a bend's movement. For this reason, we will not explicitly mention this operation further.

Note that parameter h is a non-negative integer such that vertex k is positioned either on the first available new row or on the first available new column outside the rectangle containing the current drawing. Additionally, we will identify each grid-point as its two coordinates. Finally, the coordinates on the grid of vertex v will be denoted by (x_v, y_v) and we write "edge $\{i, k\} : (x_i, y_k) - (x_i, y_i) - (x_k, y_k)$ " or something analogous to mean that edge $\{i, k\}$ connects grid-points at coordinates (x_i, y_k) and (x_i, y_i) and (x_i, y_i) and (x_k, y_k) by two straight-lines.

Algorithm *BiconnCub***Input:** $\mathcal{G} = (V, E)$ biconnected and cubic.**Output:** \mathcal{D} = orthogonal drawing of \mathcal{G} .

```

1. begin
2.    $(x_1, y_1) \leftarrow (0, 1)$ ;
3.    $(x_2, y_2) \leftarrow (0, 2)$ ;
4.   edge  $\{1, 2\}: (0, 1) - (0, 2)$ ;
5.   for each step  $k$  from 3 to  $n$  do
6.     case of number of adjacent vertices to  $k$  in  $\mathcal{D}_{k-1}$ 
7.     one adjacent vertex  $i$ :
8.       if a position above  $i$  is free
9.       then  $(x_k, y_k) \leftarrow (x_i, y_i + h)$ 
10.      else
11.        if a position to the left of  $i$  is free
12.        then  $(x_k, y_k) \leftarrow (x_i - h, y_i)$ 
13.        else  $(x_k, y_k) \leftarrow (x_i + h, y_i)$ ;
14.      edge  $\{i, k\}: (x_i, y_i) - (x_k, y_k)$ ;
15.     two adjacent vertices  $i$  and  $j$  (assume  $j < i < k$ ):
16.     three subcases:
17.     •  $x_j = x_i$  (assume  $y_i > y_j$ )
18.       if both  $i$  and  $j$  have a free position on the right
19.       then  $(x_k, y_k) \leftarrow (x_i + h, y_i)$ 
20.       else  $(x_k, y_k) \leftarrow (x_i - h, y_i)$ 
21.       edge  $\{j, k\}: (x_k, y_k) - (x_k, y_j) - (x_j, y_j)$ ;
22.       edge  $\{i, k\}: (x_i, y_i) - (x_k, y_k)$ ;
23.     •  $y_j = y_i$  (assume  $x_i < x_j$ )
24.       if both  $i$  and  $j$  have a free position above
25.       then  $(x_k, y_k) \leftarrow (x_j, y_j + h)$ 
26.       else  $(x_k, y_k) \leftarrow (x_j, y_j - h)$ ;
27.       edge  $\{i, k\}: (x_k, y_k) - (x_i, y_k) - (x_i, y_i)$ ;
28.       edge  $\{j, k\}: (x_j, y_j) - (x_k, y_k)$ ;
29.     •  $x_j \neq x_i$  and  $y_j \neq y_i$  (assume  $x_i < x_j$  and  $y_i < y_j$ 
30.       - the other possibilities are symmetric)
31.     done  $\leftarrow$  FALSE;
32.     if a position to the right of  $i$  and below  $j$  are free
33.     then begin
34.        $(x_k, y_k) \leftarrow (x_j, y_i)$ ;
35.       edge  $\{j, k\}: (x_j, y_j) - (x_k, y_k)$ ;
36.       edge  $\{i, k\}: (x_i, y_i) - (x_k, y_k)$ ;
37.       done  $\leftarrow$  TRUE;
38.     end
39.     if (a position to the left of  $j$  and above  $i$  are free) and (not done)
40.     then begin
41.        $(x_k, y_k) \leftarrow (x_i, y_j)$ ;
42.       edge  $\{j, k\}: (x_j, y_j) - (x_k, y_k)$ ;
43.       edge  $\{i, k\}: (x_i, y_i) - (x_k, y_k)$ ;

```

```

42.         done ← TRUE;
43.     end
44.     if (both  $i$  and  $j$  have a free position above) and (not done)
45.     then begin
46.          $(x_k, y_k) \leftarrow (x_j, y_j + h)$ ;
47.         edge  $\{j, k\}$ :  $(x_j, y_j) - (x_k, y_k)$ ;
48.         edge  $\{i, k\}$ :  $(x_i, y_i) - (x_i, y_k) - (x_k, y_k)$ ;
49.         done ← TRUE;
50.     end
51.     if (both  $i$  and  $j$  have a free position below) and (not done)
52.     then begin
53.          $(x_k, y_k) \leftarrow (x_j, y_j - h)$ ;
54.         edge  $\{j, k\}$ :  $(x_j, y_j) - (x_k, y_k)$ ;
55.         edge  $\{i, k\}$ :  $(x_i, y_i) - (x_i, y_k) - (x_k, y_k)$ ;
56.         done ← TRUE;
57.     end
58.     if (both  $i$  and  $j$  have a free position on the left) and (not done)
59.     then begin
60.          $(x_k, y_k) \leftarrow (x_i - h, y_j)$ ;
61.         edge  $\{i, j\}$ :  $(x_j, y_j) - (x_k, y_j) - (x_j, y_j)$ ;
62.         edge  $\{i, k\}$ :  $(x_i, y_i) - (x_k, y_k)$ ;
63.         done ← TRUE;
64.     end
65.     if (both  $i$  and  $j$  have a free position on the right) and (not done)
66.     then begin
67.          $(x_k, y_k) \leftarrow (x_j + h, y_j)$ ;
68.         edge  $\{i, j\}$ :  $(x_j, y_j) - (x_k, y_j) - (x_j, y_j)$ ;
69.         edge  $\{i, k\}$ :  $(x_i, y_i) - (x_k, y_k)$ ;
70.         done ← TRUE;
71.     end
72.     if (a position to the right of  $i$  and to the left of  $j$  are free) and (not done)
73.     then begin
74.         insert a new column  $c$  between  $x_i$  and  $x_j$ ;
75.          $(x_k, y_k) \leftarrow (c, y_j)$ ;
76.         edge  $\{i, k\}$ :  $(x_i, y_i) - (x_k, y_i) - (x_k, y_k)$ ;
77.         edge  $\{j, k\}$ :  $(x_j, y_j) - (x_k, y_k)$ ;
78.         done ← TRUE;
79.     end
80.     if (a position below  $j$  and above  $i$  are free) and (not done)
81.     then begin
82.         insert a new row  $r$  between  $y_j$  and  $y_i$ ;
83.          $(x_k, y_k) \leftarrow (x_j, r)$ ;
84.         edge  $\{i, k\}$ :  $(x_i, y_i) - (x_i, y_k) - (x_k, y_k)$ ;
85.         edge  $\{j, k\}$ :  $(x_j, y_j) - (x_k, y_k)$ ;
86.         done ← TRUE;
87.     end
88.     if not done

```

89. then make a rotation and run again from line 15
 with the same value of k . After this second
 execution one case must necessarily be verified.
90. three adjacent vertices to k , i.e. $k = n$:
91. find a common free direction of the three adjacent vertices
 (either directly or after an operation) and put n in the
 drawing by using a comb graph.
92. end.

3.2.3 Drawing Connected Graphs

In this subsection we focus on the drawing of general at most cubic graphs. The approach to the problem passes through biconnected components; hence *BiconnCub*, described in Subsection 3.2.2, will play a fundamental role.

Let $\mathcal{B}_1 = (V_1, E_1), \dots, \mathcal{B}_K = (V_K, E_K)$ be the biconnected components of $\mathcal{G} = (V, E)$ and let a_1, \dots, a_r be the articulation points ($n = |V| = |V_1| + \dots + |V_K| + r$).

The main idea of the algorithm –that from now on we will call *ConnCub*– consists of drawing each \mathcal{B}_i separately on a grid and then in connecting the orthogonal drawings by means of the points a_i . Edges incident to vertices a_i are drawn taking into account both the limitation of the total number of bends and the total area of the drawing of \mathcal{G} .

We will prove that, in this way, the upper bounds of the area and of the total number of bends remain the same as in the biconnected case. No restriction is imposed by supposing that only one articulation point a exists. Actually, if there are more than one, then by deleting a from \mathcal{G} some simply connected components will be obtained, and it is possible to inductively repeat the same arguments. So, suppose that after a is deleted from \mathcal{G} , only biconnected components remain.

Furthermore, observe that all vertices adjacent to a – v_1, v_2 and v_3 – have at most two incident edges on the drawing of their biconnected component, therefore they have at least two free directions and the algorithm maintains empty the row (or the column) from all their free directions to the boundary. So, it is not restrictive to think (just useful for the simplicity of explanation) that v_1, v_2 and v_3 lay on the boundary of their biconnected components' drawings.

It is necessary to distinguish two different cases, depending on either a disconnects the graph into two or three biconnected components (see Fig. 3.10.a and .b).

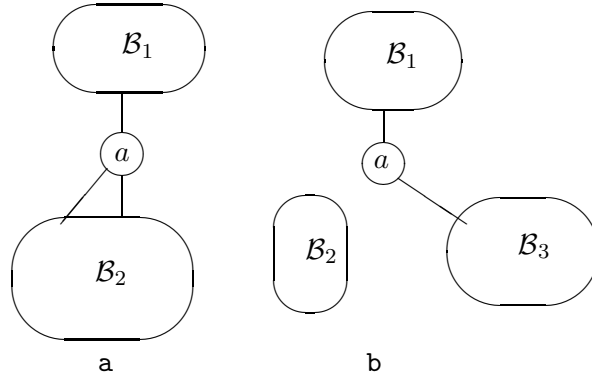


Figure 3.10: The two different types of articulation points.

There are two biconnected components \mathcal{B}_1 and \mathcal{B}_2 . Suppose $v_1, v_2 \in \mathcal{B}_1$ and $v_3 \in \mathcal{B}_2$. There are different cases according to the mutual position of v_1 and v_2 .

In Fig. 3.11.a and .b the way of connecting a to v_1, v_2 and v_3 is shown both when v_1 and v_2 have a free common direction in the drawing of \mathcal{B}_1 and when they have it on two different but consecutive sides of the drawing of \mathcal{B}_1 . In both of these cases at most two bends are introduced.

In Fig. 3.12 the case in which the free directions of v_1 and v_2 are on two different and opposite sides of the drawing of \mathcal{B}_1 is shown. It is possible to see that this case is reduced to one of the previous ones. Namely, v_1 and v_2 have at least two free directions, therefore another available row (or column) going towards the boundary must exist.

concerning the gridsize guaranteed by *ConnCub*, in the case depicted in Fig. 3.11.a the worst case happens when v_1 lies on the lower-left corner of the drawing of \mathcal{B}_1 and v_3 is on the higher-right corner of \mathcal{B}_2 (see Fig. 3.13.a). In the case of Fig. 3.11.b the worst case occurs when v_3 is exactly in the middle of the side where it lies (see Fig. 3.13.b).

Therefore, the values of the area in the two cases are $A \leq (|V_1|/2 + \frac{|V_2|}{2} + 2) \times (\frac{|V_1|}{2} + \frac{|V_2|}{2})$ and $A \leq (\frac{|V_1|}{2} + \frac{|V_2|}{4} + 1) \times (\frac{|V_1|}{2} + \frac{|V_2|}{2} + 2)$ respectively; these are both

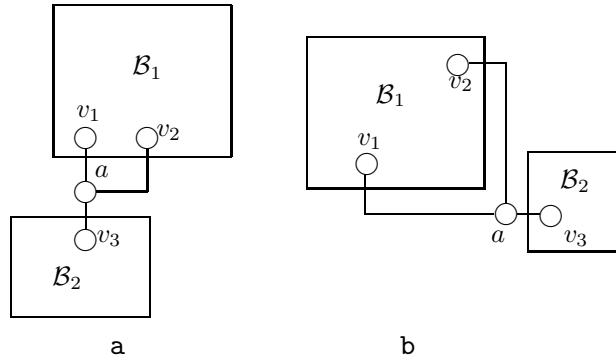


Figure 3.11: Two ways of connecting two components.

less than $n/2 \times n/2$.

There are three biconnected components $\mathcal{B}_1, \mathcal{B}_2$ and \mathcal{B}_3 . It is always possible to put in the grid the drawings of the components in such a way that the two smallest components are put beside each other, while the largest one is put on the opposite side (see Fig. 3.14.a). In order to compute the gridsize, observe that the situation in which v_1 lays on the lower-left corner of the drawing of \mathcal{B}_1 and v_3 on the higher-right corner of \mathcal{B}_3 is the worst case; it does not matter where v_2 lies (see Fig. 3.14b). It is not restrictive to suppose that $|V_2| \geq |V_1|$; therefore the width of the grid is not greater than $\frac{|V_3|}{2} + 2 + \frac{|V_2|}{2}$ and its height is not greater than $\max(\frac{|V_3|}{2}, \frac{|V_1|}{2} + \frac{|V_2|}{2} + 1, \frac{|V_3|}{2} + \frac{|V_1|}{2} - 1)$. In all cases, the area is less than $n/2 \times n/2$.

Utilizing the previous observations, it is possible to state the following theorem.

Theorem 6 *Given a connected at most cubic graph \mathcal{G} , ConnCub runs in linear time and space to draw a 1-bend drawing of \mathcal{G} on an $n/2 \times n/2$ grid with at most $n/2 + 1$ bends.*

Proof It is easy to see that the procedure for inserting an articulation point in the drawing runs in constant time, so *ConnCub* runs in linear time.

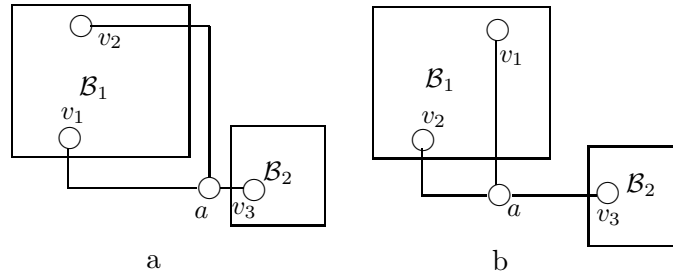


Figure 3.12: Third way of connecting two components.

In order to prove that the maximum number of bends is $n/2 + 1$, observe that the drawing of the articulation point a and its outgoing edges introduce at most two bends in the drawing of \mathcal{G} . Each biconnected component \mathcal{B}_i has at most $|V_i|/2$ bends because its t -node (in its st -numbering) is adjacent to a by construction, and in \mathcal{B}_i it has at most two adjacent vertices. Then, the increase of number of bends introduced by inserting a is balanced by the decreasing in each biconnected component.

Bounds for the gridsize immediately follows from the previous details.

3.2.4 Experimental Results

In this subsection, we present an extensive experimental study comparing three graph-drawing algorithms dealing with cubic graphs: one of them – CP in the next figures – is the algorithm *ConnCub* just described [28], the other two algorithms –BK and PT– are explained in [23] and [109] and have already been cited in Table 3.1. The further algorithms mentioned in Table 3.1 (LMP and K) have not been considered in this comparative work since they deal only with planar graphs.

The three algorithms considered are very similar: all of them run a pre-processing phase computing an st -numbering and draw the vertices one at a time according to the st -numbering. PT is the simplest one since it considers less cases than the other algorithms, while BK is the only one producing a 2-bend drawing instead of a 1-bend drawing. Furthermore, if an embedding is given in input, Algorithm BK is the only one able to draw a planar graph without crossings. CP and BK deal with simply connected graphs

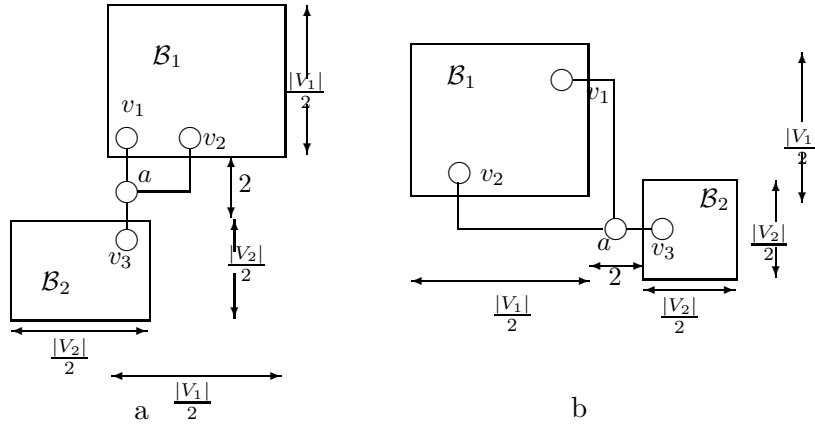


Figure 3.13: Worst situations for the cases of Fig. 3.11.

by working on separated biconnected components and then by re-splicing them in a further step. In contrast, Algorithm PT only runs on biconnected graphs. In order to allow all three algorithms to take homogeneous inputs, we have added a second phase to PT to handle separated biconnected components and to splice them. Namely, we utilize *ConnCub* (cf. Subsection 3.2.3) replacing *BiconnCub* with the algorithm described in [109].

In this way BK, CP and PT take as input general at most cubic graphs (with no restrictions on the connectivity, planarity, etc.) and construct orthogonal grid drawings.

The test data are 18,000 randomly generated graphs; half of them are simply connected graphs. The number of vertices of such graphs are multiples of 10 and are in the range between 10 and 300. The experiments provide a detailed quantitative evaluation of the performance of the three algorithms, both from an ‘aesthetic’ point of view (gridsize, number of bends and number of crossings) and from a computational one (running time). The observed practical behavior of the algorithms is consistent with their theoretical properties, but demonstrates that sometimes the algorithms work better than the theoretical results state, as we will discuss in the following comments.

In the remainder of this section we present and comment on two series of

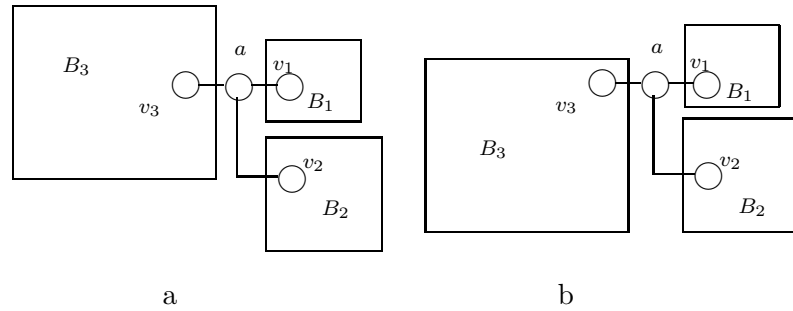


Figure 3.14: Way of connecting three components and the worst case for the gridsize.

diagrams: the first one is related to simply connected at most cubic graphs having n vertices, m edges and k biconnected components; we apply the constraints that $1 \leq k \leq n/20$ and $\frac{2.75}{2}n \leq m \leq \frac{3}{2}n$. The latter one deals with biconnected at most cubic graphs with a number of edges close to or equal to $\frac{3}{2}n$.

The following quality measures of a drawing of a graph have been considered:

Area: area of the smallest rectangle with horizontal and vertical sides covering the drawing;

Bends: total number of bends;

Crosses: total number of crossings;

Time: computational time of the algorithms, measured in milliseconds.

We would now like to make a few observations about the experiments. The figures referenced to appear on the following pages.

Area: (Figs. 3.15 and 3.16) Observe that Algorithms BK and PT both compute an average area very close to the theoretical bound of $n^2/2$ and $n^2/4$, respectively. This happens both in the connected and in the biconnected case. In contrast, since Algorithm CP tries to draw vertices without adding new bends and therefore without adding new rows and columns, it reaches

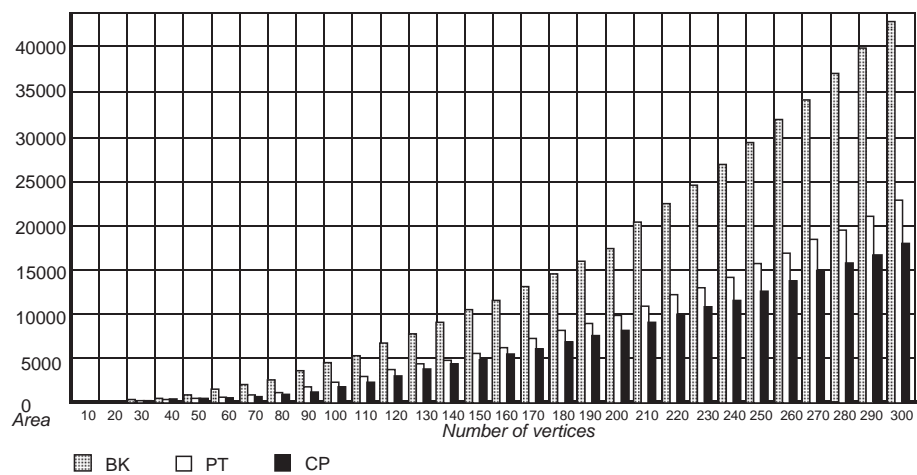


Figure 3.15: Average area versus number of vertices for simply connected graphs.

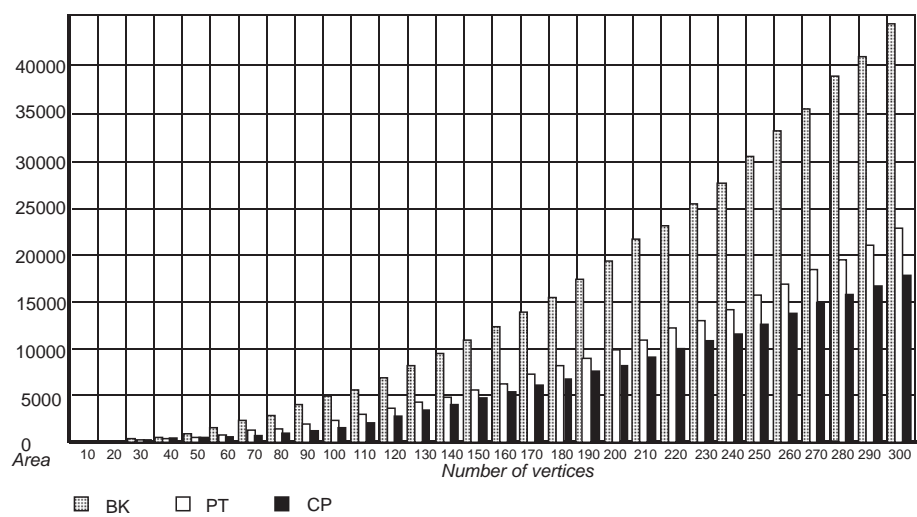


Figure 3.16: Average area versus number of vertices for biconnected graphs.

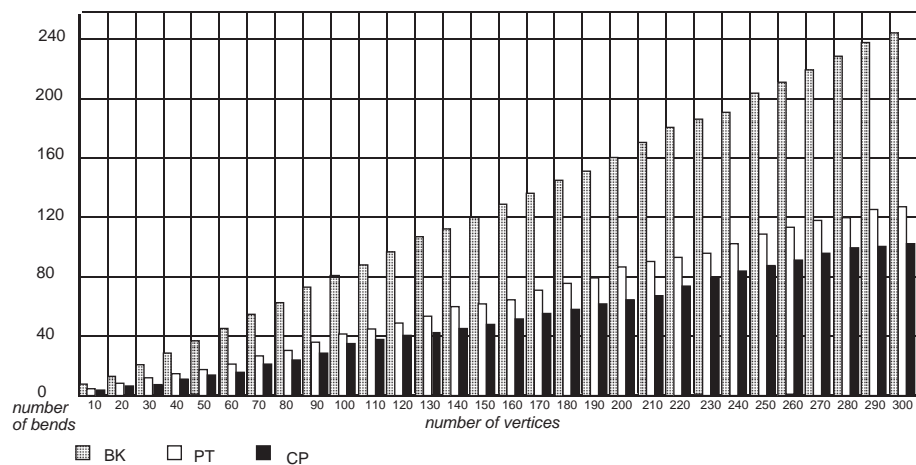


Figure 3.17: Average total number of bends versus number of vertices for simply connected graphs.

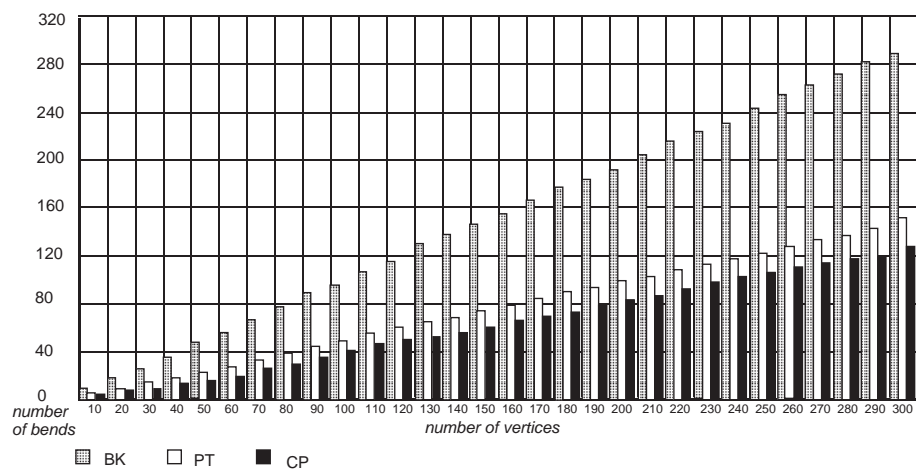


Figure 3.18: Average total number of bends versus number of vertices for biconnected graphs.

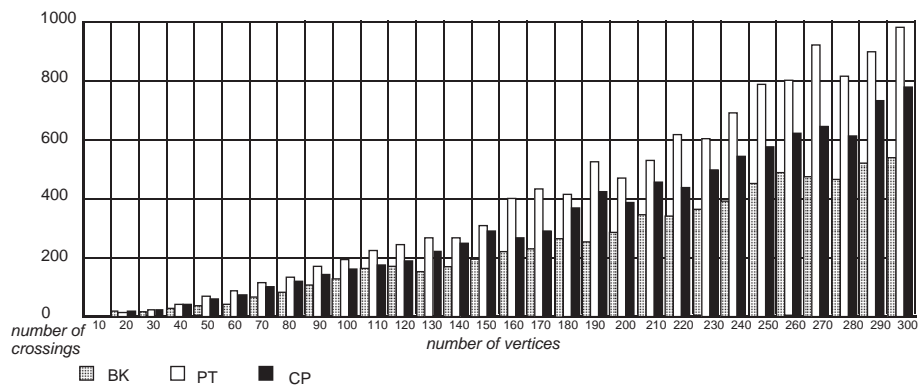


Figure 3.19: Average number of crossings versus number of vertices for simply connected graphs.

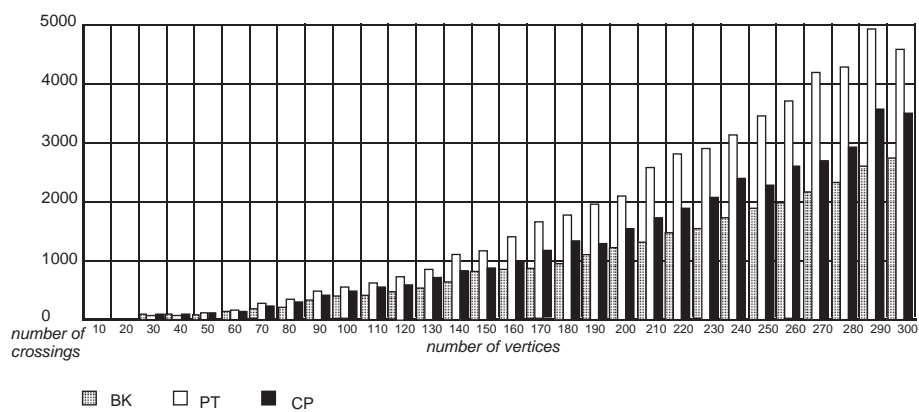


Figure 3.20: Average number of crossings versus number of vertices for bi-connected graphs.

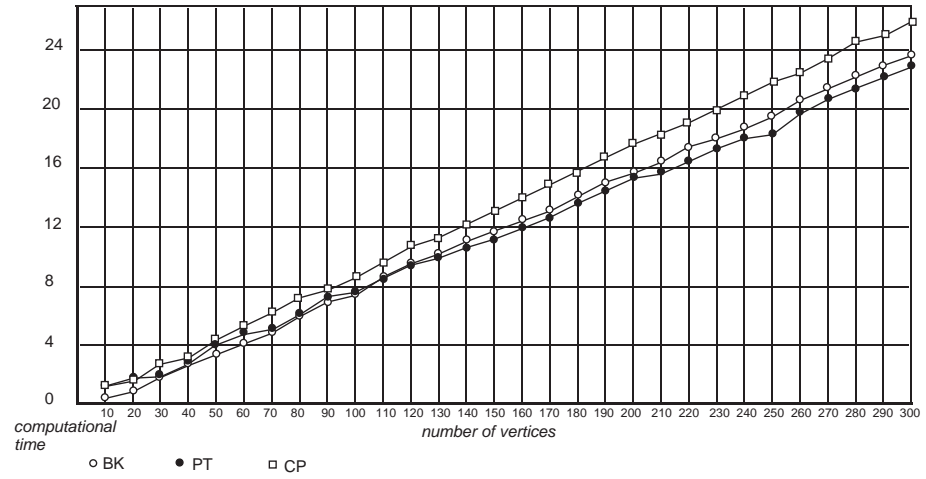


Figure 3.21: Average computational time (in ms) versus number of vertices for simply connected graphs.

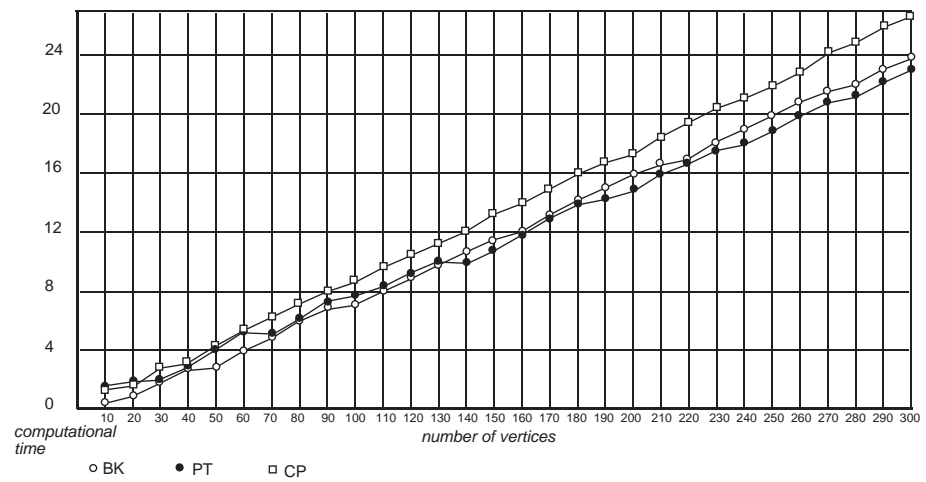


Figure 3.22: Average computational time (in ms) versus number of vertices for biconnected graphs.

a slightly better area bound (about $n^2/4.88$ instead of $n^2/4$) both in the connected and in the biconnected case.

Bends: (Figs. 3.17 and 3.18) For what concerns the total number of bends, it is clear from the plots that all three algorithms generate less bends in the simply connected case than in the biconnected one. This happens because the number of edges in connected graphs of our test data is, on average, less than the number of edges in biconnected graphs.

BK generates a total number of bends that is close to n for biconnected graphs and close to $n/1.25$ for connected graphs. Algorithm BT approaches $n/2$ and $n/2.5$ in the biconnected and connected case, respectively. Algorithm CP works better from this point of view, since it generates about $n/2.44$ and $n/3$ bends if the input graph is biconnected and connected, respectively.

Crosses: (Figs. 3.19 and 3.20) The total number of crossings does not appear to be an increasing function of the number of vertices. This is due to the fact that the number of crossings is highly random and its average value becomes stable only after a very large number of tests. Observe that the number of crossings is consistently smaller when the graphs are simply connected rather than when they are biconnected, because edges connecting the biconnected components are always drawn without crossings. The average values in the biconnected case are $n^2/33$, $n^2/25.8$ and $n^2/17.6$ for Algorithms BK, CP and PT, respectively. The good result of BK can be explained in view of the fact that it has the worst result for the gridsize; and therefore, it has more possibilities to lay out edges without crossings.

Time: (Figs. 3.21 and 3.22) All three algorithms are very fast: PT runs in $n/13$ ms; BK and CP require $n/12.6$ and $n/11.2$ ms to draw a cubic graph with n vertices, respectively. PT is the fastest algorithm because it is the simplest one and does not check as many cases and configurations as the other two algorithms do.

3.3 A Parallel Algorithm to Orthogonally Draw Cubic Graphs

3.3.1 Introduction

In the literature, there are very few parallel algorithms that orthogonally draw graphs.

In [78] an algorithm constructing a planar drawing with vertices placed at real coordinates is given but no known bound on the area is produced. The running time for their algorithm is $O(\log^2 n)$ and the number of processors required is $M(n)$, that is the number of processors needed to multiply two $n \times n$ matrices in $O(\log n)$ time on a CREW PRAM.

In 1991 this result was substantially improved: in [141] an algorithm dealing with biconnected planar graphs is presented, but an embedding must be given in input.

In [140] this result is generalized to simple graphs and the algorithm presented runs on a CREW PRAM in $O(\log n)$ time with $n/\log n$ processors and constructs layouts with $O(n)$ maximum edge length and $O(n^2)$ area. Also, the number of bends is at most $2n + 4$ if the graph is biconnected, and is at most $2.4n + 2$ if it is simple.

In this section we present a parallel algorithm that constructs an orthogonal drawing of an n vertex cubic graph \mathcal{G} with $O(n)$ bends, $O(n)$ maximum edge length and $O(n^2)$ area in $O(\log n)$ time on a CRCW PRAM with n processors. Two slight variants of the algorithm are described: the first one generates a drawing where each edge has at most two bends, the total number of bends is less than or equal to $n + 3$, and the area is less than or equal to $(\frac{3}{4}n + \frac{1}{2})^2$; the second one optimizes the number of bends for each edge (at most one) even if the values of the other functions are slightly worst.

Although this algorithm is not optimum, it is the first parallel algorithm dealing with non-planar, non-biconnected graphs. Moreover, no drawing of the graph is required as input nor is an st-numbering (or canonical numbering) computed.

3.3.2 Description of the Algorithm

Given as input an at most cubic graph \mathcal{G} , the output of the algorithm –that from now on we will call *ParCub*– consists of the set of coordinates in the grid of vertices, bends and –for every vertex– the directions of its incident edges.

To clarify the exposition, we first sketch the algorithm and then we present it in a more detailed way.

The algorithm is divided into the following three steps:

1. find a general spanning tree \mathcal{T} of the input graph \mathcal{G} and generate an embedding of \mathcal{T} in the grid to obtain an almost straight-line drawing;

2. add the remaining non-tree edges to the drawing;
3. compact the drawing by eliminating the rows and the columns that contain no vertices nor parts of edges from the grid.

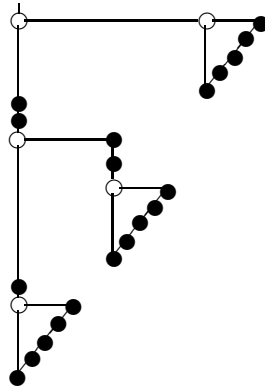
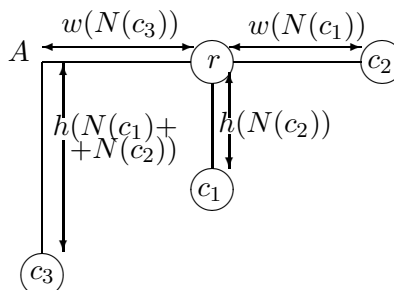


Figure 3.23: General “skeleton” of spanning tree \mathcal{T} on the grid.

We start from a spanning tree \mathcal{T} as a “skeleton” of the whole drawing. After the drawing of \mathcal{T} all the adjacent vertices of some vertices (the white ones shown in Fig. 3.23) have already been drawn, while some adjacent vertices of the other vertices (the black ones) have to be still drawn. For this reason, in Step 1 we use a particular way of drawing the spanning tree, so that every row i of the grid contains only one vertex v_i with less than three adjacent vertices and v_i is the rightmost vertex lying on row i (see Fig. 3.23). Furthermore, enough space is left to put all the remaining edges between two consecutive columns. Obviously, not every row and not every column are occupied by the edges; this is the reason why the final compaction is necessary.

Let us call r the root of \mathcal{T} . In the general case, r has three children and we can draw them as illustrated in Fig. 3.24.

In order to avoid having segments of different edges overlap, we allow only one ‘black’ vertex (Fig. 3.23) per row of the drawing. For this reason edges $\{r, c_3\}$ and $\{r, c_1\}$ (Fig. 3.24) must be long enough to allow to the entire subtrees rooted at c_1 and c_2 , respectively, to be drawn. A similar

Figure 3.24: Embedding of root r and of its three children.

reason holds for the columns. Functions w and h , whose specific meaning will be explained later, guarantee that this conditions will be satisfied.

If r has only two children, then only c_1 and c_2 are drawn and in this case the drawing of \mathcal{G} is exactly straight-line.

As well as both r and c_1, c_2 and c_3 are drawn, all the other vertices are put on the grid. For each vertex, we can repeat the previous reasonings, and a sufficient distance from the neighbor vertices both in height and in width must be left free.

After \mathcal{T} is embedded, the second step follows. The main idea of the algorithm consists in inserting as many edges of type b. (Fig. 3.25) as possible. This choice is based on the fact that such edges introduce no new rows nor new columns and therefore they do not increase the gridsize. Moreover, these edges are 1-bend and this is useful for the limitation both of the total number of bends and of the number of bends on each edge. In order to increase the number of such edges, the algorithm allows to move a vertex on a bend as shown in Fig. 3.27. Observe that the movement of a vertex on a bend does not increase the total number of bends in the drawing.

During this phase, some of the free columns become occupied by the edges; the left ones will be eliminated during the last step.

Now, we will explain the algorithm in detail, step by step. The input of the algorithm is a general cubic graph \mathcal{G} with n vertices.

The **first step** is divided into the following two parts:

1. Finding a spanning tree \mathcal{T}

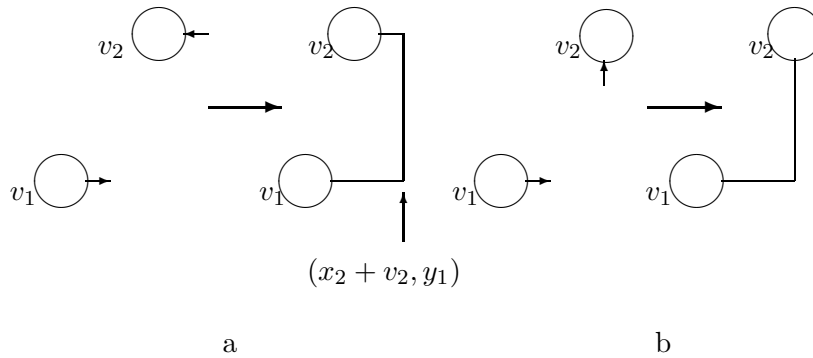


Figure 3.25: Different possibilities to embed an edge.

The parallel algorithm computing a general spanning tree of an at most cubic graph is an explicit modification of the Shiloach and Vishkin connected components algorithm [130]. The starting-point of the algorithm is a forest with n one-vertex trees, one for each vertex of \mathcal{G} . A processor is assigned to each vertex v and one of the adjacent vertices of v numbered less than v is chosen, say w . Edge $\{v, w\}$ is added to the forest. Now, each step of the algorithm is divided into two parts: in the first one all of the star trees, if they exist, are grafted into other higher trees; the second one halves the height of all the trees in the forest. In view of the fact that the first part consists in changing a constant number of pointers and the second one is just one application of the pointer jumping technique, each step works in constant time with $O(n)$ processors. Then $O(\log n)$ steps on a CRCW PRAM are sufficient to generate a spanning tree. Notice that this is the only step where a CRCW PRAM is required. Because of it, the whole algorithm runs on a CRCW PRAM. Since the procedure that embeds \mathcal{T} on the grid (Step 2) needs the root to have at least two children, if the root of the general spanning tree has only one child, further $O(\log n)$ time must be spent to perform an Euler Tour procedure rooting the tree to a vertex with degree at least two.

2. Embedding \mathcal{T} on the grid

By using the Euler Tour technique, it is possible to assign to every vertex v the labels $N(v)$ and $p(v)$, where $N(v)$ represents the number of vertices in the subtree rooted at v , included v , and $p(v)$ is the post-order numbering. From now on, v and $p(v)$ will be identified. A post-order numbering is convenient since increasing values from the leaves to the root are needed.

In order to guarantee that each row of \mathcal{T} 's embedding has no more than one 'black' vertex, the two linear functions *height* $h(x) = 2x$ and *width* $w(x) = (n + 1)\lceil \frac{x}{2} \rceil$ are introduced.

The argument of these functions depends on the size of the subtrees of the current vertex. In Fig. 3.24 the arguments of these functions are specified for the root and its children and their correlated positions on the grid are pointed out. Notice that the origin of the axes is in the higher leftmost corner. The position of a general vertex v in the grid respect to its children will be the same as r with respect to c_1 (c_1 and c_2) if v has one child (two children). If c_2 does not exist, then the distance between v and c_1 is two.

In order to calculate its own position, every vertex needs to know its parent's coordinates. This can be done with a classical pointer jumping technique and it requires $O(\log n)$ time with n processors.

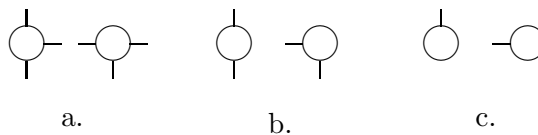


Figure 3.26: Vertices in different situations after Step 1 of *ParCub*.

After this step, vertices have either three or two or one incident edges and they belong to one of the classes represented in Fig. 3.26.

Because of the cubicity of the graph, only vertices of type b. and c. work on for the following part of the algorithm.

Before describing the second step of *ParCub*, the following observations about the embedding of \mathcal{T} can be done.

The Right position for vertices of type b. and Right and Down positions for vertices of type c. are always free. Therefore, these positions will be used to attach the remaining edges.

Let $e = \{v_1, v_2\}$ be an edge still to be drawn, and v_1 less than v_2 . In view of the post-order numbering, v_1 lies below v_2 (*Post-Order property*). We say that e is *sent* by v_1 and is *received* by v_2 , and v_1 and v_2 are called *sender* and *receiver*, respectively.

In the following we consider vertices with respect to their configuration at the end of this first step.

The **second step** is divided into the following eight parts:

1. Directions' assignment

A processor is assigned to each vertex v and the directions for the connections of its incident non-tree edges are decided:

if v is a vertex of type b. (in all this part refer to Fig. 3.26) then we assign the Right direction to the only edge remaining to be drawn;

if v is a vertex of type c., then we must consider different cases in view of the possible types of edges:

- if v is a sender for both the edges and only one of its receivers is of type b., then the Right direction is assigned to the corresponding edge, the Down direction to the other edge. This rule is applied only if the two receivers do not lie on the same column. In this case, the Right direction is assigned to the receiver having the smaller numbering (for instance, see vertex 3 of the example at the end of this subsection. These choices will give the possibility to move v on a bend during a successive step.
- if v is a sender for both the edges and both its receivers are of the same type (either .b or .c), then the Right direction is assigned to the edge corresponding to the receiver having the smaller numbering. The reason for this choice is to exploit parallelism.
- if v is a sender for an edge and a receiver for the other one, then the Right position is assigned to the sent edge and the Down position is assigned to the received edge. This choice is due to the fact that the situation of Fig. 3.25.b has to be privileged.
- if v is a receiver for both the edges, then the Right direction is assigned to the edge corresponding to the sender with the smaller

numbering. This choice is made in order to maintain the *Post-Order property*, even after bend's movements.

From now on we will consider a processor assigned to each edge $e = \{v_1, v_2\}$.

2. Directly 1-bend edges

All the edges of type b. of Fig. 3.25 are drawn on the grid.

3. Movement on the bend

Every vertex that is a sender for two edges will be moved on the bend on its right, if it exists (Fig. 3.27.a). Successively, every vertex that is a receiver for two edges will be moved on the bend on its bottom, if it exists (Fig. 3.27.b). Finally, each vertex sender (receiver) of only one edge will be moved on a bend if its direction is not Right (Down). These movements rotate the directions of non-tree edges in a counterclockwise mode, increasing the number of Right directions to connect to Down directions. The sequentiality of the previous movements is necessary to avoid two vertices moving towards the same bend.

4. Drawing of the Right edges

Let v_1 be the sender of e ; if the direction of e on v_1 is the Right one, then e is embedded on the grid (see Fig. 3.25).

5. Elimination of the Down directions

Every vertex sending an edge toward the Down direction (i.e. its sent edge along the Right direction is already drawn) is moved on the bend lying at its Right and the Down direction is transformed into the Right direction (see Fig. 3.27.a).

6. Elimination of 2-bend edges

For each edge drawn with two bends on the grid, its receiver is moved on the right according to the cases presented in Fig. 3.28. In this way a bend is moved on a straight-line edge.

7. Drawing completion

In order to complete the drawing, Steps 4., 5. and 6. have to be run again. So, all remaining edges are drawn on the grid.

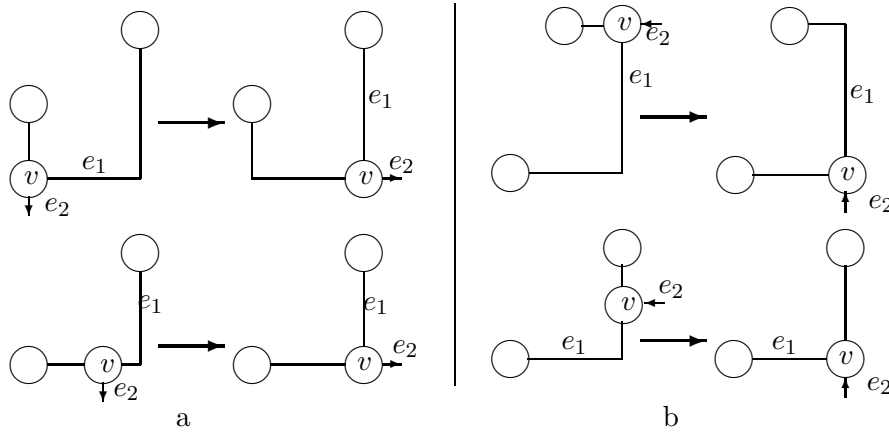


Figure 3.27: Movement on a bend of a vertex.

8. Movement of the root

In order to make the drawing at most 1-bend, it remains to check if the edge $\{r, c_3\}$ has one or two bends. In the latter case, the root has to be moved under the corner A (Fig. 3.24) in position $(0, 1)$ of the grid. In such a case its children c_1, c_2 and c_3 will be connected to the Right, Up and Down directions, respectively.

Using n processors the whole step can be computed in constant time.

Before continuing with the explanation of the algorithm, it is convenient to explain the role of the columns of the grid placed between two consecutive columns used to embed \mathcal{T} . They are necessary to embed non-tree edges and we will show that vertical segments of edges are arranged such that they lie between the column where their rightmost extreme lies and the consecutive column used to embed \mathcal{T} .

Let $e = \{v_1, v_2\}$ be an edge, and let (x_i, y_i) be the coordinates of a vertex v_i after the first step. Let $x_{max} = \max\{x_1, x_2\}$. The vertical segment of e is placed on the new column $x_{max} + v_2$. This arrangement is made in order to avoid having two segments of different edges lying on the same piece of the grid.

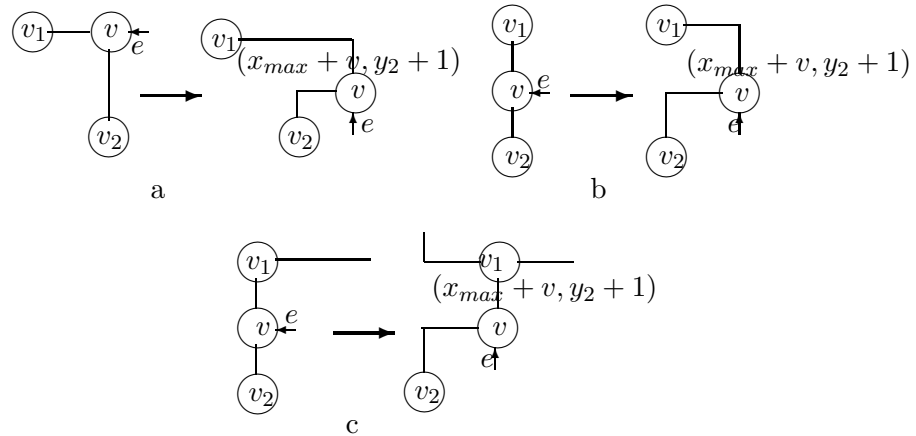


Figure 3.28: Possible translations of a receiver vertex.

Now, suppose that a movement on a bend is necessary. In such a case we have to embed two edges $e_1 = \{v_1, v_2\}$ and $e_2 = \{v_1, v_3\}$. It is not restrictive to suppose v_2 less than v_3 , therefore the Right direction of v_1 is assigned to e_1 and the Down direction of v_1 is assigned to e_2 . After e_1 is embedded, v_1 is moved on its bend and therefore their coordinates become $(x_{max} + v_2, y_1)$. The vertical segment of e_2 is drawn to the right of the vertical segment of e_1 (in view of the fact that v_2 less than v_3) on the column $\max\{x_1, x_2, x_3\} + v_3$. This column lies between two consecutive columns used to embed \mathcal{T} : the column containing v_1 and the following one.

At the beginning of the **third step**, every part of the drawing is characterized by its own coordinates. Two sorted lists are created –one for the rows and one for the columns– to store the coordinates of the occupied rows and columns. The sorting phase takes $O(\log n)$ time using n processors on a CREW PRAM (see [37]). Some coordinates appear more than once because on the same row or column different drawing elements can be present. The aim of this step is to eliminate the gap possibly existing between two consecutive elements in the lists. In every list, there are $O(n)$ elements, so a vector of $O(n)$ cells is enough to store one of each group of identical values by comparing adjacent elements in the lists and putting 1 or 0 according to the fact

that they are different or equal. The prefix sum technique can be applied to count all non zero elements in the vectors. Finally, the corresponding values in the lists can be modified by using the numbering obtained, so that consecutive elements in the lists have either the same value (if they had the same value before this computation) or consecutive values (if they had different values).

This step can be executed in $O(\log n)$ time using n processors.

From the previous considerations about the time complexity the following theorem can be easily derived.

Theorem 7 *Given an at most cubic graph \mathcal{G} with n vertices, ParCub determines a 1-bend drawing of \mathcal{G} on a grid in $O(\log n)$ time using n processors on a CRCW PRAM model.*

3.3.3 Complexity Measures

Before proving theorems describing upper bounds for area and for the number of bends, functions *height* and *width* must be justified.

Every ‘black’ vertex (Fig. 3.23) has to lie on a different row and the right-side vertices are closer to the root than the left-side ones. Then, for each vertex, we have to compute how much space can be used in the worst case by all vertices that lie under it. It is easy to see that j rows are enough to fit j vertices to the right of the current vertex. Moreover, another j rows are necessary to allow the translations shown in Fig. 3.28.a. Therefore the function $h(j)$ is justified.

The columns’ problem is not as easy as the rows’ one: indeed a lot of vertices can lie on the same column. For this reason, when newly inserted edges use some new columns it can not be allowed that two different edges use the same column in order to avoid overlapping. Therefore, a different column must be assigned to every possible new edge between every couple of columns containing vertices. The definition of function $w(j)$ then follows.

Theorem 8 *Given an at most cubic graph \mathcal{G} with n vertices, ParCub computes a layout of \mathcal{G} with at most $\frac{3}{2}n$ bends, $O(n)$ maximum edge length, and $(n+1) \times (n+1)$ maximum area of the grid. The number of bends on every edge is at most one.*

Proof The orthogonal drawing is 1-bend by construction and the length of every edge is $O(n)$.

Now, a bound for the total number of bends will be calculated: the first step of the algorithm embeds $n - 1$ straight-line edges with only one possible exception. All other $\frac{n}{2} + 1$ edges contain one bend generated either by a connection of type b. in Fig. 3.25 or by a translation. For every such edge, in the first case the total number of bends in the drawing is incremented by one while in the second case the increment is equal to three: the bend of the edge plus two new bends inserted in two old straight-line edges (see Fig. 3.28). The worst case happens when all non-tree edges introduce three bends, then each edge has one bend and the total number of bends is bounded by the number of edges in a cubic graph, i.e. $\frac{3}{2}n$.

Regarding the gridsize, observe that every vertex introduces either a new row or a new column in the drawing of \mathcal{T} but not both; therefore the embedding of \mathcal{T} can be drawn on a grid with h rows and $n - h$ columns (if we do not consider all the free rows and columns, disappearing after the last step of the algorithm). Furthermore, after the second step of the algorithm, an edge of type b. of Fig. 3.25 does not introduce new rows nor new columns. All the other edges introduce both a column and a row.

For this reason, in the worst case, $\frac{n}{2} + 1$ new columns and $\frac{n}{2} + 1$ new rows are introduced. Therefore, $Total Area \leq (h + \frac{n}{2} + 1) \times (n - h + \frac{n}{2} + 1)$ and its maximum value is reached when $h = \frac{n}{2}$; in this case we have:

$$Total Area \leq (n + 1) \times (n + 1).$$

Observe that the maintenance of 1-bend drawing requires the use of the Steps “Elimination of 2-bend edges” and “Movement of the Root.” The first one wastes both area and number of bends. For this reason, if these steps are eliminated from the algorithm, better upper bounds for area and total number of bends can be achieved, despite the increase of the maximum number of bends per edge.

Theorem 9 *Given an at most cubic graph \mathcal{G} with n vertices, ParCub without the Step “Elimination of 2-bend edges” computes a layout of \mathcal{G} with at most $n + 3$ bends, $O(n)$ maximum edge length, and $(\frac{3}{4}n + \frac{1}{2}) \times (\frac{3}{4}n + \frac{1}{2})$ maximum area of the grid. The number of bends on every edge is at most two.*

Proof Only the differences with the previous proof will be stressed.

All non-tree edges can contain either one or two bends and the worst case happens when two bends are introduced by every edge since $2(\frac{n}{2} + 1)$ bends are introduced. It follows that the maximum number of bends is $n + 3$.

Concerning the gridsize, notice that, after the second step of the algorithm, an edge having one bend does not introduce new rows nor new columns; an edge having two bends introduces exactly one column.

For this reason, in the worst case, $\frac{n}{2} + 1$ new columns are introduced. Therefore, $Total Area \leq h \times (n - h + \frac{n}{2} + 1)$ having its maximum value when $h = \frac{3}{4}n + \frac{1}{2}$; in this case we have:

$$Total Area \leq (\frac{3}{4}n + \frac{1}{2}) \times (\frac{3}{4}n + \frac{1}{2}).$$

The following example shows how the algorithm works on a typical input.

Consider cubic graph $\mathcal{G} = (V, E)$ in Fig. 3.29.a and one of its spanning trees \mathcal{T} (Fig. 3.29.b). In Fig. 3.30, \mathcal{T} is embedded in the grid according to all the parameters mentioned in the algorithm. Fig. 3.31 shows all the directions assigned to non-tree edges and the directly 1-bend edges. Finally, Fig. 3.32 and 3.33 show the output both in the complete case and without the elimination of 2-bend edges.

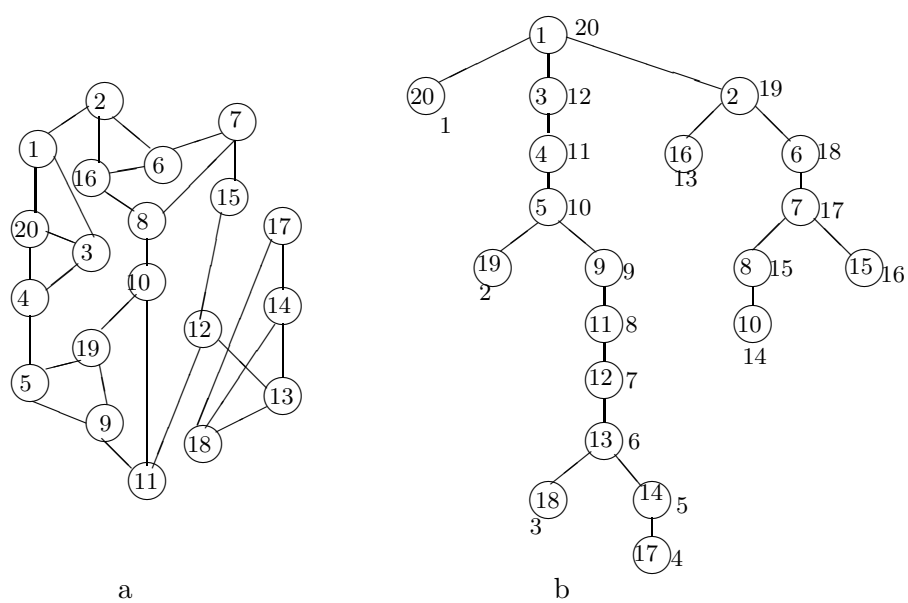


Figure 3.29: A cubic graph \mathcal{G} and a spanning tree \mathcal{T} for \mathcal{G} with a post-order numbering.

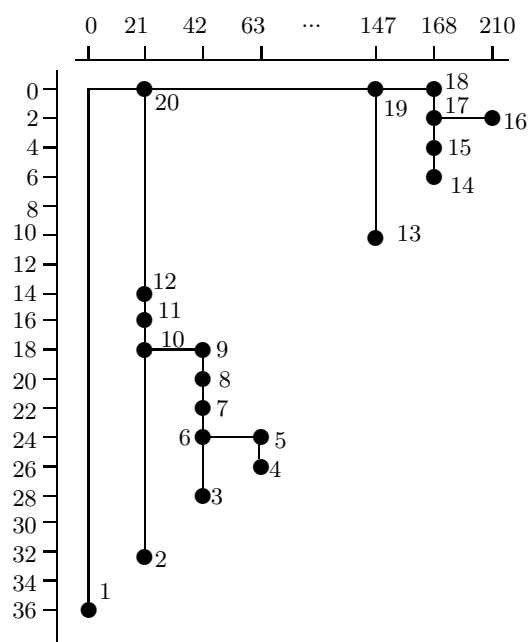


Figure 3.30: The embedding of \mathcal{T} on a grid.

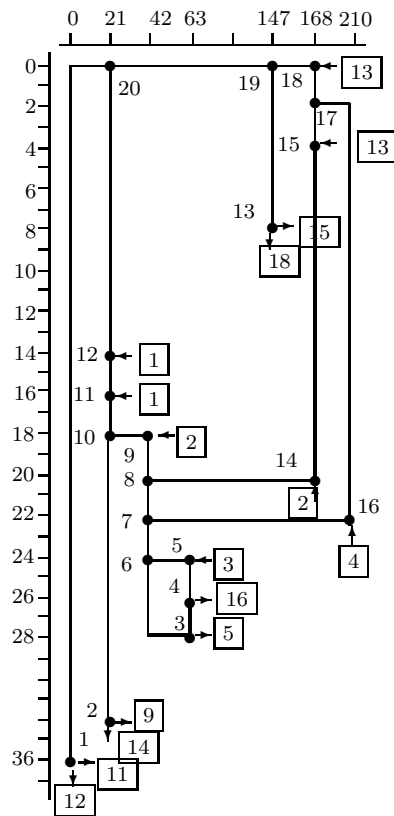


Figure 3.31: The partial drawing of \mathcal{G} after the execution of the first three phases of the second step of the algorithm. If an arrow goes either from a vertex i to a number j into a square or vice-versa, it means that the corresponding direction is assigned to the edge $\{i, j\}$.

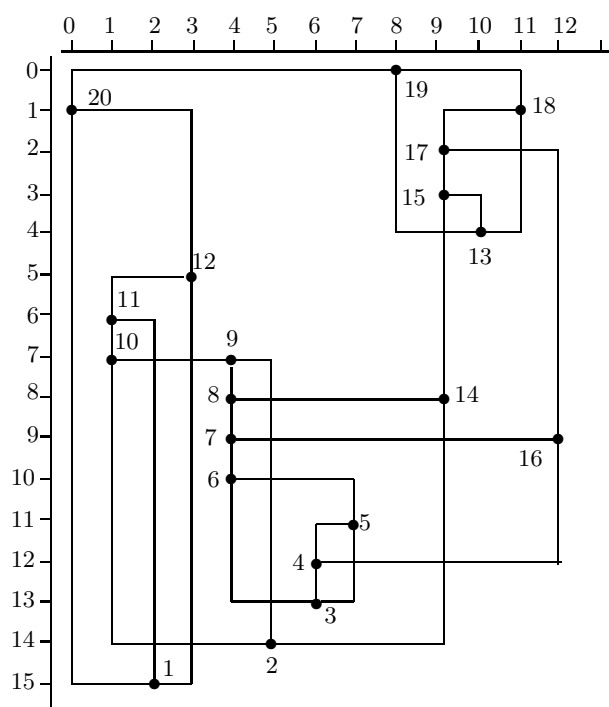


Figure 3.32: The 1-bend drawing of \mathcal{G} found by *ParCub*.

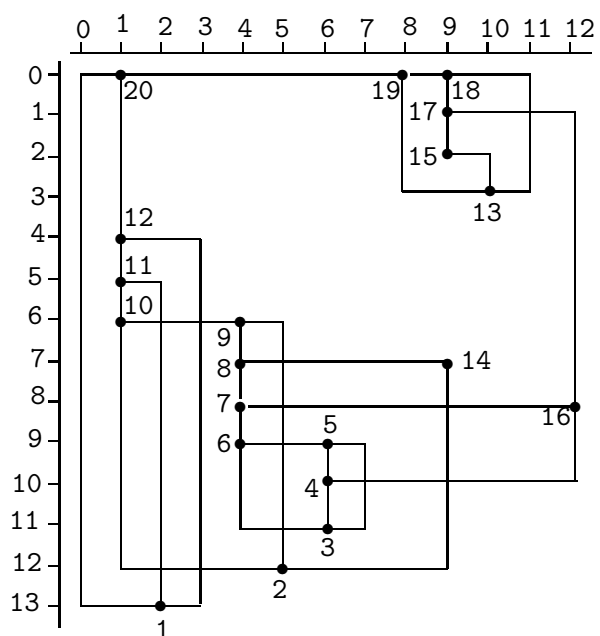


Figure 3.33: The 2-bend drawing of \mathcal{G} found by *ParCub*.

Chapter 4

Interconnection Networks

4.1 Introduction

Experience with the design and use of parallel computers indicates that a parallel computer (among other things) is largely dependent on the properties of the interconnection network that connects processors to memory or processors among themselves. Namely, the interconnection network not only affects the hardware architecture but also the nature of the system software (such as the network operating system).

Among all existing interconnection networks, the well known binary n -cube or hypercube [127] has been recognized as one of the most efficient; it has been used to design various commercial multiprocessor machines and it has been extensively studied. One of the reasons why the hypercube is so useful is that all of the algorithms for arrays, trees and meshes of trees can be “automatically” implemented on a hypercube. One drawback of hypercubes is that the degree of nodes increases (logarithmically) with the size of the network, so much that they are no longer suitable for applications involving a large number of nodes. Indeed, the complexity of the communications portion of a node can become fairly large as the number of nodes increases. Furthermore, the orthogonal layout of a network has sense only if the degree is not greater than four. All of these reasons lead us to consider low and fixed node degree networks as an important and essential component in designing any parallel and distributed system. In order to keep the good properties and, at the same time, to avoid the difficulties associated with high node degrees in hypercubes, several variations of the hypercube having similar computational properties but bounded degree (usually three or four)

have been devised. The most popular derivative networks are the butterfly [18], shuffle-exchange [87, 90, 110], de Bruijn [114], Beneš [13] and cube-connected cycles (CCC) [34, 117] networks. The latter one is modeled by a cubic graph derived from a hypercube. Informally, we can say that to obtain the CCC model, Transformation One (cf. Subsection 2.2.3) is applied to the hypercube. Besides these classical bounded degree networks, some others have been introduced very recently, like Trivalent Cayley networks [151]. They belong to the class of interconnection networks based on *Cayley graphs*, i.e. graphs whose adjacency structure is governed by a group. Usually, Cayley interconnection networks have an architecture with substantive advantages, in terms of algorithmic efficiency and fault tolerance. Support for their case comes in part by noting that many interconnection networks of algorithmic and commercial importance are Cayley graphs, including the hypercube, butterfly (with wraparound), cube-connected cycles, multiple rings [150] and star [3, 40] networks. For a survey on Cayley graphs and their properties, see [6, 7, 88].

There are several considerations to take into account in order to measure the ‘goodness’ of a network and eventually select one network instead over another one in the development of parallel computers. Some such parameters are degree, diameter, distribution of the disjoint paths between a pair of vertices in the graph and layout.

It is desirable that each pair of processors is connected but in this way the number of connections coming out from the same processor would increase arbitrarily when the network increases, while it is limited by physical characteristics. Therefore, the degree relates to the port capacity of the processors and, hence, to the hardware cost of the network.

Since not all processors are directly connected among them, in general it takes more than constant time to transfer data from one processor to another. The maximum communication delay between a pair of processors in a network is measured by the diameter of the graph. Thus, diameter is a measure of the running cost of the network.

Knowledge of the distribution of disjoint paths is crucial to the design of a routing table, which is a critical part of the network operating system. Further, since the number of parallel paths between a pair of nodes is limited by the degree of the underlying graph, the knowledge of this distribution is helpful in the evaluation of the fault tolerance of the network.

Layouts of graphs on rectilinear grids are of wide interest for their applications in the study of the VLSI layout problem for integrated circuits [144], as well as in the study of algorithms for drawing graphs. Further, each such layout is a restricted form of embedding of a graph in the grid [125], hence contributes to the study of the mapping problem for parallel architectures [17, 25], particularly the problem of mapping parallel programs onto mesh-structured parallel architectures [131].

Some networks are of primary interest in the next sections, so we collect their definitions and some related results here.

Since the processors of a network may be put in correspondence with the vertices of a graph and the communication links between processors may be seen as the edges connecting the vertices, networks can be conveniently modeled by using tools from graph theory [68]. Henceforth, we will use the terms ‘network’ and ‘graph’ interchangeably; the same holds for the terms ‘communication link’ and ‘edge,’ though usually the term ‘node’ will be preferred to ‘vertex.’

Butterfly networks. For each integer n , the n -level butterfly network \mathcal{B}_n has node-set $\{0, 1, \dots, n\} \times \{0, 1\}^n$ where $\{0, 1\}^n$ denotes the set of length- n binary strings.

For each $0 \leq \ell \leq n$, the set $\{\ell\} \times \{0, 1\}^n$ is the ℓ th level of \mathcal{B}_n . The nodes at level 0 of \mathcal{B}_n are called *inputs*, and those at level n are called *outputs* (the terms “input” and “output” derive from the fact that the $(n+1)$ -level butterfly network is the data-dependency graph of the 2^n -input Fast Fourier Transform algorithm [1]). The string $x \in \{0, 1\}^n$ is the *position-within-level string* (*PWL string*, for short) of node $\langle \ell, x \rangle$. Each node

$$\langle \ell, \beta_0\beta_1 \cdots \beta_{\ell-1}\beta_\ell\beta_{\ell+1} \cdots \beta_{n-1} \rangle$$

on level ℓ ($0 \leq \ell < n$; each $\beta_i \in \{0, 1\}$) of \mathcal{B}_n is connected by a *level- ℓ straight-edge* with node

$$\langle \ell + 1, \beta_0\beta_1 \cdots \beta_{\ell-1}\beta_\ell\beta_{\ell+1} \cdots \beta_{n-1} \rangle$$

on level $\ell + 1$, and by a *level- ℓ cross-edge* with node

$$\langle \ell + 1, \beta_0\beta_1 \cdots \beta_{\ell-1}\bar{\beta}_\ell\beta_{\ell+1} \cdots \beta_{n-1} \rangle$$

on level $\ell + 1$. When \mathcal{B}_n is drawn level by level, in such a way that, at each level, the PWL strings are the reversals of the binary representations of the

Figure 4.1: Two different views of \mathcal{B}_3 .

integers $0, 1, \dots, 2^n - 1$, in that order of the levels of \mathcal{B}_n , we get the familiar drawing of \mathcal{B}_n shown in Fig. 4.1.a when $n = 3$.

The following lemma will be useful in Section 4.2.

Lemma 2 [50] *For any non-negative integers j and k the subgraph of \mathcal{B}_n induced by the nodes of levels $j, j + 1, \dots, j + k$ is the disjoint sum of 2^{n-k} copies of \mathcal{B}_k .*

Note that we use the term “sum” here, rather than “union” to emphasize that the constituent graphs share neither nodes nor edges.

Trivalent Cayley Interconnection Networks. For each integer n , the n -dimensional Trivalent Cayley interconnection network (TCIN) $\mathcal{C}_n = (V, E)$ has a node set corresponding to a circular permutation in lexicographic order of n symbols, a_1, a_2, \dots, a_n , complemented or uncomplemented. Each edge is of the type $(v, \delta(v))$, where $\delta \in \{f, f^{-1}, g\}$, defined in the following way:

- $f(a_k^* a_{k+1}^* \cdots a_n^* a_1^* \cdots a_{k-1}^*) = a_{k+1}^* \cdots a_{k-1}^* \overline{a_k^*}$ (f -edge)
- $f^{-1}(a_k^* \cdots a_n^* a_1^* \cdots a_{k-2}^* a_{k-1}^*) = \overline{a_{k-1}^*} a_k^* \cdots a_{k-2}^*$ (f^{-1} -edge)
- $g(a_k^* \cdots a_n^* a_1^* \cdots a_{k-1}^*) = a_k^* \cdots \overline{a_{k-1}^*}$ (g -edge)

where $a_k^* \cdots a_n^* a_1^* \cdots a_{k-1}^*$ denotes the label of an arbitrary node and a_i^* denotes either a_i or $\overline{a_i}$. Notice that δ is closed under inverse since $g = g^{-1}$ and that the resulting graph is cubic.

The cardinality of V is $n2^n$ since for n distinct symbols there are exactly n different cyclic permutations and each symbol can be present either in complemented or uncomplemented form. The cardinality of E is $3n2^{n-1}$ for the cubicity of \mathcal{C}_n . Fig. 4.2 shows a TCIN of dimension three.

In the following we associate to each label starting with a_1 a binary string: the b_{n-k} -th bit is equal to 1 or to 0 according to the fact that symbol a_k , $k = 2, \dots, n$ is uncomplemented or complemented, respectively.

We call f -cycle a cycle consisting of only f -edges (equivalently f^{-1} -edges).

Fact 1 [151] *All the $n2^n$ nodes are partitioned into 2^{n-1} disjoint f -cycles of length $2n$. Nodes v and \overline{v} are in the same f -cycle.*

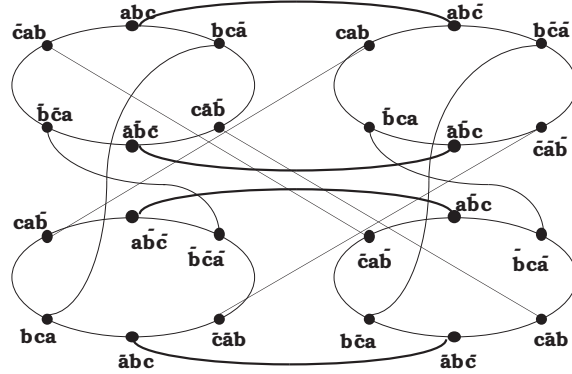


Figure 4.2: TCIN of dimension three.

Each f-cycle has a unique node starting with a_1 and the associated binary string, $b_{n-2} \dots b_0$, identifies the f-cycle.

On each f-cycle the node starting with a_1 is numbered one, all the other nodes are consecutively numbered from 2 to $2n$, clockwise.

This means that the k -th and the $(n+k)$ -th nodes start with a_k and $\overline{a_k}$ respectively, or vice-versa.

Grids. For integers m and n , the $m \times n$ grid (or, mesh) $\mathcal{M}_{m,n}$ has node-set

$$\{1, 2, \dots, m\} \times \{1, 2, \dots, n\}.$$

The edges of $\mathcal{M}_{m,n}$ connect nodes $\langle i, j \rangle$ and $\langle i', j' \rangle$ just when $|i-i'| + |j-j'| = 1$. The path induced by the set of nodes $\{i\} \times \{1, 2, \dots, n\}$ (respectively, the set $\{1, 2, \dots, m\} \times \{j\}$) is the i -th row (respectively, the j -th column) of $\mathcal{M}_{m,n}$. We call the product mn the area of grid $\mathcal{M}_{m,n}$.

Four auxiliary graphs will be used in our study. Two augmented versions of the butterfly network will be useful in constructing the layout of \mathcal{B}_n in Subsection 4.2.2. The complete bipartite graph and the Cube Connected Cycles network will be useful to prove the lower bound of area in Subsection 4.2.3 and in Subsection 4.3.2, respectively.

Augmented Butterfly Networks. We denote by \mathcal{B}'_n the network obtained by appending two new nodes, called *output terminals* to each output of

\mathcal{B}_n (see Fig. 4.3.a). We denote by \mathcal{B}_n'' the network obtained by appending two new nodes, called *input terminals* to each input of \mathcal{B}_n' (see Fig. 4.3.b). We refer to the input terminals collectively and to the output terminals collectively as a *terminal group*.

Figure 4.3: a. \mathcal{B}_2' and b. \mathcal{B}_2'' .

The following result, a proof of which can be found in [50], indicates that the designations “input” and “output” in our definitions of \mathcal{B}_n and \mathcal{B}_n'' are artificial and are only useful as an aid in visualizing the networks.

Lemma 3 *There is an automorphism of \mathcal{B}_n that maps each level $0 \leq \ell \leq n$ of the networks onto level $n - \ell$. There is a similar automorphism of \mathcal{B}_n'' , that maps the input terminals onto the output terminals and vice-versa.*

Complete Bipartite Graphs. The $N \times N$ complete bipartite graph $\mathcal{K}_{n,n}$ has n input nodes $V^{(i)}$ and n output nodes $V^{(o)}$; its edges connect every input $u \in V^{(i)}$ with every output $v \in V^{(o)}$.

Cube Connected Cycles Networks. The r -dimensional Cube Connected Cycles network CCC_r is constructed from the r -dimensional hypercube by replacing each node of the hypercube with a cycle of r nodes in CCC_r . The i -th dimension edge incident to a node of the hypercube is then connected to the i -th node of the corresponding cycle in CCC_r . CCC_r has $r2^r$ nodes each with degree three. By modifying the labeling scheme of the hypercube, we can represent each node by a pair $\langle w, i \rangle$ where i ($1 \leq i \leq r$) is the position of the node within its cycle and w (an r -ary bit binary string) is the label of the node in the hypercube that corresponds to the cycle. Therefore, two nodes $\langle w, i \rangle$ and $\langle w', i' \rangle$ are connected by an edge in CCC_r if and only if either:

$$w = w' \text{ and } i - i' = \pm 1 \pmod{r} \text{ or}$$

$$i = i' \text{ and } w \text{ differs from } w' \text{ in precisely the } i\text{-th bit.}$$

Now, we recall some definitions and give some preliminary results.

Definition 5 An embedding of graph \mathcal{G} into graph \mathcal{H} (which has at least as many nodes as \mathcal{G}) comprises a one-to-one association α of the nodes of \mathcal{G} with nodes of \mathcal{H} , plus a routing ρ which associates each edge (u, v) of \mathcal{G} with a path in \mathcal{H} that connects nodes $\alpha(u)$ and $\alpha(v)$.

Definition 6 The congestion of embedding $\langle \alpha, \rho \rangle$ is the maximum, over all edges e in \mathcal{H} , of the number of edges in \mathcal{G} whose ρ -routing paths contain edge e .

The notion of a *grid-layout* of a graph \mathcal{G} can be formulated as a special kind of embedding of \mathcal{G} into a grid; there are alternative, equivalent formulations of the notion which make it a special kind of drawing of \mathcal{G} in the plane. We follow the formulation given in [144].

Definition 7 A layout of an N -node graph \mathcal{G} in a grid $\mathcal{M}_{m,n}$, where $N \leq mn$, is an embedding $\langle \alpha, \rho \rangle$ of \mathcal{G} into $\mathcal{M}_{m,n}$ whose routing paths collectively satisfy the following conditions:

- Distinct routing paths are edge-disjoint, so the embedding that embodies a layout has unit congestion. It follows that at most two routing paths can “cross” at a node of $\mathcal{M}_{m,n}$, i.e., touch the node without terminating there.
- Routing paths sharing an intermediate node of $\mathcal{M}_{m,n}$ must cross at that node; that is, one path enters the node from the left and leaves toward the right, while the other path enters the node from the bottom and leaves toward the top. Thus, we do not allow “knock-knee” routing [101].
- A routing path may touch no image node $\alpha(u)$, except at its endpoints.

Definition 8 The minimum bisection width of a graph \mathcal{G} , $MBW(\mathcal{G})$, is the smallest number of edges whose removal partitions \mathcal{G} into two disjoint subgraphs, each containing half of the nodes.

Lemma 4 [144] For any graph \mathcal{G} , the area of the smallest grid in which \mathcal{G} can be laid out is greater or equal to $(MBW(\mathcal{G}) - 1)^2$.

Following the same reasoning detailed in Subsection 4.2.3, it is possible to prove the next lemma.

Lemma 5 [90] *Let ϵ be an embedding of a graph \mathcal{G} into a graph \mathcal{H} that has congestion C , then the following inequality holds:*

$$MBW(\mathcal{H}) \geq \frac{1}{C} MBW(\mathcal{G}).$$

As a consequence of Lemmas 4 and 5, the lower bound of the area of a network \mathcal{H} is computed through an embedding ϵ into \mathcal{H} of a graph \mathcal{G} whose MBW is known. Moreover, we need to know the congestion C of ϵ . In this way, we have that:

$$\begin{aligned} \text{lower bound on the layout area of } \mathcal{H} &\geq (MBW(\mathcal{H}) - 1)^2 \\ &\geq \left(\frac{1}{C} MBW(\mathcal{G}) - 1\right)^2. \end{aligned}$$

This formula provides a general method for computing a lower bound on the area of a layout of a graph \mathcal{H} when an embedding of congestion C for an auxiliary graph \mathcal{G} into \mathcal{H} is known, and $MBW(\mathcal{G})$ is given.

In the rest of this chapter we deal with two classes of networks both Cayley and fixed node degree graphs. In particular, in Section 4.2 an optimal layout of the butterfly network is exhibited and it is proved that no better layouts may exist. In Section 4.3 Trivalent Cayley interconnection networks are studied. Related to them, both the optimal routing and the layout problems are investigated.

4.2 A Tight Layout of the Butterfly Network

4.2.1 Introduction

The fields of graph embedding and VLSI layout have developed powerful techniques which produce embeddings and layouts which are quite efficient –often within constant factors of optimal [18, 20]. However, even a modest constant factor may render an asymptotically optimal layout or embedding unacceptably inefficient in practice. This observation motivates the work explained in this section. Namely, we find a grid-layout of the butterfly network [18] whose deviation from optimality is of lower order than a constant factor. This goal is achieved by presenting, in Subsection 4.2.2, a layout of the N -input, N -output butterfly network whose area is $(1 + o(1))N^2$, and

by proving, in Subsection 4.2.3, that no layout of this network can have area smaller than $(1 - o(1))N^2$. Thus, upper and lower bounds coincide up to a low-order additive term.

Both the upper and lower bound components of this result improve prior bounds for butterfly network layouts. The previously best known lower bound for the layout area of the N -input, N -output butterfly network was $\frac{1}{4}N^2$ [144, 145]. The 1981 upper bound of $2N^2$ for the same problem [159] was improved only in 1992, to $\frac{11}{6}N^2$ [43].

4.2.2 The Upper Bound on Layout Area

The layout of \mathcal{B}_n proceeds by finding layouts of subgraphs of \mathcal{B}_n and “splicing” them together. This intuitive operation is now formally defined.

Suppose we are given a graph \mathcal{G} , a graph \mathcal{H} , a sequence $\sigma = \langle u_1, u_2, \dots, u_k \rangle$ of distinct nodes of \mathcal{G} , and an equal-size sequence $\sigma' = \langle v_1, v_2, \dots, v_k \rangle$ of distinct nodes of \mathcal{H} . Say that \mathcal{G} has nodes $U \cup \{u_1, u_2, \dots, u_k\}$ and that \mathcal{H} has nodes $V \cup \{v_1, v_2, \dots, v_k\}$, where the sets U , V , $\{u_1, u_2, \dots, u_k\}$ and $\{v_1, v_2, \dots, v_k\}$ are pairwise disjoint. The operation of *splicing graphs \mathcal{G} and \mathcal{H} along sequences σ and σ'* produces the graph \mathcal{F} whose nodes comprise the set

$$U \cup V \cup \{\langle u_1, v_1 \rangle, \langle u_2, v_2 \rangle, \dots, \langle u_k, v_k \rangle\}$$

and whose edges connect node w_1 and w_2 just when:

- $\{w_1, w_2\} \subseteq U$ (respectively, $\{w_1, w_2\} \subseteq V$), and w_1 and w_2 are adjacent in \mathcal{G} (respectively, in \mathcal{H});
- $w_1 = \langle u_i, v_i \rangle$, $w_2 \in U$, and u_i and w_2 are adjacent in \mathcal{G} ;
- $w_1 = \langle u_i, v_i \rangle$, $w_2 \in V$, and v_i and w_2 are adjacent in \mathcal{H} ;
- $w_1 = \langle u_i, v_i \rangle$, $w_2 = \langle u_j, v_j \rangle$, and either u_i and u_j are adjacent in \mathcal{G} , or v_i and v_j are adjacent in \mathcal{H} (or both).

Examples of Splicing. (a) One can splice one copy of $\mathcal{M}_{m,n}$ along its right side to another copy of $\mathcal{M}_{m,n}$ along its left side, to produce an instance of $\mathcal{M}_{m,2n-1}$. (b) One can produce \mathcal{B}_3 by appropriately splicing the disjoint sum of two copies of \mathcal{B}_2 with the disjoint sum of four copies of \mathcal{B}_1 ; the former sum produces the first two levels of \mathcal{B}_3 , the latter sum produces the last level of \mathcal{B}_3 , and the two sums combine to produce the third level of \mathcal{B}_3 ; cf. Fig. 4.1.a.

Our final example, depicted in Fig. 4.1.b, is so relevant to our layout that we encapsulate it as a lemma (whose proof is left to the reader).

Lemma 6 *Let the sequence $\sigma = \langle u_1, u_2, \dots, u_{2^{n+1}} \rangle$ list all the output terminals of \mathcal{B}'_n , in an arbitrary order. Splicing \mathcal{B}'_n with a copy of itself along σ produces \mathcal{B}_{n+1} .*

We finally have all the machinery we need to study grid layouts of butterfly networks. To simplify our exposition, henceforth let n be an arbitrary positive integer, and let $N = 2^n$.

Theorem 10 *For all positive integers n , there is a grid-layout \mathcal{L}_n of \mathcal{B}_n such that its area is not greater than $(1 + o(1))N^2$.*

Theorem 10 will be proved via a sequence of reductions.

The First Reduction

It is possible to construct the desired layout of \mathcal{B}_{n+2} from four copies of a suitable layout of \mathcal{B}''_n .

Lemma 7 *One can construct a grid-layout \mathcal{L}_{n+2} of \mathcal{B}_{n+2} with the area indicated in Theorem 10, from four copies of a grid-layout \mathcal{L}''_n of \mathcal{B}''_n , that has the following properties.*

- \mathcal{L}''_n places \mathcal{B}''_n in a $(2N + o(N)) \times (2N + o(N))$ grid \mathcal{M} ;
- \mathcal{L}''_n places one terminal group of \mathcal{B}''_n on a vertical side of \mathcal{M} and the other terminal group on a horizontal side of \mathcal{M} .

Proof Assume, with no loss of generality, that the given layout \mathcal{L}''_n places the terminal groups on the bottom and right sides (see Fig. 4.4.a). Flip \mathcal{L}''_n around its right side to produce layout $\widetilde{\mathcal{L}''_n}$ of \mathcal{B}''_n . Splice layouts \mathcal{L}''_n and $\widetilde{\mathcal{L}''_n}$ along the pivot line, as depicted in Fig. 4.4.b. By Lemma 6, the resulting layout, call it \mathcal{L}'_{n+1} , is a layout of \mathcal{B}'_{n+1} .

Next, flip layout \mathcal{L}'_{n+1} around its bottom to produce layout $\widetilde{\mathcal{L}'_{n+1}}$ of \mathcal{B}'_{n+1} . Splice layouts \mathcal{L}'_{n+1} and $\widetilde{\mathcal{L}'_{n+1}}$ along the pivot line to produce the layout \mathcal{L}_{n+2} ; see Fig. 4.4.c. By Lemma 6, \mathcal{L}_{n+2} is a layout of \mathcal{B}_{n+2} . (Note the implicit use of Lemma 3 here.) Clearly, layout \mathcal{L}_{n+2} resides in a $(4N + o(N)) \times (4N + o(N))$ grid, hence is the layout specified in the theorem.

Figure 4.4: a. \mathcal{L}_n'' : a layout of \mathcal{B}_n'' ; b. \mathcal{L}_{n+1}' : a layout of \mathcal{B}_{n+1}' ; c. \mathcal{L}_{n+2} : a layout of \mathcal{B}_{n+2} .

Thus, the layout problem has been reduced to one of producing a layout \mathcal{L}_n'' , as used in Lemma 7.

The Second Reduction

The layout \mathcal{L}_n'' of Lemma 7 is now constructed.

Lemma 8 *Suppose it is possible to lay any \mathcal{B}_m'' out in a $(2^{m+1} + o(2^m)) \times o(4^m)$ grid, in such a way that each terminal group of \mathcal{B}_m'' resides on one of the length- $(2^{m+1} + o(2^m))$ (vertical) sides of the grid. Then one can construct the grid-layout \mathcal{L}_n'' of \mathcal{B}_n'' described in Lemma 7.*

Proof Let the levels of \mathcal{B}_n'' be numbered $-1, 0, \dots, n, n+1$, where levels (-1) and $(n+1)$ are the terminal groups. We create our layout of \mathcal{B}_n'' in stages.

First, pick any $k \in \{0, 1, \dots, n-1\}$, and construct the graph $\mathcal{B}_n^{(k)}$ from \mathcal{B}_n'' by placing a new node—called a *token*—on every edge connecting levels k and $k+1$ of \mathcal{B}_n'' (or, equivalently, by replacing every such edge by a length-2 path). Note that $\mathcal{B}_n^{(k)}$ has $n+4$ levels, numbered $-1, 0, 1, \dots, n, n+1, n+2$, with the tokens residing in level $k+1$. Clearly, any layout of $\mathcal{B}_n^{(k)}$ is also a layout of \mathcal{B}_n'' .

Next, decompose $\mathcal{B}_n^{(k)}$ along the token-level into

- $\mathcal{G}_{k,1}$: the induced subgraph of $\mathcal{B}_n^{(k)}$ on levels $-1, \dots, k+1$
- $\mathcal{G}_{k,2}$: the induced subgraph of $\mathcal{B}_n^{(k)}$ on levels $k+1, \dots, n+2$.

Easily, one can obtain $\mathcal{B}_n^{(k)}$ by splicing $\mathcal{G}_{k,1}$ and $\mathcal{G}_{k,2}$ along the replicated level. Importantly, by Lemma 2, $\mathcal{G}_{k,1}$ is the disjoint sum of 2^{n-k} copies of \mathcal{B}_k'' , while $\mathcal{G}_{k,2}$ is the disjoint sum of 2^{k+1} copies of \mathcal{B}_{n-k-1}'' .

For definiteness, let us now assume that n is odd, and let us consider the graphs $\mathcal{B}_n^{(k)}$, $\mathcal{G}_{k,1}$, and $\mathcal{G}_{k,2}$ when $k = (n-1)/2$. When n is even, we must adjust the details of our layout and its analysis, but only in ways that affect low-order terms; details are omitted because the analysis is similar. In the

case at hand, both $\mathcal{G}_{k,1}$ and $\mathcal{G}_{k,2}$ are disjoint sums of $2^{(n+1)/2}$ disjoint copies of $\mathcal{B}_{(n-1)/2}''$.

Figure 4.5: Resplicing $\mathcal{L}^{(1)}$ and $\mathcal{L}^{(2)}$.

Now it is possible to construct the desired layout of \mathcal{B}_n'' from the claimed layout of $\mathcal{B}_{(n-1)/2}''$. To this end, let \mathcal{L} be a layout of $\mathcal{B}_{(n-1)/2}''$ in a $(2^{(n+1)/2} + o(2^{n/2})) \times o(2^n)$ grid, in which the terminal groups reside on opposing vertical sides (of size $2^{(n+1)/2} + o(2^{n/2})$).

First construct a layout $\mathcal{L}^{(1)}$ of $\mathcal{G}_{(n-1)/2,1}$, by abutting $2^{(n+1)/2}$ copies of \mathcal{L} , with its token level on the left side, along their (long) horizontal sides. Note that these grids are *not* spliced: abutting two copies of the $m \times n$ grid along vertical sides creates a copy of the $m \times 2n$ grid. Layout $\mathcal{L}^{(1)}$ resides in a $(2N + o(N)) \times o(N)$ grid.

Next, rotate layout $\mathcal{L}^{(1)}$ by 90 degrees to produce $\mathcal{L}^{(2)}$, a layout of $\mathcal{G}_{(n-1)/2,2}$ with the token level on the top side.

As the next to last step, place layouts $\mathcal{L}^{(1)}$ and $\mathcal{L}^{(2)}$ in the smallest grid \mathcal{M} which will hold them in the following non-overlapping configuration. Position layout $\mathcal{L}^{(1)}$ flush with the top and right sides of \mathcal{M} , and position layout $\mathcal{L}^{(2)}$ flush with the left and bottom sides of \mathcal{M} (see Fig. 4.5). One verifies easily that a $(2N + o(N)) \times (2N + o(N))$ grid is large enough to accommodate these placements of layouts $\mathcal{L}^{(1)}$ and $\mathcal{L}^{(2)}$.

Finally, splice $\mathcal{G}_{(n-1)/2,1}$ and $\mathcal{G}_{(n-1)/2,2}$ along the token level, in order to recreate $\mathcal{B}_n^{((n-1)/2)}$. Since all of the nodes to be “merged” have unit degree, we can accomplish the splicing by routing a specific bijection from nodes on the left side of $\mathcal{L}^{(1)}$ to nodes on the top side of $\mathcal{L}^{(2)}$. Our positioning of layouts $\mathcal{L}^{(1)}$ and $\mathcal{L}^{(2)}$ within \mathcal{M} has left a large unpopulated area (as one can see in Fig. 4.5) in which any such bijection can be routed in a cross-bar fashion.

This completes the layout of $\mathcal{B}_n^{((n-1)/2)}$, hence of \mathcal{B}_n'' .

The Third Reduction

The final task is to construct the layouts of \mathcal{B}_n'' demanded in Lemma 8. These layouts will be constructed implicitly, by appealing to a result of Pinter [113] on channel routing.

An (h, l, k) cross-channel routing problem involves an $h \times l$ grid (the channel) and a set of k two-point nets: each net is a pair of gridpoints that reside on opposite vertical sides of the grid. The problem is to construct k edge-disjoint paths that connect every net and which can simultaneously be laid out in the grid. Such a layout may not be possible if the grid is too small; Pinter guarantees that a grid, which is not too big, is enough.

Lemma 9 [113] *Any (h, l, k) cross-channel routing problem satisfying $h > k$ and $l > \frac{3}{2}k + 1$ can be routed within the given grid.*

Lemma 9 enables the desired layouts of \mathcal{B}_n'' in the following way.

Lemma 10 *One can lay \mathcal{B}_n'' out in an $(2^{n+1} + 1) \times O(n2^n)$ grid, in such a way that each terminal group of \mathcal{B}_n'' resides on one of the vertical sides.*

Proof We place each of the $n + 1$ internal levels of nodes of \mathcal{B}_n'' in a $(2^{n+1} + 1) \times (2^n + 2)$ grid, in the staggered fashion depicted in Fig. 4.6. That is, the nodes of a level are placed on grid-points of the form $\langle 2i, i + 1 \rangle$, where $i = 1, \dots, 2^n$. (We shall see momentarily that Lemma 9 allows us to specify the exact mapping of butterfly nodes to these grid-points in any arbitrary way.)

Figure 4.6: A layout of \mathcal{B}_2'' .

Now, route four edges out of each node to four “terminals,” two on each vertical side of the grid, as depicted in Fig. 4.6. Align the layouts of the $n + 1$ internal levels of \mathcal{B}_n'' horizontally, keeping a space of $\frac{3}{2}2^{n+1} + 2$ between them, so that we can apply Lemma 9. Easily, this produces the claimed layout of \mathcal{B}_n'' in a $(2^{n+1} + 1) \times O(n2^n)$ grid.

Figure 4.7: The overall layout of \mathcal{B}_n .

Now all the machinery necessary to create the layout of Theorem 10 has been provided. The overall layout of \mathcal{B}_n implicit in our proof is depicted in

Fig. 4.7; the inputs and outputs of the network are on the middle vertical and horizontal lines, respectively; the shadowed areas contain no butterfly nodes, being dedicated to routing butterfly edges.

4.2.3 The Lower Bound on Layout Area

This section is devoted to proving the lower bound on the layout area of \mathcal{B}_n .

Theorem 11 *For all positive integers n , any grid-layout of \mathcal{B}_n has area at least $(1 - o(1))N^2$.*

We modify the basic lower-bound strategy invented in [144] and hinted at in Section 4.1 via the non-standard notion of *special-bisection* of a graph.

Let \mathcal{G} be a graph having a designated set of $2c > 0$ *special* nodes. The *minimum special-bisection width* of \mathcal{G} , denoted $MSBW(\mathcal{G})$, is the smallest number of edges whose removal partitions \mathcal{G} into two disjoint subgraphs, each containing half of \mathcal{G} 's special nodes.

Lemma 11 *For any graph \mathcal{G} , it is not possible to lay \mathcal{G} out in an area less than $(MSBW(\mathcal{G}) - 1)^2$.*

Proof We consider an arbitrary layout of \mathcal{G} in $\mathcal{M}_{m,n}$, where, without loss of generality, $m \leq n$. As in [144], we find there is a line L that has a single unit-length jog, which can be positioned on a drawing of $\mathcal{M}_{m,n}$ in the following way.

- L is aligned with the columns of $\mathcal{M}_{m,n}$ in such a way that the portion of L above the jog lies to the left of some column c of $\mathcal{M}_{m,n}$; the jog of L lies below some row of $\mathcal{M}_{m,n}$; the portion of L below the jog either lies outside of $\mathcal{M}_{m,n}$, or it lies to the right of column c .
- Removing the grid-edges crossed by L yields a special-bisection of \mathcal{G} .

By definition, at least $MSBW(\mathcal{G})$ edges of \mathcal{G} must cross line L . By construction, at most $m + 1$ edges of $\mathcal{M}_{m,n}$ cross line L . It follows that $m \geq MSBW(\mathcal{G}) - 1$, hence the lemma follows.

The next goal is to show that, when the input and output nodes of \mathcal{B}_n are designated as special, then $MSBW(\mathcal{B}_n) \geq 2^n$. To this end, employ the $N \times N$ complete bipartite graph $\mathcal{K}_{N,N}$ as an auxiliary graph. First, we note that, if we designate all nodes of $\mathcal{K}_{N,N}$ as special, we obtain the following lower bound on $MSBW(\mathcal{K}_{N,N})$.

Lemma 12 $MSBW(\mathcal{K}_{N,N}) = \frac{1}{2}N^2$ when all nodes of $\mathcal{K}_{N,N}$ are special.

Proof Consider an arbitrary linearization of the nodes of $\mathcal{K}_{N,N}$. Cut the linearization in half. Say that this cut places K input nodes on one side of the cut, hence $N - K$ on the other. Clearly, the output nodes of $\mathcal{K}_{N,N}$ are cut in exactly complementary proportions. Since $\mathcal{K}_{N,N}$ has an edge between every input and every output, the K inputs and K outputs that are separated by the cut give rise to K^2 edges crossing the cut, while the $N - K$ inputs and $N - K$ outputs that are separated by the cut give rise to $(N - K)^2$ cut edges. We thus have $K^2 + (N - K)^2$ edges of $\mathcal{K}_{N,N}$ crossing the cut. This quantity is minimized when $K = \frac{1}{2}N$, in which case $\frac{1}{2}N^2$ edges cross the cut.

We employ a technique for bounding unknown $MSBW$'s from known ones, which derives from a technique originated in [90] and refined in [126] making use of congestion arguments.

Focus on a graph \mathcal{H} which has k special nodes, whose $MSBW$ we wish to bound from below. Suppose to have an auxiliary graph \mathcal{G} which has k special nodes, whose $MSBW$ we know. Suppose further to have an embedding ϵ of \mathcal{G} into \mathcal{H} , which maps the special nodes of \mathcal{G} onto the special nodes of \mathcal{H} , such that the congestion of ϵ does not exceed C . The claim is that $MSBW(\mathcal{H}) \geq (1/C)MSBW(\mathcal{G})$. This inequality holds because the embedding ϵ allows us to view the act of partitioning \mathcal{H} into two disjoint subgraphs having equinumerous sets of special nodes as simultaneously partitioning \mathcal{G} into two disjoint subgraphs having the same partition of special nodes. With this view in mind, one can consider the act of removing any particular edge e of \mathcal{H} as effectively removing all edges of \mathcal{G} that are routed over e by the embedding ϵ . If we know that ϵ never routes more than C edges of \mathcal{G} over any edge of \mathcal{H} , which is what our upper bound on the congestion of ϵ means, then we know that cutting an edge of \mathcal{H} simultaneously cuts no more than C edges of \mathcal{G} . Since we also know that at least $MSBW(\mathcal{G})$ edges of \mathcal{G} must be cut in order to effect the desired partition of \mathcal{G} , we can infer that at least $MSBW(\mathcal{H}) \geq (1/C)MSBW(\mathcal{G})$ edges of \mathcal{H} must be cut in order to effect the desired partition of \mathcal{H} . This argument yields the following lemma.

Lemma 13 (The Congestion Lemma) *Let \mathcal{G} and \mathcal{H} be graphs having equal numbers of special nodes. If there is an embedding of \mathcal{G} into \mathcal{H} which maps special nodes to special nodes and which has congestion less than or equal to C , then*

$$MSBW(\mathcal{H}) \geq (1/C)MSBW(\mathcal{G}).$$

The lower bound can be obtained via the congestion technique, by letting $N = 2^n$ and analyzing the “natural” embedding of $\mathcal{K}_{N,N}$ (which plays the role of the guest graph \mathcal{G}) into \mathcal{B}_n (which plays the role of the host graph \mathcal{H}).

Lemma 14 *One can embed $\mathcal{K}_{N,N}$ into \mathcal{B}_n with congestion $2^{n-1} = \frac{1}{2}N$, in such a way that the inputs and outputs of $\mathcal{K}_{N,N}$ map, respectively, to the inputs and outputs of \mathcal{B}_n .*

Proof Note that \mathcal{B}_n has the ‘banyan’ property: each input node u is connected to each output node v by exactly one path of length n . Consider any embedding ϵ of $\mathcal{K}_{N,N}$ into \mathcal{B}_n which assigns inputs of $\mathcal{K}_{N,N}$ to inputs of \mathcal{B}_n and outputs of $\mathcal{K}_{N,N}$ to outputs of \mathcal{B}_n , and which routes the edges of $\mathcal{K}_{N,N}$ via the unique length- n path which connects the two end-points in \mathcal{B}_n .

Now the congestion of embedding ϵ will be analyzed. Call a path of \mathcal{B}_n *simple* if it does not visit any level twice. Let e be a level- k edge of \mathcal{B}_n . Since \mathcal{B}_n has the banyan property, one endpoint of e reaches precisely 2^{n-k-1} distinct output nodes via simple paths, while the other endpoint of e reaches precisely 2^k distinct input nodes via simple paths. Hence, edge e lies on precisely 2^{n-1} input-to-output simple paths; i.e., its congestion is precisely 2^{n-1} .

If we now designate all nodes of $\mathcal{K}_{N,N}$ as special and the input and output nodes of \mathcal{B}_n as special, we infer from Lemmas 12, 13 and 14 a lower bound on the *MSBW* of \mathcal{B}_n .

Lemma 15 $MSBW(\mathcal{B}_n) \geq 2^n$.

Finally, Lemma 15 combines with Lemma 11 to yield the desired lower bound, Theorem 11, on the area of grid-layouts of \mathcal{B}_n .

4.3 On Trivalent Cayley Interconnection Networks

4.3.1 Introduction

Trivalent Cayley interconnection networks (TCIN) [151] were introduced in 1995 as ‘good’ networks since having interesting properties like fixed degree, regularity, logarithmic diameter and maximal fault tolerance. Simply because of some of these properties, it make sense to study TCINs and to

use their cubicity to solve some typical networks' problems. In this section we look into Trivalent Cayley interconnection networks and show a new three-dimensional representation of their structure. This model consists in layering certain cycles of the network and allows us to bring to the surface some properties that are useful both in obtaining an optimal drawing on the grid and in computing a minimum routing.

For the first problem we prove that the lower bound on area is $\Omega(2^{n-1} \times 2^{n-1})$ and we show a method to suitably draw the network in a grid having size of the same order. For the second problem, we present a simple algorithm—more intuitive than the algorithm presented in [152]—working in linear time with respect to the length of the route found.

The formal definition of n -dimensional TCIN was given in Section 4.1. We have also already underlined the existence of 2^{n-1} disjoint f -cycles (Fact 1, Section 4.1) Here some observations and preliminary results are listed.

Definition 9 We define distance $d(u, v)$ on a f -cycle between two nodes u and v the minimum between the number of edges on the path $P(u, v)$ and on the path $P(v, u)$.

Following function g , each f -cycle is connected to exactly n different f -cycles.

Now, the connections between different f -cycles are analyzed. For the symmetry of the structure, it is possible to consider only nodes from 1 to n on each f -cycle, since each node $n + k$ has the same behavior than node k .

Fact 2 Given an f -cycle $b_{n-2} \dots b_0$, each node $k = a_k^* \dots a_n^* a_1^* \dots a_{k-1}^*$ on it is connected by function g to the node $v = a_k^* \dots a_n^* a_1^* \dots a_{k-1}^*$. If $k \neq 2$, node v is the k -th node on the cycle $b_{n-2} \dots b_{n-k+1} \dots b_0$. In other words, we may say that node k skips $2^{n-k+1} - 1$ cycles to find its adjacent node v . Namely, node 1 skips no cycles and nodes 3, 4, 5, ... skip 1, 3, 7, ... cycles, respectively. The only node that has a behavior different from the other ones is node 2 ($n + 2$) in view of the fact that function g complements a_1 . Then, node 2 of a fixed cycle $b_{n-2} \dots b_0$ is connected to the $(n + 2)$ -th node (2 -nd node) of the cycle $b_{n-2} \dots b_0$.

Fact 1 (Section 4.1) and Fact 2 above lead us to see the network as the stratification of 2^{n-1} f -cycles jointed by vertical and slanting connections (see Fig. 4.8).

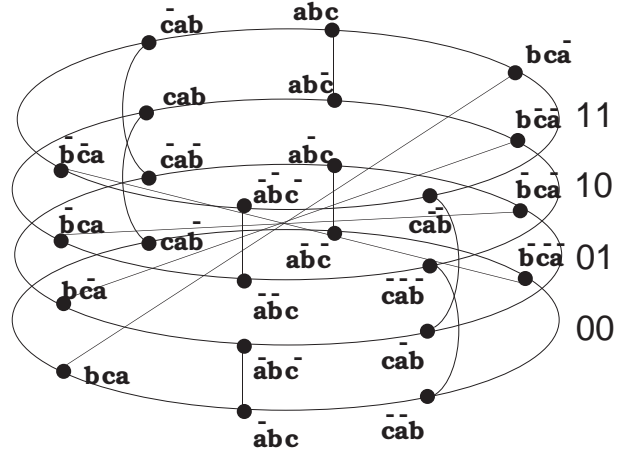


Figure 4.8: Stratification of f -cycles of the TCIN of dimension 3.

Then, given any two cycles C_i and C_j either they are not joined or they are joined by exactly two symmetric g -edges. These g -edges will be called *gate edges*, shortly *gates*. In the following, we will analyze the properties of the stratification structure through its projection on a plane in the direction of its axis. It is to notice that the projection of a gate is a point except the gate $(2, n + 2)$ projected in a segment called *bridge*. Working on the projection transforms the shortest routing problem on a TCIN into a shortest path problem on a cycle.

4.3.2 Optimal Layout of a TCIN

In this section we will present a method to lay a TCIN out in $O(2^{n-1} \times 2^{n-1})$ area, and then we will prove that this is an optimum value for the area.

The stratification structure of Fig. 4.8 let us understand that the layout depends strongly on the mutual position of f -cycles and on the drawing of g -edges. Indeed, the drawing of f - and f^{-1} -edges consists of the representation on the grid of the cycles to which they belong.

We first use the standard representation of cycles in networks: the mapping to rectangles of height one [91].

All vertices lay on the same side of each rectangle; therefore we can represent a rectangle as flattened on a segment (see Fig. 4.9). Moreover,

because of the symmetry of TCINs, we may consider only half of the nodes.

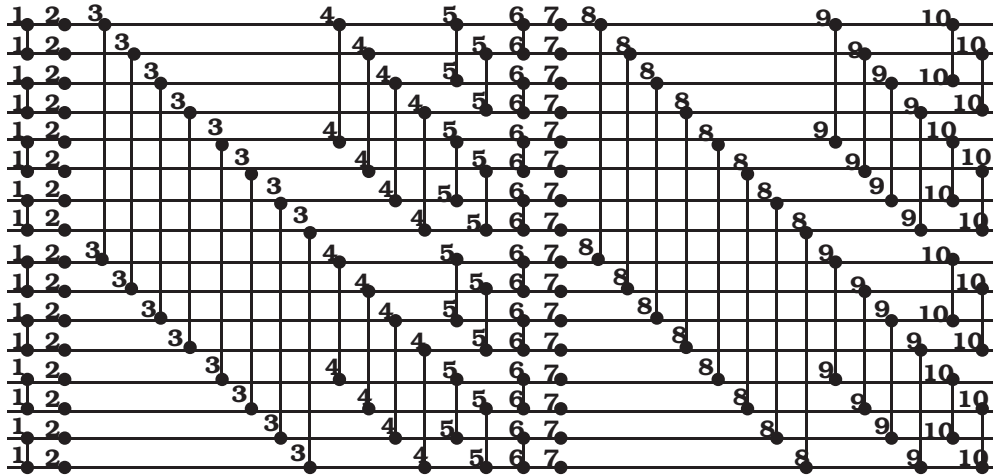


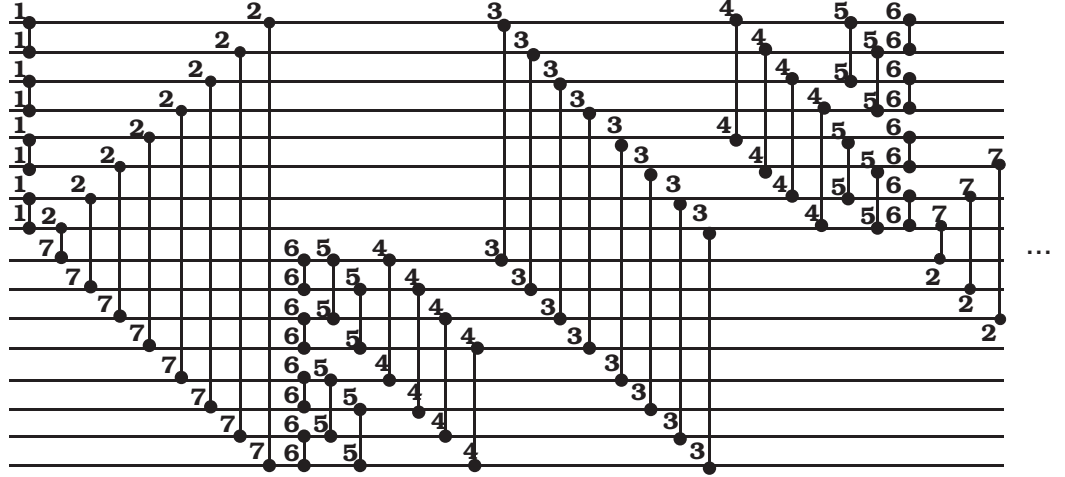
Figure 4.9: Rectilinear scheme of the stratification model.

From the previous considerations about f -cycles and from Fact 2, it follows that, for a fixed $k \neq 2$, the length of a g -edge connecting two k -th nodes is 2^{n-k+1} , while if $k = 2$, the length of g -edges incident to a second node is not fixed. The fact that 2 is not connected to one of its homonyms (not considering the cycles which they belong to) implies that all these g -edges are not rectilinearly drawn in the scheme of Fig. 4.9.

In order to obtain an orthogonal drawing and to optimize the area, it is profitable to change the order of the nodes on the cycles from $2^{n-2} + 1$ to 2^{n-1} (see Fig. 4.10 that is obtained from Fig. 4.9). This choice implies that all the blocks of g -edges with the same end-points k and length no greater than 2^{n-2} are divided into two symmetrical sub-blocks of width 2^{n-k+1} , where $4 \leq k \leq n+1$. As consequence, the area for representing g -edges with end-point k will be duplicated.

Notice that blocks related to g -edges with end-points numbered 4 can be fit into the block related to node 3. The general layout for a TCIN is as shown in Fig. 4.11 and its area is $6 \cdot 2^{n-1} \times 2^{n-1}$, obtained by multiplying:

- height= $2 \cdot 2^{n-1}$;
 each of 2^{n-1} cycles is drawn by using two rows.

Figure 4.10: Layout of C_5 with the stratification model.

- width = $3 \cdot 2^{n-1}$:

1. g -edges with end-point numbered 2 and 3 are both drawn in 2^{n-1} columns;
2. also g -edges with end-point numbered 4 need 2^{n-1} columns; since they share $2^{n-1} - 2$ columns with g -edges connecting nodes numbered 3, two columns is their contribution to the computation of the total width;
3. each of the remaining blocks of g -edges with end-point k needs $2 \cdot 2^{n-k+1}$ columns, $4 < k \leq n + 1$.

Since we are considering half of all the nodes on each f -cycle, the sum of all these contributions gives half of the total width.

The layout scheme in Fig. 4.11 shows that some space is wasted. In order to reduce the area, we transform the drawing of cycles from a stratification model to a concentric one, as shown in Fig. 4.12. The area of this new scheme is the best one achieved here and is $9/2 \cdot 2^{n-1} \times 2^{n-1}$. Height is the same as in the flattened model. Width differs from high just for the width of the most interior cycle, that is equal to one plus the value computed in Step 3

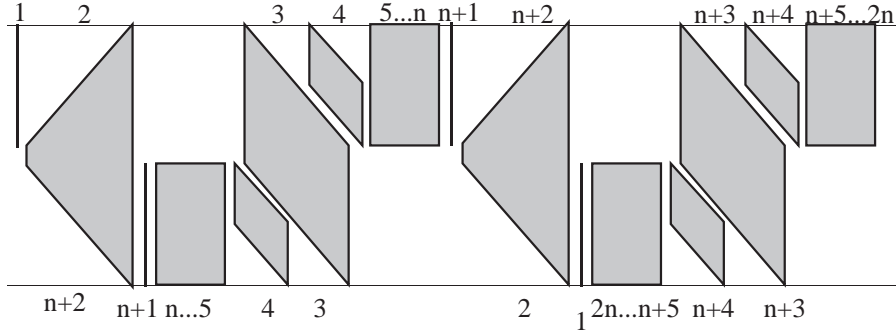


Figure 4.11: General scheme of a TCIN on flattened rectangles.

of the flattened model. The addition of one corresponds to the contribution of one of the two columns of Step 2.

As consequence of all previous arguments, we have:

Theorem 12 *An n -dimensional TCIN can be orthogonally drawn in area $O(2^{n-1} \times 2^{n-1})$.*

This result is optimal to within a constant factor since it is of the same order of the lower bound, as the following theorem states:

Theorem 13 *$\Omega(2^{n-1} \times 2^{n-1})$ area is necessary to orthogonally draw an n -dimensional TCIN.*

Proof We use the strategy already shown in Section 4.1.

We choose as guest network \mathcal{G} of the embedding ϵ the $(n-1)$ -dimensional CCC network, whose MBW is $\Theta(2^{n-1})$ [91].

Contracting each f -cycle of a n -dimensional TCIN in a new node, we obtain a reduced graph RTCIN that contains a $(n-1)$ -dimensional hypercube network [151]. It is also well known that a reduced $(n-1)$ -dimensional CCC, RCCC, is a $(n-1)$ -dimensional hypercube network [117]. Then, to compute the embedding ϵ , we first define the natural correspondence ϕ from vertices of RCCC to vertices of RTCIN, and from edges of RCCC to edges of the hypercube contained into RTCIN (i.e. all the edges of RTCIN except edges derived from the edges with endpoint 2 in TCIN). Each vertex $rv =$

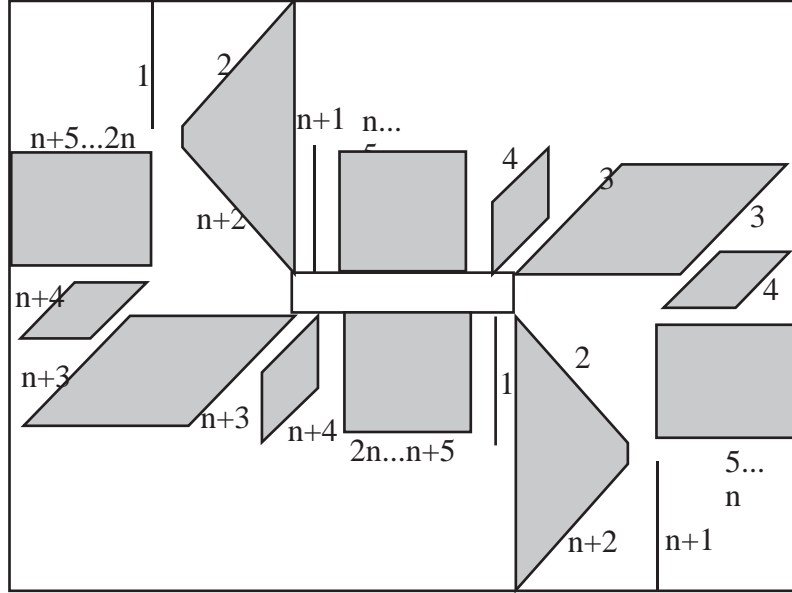


Figure 4.12: General scheme of a TCIN on concentric rectangles.

$\{v_1, v_2, \dots, v_{n-1}\}$ in RCCC and its correspondent $\phi(rv) = \{1, 2, \dots, 2n\}$ in RTCIN are now expanded to define α :

- $\alpha(v_1) = 1$;
- $\alpha(v_i) = i + 1$, for $i = 2, \dots, n - 1$.

For each edge e in CCC we define the following correspondence ρ :

1. if $e = (v_i, v_{i+1})$, i.e. e connects two vertices in the same “reduced” node, then:

$$\rho((v_i, v_{i+1})) = \begin{cases} \text{path}(1, 2, 3) & \text{if } i = 1; \\ (i + 1, i + 2) & \text{for } i = 2, \dots, n - 2; \\ \text{path}(n, n + 1, \dots, 2n, 1) & \text{if } i = n - 1. \end{cases}$$

2. if $e = (v_i, v'_i)$, i.e. e coincides with an edge in RCCC, then:
 $\rho(e) = f(e)$.

It is easy to see that the congestion C of such an embedding is one because vertices in CCC are numbered by using the same method shown in Fact 2 for the TCIN, and it guarantees both that the definition of ρ considers each edge just once and that item 2 is well defined.

Let us consider the reduced networks RCCC and RTCIN obtained from the CCC and the TCIN, respectively (cf. the proof of Theorem 13). It is possible to see that RCCC coincides with a $(n - 1)$ -dimensional hypercube while RTCIN is a graph given by a $(n - 1)$ -dimensional hypercube having each edge duplicated and all its diagonals.

This consideration leads to deduce that these two models of networks are similar and that our three-dimensional representation may be considered also for CCC networks. However, the TCIN is to prefer respect to the CCC network, when a routing problem is considered. In particular, as shown in the following subsection, the one-to-one routing has better solution on the TCIN because the presence of the diagonals in RTCIN halves the diameter and the duplication of the hypercube edges allows to send messages in both the directions, at the same time.

4.3.3 A Shortest Routing Algorithm

In the following we will present a shortest routing algorithm for connecting a source node s with a destination node d . Since \mathcal{G}_n is node symmetric, it is always possible to suitably rename the symbols representing the permutations in order to map the destination node to the identity node i [151]. Then our algorithm finds a shortest route to go from a node $s = a_s^* \dots a_n^* a_1^* \dots a_{s-1}^*$ belonging to a cycle $C = b_{n-2} \dots b_0$ to the node $i = a_1 \dots a_n$ in the cycle $I = 11 \dots 1$ (n ones).

Let us call z the number of zeros in the string $b_{n-2} \dots b_0$. Before describing the two focal points which the algorithm is based on, we need to recall, from Fact 2 that only function g moves a cycle C to a cycle C' . In particular, if a g -edge starts from node 2 (node $n + 2$) of a cycle C , it reaches a node $n + 2$ (node 2) on cycle \bar{C} . All the other g -edges connect cycles whose identification numbers differ in just one bit.

Our routing algorithm is based onto the following two considerations:

1. A path from s to i has to pass along a fixed number of g -edges: either z or $n - z$. If all 0-bits of the string $b_{n-2} \dots b_0$ are complemented into 1, then z g -edges are crossed and $I = 11 \dots 1$ is reached. Otherwise, it is possible first to go to $\bar{I} = 00 \dots 0$, by switching all $n - z - 1$ 1-bits

into 0, and then to run across the slanting g -edge $(2, n + 2)$ to arrive to cycle $I = 11 \dots 1$. In other words, every path from s to i either will “climbs up” from C to I or first will “falls down” to cycle $\bar{I} = 00 \dots 0$ and then, through the g -edge $(2, n + 2)$, will reaches I . To go along different g -edges, some f - (f^{-1} -) edges must be crossed in order to find the endpoints of the gates. If the path directly goes from C to I through z gates, these endpoints are exactly all the nodes starting either with a_{k+1} or with \bar{a}_{k+1} for each $b_{n-k} = 0$ in the string identifying C . Actually, for each $b_{n-k} = 0$ the $(k + 1)$ -th and $(n + k + 1)$ -th nodes are endpoints of a suitable couple of gates. Then we have to choose z gates among $2z$, one for each couple. In order not to overburden the exposition, we do not specify which gate in the couple we choose, because it will appear clear from the context. If the path first goes from C to \bar{I} , the gates are exactly the $2n - 2z$ remaining nodes.

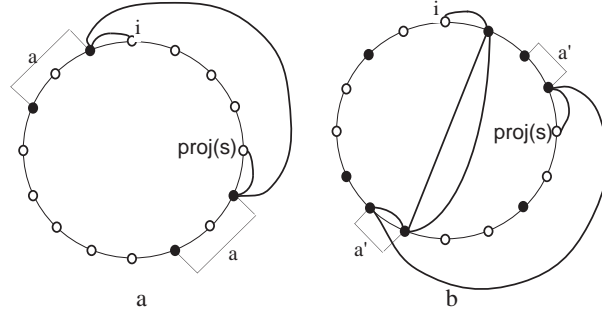
2. Without loss of generality, we choose the plane containing cycle I as projection plane (see Fig. 4.13.a). Then, the problem of finding a route from s to i is reduced to finding a path on cycle I from the projection of s to i , crossing the projections of the gates. The route requested will be found combining this path with the g -edges of the previous item. Actually, if z gates are used, then the minimum path P_1 on I from the projection of s to i , through the projection of z gates, must be found. In the other case, since gate $(2, n + 2)$, projected on cycle I , introduces a bridge from node 2 to node $n + 2$, the minimum path P_2 on I has to pass through $n - z$ projections of gates and one of them is the bridge (see Fig. 4.13.b).

The shortest routing problem on a TCIN is equivalent to finding on I the two paths P_1 and P_2 described in item 2. and to choose the minimum between the routes obtained by adding z g -edges to P_1 and $n - z$ g -edges to P_2 , according to item 1.

Before presenting the shortest routing algorithm, we need to introduce two new sizes: a and a' . Let g_i and g_{i+1} be two consecutive gates and g_{a2}, g_{bs}, g_{as} and $g_{\bar{b}\bar{i}}$ the gates immediately after node 2, before node s , after node s and before node \bar{i} , respectively. Then we define:

on arc $[s, \bar{i}]$: $a = d(v_1, v_2) = \max\{d(s, g_{as}), d(g_{\bar{b}\bar{i}}, \bar{i}), d(g_i, g_{i+1}) \text{ for } g_i, g_{i+1} \in (s, \bar{i})\}$

on arc $[2, s]$: $a' = d(v'_1, v'_2) = \max\{d(2, g_{a2}), d(g_{bs}, s), d(g_i, g_{i+1}) \text{ for } g_i, g_{i+1} \in (2, s)\}$.

Figure 4.13: Paths P_1 and P_2 on cycle I .

In the following algorithm we make the hypothesis that s is on the right side of the bridge edge. When s is on the other side, similar considerations generate the shortest route.

Observe that, while Steps 2-4 work on the structure projected on I , Step 5 comes back to the entire network in order to detail the route whose P , found in Step 4, is the projection.

- Algorithm SHORTEST ROUTE;**
Input: a TCIN $G = (V, E)$; the destination node s ;
Output: a shortest route from i to s ;
begin
1. For a given node s , find cycle $b_{n-2} \dots b_0$ which it belongs to;
 2. generate the set SG of $2z$ projections of gates:
the $(k+1)$ -th and the $(n+k+1)$ -th nodes in I are in SG
iff $b_{n-k} = 0$;
compute $d(s, i)$ and $a = d(v_1, v_2)$;
find on I $P_1 = s \xrightarrow{f} v_1 \xrightarrow{f^{-1}} v_2 \xrightarrow{f} i$;
 $|P_1| = 2n - d(s, i) - 2a$;
 3. generate set \overline{SG} ;
add the bridge on I ;
compute $a' = d(v'_1, v'_2)$;
find on I $P_2 = s \xrightarrow{f^{-1}} v'_1 \xrightarrow{f} v'_2 \xrightarrow{f^{-1}} n+2 \xrightarrow{g} 2 \xrightarrow{f^{-1}} i$;
 $|P_2| = n + d(s, i) - 2a' + 1$;

4. **if** $|P_1| + z \leq |P_2| + n - z - 1$
then $P = P_1$
else $P = P_2$;
5. **shortest route:**
starting from s **on** C , **following** P
repeat
go on the current cycle, along successive
 f -(f^{-1} -)edges, whose projections are on P , until
an endpoint v of a gate is reached;
if it is the last time that the proj. of v is reached by P
then go along the gate to a new cycle;
until i on cycle I is reached;
end.

The following example shows how the algorithm works on a typical input.

Let us consider a 4-dimensional TCIN. We want to compute a shortest route from node $s = \overline{a_3 a_4 a_1 a_2}$ to the identity node $i = a_1 a_2 a_3 a_4$. Node s belongs to cycle $C = 101$ since the node starting with a_1 in C is $a_1 a_2 \overline{a_3} a_4$. SG contains node numbered 4 and its symmetrical node numbered 8 because $b_2 = 0$ and $z = 1$. Now, let us project all gates on cycle I (see Fig. 4.14.a). $d(s, i) = 2$ and $a = \max\{d(s, 4), d(4, \overline{i}), d(4, 4)\} = 1$, with $v_1 = s$ and $v_2 = 4$. Path $P_1 = s \xrightarrow{f^{-1}} 2 \xrightarrow{f^{-1}} i \xrightarrow{f^{-1}} 8 \xrightarrow{f} i$ and $|P_1| = 8 - 2 - 2 = 4$.

It is easy to see that if one would choose $a = d(4, \overline{i})$, then the path $s \xrightarrow{f} 4 \xrightarrow{f^{-1}} s \xrightarrow{f^{-1}} 2 \xrightarrow{f^{-1}} i$ were found, that has the same length of the previous P_1 .

\overline{SG} contains nodes numbered 1, 2, 3, 5, 6 and 7, and the bridge $\{2, 6\}$ is considered on I (see Fig. 4.14.b). $a' = \max\{d(2, s), d(2, s)\} = 1$; then $v'_1 = s$ and $v'_2 = 2$. Path $P_2 = s \xrightarrow{f} 4 \xrightarrow{f} 5 \xrightarrow{f} 6 \xrightarrow{g} 2 \xrightarrow{f^{-1}} i$ and $|P_2| = 4 + 2 - 2 + 1 = 5$.

We choose $P = P_1$ because $|P_1| + z < |P_2| + n - z - 1$. Then, the shortest route is given:

$$s = \overline{a_3 a_4 a_1 a_2} \xrightarrow{f^{-1}} a_2 \overline{a_3} a_4 \overline{a_1} \xrightarrow{f^{-1}} a_1 a_2 \overline{a_3} a_4 \xrightarrow{f^{-1}} \overline{a_4} a_1 a_2 \overline{a_3} \xrightarrow{g} \overline{a_4} a_1 a_2 a_3 \xrightarrow{f} a_1 a_2 a_3 a_4 = i.$$

Now consider $s = \overline{a_1} a_2 a_3 \overline{a_4}$, belonging to cycle $C = 001$, $z = 2$. In Fig. 4.15 cycle I is represented together with SG (Fig. 4.15.a) and \overline{SG} (Fig. 4.15.b).

$$P_1 = s \xrightarrow{f} 6 \xrightarrow{f} 7 \xrightarrow{f} 8 \xrightarrow{f} i \text{ and } |P_1| = 8 - 4 = 4.$$

$$P_2 = s \xrightarrow{f} 6 \xrightarrow{g} 2 \xrightarrow{f^{-1}} i \text{ and } |P_2| = 4 + 4 - 6 + 1 = 3.$$

Since $|P_1| + z > |P_2| + n - z - 1$, P_2 is chosen, and the shortest route is:

$$s = \overline{a_1}a_2a_3\overline{a_4} \xrightarrow{f} a_2a_3\overline{a_4}a_1 \xrightarrow{g} a_2a_3\overline{a_4}\overline{a_1} \xrightarrow{f^{-1}} a_1a_2a_3\overline{a_4} \xrightarrow{g} a_1a_2a_3a_4 = i.$$

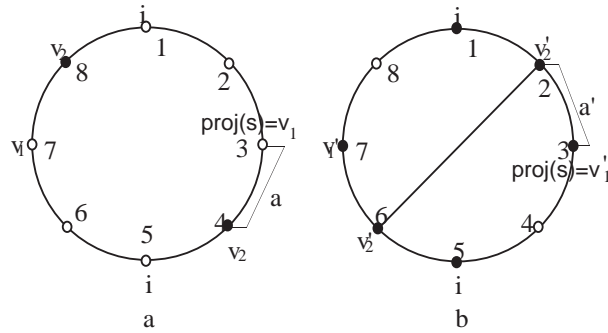


Figure 4.14: Cycle I related to the first part of the Example.

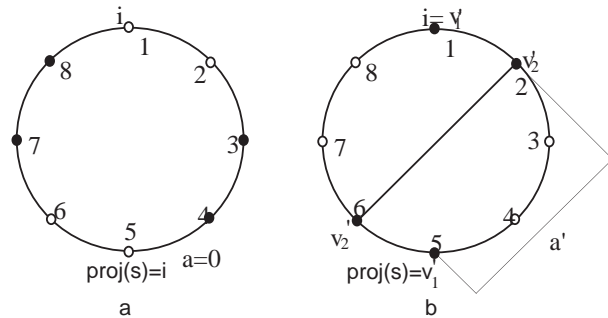


Figure 4.15: Cycle I related to the second part of the Example.

Theorem 14 *The algorithm SHORTEST ROUTE correctly computes in $O(n)$ time the shortest route from an arbitrary node s to the identity node i in an n -dimensional TCIN.*

Proof For what concerns the correctness, the path computed in Step 5 is a route because:

- it is a path going through consecutive edges defined by f, f^{-1}, g ;

- it goes from s to i because g -edges lead from cycle C of s to the identity cycle I , and f - (f^{-1} -)edges allow the path to move along the cycles to find gates.

This route is minimum because:

- it is possible to prove in an exhaustive way that P_1 and P_2 (Fig. 4.13) cover all the possible paths from the projection of s to i crossing all projections of the gates.
- the projection of all gates on I allow to transfer all the paths between consecutive gates on the same cycle and then to have the possibility to minimize the global walk;
- it is not possible to reduce the number of g -edges because each of them switches just one bit on the binary string of the current cycle. Moreover the algorithm computes both the paths switching 0's in 1's and the path switching 1's in 0's.

For what concerns the complexity, it is always possible to compute in $O(n)$ time the number of the cycle to which s belongs, that is the binary string associated either to $\overline{a_1^* \dots a_{s-1}^* a_s^* \dots a_n^*}$ or to its complement.

All the operations in Steps 2 and 3 are executed in $O(n)$ time, except the computation of $d(s, i)$, the addition of the bridge and the computation of $|P_1|$ and $|P_2|$, that run in constant time; so is Step 4.

Step 5 runs in time proportional to the length of the found route, that is not greater than $2n$. Indeed, it is easy to see that $3n/2$ is an upper bound for the length of P in view of the computation of the paths P_1 and P_2 and of the constants a and a' . Furthermore, no more than $n/2$ g -edges are necessary to jump through the cycles because if the number of 0-bits is greater than the number of 1-bits, the bridge edge is introduced at the beginning to reach \overline{C} .

Chapter 5

Approximation

5.1 Introduction

It is widely known that \mathcal{NP} -complete problems are such a vast variety of commonly encountered problems for so many fields that their intractability must somehow be overcome by computing at least an approximate solution in polynomial time. Just these practical reasons have raised the interest and increased the importance of approximation theory.

Among combinatorial optimization problems that are computationally hard to solve, \mathcal{NP} -complete optimization problems on graphs have a great relevance both from theoretical and practical point of view.

Therefore, in the literature, a large amount of papers describing algorithms approximating problems and theoretical results have been written. In [38] a large number of these results are collected together.

As we have already observed in Chapters 1 and 2, despite the apparent simplicity of cubic and at most cubic graphs, many graph problems are no easier to solve when restricted to them. The problem of finding a largest bipartite subgraph contained in a given graph is only an example: it is \mathcal{NP} -complete in general and it remains \mathcal{NP} -complete even if the considered graph is triangle-free and has maximum degree 3 [162]. More generally, most graph problems, whose decision version is \mathcal{NP} -complete, remain \mathcal{NP} -complete even for this class of graphs, but they become solvable in polynomial time for graphs of degree two [58, 62]. Therefore, it would be desirable to understand if cubic graphs are a boundary class of graphs, i.e. if they really are –as they seem to be with regard to numerous problems– the ‘smallest’ class of graphs for which problems are as difficult as in the general

case. More generally, it is still not clear if and how much boundedness of the graph's degree is helpful in approximation.

In the following we deal with some problems that exploit the cubic property. Namely, we will show examples of some problems that are \mathcal{NP} -complete in general but that are polynomially solvable for cubic graphs; some problems that are approximable for general graphs but for which better approximation ratios have been achieved for graphs of low degree; some problems that for general graphs cannot be approximated within any constant approximation ratio but that have been shown to be in \mathcal{APX} (i.e. approximable within some constant) for bounded degree graphs. Our attempt is not to draw up an exhaustive list, but only to show how cubicity –and more in general, boundedness of the degree– may help in the approximation of \mathcal{NPO} problems.

Problems that are \mathcal{NP} -complete in general but that are polynomially solved for cubic graphs.

A triangle packing for a graph is a collection of disjoint subsets of the set of vertices, each containing exactly three vertices and inducing a triangle in the graph. The *Maximum Triangle Packing Problem* consists of finding a triangle packing of maximum cardinality. This problem is \mathcal{APX} -complete in general [83] and it remains \mathcal{APX} -complete also if the degree of the graph is bounded by a constant d . But if the input graph is cubic, then the problem is polynomially solvable since only two cases arise: either the triangles are disjoint or two of them share one edge (diamond). Therefore, it is easy to compute the optimal solution for at most cubic graphs.

The *Maximum Clique Problem* in a graph consists in finding a clique of maximum cardinality. In general, this problem is not approximable within any constant factor [12], but for cubic graphs the problem is trivially solvable since the maximum clique one can find in a cubic graph (different from K_4) is a triangle.

Problems that are approximable for general graphs but for which better approximation ratios have been achieved for cubic graphs.

Given a graph \mathcal{G} , the *Minimum Vertex Cover Problem* consists in finding a minimum cardinality set of vertices V' such that, for any edge at least one of its extremes is in V' . It is obvious that any maximal matching defines a vertex cover, which is at most twice as large as the optimal one. In [105] it is proved that this problem is approximable within $2 - \frac{\log \log |V|}{2 \log |V|}$. Some improvements can be made for graphs with degree bounds: $2 - \frac{5}{d+3} + \epsilon$ is the approximation ratio achieved by the algorithm in [15] for odd d , improving

the result of [104] achieving $5/4$ for at most cubic graphs and slightly worst ratios for the other bounded degree graphs.

Problems that cannot be approximated within any constant approximation ratio for general graphs but that have been shown to be in \mathcal{APX} for bounded degree graphs.

An independent set in a graph is a set of vertices in which no two of them are adjacent, and in the *Maximum Independent Set Problem* such a vertex set of maximum cardinality is sought. The general maximum independent set problem is notorious for its intractability. The problem for bounded degree graphs is still \mathcal{NP} -complete even when instance graphs are restricted to be cubic and planar [59], but is approximable within a constant factor [16] that is arbitrarily close to $6/5$ for at most cubic graphs and to $7/6$ for cubic graphs [15].

A dominating set is a set of vertices such that, if a vertex of the graph is not in this set, then one of its adjacent vertices is. The *Minimum Dominating Set Problem* is not in \mathcal{APX} [11, 98] and is approximable within $1 + \log n$ [82]. Variation in which the degree of the graph is bounded by a constant d is \mathcal{APX} -complete [108] and is approximable within $\sum_{i=1}^{d+1} \frac{1}{i} - 0.433$.

In the next section, a problem relating to the latter group is taken into consideration and the previously known results are improved for cubic, at most cubic and, in general, bounded degree graphs.

5.2 Approximation of Independent Dominating Set in Bounded Degree Graphs

5.2.1 Introduction

An independent dominating set in a graph is a collection of vertices such that it is adjacent to all other vertices, and vertices in the collection are mutually non-adjacent. In the *Minimum Independent Dominating Set Problem* a set of minimum cardinality is required. The problem of finding an independent dominating set of minimum cardinality is \mathcal{NP} -hard, even if we restrict ourselves to graphs of degree bounded by a constant $d \geq 3$. Furthermore, the problem for general graphs cannot be approximated within any constant approximation ratio [67], while it has been shown to be \mathcal{APX} -complete and approximable within $d + 1$ [84] for bounded degree graphs.

In this section we give approximate heuristics for MIDS in cubic and at most cubic graphs, based on greedy and local search techniques.

Our algorithms achieve approximation ratios:

- 1.923 for cubic graphs;
- 2 for at most cubic and 4-regular graphs;
- $\frac{(d^2-2d+2)(d+1)}{d^2+1}$ for d -regular graphs, $d \geq 5$;
- $\frac{(d^2-d+1)(d+1)}{d^2+1}$ for graphs of bounded degree $d \geq 4$;

improving the previously known ratios of $d + 1$ [84], as shown in Table 5.1.

	$d = 3$	$d = 4$	$d = 5$	$d = 6$	$d = 7$	$d = 8$	$d = 9$
previous results	4	5	6	7	8	9	10
bounded degree graphs	2	3.824	4.846	5.865	6.880	7.892	8.902
regular graphs	1.923	2	3.923	4.919	5.920	6.923	7.927

Table 5.1: Table of results summarising previous and our results.

Throughout this section we consider only finite, simple, loopless and possibly disconnected graphs. However, we do not allow isolated vertices. Of course, since isolated vertices belong to any independent dominating set, the previous assumption is not restrictive.

Before describing the two algorithms in detail, we give some formal definitions and preliminary results that will be useful in the next subsections.

Definition 10 *Given a graph $\mathcal{G} = (V, E)$ the Minimum Independent Dominating Set Problem (MIDS) is the problem of finding the smallest possible set $S^* \subseteq V$ of vertices such that for all $u \in V - S^*$ there is a $v \in S^*$ for which $\{u, v\} \in E$, and such that no two vertices in S^* are joined by an edge in E . Variation in which the degree of \mathcal{G} is bounded by a constant d is denoted by MIDS- d .*

Given a graph $\mathcal{G} = (V, E)$, we denote by S^* a MIDS and by S the solution determined by our algorithms.

Lemma 16 *If \mathcal{G} is a graph of bounded degree d , then $|S^*| \geq \frac{n}{d+1}$.*

Proof The claim follows from the fact that S^* is a dominating set, and each vertex $v \in S^*$ can dominate at most d vertices.

Fact 3 Given a connected graph $\mathcal{G} = (V, E)$ of bounded degree 2, $|S^*| \leq |V|/2$. Suppose now that k vertices are forbidden to be in the MIDS. The optimal value of such a constrained solution (if it exists) remains no greater than $|V|/2$, but the k vertices are the even vertices of an odd length chain (see Fig. 5.1). From now on we will denote by *peaks* such even vertices.

Figure 5.1: An example of optimal constrained solution of cardinality greater than $|V|/2$

Let $\mathcal{G} = (V, E)$ be a graph with bounded degree d . We denote by $adj(v) = \{u \in V | \{u, v\} \in E\}$ and by $adj^2(v) = \bigcup_{u \in adj(v)} adj(u) - \{v\} - adj(v)$.

The proposed algorithms use an auxiliary graph $\mathcal{G}' = (V', E')$, which at the beginning is equal to \mathcal{G} . Let V'_k be the set of vertices of degree $k \leq d$ in \mathcal{G}' .

The rest of this chapter is devoted to the description of the algorithms, that are both composed of two phases. In the former one, the algorithms greedily select vertices of degree three and remove them and all their adjacent vertices until the graph becomes of degree two. In the latter one, a sort of local search phase is performed to complete the solution. When the graph is cubic a preprocessing phase is also executed in order to improve the value of the solution.

Then, we consider graphs of bounded degree d and d -regular graphs, for $d \geq 4$. In particular, the greedy phase proposed for at most cubic graphs is extended to any d and iteratively repeated until the degree of the remaining graph is greater than three. Finally, the algorithm for at most cubic graphs is executed.

5.2.2 MIDS in at Most Cubic Graphs

In the following we first give an approximate algorithm for MIDS-3 that we call *CubMids* from now on. We then prove that the proposed heuristic

approximates MIDS for at most cubic graphs within two. Finally, we show that a slight variation of *CubMids* allows to achieve a guaranteed performance ratio of 1.923 for cubic graphs.

To make the exposition clearer, we first sketch the basic strategy and then analyze the algorithms' performance.

The overall strategy we use to solve MIDS-3 is the following:

- *while-loop* (lines 4-11)

In this phase vertices of degree three in \mathcal{G}' are sequentially considered. Given a vertex $v \in V'_3$, it is put in the independent dominating set S . Then v and its adjacent vertices are removed from \mathcal{G}' . Indeed, every vertex $u \in \text{adj}(v)$ is dominated by v , and adjacent vertices cannot belong to an independent set.

At the end of this phase, $\mathcal{G}' = (V', E')$ has bounded degree two. After the execution of the while-loop, an optimal MIDS could be found in \mathcal{G}' . Since all isolated vertices of any graph must be put in the MIDS, it would be desirable to reduce their number as much as possible, even if, at the same time, a small disadvantage is introduced.

- *swap-step* (lines 12-31)

The aim of this step is to benefit by a sort of local search to slow the number of isolated vertices of \mathcal{G}' down, providing that the drawback is not too large. We define the “neighborhood structure” of any vertex $v \in S$ as the set of independent dominating sets of the subgraph of \mathcal{G} induced by $\text{adj}(v)$.

Algorithm *CubMids*

Input: $\mathcal{G} = (V, E)$.

Output: S .

1. **begin**
2. $\mathcal{G}' = (V', E') \leftarrow \mathcal{G} = (V, E)$;
3. $S \leftarrow \emptyset$;
4. **while** $(V'_3 \neq \emptyset)$ **do**
5. **choose** $v \in V'_3$;
6. $S \leftarrow S \cup \{v\}$;
7. $V' \leftarrow V' - \{v\} - \text{adj}(v)$;
8. $I_v \leftarrow \emptyset$;
9. $T_v \leftarrow \emptyset$;

```

10.     for each  $k$  do update  $V'_k$ ;
11.   endwhile
12.    $\mathcal{C} \leftarrow \{C \text{ such that } C \text{ is a connected component of } \mathcal{G}'\}$ ;
13.   for each  $v \in S$  do
14.     for each neighbor  $\overline{N}_v$  of  $v$  do
15.       for each  $C \in \mathcal{C}$  such that  $\exists w \in C \cap \text{adj}^2(v)$  do
16.         case of  $C$ 
17.            $T$ -component of  $v$  w.r.t.  $\overline{N}_v$ :  $\overline{T}_v \leftarrow \overline{T}_v \cup C$ ;
18.           isolated vertex  $w$ :  $I_v \leftarrow I_v \cup \{w\}$ ;
19.          $\mathcal{C} \leftarrow \mathcal{C} - C$ ;
20.       endfor
21.        $N_v \leftarrow \overline{N}_v$  such that  $p(\overline{N}_v)$  is max;
22.        $T_v \leftarrow \overline{T}_v$ ;
23.     endfor
24.     if  $(p(N_v) > 0)$  then
25.        $S \leftarrow S \cup N_v - \{v\}$ ;
26.        $V' \leftarrow V' - \text{adj}(N_v)$ ;
27.       for each  $k$  do
28.         update  $V'_k$ ;
29.       update  $\mathcal{C}$ ;
30.     endif
31.   endfor
32.    $S \leftarrow S \cup \text{optimal MIDS for } \mathcal{G}'$ ;
33. end.

```

Then, for each neighbor of v , we compute a profit function p . We say N_v a neighbor corresponding to the maximum $p(N_v)$. Moreover we indicate by $\text{adj}(N_v) = \bigcup_{u \in N_v} \text{adj}(u) - \{v\}$ the set of vertices adjacent in \mathcal{G} to vertices in N_v but v (see Fig. 5.2).

Each vertex v put in S in the while-loop is now processed. If $p(N_v) \leq 0$, v is left in S , otherwise the pair (v, N_v) is swapped, that is v is removed from S , all vertices in N_v are put in S , and all vertices in $\text{adj}(N_v)$ are removed from the current graph \mathcal{G}' .

In particular, among the vertices removed from the graph there will be some isolated vertices and there could be some non-isolated vertices. Function $p(N_v)$ takes into account both the benefit of the deletion of isolated vertices (stored in I_v) and of the disadvantage possibly introduced by removing non-isolated vertices (T -components stored in T_v).

- *final-step* (line 32)

Finally, an optimal MIDS for the remaining graph \mathcal{G}' is found and added to S .

Figure 5.2: An example of N_v , I_v and $adj(N_v)$.

Now we show that the proposed heuristic finds a feasible solution of MIDS-3, and then prove that it approximates the problem within two. Finally, we focus on cubic graphs and show that a better guaranteed approximation ratio is achieved with a slight variation in the while-loop.

Theorem 15 *Given an at most cubic graph \mathcal{G} , Algorithm CubMids finds an independent dominating set.*

Proof In order to show that S is a dominating set, observe that, during the execution of the while-loop, each vertex v put in S dominates vertices that are removed with it, i.e. it dominates all vertices in $\{v\} \cup adj(v)$. Then, in the swap-step, every time a pair (v, N_v) is swapped, the set N_v dominates all vertices in $\{v\} \cup adj(v)$, and all vertices in $adj(N_v)$ are removed from \mathcal{G}' . Finally, an optimal MIDS is determined in the remaining graph.

To prove that S is an independent set, first consider that the while-loop finds an independent set: when a vertex v is put in S , its adjacent vertices are deleted from \mathcal{G}' . Then, in the swap-step, every time a pair (v, N_v) is swapped, all vertices in $adj(N_v)$ are removed from \mathcal{G}' . Then, at each iteration of the swap step vertices in the current \mathcal{G}' cannot be adjacent to any vertex in S (line 26 of *CubMids*). Finally, an optimal MIDS is determined in the remaining graph.

Before proving the performance ratio of the *CubMids*, we discuss the definition of the profit function leading the swap-step.

Let (v, N_v) be a pair where v is a vertex in S after the while-loop and N_v an independent dominating set in the subgraph induced in \mathcal{G} by $adj(v)$ such that the corresponding $p(N_v)$ is maximum. In order to figure out how the profit function is defined, we will show in which way p evaluates whether (v, N_v) must be swapped or not. Notice that, as far as N_v is defined, if the pair (v, N_v) is not swapped, then no other pair (v, N'_v) is.

Let $w \in adj(N_v) \cap V'$ be a vertex in a connected component of \mathcal{G}' , then one of the following facts holds:

- a. $w \in V'_0 \cap adj(N_v)$

Let v be the general vertex put in S in the while-loop. In the com-

putation of the profit function $p(N_v)$, only isolated vertices in $adj(N_v)$ must be considered, say I_v . It is easy to see that, as far as *CubMids* is concerned, the sets I_v are mutually disjoint and that for each v there is up to one vertex in $V'_0 \cap adj^2(v)$ which does not belong to I_v .

Suppose (v, N_v) is not swapped, then w would be put in S in the final-step. Otherwise, if (v, N_v) is swapped, w is deleted from G' . It follows that it is suitable to swap (v, N_v) if the number of isolated vertices in $adj(N_v)$ overcomes the increase of $|S|$ due to the swap (i.e. $|N_v| - 1$) and the possible negative contribution produced by the change of \mathcal{G}' (see item c).

- b. $w \in V'_1 \cap adj(N_v)$

Consider the connected component C containing w in \mathcal{G}' . Since \mathcal{G}' has bounded degree two and w has degree one, then C must be a chain, and w is one of its extremes. Hence the cardinality of the optimal independent dominating set in the chain cannot increase by swapping (v, N_v) , that corresponds to the deletion of w . Then, no contribution is given by such a vertex in the computation of p .

- c. $w \in V'_2 \cap adj(N_v)$

Consider the connected component C containing w in \mathcal{G}' . As \mathcal{G}' has bounded degree two, and w has degree two, then C must be either a cycle or a chain and w is an internal vertex.

Notice that, since we are interested in computing an independent dominating set of cardinality no greater than $n/2$, from Fact 3, we can restrict to consider only the case in which w is a peak of an odd chain. Indeed, only the deletion of all peaks in an odd chain leads the value of the solution to overcome the bound of half of the vertices. Therefore, we define *T-component* of v with respect to N_v either a chain of length 3 such that its peak is w , or a chain of length five such that both its peaks belong to $adj(N_v)$, or a chain of length seven such that its three peaks belong to $adj(N_v)$. Because of the cubicity of the graph, no odd chains of length greater than seven must be considered. From now on, and when no confusion arises, we will simply call T_v the set of all *T-components* of v with respect to N_v .

It remains to analyze the case in which the peaks of an odd chain belong to different $adj(N_{v_j})$, where v_j is a vertex put in S during the while-loop. Such an odd chain becomes a *T-component* in T_{v_j} if and only if the pairs (v_i, N_{v_i}) swap, $i \neq j$ (see Fig. 5.3). This is the rea-

son why the assignment of T -components to vertices put in S in the while-loop must be done when they are processed in the swap-step.

Figure 5.3: “Shared” T -components.

As far as *CubMids* is concerned, after the execution of the swap-step, graph \mathcal{G} is implicitly partitioned into the following disjoint subgraphs:

- for each vertex v put in S during the while-loop, subgraph \mathcal{G}_v induced in \mathcal{G} by $v \cup N_v \cup I_v \cup T_v$.
- the possibly not connected subgraph \mathcal{G}_R induced in \mathcal{G} by the remaining vertices.

The previous remarks make us able to define the profit function p , that leads the execution of the swap-step, $p(N_v) = |I_v| - |N_v| + 1 - |T_v|$.

We will prove that $|S| \leq n/2$ by showing that, for each subgraph, up to half of their vertices belong to S found by the algorithm.

Lemma 17 *Let v be a vertex put in S in the while-loop and n_v be the number of vertices in \mathcal{G}_v . Then, at most $n_v/2$ vertices belong to S .*

Proof Notice that if (v, N_v) is not swapped, then v , all vertices in I_v and all peaks of T -components in T_v are put in S . Otherwise, if (v, N_v) is swapped, then all vertices in N_v and all non-peaks of T -components in T_v are put in S .

Since the previous sets are disjoint, at least one of them has cardinality no greater than $n_v/2$.

Function $p(N_v) = |I_v| - |N_v| + 1 - |T_v|$ leads the choice.

Lemma 18 *Let n_R be the number of vertices in \mathcal{G}_R . Then, at most $n_R/2$ vertices belong to S .*

Proof After the swap-step, vertices in \mathcal{G}_R are partitioned into the following way:

- n_d vertices adjacent to N_v for some swapped pair (v, N_v) , and thus already dominated;
- n_i vertices made isolated by the swap-step;
- $n_R - n_i - n_d$ vertices of bounded degree 2.

In view of Remark 3, up to $\frac{n_R - n_i - n_d}{2} + n_i = \frac{n_R + n_i - n_d}{2}$ vertices must be put in S .

The only case in which one dominated vertex leaves more than one isolated vertex is due to T -components. Since T -components have already been considered in subgraphs \mathcal{G}_v , then $n_i \leq n_d$ holds. The result follows.

Theorem 16 *Algorithm CubMids approximates MIDS-3 within a factor of two.*

Proof In view of Lemmas 16, 17 and 18, we obtain

$$\frac{|S|}{|S^*|} \leq \frac{n/2}{n/4} = 2$$

Consider a cubic graph $\mathcal{G} = (V, E)$ and call i_w the cardinality of S after the execution of the while-loop. Every time a vertex is put in S in the while-loop, exactly four vertices and at most six edges incident to the remaining vertices are deleted from the graph (see Fig. 5.4). Then \mathcal{G}' contains no more than $2i_w$ isolated vertices, $|V'_0| \leq 2i_w$.

Figure 5.4: Extremal cases in the while-loop.

As a consequence of such a bound for $|V'_0|$, the swap-step is not crucial anymore to achieve guaranteed approximation ratio 2. Indeed, if we run the final-step right after the while-loop the cardinality of the solution is $|S| \leq i_w + |V'_0| + (n - 4i_w - |V'_0|)/2 \leq n/2$.

Finally, we will show that for cubic graphs the performance ratio achieved by *CubMids* can be improved by running a preprocessing phase. Informally speaking, it would be desirable to put into S several vertices adjacent to three vertices not dominated yet. The preprocessing phase exploits cubicity

of the graph choosing vertices that can be put in the solution without leaving isolated vertices. Then *CubMids* is run on the remaining possibly not connected at most cubic graph.

Let $\mathcal{G} = (V, E)$ be a cubic graph. In the preprocessing phase vertices of degree 3 are selected and put in the solution S , similarly to the while-loop of *CubMids*, but a further condition must hold in order to guarantee that no isolated vertices are left in the remaining graph \mathcal{G}' .

Namely, vertices put in S must have distance at least five. Because of the cubicity of the graph no isolated vertices are produced but in the case drawn in Fig. 5.5, and then no isolated vertices are left in the remaining graph by assuming that in this case the produced isolated vertex is also put in S .

Figure 5.5: Extremal cases in the preprocessing phase run before *CubMids* for cubic graphs.

Theorem 17 *Algorithm CubMids with the preprocessing phase approximates MIDS for cubic graphs within 25/13.*

Proof As far as the preprocessing phase is concerned, vertices are put in S in the following way:

- either one vertex is put in S and up to 45 vertices are forbidden; (Fig. 5.5.a);
- or two vertices are put in S and up to 24 are forbidden (Fig. 5.5.b).

Let n_1 be the number of vertices following 1. and n_2 be the number of pair of vertices following 2. Since $46n_1 + 26n_2 \leq n$ after the execution of *CubMids* we have:

$$|S| \leq n_1 + n_2 + \frac{n - 4n_1 - 5n_2}{2} \leq \frac{25}{52}n.$$

Therefore, in view of Lemma 16

$$\frac{|S|}{|S^*|} \leq 1.923.$$

5.2.3 MIDS in Bounded Degree Graphs

In this section we generalize the above results to any bounded degree graphs ($d \geq 4$), and give an heuristic –we will call *BounDegMIDS*– that approximates MIDS- d within

- $K_d(d+1)$ for bounded degree graphs, $d \geq 4$
- 2 for 4-regular graphs
- $(K_d - \frac{d-2}{d^2+1})(d+1)$ for d -regular graphs, $d \geq 5$

where $K_d = \frac{(d^2-d+1)}{d^2+1}$.

BounDegMIDS works as follows. Let $\mathcal{G} = (V, E)$ be a graph with bounded degree d . First, vertices of degree d are processed. Then, the degree of the remaining graph \mathcal{G}' is at most $d-1$. If $d-1 > 3$ then the same algorithm is executed on \mathcal{G}' ; if $d \leq 3$ then *CubMIDS* is executed on \mathcal{G}' .

Algorithm *BounDegMIDS*

Input: $\mathcal{G} = (V, E)$, d .

Output: S .

1. **begin**
2. $\mathcal{G}' = (V', E') \leftarrow \mathcal{G} = (V, E)$;
3. $S \leftarrow \emptyset$;
4. **while** ($V'_d \neq \emptyset$) **do**
5. **begin**
6. choose $v \in V'_d$;
7. $S \leftarrow S \cup \{v\}$;
8. $V' \leftarrow V' - \{v\} - \text{adj}(v)$;
9. **for each** k **do** update V'_k
10. **end**;
11. $S \leftarrow S \cup V'_0$;
12. $V' \leftarrow V' - V'_0$;
13. $S \leftarrow S \cup$ approximate MIDS for \mathcal{G}' ;
14. **end**.

Theorem 18 *Given a bounded degree graph, Algorithm BounDegMIDS finds an independent dominating set.*

Proof Similar to the proof of Theorem 15 and therefore we cut it.

Lemma 19 *Consider just one iteration of BounDegMIDS (lines 1-12) on a graph of bounded degree d . Call \mathcal{G}' the remaining graph after the iteration.*

Let i_w be the number of vertices put in S and $|V'_0|$ be the number of vertices made isolated during the while-loop of this iteration.

The following inequalities hold:

- a. $i_w \geq \frac{|V'_0|}{d(d-1)}$
- b. $i_w \leq \frac{n-|V'_0|}{d+1}$
- c. $|V'_0| \leq \frac{d(d-1)n}{(d^2+1)}$.

Proof During the execution of the while-loop (line 4-10), every time a vertex is put in S , exactly $d+1$ vertices and at most $d(d-1)$ edges incident to the remaining vertices are deleted from the graph. Therefore:

- a. the graph contains no more than $d(d-1)i_w$ isolated vertices.
- b. the graph has at least $|V'_0|$ vertices, and thus the number of vertices removed from the graph is $i_w(d+1) \leq n - |V'_0|$
- c. From a. and b. the lemma follows.

Theorem 19 Algorithm *BounDegMIDS* approximates *MIDS-d* within $K_d(d+1)$, for $d \geq 4$, where $K_d = \frac{(d^2-d+1)}{d^2+1}$.

Proof By induction on the maximum degree d , we will show that, for each $d \geq 4$, *BounDegMIDS* determines S such that

$$|S| \leq K_d n = \left(1 - \frac{d}{d^2+1}\right) n.$$

Then, from Lemma 16,

$$\frac{|S|}{|S^*|} \leq \frac{(d^2-d+1)(d+1)}{d^2+1}$$

which will conclude the proof.

Basis: $d = 4$. *BounDegMIDS* finds a MIDS S of cardinality

$$|S| \leq i_w + |V'_0| + \frac{1}{2}(n - 5i_w - |V'_0|)$$

Indeed, after the while-loop, the remaining at most cubic graph \mathcal{G}' has $(n - 5i_w - |V'_0|)$ vertices. Then, Algorithm *CubMIDS* is executed, and at most half of such vertices are put in S .

Since $d = 4$ and in view of Lemma 19.a and c, the basis of the induction is proved.

Inductive Step: Consider now $d > 4$; after each iteration, *BounDegMIDS* is executed on \mathcal{G}' , then the inductive hypothesis can be used:

$$|S| \leq i_w + |V'_0| + K_{d-1}(n - (d+1)i_w - |V'_0|)$$

Since the coefficient of i_w is negative, from Lemma 19.a we have:

$$\begin{aligned} |S| &\leq nK_{d-1} + \frac{|V'_0|}{d(d-1)}(1 - dK_{d-1} - K_{d-1}) + |V'_0|(1 - K_{d-1}) = \\ &= nK_{d-1} + |V'_0| \frac{d^2 - K_{d-1}d^2 - d + 1 - K_{d-1}}{d(d-1)}. \end{aligned}$$

From Lemma 19.c

$$|S| \leq \left(1 - \frac{d}{d^2+1}\right)n.$$

Exploiting specific properties of regular graphs, a better performance ratio can be achieved if the input graph is d -regular.

Lemma 20 *Consider just one iteration of BounDegMIDS (lines 1-12) on a d -regular graph. Let us call \mathcal{G}' the remaining graph after the iteration, i_w the number of vertices put in S and $|V'_0|$ the number of vertices made isolated during the while-loop of this iteration.*

The following inequalities hold:

- a. $i_w \geq \frac{n}{d^2+1}$
- b. $|V'_0| \leq (d-1)i_w$

Proof During the execution of the while-loop, every time a vertex is put in S :

- a. up to $d^2 + 1$ vertices of degree d become of lower degree (see Fig. 5.4);
- b. exactly $d+1$ vertices and at most $d(d-1)$ edges incident the remaining vertices are deleted from the graph. Then \mathcal{G}' contains no more than $(d-1)i_w$ isolated vertices.

Theorem 20 *Algorithm BounDegMIDS approximates MIDS for 4-regular graphs within 2 and for d -regular graphs within $(K_d - \frac{d-2}{d^2+1})(d+1)$, for $d \geq 5$, where $K_d = \frac{(d^2-d+1)}{d^2+1}$.*

Proof We first prove that the solution S found by *BounDegMIDS* for d -regular graphs ($d \geq 5$) is:

$$|S| \leq \left(1 - \frac{2d-1}{d^2+1}\right)n.$$

Since after the while-loop *BounDegMIDS* is run on \mathcal{G}' of bounded degree $d-1$, we use the result of Theorem 19:

$$|S| \leq i_w + |V'_0| + K_{d-1}(n - (d+1)i_w - |V'_0|).$$

From Lemma 20.b.

$$|S| \leq nK_{d-1} + i_w(1 - dK_{d-1} - K_{d-1}) + (1 - K_{d-1})(d-1)i_w = nK_{d-1} + i_w(d - 2dK_{d-1})$$

Therefore, in view of Lemma 20.a and of the definition of K_d , we see

$$|S| \leq \frac{n}{d^2+1} (K_{d-1}(d-1)^2 + d) \leq \left(1 - \frac{2d-1}{d^2+1}\right)n.$$

Similarly to the proof of Theorem 19, the theorem follows.

The proof for 4-regular graphs is analogous to the previous one by assuming $K_3 = 1/2$, and therefore is omitted.

The algorithm proposed for the minimum independent dominating set problem in cubic and at most cubic graphs (MIDS-3) is made up by two phases. In the former greedy phase a partial solution is constructed by sequentially selecting vertices of degree three and removing them and their adjacent vertices from the graph. Then a local search phase (swap-step) is performed in order to obtain a better complete solution.

For the minimum independent dominating set problem in any bounded degree and regular graphs (MIDS- d) the proposed heuristic iteratively works like the greedy phase in *CubMIDS*. Then, when the degree of the graph is bounded by three, *CubMIDS* is applied.

Unfortunately it is not easy to extend the local search phase to graphs of bounded degree higher than three. Indeed, when the degree of the graph increases (even for d equal to four), connected components giving a negative contribution to each swap can be more complex and shared among vertices put in the solution in the first step. This fact leads to the lower improvement in the quality of the solution achieved by our heuristics for graphs of degree higher than three.

Besides, it is worth to notice that for cubic and at most cubic graphs we have found independent dominating sets of size at most $0.48n$ and $0.5n$, respectively. Finding independent dominating sets of size less or equal than half of the vertices could be not possible for any bounded degree graph. Indeed, for degree higher than three there exist infinite instances for which the cardinality of optimal solutions is greater than half of the number of vertices. Such graphs can be obtained hanging two degree one adjacent vertices up to each vertex of an odd length cycle (e.g. see Fig. 5.6).

Figure 5.6: Optimal MIDS greater than half of the vertices for a class of graphs of bounded degree four.

Glossary

Ackermann function (inverse):

‘The inverse Ackermann function arises in many applications in logic, combinatorics and computer science. It approaches infinity as n grows, but does this extremely slowly; for example, it does not exceed 5 for n up to an exponential tower $2^{2^{\dots}}$ having 65536 2s. See [71] for more details concerning this function.’ [153]

\mathcal{AC}^0 :

‘The class \mathcal{AC}^0 consists of all decision problems solvable by constant depth, polynomial size, unbounded fan-in circuits.’ [153]

Cayley graph:

‘Let $p = p_1 p_2 \dots p_n$ be a permutation of $\{1, 2, \dots, n\}$. Let S_n denote the set of all permutations over $\{1, 2, \dots, n\}$. Let $p, q \in S_n$. Define an associative binary operation ‘ \cdot ’ (called product) as $(p \cdot q)_x = p(q_x)$, that is, ‘ \cdot ’ denotes the usual (right to left) composition of functions. This operation is not commutative. (S_n, \cdot) forms a group called the *symmetric permutation group*. [...] Let (Γ, \cdot) be a finite permutation group with I as identity. Let $\Omega \subseteq \Gamma$ be a *generator set* for Γ , such that

- a. if $g \in \Omega$ then $g^{-1} \in \Omega$
- b. $I \notin \Omega$.

Given (Γ, Ω) , define a Cayley graph $\mathcal{G} = (V, E)$ as follows:

$$V = \Gamma$$
$$E = \{(x, y)_g \mid x, y \in V \text{ and } g \in \Omega \text{ such that } y = x \cdot g\}.$$

i.e. two directed edges $(x, y)_g$ and $(x, y)_{g^{-1}}$ are viewed as an undirected edge (x, y) in the graph \mathcal{G} . Since Ω is a generator set for Γ , clearly \mathcal{G} is connected and $|\Omega|$ dictates the degree and the diameter of the Cayley graph \mathcal{G} .' [88]

clique:

'In an undirected graph \mathcal{G} a set of vertices C is called a clique if every two vertices of C are connected by an edge.' [49]

colorability:

Given an arbitrary graph \mathcal{G} :

chromatic index:

See 'Edge Colorability.'

chromatic number:

See 'Vertex Colorability.'

edge colorability:

'An edge coloration is a decomposition of the edges in a graph \mathcal{G} into l classes $\mathcal{G} = \mathcal{H}_1 + \mathcal{H}_2 + \dots + \mathcal{H}_l$ where no edges in the same class have vertices in common. This may be considered as a coloration of the edges such that no edges with the same color are incident. [...] The smallest number l for any edge coloration is the *chromatic index* $\chi'(\mathcal{G})$.' [106]

vertex colorability:

'An arbitrary graph \mathcal{G} is said to be vertex colorable in k colors when its vertex set V can be decomposed into k disjoint sets: $V = C_1 + C_2 + \dots + C_k$ such that no edges connect vertices in the same set, that is, the sets C_i are *independent* in \mathcal{G} . The decomposition defines a *color function* $f(v)$ for the vertices v of \mathcal{G} when one puts $f(v) = i$, when $v \in C_i$. Then, no vertices with the same color value i are connected by an edge. [...] The smallest number k such that \mathcal{G} is k -colorable is the *chromatic number* $\chi(\mathcal{G})$. When \mathcal{G} is the complete graph on n vertices, one has $\chi(\mathcal{G}) = n$; when \mathcal{G} is the null-graph, that is, has no edges, then $\chi(\mathcal{G}) = 1$. [106]

connectivity:

articulation point:

‘A vertex v of a graph \mathcal{G} is an articulation point of \mathcal{G} if the graph $\mathcal{G} - v$ will consist of a greater number of components than \mathcal{G} . If \mathcal{G} is connected, then $\mathcal{G} - v$ will contain at least two components, that is, $\mathcal{G} - v$ will be not connected. [...] The following theorem presents an equivalent definition of an articulation point.

Theorem 21 *A vertex v is an articulation point of a connected graph \mathcal{G} if and only if there exist two vertices u and w distinct from v such that v is on every path from u to w .* [147]

biconnectivity:

‘A *biconnected graph* is a connected graph with no articulation points. A maximal biconnected subgraph of a graph is called *connected component* of the graph.’ [147]

bridge:

‘A bridge of a graph \mathcal{G} is an edge e such that $\mathcal{G} - e$ has more components than \mathcal{G} .’ [147]

bridgeless graph:

A bridgeless graph is a graph having no bridges.

cut:

‘Let S be a subset of vertices of a graph $\mathcal{G} = (V, E)$. The set of edges connecting vertices of S with $V - S$ is called cut defined by S .’ [49]
See also [97] for cuts of cubic graphs.

edge connectivity:

The edge connectivity $\kappa'(\mathcal{G})$ of a graph \mathcal{G} is the minimum number of edges whose removal from \mathcal{G} results in a disconnected graph. In other words, $\kappa'(\mathcal{G})$ is the number of edges in a cut having the minimum number of edges.’ [147]

vertex connectivity:

‘The vertex connectivity $\kappa(\mathcal{G})$ of a graph \mathcal{G} is the minimum number of vertices whose removal from \mathcal{G} results in a disconnected graph.’ [147]

Euler Tour procedure:

‘Let $\mathcal{T} = (V, E)$ be a given tree and let $\mathcal{T}' = (V, E')$ be the directed graph obtained from \mathcal{T} when each edge $\{u, v\} \in E$ is replaced by two arcs (u, v) and (v, u) . Since the indegree of each vertex of \mathcal{T}' is equal to its outdegree, \mathcal{T}' is an eulerian graph; that is, it has a direct circuit that traverses each arc exactly once. It turns out that an Euler circuit of \mathcal{T}' can be used for the optimal parallel computation of many functions on \mathcal{T} . [...] An Euler tour of $\mathcal{T}' = (V, E')$ can be defined by specifying the successor function s mapping each arc $e \in E'$ into the arc $s(e) \in E'$ that follows e on the circuit. There is a simple way to introduce a suitable successor function. For each vertex $v \in V$ of the tree \mathcal{T} , we fix a certain ordering on the set of vertices adjacent to v —say, $adj(v) = (u_0, u_1, \dots, u_{d-1})$, where d is the degree of v . We define successor of each arc $e = (u_i, v)$ as follows: $s((u_i, v)) = (v, u_{(i+1) \bmod d})$, for $0 \leq i \leq d - 1$.’ [77]

girth:

‘The girth of a graph \mathcal{G} is the length of a shortest cycle (if any) in \mathcal{G} .’ [68]

independence:

independence set:

‘Consider a graph $\mathcal{G} = (V, E)$. A subset S of V is an *independent set* of \mathcal{G} if no two vertices of S are adjacent in \mathcal{G} . An independent set S of \mathcal{G} is maximum if \mathcal{G} has no independent set S' with $|S| > |S'|$.’ [147]

independence number:

The number of vertices in a maximum independent set of \mathcal{G} is called the independence number of \mathcal{G} and is denoted by $\alpha_0(\mathcal{G})$.’ [147]

matching:

‘A set of edges M of a graph $\mathcal{G} = (V, E)$ with no self-loops is called a matching if every vertex is incident to at most one edge of M .’ [49]

\mathcal{NC} :

‘The class \mathcal{NC} consists of all those decision problems that are solvable on a PRAM that simultaneously obeys a polylogarithmic bound on

the running time and a polynomial bound on the number of processors used. More informally, we might say that \mathcal{NC} consists of those problems solvable with a polynomial-bounded amount of hardware in polylog time. As with the sequential complexity classes, this class is substantially model-independent.’[153]

network:

A network \mathcal{N} can be assimilated to a graph, therefore it is a pair (V, E) , where V is a set of processors or other identities (that in general we will call nodes) and E is a set of communication links (undirected edges, where not differently specified), that connect nodes among them. Sometimes, certain nodes acquire a particular role, since they are chosen as *input* and *output* nodes. All other nodes are considered as switches of the network, routing messages toward their incident edges.

array network:

Processors are disposed on a straight line, therefore each (interior) node has a right neighbor and a left neighbor, except two (outermost) nodes, having only one neighbor. ‘Each interior processor is connected with bidirectional links to its left neighbor and its right neighbor. The outermost processors may have just one connection each, and may serve as input/output points for the entire network [...] An *r-dimensional N-sided array* has N^r nodes and $rN^r - rN^{r-1}$ edges. Each node corresponds to an N -ary r -vector (i_1, i_2, \dots, i_r) where $1 \leq i_j \leq N$ for $1 \leq j \leq r$. Two nodes are linked by an edge if they differ in precisely one coordinate and if the absolute value of the difference in that coordinate is 1.’ [91]

rearrangeable network:

‘A network with N inputs and N outputs is said to be rearrangeable if for any one-to-one mapping π of the inputs to the outputs, we can construct edge-disjoint path in the network linking the i -th input to the $\pi(i)$ -th output for $1 \leq i \leq N$.’[91]

Beneš network:

‘The Beneš network consists of two back-to-back butterflies. [...] Overall, the r -dimensional Beneš network has $2^r + 1$ levels, each with 2^r nodes. The first and last $r + 1$ levels in the network form an r -dimensional butterfly (the middle level of the Beneš network is shared

by these butterflies). Not surprisingly, the Beneš network is very similar to the butterfly, in terms of both its computational power and its network structure. Indeed, at first glance, the network hardly seems worth defining at all. The reason for defining the Beneš network is that it is an excellent example of rearrangeable network. Indeed, we can have two inputs for each level 0 node and two outputs for every level $2r$ node, and still connect every permutation of inputs to outputs with edge-disjoint paths.’ [91]

butterfly network:

See Section 4.1.

de Bruijn network:

‘The r -dimensional de Bruijn network consists of 2^r nodes and 2^{r+1} directed edges. Each node corresponds to an r -bit binary string, and there is a directed edge from each node $u_1u_2 \dots u_{\log N}$ to nodes $u_2 \dots u_{\log N}0$ and $u_2 \dots u_{\log N}1$. [...] In addition to having outdegree 2, every node of the de Bruijn network also has indegree 2.’ [91]

hypercube network:

‘The r -dimensional hypercube has $N = 2^r$ nodes and $r2^r$ edges. Each node corresponds to an r -bit binary string, and two nodes are linked with an edge if and only if their binary strings differ in precisely one bit. As a consequence, each node is incident to $r = \log N$ other nodes, one for each bit position.’ [91]

shuffle-exchange network:

‘The r -dimensional shuffle-exchange network has $N = 2^r$ nodes and $3 \cdot 2^{r-1}$ edges. Each node corresponds to a unique r -bit binary number, and two nodes u and v are linked by an edge if either

1. u and v differ in precisely the last bit, or
2. u is a left or right cyclic shift of v .

If u and v differ in the last bit, the edge is called an *exchange edge*. Otherwise, the edge is called a *shuffle edge*.’ [91]

star network:

‘An n -dimensional star network \mathcal{S}_n has $n!$ nodes that are in a 1-1 correspondence with the permutations $[p_1p_2 \dots p_n]$ of the set $\{1, 2, \dots, n\}$. Two nodes of \mathcal{S}_n are connected by one of the $n! \times \frac{(n-1)}{2}$ edges if and only if the permutation of one node can be obtained from the other by

interchanging the first symbol p_1 with the i -th symbol $p_i, 2 \leq i \leq n$.
 [...] The star network has uniform node degree $n - 1$.' [66]

\mathcal{NP} :

'The class \mathcal{NP} is defined informally to be the class of all decision problems that, under reasonable encoding schemes, can be solved by polynomial time non-deterministic algorithms. [...] It should be evident that a "polynomial time non-deterministic algorithm" is basically a definition device for capturing the notion of polynomial time verifiability, rather than a realistic method for solving decision problems.' [58]

\mathcal{NP} -complete problem:

'Let \mathcal{D}_1 and \mathcal{D}_2 be decision problems. We say that there exists a *polynomial reduction* of \mathcal{D}_1 to \mathcal{D}_2 ($\mathcal{D}_1 \propto \mathcal{D}_2$) if there exists a function $f(I_1)$ from the set of inputs of \mathcal{D}_1 to the set of inputs of \mathcal{D}_2 , such that the answer to I_1 is 'yes' with respect to \mathcal{D}_1 , if and only if the answer to $f(I_1)$ is 'yes' with respect to \mathcal{D}_2 , and there exists a polynomially bounded algorithm to compute $f(I_1)$. [...] If $\mathcal{D}_1 \propto \mathcal{D}_2$ and \mathcal{D}_2 can be answered by a polynomially bounded algorithm \mathcal{A}_2 then \mathcal{D}_1 is also solvable by a polynomially bounded algorithm: Given an input I_1 for \mathcal{D}_1 , use first the polynomially bounded algorithm to produce $f(I_1)$, and assume this computation is bounded by the polynomial $q(n)$, where n is the length of I_1 . Now, use \mathcal{A}_2 to answer $f(I_1)$ with respect to \mathcal{D}_2 . Let $p(m)$ be the polynomial bounding the computation time of \mathcal{A}_2 , where m is the length of $I_2 = f(I_1)$. Since $m \leq q(n)$, the total computation time to answer \mathcal{D}_1 is bounded by $q(n) + p(q(n))$, which is clearly polynomial. Following Karp's approach [86], let us define a decision problem \mathcal{D} to be \mathcal{NP} -complete if $\mathcal{D} \in \mathcal{NP}$ and for every $\mathcal{D}' \in \mathcal{NP}$, $\mathcal{D}' \propto \mathcal{D}$. [...] Note that the relation \propto is transitive; i.e. if $\mathcal{D}_1 \propto \mathcal{D}_2$ and $\mathcal{D}_2 \propto \mathcal{D}_3$ then $\mathcal{D}_1 \propto \mathcal{D}_3$.' [49]

optimization:

\mathcal{NP} Optimization problem (\mathcal{NPO}):

'An \mathcal{NP} Optimization problem B is a fourtuple $(I, sol, m, goal)$ such that:

1. I is the set of the instances of B and it is recognizable in polynomial time.

2. Given an instance x of I , $sol(x)$ denotes the set of feasible solutions of x . These solutions are short, that is, a polynomial p exists such that, for any $y \in sol(x)$, $|y| \leq p(|x|)$. Moreover, it is decidable in polynomial time whether, for any x and for any y such that $|y| \leq p(|x|)$, $y \in sol(x)$.
3. Given an instance x and a feasible solution y of x , $m(x, y)$ denotes the positive integer measure of y . The function m is computable in polynomial time and is also called the objective function.
4. $goal \in \{max, min\}$.

The class \mathcal{NPO} is the set of all \mathcal{NP} Optimization problems.' [38]

performance ratio:

'Let B be an \mathcal{NP} Optimization problem. Given an instance x and a feasible solution y of x , we define the performance ratio of y with respect to x as

$$R(x, y) = \max \left\{ \frac{m(x, y)}{opt(x)}, \frac{opt(x)}{m(x, y)} \right\}.$$

The performance ratio is always a number greater than or equal to 1 and is as close to 1 as y is close to the optimum solution.' [38]

\mathcal{APX} :

'Let B be a \mathcal{NP} Optimization problem and let \mathcal{A} be an algorithm that, for any instance x of B , returns a feasible solution $\mathcal{A}(x)$ of x . Given an arbitrary function $r : \mathbb{N} \rightarrow (1, \infty)$, we say that \mathcal{A} is an $r(n)$ -approximate algorithm for B if, for any instance x , the performance ratio of the feasible solution $\mathcal{A}(x)$ with respect to x verifies the following inequality:

$$R(x, \mathcal{A}(x)) \leq r(|x|).$$

If a problem admits an $r(n)$ -approximate polynomial time algorithm we say that it is approximable within $r(n)$. An \mathcal{NP} Optimization problem B belongs to the class \mathcal{APX} if it is approximable within ϵ , for some constant $\epsilon > 1$.' [38]

planarity:

planar graph:

A graph \mathcal{G} is said to be *embeddable* on a surface S if it can be drawn on S so that its edges intersect only at their end points. A graph is said to be planar if it can be embedded in the plane.

plane graph:

A plane graph has already been embedded in the plane.

face:

We will refer to the regions defined by a plane graph as its faces, the unbounded region being called *exterior face*.

Euler formula:

for any plane graph having n vertices, m edges and f faces, $n - m + f = 2$.' [68]

pointer jumping technique:

'The pointer jumping (or path doubling) technique allows the fast processing of data stored in the form of a set of rooted-directed trees. [...] This technique consists of updating the successor of each vertex by that successor's successor. As the technique is applied repeatedly, the successor of a vertex is an ancestor that becomes closer and closer to the root of the tree containing that vertex. As a matter of fact, the distance between a node and its successor *doubles* unless the successor of the successor vertex is a root.' [77]

PRAM:

'In the *shared-memory model* many processors have access to a single shared memory unit. More precisely, the shared-memory model consists of a number of processors, each of which has its own local memory and can execute its own local program, and all of which communicate by exchanging data through a shared memory unit. [...] In the *synchronous* mode of operation, all the processors operate synchronously under the control of a common clock. A standard name for the synchronous shared-memory model is the *parallel random-access machine (PRAM) model*. [...] There are several variations of the PRAM model based on the assumptions regarding the handling of the simultaneous access of several processors to the same location of the global memory. The *exclusive read exclusive write (EREW)* PRAM does not allow any simultaneous access to a single memory location. The *concurrent read exclusive write (CREW)* PRAM allows simultaneous access for a read instruction only. Access to a location for a read or a write instruction is allowed in the *concurrent read concurrent write (CRCW)* PRAM. The

three principal varieties of CRCW PRAMs are differentiated by how concurrent writes are handled. The *common* CRCW PRAM allows concurrent writes only when all processors are attempting to write the same value. The *arbitrary* CRCW PRAM allows an arbitrary processor to succeed. The *priority* CRCW PRAM assumes that the indices of the processors are linearly ordered, and allows the one with the minimum index to succeed. Other variations of the CRCW PRAM model exist.' [77]

prefix sum technique:

'Consider a sequence of n elements $\{x_1, x_2, \dots, x_n\}$ drawn from a set S with a binary associative operation, denoted by $+$. The prefix sums of this sequence are the n partial sums defined by $s_i = x_1 + x_2 + \dots + x_i$, $1 \leq i \leq n$. [...] We can use a balanced binary tree to derive a fast parallel algorithm to compute the prefix sums. Each internal vertex represents the application of the operation $+$ to its children during a forward traversal of the tree. Hence, each vertex v holds the sum of the elements stored in the leaves of the subtree rooted at v . During a backward traversal of the tree, the prefix sums of the data stored in the vertices at a given height are computed.' [77]

spanning tree:

'Assume $\mathcal{G}=(V, E)$ is a finite, connected (undirected) graph. [...] A subgraph of \mathcal{G} , which contains all of its vertices and is a tree is called a spanning tree of \mathcal{G} .' [49] Set E results implicitly partitioned into two subsets: all edges of \mathcal{G} belonging to the spanning tree are called *tree edges*, the other ones, giving no contribution to the tree are called *non-tree edges*.

Maximum Leaf Spanning Tree Problem:

The problem of finding a spanning tree of any graph \mathcal{G} is easy to solve. Not always easy is the problem of finding a spanning tree with some constraints. The Maximum Leaf Spanning Tree Problem has as instance a graph \mathcal{G} and the solution is a spanning tree such that the number of its leaves is maximized. The problem is, in general, \mathcal{NP} -complete.

vertex cover:

‘A vertex and an edge of a graph are said to *cover* each other if they are incident. A set of vertices which covers all the edges of a graph \mathcal{G} is called a vertex cover for \mathcal{G} . [...] The smallest number of vertices in any vertex cover for \mathcal{G} is called its *vertex covering number* and is denoted by $\alpha_0(\mathcal{G})$. [...] A vertex cover is called minimum if contains $\alpha_0(\mathcal{G})$ elements.’ [68]

Bibliography

- [1] AHO, A. – HOPCROFT, J. K. – ULLMAN, J. D.: *The design and analysis of computer algorithms*. Addison Wesley, Reading, MA, 1973.
- [2] AHUJA, R. K. – MAGNANTI, T. L. – ORLIN, J. B.: *Network flows*. Prentice-Hall, 1993.
- [3] AKERS, S. B. – KRISHNAMURTHY, B.: A group-theoretic model for symmetric interconnection networks. *International Conference Parallel Processing*, 1986, pp 216–233.
- [4] ALBERTSON, M. O. – HAAS, R.: Parsimonious Edge Coloring. *Discrete Math.* 148, 1996, pp 1–7.
- [5] ALIMONTI, P. – CALAMONERI, T.: Improved Approximations for Independent Dominating Set in Bounded Degree Graphs. *Proc. 22-th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '96)*, Lectures Notes in Computer Science 1197, Springer-Verlag, 1996, pp 2–16.
- [6] ANNEXSTEIN, F. – BAUMSLAG, M. – ROSENBERG, A. L.: Group Action Graphs and Parallel Architectures. *SIAM J. Comput.*, 19(3), 1990, pp 544–569.
- [7] ARDEN, B. W. – TANG, K. W.: Representations and Routing for Cayley Graphs. *IEEE Trans. on Commun.*, 39(11), 1991, pp 1533–1537.
- [8] AVIOR, A. – CALAMONERI, T. – EVEN, S. – LITMAN, A. – ROSENBERG, A. L.: A Tight Layout of the Butterfly Network. *Proc. 8-th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '96)*, ACM Press Ed., 1996, pp 170–175.
- [9] BAMPIS, E. – GIANNAKOS, A. – KARZANOV, A. – MANOUSSAKIS, Y. – MILIS, I.: Perfect Matching in General vs. Cubic Graphs: The Planar and Bipartite Cases. *Tech. Rpt. No.12*, 1995.
- [10] BARNETTE, D.: Trees in polyedral graphs. *Canad. J. Math.*, 18, 1966, pp 731–736.
- [11] BELLARE, M. – GOLDWASSER, S. – LUND, C. – RUSSELL, A.: Efficient probabilistically checkable proofs and applications to approximation. *Proc. of the 25th Annual ACM Symp. on Theory of Comp.*, 1995, pp 294–304.
- [12] BELLARE, M. – SUDAN, M.: Improved non-approximability results. *Proc. of the 26th Annual ACM Symp. on Theory of Comp.*, 1994, pp 184–193.
- [13] BENEŠ, V.: Permutation groups, complexes, and rearrangeable multistage connecting networks. *Bell System Technical Journal*, 43, July 1964, pp 1619–1640.

- [14] BERGE, C.: Alternating chain methods: a survey. In [121]
- [15] BERMAN, P. – FUJITO, T.: On Approximation Properties of the Independent Set Problem for Low Degree Graphs. *Proc. 4th Workshop on Algorithms and Data Structures (WADS'95)*, Lecture Notes in Computer Science 955, Springer Verlag, 1995, pp 449–460.
- [16] BERMAN, P. – FÜRER, M.: Approximating Maximum Independent Set in Bounded Degree Graphs. *Proc. 5th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'94)*, ACM-SIAM, 1994, pp 365–371.
- [17] BERMAN, F. – SNYDER, L.: On mapping parallel algorithms into parallel architectures. *J. Parallel Distr. Comput.* 4, 1987, pp 439–458.
- [18] BHATT, S. N. – CHUNG, F. R. K. – HONG, J.-W. – LEIGHTON, F. T. – OBRENIĆ, B. – ROSENBERG, A. L. – SCHWABE, E. J.: Optimal emulations by butterfly-like networks. *J. ACM*, 1996, to appear.
- [19] BHATT, S. N. – COSMADAKIS, S.: The Complexity of Minimizing Wire Lengths in VLSI Layouts. *Inform. Processing Letters*, 25(4), pp 263–267, 1987.
- [20] BHATT, S. N. – LEIGHTON, F. T.: A Framework for Solving VLSI Graph Layout Problems. *Journ. of Computer and Systems Sciences*, 28, 1984, pp 300–343.
- [21] BHATT, S. N. – LEISERSON, C. E.: Minimizing the Longest Edge in a VLSI Layout. *MIT VLSI Memo*, 1982, pp 82–86.
- [22] BIEDL, T.: New Lower Bounds for Orthogonal Graph Drawings. *Proc. Graph Drawing 95 (GD '95)*, Lectures Notes in Computer Science 1027, Springer-Verlag, 1995, pp 28–39.
- [23] BIEDL, T. – KANT, G.: A Better Euristic for Orthogonal Graph Drawings. *Proc. 2nd European Symposium on Algorithms (ESA '94)*, Lectures Notes in Computer Science 855, Springer-Verlag, 1994, pp 24–35.
- [24] BJORKEN, J. D. – DRELL, S. D.: *Relativistic Quantum Fields*. Mc-Graw Hill, New York, 1965.
- [25] BOKARI, S. H.: On the mapping problem. *IEEE Trans. Comp., C-30*, 1981, pp 207–214.
- [26] BONDY, J. A. – ENTRINGER, R. C.: Longest cycles in 2-connected graphs with prescribed maximum degree. *Canad. J. Math.*, 32, 1980, pp 1325–1332.
- [27] BROOKS, R. L.: On colouring the nodes of a network. *Proc. Cambridge Phil. Soc.*, 37, 1941 pp 194–197.
- [28] CALAMONERI, T. – PETRESCHI, R.: An Efficient Orthogonal Grid Drawing Algorithm for Cubic Graphs. *Proc. 1-st Annual International Conference on Computing and Combinatorics (Cocoon '95)*, Lectures Notes in Computer Science 959, Springer-Verlag, 1995, pp 31–40.
- [29] CALAMONERI, T. – PETRESCHI, R.: A Parallel Algorithm for Orthogonal Grid Drawings of Cubic Graphs. *Proc. 5-th Italian Conference on Theoretical Computer Science (ICTCS '95)*, World Scientific Publ. , 1995, pp 118–133.
- [30] CALAMONERI, T. – PETRESCHI, R.: Cubic Graphs as model of Real Systems. *Proc. Matrices and Graphs: Theory and Applications*, World Scientific Publ., 1995, to appear.

- [31] CALAMONERI, T. – PETRESCHI, R.: Visual Representations of Trivalent Cayley Interconnection Networks. *Proc. 11-th International Symposium on Computer and Information Sciences (ISCIS-XI)*, 1996, pp 555–564.
- [32] CALAMONERI, T. – PETRESCHI, R.: A New 3D Representation of Trivalent Cayley Networks. To appear on *Inform. Processing Letters*, 1997.
- [33] CALAMONERI, T. – STERBINI, A.: Drawing 2-, 3- and 4-colorable Graphs in $O(n^2)$ volume. *Proc. Graph Drawing '96*, Lectures Notes in Computer Science, Springer-Verlag, 1996, to appear. Also to appear on *Inform. Processing Letters*, 1997
- [34] CARLSSON, G. – CRUTHIRDS, J.E. – SEXTON, H.B. – WRIGHT, C.G.: Interconnection networks based on a generalization of cube-connected cycles. *IEEE Trans. Comp.*, C-34, 1985, pp 769–772.
- [35] CHVÁTAL, V.: Tough graphs and hamiltonian circuits. *Discrete Math.*, 5, 1973, pp 215–228.
- [36] COHEN, R. F. – EADES, P. – LIN, T. – RUSKEY, F.: Three-dimensional graph drawing. *Proc. Graph Drawing '96*, Lecture Notes in Computer Science, Springer-Verlag , 894, 1994, pp 1-11. Also in *Algorithmica*, 17, 1997, pp 199–209.
- [37] COLE, R.: Parallel Merge Sort. *SIAM J. Comput.*, 17(4), 1988, pp 770–785.
- [38] CRESCENZI, P. – KANN, V.: A compendium of NP optimization problems. *Tech. Rep. SI/RR-95/02, Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza"*, 1995. The list is updated continuously. The latest version is available as <http://www.nada.kth.se/theory/problemlist.html>.
- [39] DAHLHAUS, E. – KARPINSKI, M.: Perfect Matching for regular graphs is AC^0 -hard for the general matching problem, *Journal of Computer and System Sciences*, 44, 1992, pp 94–102.
- [40] DAY, K. – TRIPATHI, A.: Arrangement graphs: A class of generalized star graphs, *Inform. Process. Letters*, 42, 1992, pp 235–241.
- [41] DI BATTISTA, G. – EADES, P. – TAMASSIA, R. TOLLIS: Algorithms for Drawing Graphs: an Annotated Bibliography *Computational Geometry: Theory and Applications.*, 4(5), 1994, pp 235–282.
- [42] DI BATTISTA, G. – LIOTTA, G. – VARGIU, F.: Spirality of Orthogonal Representations and Optimal Drawings of Series-Parallel Graphs and 3-Planar Graphs. *Proc. 2-nd Workshop on Algorithms and Data Structures (WADS '93)*, Lectures Notes in Computer Science 709, Springer-Verlag ,1993, pp 151–162.
- [43] DINITZ, Y.: A compact layout of butterfly on the square grid. *Tech. Rpt. 873, The Technion*, 1995.
- [44] EDMONDS, J.: Paths, trees and flowers. *Canad. J. Math*, 17, 1995, pp 449–467.
- [45] EPPSTEIN, D. – GALIL, Z. – ITALIANO, G.F. – NISSENZWEIG, A.: Sparsification– A technique for speeding up dynamic graph algorithms. *Proc. of 33rd Ann. Symp. on Found. of Comp. Science (FOCS '92)*, 1992.
- [46] ERDÖS, P.: Problems and results in combinatorial analysis and graph theory. *Discrete Math.*, 72, 1988, pp 81–92.

- [47] EULER, L.: Solutio Problematis ad Geometriam Situs Pertinantis. *Academiae Petropolitanae*, 7, 1736, pp 128–140. Engl. Transl.: The Königsberg bridges. *Sci.Amer.*, 189, 1953, pp 66–70.
- [48] EVANS, J.R. – MINIEKA, E.: *Optimization Algorithms for Networks and Graphs*. Marcel Dekker Inc., 1992.
- [49] EVEN, S.: *Graph Algorithms*. Pitman, 1979.
- [50] EVEN, S. – LITMAN, A.: Layered cross product — a technique to construct interconnection networks. *4th ACM Symp. on Parallel Algorithms and Architectures*, 1992, pp 60–69.
- [51] EVEN, S. – TARJAN, R.E.: Computing an st-numbering. *Theoret. Comp. Sci.*, 2, 1976, pp 436–441.
- [52] FOUQUET, J. L: These d’etat, Universite de Paris-Sud, 9 Juin 1981.
- [53] FRANK, G. – HUI, D. – WARE, C.: Visualizing object oriented software in three dimensions. *Proc.CASCAN*, 1993.
- [54] FREDERICKSON, G. N.: Data Structures for On-line Updating of minimum Spanning Trees, with Applications. *SIAM J.Comput.*, 14(4), 1985, pp 781–798.
- [55] GALIL, Z. – ITALIANO, G. F.: Fully dynamic algorithms for edge connectivity problems. *Proc. 23rd ACM Symp. Theory of Computing (STOC ‘91)*, 1991, pp 317–327.
- [56] GALIL, Z. – ITALIANO, G. F.: Fully dynamic algorithms for 2-edge-connectivity. *SIAM Journal on Computing*, 21, 1992, pp 1047–1069.
- [57] GARDNER, M.: Mathematical Games: Snarks, Boojums and other conjectures related to the four-color-map theorem. *Scientific American*, 234 (4), 1976, pp 126–130.
- [58] GAREY, M. R. – JOHNSON, D. S.: *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Co., 1979.
- [59] GAREY, M. R. – JOHNSON, D. S. – STOCKMEYER, L.: Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1, 1976, pp 237–267.
- [60] GODDARD, W.: The Toughness of Cubic Graphs. *Graphs and Combinatorics.*, 12, 1996, pp 17–22.
- [61] GOEMANS, M. X.: A generalization of Petersen’s theorem. *Discrete Math.*, 115, 1993, pp 277–282.
- [62] GREENLAW, R. – PETRESCHI, R.: Cubic graphs. *ACM Computing Surveys*, 27 (4), 1995, pp 471–495.
- [63] GRÜNBAUM, B. – MOTZKIN, T.S.: Longest simple paths in polyedral graphs. *J. London Math. Soc.*, 37(2), 1962, pp 152–160.
- [64] GRÜNBAUM, B. – SHEPHARD, G.C.: *Tilings and Patterns*. W.H. Freeman and Co., New York, 1987.
- [65] GRÜNBAUM, B. – WALTHER, H.: Shortness exponents of families of graphs. *J. Combin.Theory ser.A*, 14, 1973, pp 364–385.
- [66] GU, Q. – PENG, S.: Node-to-node cluster fault tolerant routing in star graphs. *Inform. Process. Letters*, 56, 1995, pp 29–35.

- [67] HALLDORSSON, M. M.: Approximating the minimum maximal independence number. *Inform. Process. Lett.*, 46, 1993, pp 169–172.
- [68] HARARY, F.: *Graph Theory*. Addison-Wesley, Reading, MA, 1969.
- [69] HARARY, F. – KOVÁCS, P.: Smallest graphs with given girth pair. *Carribb. J. Math.*, 1, 1982, pp 24–26.
- [70] HARARY, F. – KOVÁCS, P.: Regular graphs with given girth pair. *J. of Graph Theory*, 7, 1983, pp 209–218.
- [71] HART, S. – SHARIR, M.: Nonlinearity of Davenport-schinzel sequences and of generalized path compression schemes. *Combinatorica*, 6, 1986, pp 151–177.
- [72] HOBBS, M. – SCHMEICHEL, E.: On the Maximum Number of Independent Edges in Cubic Graphs. *Discrete Math.*, 42, 1982, pp 317–320.
- [73] HOEL, P. G. – PORT, S. C. – STONE, C. J.: *Introduction to stochastic processes*. Houghton Mifflin Company, 1972.
- [74] HOLYER, I.: The NP-completeness of edge-coloring. *SIAM J. Comp.*, 10, 1981, pp 718–720.
- [75] HOPKINS, G. – STATON, W.: Extremal Bipartite Subgraphs of Cubic Triangle-Free Graphs. *J. Graph Theory*, 6, 1982, pp 115–121.
- [76] HORÁK, P. – QING, H. – TROTTER, W. T.: Induced Matchings in Cubic Graphs. *J. of Graph Theory.*, 17(2), 1993, pp 151–160.
- [77] JÁJÁ, J.: *An Introduction to Parallel Algorithms*. Addison Wesley, 1992.
- [78] JÁJÁ, J. – SIMON, J.: Parallel Algorithms in Graph Theory: Planarity Testing. *SIAM J. Comput.*, 11, 1982, pp 314–328.
- [79] JACKSON, B.: Longest Cycles in 3-Connected Cubic Graphs. *J. Combin. Theory ser. B*, 41, 1986, pp 17–26.
- [80] JOHNSON, D. S.: The NP-completeness column: An ongoing guide. *Journal of Algorithms*, 2(4), 1981, pp 393–405.
- [81] JOHNSON, E. L.: A proof of the four-coloring of the edges of a regular three-degree graph. *O.R.C., 63-28 (R.R.) Mimeographed report, Operation Research Center, Univ. of Calif.*, 1963.
- [82] JOHNSON, D. S.: Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9, 1974, pp 256–278.
- [83] KANN, V.: Maximum bounded degree 3-dimensional matching is MAX SNP complete. *Inform. Process. Lett*, 37, 1991, pp 27–35.
- [84] KANN, V.: On the Approximability of NP-Complete Optimization Problems. *PhD Thesis*, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, 1992.
- [85] KANT, G.: Drawing Planar Graphs Using the canonical ordering. *Proc. 33th Ann. IEEE Symp. on Found. of Comp. Science (FOCS '92)*, 1992, pp 101–110. Revised version in *Algorithmica - Special Issue on Graph Drawing*, 16, 1996, pp 4–32.
- [86] KARP, R. M.: Reducibility among Combinatorial Problems. In [142], pp 85–104.

- [87] KLEITMAN, D. – LEIGHTON, F.T. – LEPLEY, M. – MILLER, G.L.: New Layouts for the Shuffle-Exchange Graph, *Thirteenth Annual ACM Symposium on Theory of Computings (STOC '81)*, 1981, pp 278–292.
- [88] LAKSHMIVARAHAN, S. – JUNG-SING JWO – DHALL, S.K.: Symmetry in interconnection networks based on Cayley graphs of permutation groups: A survey. *Paral. Comput.*, 19, 1993, pp 361–407.
- [89] LANG, R. – WALTHER, H.: Über Längste Kreise in regulären Graphen. in *“Beiträge zur Graphentheorie”*, *Kolloquium Manebach*, 1967, pp 91–98.
- [90] LEIGHTON, F. T.: *Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks*. MIT Press, Cambridge, Mass, 1983.
- [91] LEIGHTON, F. T.: *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, Inc., 1992.
- [92] LEV, A. – PIPPENGER, N. – VALIANT, L.G.: A fast parallel algorithm for routing in permutation networks. *IEEE Trans. on Computers*, 30(2), 1981, pp 93–100.
- [93] LEWIN, K.: *Principles of Topological Psychology*. Mc-Graw-Hill, New York, 1936.
- [94] LIU, Y. – MARCHIORO, P. – PETRESCHI, R. – SIMEONE, B.: Theoretical Results on at most 1-bend embeddability of graphs. *Acta Mathematicae Applicatae Sinica*, 8(2), 1992, pp 188–192.
- [95] LIU, Y. – MARCHIORO, P. – PETRESCHI, R.: At most single bend embedding of cubic graphs. *Applied Mathematics (Chin. Journ.) ser.B*, 9(2), 1994, pp 127–142.
- [96] LOCKE, S. C.: Maximum k-colorable subgraphs. *J. Graph Theory*, 6, 1982, pp 123–132.
- [97] LOEBL, M.: Efficient Maximal Cubic Graphs Cuts. *Proc. of 18th International Colloquium on Automata, Languages and Programming (ICALP '91)*, Lecture Notes in Computer Science 510, Springer-Verlag, 1991, pp 351–362.
- [98] LUND, C. – YANNAKAKIS, M.: On the Hardness of Approximating Minimization Problems. *J. ACM* 41, 1994, pp 960–981.
- [99] MACKINLEY, J. – ROBERTSON, G. – CARD, S.: Cone trees: Animated 3d visualization of hierarchical information. *Proc. SIGCHI Conf. on Human Factors in Computing*, 1991, pp. 189–194.
- [100] MEAD, C. – CONWAY, L.: *Introduction to VLSI Systems*. Addison-Wesley, Reading, Mass., 1980.
- [101] MEHLHORN, K. – PREPARATA, F. P. – SARRAFZADEH, M.: Channel routing in knock-knee mode: simplified algorithms and proofs. *Algorithmica*, 1, 1986, pp 213–221.
- [102] MICALI, S. – VAZIRANI, V. V.: An $O(|V|^{1/2}|E|)$ algorithm for finding maximum matchings in general graphs. *Proc. of 21st Ann. Symp. on Found. of Comp. Science (FOCS '80)*, 1980, pp 17–23.
- [103] MOLLOY, M. – REED, B.: The Dominating Number of a Random Cubic Graph. *Random Structures and Algorithms*, 7 (3), 1995, pp 209–221.
- [104] MONIEN, B. – SPECKENMEYER, E.: Some Further Approximation Algorithms for the Vertex Cover Problem. *Proc. 8th Colloquium on Trees in Algebra and Programming (CAAP '83)*, Lecture Notes in Computer Science 159, Springer-Verlag, pp 341–349, 1983.

- [105] MONIEN, B. – SPECKENMEYER, E.: Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica*, 22, 1985, pp 115–123.
- [106] ORE, O.: *The four-color Problem*. Accademic Press, 1967.
- [107] PACH, J. – TÓTH, G.: Private communications, November 1996, and Three-dimensional grid drawings of graphs, Manuscript, 1997.
- [108] PAPADIMITRIOU, C. – YANNAKAKIS, M.: Optimization, Approximation, and Complexity Classes. *Journal of Computer and System Sciences*, 43, 1991, pp 425–440.
- [109] PAPAKOSTAS, A. – TOLLIS, I. G.: Improved Algorithms and Bounds for Orthogonal Drawings. *Proc. Graph Drawing '94 (GD '94)*, Lectures Notes in Computer Science 894, Springer-Verlag, 1994, pp 40–51.
- [110] PARKER, D. S.: Notes on Shuffle-Exchange Type Switching Networks. *IEEE Transactions on Computers*, C-29(3), 1980, pp 213–222.
- [111] PATERSON, M. – RUZZO, W. – SNYDER, L.: Bounds on Minimax Edge for Complete Binary Trees. *Thirteenth Annual ACM Symposium on Theory of Computings (STOC '81)*, 1981, pp 293–299.
- [112] PETERSEN, J.: Die Theorie der regulären graphen. *Acta Mathematica*, 15, 1891, pp 193–220. Engl. vers. in [158].
- [113] PINTER, R. Y.: On routing two-point nets across a channel. *19th ACM-IEEE Design Automation Conf.*, 1982, pp 894–902.
- [114] PRADHAN, D. K. – SAMATHAM, M. R.: The deBruijn multiprocessor network: a versatile parallel processing and sorting network for VLSI. *IEEE Trans. Comput.*, 38, 1989, pp 567–581.
- [115] PRATHER, R. E.: Design and Analysis of Hierarchical Software Metrics. *ACM Computing Surveys.*, 32, 1980, pp 331–334.
- [116] PREISSMANN, M.: Even Polyedral Decompositions of Cubic Graphs. *Discrete Math.*, 27(4), 1995, pp 331–334.
- [117] PREPARATA, F. P. – VUILLEMIN, J.: The Cube-Connected Cycles: A Versatile Network for Parallel Computation. *Communications of ACM*, 24(5), 1981, pp 300–309.
- [118] RABIN, M. O.: Maximum Matching in general graphs through randomization. *J. of Algorithms*, 10, 1989, pp 557–567.
- [119] RAUCH, M.: Fully dynamic biconnectivity in graphs. *Proc. 33rd IEEE Symp. Foundations of Computer Science (FOCS '92)*, 1992, pp 50–59.
- [120] RAUCH, M.: Improved data structures for fully dynamic biconnectivity. *Proc. 26th Symp. Theory of Computing (STOC '94)*, 1994, pp 686–695.
- [121] READ, R. C. editor: *Graph Theory and Computing*, Academic Press, New York and London, 1972.
- [122] READ, R. C.: *Some Enumeration Problems in Graph Theory*, Doctoral Thesis, London University, 1958.
- [123] REID, P.: Dynamic Interactive Display of Complex Data Structures. *Graphics Tools for Software Engineers*, Cambridge, 1989, pp. 62–70.

- [124] ROBINSON, R. W. – WORMALD, N. C.: Almost all Cubic Graphs are Hamiltonian. *Random Structures and Algorithms*, 3(2), 1992, pp 117–125.
- [125] ROSENBERG, A. L.: Issues in the Study of Graphs Embeddings. *Proc. 6-th International Workshop on Graph-Theoretic Concepts in Computer Science (WG80)*, Lectures Notes in Computer Science 100, Springer-Verlag, 1980, pp 150–176.
- [126] ROSENBERG, A. L. – HEATH, L. S.: *Graph Separators, with Applications*, 1996, in preparation.
- [127] SAAD, Y. – SCHULTZ, M. H.: Topological properties of hypercubes. *IEEE Trans. Comput.*, 37, 1988, pp 867–872.
- [128] SACHS, H.: On Regular Graphs with Given Girth. *Theory of Graphs and its Applications, Proc. of the Symp. held in Smolenice*, Academic Press 1963, pp 91–97.
- [129] SHARAN, R. – WIGDERSON, A.: A new \mathcal{NC} algorithm for Perfect Matching in Bipartite Cubic Graphs. *4-th Israel Symposium on Theory of Computing and Systems*, IEEE, 1996, pp 202–207.
- [130] SHILOACH, Y. – VISHKIN, U.: An $O(\log n)$ Parallel Connectivity Algorithm. *Journal of Algorithms*, 3(1), 1982, pp 57–67.
- [131] SNYDER, L.: Type architectures, shared memory, and the corollary of modest potential. *Ann. Rev. Computer Science 1*, 1986, pp 289–317.
- [132] STATON, W.: Edge deletion and the chromatic number. *Ars Combinatoria*, 10 1980, pp 103–106.
- [133] STEWART, I. A.: Deciding wheather a planar graph has a cubic subgraph is NP-complete. *Discr. Math.*, 126(1-3), 1994, pp 349–357.
- [134] STORER, J. A.: Constructing Full Spanning Trees for Cubic Graphs. *Inform. Process. Letters*, 13(1), 1981, pp 8–11.
- [135] STORER, J. A.: On Minimal-Node-Cost Planar Embeddings. *Networks*, 14, 1984, pp 181–212.
- [136] SZEKERES, G.: Polyhedral decompositions of cubic graphs. *Bull. Aust. Math. Soc.*, 8, 1973, pp 367–387.
- [137] TAIT, P. G.: On the Colouring of Maps. *Proc. Roy. Soc. Edinb.*, 10, 1878, pp 501–503.
- [138] TAIT, P. G.: Note on a theorem in the geometry of position. *Trans. Roy. Soc. Edinb.*, 29, 1880, pp 657–660.
- [139] TAMASSIA, R.: On Embedding a Graph in the Grid with the Minimum Number of Bends. *SIAM J. Comput.*, 16(3), 1987, pp 421–444.
- [140] TAMASSIA, R. – TOLLIS, I. G., – VITTER, J. S.: Lower Bounds and Parallel Algorithms for Planar Orthogonal Grid Drawings. *Proceedings IEEE Symposium on Parallel and Distributed Processing*, 1991, pp 1–8.
- [141] TAMASSIA, R. – VITTER, J. S: Parallel Transitive Closure and Point Location in Planar Structures. *SIAM J. Comput.*, 20(4), 1991, pp 708–725.
- [142] THATCHER, J. W. editor: *Complexity of Computer Computations*, Plenum Press, 1972.

- [143] THOMPSON, C. D.: Area-time complexity for VLSI. *11-th Annual ACM Symposium on Theory of Computing (STOC '79)*, 1979, pp 81–88.
- [144] THOMPSON, C. D.: *A complexity theory for VLSI*. Ph.D. thesis, Carnegie-Mellon Univ. Pittsburgh, 1980.
- [145] THOMPSON, C. D.: Fourier transforms in VLSI. *IEEE Trans. Comp. C-32*, 1983, pp 1047–1057.
- [146] THULLER, H: *Contribution à l' étude des graphes cubiques et des graphes gracieux*. Ph.D. thesis, University Paris-XI, 1987.
- [147] THULASIRAMAN, K. – SWAMY, M. N. S.: *Graphs: Theory and Algorithms*. John Wiley & Sons, Inc., 1992.
- [148] TUTTE, W. T.: On hamiltonian circuits. *J. London Math. Soc.*, 21, 1946, pp 98–101.
- [149] TVERSKY, O. – SNIBBE, O. – ZELEZNIK, R.: Cone trees in the uga graphics system: suggestions of a more robust visualization tool. *Technical Report CS-93-07, Brown University*, 1993.
- [150] TZVIELI, D.: *Minimal diameter double-ring networks I: Some very large infinite optimal families*. Louisiana State Univ., Baton Rouge, LA, 1988.
- [151] VADAPALLI, P. – SRIMANI, P. K.: Trivalent Cayley graphs for Interconnection Networks. *Informat.Process.Letters*, 54, 1995, pp 329–335.
- [152] VADAPALLI, P. – SRIMANI, P. K.: Shortest Routing in Trivalent Cayley Graph Networks. *Informat.Process.Letters*, 57(4), 1996, pp 183–188.
- [153] VAN LEEUWEN, J., editor: *Algorithms and Complexity*. Elsevier Science Publ., 1990.
- [154] VAZIRANI, V. V.: A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{VE})$ general graph maximum matching algorithm. *Combinatorica*, 14, 1994, pp 71–109.
- [155] VIZING, V. G.: On an estimate of the chromatic class of a p-graph. *Diskret. Analiz.*, 3, 1964, pp 23–30.
- [156] WALTHER, H.: Über die Anzahl der Knotenpunkte eines längsten Kreises in planaren, kubischen dreifach knotenzusammenhängenden Graphen. *Studia Sci. Math. Hungar.*, 2, 1967, pp 391–398.
- [157] WALTHER, H.: Über Extremalkreise in regulären Landkarten. *Wiss. Z. Techn. Hochsch. Ilmenau*, 15, 1969, pp 139–142.
- [158] WATKINS, J. J. – WILSON, R. J.: A survey of Snarks. *Graph Theory, Combinatorics and Applications, vol.II – Proc. of the 6th quadre. intern. conf. on the theory and applications of graphs*, 1991, pp 1129–1144.
- [159] WISE, D. S.: Compact layouts of banyan/FFT networks. *VLSI Systems and Computations* (H.T. Kung, B. Sproull, G. Steele, eds.) Computer Science Press, Rockville, Md., 1981, pp 186–195.
- [160] WONG, P. K.: Cages – A Survey. *J. Graph Theory*, 6 1982, pp 1–22.
- [161] WORMALD, N. C.: The Asymptotic Distribution of Short Cycles in Random Regular Graphs. *J. Combin. Theory ser. B*, 31 1981, pp 168–182.

- [162] YANNAKAKIS, M.: Node- and edge-deletion NP-complete problems. *Proc. of 10th Annual ACM Symp. on the Theory of Computing (STOC '78)*, Association for Computing Machinery, New York, 1978, pp 253–264.
- [163] ZÝKA, O.: On the Bipartite Density of Regular Graphs with Large Girth. *Journ. of Graph Theory*, 14(6) 1990, pp 631–634.