PANNON Egyetem
TÁMOP-4.1.2.A/1-11/1-2011-0088

Magyarország a Kelet-Európai logisztika központja – Innovatív logisztikai képzés e-learning alapú fejlesztése

Combinatorial Methods and Algorithms

Zsolt Tuza, Csilla Bujtás, Máté Hegyháti

2014

A tananyag a TÁMOP-4.1.2.A/1-11/1-2011-0088 projekt keretében a Pannon Egyetem és a Miskolci Egyetem oktatói által készült.

H-8200 Veszprém, Egyetem u. 10.
H-8201 Veszprém, Pf. 158.
Telefon: (+36 88) 624-911
Fax: (+36 88) 624-751
Internet: www.uni-pannon.hu

Nemzeti Fejlesztési Ügynökség
www.ujszechenyiterv.gov.hu
06 40 638 638

MAGYARORSZÁG MEGÚJUL

A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Contents

List of Figures

List of Tables

List of Algorithms

Preface

These lecture notes contain the material of the course "Combinatorial methods and algorithms", taught in the first or second semester of MSc in Engineering Information Technology and MSc in Logistic Engineering at the University of Pannonia.

Combinatorics and related optimization algorithms constitute a very rich and wide area, therefore we had no chance to survey the full range of important directions and approaches. In the current selection our intention was to collect problems of various kinds which, although are different, have several relations among themselves and, moreover, offer a way to present some characteristic methods in proofs and algorithms. To emphasize this aspect, in some cases we describe several substantially different arguments which prove the same assertion. Certainly, already one proof validates a theorem;[1] but there is a message in showing that the same goal can be reached along various ways.

The text is strictly theoretical in the sense that it follows the precise "mathematical" structure of definition–theorem–proof. On the other hand, the algorithms described here can be applied in a wide range of practical problems, and in fact they *are* applied in them. Still, we do not tell much about these connections in the lecture notes. The reason is that we traditionally give it as a homework to search for one or more applications which use the notions and methods of the course. Experience shows that students, depending on their fields of interest or specialization, find a large variety of quite different such themes on the internet. If time permits near the end of the course, it is very useful to organize a session of presentations during which everyone can learn what kinds of applications their classmates have found.

The course assumes some basic knowledge in graph theory. In many (but not all) curricula, this is taught in the first year of BSc studies. The first chapter of our notes helps in refreshing memory and to learn parts of earlier material which have possibly been missed. Throughout the text, we include

[1] provided that the proof is correct...

figures to make the formal discussion easier to follow; and also illustrate the algorithms with some explicit examples.

At several points, additional pieces of information of various kinds are provided in the "Notes" sections at the end of some chapters and also in footnotes. They may deal with topics which are beyond but related to the course material, or which possibly occurred in a way in previous studies of the reader; and occasionally minor details are mentioned which may clarify some steps of an argument if necessary. We also mention alternative terminology; it may be helpful in searching for applications and related literature.

We hope that students and readers will find these notes not only useful but also enjoyable.

Veszprém, January 2013

Zsolt Tuza
Csilla Bujtás
Máté Hegyháti

Chapter 0

Basic graph theory

In this preliminary section we list some definitions on graphs and set systems, that will be used throughout the text. In many curricula this material occurred already in some course before a student attends the current one, nevertheless it is safer to give a summary here, making these notes more self-contained.

0.1 Basic definitions

A fundamental mathematical structure studied in this course is the (undirected simple) graph.

Definition 0.1 (Graph (Undirected Simple); Vertex; Edge) *A graph is a pair $G = (V, E)$, where*

- *V is the set of vertices,*
- *E is the set of edges.*

Each edge connects exactly two vertices, i.e., $E \subseteq \{\{v, v'\} \mid v, v' \in V,\ v \neq v'\}$. In other words, the edges are unordered pairs of vertices.

Thus, a simple graph is a set of vertices and a set of edges connecting some vertex pairs. A vertex is sometimes called ***node*** — we shall use this term in a particular context — and an edge is sometimes called 'line' or 'link'. The number $|V|$ of vertices is called the ***order*** of G, and is very often denoted by n.

In general, an undirected graph may contain loops or parallel (or multiple) edges; for the sake of simplicity, however, if it is not stated otherwise, the term ***graph*** will be used for undirected simple graphs.

The following notation will also be used for $G = (V, E)$:

- $V(G)$: set of vertices of G, thus V;
- $E(G)$: set of edges of G, thus E.

Usually the vertices of a graph will be denoted by $v, v', v_1, v_2, \ldots$, while the edges will be denoted by $e, e_1, e_2, \ldots$ or by the pairs of their endpoints, like $vv', v_1v_2, \ldots$. In case of an undirected graph, the order of vertices in an edge is arbitrary, thus v_iv_j denotes the same edge as v_jv_i. Also, if $e = v_iv_j$ is an edge, v_i is said to be connected or joined to v_j by e, or that e joins v_i with v_j; and v_i, v_j are incident with e. Vertices v_i and v_j are the endpoints or the ends of edge e. A standard term to express that v_iv_j is an edge is that v_i and v_j are ***adjacent***, or that v_i is adjacent to v_j.

The graph $(\emptyset, \emptyset)$ is called the ***null graph***; we shall mention it only occasionally at some points, and usually leave it to the reader to decide which of the stated assertions are or are not valid for this degenerate case.

An example of a graph with 12 vertices and 18 edges is given in Figure 1. The vertices are denoted by dots, while edges are represented by continuous lines (curves).

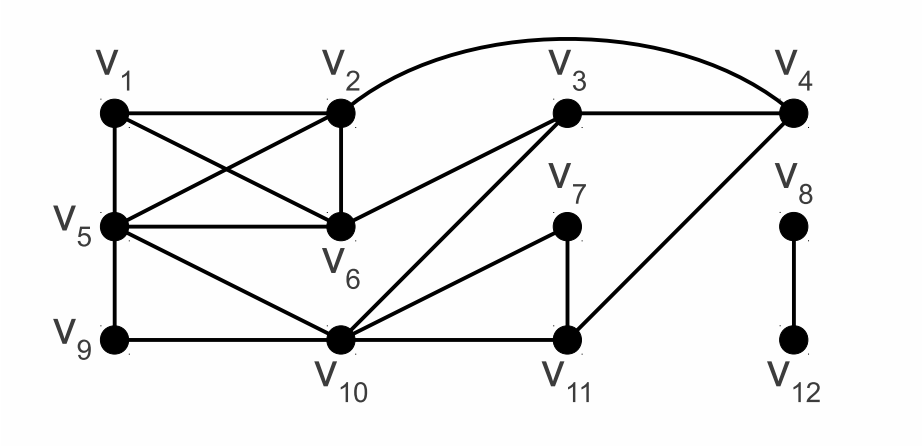


Figure 1: *An undirected simple graph*

Definition 0.2 (Isomorphism; Isomorphic graphs) *Two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic if there exists a mapping $f : V \to V'$ such that*

- *f establishes a bijection between V and V', and*
- *for any two vertices $v, v' \in V$, the pair $f(v)f(v')$ is an edge in G' if and only if vv' is an edge in G.*

A mapping f with these properties is called an isomorphism between G and G'.

Unless there is a special meaning attributed to the vertices and/or edges, isomorphic graphs need not be distinguished from each other.

A subgraph of a graph is obtained by removing some of the vertices and edges of the graph such that an edge must be removed if so is any of its endpoints.

Definition 0.3 (Subgraph) *Graph $G' = (V', E')$ is a subgraph of $G = (V, E)$ if G' is a graph, moreover $V' \subseteq V$ and $E' \subseteq E$. This relation is denoted by $G' \subseteq G$.*

Figure 2 shows three subgraphs of the graph in Figure 1. The first of them is the graph itself; it is always true that $G \subseteq G$. The second graph in the figure contains only a subset of vertices and edges; and finally, the third example has the same vertices, but no edges.

Induced subgraph is a particular kind of subgraph: it has to contain all the edges that have none of their endpoints removed.

Definition 0.4 (Induced subgraph) *Graph $G' = (V', E')$ is an induced subgraph of $G = (V, E)$ if $V' \subseteq V$, and $E' = \{e \in E \mid e \subseteq V'\}$.*

If we want to describe a subgraph $G' = (V', E')$, we have to specify both V' and E'. But an *induced* subgraph is determined already by its vertex set V'. We say that this is the subgraph induced by V', and denote it by $G[V']$.

In Figure 2 the second and third examples are not induced subgraphs. The first example, however, is an induced subgraph, as $G = G[V(G)]$ is always true. Some other induced subgraphs of the graph in Figure 1 are shown in Figure 3.

The neighbors of a vertex $v \in V(G)$ are the vertices adjacent to v in G. Formally:

Definition 0.5 (Neighborhood of a vertex)

$$N_G(v) = \{v' \in V(G) \mid vv' \in E(G)\}.$$

This is often called the open neighborhood of v. (The closed neighborhood also includes v itself.) Unless specified otherwise, 'neighborhood' always means open neighborhood.

In Figure 1, the neighbors of v_{10} are v_3, v_5, v_7, v_9, and v_{11}. Similarly, $N_G(v_7) = \{v_{10}, v_{11}\}$ and $N_G(v_9) = \{v_5, v_{10}\}$. If there is no danger of ambiguity, the subscript G is omitted to have a simpler notation, like $N(v_2) = \{v_1, v_4, v_5, v_6\}$.

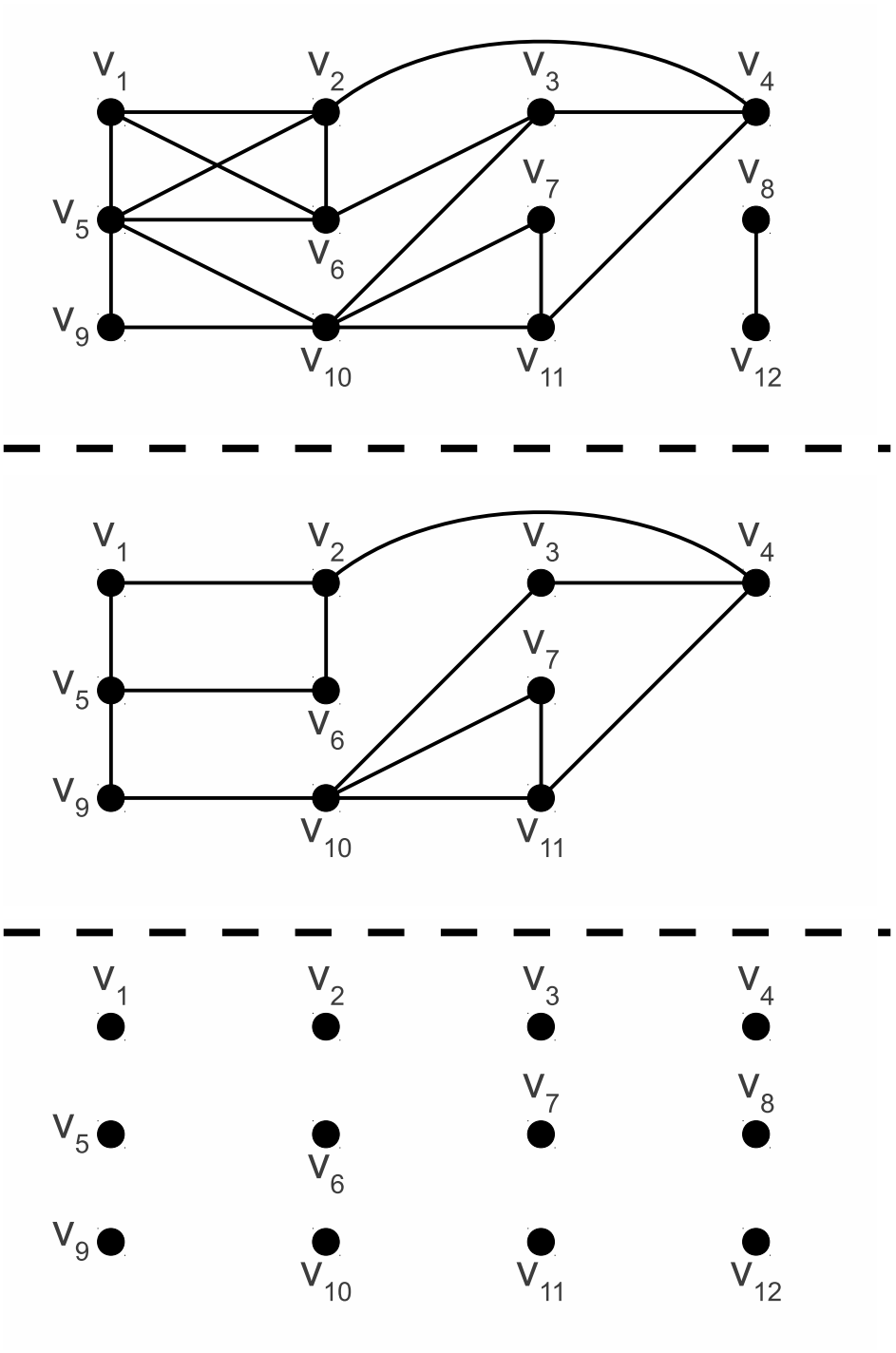


Figure 2: *Some subgraphs of the graph in Figure 1*

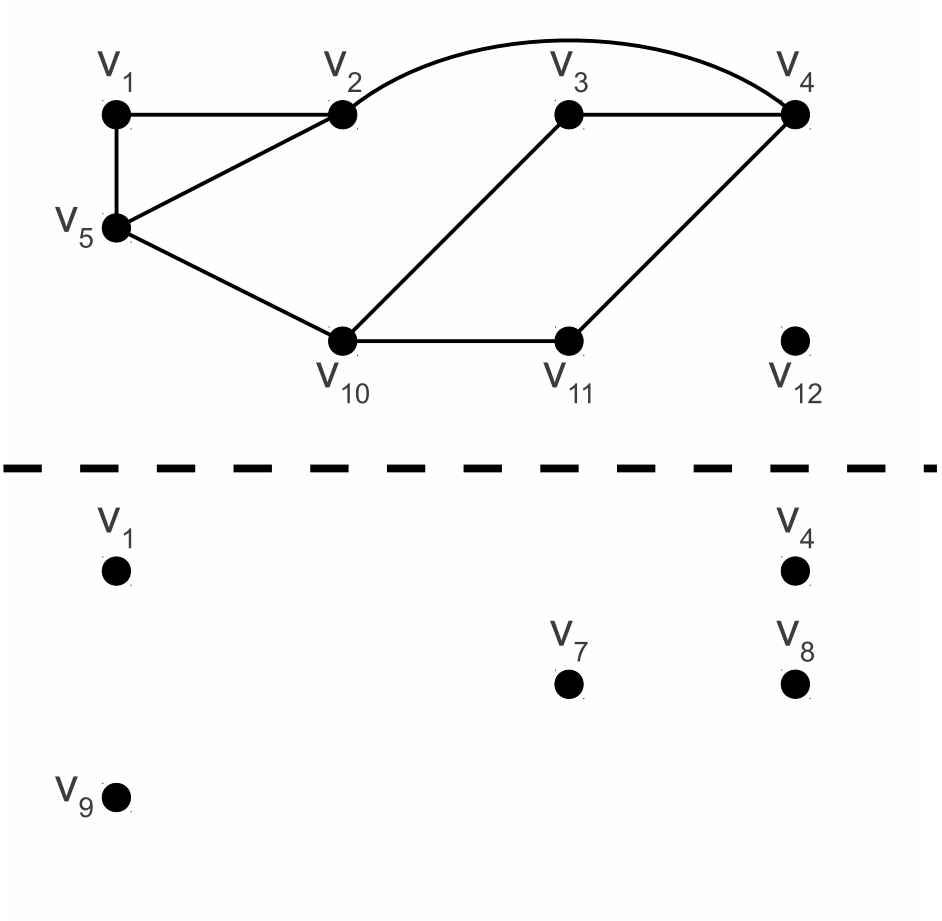


Figure 3: *Some induced subgraphs of the graph in Figure 1*

The degree of a vertex v, denoted by $d(v)$, is the number of edges containing that vertex.

Definition 0.6 (Degree) *In a simple undirected graph G,*

$$d(v) = |\{e \in E(G) \mid v \in e\}| = |N_G(v)|.$$

In Figure 1, $d(v_1) = 3$, $d(v_2) = 4$. A vertex of degree zero, i.e., a vertex not connected to any other vertex, is called an ***isolated vertex***.

A graph, all of whose vertices have the same degree, is called regular:

Definition 0.7 (Regular graph) *Graph G is k-regular if $d(v) = k$ for all $v \in V(G)$.*

There is a simple relation between the number of vertices and edges in a k-regular graph. It follows from the more general relation between degree sum and number of edges. The proof is based on the observation that each edge is incident with precisely two vertices.

Proposition 0.1 *If G is k-regular, then $k \cdot |V(G)| = 2 \cdot |E(G)|$. More generally, $\sum_{v \in V(G)} d(v) = 2 \cdot |E(G)|$ holds in every graph G.*

From this, it also follows that the number of odd-degree vertices is even in every graph.

A path is an alternating sequence of vertices and edges such that each edge connects two consecutive vertices. The sequence starts and ends with a vertex. Formally:

Definition 0.8 (Path) *A subgraph $v_0, e_1, v_1, \ldots, e_k, v_k$ is a path from v_0 to v_k in G if the vertices $v_0, v_1, \ldots, v_k$ are all distinct, and if $k \geq 1$ then $e_i = \{v_{i-1}, v_i\}$ and $e_i \in E(G)$ for all $i \in \{1, 2, \ldots, k\}$. (The single vertex v_0 itself is also considered to be a path.) The length of a path is the number of its edges.*

As an example, in Figure 1, v_1 , v_1v_6 , v_6 , v_6v_2 , v_2 , v_2v_4 , v_4 is a path of length three from v_1 to v_4. Since the edge set of a path is uniquely determined by the sequence of its vertices, we can simplify notation to just listing the vertices:

$$v_1v_6v_2v_4$$

means the same path as v_1 , v_1v_6 , v_6 , v_6v_2 , v_2 , v_2v_4 , v_4.

A graph is said to be connected if there is a path from each of its vertices to any other vertex.

Definition 0.9 (Connected graph; Disconnected graph) *Graph G is connected if there exists a path from v to v' for all $v, v' \in V(G)$. Graphs which are not connected are called disconnected graphs.*

Note that the graph in Figure 1 is not connected (i.e., is disconnected), as there is no path from v_{11} to v_{12}. These vertices are in different components of the graph. The components of a graph are its inclusionwise largest connected (induced) subgraphs, more formally:

Definition 0.10 (Component) *The induced subgraph $G[V']$ is a component of G — also called connected component — if it is connected, and for all $v \in V(G) \setminus V'$, the subgraph $G[V' \cup \{v\}]$ is disconnected.*

It is important to note that the components are mutually vertex-disjoint.

Proposition 0.2 *The components partition the vertex set, and this partition is unique.*[1]

If we identify the two ends of a path, we obtain a cycle. Formally:

Definition 0.11 (Cycle) *A subgraph $v_0, e_1, v_1, \ldots, e_k, v_k$ is a cycle in G if $v_0 = v_k$, $e_i = \{v_{i-1}, v_i\}$, $e_i \in E(G)$ for all $i \in \{1, 2, \ldots, k\}$, and $v_i \neq v_j$ for any $i, j \in \{1, 2, \ldots, k\}$, $i \neq j$. Similarly to paths, the length of a cycle is the number of its edges.*

As an example, in Figure 1, v_1 , v_1v_6 , v_6 , v_6v_2 , v_2 , v_2v_1 , v_1 is a cycle of length 3. Notation can again be simplified:

$$v_1v_6v_2$$

means the same cycle.

A graph is called bipartite if its vertices can be organized into two disjoint sets such that all edges have exactly one endpoint in each of the sets. Formally:

Definition 0.12 (Bipartite graph) *Graph $G = (V, E)$ is bipartite if there exist $A, B \subseteq V$ such that $A \cup B = V$, $A \cap B = \emptyset$, and $E \subseteq \{v_av_b \mid v_a \in A, v_b \in B\}$.*

[1] It is not hard to show that the binary relation "there is a path connecting the vertices v and v' in G" is an equivalence relation. Then the components of G are the subgraphs induced by the equivalence classes.

The subsets A and B in this partition of the vertex set are called the ***partite set***s or the ***vertex class***es of G. Note that $G[A]$ and $G[B]$ have no edges. (One an see that the partition is unique apart from possibly switching A and B if G is connected; and not unique if G is disconnected.)

Trees are connected graphs without cycles.

Definition 0.13 (Tree (graph)) *Graph G is a tree if it is connected and contains no cycle as a subgraph.*

The definition implies that there exists exactly one path from any vertex to any other vertex in a tree.

Although trees may seem to look rather simple at first sight, they are very important kinds of graphs. Some combinations of their properties turn out to be equivalent to the definition above.

Proposition 0.3 *For any graph $G = (V, E)$ with at least one vertex, any two of the following three properties imply the third one:*

- *G is connected.*
- *G contains no cycle as a subgraph.*
- $|E| = |V| - 1$.

By selecting one vertex as a so-called ***root***, any tree can be drawn in a tree-like layout:

- The root is placed on the top, composing the top level, often denoted as level 0 itself.
- Then, the neighbors of the root are placed below it, forming level 1.
- The neighbors of the neighbors of the root are placed on level 2 (below level 1) with the exception of the root itself.
- In a similar fashion, the tree is plotted recursively, i.e., the neighbors of the vertices on level i (if any) are placed below them on level $i + 1$ with the exception of their neighbors on level $i - 1$.

The tree together with a selected root (and often together with the corresponding layout) is referred to as a ***rooted tree***.

In Figure 4 a tree subgraph of the graph from Figure 1 is shown with two tree-like layouts by choosing v_6 and v_1 as roots.

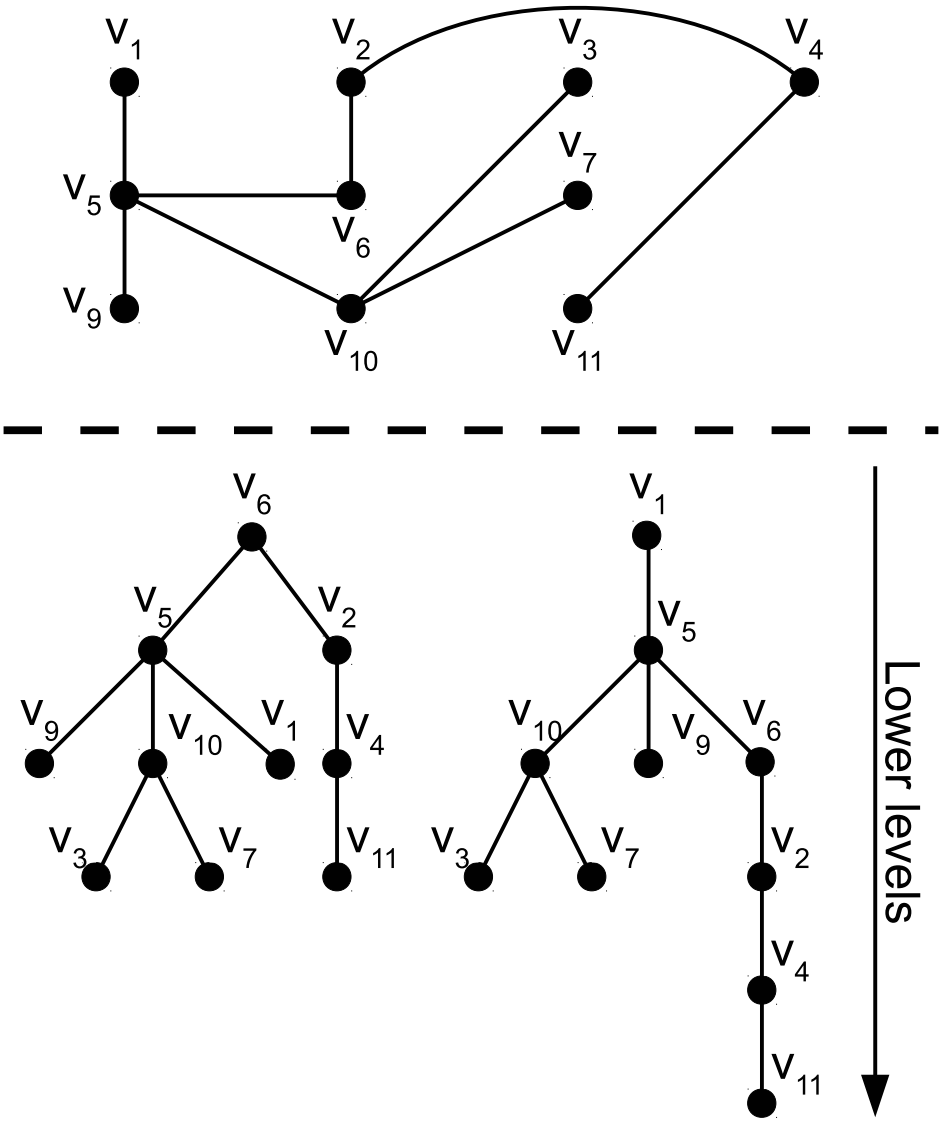


Figure 4: *A tree subgraph of the graph given in Figure 1 and two of its tree-like layouts*

Note that all neighbors of a vertex are placed on the neighbor levels, i.e., there is no edge between vertices whose levels are not consecutive.[2]

Also, except for the root, each vertex v of a level i has exactly one neighbor on the higher neighbor level, and $|N(V)|-1$ neighbors on the lower neighbor level.[3]

To describe algorithms on rooted trees in a simpler manner, the unique neighbor on the higher level is called the ***parent*** of a vertex, while the neighbors in the lower level (if any) are called its ***children***. The vertices sharing the same parent (and thus being on the same level) are often called ***siblings***.[4] A vertex without any children is called ***leaf*** of a rooted tree. Hence, any two leaves are nonadjacent.

We note that the above procedure to draw a tree-like layout, or the definition of the rooted tree does not specify the order of the vertices within a level. Unlike many applications of rooted trees, the algorithms presented in this book do not require and do not depend on the ordering of vertices. In visual representations, however, if a vertex "precedes" another one on the same level, then all of its children precede the children of the other vertex, in order to avoid crossing edges.

Trees are always bipartite. Indeed, fixing a tree-like layout arbitrarily, the even levels can be selected for one partite set and the odd levels for the other. Then definition of bipartite graph is satisfied.

In many applications a special type of rooted trees, so-called ***binary trees*** are used, where the number of children is limited by 2. Note, that the term "binary tree" is a bit misleading, as a tree itself can not be binary, only its tree-like layout. The graph in Figure 4 is a tree, for which the second layout (with root v_1) is binary, and the other (with root v_6) is not. Thus, when the term binary tree is used, it immediately refers not only to a tree, but to a rooted tree.

Trees containing all vertices of a given connected graph are of great importance.

Definition 0.14 (Spanning subgraph; Spanning tree) *A graph H is a spanning subgraph of graph G if H is a subgraph of G and $V(H) = V(G)$. A spanning tree of G is a spanning subgraph which is a tree at the same time.*

[2] In the visual representation of a tree-like layout, in different "branches" of the tree the vertices of the same level may be vertically shifted to different height in order to provide a nicer visual representation, or vertices of different levels may be aligned vertically according to their shared properties from a certain aspect.

[3] The terms 'lower' and 'higher' are interpreted visually, i.e., the levels with higher number are lower, and vice versa.

[4] The vertices on the same level which have different parents are not siblings, and terms like grandparents and cousins are not introduced.

Graphs can also be transformed into other graphs. The complement of a graph is a graph with exactly the same vertices, and with all the edges that are not present in the graph. Formally:

Definition 0.15 (Complement) *The complement (or complementary graph) of G is defined as*

$$\overline{G} = (V(G), \{\{v_1, v_2\} \mid v_1, v_2 \in V(G),\ v_1 \neq v_2,\ \{v_1, v_2\} \notin E(G)\}).$$

In the complement of a graph, the degree of a vertex is the number of vertices which were not adjacent to the vertex in the original graph, therefore we have:

Proposition 0.4 *For any vertex v in any graph G, $d_G(v)+d_{\overline{G}}(v) = |V(G)|-1$.*

The definition also implies

Proposition 0.5 *For every graph G, $|E(G)| + |E(\overline{G})| = \binom{|V(G)|}{2}$ holds.*

One can also see that the complement of the complement is the original graph:

$$\overline{\overline{G}} = G.$$

The line graph of a graph expresses the intersection relation between the edges. In the line graph of G, the vertices correspond to the edges of the original graph, and two of them are connected by an edge if the two corresponding edges share a vertex in the original graph. Formally:

Definition 0.16 (Line graph, $L(G)$) *The line graph of graph G is the graph*

$$L(G) = (E(G), \{e_i e_j \mid e_i, e_j \in E(G), e_i \cap e_j \neq \emptyset, e_i \neq e_j\}).$$

The line graph of the graph of Figure 1 is shown in Figure 5. This line graph has 18 vertices, as the original graph had 18 edges. The number of edges can also be easily calculated from the vertex degrees of original graph. A vertex of degree d in the original graph yields a subgraph with d pairwise adjacent vertices in the line graph, thus:

Proposition 0.6 *For every graph G, $|E(L(G))| = \sum_{v \in V(G)} \binom{d(v)}{2}$.*

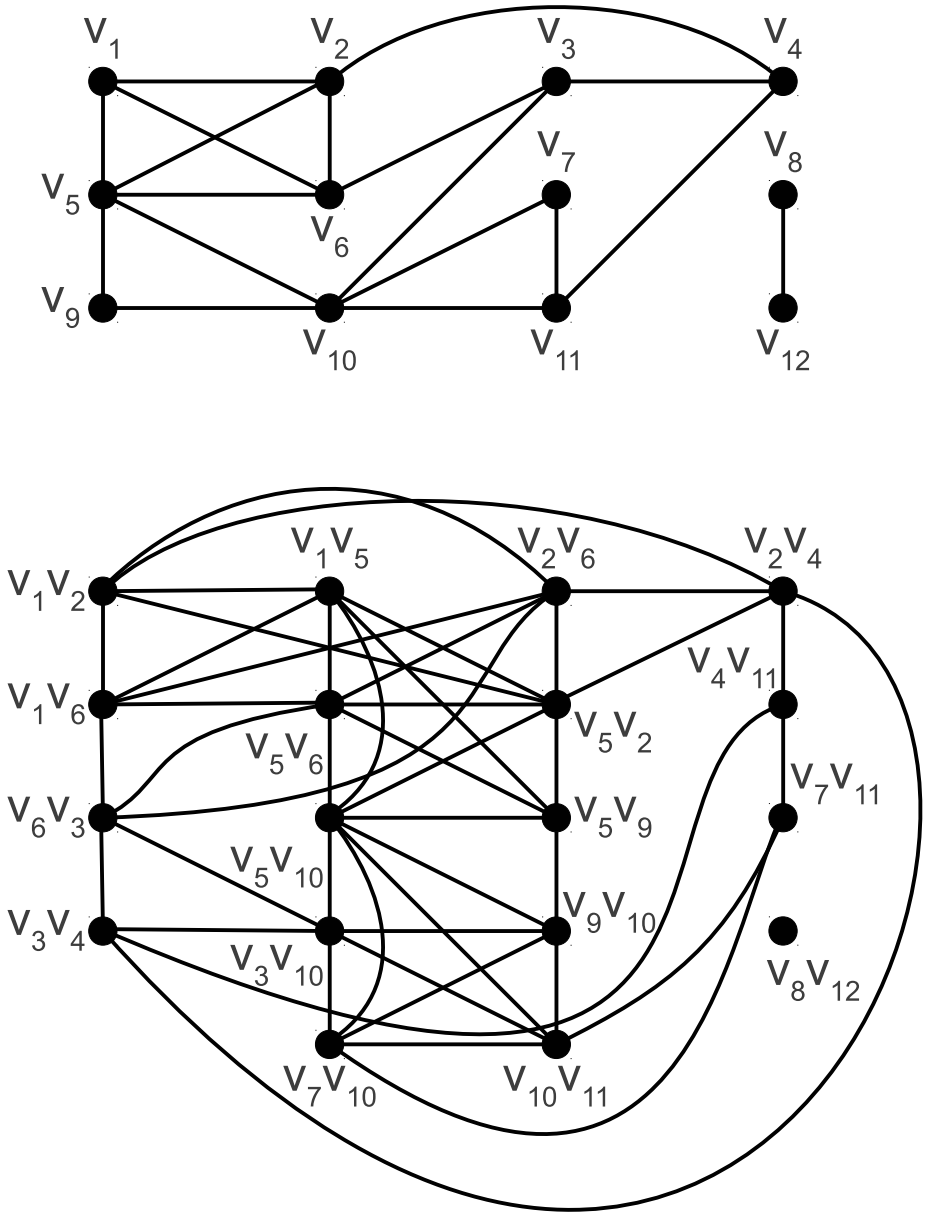


Figure 5: *The graph of Figure 1 with its line graph*

0.2 Special significant graphs

Next, we introduce some particular types of graphs. Later on we shall compute their various parameters, too. At those points we shall refer to these graphs together as 'special graphs'; this is not a standard term, however: we use it just for the sake of simplifying the text.

The empty graph on n vertices is a graph with no edges; it is denoted by E_n. Formally:

Definition 0.17 (Empty graph, E_n)

$$E_n = (\{v_1, v_2, \ldots, v_n\}, \emptyset).$$

The smallest empty graphs are given in Figure 6. Note that in an

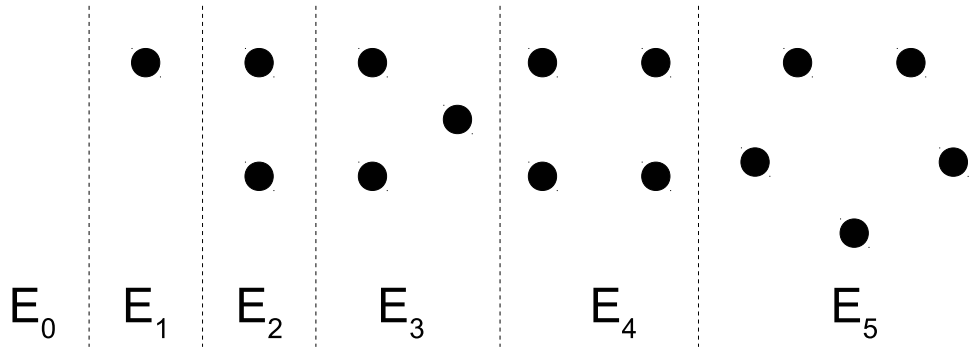


Figure 6: *Smallest empty graphs, also indicating the null graph E_0 with zero vertices and zero edges (which is obviously invisible...)*

empty graph, the degree of each vertex is 0, and the graph has exactly n components. As the empty graph has no edges, its line graph is the null graph. The empty graphs are the 0-regular graphs.

The graph on n vertices containing the edges of a single path of length $n-1$ is called a path graph, and is denoted by P_n. Formally:

Definition 0.18 (Path graph, P_n)

$$P_n = (\{v_1, v_2, \ldots, v_n\}, \{v_i v_{i+1} \mid i \in \{1, 2, \ldots, n-1\}\}).$$

The smallest path graphs are given in Figure 7. Note that every path graph is a tree, hence always bipartite, connected, and two of its vertices have degree 1 (except for P_1), all the other vertices have degree 2. The line graph of P_1 is the null graph, and the line graph of any other path is the path with one fewer vertices:

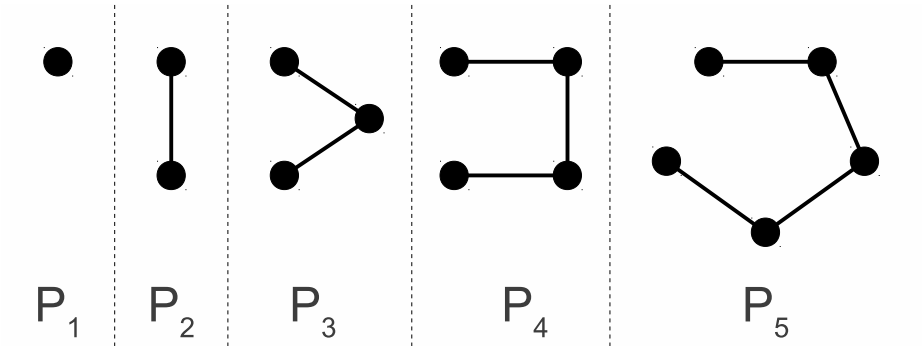


Figure 7: *Smallest path graphs*

Proposition 0.7 *For every $n \geq 2$, we have $L(P_n) = P_{n-1}$.*

Identifying the two ends of the path graph P_{n+1} of length n we obtain a cycle graph, denoted by C_n. Formally:

Definition 0.19 (Cycle graph, C_n) *For $n \geq 3$,*

$$C_n = (\{v_1, v_2, \ldots, v_n\}, \{v_i v_{i+1} \mid i \in \{1, 2, \ldots, n-1\}\} \cup \{v_1 v_n\}).$$

The smallest cycle graphs are given in Figure 8.[5]

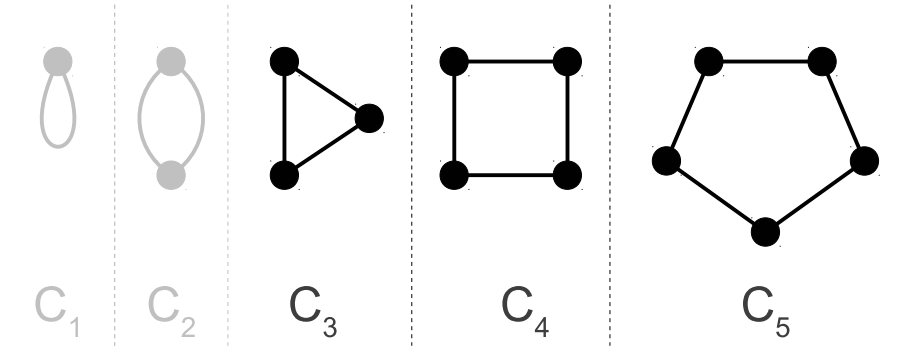


Figure 8: *Smallest cycle graphs*

Note that a cycle graph is always connected, 2-regular (i.e., all of its vertices have degree 2), and it is bipartite if the number of vertices is even. The line graph of a cycle is (isomorphic to) itself:

[5] Note that the cycle graph is not defined for $n = 1, 2$ for simple graphs. But C_1 and C_2 can be defined when loops and parallel edges are introduced, respectively, as shown with gray color in Figure 8.

Proposition 0.8 *For every $n \geq 3$, $L(C_n) = C_n$.*

The graph on n vertices, in which all pairs of vertices are adjacent, is called the complete graph of order n and is denoted by K_n. Formally:

Definition 0.20 (Complete graph, K_n)

$$K_n = (\{v_1, v_2, \ldots, v_n\}, \{v_i v_j \mid 1 \leq i < j \leq n\}).$$

Of course, it does not matter if we write $v_i v_j$ or $v_j v_i$, these two define the same edge. The smallest complete graphs are given in Figure 9. A

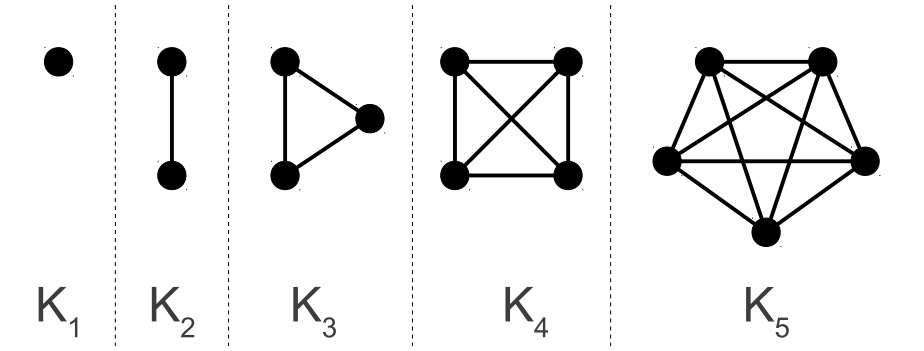


Figure 9: *Smallest complete graphs*

complete graph is always connected and $(n-1)$-regular: all of its vertices have degree $n-1$. Moreover, $K_3 = C_3$, the complement of a complete graph is an empty graph ($\overline{K_n} = E_n$), and all induced subgraphs of a complete graph are complete graphs. The line graph of the complete graph K_n consist of n subgraphs that are K_{n-1}, any two of these subgraphs share exactly one vertex, and their other vertices are connected in pairs, with $n-2$ disjoint edges. The line graph of K_4 is shown in Figure 10.

A bipartite graph with partite sets A and B is called the complete bipartite graph over $|A| + |B|$ vertices if all vertices of A are connected to all vertices of B. Formally:

Definition 0.21 (Complete bipartite graph, $K_{p,q}$)

$$K_{p,q} = (\{a_1, a_2, \ldots, a_p, b_1, b_2 \ldots, b_q\}, \{a_i b_j \mid 1 \leq i \leq p, 1 \leq j \leq q).$$

The smallest complete bipartite graphs are given in Figure 11. A complete bipartite graph with $p, q \geq 1$ is always connected, and the vertex degrees are equal to the size of the other partition class. Moreover, the complement of

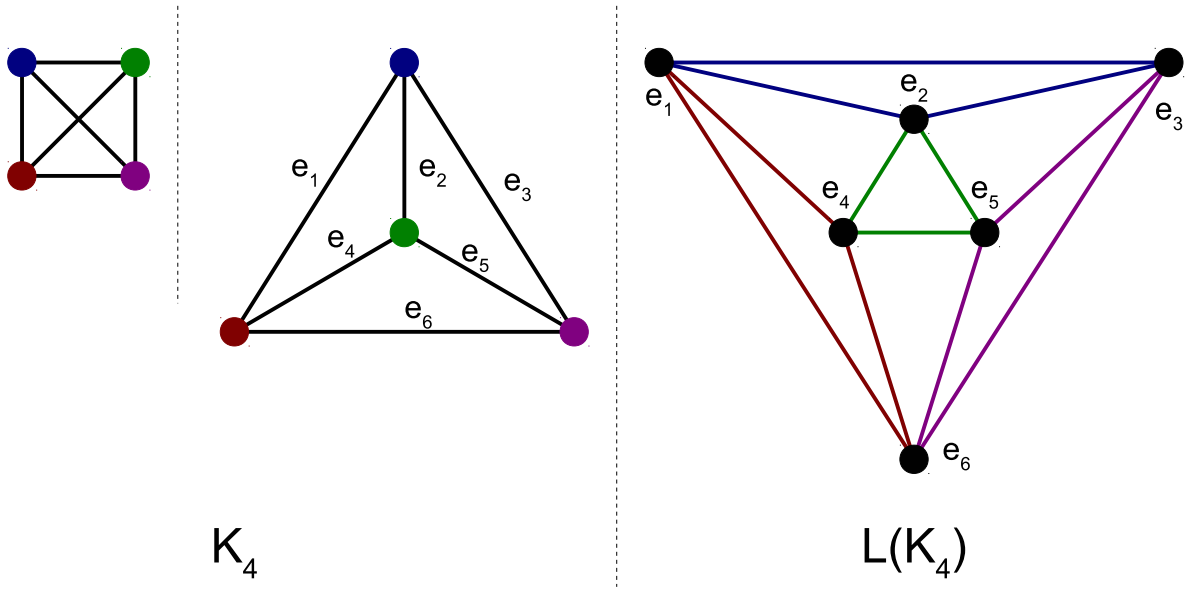


Figure 10: K_4 *and its line graph*

a complete bipartite graph is a graph with two components, each of them being a complete subgraph. The line graph of $K_{p,q}$ has p copies of K_q and q copies of K_p. Those two families of subgraphs are cross-intersecting: each K_q shares exactly one vertex with each K_p Moreover, any two copies of K_p are vertex-disjoint and their vertices are connected in pairs by p disjoint edges; and the structure is similar for any two copies of K_q. See Figure 12 for illustration.

Remark 0.1 *In the particular case of $p = 1$, the graph $K_{1,q}$ is often called a* ***star****.*

0.3 Graph parameters

There are some important parameters of graphs, which play important roles in practical applications.

Minimum and maximum degree

The minimum and maximum degree of a graph is the smallest and the largest value of its vertex degrees. Formally:

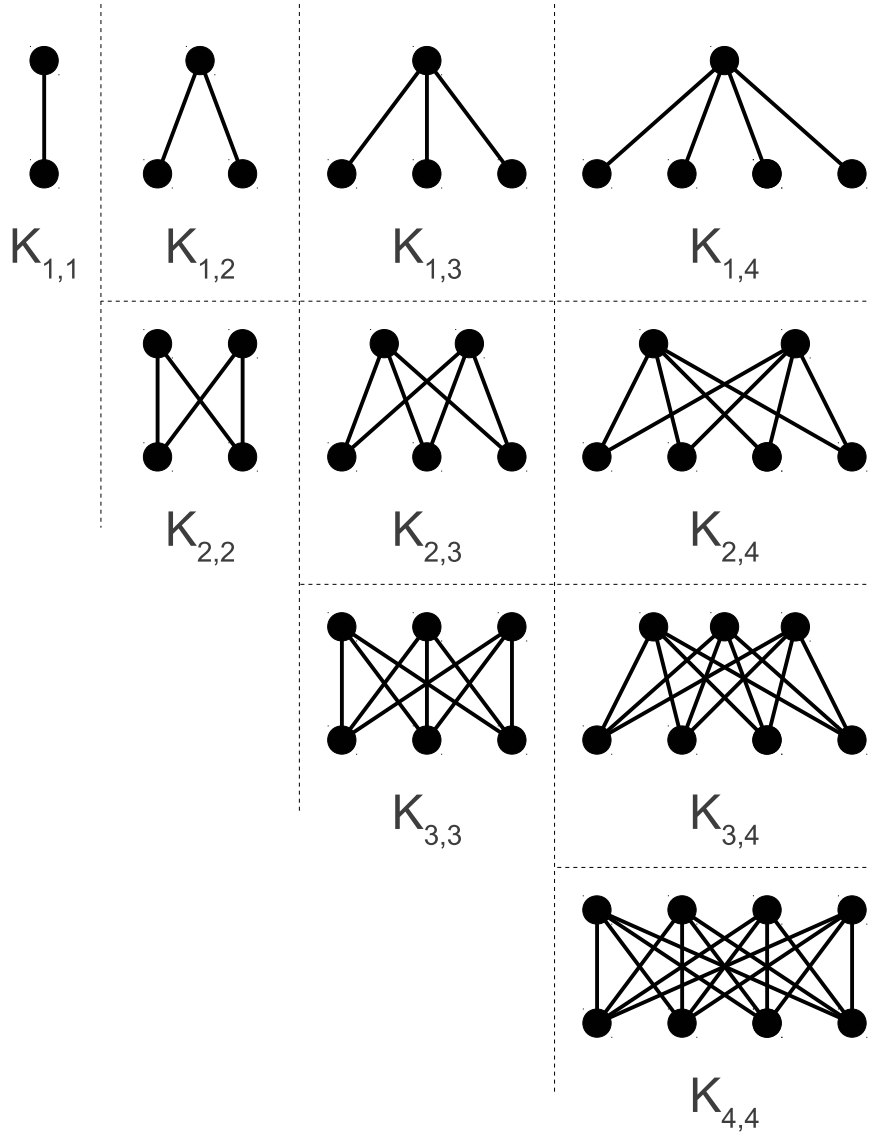


Figure 11: *Smallest complete bipartite graphs*

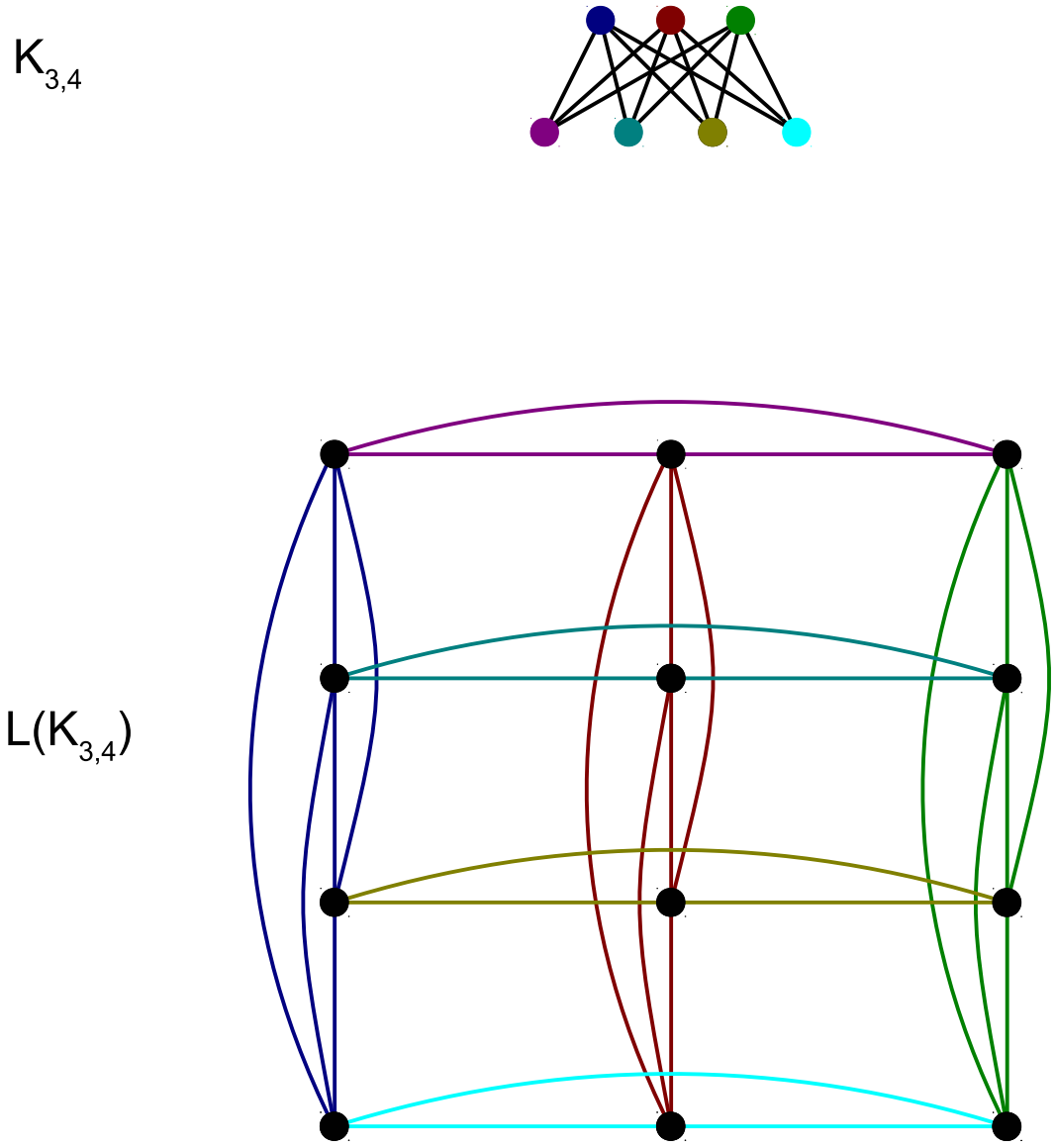


Figure 12: $K_{3,4}$ *and its line graph*

Definition 0.22 (Maximum degree, Δ)

$$\Delta(G) = \max_{v \in V(G)} d(v).$$

Definition 0.23 (Minimum degree, δ)

$$\delta(G) = \min_{v \in V(G)} d(v).$$

Proposition 0.9 *Both the minimum and maximum degree are 0 for all empty graphs and for the graph with one vertex (which is empty and complete at the same time). For the other 'special graphs' we have:*

- $\delta(P_n) = 1$ *and* $\Delta(P_n) = 2$*, for all* $n \geq 3$*; and* $\Delta(P_2) = 1$.
- $\delta(C_n) = \Delta(C_n) = 2$ *for all* $n \geq 3$.
- $\delta(K_n) = \Delta(K_n) = n - 1$ *for all* $n \geq 1$.
- $\delta(K_{p,q}) = \min(p, q)$ *and* $\Delta(K_{p,q}) = \max(p, q)$.

The degree of a vertex v in the complementary graph is the number of vertices not adjacent to v in the original graph. This implies:

Proposition 0.10 *For every graph G with at least one vertex,*

$$\delta(G) + \Delta(\overline{G}) = |V(G)| - 1 = |V(\overline{G})| - 1.$$

0.3.1 Clique number: ω

We use the word 'clique' as a synonym of 'complete subgraph'. (A more restrictive meaning also occurs in the literature of graph theory.)

Definition 0.24 (Clique) *An induced subgraph $G[V']$ is called a clique in G if it is a complete graph.*

In the example from Figure 1, the induced subgraphs $G[\{v_1, v_2, v_5, v_6\}]$ and $G[\{v_7, v_{10}, v_{11}\}]$ are cliques. Also, a single edge with its two endpoints is a clique, and so is a single vertex, too.

The number of vertices of largest cliques in a graph is denoted by ω and is termed the clique number:

Definition 0.25 (Clique number, ω)

$$\omega(G) = \max_{G[V'] \text{ is a clique in } G} |V'|.$$

Proposition 0.11 *The clique number of E_n and of $P_1 = K_1$ is 1, as there is not even a single edge in these graphs. For the other 'special graphs' we have:*

- *$\omega(P_n) = \omega(K_{p,q}) = 2$ (with $n \geq 2$ and $p, q \geq 1$), and also $\omega(C_n) = 2$ for $n \geq 4$, attained by selecting a single edge from the graph;*
- *$\omega(K_n) = n$ (including the case $\omega(C_3) = 3$, too), attained by selecting all vertices of the graph.*

As the number of edges in a clique of size k is $k \cdot (k-1)/2$, the clique number is bounded from above in terms of the number of edges in the graph:

Proposition 0.12 $\omega(G) \leq \sqrt{2 \cdot |E(G)| + \frac{1}{4}} + \frac{1}{2}$.

Also, the degrees of the vertices in the largest clique are at least $\omega(G)-1$, thus

Proposition 0.13 $\omega(G) \leq \Delta(G) + 1$.

The edges incident with the same vertex in a graph induce a clique in the line graph, thus

Proposition 0.14 $\omega(L(G)) \geq \Delta(G)$.

Equality does not hold for some graphs; e.g., the 3-cycle has $L(C_3) = C_3$, $\omega(L(C_3)) = 3$, and $\Delta(C_3) = 2$. However, for graphs with $\Delta(G) \geq 3$, equality must hold:

Proposition 0.15 *If $\Delta(G) \geq 3$, then $\omega(L(G)) = \Delta(G)$.*

0.3.2 Clique covering number: θ

The vertices of a graph can always be partitioned into (disjoint) subsets such that the subgraphs induced by these sets are complete. This is called a clique covering.

Definition 0.26 (Clique covering) *The induced subgraphs $G[V_1], G[V_2], \ldots, G[V_k]$ form a clique covering of G if $\bigcup_{i=1}^{k} V_i = V(G)$ and $G[V_i]$ is a clique for all $i = 1, \ldots, k$.*

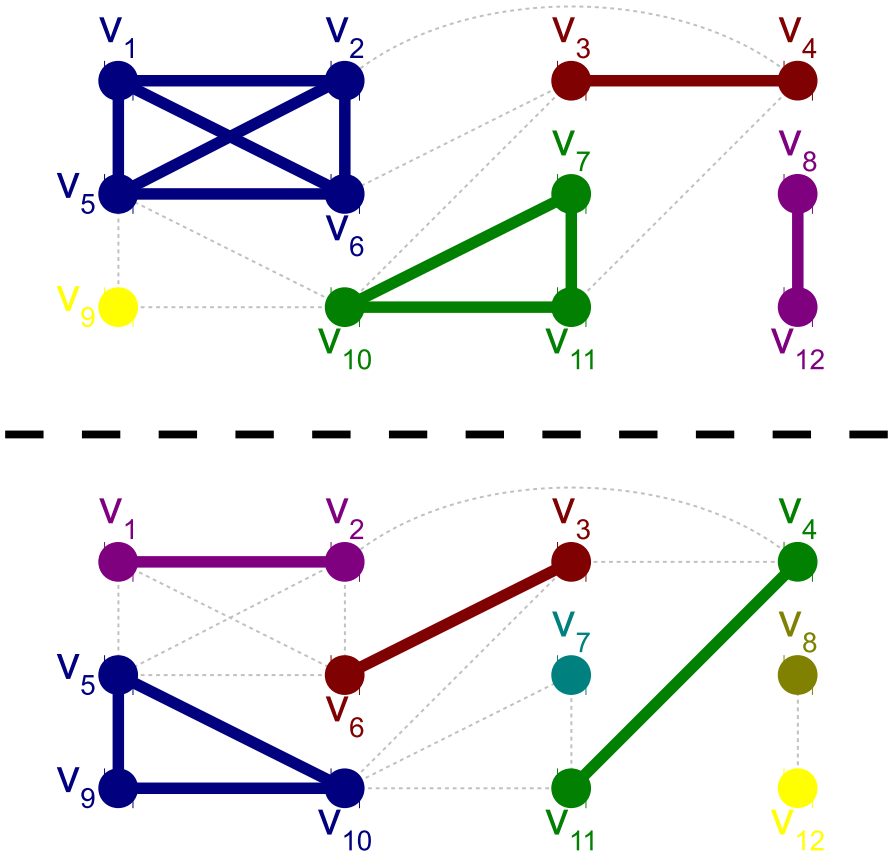


Figure 13: *Some clique coverings for the graph in Figure 1*

Two examples of clique coverings of the graph in Figure 1 are given in Figure 13.

The coverings in Figure 13 use 5 and 7 cliques, respectively. An obvious way that can be done in every graph is to take one-vertex subgraphs. But using larger subgraphs we can have a vertex partition into fewer cliques, which is more preferable. The minimum number of cliques needed is called clique covering number:

Definition 0.27 (Clique covering number, θ)

$$\theta(G) = \min_{G[V_1],\ldots,G[V_k] \text{ is a clique covering in } G} k.$$

The first covering in Figure 13 is minimal. We note that the minimal clique covering of a graph is not necessarily unique.

Proposition 0.16 *The clique covering number of the 'special graphs' and an example minimal cover is:*

- $\theta(E_n) = n$, *by covering each vertex with a different clique of size one.*
- $\theta(P_n) = \theta(C_n) = \left\lceil \frac{n}{2} \right\rceil$, *by covering with* $G[\{v_1, v_2\}]$, $G[\{v_3, v_4\}]$, $\ldots$, $G[\{v_{2\cdot\left\lfloor \frac{n}{2} \right\rfloor - 1}, v_{2\cdot\left\lfloor \frac{n}{2} \right\rfloor}\}]$, *plus the single vertex* $G[\{v_n\}]$ *in case if the number of vertices is odd; the only exception is* C_3, *where* $\theta(C_3) = 1$ *as the graph is a clique itself.*
- $\theta(K_n) = 1$, *by selecting the whole graph as one clique.*
- $\theta(K_{p,q}) = \max(p, q)$, *by selecting cliques* $G[\{a_1, b_1\}]$, $G[\{a_2, b_2\}]$, $\ldots$, $G[\{a_{\min(p,q)}, b_{\min(p,q)}\}]$, *and covering the rest of the vertices with* $\max(p, q) - \min(p, q)$ *single vertices.*

The clique covering numbers of the 'special graphs' are illustrated in Figure 14 for both even and odd numbers of vertices.

A function of the clique number provides a lower bound on the clique covering number:

Proposition 0.17 $\theta(G) \geq \left\lceil \frac{|V(G)|}{\omega(G)} \right\rceil$.

If we take the set of edges incident with an arbitrarily chosen vertex, the vertices representing those edges in the line graph induce a clique. Each edge of the original graph has two vertices, therefore taking those stars for all but one vertices, already a clique covering of the line graph is obtained. This yields the following rough upper bound:

Proposition 0.18 $\theta(L(G)) \leq |V(G)| - 1$.

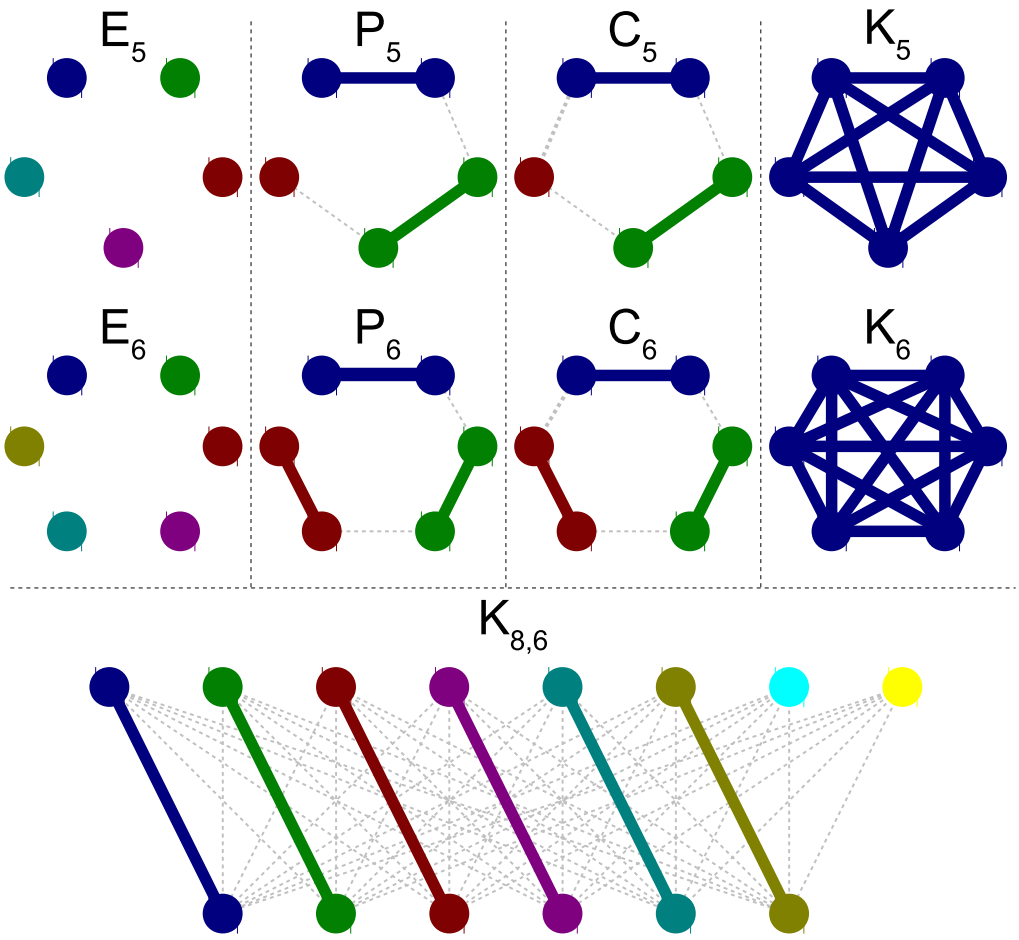


Figure 14: *Clique covering number of special graphs*

0.3.3 Independence number: α

A set of vertices in a graph is independent if no two vertices are connected in the graph. Formally:

Definition 0.28 (Independent vertex set) *A set $V' \subseteq V(G)$ of vertices is independent if $\nexists v_1, v_2 \in V' : \{v_1, v_2\} \in E$; or alternatively if $\forall v_1, v_2 \in V' : \{v_1, v_2\} \notin E$.*

In the graph of Figure 1 the following vertex sets are independent for example: $\{v_1, v_{10}, v_8\}$, $\{v_2, v_3, v_9, v_{11}\}$. Also, the emptyset is independent, every single vertex is independent, and each pair of nonadjacent vertices is independent in every graph. The independence number is the size of the largest independent set, i.e., maximum number of vertices in a graph, such that no two of them are adjacent. Formally:

Definition 0.29 (Independence number, α)

$$\alpha(G) = \max_{V' \text{ is an independent set in } G} |V'|.$$

The independence number of the graph in Figure 1 is 5. The largest independent vertex set is not necessarily unique, Figure 15 shows two of them.

Proposition 0.19 *For the 'special graphs' the independence numbers and an example largest independent vertex sets are:*

- $\alpha(E_n) = n$, *as all of the vertices are independent.*
- $\alpha(P_n) = \lceil \frac{n}{2} \rceil$, *by selecting* $v_1, v_3, \ldots, v_{2\lceil \frac{n}{2} \rceil - 1}$.
- $\alpha(C_n) = \lfloor \frac{n}{2} \rfloor$, *by selecting* $v_1, v_3, \ldots, v_{2\lfloor \frac{n}{2} \rfloor - 1}$.
- $\alpha(K_n) = 1$ *by selecting any of the vertices.*
- $\alpha(K_{p,q}) = \max(p, q)$.

The largest independent vertex sets of the 'special graphs' are illustrated in Figure 16 for both even and odd numbers of vertices.

It is easy to see that the independence number of a disconnected graph is the sum of the independence numbers of its components. Moreover, the independence number is at least 1 for each component, thus, the independence number is at least as large as the number of components. Also, at most one vertex can be selected into an independent set from any clique, thus

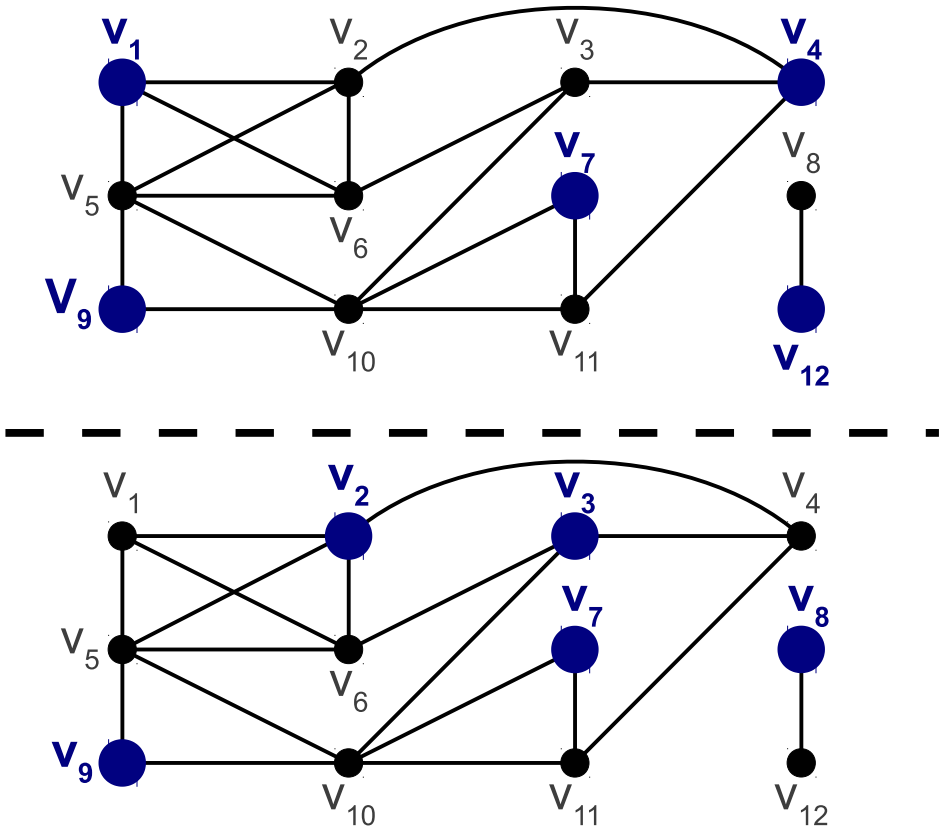


Figure 15: *Largest independent vertex sets for the graph in Figure 1*

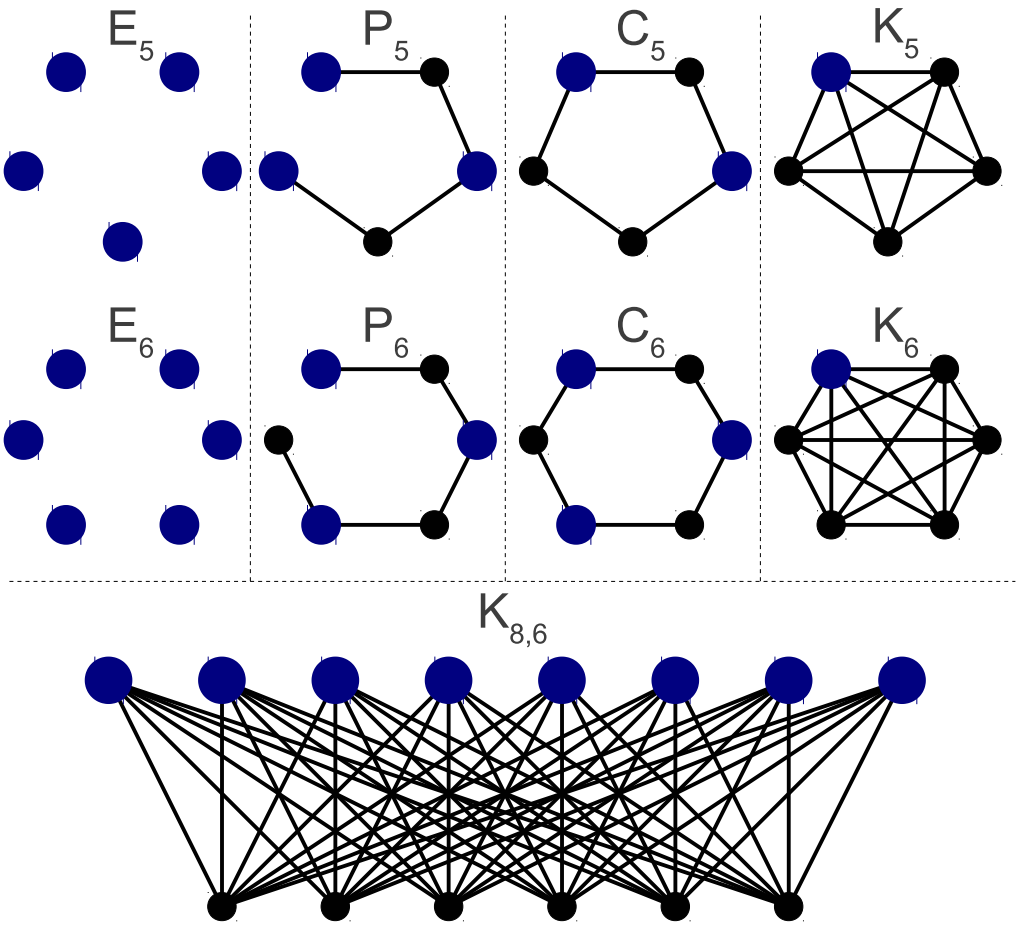


Figure 16: *Some largest independent vertex sets of special graphs*

Proposition 0.20 $\alpha(G) \leq \theta(G)$.

Since at most one vertex can be selected into any independent set from the largest clique, we also have:

Proposition 0.21 $\alpha(G) + \omega(G) \leq |V(G)| + 1$.

An independent vertex set in a graph G is a clique in its complement $\overline{G}$, and vice versa. Thus:

Proposition 0.22 $\alpha(G) = \omega(\overline{G})$.

0.3.4 Transversal number: τ

A set of vertices is called a transversal if all edges of the graph are incident with at least one of them. Formally:

Definition 0.30 (Transversal) *A set $V' \subseteq V(G)$ is a transversal of G if $e \cap V' \neq \emptyset$ holds for all edges $e \in E(G)$.*

In Figure 17 two different transversals are shown for the graph of Figure 1.

The transversal number is the smallest size of a transversal set, i.e., minimum number of vertices in a graph, with which all of the edges can be "covered". Formally:

Definition 0.31 (Transversal number, τ)

$$\tau(G) = \min_{V' \text{ is a transversal set in } G} |V'|$$

The smallest transversal is not necessarily unique, Figure 17 shows actually two minimum transversals for the graph of Figure 1.

Proposition 0.23 *The transversal numbers and example smallest transversals of the 'special graphs' are:*

- $\tau(E_n) = 0$, *as the graph has no edges.*
- $\tau(P_n) = \left\lfloor \frac{n}{2} \right\rfloor$, *by selecting* $v_2, v_4, \ldots, v_{2 \cdot \left\lfloor \frac{n}{2} \right\rfloor}$.
- $\tau(C_n) = \left\lceil \frac{n}{2} \right\rceil$, *by selecting* $v_2, v_4, \ldots, v_{2 \cdot \left\lfloor \frac{n}{2} \right\rfloor}$, *and* v_n *in case of* n *odd.*
- $\tau(K_n) = n - 1$, *by selecting all the vertices except one, e.g.,* $v_1, v_2, \ldots, v_{n-1}$.

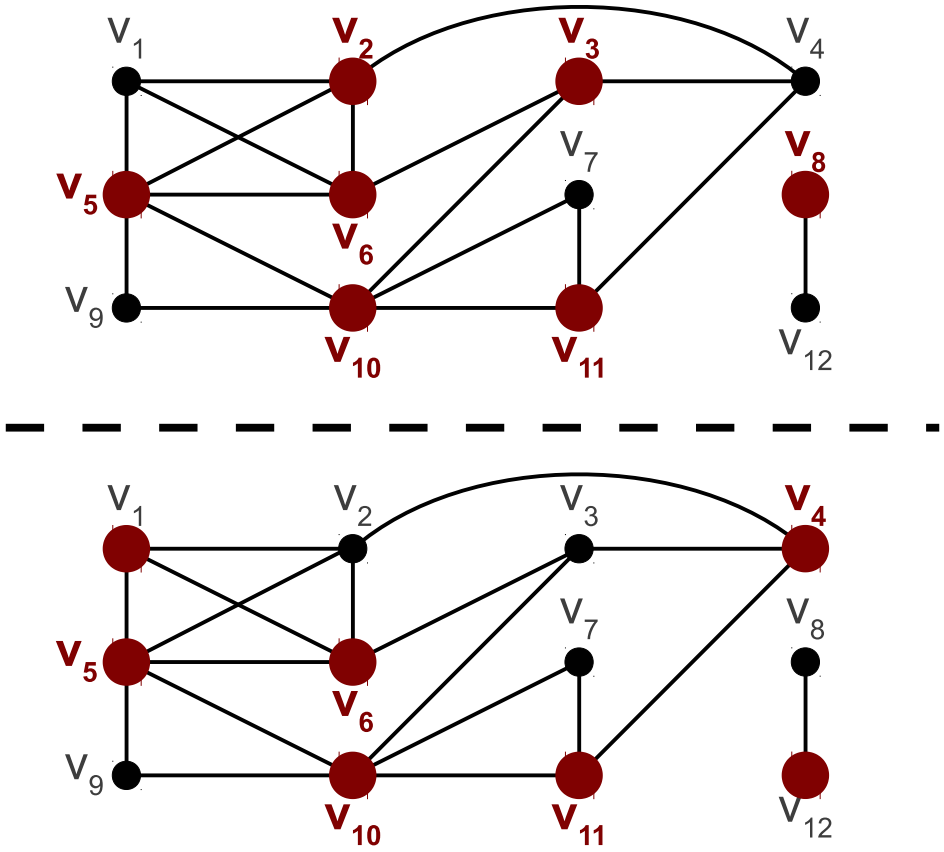


Figure 17: *Transversals for the graph in Figure 1*

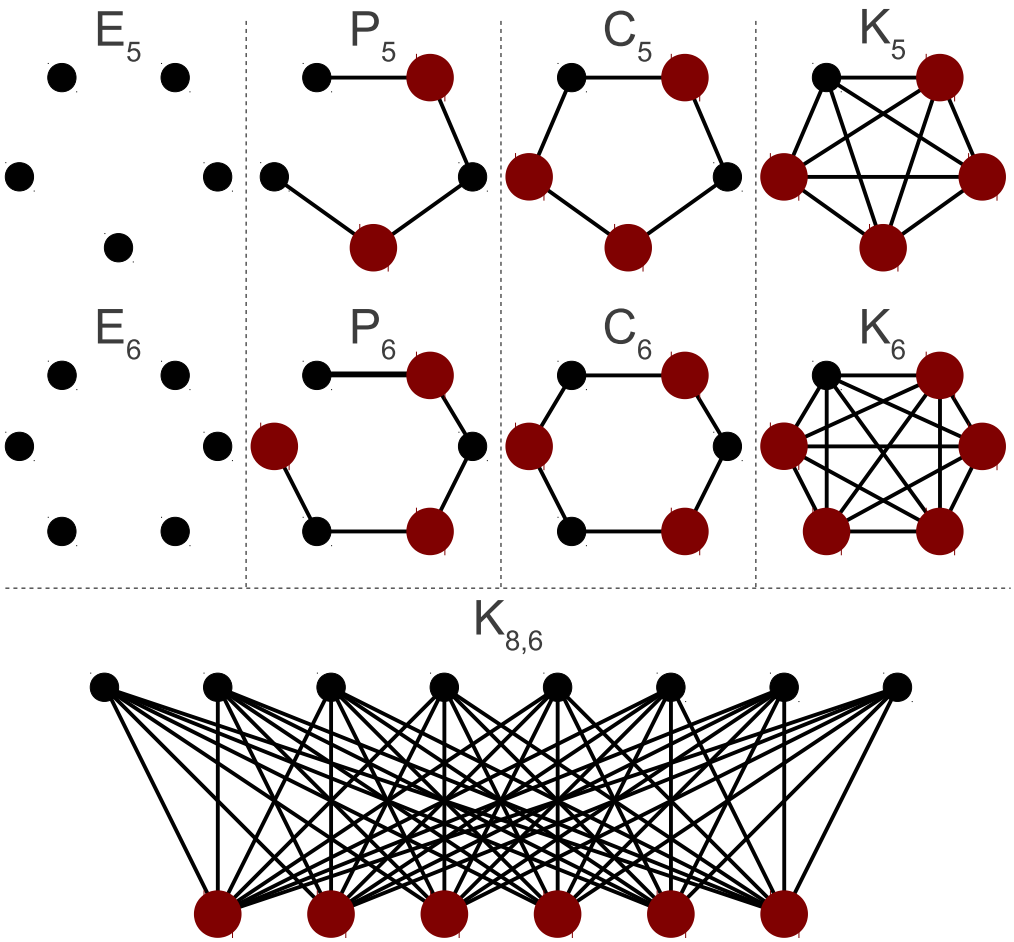


Figure 18: *Smallest transversals of special graphs*

- $\tau(K_{p,q}) = \min(p, q)$, *by selecting all the vertices in the smaller vertex class.*

The smallest transversals of the 'special graphs' are illustrated in Figure 18 for both even and odd numbers of vertices.

Observe that the selected transversal vertices in Figure 18 are exactly the ones that were not selected for the independent vertex sets in Figure 16. The relation between the transversals and independent vertex sets of the examples in Figures 17 and 15 is the same. These examples suggest the following result:

Theorem 0.1 $\tau(G) + \alpha(G) = |V(G)|$.

The assertion follows from two facts. First, the vertices not included in a transversal must be independent, otherwise the transversal would not cover all the edges. This holds for the smallest transversal as well, thus there is an independent vertex set with $|V(G)| - \tau(G)$ vertices, giving a lower bound on the independence number, i.e., $\alpha(G) \geq |V(G)| - \tau(G)$. Similarly, the vertices not included in an independent vertex set must form a transversal, otherwise there would be an edge between two vertices of the independent vertex set. This holds for the largest independent vertex set as well, thus there is a transversal with $|V(G)| - \alpha(G)$ vertices, giving an upper bound on the transversal number, i.e., $\tau(G) \leq |V(G)| - \alpha(G)$. Combining these two inequalities, the theorem is proved.

0.3.5 Chromatic number: χ

Similarly to clique covering, the vertices of a graph can be covered with independent vertex sets as well. However, this type of covering of the vertices is often considered from a different aspect: proper vertex coloring. To begin with something more general, a vertex coloring of a graph is an assignment of a single color to each vertex. Formally:

Definition 0.32 (Vertex coloring) *A mapping $\varphi : V \to \{1, 2, \ldots, k\}$ is called a (vertex-) coloring with k colors, or simply a k-coloring of (the vertices of) G.*

The assigned numbers $(1, 2, \ldots, k)$ are referred to as colors, and graphically often are represented by coloring the vertices of the graph. Figure 19 shows three different colorings of the graph given in Figure 1. The colorings use 4, 3, and 6 colors, respectively.

After the vertices of a graph have been colored, an edge has either the same color on both of its endpoints, or two different colors. A coloring that does not result in any monochromatic edges is called proper. Formally:

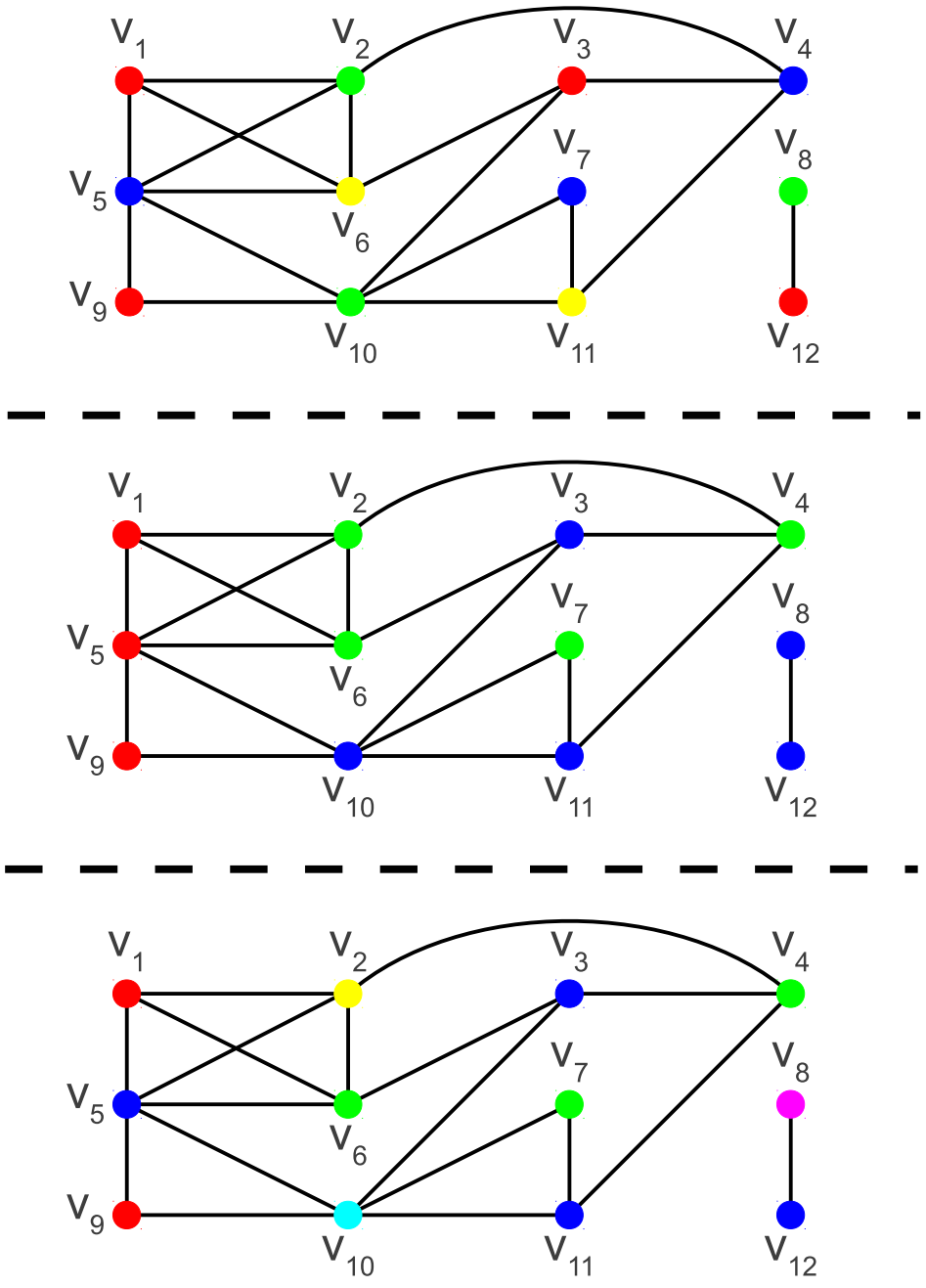


Figure 19: *Three different colorings of the example in Figure 1*

Definition 0.33 (Proper vertex coloring) *A coloring φ is called a proper coloring of G if $\nexists\{v_1, v_2\} \in E(G)$ such that $\varphi(v_1) = \varphi(v_2)$. In other words, $\{v_1, v_2\} \in E(G)$ implies $\varphi(v_1) \neq \varphi(v_2)$.*

In Figure 19 the second coloring is not proper as v_1v_5, v_2v_4, $v_{10}v_{11}$, and also some further edges are monochromatic. Note that there exists at least one proper coloring for any graph: namely, the coloring that assigns different colors to all of the vertices satisfies the condition of being proper.

In this book improper vertex colorings will not be considered, therefore the term 'vertex coloring' itself will be applied for proper vertex colorings; and the term proper/improper will only be used when this property of a coloring needs to be emphasized.

The set of nodes sharing the same assigned color is called a ***color class***. In a proper coloring two vertices in the same color class cannot be connected by an edge, as it would be monochromatic; thus, the color classes are independent vertex sets. Consequently, a proper coloring unambiguously defines a partition of the vertex set into independent sets. Similarly, a proper vertex coloring can easily be constructed from an independent vertex set covering, thus the two notions are the same from a different point of view.

Similarly to the clique covering number, the minimum number of independent sets covering the vertex set, or equivalently the minimum number of colors in a proper vertex coloring is an important parameter of a graph, called chromatic number:

Definition 0.34 (Chromatic number)

$$\chi(G) = \min_{G \text{ has a proper } k\text{-coloring}} k.$$

In Figure 19 the first coloring is minimal, i.e., the chromatic number of the graph in Figure 1 is 4.

Proposition 0.24 *The chromatic numbers of the 'special graphs' and example colorings with minimal number of colors are:*

- $\chi(E_n) = 1$*, as all the vertices can be colored with the same single color.*
- $\chi(P_n) = 2$*, by coloring the odd and even vertices with two different colors; the exception is P_1 where $\chi(P_1) = 1$.*
- $\chi(C_n) = \begin{cases} 2 & \text{if } n \text{ is even,} \\ 3 & \text{if } n \text{ is odd,} \end{cases}$
  by coloring the odd and even vertices with two different colors except for the last vertex if n is odd, which should be assigned to a third color.

- $\chi(K_n) = n$, *as all the vertices need different colors.*
- $\chi(K_{p,q}) = 2$, *by selecting the partite sets as color classes.*

Minimal vertex colorings of the 'special graphs' are illustrated in Figure 20 for both even and odd numbers of vertices.

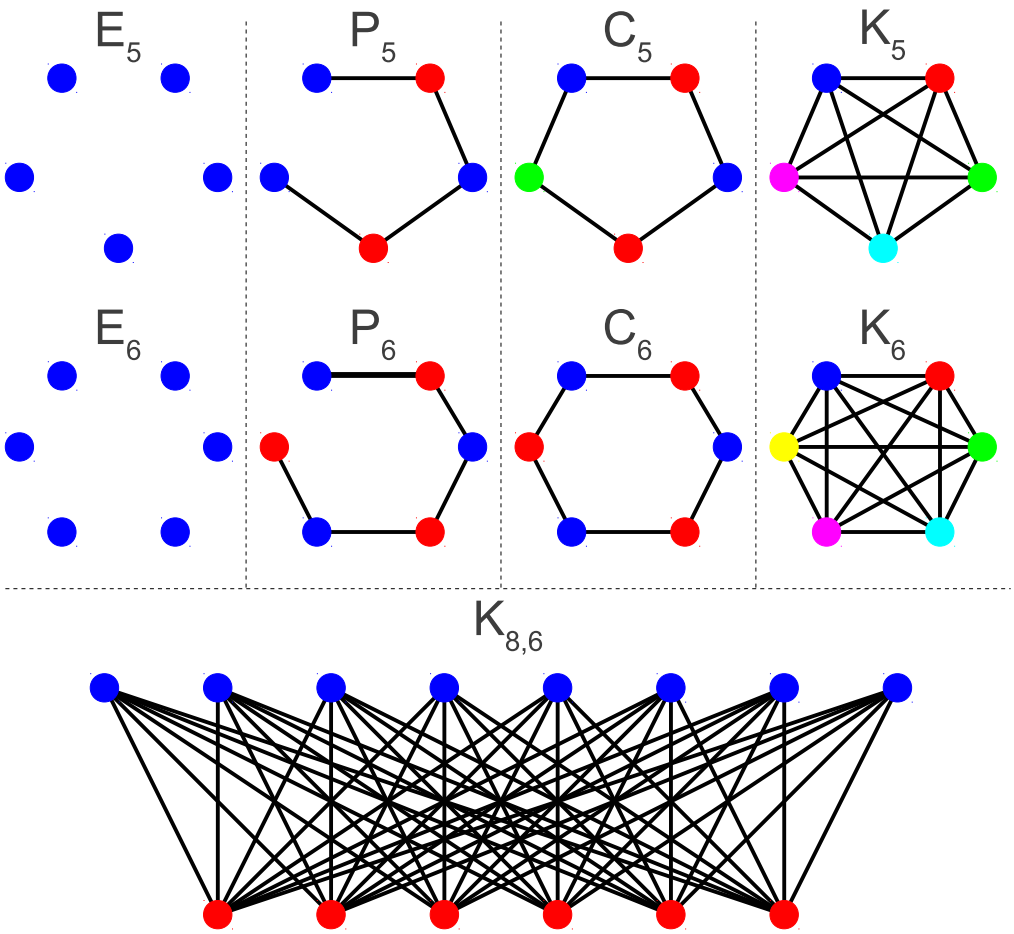


Figure 20: *Minimal vertex colorings of special graphs*

Note that not only the complete bipartite graphs, but every bipartite graph can be colored with exactly two colors, as the two partite sets of the graph are independent.

It is also easy to see that for any subgraph $G' \subseteq G$, the inequality $\chi(G) \geq \chi(G')$ must hold, as a proper coloring of G is a proper coloring of G' as well.

Based on this simple observation, and on the fact that a clique is a complete graph whose vertices need pairwise different colors, the following lower bound on the chromatic number could be derived:

Proposition 0.25 $\chi(G) \geq \omega(G)$.

Also, as $\alpha(G)$ provides an upper bound on the size of the color classes, it also provides a lower bound on the number of necessary colors:

Proposition 0.26 $\chi(G) \geq \left\lceil \frac{V(G)}{\alpha(G)} \right\rceil$.

As an independent vertex set is a clique in the complementary graph, the chromatic number of a graph must be equal to the clique covering number of its complement:

Proposition 0.27 $\chi(G) = \theta(\overline{G})$.

0.3.6 Matching number: ν

A set of pairwise vertex-disjoint edges in a graph is called matching. Formally:

Definition 0.35 (Matching) *A set $E' \subseteq E(G)$ of edges is a matching if $e_1 \cap e_2 = \emptyset$ holds for all $e_1, e_2 \in E'$, $e_1 \neq e_2$.*

Some matchings of the graph in Figure 1 are shown in Figure 21. These matchings include 5, 4, and 6 edges, respectively. The second matching can easily be extended with two more edges (v_3v_4 and v_9v_{10}); the first one, however, cannot be extended with a single edge to have size 6.

The maximum size of matchings in a graph is called the matching number:

Definition 0.36 (Matching number, ν)

$$\nu(G) = \max_{E' \text{ is a matching in } G} |E'|.$$

The third matching in Figure 21 is maximum, as it covers all of the vertices. This observation results in a simple upper bound that the maximum size of a matching cannot exceed the half of the number of vertices, otherwise two edges would definitely share a vertex:

Proposition 0.28 $\nu(G) \leq \left\lfloor \frac{|V(G)|}{2} \right\rfloor$.

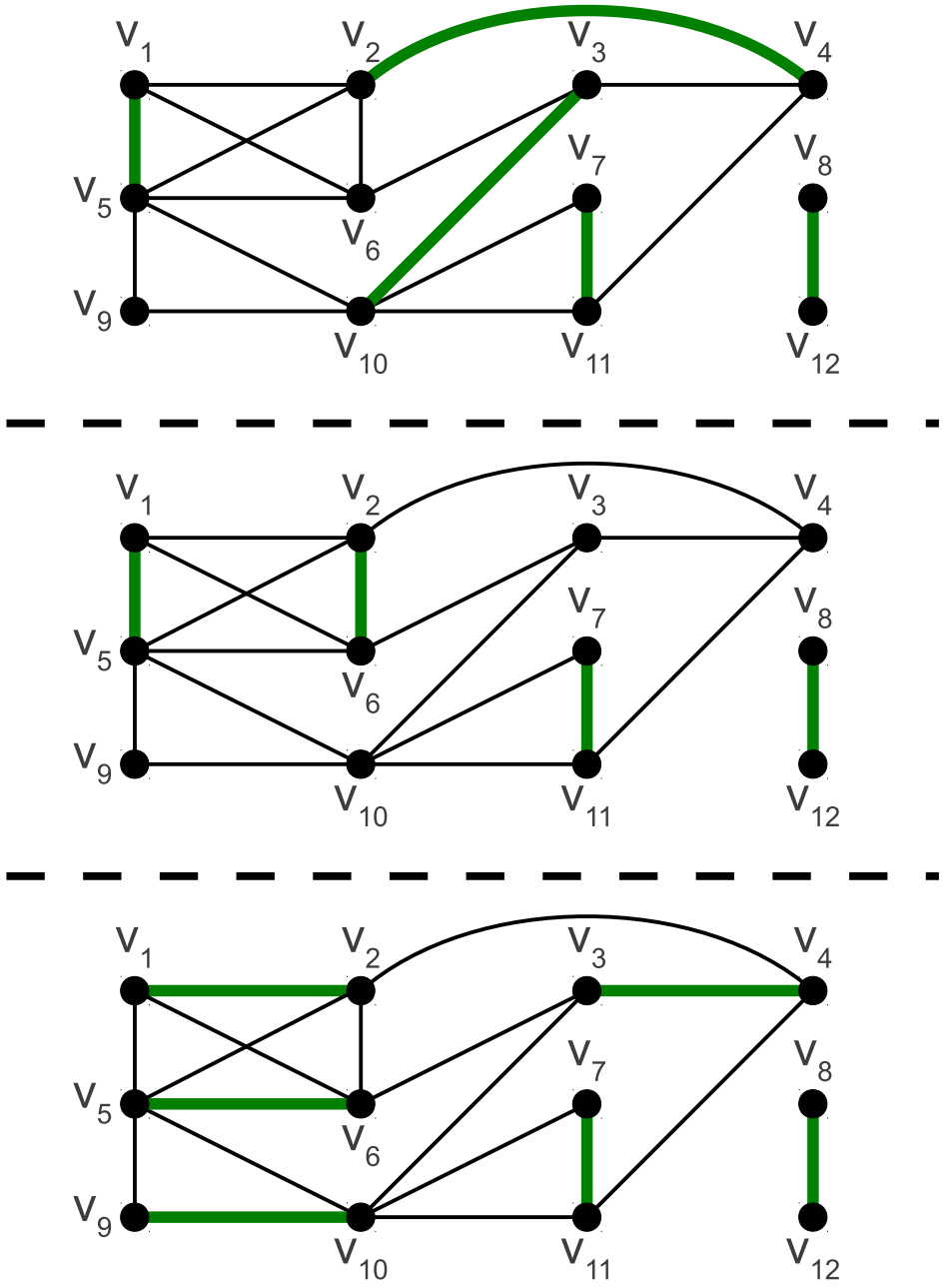


Figure 21: *Three different matchings in the graph of Figure 1*

Proposition 0.29 *The matching numbers of the 'special graphs' and examples for maximal matchings are:*

- $\nu(E_n) = 0$, *as the graph has no edges.*
- $\nu(P_n) = \nu(C_n) = \nu(K_n) = \left\lfloor \frac{n}{2} \right\rfloor$ *by selecting* v_1v_2, v_3v_4, $\ldots$, $v_{2\cdot\left\lfloor \frac{n}{2} \right\rfloor - 1}v_{2\cdot\left\lfloor \frac{n}{2} \right\rfloor}$.
- $\nu(K_{p,q}) = \min(p, q)$, *by selecting* a_1b_1, a_2b_2, $\ldots$, $a_{\min(p,q)}b_{\min(p,q)}$.

Maximum matchings of the 'special graphs' are illustrated in Figure 22 for both even and odd numbers of vertices.

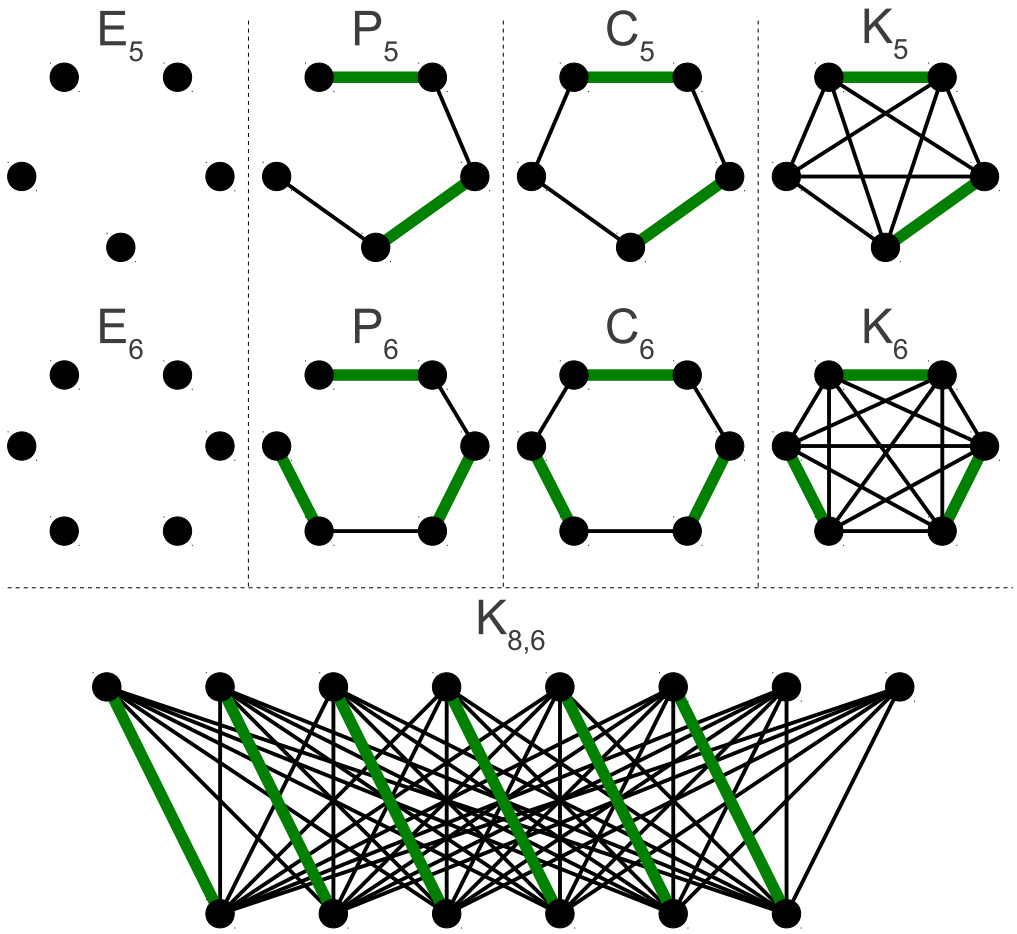


Figure 22: *Maximum matchings of special graphs*

A transversal covers all the edges, thus, every edge in a matching is incident with at least one vertex in a transversal. Since matching edges are

vertex-disjoint, it follows that the size of a matching cannot exceed the size of a transversal:

Proposition 0.30 $\nu(G) \leq \tau(G)$.

A matching in a graph corresponds to an independent vertex set of the same cardinality in the line graph, and vice versa. Thus:

Proposition 0.31 $\alpha(L(G)) = \nu(G)$.

0.3.7 Chromatic index: χ'

Similarly to the coloring of the vertices, the edges can also be colored:

Definition 0.37 (Edge coloring) *A mapping $\phi : E \to \{1, 2, \ldots, k\}$ is called an edge coloring with k colors, or a k-coloring of the edges.*

Analogously to vertex coloring, an edge coloring is proper if two adjacent edges do not share the same color:

Definition 0.38 (Proper edge coloring) *An edge coloring ϕ is called a proper edge coloring of G if, for all $e_1, e_2 \in E(G)$, $e_1 \neq e_2$ with $e_1 \cap e_2 \neq \emptyset$ we have $\phi(e_1) \neq \phi(e_2)$.*

As in the case of vertex coloring, the term edge coloring will mostly be used for proper edge colorings, and the words proper/improper will only be used to emphasize this property. Two proper edge colorings of the graph in Figure 1 are shown in Figure 23, using 6 and 5 colors, respectively.

The minimum number of colors in a proper edge coloring is called chromatic index (should not be confused with chromatic number, that is the minimum for proper vertex coloring):

Definition 0.39 (Chromatic index)

$$\chi'(G) = \min_{G \text{ has a proper edge coloring with } k \text{ colors}} k.$$

Proposition 0.32 *The chromatic index of the 'special graphs' and example minimal edge colorings are:*

- $\chi'(E_n) = 0$, *as the graph has no edges.*
- $\chi'(P_n) = 2$, *with color classes* $\{v_1v_2,\ v_3v_4,\ \ldots,\ v_{2\cdot\lfloor\frac{n}{2}\rfloor-1}v_{2\cdot\lfloor\frac{n}{2}\rfloor}\}$ *and* $\{v_2v_3,\ v_4,v_5,\ \ldots,\ v_{2\cdot\lceil\frac{n}{2}\rceil-2}v_{2\cdot\lceil\frac{n}{2}\rceil-1}\}$*; the exceptions are* $\chi'(P_2) = 1$ *and* $\chi'(P_1) = 0$.

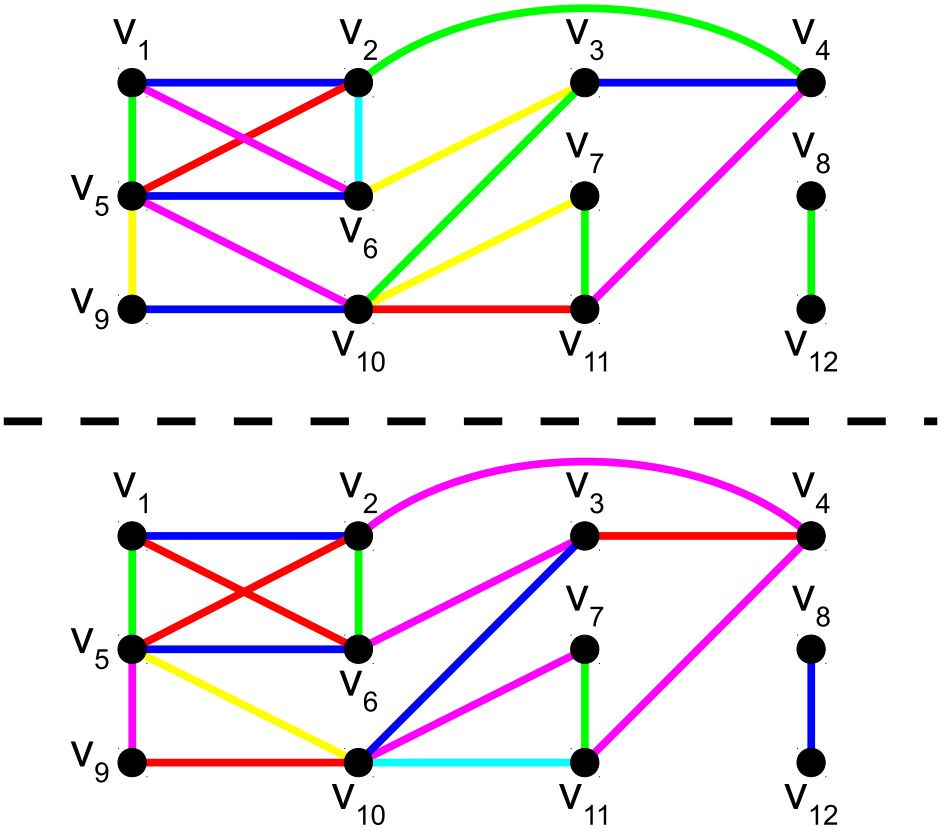


Figure 23: *Two different (proper) edge colorings of the graph given in Figure 1*

- $\chi'(C_n) = \begin{cases} 2 & \text{if } n \text{ is even,} \\ 3 & \text{if } n \text{ is odd,} \end{cases}$
  with color classes $\{v_1v_2,\ v_3v_4,\ \ldots,\ v_{2\cdot\lfloor\frac{n}{2}\rfloor-1}v_{2\cdot\lfloor\frac{n}{2}\rfloor}\}$ *and* $\{v_2v_3,\ v_4,v_5,\ \ldots,\ v_{2\cdot\lfloor\frac{n}{2}\rfloor-2}v_{2\cdot\lfloor\frac{n}{2}\rfloor-1}\}$, *and an additional color class* $\{v_nv_1\}$ *in case of* n *odd. (In fact, there are many proper 3-colorings.)*

- $\chi'(K_n) = \begin{cases} n-1 & \text{if } n \text{ is even,} \\ n & \text{if } n \text{ is odd,} \end{cases}$
  except that $\chi'(K_1) = 0$. *For* n *even, a proper edge coloring of* K_n *with* $n-1$ *colors will be constructed in Section 8.1. From that, one can construct an (edge)* n*-coloring of* K_n *for* n *odd by taking an* n*-coloring of* K_{n+1} *(then* $n+1$ *is even) and by deleting one vertex.*[6]

- $\chi'(K_{p,q}) = \max(p,q)$, *by having color classes* $\{a_ib_{(i+k \mod q)+1} \mid i = 1,2,\ldots,p\}$ *for each* $k = 0,1,\ldots,q$ *if* $q \geq p$; *the* $q < p$ *case is analogous.*

Edge colorings of the 'special graphs' with minimum number of colors are illustrated in Figure 24 for both even and odd numbers of vertices.

The color classes of a proper edge coloring are independent edge sets, i.e. matchings, thus connection between the matchings and edge colorings is the same as between the independent vertex sets and the vertex colorings: an edge coloring is a partition of the edge set into matchings.

As a consequence, we obtain the following lower bound:

Proposition 0.33 $\chi'(G) \geq \left\lceil \frac{|E(G)|}{\nu(G)} \right\rceil$.

The edges adjacent to a vertex all must have distinct colors, thus:

Proposition 0.34 $\chi'(G) \geq \Delta(G)$.

Because of this, the first example in Figure 23 is a minimal coloring, as it uses 5 colors, and the degree of some of the vertices is 5.

As a matching corresponds to an independent vertex set in the line graph and vice versa, a proper edge coloring of a graph is equivalent to a proper vertex coloring of its line graph, thus:

Proposition 0.35 $\chi(L(G)) = \chi'(G)$.

[6] Fewer colors are not enough. Indeed, we have to color $\frac{n(n-1)}{2}$ edges, and a color class can contain at most $\frac{n}{2}$ edges if n is even, and at most $\frac{n-1}{2}$ edges if n is odd.

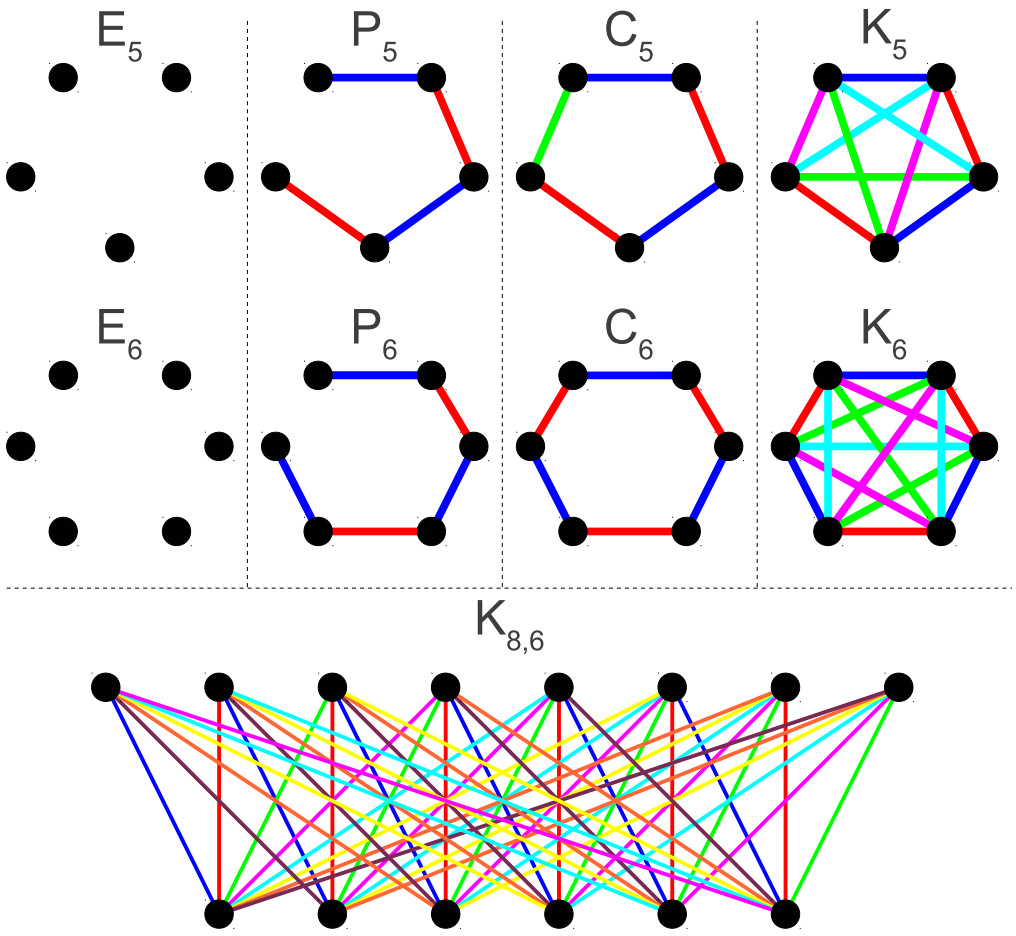


Figure 24: *Edge colorings with minimum number of colors for special graphs*

0.4 Graph extensions

Graphs have many kinds of extensions. This section introduces the basics for two of them, that will be needed in later chapters.

0.4.1 Directed graphs

A directed graph or digraph is defined similarly as its undirected counterpart:

Definition 0.40 (Directed graph (Digraph)) *A directed graph or digraph is a pair $D = (V, A)$, where*

- *V is the set of vertices,*
- *A is the set of arcs, which are ordered pairs of vertices; i.e., $A \subseteq V \times V$.*

Thus, compared to edges of a simple graph which are unordered vertex pairs, the arcs in a digraph are ordered vertex pairs. Note that the definition also allows loops (arcs starting and ending at the same vertex, i.e. (v_i, v_i)). Moreover, although parallel arcs (more than one arc from some v_i to a v_j) are not allowed, oppositely oriented arcs (v_i, v_j) and (v_j, v_i) may occur at the same time, and are considered to be distinct.

The following notation will also be used for $D = (V, A)$:

- $V(D)$: set of vertices of D, thus V;
- $A(D)$: set of arcs of D, thus A.

The vertices of a digraph will be denoted in the same way as in the undirected case. The arcs will be denoted by $a_1, a_2, \ldots$ or by the pairs of their endpoints, like $vv', v_1v_2, \ldots$. Note that in case of a directed graph, the order of vertices in an arc is not arbitrary, thus v_iv_j does not denote the same arc as v_jv_i.

An example of a digraph with 6 vertices and 11 arcs is given in Figure 25, where the arcs are represented by continuous lines (straight lines and curves) with arrows indicating the direction of the arc: for arc v_iv_j the arrow is directed from its 'tail' v_i to its 'head' v_j.

In this digraph, there is a loop on v_4 and there is an arc in each direction between v_5 and v_8.

Subdigraphs and induced subdigraphs are defined analogously as in the case of undirected graphs:

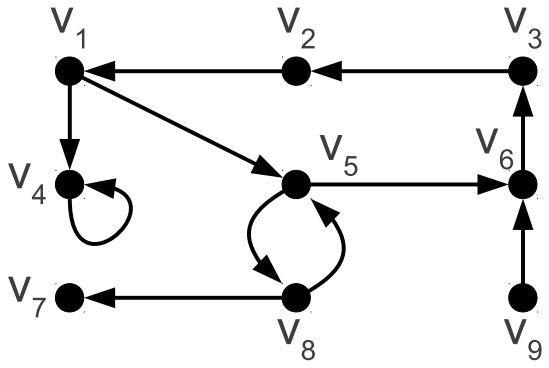


Figure 25: *A directed graph*

Definition 0.41 (Subdigraph) *A directed graph $D' = (V', A')$ is called a subdigraph of $D = (V, A)$ — or simply its subgraph, if this term causes no ambiguity in the context — if D' is a digraph, moreover $V' \subseteq V$ and $A' \subseteq A$. This relation is denoted by $D' \subseteq D$.*

Definition 0.42 (Induced subdigraph) *A directed graph $D' = (V', A')$ is an induced sub(di)graph of $D = (V, A)$ if $V' \subseteq V$, and $A' = A \cap V' \times V'$. The subdigraph induced by V' in D is denoted by $D[V']$.*

Neighbors are defined similarly as in the undirected case; based on the direction of the arcs, however, we can distinguish between in- and out-neighbors:

Definition 0.43 (Neighbors of a vertex in digraphs)

- $N^-(v) = \{v' \in V(D) \mid v'v \in A(D)\}$: *in-neighborhood;*
- $N^+(v) = \{v' \in V(D) \mid vv' \in A(D)\}$: *out-neighborhood;*
- $N(v) = N^-(v) \cup N^+(v)$: *neighborhood.*

In Figure 25, the in-neighbours of v_4 are $\{v_1, v_4\}$, and the out-neighbours of v_5 are $\{v_6, v_8\}$.

Similarly, the vertices in a digraph have in-degree, out-degree and degree:

Definition 0.44 (Degree, In-degree, Out-degree)

- $d^-(v) = |N^-(v)|$: *in-degree;*
- $d^+(v) = |N^+(v)|$: *out-degree;*
- $d(v) = d^+(v) + d^-(v) = |N^-(v)| + |N^+(v)|$: *degree.*

In the graph of Figure 25 we have $d^+(v_1) = 2$, $d^-(v_2) = 1$, $d(v_8) = d(v_4) = 3$.

A directed path is an alternating sequence of vertices and arcs such that, for each arc, the preceding vertex is its tail and the successor vertex is its head. Formally:

Definition 0.45 (Directed path) *A sequence $v_0, a_1, v_1, \ldots, a_k, v_k$ is a directed path from v_0 to v_k in D if the vertices $v_0, v_1, \ldots, v_k$ are all distinct, and if $k \geq 1$ then $a_i = (v_{i-1}, v_i)$ and $a_i \in A(D)$ for all $i \in \{1, 2, \ldots, k\}$. The vertex v_0 alone is also considered to be a path; this does not require (although allows) the presence of a loop at v_0.*

In Figure 25, there is a directed path from v_8 to v_4 through v_5, v_6, v_3, v_2, and v_1, however there is no directed path from v_4 to v_8. This suggest the definition of strong connectivity.

Definition 0.46 (Strongly connected digraph) *A digraph D is strongly connected if there exists a directed path from every vertex v to every other vertex v' ($v, v' \in V(D)$).*

Directed cycles are defined similarly to undirected ones:

Definition 0.47 (Directed cycle) *A sequence $v_0, a_1, v_1, \ldots, a_k, v_k$ is a directed cycle in D if $v_0 = v_k$, $a_i = (v_{i-1}, v_i)$ and $a_i \in A(D)$ for all $i \in \{1, 2, \ldots, k\}$, and $v_i \neq v_j$ for any $i, j \in \{1, 2, \ldots, k\}, i \neq j$).*

In Figure 25 there is a directed cycle of length 5 on v_1, v_5, v_6, v_3, and v_2; and the loop on v_4 is a cycle of length 1. Also, the two arcs between v_5 and v_8 form a cycle of length 2.

If it is obvious from the context, the terms path and cycle will be used without emphasizing their directed nature. It has to be clear in any case, however, that 'directed path/cycle' always means that the edges are oriented consecutively (and cyclically in case of a cycle).

A digraph can always be converted to an undirected graph by removing the direction of the arcs:

Definition 0.48 (Underlying undirected graph) *The underlying undirected graph of a loopless digraph D is obtained by omitting the orientations:*

$$G(D) = (V(D), \{vw \mid (v, w) \in A(D)\})$$

where we keep just one edge joining v and w if multiple edges occurred.

A graph constructed in this way will not have parallel edges for arcs v_iv_j and v_jv_i. Therefore, D can be obtained by suitably orienting the edges of $G(D)$ exactly when D has no cycles of length two. And if D had some loops, they would remain there after the removal of orientations, hence $G(D)$ would not be a simple graph. (Loops could be avoided by modifying the definition with the further condition $v \neq v'$, but we do not want to impose this.)

Definition 0.49 (Orientation, Oriented graph) *Given a graph $G = (V, E)$, an orientation $\vec{G}$ of G is obtained by making each edge $vv' \in E$ an arc in one direction. A digraph D is called an oriented graph if it is an orientation of a simple undirected graph. This is exactly when D contains no cycles shorter than 3.*

If D is an orientation of G, then G is the underlying undirected graph of D. Bipartite oriented graphs and oriented trees can be defined as oriented graphs whose undirected underlying graph is a bipartite graph or a tree, respectively. In case of rooted trees, however, it is often required that all arcs should be oriented from the root toward the leaves, or all of them be oriented from the leaves toward the root.

The directed path graph and directed cycle graph can be defined analogously as in the undirected case.

0.4.2 Set systems

Definition 0.50 (Set system; Underlying set) *A set system is a set $\mathcal{S}$, in which each member $S \in \mathcal{S}$ is a nonempty[7] set. If $\mathcal{S}$ is given in the form $\mathcal{S} = \{S_1, \ldots, S_m\}$, then we allow members $S_i, S_j \in \mathcal{S}$ of different indices $i \neq j$ to be the same set of elements. (This is in analogy with multiple edges in graphs which are not simple.) Any set containing the union of the members, i.e., any $X \supseteq \bigcup_{S\in\mathcal{S}} S$ can be viewed as an underlying set of the system. If the underlying set is not specified, we may assume that $X = \bigcup_{S\in\mathcal{S}} S$.*

[7] Sometimes the empty set is not forbidden to be a member of a set system; but our discussion needs to exclude this case.

Throughout, we will consider only finite set systems, which are the ones containing only a finite number of members. In notation, we usually refer to the system as $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$, and the elements of the underlying set X are usually denoted by $x, x', x_1, x_2, \ldots$. A system $\mathcal{S}' \subseteq \mathcal{S}$ will be called a ***subsystem*** of $\mathcal{S}$.

A set system with 3 members is given in Figure 26. The elements of the underlying set are denoted by dots, while members of the system are represented by curvy areas.

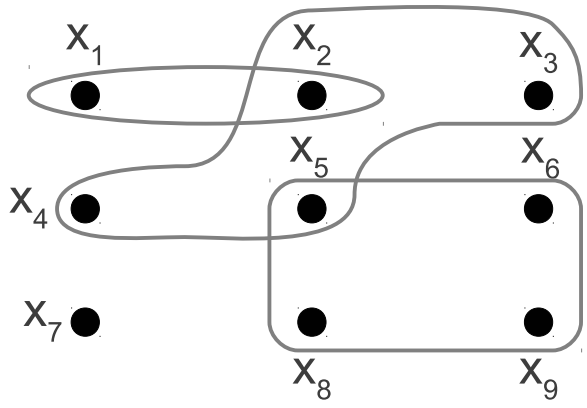


Figure 26: *A set system*

Definition 0.51 (Uniform set system) *A set system $\mathcal{S}$ is k-uniform if every member $S \in \mathcal{S}$ is of the same cardinality k.*

Hence, the system in Figure 26 is not uniform as it contains a member of size 2, and one of size 4 as well.

A simple undirected graph G (without isolated vertices) can be considered as a 2-uniform set system, whose members are the edges of G and whose underlying set corresponds to the vertex set $V(G)$. Therefore, set systems are generalizations of graphs.[8] Due to this, most of our definitions given for graphs can be extended to set systems analogously. Here we mention only some of them which will be used in later chapters.

Definition 0.52 (Degree; Minimum/Maximum degree) *The degree of an element $x \in X$ is the number of members $S \in \mathcal{S}$ containing it. Formally,*

$$d(x) = |\{S \in \mathcal{S} \mid x \in S\}|.$$

[8] With another terminology not used in this book, set systems are called hypergraphs; they have vertices and so-called hyperedges.

The minimum and maximum degree of the set system $\mathcal{S}$ is the minimum and maximum of the degrees of elements in the underlying set, respectively. The minimum degree is denoted by $\delta(\mathcal{S})$ whilst the maximum degree is denoted by $\Delta(\mathcal{S})$.

Definition 0.53 (Matching; Matching number) *For a set system $\mathcal{S}$, a subsystem $\mathcal{M} \subseteq \mathcal{S}$ is a matching if no two members of $\mathcal{M}'$ share an element; i.e., for any two distinct $S_i, S_j \in \mathcal{M}'$, $S_i \cap S_j = \emptyset$ holds. The matching number $\nu(\mathcal{S})$ is the maximum cardinality of a matching in $\mathcal{S}$.*

Definition 0.54 (Independent set; Independence number) *For a set system $\mathcal{S}$, a subset I of its underlying set is independent if I contains no member $S \in \mathcal{S}$ entirely. The maximum cardinality of an independent set of $\mathcal{S}$ is called independence number and is denoted by $\alpha(\mathcal{S})$.*

Definition 0.55 (Transversal; Transversal number) *For a set system $\mathcal{S}$, a subset T of its underlying set is called a transversal if T contains at least one element from each member $S \in \mathcal{S}$; that is, if $T \cap S \neq \emptyset$ holds for all $S \in \mathcal{S}$. The minimum cardinality of a transversal of $\mathcal{S}$ is called transversal number and is denoted by $\tau(\mathcal{S})$.*

In some parts of the literature, the terms *vertex cover* and *hitting set* are also used for transversal. In this way it is customary to say that an element *covers* the sets which contain it.[9]

It is clear by the definitions above that the complement of an independent set is necessarily a transversal and vice versa. Consequently, for the maximum and minimum cardinalities

$$\alpha(\mathcal{S}) + \tau(\mathcal{S}) = |X|$$

necessarily holds. Further parameters for set systems, relations between them, and connections between the parameters of graphs and set systems will be discussed in later chapters.

0.5 Complexity of algorithms

An algorithmic problem is given with a problem instance (input data) and a task to be solved. For example, a decision problem asks whether the

[9] This meaning of 'covering' differs from everyday usage. Nevertheless, it becomes logical not only in the context of set systems and hypergraph theory but also in connection with the duality principle between points and lines of projective planes, which we shall see in Chapter 9.

problem instance satisfies a certain property (e.g., whether $\chi(G) \leq 3$); a search problem requires to find a structure in the instance with a specified property (e.g., find a proper vertex coloring of G with three colors if such a coloring exists); an optimization problem asks for the minimum or maximum value of a function on the problem instance (e.g., determine $\chi(G)$); etc.

There is a mathematical theory of computing, which also deals with the complexity of computational problems and that of the algorithms solving them. It is not the subject of the present lecture notes to give an introduction to it, but we would like to mention at least some basic details.

One measure of algorithms is their worst-case behavior. This compares the number of steps with the size of the given instance; more precisely one asks what is the largest number of steps the algorithm performs on problem instances of size n. Loosely speaking, if a graph $G = (V, E)$ is given in the input, it counts with $|V| + |E|$ in size, while the contribution of a positive integer to input size is considered to be the number of its digits (e.g., when written as a binary number).

Restricting ourselves to algorithms which perform one elementary computational operation in each time step, an algorithm (and its running time) is said to be bounded by a function f if, for all possible problem instances, the amount of steps needed is not greater than $f(n)$, where n is the size of the instance.

In theory and in practice, especially, algorithms that are bounded by polynomial functions are of special interest, as the computational need to solve larger problems does not increase 'drastically', and remains in a manageable level.

In the text we shall use the informal phrase 'efficient algorithm' for those algorithms which are bounded by polynomial functions, i.e., they are guaranteed to terminate with a solution in polynomial time.

Similarly, problems for which there exists an efficient algorithm are called 'efficiently solvable'. Note that a problem can usually be solved by several algorithms, which may differ in complexity. Thus, an 'efficiently solvable' problem admits at least one fast solution (and it is irrelevant that many 'non-efficient algorithms' can also solve the same problem).

The phrase 'algorithmically hard' will be used for the problems for which no efficient algorithms are known so far and for which the tools of the theory of computing have proved strong indication that those problems are indeed harder than the efficiently solvable ones. Note that the absence of efficient algorithms is not proven for all of these problems at the current state of science. A certain subset of them still may turn out to be efficiently solvable; in fact, if one of them is, it implies the same for the others as well.

For readers familiar with the theory of computing, the problems being

efficiently solvable, algorithmically hard, and the mentioned subset of the latter (i.e., still being open whether they are efficiently solvable or not) refer to P, NP-hard, and NP-complete problems, respectively.

Chapter 1

Interval systems

A finite closed interval $[a, b]$ is defined on the real line as

$$[a, b] = \{x : a \leq x \leq b\} \subset \mathbb{R}$$

where the real numbers a and b are called the left and the right ends or endpoints of the interval, respectively. As a convention, the left and right ends of an interval I_i will be denoted by a_i and b_i. An ***interval system*** $\mathcal{I}$ is a set system whose members are finite closed intervals. We will consider finite systems, that are the ones containing only a finite number of intervals.

1.1 Helly's theorem

The classical theorem of Helly concerns set systems $\mathcal{F}$ whose members are bounded, closed, convex sets from the d-dimensional Euclidean space $\mathbb{R}^d$. Helly's theorem states that if any $d + 1$ members of $\mathcal{F}$ have a nonempty intersection, then the whole system has a nonempty intersection: $\bigcap_{F \in \mathcal{F}} F \neq \emptyset$. The real line is the 1-dimensional Euclidean space, and its convex sets are the intervals. Hence, Helly's theorem[1] with $d = 1$ gives:

Theorem 1.1 (Helly property for intervals) *If any two intervals from the interval system $\mathcal{I}$ share a point, then there exists a point contained in all intervals from $\mathcal{I}$.*

Proof: Let $L = a_i$ be the rightmost left endpoint and $R = b_j$ be the leftmost right endpoint over the intervals of $\mathcal{I}$. If $i = j$—and also when $i \neq j$ but the intervals I_i and I_j coincide—each point of the interval I_i is a common point

[1] The assertion is valid for infinitely many bounded, closed intervals, too; here we give a proof for finite systems only.

of all intervals. Otherwise, I_i and I_j are different intervals, but they have a nonempty intersection due to our assumption. This implies $L = a_i \leq b_j = R$ and by the choice of L and R, $a_k \leq L \leq R \leq b_k$ must hold for every interval I_k from the system. Then, every point of $[L, R]$ belongs to all intervals from $\mathcal{I}$. □

Figure 1.1 gives an illustration for the proof. Remark that the Helly property with $d = 1$ does not hold in a dimension higher than 1. Indeed, just consider a (non-degenerate) triangle ABC in the 2-dimensional plane and observe that the three sides AB, BC and CA are pairwise intersecting but there is no point contained in all the three sides.

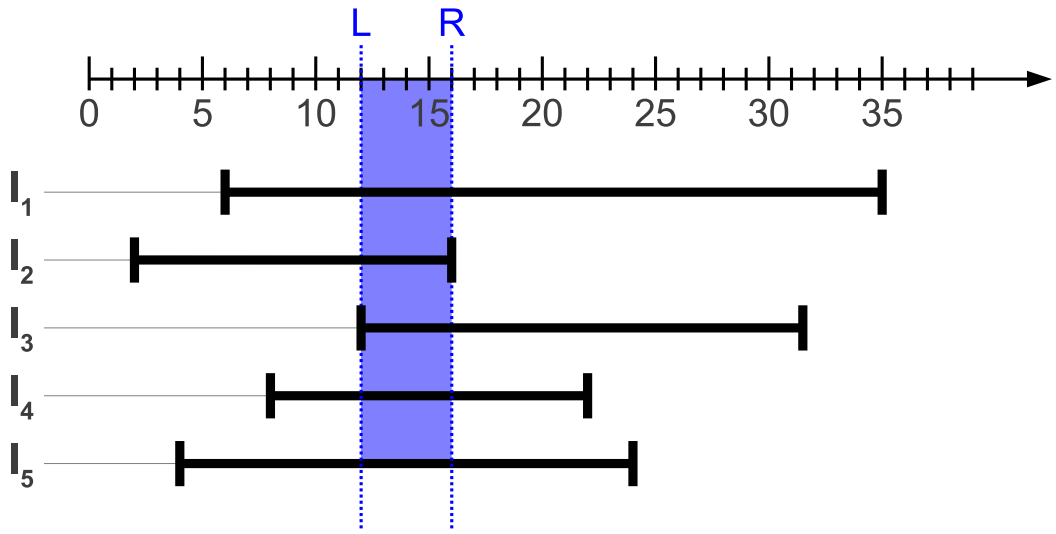


Figure 1.1: *Illustration of the Helly property for intervals*

1.2 Transversals and matchings

The two main concepts considered in this section are transversal and matching of an interval system $\mathcal{I}$. Due to Definitions 0.55 and 0.53, a transversal T of $\mathcal{I}$ is a set of points containing at least one element from each interval $I_i \in \mathcal{I}$, whilst a matching $\mathcal{M}$ is a subsystem of $\mathcal{I}$ which contains pairwise disjoint intervals. For illustration see Figure 1.2.

As we have already proved, the inequality $\tau(G) \geq \nu(G)$ holds for every graph G. A similar argument can be given for the more general case.

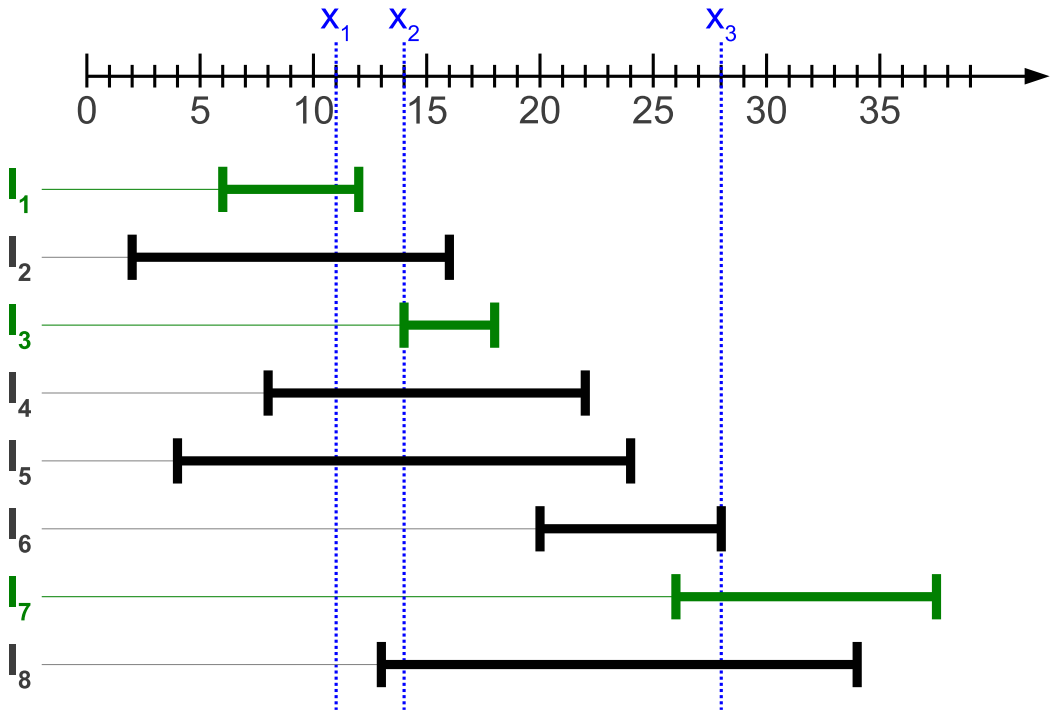


Figure 1.2: *An interval system with a transversal* $T = \{x_1, x_2, x_3\}$ *and with a matching* $\mathcal{M} = \{I_1, I_3, I_7\}$. *As it can be shown,* T *is of minimum and* $\mathcal{M}$ *is of maximum cardinality.*

Proposition 1.1 *For every set system $\mathcal{S}$, the transversal number is not smaller than the matching number:*

$$\tau(\mathcal{S}) \geq \nu(\mathcal{S}).$$

Proof: Consider a maximum matching $\mathcal{M}$. The ν sets in $\mathcal{M}$ are pairwise disjoint and hence, they are covered with ν different elements in a transversal. Thus, each transversal is of cardinality not smaller than ν. □

The inequality may hold in the strong form $\tau > \nu$ for set systems, as shown by the following simple examples:

- Take an n-element underlying set X and all k-element subsets of it. The set system $\mathcal{K}_n^k$ obtained has transversal number $n-k+1$ (in fact any $(n-k+1)$-element subset of X is a minimum transversal) and matching number $\lfloor n/k \rfloor$. This yields $\tau > \nu$ for every n and k which satisfy $2 \leq k < n$.

- In Chapter 0, we gave examples for transversal numbers and matching numbers of specified graphs. For instance, the transversal number of an odd cycle C_{2k+1} equals $k+1$, whilst its matching number is only k.

In this section we prove that for interval systems the two parameters are always equal.

In optimization problems a typical task is to determine a smallest transversal or a largest matching. For set systems in general, both problems are NP-complete, but we will see that for interval systems they can be solved in polynomial time.[2] The following algorithm determines a transversal T and a matching $\mathcal{M}$ with the same cardinality for a generic input interval system $\mathcal{I}$.

Algorithm 1.1

1. Arrange the intervals in a list in the order of increasing[3] right ends. Let $T = \emptyset$, $\mathcal{M} = \emptyset$.

2. Take the first (smallest) right endpoint, say b_j, and put it into T.

[2] If the set system consists of the edges of a graph, then the matching number can be determined in polynomial time, but to determine the transversal number is an NP-complete problem.

[3] We use the term 'increasing sequence' in the sense of what is also often called 'non-decreasing'; that is, we say that a sequence $a_1, a_2, \ldots$ is increasing if $a_i \leq a_{i+1}$ holds for every $i \geq 1$. Similarly, $a_1, a_2, \ldots$ is a decreasing sequence if $a_i \geq a_{i+1}$ holds for every $i \geq 1$.

3. Take the interval I_j (with right end b_j) and put it into $\mathcal{M}$.
4. Delete all intervals from the list which contain b_j.
5. If the list is not empty, go to Step 2, otherwise stop.

Described in pseudo code:

Algorithm 1.1 Algorithm to determine τ and ν for interval systems

$T := \emptyset$, $\mathcal{M} := \emptyset$, $\mathcal{I} := \{I_1, I_2, \ldots, I_n\}$ with increasing right ends
while $\mathcal{I} \neq \emptyset$ **do**
  $b' := \min_{I_i \in \mathcal{I}} b_i$
  $T := T \cup \{b'\}$
  Select I' form $\{I_i \in \mathcal{I} \mid b_i = b'\}$ arbitrarily
  $\mathcal{M} := \mathcal{M} \cup \{I'\}$
  $\mathcal{I} := \{I \in \mathcal{I} \mid b' \notin I\}$
end while
$\tau = |T|$, $\nu = |\mathcal{M}|$, T is a smallest transversal and $\mathcal{M}$ is a largest matching.

At the end of the procedure all intervals are deleted and hence, each of them is covered by at least one point from T. Moreover, at the end of any turn the list consists of intervals which are disjoint from each interval in $\mathcal{M}$, because the left end of any interval remaining in the list is larger than the right ends of all selected intervals. Consequently, T is a transversal, $\mathcal{M}$ is a matching, and they are of the same cardinality. Since τ is the ***minimum*** size of a transversal and ν is the ***maximum*** size of a matching, moreover we have seen that $\nu(\mathcal{S}) \leq \tau(\mathcal{S})$ holds for every set system $\mathcal{S}$, we obtain

$$\tau(\mathcal{I}) \leq |T| = |\mathcal{M}| \leq \nu(\mathcal{I}) \leq \tau(\mathcal{I}).$$

This implies that all inequalities in the chain above must hold with equality. Then, $\tau(\mathcal{I}) = |T|$ and $|\mathcal{M}| = \nu(\mathcal{I})$ follow; that is, the algorithm outputs a minimum transversal and a maximum matching. Furthermore, the last relation also holds with equality and proves the following theorem.

Theorem 1.2 *For every interval system $\mathcal{I}$,*

$$\tau(\mathcal{I}) = \nu(\mathcal{I}).$$

This statement generalizes the 1-dimensional Helly theorem. The condition of Theorem 1.1 says that any two intervals intersect that is $\nu = 1$ holds, and the conclusion is the existence of a common point that is $\tau = 1$. Now, Theorem 1.2 equivalently means that $\nu = k$ implies $\tau = k$, for every integer k.

In Subsection 1.5 we present an example for applying Algorithm 1.1.

1.3 Decomposition into intersecting subsystems

Definition 1.1 *A set system is called* ***intersecting*** *if any two members of it have a nonempty intersection.*

Equivalently, a set system $\mathcal{S}$ is intersecting if and only if no two of its members are disjoint that means $\nu(\mathcal{S}) = 1$. In general, $\tau(\mathcal{S}) = 1$ implies that $\mathcal{S}$ is intersecting, but the converse is not true. For example, over the set $\{a, b, c\}$ the system $\mathcal{S} = \{\{a, b\}, \{a, c\}, \{b, c\}\}$ is intersecting but $\tau(\mathcal{S}) = 2$.

By definition, every set system containing only one set is intersecting. Hence, every system can be decomposed (partitioned) into intersecting subsystems. The goal is to find a decomposition of $\mathcal{S}$ into the ***minimum number*** $k(\mathcal{S})$ of intersecting subsystems.

Theorem 1.3 *For every set system $\mathcal{S}$ and for the minimum number $k(\mathcal{S})$ of intersecting subsystems into which $\mathcal{S}$ can be decomposed, we have*

$$\nu(\mathcal{S}) \leq k(\mathcal{S}) \leq \tau(\mathcal{S}).$$

Proof: No two disjoint sets can belong to the same intersecting subsystem. Hence, the ν sets in a maximum matching are contained in ν different intersecting subsystems, which proves $\nu(\mathcal{S}) \leq k(\mathcal{S})$. On the other hand, if we have a $\tau(\mathcal{S})$-element transversal T and each set $S \in \mathcal{S}$ is assigned to an $x \in T$ which covers S, the obtained $t \leq \tau(\mathcal{S})$ nonempty intersecting subsystems together decompose $\mathcal{S}$. Therefore, $k(\mathcal{S}) \leq \tau(\mathcal{S})$ must hold. □

By Theorem 1.2, for any interval system its matching number and its transversal number are equal. This fact, together with Theorem 1.3, immediately implies the following chain of equalities.

Corollary 1.1 *For every interval system $\mathcal{I}$ and for the minimum number $k(\mathcal{I})$ of intersecting subsystems into which $\mathcal{I}$ can be decomposed,*

$$\nu(\mathcal{I}) = k(\mathcal{I}) = \tau(\mathcal{I}).$$

With a simple extension, Algorithm 1.1 can produce a decomposition into the minimum number of intersecting subsystems. Let Step 4 be replaced with the following:

4′ If an interval contains b_j, put it into the subsystem $\mathcal{K}(b_j)$ and delete the interval from the list.

Then, the $k = \tau(\mathcal{I})$ subsystems $\mathcal{K}(b_j)$ (indexed by the elements $b_j \in T$) are intersecting and decompose $\mathcal{I}$.

1.4 Decomposition into matchings

A further typical type of optimization problems is when a system has to be decomposed into subsystems which are matchings. This can always be done, because the decomposition consisting of 1-element subsystems satisfies the requirement. If the system considered is intersecting, we have no smaller decomposition. In general, the decomposition into the minimum number of matchings is an NP-complete problem. But as we will see, also this problem is easy to solve for interval systems.

The decomposition into matchings is also known as a coloring problem; it is a generalization of edge colorings of graphs. Each set is assigned to a color such that intersecting sets must get different colors. Sets assigned to a common color correspond to matchings, as illustrated in Figure 1.3.

The following greedy algorithm solves the problem for interval systems with a method which is called ***First Fit***. The input is an interval system whose members are colored one by one in a fixed order. 'First Fit' means that each interval is assigned to the smallest possible integer not forbidden for it.

Algorithm 1.2

1. Arrange the intervals in a list in the order of increasing *left* ends: $I_1, I_2, \ldots, I_n$. Let $i = 1$.

2. Assign I_i to the smallest possible color that is, to the smallest positive integer which has not been assigned to any intervals intersecting I_i. If $i < n$, let $i := i + 1$, otherwise stop.

In the pseudocode, we denote by $\mathrm{Dom}(\varphi)$ the set of intervals which have already received their colors under the current coloring φ.

Algorithm 1.2 Algorithm to determine $q(\mathcal{I})$ for an interval system $\mathcal{I}$

$\varphi := \emptyset$, $\mathcal{I} := \{I_1, I_2, \ldots, I_n\}$ with increasing left ends
while $\mathrm{Dom}(\varphi) \neq \mathcal{I}$ **do**
  $a' := \min_{I_i \in \mathcal{I} \setminus \mathrm{Dom}(\varphi)} a_i$
  Select I' form $\{I_i \in \mathcal{I} \setminus \mathrm{Dom}(\varphi) \mid a_i = a'\}$ arbitrarily
  $k := \min_{\substack{k \in \mathbb{Z}^+ \\ \nexists I \in \mathrm{Dom}(\varphi),\, a' \in I,\, \varphi(I) = k}} k$
  $\varphi := \varphi \cup \{(I', k)\}$
end while
φ is a minimal coloring of $\mathcal{I}$

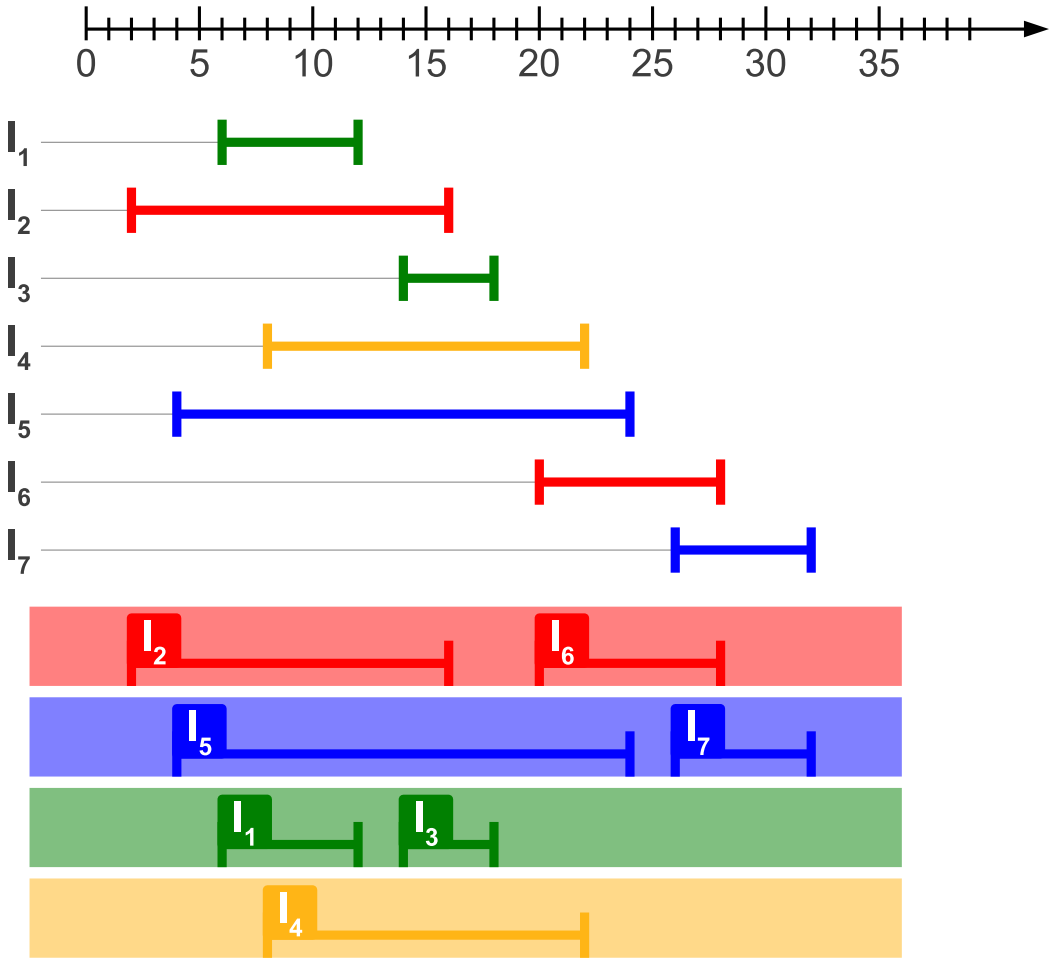


Figure 1.3: *Decomposition of a system of 7 intervals into matchings*

Denote by $\Delta(\mathcal{I})$ the maximum degree of the interval system $\mathcal{I}$; that is, the maximum number of intervals in $\mathcal{I}$ sharing a point. We prove that Algorithm 1.2 uses $\Delta(\mathcal{I})$ colors and yields an optimal coloring.

Theorem 1.4 *For every interval system $\mathcal{I}$, the possible minimum number of matching subsystems into which $\mathcal{I}$ can be decomposed is equal to the maximum degree $\Delta(\mathcal{I})$. Moreover, Algorithm 1.2 yields a decomposition into $\Delta(\mathcal{I})$ matchings.*

Proof: First, in every coloring (decomposition into matchings) of $\mathcal{I}$ the Δ intervals sharing a point must have Δ different colors. On the other hand, we prove that Δ colors are enough. Since the intervals are ordered according to their left endpoints, for every index-pair $i < j$ the interval I_i meets $I_j = [a_j, b_j]$ if and only if I_i contains a_j. Since a_j is incident with at most Δ intervals and one of them is I_j itself, at the moment when Algorithm 1.2 colors I_j, only at most $\Delta - 1$ intervals intersecting it were colored previously. Consequently, no interval is assigned to a color greater than Δ. This proves that the minimum number of colors (i.e., matching subsystems in the decomposition) equals Δ, moreover Algorithm 1.2 gives an optimal coloring. □

1.5 Example

Procedures of Algorithms 1.1 and 1.2 are illustrated with the following example. Let system $\mathcal{I}$ consist of eight intervals, also shown in Figure 1.4.

$I_1 = [1, 13]$ $I_2 = [17, 24]$ $I_3 = [12, 20]$
$I_4 = [3, 7]$ $I_5 = [9, 15]$ $I_6 = [5, 10]$
$I_7 = [14, 26]$ $I_8 = [18, 22]$

Algorithm 1.1

- The intervals have to be ordered with respect to increasing right ends:

$$I_4,\ I_6,\ I_1,\ I_5,\ I_3,\ I_8,\ I_2,\ I_7$$

- Choose $b_4 = 7$;
  $T \leftarrow b_4$;
  $\mathcal{M} \leftarrow I_4$;
  Delete: I_4, I_6, I_1;
  Remaining: I_5, I_3, I_8, I_2, I_7.

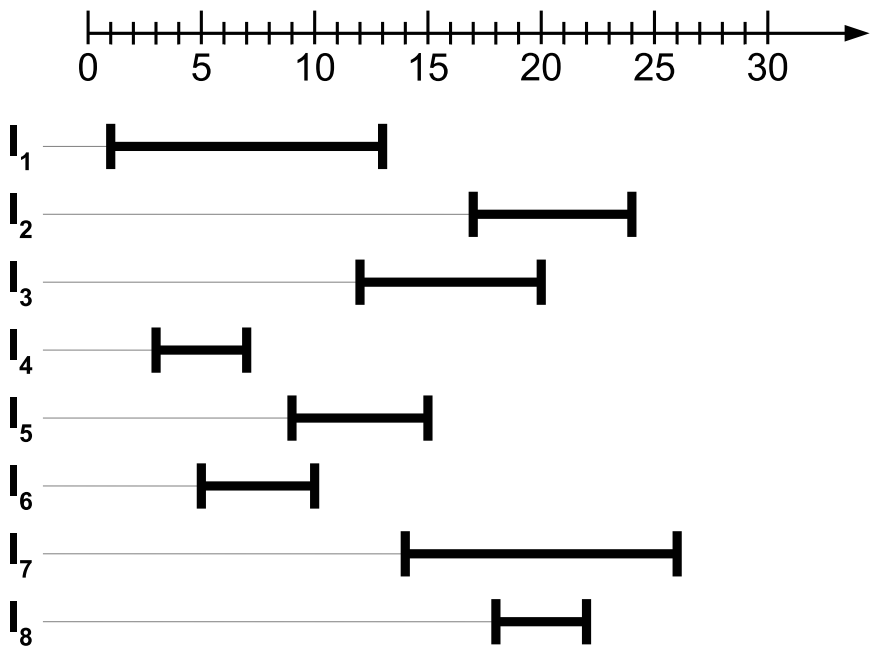


Figure 1.4: *System $\mathcal{I}$ on which Algoritms 1.1 and 1.2 are illustrated*

- Choose $b_5 = 15$;
  $T \leftarrow b_5$;
  $\mathcal{M} \leftarrow I_5$;
  Delete: I_5, I_3, I_7
  Remaining: I_8, I_2.

- Choose $b_8 = 22$;
  $T \leftarrow b_8$;
  $\mathcal{M} \leftarrow I_8$;
  Delete: I_8, I_2
  No interval remains.

- Output:
  $\tau(\mathcal{I}) = \nu(\mathcal{I}) = k(\mathcal{I}) = 3$;
  Minimum transversal $T = \{b_4, b_5, b_8\}$;
  Maximum matching $\mathcal{M} = \{I_4, I_5, I_8\}$;
  Decomposition into minimum number of intersecting subsystems

$$\{I_4, I_6, I_1\}, \ \{I_5, I_3, I_7\}, \ \{I_8, I_2\}.$$

Algorithm 1.2

- The intervals have to be ordered with respect to increasing left ends:

$$I_1,\ I_4,\ I_6,\ I_5,\ I_3,\ I_7,\ I_2,\ I_8$$

- $I_1 \rightarrow$ color 1
  Color 1 is forbidden for I_4, I_6, I_5, I_3.

- $I_4 \rightarrow$ color 2
  Color 2 is forbidden for I_6.

- $I_6 \rightarrow$ color 3
  Color 3 is forbidden for I_5.

- $I_5 \rightarrow$ color 2
  Color 4 is forbidden for I_3, I_7.

- $I_3 \rightarrow$ color 3
  Color 3 is forbidden for I_7, I_2, I_8

- $I_7 \rightarrow$ color 1
  Color 1 is forbidden for I_2, I_8.

- $I_2 \rightarrow$ color 2
  Color 2 is forbidden for I_8.

- $I_8 \rightarrow$ color 4
  No further interval.

- Output:
  Coloring of intervals with minimum number (4) of colors;
  Decomposition into minimum number (4) of matching subsystems

$$\{I_1, I_7\},\ \{I_4, I_5, I_2\},\ \{I_6, I_3\},\ \{I_8\}.$$

One can check that the maximum degree in $\mathcal{I}$ equals 4, for example point $a_8 = 18$ is contained in exactly four intervals.

1.6 Interval systems and subpaths of a path

Let us note at the end of this chapter that the combinatorial properties of finite interval systems can also be represented with discrete mathematical models. For example, stretching the subintervals to have integer lengths

between any two consecutive endpoints[4] will change only the sizes of some intervals but has no effect on the structure and intersection/disjointness relation, neither on the parameters τ, ν, k, Δ studied above. Further, assuming that all endpoints are integers, the structure and the quantitative parameters remain unchanged if we only keep the *integer points* in each interval. In this way, we obtain a representation in terms of graphs and hypergraphs. An interval system may be viewed as a collection of *subpaths of a path* which is also called interval hypergraph. (For instance, if the smallest left endpoint is 1 and the largest right endpoint is n, and one of the intervals is $[3, 7]$, then this interval is represented with the subpath $v_3v_4v_5v_6v_7$ in the path $P_n = v_1v_2 \dots v_n$.)

[4] With this modification, the distance between *any two* endpoints becomes an integer, not only between consecutive ones.

Chapter 2

Interval graphs and sequential coloring

2.1 Intersection graph of an interval system

Given a set system $\mathcal{S}$, its intersection graph[1] expresses the structure of the pairwise intersections between the members of $\mathcal{S}$.

Definition 2.1 *The **intersection graph** $G(\mathcal{S})$ of a set system $\mathcal{S}$ has one vertex v_i for each set $S_i \in \mathcal{S}$ moreover two different vertices v_i and v_j are adjacent in $G(\mathcal{S})$ if and only if the corresponding members S_i and S_j of $\mathcal{S}$ have a nonempty intersection.*

Since an edge uv of a graph G is defined as the set $\{u, v\}$, the line graph $L(G)$ is exactly the intersection graph of the system whose members are the edges of G. We also note that essentially different intersection patterns can yield the same intersection graph, as shown in Figure 2.1. Moreover, every simple graph can be obtained as an intersection graph. Indeed, for any graph G with vertices $v_1, v_2, \ldots, v_n$, and with edges $e_1, e_2, \ldots, e_m$, take the system of sets $S_1, S_2, \ldots, S_n$ over an underlying set $\{x_1, \ldots, x_m\} \cup \{y_1, \ldots, y_n\}$ of cardinality $m+n$, where $S_i = \{x_j : v_i \in e_j\} \cup \{y_i\}$. This construction always yields a set system in which no three members have a common element, moreover its intersection graph is G. For instance, starting with the graph G on Figure 2.1 we obtain system $\mathcal{S}_1$ of the same figure.

Now, we turn to the intersection graphs of interval systems.

Definition 2.2 *A graph which is an intersection graph of some interval system is called an **interval graph**.*

[1] Intersection graph is also called 'representative graph' in the literature.

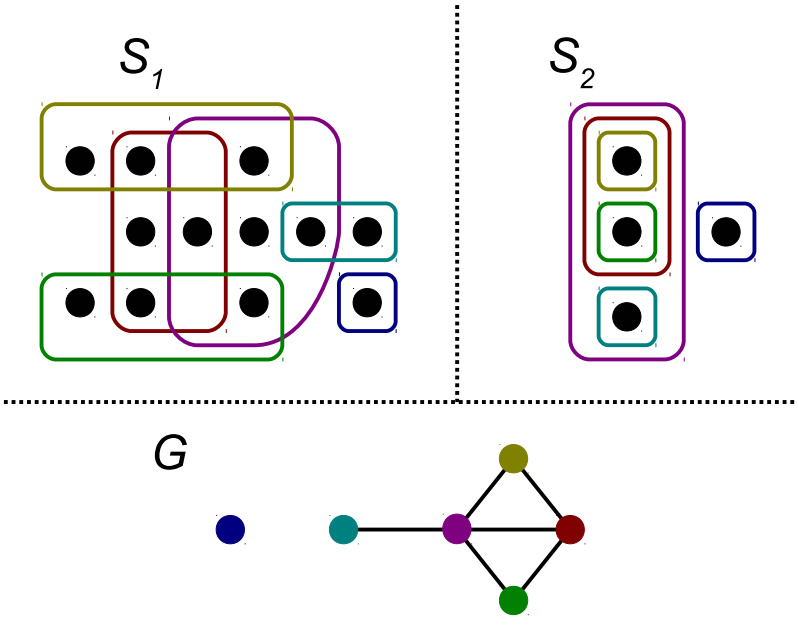


Figure 2.1: *G is the intersection graph of* $\mathcal{S}_1$*, and that of* $\mathcal{S}_2$*, as well*

That is, the intervals are represented by vertices and adjacency means that the corresponding two intervals intersect. See Figure 2.2 for the intersection graph of system $\mathcal{I}$ which was considered in Section 1.5.

There exist graphs which are not interval graphs. For instance, no cycle of length at least four can be obtained as the intersection graph of any interval system. This assertion can be verified by observing that in any interval system an interval with smallest right end either meets at most one member of the system or two members intersecting it also meet each other; consequently in the intersection graph the interval in question corresponds to a vertex which either has degree less than two or is contained in a triangle K_3. No such vertices occur in a cycle longer than three.

Another example is shown in Figure 2.3. This is not an interval graph, even though it contains no induced cycle of length greater than 3, moreover every induced subgraph of it is an interval graph. (To see that it is not an interval graph, observe that the three mutually non-adjacent external vertices should correspond to three pairwise disjoint intervals, say I_1, I_2, I_3 on the real line in this order, and there should also occur an interval which meets both I_1 and I_3 but is disjoint from I_2. This is impossible.)

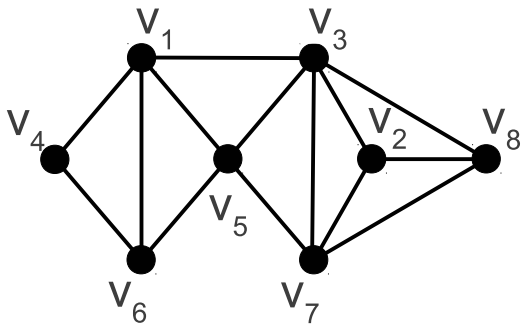


Figure 2.2: *Intersection graph of system $\mathcal{I}$ shown on Figure 1.4*

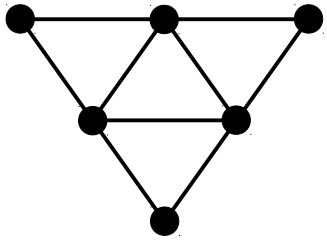


Figure 2.3: *This is not an interval graph; note that every induced cycle of it has length 3.*

Parameters studied in the previous chapter for an interval system $\mathcal{I}$ have their corresponding pairs in the intersection graph $G(\mathcal{I})$.

1. By definition, an intersecting subsystem of $\mathcal{I}$ yields pairwise adjacent vertices that is a clique in $G(\mathcal{I})$, and vice versa. By Helly's theorem, the maximum size of an intersecting subsystem equals the maximum number $\Delta(\mathcal{I})$ of intervals incident with a point in $\mathcal{I}$. Consequently, the maximum order of a clique in the intersection graph equals the maximum degree of the system: $\Delta(\mathcal{I}) = \omega(G(\mathcal{I}))$.

2. By the previous correspondence, the minimum number $k(\mathcal{I})$ of intersecting subsystems to which the intervals of $\mathcal{I}$ can be decomposed gives the minimum number of cliques which together cover all vertices in $G(\mathcal{I})$: $k(\mathcal{I}) = \theta(G(\mathcal{I}))$.

3. It is clear by definition that the matchings of $\mathcal{I}$ are in one-to-one correspondence with the independent vertex sets of $G(\mathcal{I})$. Then, for their maximum cardinalities $\nu(\mathcal{I}) = \alpha(G(\mathcal{I}))$ holds.

4. When the system $\mathcal{I}$ is decomposed into matchings, the vertex set of $G(\mathcal{I})$ is partitioned into independent vertex classes, and vice versa. Thus, the proper colorings of the intervals are in one-to-one correspondence with the proper vertex colorings of $G(\mathcal{I})$. For the minimum number of colors (partition classes) we get $q(\mathcal{I}) = \chi(G(\mathcal{I}))$.

Table 2.1 summarizes the correspondences between properties and parameters of interval systems and their intersection graphs.

Taking into account that for interval systems $q(\mathcal{I}) = \Delta(\mathcal{I})$ and $k(\mathcal{I}) = \tau(\mathcal{I}) = \nu(\mathcal{I})$ were proved, we obtain:

Theorem 2.1 *For any interval graph G,*

$$\chi(G) = \omega(G) \qquad \text{and} \qquad \theta(G) = \alpha(G).$$

Graph G in Figure 2.2 is the intersection graph of system $\mathcal{I}$ from Section 1.5. By the above correspondences and by the results obtained in Section 1.5 we have:

- $\alpha(G) = 3$ and a maximum independent set is $\{v_4, v_5, v_8\}$.
- $\theta(G) = 3$ and a minimum clique cover is

$$\{v_4, v_6, v_1\}, \ \{v_5, v_3, v_7\}, \ \{v_8, v_2\}.$$

Table 2.1: Correspondence between the parameters of interval systems and their intersection graphs

interval system $\mathcal{I}$	intersection graph $G(\mathcal{I})$
intersecting intervals	adjacent vertices
intersecting subsystem $\Delta(\mathcal{I})$	clique $\omega(G(\mathcal{I}))$
decomposition into intersecting subsystems $k(\mathcal{I})$	clique cover $\theta(G(\mathcal{I}))$
disjoint intervals	non-adjacent vertices
matching $\nu(\mathcal{I})$	independent vertex set $\alpha(G(\mathcal{I}))$
decomposition into matchings (coloring of intervals) $q(\mathcal{I})$	proper vertex coloring (partition to independent vertex sets) $\chi(G(\mathcal{I}))$

- $\omega(G) = 4$ and a maximum clique is induced by $\{v_3, v_7, v_2, v_8\}$ (because in $\mathcal{I}$ point a_8 is of maximum degree 4 and the intervals incident with it are I_3, I_7, I_2 and I_8.)
- $\chi(G) = 4$ and a 4-coloring is obtained with color classes

$$\{v_1, v_7\}, \ \{v_4, v_5, v_2\}, \ \{v_6, v_3\}, \ \{v_8\}.$$

We note that interval graphs can be recognized in time proportional to the number $|V| + |E|$ of vertices plus edges, moreover for interval graphs a corresponding interval system can also be constructed efficiently. Then by Algorithms 1.1 and 1.2, the significant graph parameters ω, α, θ, χ and also $\tau = |V| - \alpha$ (each of them is hard to determine for graphs in general) can be computed by fast algorithms on the class of interval graphs.

2.2 Sequential coloring

We applied a First Fit algorithm to color intervals ordered due to increasing left ends. If the vertices of the intersection graph are considered in the corresponding order, the same method yields a proper vertex coloring with minimum number of colors.

In general, First Fit can be used for coloring vertices of any graph in any fixed order.

First Fit coloring:
Given a graph G and a vertex order $v_1, v_2, \ldots, v_n$, color the vertices in this order with colors $1, 2, \ldots,$ such that each vertex v_i gets the smallest color which has not been assigned to any previously colored neighbors of v_i.

The coloring is necessarily proper as we do not use the same color on adjacent vertices, but the optimality is not guaranteed.[2] Just consider the following bipartite graph G:

$$V(G) = \{v_1, v_2, \ldots, v_k, u_1, u_2, \ldots, u_k\}$$

$$E(G) = \{v_i u_j \mid i \neq j \ \wedge \ 1 \leq i, j \leq k\}$$

(This is the complete bipartite graph $K_{k,k}$ from which a 'perfect matching' consisting of the edges $v_1u_1, v_2u_2, \ldots, v_ku_k$ is deleted.) If we apply the First Fit coloring with the original vertex order as listed above, we obtain a coloring with two colors, which is optimal as $\chi(G) = 2$. But if the order $v_1, u_1, v_2, u_2 \ldots, v_k, u_k$ is taken, then for every vertex v_i or u_i exactly the colors smaller than i are forbidden. And then, First Fit outputs a coloring with k colors. This indicates that First Fit is sensitive to vertex order.

A First Fit coloring assigns each vertex v_i to a color which is not greater than the number of neighbors preceding v_i plus 1. Hence, our goal is to bound the number of preceding neighbors in the coloring order.

Definition 2.3 *Given a graph G and a vertex order $v_1, v_2, \ldots v_n$, let $d^-(v_i)$ denote the number of neighbors of v_i which precede it:*

$$d^-(v_i) = |\{v_j \mid v_i v_j \in E(G) \ \wedge \ j < i\}| \,.$$

Then, the ***coloring number*** $\mathrm{col}(G)$ *of graph G is the minimum of the maximum value of $d^-(v_i)$ plus 1, taken over all vertex orders:*

$$\mathrm{col}(G) = \min_{\text{vertex orders}} \max \{d^-(v_i) + 1 \mid 1 \leq i \leq n\}.$$

[2] Nevertheless, for every graph G there exists a vertex order such that First Fit produces a coloring with minimum number $\chi(G)$ of colors. But this order is hard to determine in general.

A vertex order where max $\{d^-(v_i) + 1 \mid 1 \leq i \leq n\}$ equals the coloring number is called optimal. The coloring number is an upper bound on the chromatic number.

Theorem 2.2 *For every graph G,*

$$\chi(G) \leq \mathrm{col}(G).$$

Proof: Consider an optimal vertex order $v_1, v_2, \ldots, v_n$ of G and color the vertices using the First Fit algorithm. As every vertex v_i is preceded by $d^-(v_i)$ of its neighbors, when we color v_i, not more than $d^-(v_i)$ colors are forbidden for v_i. Then, v_i gets a color which is not greater than $d^-(v_i) + 1$. By definition $d^-(v_i)+1 \leq \mathrm{col}(G)$ for every v_i, hence First Fit yields a coloring with at most $\mathrm{col}(G)$ colors and we conclude $\chi(G) \leq \mathrm{col}(G)$. □

Recall from the introductory chapter that $\Delta(G) + 1$ is a trivial upper bound on the chromatic number. Since $\mathrm{col}(G) \leq \Delta(G) + 1$, now we have a better upper bound on $\chi(G)$. Moreover, whilst the determination of $\chi(G)$ is algorithmically hard, the following theorem shows that $\mathrm{col}(G)$ can be calculated efficiently. An interesting aspect of this result is that in the definition of $\mathrm{col}(G)$ all the $n!$ vertex orders of G are involved, which grows superexponentially as a function of the number n of vertices.

Theorem 2.3 *The coloring number* $\mathrm{col}(G)$ *can be determined for any graph G in polynomial time.*

Proof: We construct the following order of the n vertices of G:

- Choose a vertex of minimum degree in G and let it be called v_n. This will be the last vertex in the order.
- Then, for every $i = n-1, n-2, \ldots, 1$ select a vertex of minimum degree in the subgraph induced by the set $V(G) \setminus \{v_j \mid j > i\}$ of remaining vertices. Let it be called v_i.

This procedure results in a vertex order $v_1, v_2, \ldots, v_n$. We prove that this is an optimal one.

Consider any optimal order of the vertices. If it is $v_1, v_2, \ldots, v_n$, there is nothing to prove. Otherwise select the largest index i where the two orders differ. In the original order $v_1, v_2, \ldots, v_n$ we have v_i and in the optimal one we have v_k with $k < i$. Modify the optimal order by placing v_i right after v_k. By this change $d^-(v_i)$ may increase but it cannot be higher than $d^-(v_k)$ was before the modification, since v_i has minimum degree in $V(G) \setminus \{v_j \mid j > i\}$.

Moreover, for a vertex v_ℓ with $\ell > i$ the degree $d^-(v_\ell)$ does not change, while for a v_ℓ with $\ell < i$ (including $\ell = k$), $d^-(v_\ell)$ either decreases or remains the same. Consequently, max d^- does not increase and the order remains optimal. Repeating this procedure, at each turn the largest index where the order $v_1, v_2, \ldots, v_n$ and the optimal one differ will be smaller by at least 1, and finally the optimal one is transformed into $v_1, v_2, \ldots, v_n$ preserving optimality. □

Here is another way to obtain an optimal order:

- All the vertices of minimum degree are taken at the end of the order. Then, these vertices are deleted and the step is repeated for the subgraph induced by the remaining vertices.

In Section A.3, we present some examples for the First Fit coloring and for the determination of an optimal vertex order and the coloring number of graphs.

Chapter 3

Chordal graphs

3.1 Subtrees of a tree

We have already seen that a finite interval system can be represented in terms of graphs, as it corresponds to a collection of subpaths of a path. Various properties of interval systems remain valid in the more general structure class composed by collections of subtrees of trees.[1] Here we mention the following important one.

Theorem 3.1 (Helly property for subtrees) *Let $T_1, T_2, \ldots, T_n$ be subtrees of a tree graph T. If any two T_i, T_j share at least one vertex ($1 \leq i < j \leq n$), then some vertex is contained in all T_i ($1 \leq i \leq n$).*

Later, our results on chordal graphs imply further statements on parameters of systems of subtrees.

3.2 Chordal graphs and simplicial order

Definition 3.1 *A graph is **chordal**[2] if it contains no induced cycle of length greater than 3.*

The name 'chordal' expresses that every cycle of the graph whose length is at least 4 cannot be induced, i.e. it has some chord(s). It is immediate from the definition that every tree is a chordal graph, as it has no cycle at

[1] A subtree—as already the name indicates—of a tree T is a connected subgraph of T. If the subtrees are represented with the vertex sets covered, the system is also called hypertree.

[2] A chordal graph is also called 'rigid circuit graph' and 'triangulated graph' in the literature.

all. Another important subclass of chordal graphs is the class of interval graphs as we have seen that no chordless cycles longer than 3 can occur in the latter. We shall return to this connection between graph classes later in Corollary 3.1.

We introduce the following notions in terms of whom the class of chordal graphs can also be defined.

Definition 3.2 *A vertex v is **simplicial** in G if and only if any two of its neighbors are adjacent.*

Hence, every vertex with degree 1 or 0 is simplicial. Moreover, if $d(v) \geq 1$, the definition is equivalent to the requirement that $N(v)$ induces a clique in G.

Definition 3.3 *A **simplicial order**[3] of G is an order $v_1, v_2, \ldots, v_n$ of its vertices such that for every $1 \leq i \leq n-1$, vertex v_i is simplicial in the subgraph induced by $v_i, v_{i+1}, \ldots, v_n$.*

For example, for the graph G on six vertices a, b, c, d, e, f and with eight edges $ab, ac, ad, ae, bc, bf, cd, de$ (cf. Figure 3.1) a possible simplicial order is e, d, a, f, c, b and also f, b, c, e, a, d and several further ones are simplicial. But e, f, c, d, a, b is not a simplicial order because c is not a simplicial vertex in the subgraph induced by c, d, a, b, as it has two non-adjacent neighbors, namely d and b.

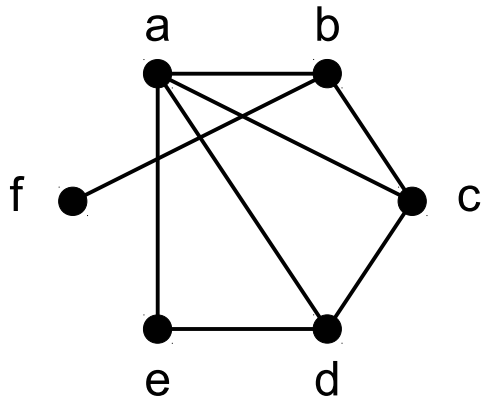


Figure 3.1: *A graph with several simplicial orders*

[3] A simplicial order is also called perfect elimination order in the literature.

Theorem 3.2 *A graph has a simplicial order if and only if every induced subgraph of it has a simplicial vertex.*

Proof: If every induced subgraph of G (including G itself) has a simplicial order then we can choose an arbitrary simplicial vertex in G and a further one in the subgraph induced by the remaining vertices, and so on. Finally, we have a simplicial order, definitely.

On the other hand, assuming a simplicial order $v_1, v_2, \ldots, v_n$ of G, for every induced subgraph $G' \subseteq G$, the vertex of G' which has the smallest index in the order above is surely simplicial in G'. □

This theorem implies that if a graph admits a simplicial order, then the following procedure always yields an appropriate one; no matter which simplicial vertex is chosen in a step (if there exist more than one).

- For $i = 1, 2, \ldots n$:
  Let G_i be the subgraph induced by $V(G) \setminus \{v_j : j < i\}$.
  - If there is no simplicial vertex in G_i, then G has no simplicial order. STOP
  - Otherwise let v_i be an arbitrary simplicial vertex of G_i.
- If all the n iterations have been executed, the output is $v_1, v_2, \ldots, v_n$, which is a simplicial order of G.

Remark 3.1 *For a fixed vertex $v \in V(G)$, deciding whether v is simplicial in G means deciding whether every two neighbors of v are adjacent. This needs at most $(n-1)(n-2)/2$ steps. Then, the determination of a simplicial vertex in a graph G (or finding out that there is no such vertex) takes polynomial time. Therefore, applying Theorem 3.2 and its proof, deciding whether there is a simplicial vertex order and determining one, if it exists, can be done in polynomial time.*

The following important characterization theorem establishes a close connection between the notions defined in this section.

Theorem 3.3 *For any graph G, the following statements are equivalent:*

(*i*) *G is chordal;*

(*ii*) *G has a simplicial order;*

(*iii*) *G is the intersection graph of a collection of subtrees of some tree T.*

Remark 3.2 *In part (iii), a subtree T' is considered as the set of vertices contained in it and hence, in the intersection graph adjacency means that the corresponding subtrees share a vertex.*

As mentioned above, induced cycles longer than 3 cannot occur in interval graphs. Theorem 3.3 offers us the possibility to point out the relation between chordal graphs and interval graphs from a different approach, too.

Corollary 3.1 *Every interval graph is chordal.*

Proof: We have already noted that an interval system can be viewed as a set of subpaths of a path (graph). Hence, it is a collection of subtrees of a special kind of tree (i.e. a path). Thus, interval graphs, which are precisely their intersection graphs, satisfy part (*iii*) of Theorem 3.3. Consequently, they are chordal graphs. □

Remark that a simplicial order of an interval graph can be determined via ordering the intervals represented due to increasing right ends. When Algorithms 1.1 and 1.2 were applied for an interval system, significant parameters of its intersection graph were also computed. We will see that these methods have their analogous versions for chordal graphs.

Each algorithm discussed in this chapter proceeds by simplicial order (or its inverse). Roughly speaking, simple problems are typically solved using simplicial order, but more difficult tasks often require the representation by subtrees.

Building a representation of a chordal G with nonempty subtrees. We proceed in inverse simplicial order $v_n, \ldots, v_1$. Vertex v_n is represented by the one-vertex subtree $\{x_1\}$ of the tree T consisting of only this vertex. When a vertex v_i is considered, the vertices $v_{i+1}, \ldots, v_n$ are already represented by subtrees $T_{i+1}, \ldots, T_n$ of a tree T. Since the (already represented) neighbors of v_i form a clique in G, the corresponding subtrees are pairwise intersecting. By Theorem 3.1, these subtrees have a vertex, say x_j, in common. It may happen that x_j is contained in subtrees representing vertices non-adjacent with v_i. For this reason we take a new vertex x_i which is a leaf adjacent just to x_j in T, and represent v_i by the one-vertex subtree $\{x_i\}$; moreover, each subtree assigned to a neighbor of v_i is extended by this leaf x_i. If v_i has no neighbors among $v_{i+1}, \ldots, v_n$, we join the new vertex x_i by an edge to an arbitrarily chosen vertex of T, in order to keep the extended T connected. In this way, the representation contains subtrees of a tree in every step and vertices adjacent in G correspond to intersecting subtrees while non-adjacent vertices correspond to vertex-disjoint subtrees.

3.3 Algorithms for chordal graphs

In this section we present algorithms determining the independence number α, clique covering number θ, clique number ω and chromatic number χ for chordal graphs. On the one hand, these parameters are algorithmically hard to determine for graphs in general. On the other hand, we have seen that on the class of interval graphs, which is a subclass of chordal graphs, the parameters above can be computed efficiently. Here we show that this efficiency does not change and also the basic ideas of the algorithms remain applicable for chordal graphs.

Each algorithm discussed in this section starts with the determination of a simplicial order. This can be executed in polynomial time in the way described in the previous section, so here we assume that a simplicial order denoted by $v_1, v_2, \ldots v_n$ is at hand.

3.3.1 Determination of α and θ

Algorithm 3.1

1. Take the first element v_i from the simplicial order and put it into I.

2. Delete v_i and all neighbors of v_i from the list. If the list is not empty, proceed with (1).

In each iteration, the neighbors of the vertex chosen are deleted, hence I is an independent vertex set, it contains $|I| = s$ elements for some s. On the other hand, by the definition of simplicial order, the set of vertices deleted in any step form a clique. Finally, these s cliques together cover all vertices of the graph. Recall that $\theta \geq \alpha$ holds for every graph. Moreover, in the present case, since α is defined to be the maximum cardinality of an independent set, we have $\alpha \geq s$; and since θ is the minimum number of cliques covering all vertices, we have $\theta \leq s$. That is, for a chordal graph and for the number s of iterations in the algorithm:

$$\theta \leq s \leq \alpha \leq \theta.$$

Consequently, in the chain above each relation holds with equality.

As follows, the algorithm determines a maximum independent set I with $s = \alpha$ vertices and a clique cover with the minimum number $s = \theta$ of cliques. This verifies the optimality of the outputs. In addition, this proves the following theorem:

Theorem 3.4 *For every chordal graph G,*

$$\theta(G) = \alpha(G)$$

holds.

3.3.2 Determination of ω and χ

Definition 3.4 *Let $v_1, v_2, \ldots, v_n$ be a simplicial order in a graph G. Relating to this order, the **forward degree** $d^+(v_i)$ of a vertex v_i is*

$$d^+(v_i) = |\{v_j : v_i v_j \in E \ \ \wedge \ \ i < j\}|$$

that is, the number of those neighbors of v_i which are later in the order than v_i.

Theorem 3.5 *For every chordal graph G and for every simplicial order of it,*

$$\omega(G) = \max_{1 \leq i \leq n} \{d^+(v_i) + 1\}.$$

Proof: Consider a chordal graph G and a simplicial order $v_1, v_2, \ldots, v_n$. Then, for every $1 \leq i \leq n$, vertex v_i is simplicial in the subgraph induced by $\{v_i, v_{i+1}, \ldots, v_n\}$. Thus, a clique of $d^+(v_i) + 1$ vertices surely occurs in G implying that

$$\omega(G) \geq d^+(v_i) + 1$$

holds for every i. Then,

$$\omega(G) \geq \max_{1 \leq i \leq n} \{d^+(v_i) + 1\}$$

holds as well.

On the other hand, for a clique of ω vertices consider the vertex v_i which is the earliest one among them in the order. For this vertex, $d^+(v_i) = \omega(G) - 1$ holds and we have

$$\omega(G) \leq \max_{1 \leq i \leq n} \{d^+(v_i) + 1\}.$$

This completes the proof. □

As follows, the clique number ω can be efficiently computed on the class of chordal graphs. Just determine any simplicial order and the forward degrees of vertices. Theorem 3.5 then shows a direct way to compute ω. Choosing some v_i with largest $d^+(v_i)$ and taking it together with its later neighbors, we also find a maximum clique efficiently.

To determine the chromatic number of chordal graphs, we propose the following algorithm, which proceeds in inverse simplicial order $v_n, v_{n-1}, \ldots, v_2, v_1$.

Algorithm 3.2

- Apply First Fit coloring in order $v_n, v_{n-1}, \ldots, v_2, v_1$.

At the moment when a color is assigned to vertex v_i, exactly $d^+(v_i)$ neighbors of it were colored previously. Therefore, precisely $d^+(v_i)$ colors are forbidden for v_i (as those neighbors are mutually adjacent), and First Fit can assign a color to v_i from the set $\{1, 2, \ldots, d^+(v_i) + 1\}$. Consequently, the coloring uses at most $\max_{1 \le i \le n}\{d^+(v_i) + 1\}$ colors. This maximum value is an upper bound for χ. On the other hand, ω is a trivial lower bound for χ (not only for chordal graphs), and together with Theorem 3.5, we have

$$\omega(G) \le \chi(G) \le \max_{1 \le i \le n}\{d^+(v_i) + 1\} = \omega(G).$$

Thus, equalities must hold all along the chain, which proves that the algorithm above results in an optimal χ-coloring for any chordal graph. Also, the leftmost equality implies the following theorem.

Theorem 3.6 *For every chordal graph G,*

$$\omega(G) = \chi(G).$$

It can also be shown that inverse simplicial order (if exists) is also an optimal order for sequential coloring. Moreover, the chromatic and the coloring numbers are equal for any chordal graph. It is important to note, however, that the algorithms presented in the previous chapter for finding an optimal vertex order with respect to $\mathrm{col}(G)$ cannot be applied to find simplicial orders of chordal graphs in general. The reason is that in some chordal graphs the vertices of minimum degree are not simplicial.

Chapter 4

Tree decompositions of graphs

In the previous chapter we proved several useful structural properties of chordal graphs. The algorithmic ideas to be presented here will be based on them (although we shall not mention this at every point).

We are going to develop a method which leads to efficient algorithms for a large class of hard problems on a certain class of graphs. In fact, here "class of graphs" means a nested sequence $\mathcal{G}_1 \subset \mathcal{G}_2 \subset \cdots$ of graph classes; the individual members of this sequence are relatively small[1] when compared to the class of all graphs, nevertheless each graph occurs in some member of this sequence. The rather general structure of this approach is exhibited in the following scheme, in which we build various kinds of structures one after the other. The larger graph H in the second line of the scheme is often called a *chordal supergraph*[2] of G. The middle step from H to $\mathcal{T}$ is feasible because of the characterization theorem of chordal graphs as those representable as intersection graphs of trees (Theorem 3.3).

graph $G = (V, E)$
$\downarrow$
chordal graph H containing G as a subgraph
$\downarrow$
collection $\mathcal{T}$ of subtrees of a tree
such that H is the intersection graph of $\mathcal{T}$
$\downarrow$
tree graph T with nodes (vertices) x_k and associated subsets $S_k \subseteq V$
satisfying specific properties

[1] This is not surprising, because for very large classes one cannot expect efficient algorithms on provably hard (NP-hard) problems.

[2] In general, the phrases 'G_1 is a subgraph of G_2' and 'G_2 is a supergraph of G_1' are equivalent, nevertheless the latter is used in particular contexts only; e.g., the one here.

It is the model obtained in the last phase of this scheme which will be the base to design algorithms. The tree structure behind the sets S_k allows to apply the principle of *dynamic programming*. To execute those algorithms we shall not need the intermediate graph H and the subtrees occurring in $\mathcal{T}$; we shall use information from G and from the tree T and its associated sets S_k only. Before illustrating this with an example, let us give the formal definition.

Definition 4.1 *Let $G = (V, E)$ be a graph with nonempty vertex set $V = \{v_1, \ldots, v_n\}$ and (possibly empty) edge set E. A **tree decomposition** of G is a pair $(T, \mathcal{S})$ where $T = (X, F)$ is a tree graph with node set[3] $X = \{x_1, \ldots, x_m\}$ and edge set F, and $\mathcal{S} = \{S_1, \ldots, S_m\}$ is a set system over V (where the same vertex subset is allowed to occur for more than one of the S_i), indexed according to the nodes of T, that satisfies the following three requirements:*

1. *Every vertex $v_i \in V$ of G occurs in some set $S_k \in \mathcal{S}$.*
2. *The two ends of any edge v_iv_j of G occur together in some set $S_k \in \mathcal{S}$.*
3. *If $v_i \in S_{k'}$ and $v_i \in S_{k''}$ for two indices k', k'', then $v_i \in S_k$ also holds whenever the node x_k is on the $x_{k'}$–$x_{k''}$ path in T.*

The requirements are illustrated in Figure 4.1. The two ends of each edge v_1v_i $(i = 2, 3, 4)$ occur together in the set S_i assigned to node x_i. Since v_1 appears in both S_2 and S_4, it has to be present in the set S_1 between them, too.

The first condition is included only to ensure that the isolated vertices (if there are any) should also appear in the representation. The essence is captured in conditions 2 and 3.

Below we first show how this kind of structure can be created for a given graph G. Then we fine-tune the model for algorithmic purposes and show how it can be applied in solving optimization problems.

We note already at this point that not all chordal extensions are equally good for algorithmic purposes. For example, inserting edges between all nonadjacent vertex pairs we certainly obtain the chordal graph K_n but it would be of no use at all. It would actually lead to a tree decomposition where some set S_k would be identical to the entire vertex set V. We wish just the contrary, to keep the sets S_i as small as possible.

[3] In order to avoid ambiguity, in the present context we use the term "node" for the vertices of the tree T; i.e., the word "vertex" is reserved for the elements of V.

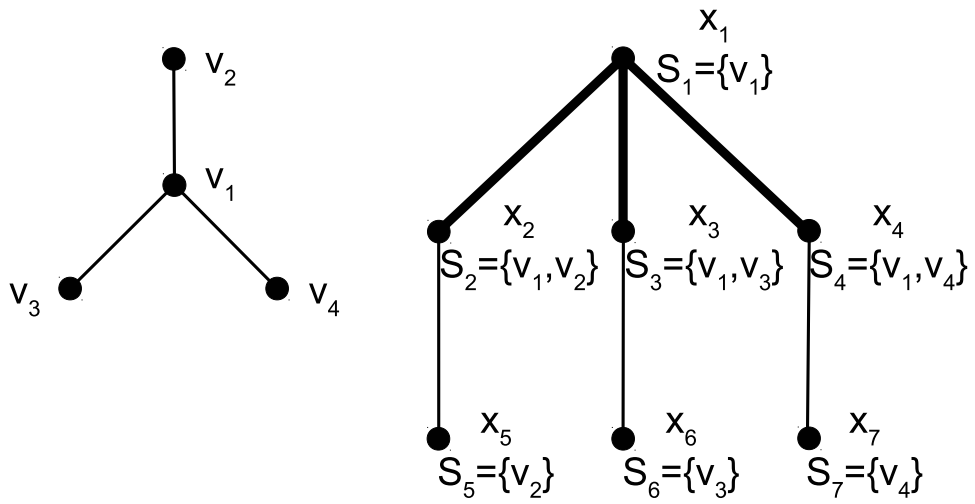


Figure 4.1: $K_{1,3}$ *and a tree decomposition; the subgraph with bold edges indicates the occurrences of* v_1 *in the sets* S_i

Definition 4.2 *The* ***width*** *of a tree decomposition* $(T, \mathcal{S})$ *is*

$$\max_{1 \le k \le |V(T)|} |S_k| - 1.$$

The ***treewidth of*** G*, denoted by* $tw(G)$*, is the smallest possible width of a tree decomposition:*

$$tw(G) = \min_{(T,\mathcal{S}):\ \text{tree decomposition of } G} \ \max_{1 \le k \le |V(T)|} |S_k| - 1.$$

This definition is quite technical and the relevance of its condition will become clear only when we see how the structure can be applied in the design of efficient algorithms. A more plausible alternative approach to treewidth is expressed in terms of *clique number*, as shown by the following result. We shall give the proof at the end of Section 4.1, because it is closely related to the way how a tree representation can be constructed for G according to the scheme above, and we prefer to describe the construction first.

Theorem 4.1 *The treewidth of a graph* G *is equal to*

$$tw(G) = \min_{H:\ \text{chordal},\ G \subseteq H} \omega(H) - 1.$$

The treewidth of a general graph is hard to determine. Nevertheless, for any *fixed* t, it can be checked efficiently whether or not $tw(G) \leq t$ holds; and if it does, then also a tree decomposition of width at most t can be determined. The proof of this result is beyond the scope of the present course.

Graphs with very small treewidth have a transparent structure:

- $tw(G) = 0$ if and only if G has no edges.
- $tw(G) = 1$ if and only if G has no cycles but has at least one edge.
- $tw(G) \geq 2$ for every other graph.

These facts follow directly from Theorem 4.1, and can also be shown without applying it.

We note further that if the initial graph G is chordal, then there is no need to insert new edges, i.e. we may take $H = G$. In this way we obtain:

Theorem 4.2 *If G is a chordal graph, then $tw(G) = \omega(G) - 1$.*

Since the chromatic number of a chordal graph is equal to its clique number, this theorem equivalently means that $tw(G) = \chi(G) - 1$ holds whenever G is chordal.

Trees are particular cases: they have clique number 2 — as well as chromatic number 2 — and treewidth 1, assuming that there are at least two vertices. The assertion of Theorem 4.2 does not extend from trees to bipartite graphs, however; already the even cycles have treewidth 2.

Remark 4.1 *The reason for writing $|S_k| - 1$ in the definition of $tw(G)$ rather than $|S_k|$ is that we wish trees have treewidth 1 rather than 2.*

4.1 Creating a tree decomposition

As we indicated at the beginning of this chapter, a tree decomposition can be obtained for a given graph G in three steps. We illustrate the method with the following graph:

$$G = (V, E), \qquad V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\},$$

$$E = \{v_1v_2, v_1v_4, v_2v_3, v_2v_4, v_3v_6, v_4v_5, v_4v_6, v_5v_7, v_6v_8, v_7v_8\}.$$

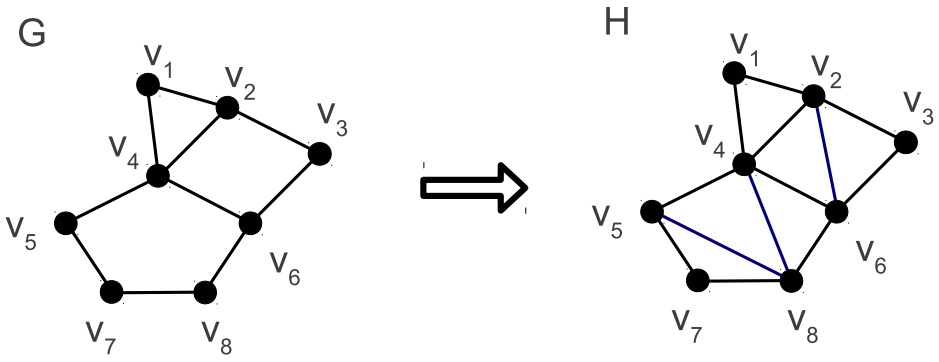


Figure 4.2: *First step of creating a tree decomposition: making the graph chordal*

1. Finding a chordal supergraph H of G. This step is easy to describe: edges have to be inserted into the graph as long as it contains some chordless cycles longer than 3. In our example, by inserting the edges

$$v_2v_6,\ v_4v_8,\ v_5v_8$$

we eliminate all chordless cycles, as shown in Figure 4.2.

Note that there are several ways to achieve this goal, for instance in our example the other diagonal v_3v_4 might also have been chosen in the 4-cycle $v_2v_3v_6v_4$, and/or any two non-crossing chords (or more) of the 5-cycle $v_4v_5v_7v_8v_6$ could have been taken.

Insertion of edges may create new chordless cycles, hence this step needs some care. Keeping in mind that the resulting graph has to be chordal, and that the chordal graphs are precisely the graphs admitting a simplicial order, a systematic way for a chordal extension is to create a simplicial order for H and to insert precisely those edges which are missing in the assumed order. In our example the order $v_1v_3v_2v_6v_4v_5v_7v_8$ corresponds exactly to the insertions chosen.

2. Finding a tree representation. This can be done in the way described in the previous chapter. In our example, we may take for instance the host tree with vertex set $\{x_1, x_2, x_3, x_4, x_5, x_6\}$ and edge set $\{x_1x_2, x_1x_4, x_2x_3, x_4x_5, x_4x_6\}$ as illustrated in Figure 4.3. If we denote by T_i the vertex set of the subtree

that represents v_i, a possible choice is:

$$\begin{aligned} T_1 &= \{x_5\} \\ T_2 &= \{x_4, x_5, x_6\} \\ T_3 &= \{x_6\} \\ T_4 &= \{x_1, x_2, x_4, x_5\} \\ T_5 &= \{x_2, x_3\} \\ T_6 &= \{x_1, x_4, x_6\} \\ T_7 &= \{x_3\} \\ T_8 &= \{x_1, x_2, x_3\} \end{aligned}$$

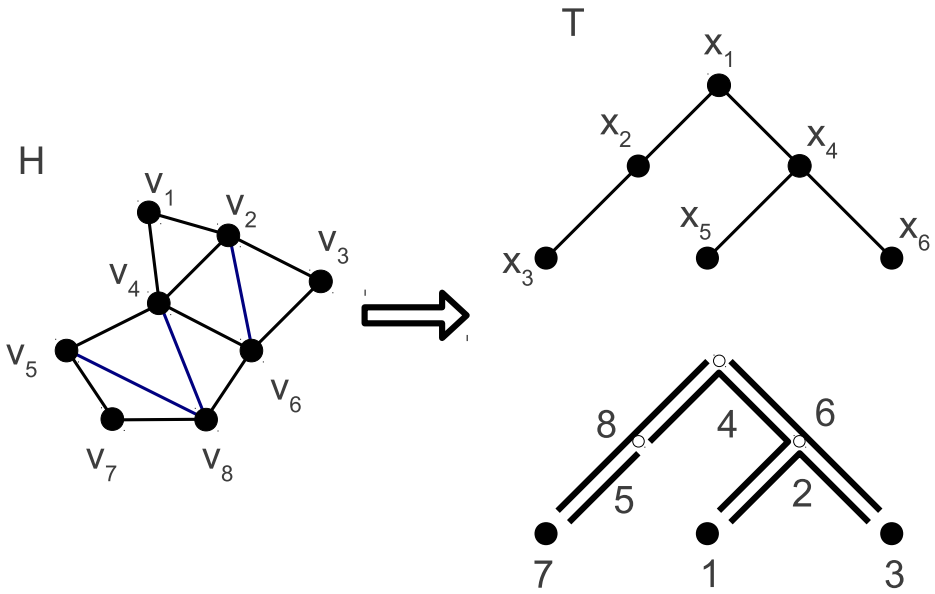


Figure 4.3: *Second step of creating a tree decomposition: finding subtrees in some tree, whose intersection graph is the chordal graph*

As we have seen in Section 3.2, subtree representations are constructed in inverse simplicial order. In our current example, it is derived from the following simplicial order of H: $v_3, v_1, v_2, v_7, v_5, v_4, v_6, v_8$.

3. Finding the sets S_k. Formally, this step can be done by setting $\mathcal{S} = \{S_1, \ldots, S_m\}$ where

$$S_k := \{v_i \mid x_k \in T_i\}$$

for all $1 \leq k \leq m$. That is, in the set assigned to x_k we list the vertices of G with the indices of subtrees containing x_k, as illustrated in Figure 4.4.

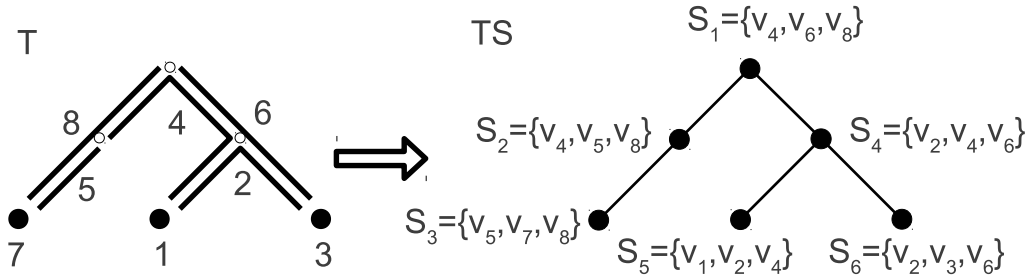


Figure 4.4: *Third step of creating a tree decomposition: assigning sets to the nodes of the tree*

Lemma 4.1 *The pair $(T, \mathcal{S})$ constructed above satisfies the requirements of tree decompositions.*

Proof: Every tree representing the vertices is nonempty, therefore each vertex of G occurs in at least one S_i, verifying the first condition. If v_iv_j is an edge of G, then it is an edge of H as well. The definition of intersection graph then implies that T_i and T_j share a vertex, say x_k. And then v_i and v_j occur together in S_k, according to the construction. This ensures that the second condition holds. Finally, the occurrences of any v_i in the sets S_k correspond to the nodes x_k which are contained in the subtree T_i. That is, those occurrences form a subtree of T, implying that the entire path connecting any two of them is inside the set of occurrences. This verifies the third condition. □

Now we are in a position to prove Theorem 4.1.

Proof of Theorem 4.1: The term '-1' occurs in both formulas, therefore the quantitative equivalence of the two approaches can be simplified to showing that the smallest possible cardinality of the largest set S_k in a tree decomposition $(T, \mathcal{S})$ of G is precisely the minimum clique number of H where H is a chordal supergraph of G.

Assume first that $H \supseteq G$ is a chordal graph with as small clique number as possible. We prove that all the numbers $|S_k|$ in some tree decomposition of G can be kept to be at most $\omega(H)$. Let us represent H as the intersection graph of subtrees $T_1, \ldots, T_n$ of a tree T. Subtrees containing any one node of T correspond to mutually adjacent vertices of H, that is a complete subgraph. For this reason, every node x_k of T occurs in at most $\omega(H)$ subtrees. Hence,

the three-step construction described above yields $|S_k| \leq \omega(H)$ for all nodes x_k of T.

Conversely, assume that $(T, \mathcal{S})$ is a tree decomposition of G, in which the largest value of $|S_k|$ is as small as possible. For $1 \leq i \leq n$, we consider the subtree T_i formed by the occurrences of v_i in the sets assigned to the nodes of T. This is indeed a subtree for every i, by condition 3 of tree decompositions. Denote by H the intersection graph of $\{T_1, \ldots, T_n\}$, where "intersection" means to share a node. We know from the previous chapter that H is chordal, moreover condition 2 implies that $G \subseteq H$ also holds.

Let K be the vertex set of a clique of cardinality $\omega(H)$ in H. Any two vertices of H are adjacent, therefore any two of the corresponding subtrees share a node. By the Helly property of subtrees we obtain that all the $|K|$ subtrees representing the vertices of K have a nonempty intersection, say all of them contain node x_k. Consequently, $\omega(G) \leq |S_k|$ holds, completing the proof of the theorem. □

Remark 4.2 *We have constructed a tree decomposition from a chordal supergraph. The transformation can be done in the other direction, too. Indeed, if $(T, \mathcal{S})$ is a tree decomposition of G, then we obtain a chordal graph $H \supseteq G$ by inserting edges inside the subsets S_k in such a way that all of them become complete subgraphs. This yields a chordal H whose clique number is equal to the width of $(T, \mathcal{S})$ plus 1. Graph H is the intersection graph of the collection $\mathcal{T}$ of subtrees, where $T_i \in \mathcal{T}$ is the set of nodes x_k such that $v_i \in S_k$. (Insertions of edges not contained in any S_k may generate further chordless cycles and therefore may lead to larger clique number in extending G to a chordal graph. This is the reason why we do not insert any such edge.)*

4.2 Nice tree decomposition

Here we introduce a particular kind of tree decompositions, which helps to design algorithms in a more transparent way.

Definition 4.3 *Suppose that $(T, \mathcal{S})$ is a tree decomposition of G. It is called a **nice tree decomposition** if the following further conditions are met, too:*

1. *Viewing T as a rooted tree (with a suitably chosen root) every node of T has at most two children.*

2. *Each node is one of the following four types:*

- ***start node***: *x_k has no children*[4] *(it is a leaf of T);*
- ***forget node***: *x_k has exactly one child, say $x_{k'}$, and $S_k = S_{k'} \setminus \{v_i\}$ for some $v_i \in V$;*
- ***introduce node***: *x_k has exactly one child, say $x_{k'}$, and $S_k = S_{k'} \cup \{v_i\}$ for some $v_i \in V$;*
- ***join node***: *x_k has exactly two children, say $x_{k'}$ and $x_{k''}$, and $S_k = S_{k'} = S_{k''}$.*

See Figure 4.5 for an illustration.

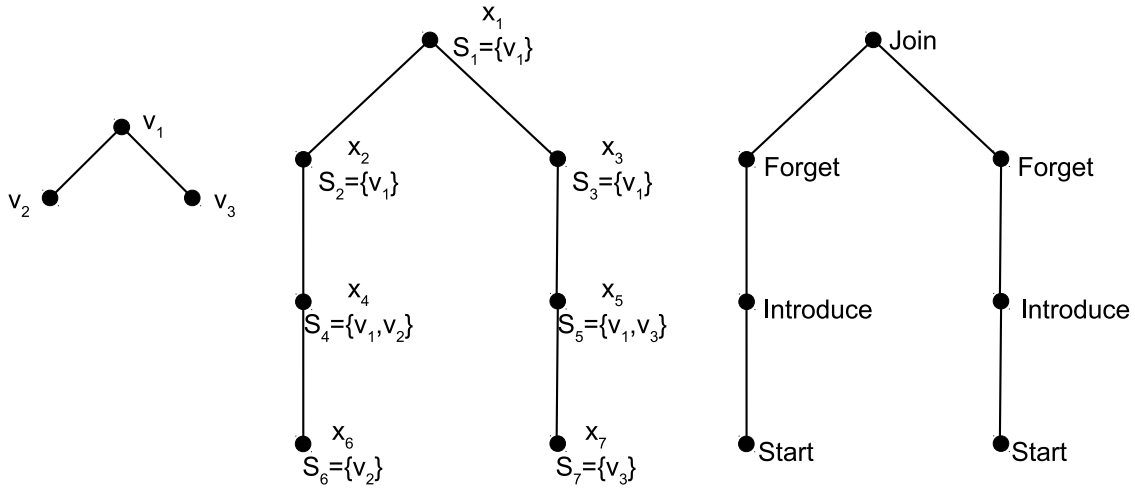


Figure 4.5: *A nice tree decomposition of P_3, and the types of its nodes*

The explanation for those names is that we shall traverse the tree in a bottom-up way, from its leaves towards the root, and in this direction when we move e.g. to a "forget" node from its unique child, the associated set becomes smaller by one, which may be interpreted as making a vertex v_i "forgotten". The reason for the term "introduce" is analogous.

Remark 4.3 *From any given tree decomposition of G, one can construct a nice tree decomposition of the same width.*

The method is illustrated in Figure 4.6 on a binary tree with six nodes.

[4] In some part of the literature, $|S_k| = 1$ is required and the term ***leaf node*** is used. An advantage of this further condition is that it makes the beginning of computation trivial; but on the other hand it does not decrease the total number of steps needed in the entire algorithm. From our model with the current definition of start node, one can create the situation with $|S_k| = 1$ for all start nodes, by inserting a sequence of 'introduce' nodes.

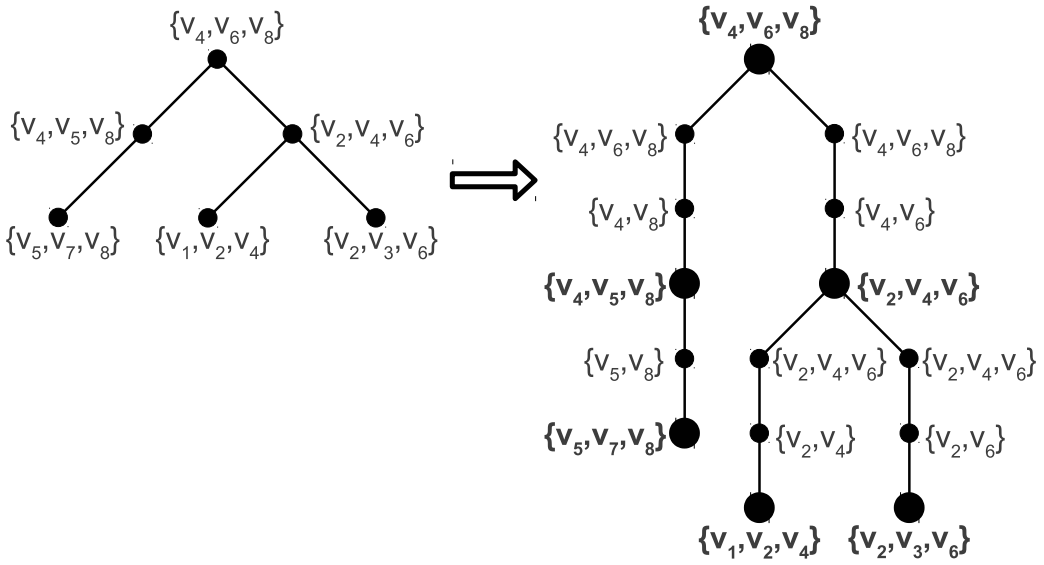


Figure 4.6: *Transforming a tree decomposition to a nice tree decomposition*

Consider for instance the edge between $S_2 = \{v_4, v_5, v_8\}$ and $S_3 = \{v_5, v_7, v_8\}$. An additional node is inserted on this edge whose set is $S_2 \cap S_3 = \{v_5, v_8\}$. Essentially the same is done between S_4 and S_5, and also between S_4 and S_5; but the two sets $S_4 \cap S_5 = \{v_2, v_4\}$ and $S_4 \cap S_6 = \{v_2, v_6\}$ are different, therefore they cannot occur directly under the 'join' node x_4. For this reason we repeat S_4 under the 'join' node in both branches and only then we insert the nodes assigned with the intersections.

A similar operation can be done also in cases where the intersection of adjacent sets is smaller. For example, should we have an edge $x_i x_j$ in the tree T with $S_i = \{v_1, v_2, v_3, v_4\}$ and $S_j = \{v_4, v_5, v_6\}$, we can insert four new nodes between x_i and x_j. Then, together with the sets S_i and S_j, the sequence of sets in the extended tree is:

$$S_i = \{v_1, v_2, v_3, v_4\}, \{v_2, v_3, v_4\}, \{v_3, v_4\}, \{v_4\}, \{v_4, v_5\}, \{v_4, v_5, v_6\} = S_j.$$

Of course, it does not matter whether the second set omits v_1 or v_2 or v_3 from S_i, the essence is to include $S_i \cap S_j$ in the sequence and to create smooth transition to it from both ends. This requires the insertion of

$$|S_i \setminus S_j| + |S_j \setminus S_i| - 1$$

new nodes, one of them assigned to $S_i \cap S_j$, moreover $|S_i \setminus S_j| - 1$ new nodes in the direction of S_i and $|S_j \setminus S_i| - 1$ new nodes in the direction of S_j from the $(S_i \cap S_j)$-node.

One can also modify the tree to a binary one if some node x_k has d children, $d > 2$. We then create two new children x'_k and x''_k for x_k with the same subset $S'_k = S''_k = S_k$. Then we distribute the children of x_k equally between x'_k and x''_k: we remove their direct connection to x_k, and draw new edges so that $\lceil d/2 \rceil$ of them become children of x'_k while the other $\lfloor d/2 \rfloor$ of them become children of x''_k, as illustrated in Figure 4.7.

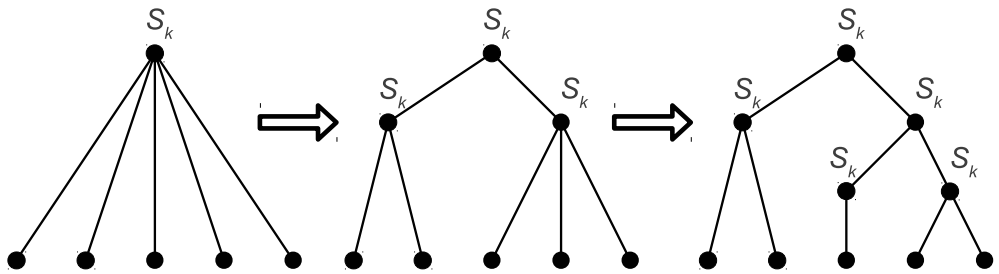


Figure 4.7: *Modifying a tree containing a node with more than 2 children to a binary tree; the newly created intermediate nodes must be assigned to the same vertex subset S_k*

4.3 Example: Largest independent set

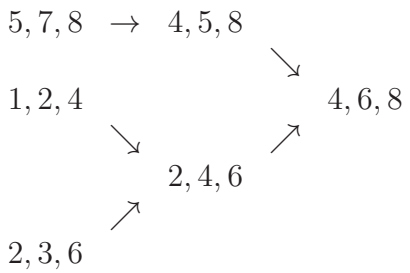
Suppose that we have to solve an optimization problem on a graph G. In this section we illustrate the basic ideas by showing how they work when the maximum size $\alpha(G)$ of independent sets has to be determined; the general method is analogous for any computational problem on G.

Once a nice tree representation $(T, \mathcal{S})$ of G is at hand, we can proceed along the tree T from leaves towards the root; that is, a node s_k is treated only after all of its children have already been processed. There are many possible orders to proceed.[5] On the original six-node tree above (before modifying it to a nice tree decomposition) one may take the nodes, for example, in the order

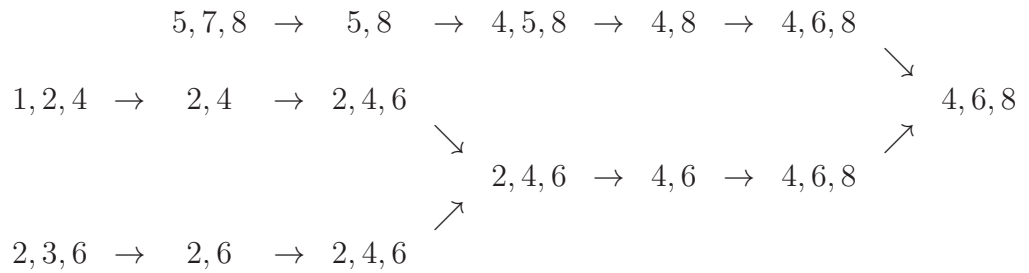
$$x_3,\ x_2,\ x_5,\ x_6,\ x_4,\ x_1.$$

[5] Actually, the feasible orders of this kind are precisely those simplicial orders of T which put the root at the end. This number may be small in some cases, e.g. for a path if one end is the root. But very often it is an exponential function of the number of nodes because we may have two or more possible choices for the next node in each step except the last two. And the number of orders may even be factorial: the star $K_{1,n-1}$ admits any ordering of its leaves. So, the situation is fairly similar to that in the general class of chordal graphs.

The condition of proceeding from the leaves towards the root corresponds to a natural orientation of the tree edges. Writing just the subscripts of nodes instead of the nodes themselves (e.g., '$5, 7, 8$' stands for $\{v_5, v_7, v_8\}$, which is S_3), for the original tree decomposition it means

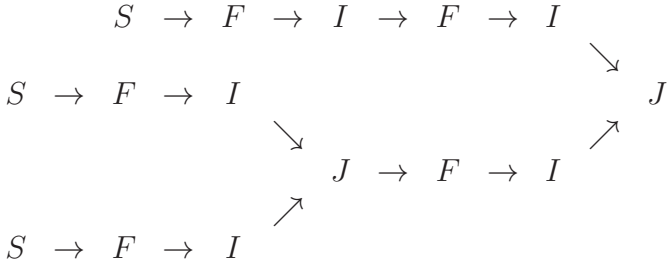


while for the nice tree decomposition derived from it we have the larger structure



on the extended nice tree decomposition. Although the latter seems more complicated for first sight, in fact the computation for each S_i (except for start nodes) will be dramatically simpler than what would be needed without modifying the original tree decomposition to a nice one.

We can observe that the types of nodes occur according to the following scheme:



S: start, F: forget, I: introduce, J: join.

This latter information is not needed explicitly while performing the algorithm, but of course the way of executing each step will depend on the type of the node we currently handle.

For node x_k we consider the *subtree rooted at* x_k in T; we denote this subtree by T^k. It consists of x_k and the nodes from which an edge is oriented towards x_k; the latter are the nodes which must be processed before x_k in any order allowed for the algorithm. We are interested in those vertices of G which appear in the sets S_ℓ of nodes $x_\ell \in T^k$. Let

$$V_k = \bigcup_{x_\ell \in T^k} S_\ell$$

and let G_k be the subgraph of G induced by V_k.

For each node we compute values in a table. The table of x_k has as many rows as the number of independent sets in the subgraph induced by S_k in G. For example, the table of $S_2 = \{v_4, v_5, v_8\}$ contains six rows because, from the eight subsets of S_2, only $\{v_4, v_5, v_8\}$ itself and the pair $\{v_4, v_5\}$ are the two non-independent sets. (The emptyset and the singletons always are independent.)

The tables have three columns. In each row of the table

- the first entry specifies an independent subset $S \subset S_k$;
- the second entry is the maximum cardinality[6] of independent sets I in G_k such that $I \cap S_k = S$, this restriction is substantial with respect to the organization of the computation;
- the third entry is one possible subset S' at the child $x_{k'}$ of x_k, for which an independent set I attaining the maximum with $I \cap S_k = S$ exists and $I \cap S_{k'} = S'$ holds; if x_k is a start node, then it has no children and we write the dummy symbol NIL as third entry in each row of its table.

The algorithm determines these pieces of information for each node in the order given above.

We have to emphasize that the sets S have to be independent in G but not necessarily in the extended graph H. In our example this means

- $S_1 = \{v_4, v_6, v_8\} \;\rightarrow\; \emptyset,\ \{v_4\},\ \{v_6\},\ \{v_8\},\ \{v_4, v_8\}$;
- $S_2 = \{v_4, v_5, v_8\} \;\rightarrow\; \emptyset,\ \{v_4\},\ \{v_5\},\ \{v_8\},\ \{v_4, v_8\},\ \{v_5, v_8\}$;
- $S_3 = \{v_5, v_7, v_8\} \;\rightarrow\; \emptyset,\ \{v_5\},\ \{v_7\},\ \{v_8\},\ \{v_5, v_8\}$;
- $S_4 = \{v_2, v_4, v_6\} \;\rightarrow\; \emptyset,\ \{v_2\},\ \{v_4\},\ \{v_6\},\ \{v_2, v_6\}$;

[6] This maximum can be viewed as a function $\alpha(G_k, S)$ of the subgraph G_k and of the specified subset $S \subset S_k$.

- $S_5 = \{v_1, v_2, v_4\} \ \rightarrow \ \emptyset,\ \{v_1\},\ \{v_2\},\ \{v_4\}$;
- $S_6 = \{v_2, v_3, v_6\} \ \rightarrow \ \emptyset,\ \{v_2\},\ \{v_3\},\ \{v_6\},\ \{v_2, v_6\}$.

Beside these, the nice tree decomposition contains some further sets:

- $\{v_2, v_4\} \ \rightarrow \ \emptyset,\ \{v_2\},\ \{v_4\}$;
- $\{v_2, v_6\} \ \rightarrow \ \emptyset,\ \{v_2\},\ \{v_6\},\ \{v_2, v_6\}$;
- $\{v_4, v_6\} \ \rightarrow \ \emptyset,\ \{v_4\},\ \{v_6\}$;
- $\{v_4, v_8\} \ \rightarrow \ \emptyset,\ \{v_4\},\ \{v_8\},\ \{v_4, v_8\}$;
- $\{v_5, v_8\} \ \rightarrow \ \emptyset,\ \{v_5\},\ \{v_8\},\ \{v_5, v_8\}$.

Next, we describe how the entries of the tables for the nodes can be determined.

Start node. Start nodes have no predecessors in the process, therefore the value α in each row of their tables depends just on the corresponding set S. Since S is independent by assumption, we write $|S|$ as its value, which is indeed the largest cardinality of an independent set I in G_k if x_k is a start node and $I \cap S_k = S$. In fact, under the assumption $I \cap S_k = S$, we have $I = S$ as the unique choice for I, because start nodes have $V_k = S_k$. For example, the table of '5,7,8' is

S	α	S'
$\emptyset$	0	NIL
5	1	NIL
7	1	NIL
8	1	NIL
5,8	2	NIL

Forget node. If x_k is a forget node whose child is $x_{k'}$, and S_k is obtained from $S_{k'}$ by removing the vertex v_i, then a subset $S \subset S_k$ can possibly originate from two subsets of $S_{k'}$: either from S itself or from $S \cup \{v_i\}$. The former always occurs in the table of $x_{k'}$; the latter is there if and only if $S \cup \{v_i\}$ is independent. If it is, then we write the larger of the two corresponding α-values for S in the table of x_k, and in the third column we indicate whether this value originates from the former S of $x_{k'}$ or from $S \cup \{v_i\}$. (It depends on the structure of G, which of them has larger α-value.) If $S \cup \{v_i\}$ is not independent, we just copy the α-value of S from the table of $x_{k'}$ into that of x_k.

In our example, consider the $S \to F \to I$ part of the branch starting with '$5, 7, 8$'. The forget node was obtained by removing v_7 from $S_3 = \{v_5, v_7, v_8\}$. When we calculate the first row in the '$5, 8$' table, the emptyset means that neither v_5 nor v_8 belongs to the independent set I. Under this condition the intersection $I \cap S_3$ can be either $\emptyset$ or the 'forgotten' vertex v_7. In the table of x_3 the former has value 0 and the latter has value 1, the latter being larger. Therefore, in the table of '$5, 8$', we write 1 for α and indicate that the predecessor of $\emptyset$ is $\{v_7\}$. On the other hand, if one or both of v_5 and v_8 are contained in S, then $S \cup \{v_7\}$ is not independent, therefore we just copy the preceding α-value and put S as third entry. This yields

S	α	S'
$\emptyset$	0	NIL
5	1	NIL
7	1	NIL
8	1	NIL
5,8	2	NIL

$\to$

S	α	S'
$\emptyset$	1	7
5	1	5
8	1	8
5,8	2	5,8

A similar situation occurs in computation (exhibited later) when node '$2, 4$' forgets v_1 from '$1, 2, 4$', or '$2, 3$' forgets v_3 from '$2, 3, 6$', and so on. We shall put some further comments on this step near the end of the computation where '$4, 8$' and '$4, 6$' are calculated.

Introduce node. If x_k is an introduce node, and the new element of S_k is v_i, then S may or may not contain v_i. If $v_i \notin S$, then for the predecessor S' of S we necessarily have $S' = S$, hence we just copy the α-value from the S-row of the predecessor table and indicate that it originates from $S' = S$. This is the case of rows '$\emptyset$', '5', '8', and '$5, 8$' in the table of '$4, 5, 8$', where the vertex v_4 is introduced. On the other hand, if $v_i \in S$, we must have $S' = S \setminus \{v_i\}$, and then — since v_i is an additional new element in the independent set — we add 1 to the α-value of the predecessor S' when we compute the S-row for x_k. This is the case of rows '4' and '$4, 8$' in the table of '$4, 5, 8$'.

S	α	S'
$\emptyset$	1	7
5	1	5
8	1	8
5,8	2	5,8

$\to$

S	α	S'
$\emptyset$	1	$\emptyset$
4	2	$\emptyset$
5	1	5
8	1	8
4,8	2	8
5,8	2	5,8

The steps above are applied also to compute the first three tables in the other two branches of the tree, starting with '$1, 2, 4$' and '$2, 3, 6$'. In the

next tables we replace the general top left entry S with the particular set of indices of vertices which S contains. Those two sequences of 'start $\rightarrow$ forget $\rightarrow$ introduce' computation then yield

1,2,4	α	S'
$\emptyset$	0	NIL
1	1	NIL
2	1	NIL
4	1	NIL

$\rightarrow$

2,4	α	S'
$\emptyset$	1	1
2	1	2
4	1	4

$\rightarrow$

2,4,6	α	S'
$\emptyset$	1	$\emptyset$
2	1	2
4	1	4
6	2	$\emptyset$
2,6	2	2

2,3,6	α	S'
$\emptyset$	0	NIL
2	1	NIL
3	1	NIL
6	1	NIL
2,6	2	NIL

$\rightarrow$

2,6	α	S'
$\emptyset$	1	3
2	1	2
6	1	6
2,6	2	2,6

$\rightarrow$

2,4,6	α	S'
$\emptyset$	1	$\emptyset$
2	1	2
4	2	$\emptyset$
6	1	6
2,6	2	2,6

These two branches are connected by a 'join' node, for which we describe the rules of computation next.

Join node. A 'join' node x_k has two children, say $x_{k'}$ and $x_{k''}$. Since $S_{k'} = S_{k''} = S_k$ holds by assumption, we have $I \cap S_{k'} = I \cap S_{k''} = I \cap S_k$ for every independent set I. Hence, for any $S \subset S_k$, we must take $S' = S$ from both predecessor tables, and the α-value for S has to be computed from those of their S-rows. Due to the properties of tree decomposition — especially from property 3 — we see that $V_{k'} \cap V_{k''} = S_k$ and $V_{k'} \cup V_{k''} = V_k$. So, a set $I \subset V_k$ is independent if and only if $S := I \cap S_k$ is independent, moreover both $I \cap V_{k'}$ and $I \cap V_{k''}$ are independent. Consequently, having chosen the set S, we obtain the α-value for S in the table of x_k by *adding the α-values for S in the tables of $x_{k'}$ and $x_{k''}$ and subtracting $|S|$*. The reason for subtraction is that by adding we count the elements of S twice: once in $V_{k'}$ and once in $V_{k''}$. For example, computing for the 'join' node '$2, 4, 6$',

- $\emptyset$ has no repeated element: $1 + 1 - 0 = 2$;
- each of '2', '4', and '6' has exactly one repeated element: $1 + 1 - 1 = 1$, $1 + 2 - 1 = 2$, $2 + 1 - 1 = 2$;
- '$2, 6$' has exactly two repeated elements: $2 + 2 - 2 = 2$.

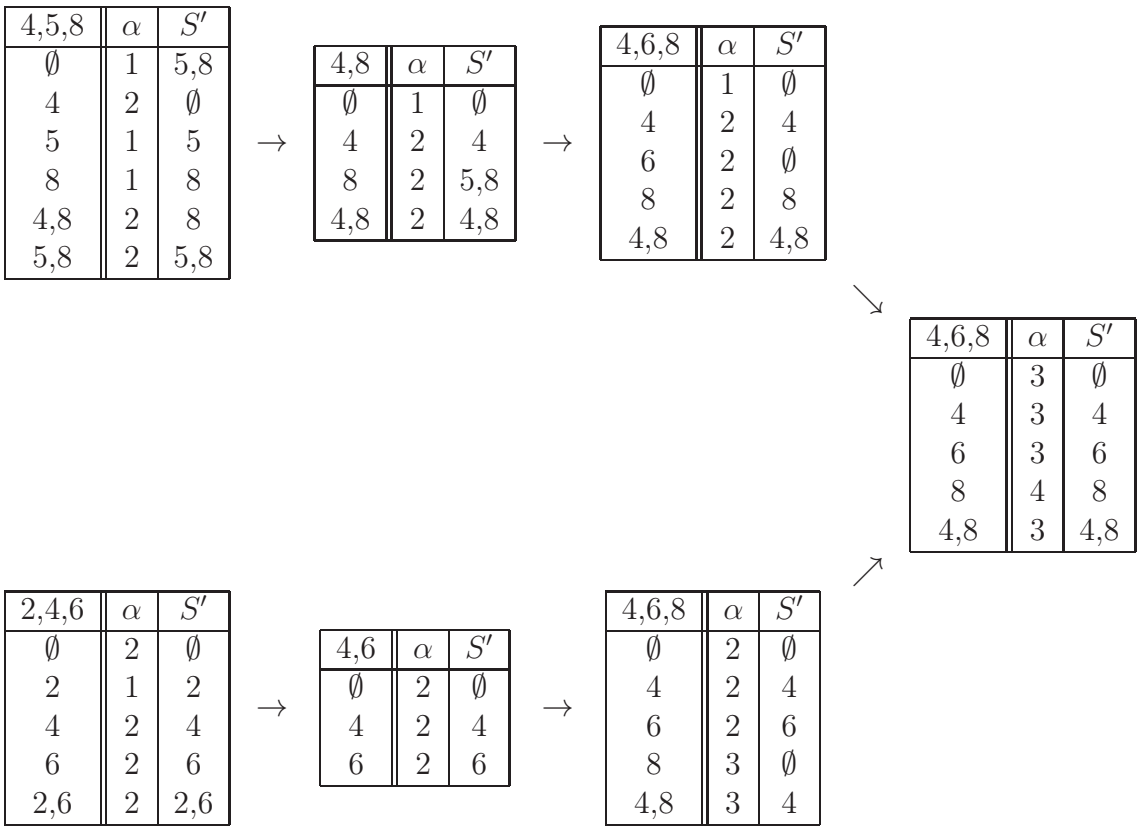
2,4,6	α	S'
$\emptyset$	1	$\emptyset$
2	1	2
4	1	4
6	2	$\emptyset$
2,6	2	2

2,4,6	α	S'
$\emptyset$	2	$\emptyset$
2	1	2
4	2	4
6	2	6
2,6	2	2,6

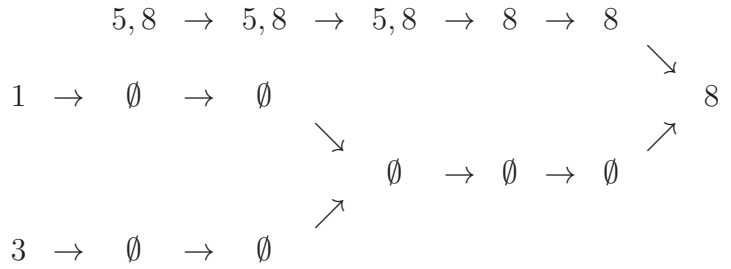
2,4,6	α	S'
$\emptyset$	1	$\emptyset$
2	1	2
4	2	$\emptyset$
6	1	6
2,6	2	2,6

By these rules we can now complete the computation for the rest of the nodes in an analogous way, from the middle forget node '$4, 5, 8$' and from the join node '$2, 4, 6$'. Those two branches meet in the join node '$4, 6, 8$'. It can be observed that the '8'-row of '$4, 8$' may originate from the predecessor '8'-row and also from the '$5, 8$'-row; we choose the latter because its α-value is greater. The '$\emptyset$'-row of '$4, 8$' may also originate from two sources: from '$\emptyset$' itself and from the '5'-row; in the predecessor table both of them have α-value 1, therefore it does not matter whether we take $S' = \emptyset$ or $S' = \{5\}$.

We write the first one.



Determining the optimum value and an optimal set. Having computed all tables, $\alpha(G)$ is equal to the largest α-value in the table of x_1, the root of T. In our case it yields $\alpha(G) = 4$, occurring in the '8'-row. The S'-entry of this row tells us that this value has been achieved by choosing $S' = \{v_8\}$. Looking into the '8'-rows of the predecessor introduce tables, we find there $S' = \{v_5, v_8\}$ and $S' = \emptyset$, respectively. Following those links we can trace back the origin of an optimal solution; in the next chart we exhibit the vertex indices in the corresponding S'-entries:



The set of indices occurring in this scheme tells us that an independent set of cardinality 4 is

$$\{v_1, v_3, v_5, v_8\}.$$

Remark 4.4 *We have written "the" optimum value and "an" optimal set. Indeed, while the value of $\alpha(G)$ is unambiguous, the number of optimal sets (independent sets of maximum cardinality) can be very large. For instance, if n is even, then $n/2$ disjoint copies of K_2 together (that is, the graph with $n/2$ connected components, each of them being just one edge) have exactly $2^{n/2}$ largest independent sets. Even more, if n is a multiple of 3, then $n/3$ vertex-disjoint copies of K_3 together form a graph on n vertices, with $n/3$ connected components and $3^{n/3}$ largest independent sets. In fact, this is the extremum: it is known that every graph on n vertices contains at most $3^{n/3}$ maximal (i.e., non-extendable) independent sets.*

4.4 Graphs of bounded treewidth

Analyzing the running time of the previous algorithm, we can observe that the computation of α-values and the determination of predecessor sets S' in each row of the tables involved is fairly simple. Hence, the overall running time on G essentially depends on the number of rows in the tables. This fact explains why it is substantial to keep the sets S_k small: the corresponding table may be quite large if S_k is large. For instance, if S_k itself is independent, then so are all of its subsets, hence we have $2^{|S_k|}$ choices for the subset S, and that many rows in the table of x_k. For this reason, the worst-case running time of the algorithm is an exponential function of treewidth.

With a more optimistic view, however, we see that if a constant universal upper bound is put on the treewidth of all graphs in a class of graphs to be handled, also the size of tables will be bounded above by a constant. Moreover, it is known that for every fixed t, tree decompositions of graphs of treewidth at most t can be found in cn steps where n denotes the number of vertices and c is an absolute constant. These facts offer us the opportunity to solve many kinds of optimization problems efficiently on fairly large classes of graphs.

An example: Chromatic number. If G has treewidth at most t, then it is a subgraph of a chordal graph of chromatic number t, consequently $\chi(G) \leq t$. Although chromatic number is hard to find in general, it becomes computable in linear time once we restrict the treewidth of input graphs. Theorem 4.4 guarantees this in a very general framework; but when we have

to solve a particular problem, it is more convenient to design an algorithm tailored directly to the problem in question, rather than to formulate it in the language of monadic logic.

Suppose that we have a graph G given together with its nice tree decomposition $(T, \mathcal{S})$, and let the task be to decide whether G admits a proper vertex coloring with three colors. This can be done with the help of tables having *two columns*. Indeed, observe that the partial solutions of this problem are proper 3-colorings of the subgraphs induced by the sets S_k in G. Hence, when studying the 3-colorings of G, we investigate partial 3-colorings they induce on the sets S_k. Since the two vertices of any edge occur together in some S_k (by condition 2 of tree decompositions), it suffices to decide whether there exists a color assignment of three colors to the vertices, such that each color class is independent in each S_k.

This can be done in the following way. The first entry in each row of the table of node x_k is a partition $P = (S', S'', S''')$ of S_k into three independent sets (some of which are allowed to be empty). That is, the table has as many rows as the number of proper colorings of the subgraph induced by S_k in G with at most three colors. The second entry tells us whether this partition belongs to a proper 3-coloring of the subgraph G_k. For an introduce or forget node it is so if and only if P is compatible with some feasible 3-partition of $S_{k'}$ where $x_{k'}$ is the child of x_k in T. We write the corresponding 3-partition P' in the second entry of the row if it exists, and write 'NO' if it doesn't. This 'NO' is of course different from the dummy symbol NIL which we use to indicate that P belongs to a start node. For a join node the 3-partition P is feasible for G_k if and only if it is feasible for both $G_{k'}$ and $G_{k''}$, where $x_{k'}$ and $x_{k''}$ are the children of x_k in T.

For start nodes x_k the step is immediate, every proper 3-coloring of G_k is feasible. The step is simple for join nodes, because we just need to check whether P is feasible for both predecessor sets $S_{k'}$ and $S_{k''}$ of S_k. (Recall that $S_{k'} = S_{k''} = S_k$ is required for any join node of a nice tree decomposition.) For an introduce node, say if $S_k = S_{k'} \cup \{v_i\}$, we obtain P' by removing v_i from its class in P, and then check whether P' is feasible for $S_{k'}$. The situation may be more complicated for a forget node, say with $S_k = S_{k'} \setminus \{v_i\}$. Depending on adjacencies between v_i and $S_{k'}$, we have to consider three partitions:

$$(S' \cup \{v_i\}, S'', S'''), \quad (S', S'' \cup \{v_i\}, S'''), \quad (S', S'', S''' \cup \{v_i\}).$$

Then P is feasible for S_k if and only if at least one of these three partitions is feasible for $S_{k'}$. The entire graph is 3-colorable if the table of the root node contains at least one feasible partition.

Algorithms without generating nice tree decompositions. Nice tree decomposition simplifies the computation of tables for the nodes. Nevertheless, the original notion of tree decomposition without the additional restrictions on node types is often good enough for designing efficient algorithms. If S_k and $S_{k'}$ are sets assigned to adjacent nodes, then proceeding from $x_{k'}$ to compute row S for x_k (where S is a partial solution on the subgraph induced by S_k) we have to consider all those partial solutions S' on $S_{k'}$ which are compatible with S; that is, the restrictions of S and S' to the subset $S_k \cap S_{k'}$ must be the same. Choosing the best S' for each child $x_{k'}$ of x_k, their combination yields the optimum for S. There are some optimization problems, however, for which in order to have a polynomial-time algorithm one needs to assume that the degrees in T are bounded.

4.5 Tree decompositions of small width

We shall discuss the wide applicability of tree decompositions in the area of efficient algorithms in the Notes section at the end of this chapter. Before that, let us mention some further properties of graphs whose treewidth does not exceed a fixed upper bound.

Interestingly enough, for every fixed t it can be checked efficiently — even in linear time, where the number of steps is proportional to the number of vertices — whether a given graph has treewidth at most t. The algorithm that tests $tw(G) \leq t$ and finds a tree decomposition of minimum width for such graphs is quite sophisticated, and is beyond the scope of these lecture notes.

The next result describes a useful structural property concerning the existence of small vertex cutsets.

Theorem 4.3 *Let G be a connected graph with n vertices. Then every tree decomposition of G contains a node x_k such that each connected component of $G - S_k$ has fewer than $n/2$ vertices.*

Proof: Let $(T, \mathcal{S})$ be a tree decomposition of $G = (V, E)$. Although not required formally by definition, we may assume without loss of generality that $S_i \neq \emptyset$ holds for all sets at the nodes x_i of T. For any edge $e = x_{x'}x_{k''}$ of T, consider the set $S_e = S_{k'} \cap S_{k''}$. We have $Y_e \neq \emptyset$ because G is connected. The graph $T - e$ has precisely two components, denote them by T' and T'' (both are trees).

Due to conditions 1 and 3, each vertex $v_i \notin S_e$ of G occurs in some sets at the nodes of T' or of T'', but not of both. That is, disregarding the elements of Y_e, we have $V' \cup V'' = V \setminus Y_e$ and $V' \cap V'' = \emptyset$ for the set V' of vertices

occurring in the sets of T' and the set V'' of vertices occurring in the sets of T''. If $|V'| < |V''|$, we orient the edge e from $x_{x'}$ to $x_{k''}$; and if $|V'| > |V''|$, we orient it from $x_{x''}$ to $x_{k'}$. In case of $|V'| = |V''|$ we orient e towards the root of T.

There must be a node x_k from which no edges are oriented outward. Indeed, should all vertices have positive out-degree, we would find a directed cycle in T, but this is impossible because T is a tree. We claim that S_k satisfies the requirements of the lemma. For any component G_i of $G - S_k$, all of its vertices occur in the sets of just one and the same tree component of $T - x_k$. Let $x_{k'}$ be any neighbor of x_k, and consider the edge $e = x_k x_{k'}$. By the choice of the orientation, at most $\frac{n-|Y_e|}{2}$ vertices different from the ones in S_k occur in the subtree containing $x_{k'}$ in $T - e$. Thus, $(n-1)/2$ is an upper bound on the number of vertices in G_i. □

A node with the property described in the theorem does not always exist in tree decompositions of disconnected graphs. Counterexamples occur when the connected components can be grouped into two parts of $n/2$ vertices each. On the other hand, for a disconnected graph G with c connected components $G_1, \ldots, G_c$, a tree decomposition can be determined componentwise, and we have

$$tw(G) = \max_{1 \le i \le c} tw(G_i).$$

We prove one more result, showing that the tree T in the tree decomposition can be chosen to be fairly small. This property is important in guaranteeing a linear upper bound on the running time of algorithms designed for graphs of bounded treewidth.

Lemma 4.2 *Every graph G on n vertices has a nice tree decomposition of width $tw(G)$ over a tree having fewer than $4n$ nodes.*

Proof: It suffices to show the validity of the assertion for chordal graphs. At the end of Section 3.2 we described a method to find subtrees $T_1, \ldots, T_n$ of a tree, whose intersection graph is a given chordal graph G. As one can see, each step of that procedure requires at most one new node for the tree T, and the newly inserted subtree has a single node. Moreover, there is a natural correspondence between the collection $(T_1, \ldots, T_n)$ of subtrees and the system $\mathcal{S} = \{S_1, \ldots, S_m\}$ of associated sets. We can modify the steps in a way that a nice tree decomposition is obtained at the end.

We view T as a rooted tree, already from the beginning of the process. While handling vertex v_i, one possibility is that T remains unchanged and T_i is created on some node, say x_k, which is already in T. For the tree decomposition it means that the set S_k of node x_k is extended as $S_k^+ =$

$S_k \cup \{v_i\}$. Then we can modify T by replacing x_k with the chain x'_k, x''_k, x'''_k and assign the respective associated sets $S_{k'} = S_k$, $S_{k''} = S_k^+$, $S_{k'''} = S_k$. The parent of x'_k is the original parent of x_k, while the child(ren) of x'''_k is (are) the original child(ren) of x_k. Then x'_k is a forget node, x''_k is an introduce node, and the type of x'''_k is the same as that of the original x_k. We have inserted two nodes for the one vertex v_i.

The modification is somewhat more complicated if v_i requires a new node x_j for T with some set $S \cup \{v_i\}$ (with $|S| = d^+(v_i)$, the forward degree of v_i in the simplicial order of G). Say, x_j becomes adjacent to x_k, where $S \subseteq S_k$. Since a join node has children assigned to the same subset of vertices, we may assume that x_k is not a join node.

We denote by $x_{k'}$ the child of x_k if x_k is not a leaf in the current T. Hence, for the moment, x_k has either just one child x_j or exactly two children x_j and $x_{k'}$. Next we insert one new node on each edge from x_k to its child(ren), with the same set S_k. If $S = S_k$, then this extension of T remains a nice tree decomposition with at most three new nodes (x_j and the at most two newly inserted children of x_k).

If $S \neq S_k$, then we select a vertex $v_\ell \in S_k \setminus S$, and insert a further new node as parent of x_j, so that a path with nodes x_k, x'_k, x''_k, x_j occurs; we assign the nodes of this path to the sets S_k, S_k, $S_k \setminus \{v_\ell\}$, and $(S_k \setminus \{v_\ell\}) \cup \{v_i\}$ in this order. Hence, it may happen that a set with more than $|S|+1$ elements is assigned to x_j in this case; but the set is not larger than S_k, therefore the width of the decomposition is not increased.

In this way, three new nodes have been inserted on a new branch starting from x_k: forget node x'_k, introduce node x''_k, and start node x_j; moreover there is possibly a fourth new node on the edge to the child of x_k if x_k is not a start node. Then this fourth new node has the same type as x_k originally had, and the type of x_k gets updated to join node.

For the first vertex v_1, the three T has just one node x_1, hence $|V(T)| < 4n$ holds if $n = 1$. Then, the assertion follows by induction on n. □

4.6 Notes

4.6.1 Traversing a rooted tree

A basic task in the area of graph algorithms is to traverse the nodes of a rooted tree. In the algorithms of this chapter we applied two kinds of them:

postorder traversal: traverse a node *after* all its children are traversed;

preorder traversal: traverse a node *before* all its children are traversed.

Both ways can be implemented in time proportional to the number of nodes.

Applying the machinery of tree decompositions, we proceeded in post-order while computing the optimum value, and after that in preorder while determining an optimal solution.

4.6.2 Monadic logic

The algorithmic applicability of tree decomposition can be demonstrated in the framework of logic. Within first-order logic — which is included in most curricula — the universal and existential quantifiers $\forall$, $\exists$ can be used on individual variables (variables that range over individuals) only. Second order logic allows the use of quantifiers not only over individuals but also over n-ary relations. Particularly, in *monadic second-order* logic quantifications over individuals and over sets of individuals (unary/monadic relations) are used.

When graphs are considered as logical structures, we are given a set V of vertices (individuals) and a binary (adjacency) relation $Adj(x, y)$ over V. In the monadic second-order logic we can use individual variables $x, y, z, \ldots$ referring to vertices, and set variables $X, Y, Z, \ldots$ referring to subsets of V. For example, the condition that Y is an independent set in graph $G = (V, E)$, can be expressed as follows:

$$\forall x \forall y[((x \in Y) \wedge (y \in Y)) \rightarrow \neg\, Adj(x, y)].$$

In fact, this formalization still fits the framework of first-order logic, as '$\forall$' is applied just on the individual variables x and y. Note that testing whether a given set is independent requires no more than linear time, even without knowing anything about the structure of the graph. This kind of description, however, is not strong enough to deal with the maximum cardinality of independent sets; we need higher-order logic for that.

A sufficient condition for the applicability of the algorithmic machinery based on tree decompositions is given in the following general result.

Theorem 4.4 *If a computational problem on graphs is expressible in monadic second-order logic, then on any class of graphs with bounded treewidth it is solvable in linear time, that is in Cn steps for some absolute constant C, where n denotes the number of vertices in the input graph.*

Among many further properties, 3-colorability can also be described in terms of monadic second-order logic.

$$\begin{aligned}\exists\, X\, \exists\, Y\, \exists\, Z\, [&\forall x(x \in X \ \vee\ x \in Y \ \vee\ x \in Z) \wedge \\ &\forall x \forall y(((x \in X \ \wedge\ y \in X) \vee (x \in Y \ \wedge\ y \in Y) \vee (x \in Z \ \wedge\ y \in Z)) \\ &\qquad \rightarrow\ \neg Adj(x, y))].\end{aligned}$$

The optimization problem of determining the smallest possible number of colors in a proper vertex coloring — that is, to determine the chromatic number — of a graph in a class of graphs of bounded treewidth can also be expressed using monadic second-order logic. The reason is that $\chi(G) \leq tw(G)+1$ holds for all graphs, therefore if treewidth is bounded then chromatic number is bounded, too. Hence, k-colorability has to be tested for a bounded range of k only, and this can be handled with a bounded number of symbols in a logical formula.

One can see from these examples, however, that explicit algorithms are more transparent than those derived from the framework of logic. Expressing an algorithmic problem within monadic second-order logic is useful in a situation where we want to show that the problem in question admits an efficient solution but we do not need to see an algorithm itself.

Chapter 5

Bipartite graphs

We recall that bipartite graphs have been defined as those graphs whose vertex sets can be partitioned into two independent sets. This means that the graph has a proper vertex coloring with two colors (or, with just one color if the graph has no edges; but we will not deal with this trivial case in the current section). An equivalent characterization can be given in terms of cycle lengths.

Theorem 5.1 *A graph is bipartite if and only if it contains no cycles of odd lengths.*

This characterization leads to an efficient recognition algorithm for bipartite graphs.

Theorem 5.2 *Bipartite graphs can be recognized in time proportional to the number of vertices and edges.*

Sketch of Proof: A graph $G = (V, E)$ is bipartite if and only if so is each of its connected components. There are many fast ways to traverse the components of a graph; here we apply the one called Breadth-First Search. It is known to admit a linear-time implementation, that is in at most constant times $|V| + |E|$ steps.

If G is connected, we choose a starting vertex, say v, and create the set $V_0 = \{v\}$. The vertices of G will be listed in increasing order of their distance from v. So, the set V_1 consists of all neighbors of v. Further, recursively, if $V_0, \ldots, V_{i-1}$ are already at hand and $V_0 \cup \cdots \cup V_{i-1} \neq V$, then let V_i be the set of those vertices which are not in $V_{i-1} \cup V_{i-2}$ but have at least one neighbor in V_{i-1}. If all those sets are independent, then G is bipartite; and if at least one of them is not independent, then G contains an odd cycle and hence is not bipartite. In the former case, we obtain a vertex bipartition into two

independent sets by taking the union of subsets whose indices have the same parity; that is, $V_0 \cup V_2 \cup \cdots$ and $V_1 \cup V_3 \cup \cdots$.

For disconnected graphs we run the same algorithm, starting from an arbitrarily chosen vertex, as long as the next V_i is not empty. If the procedure stops but not all vertices have been visited so far, we continue in the same way with another arbitrarily chosen vertex which has not yet been taken. We need to perform this as many times as the number of connected components in G. □

Our main concern in this chapter is to find matchings which are optimal under some criteria. We shall mostly deal with matchings of maximum size in bipartite graphs; but a different kind of optimality will also be considered in a more general model. Moreover, in one of the sections we present some applications of the methods developed for maximum matchings.

5.1 Maximum matchings in bipartite graphs

We know from the early part of this course that the inequality

$$\tau \geq \nu$$

between transversal number and matching number is valid in a very general form, in every set system. It holds with equality in some cases (e.g. in interval systems) but one can say that structure classes satisfying $\tau = \nu$ are rare. One simple infinite sequence for which $\tau > \nu$ holds with strict inequality is that of odd cycles. Indeed, $\tau(C_{2t+1}) = t + 1$, while $\nu(C_{2t+1}) = t$ for all integers $t \in \mathbb{N}$.

It is quite interesting that the exclusion of odd cycles implies $\tau = \nu$ in graphs.[1] This is expressed in the following important result.

Theorem 5.3 (Kőnig's Theorem) *If a graph G is bipartite, then $\tau(G) = \nu(G)$ holds. Moreover, a matching of maximum size and a transversal of minimum cardinality can be found in bipartite graphs efficiently.*

Proof: Let $G = (V, E)$ be a bipartite graph with vertex bipartition $A \cup B = V$. The scheme of the algorithm is:

- Find a maximal matching.

[1] In this way Theorem 5.3 leads to a characterization in the following sense: All subgraphs H of a graph G satisfy the equality $\tau(H) = \nu(H)$ *if and only if* G contains no odd cycles.

- Enlarge the matching by a well-defined kind of transformation.

- When no more improvement is possible, prove that the matching found is maximum and also a minimum transversal set can be determined from it.

The crucial point is how we design the improving transformation. It will turn out that the phase of generating a maximal matching can also be viewed as a particular case of the second phase; but it is more transparent if we separate the two from each other. At any moment of the algorithm, the currently available matching will be denoted by M.

A non-extendable matching can be found in a very simple way. We can take the vertices in the first vertex class A in any order, select an edge incident with the first vertex and then for every subsequent vertex we select an edge which does not meet any edges selected earlier, if such an edge incident with the current vertex exists. At the end of this procedure we reach a matching M which is maximal, but there is no guarantee that it is maximum.[2]

Assume that we have reached a non-extendable matching M. If the edges of M cover the entire vertex class A, then of course M is a maximum matching because there is no room for more than $|A|$ mutually disjoint edges in G. Also, the $|A|$ vertices of A meet all edges of G. Hence, in this case we have $\tau(G) = \nu(G) = |A|$, moreover the maximum matching M and the minimum transversal set A have been found in at most $|V| + |E|$ steps. The situation is similar if M covers the entire vertex class B.

If this is not the case, then we consider the set $X \subset A$ of vertices not contained in the edges of M, and the set $Y \subset B$ of vertices not contained in the edges of M. Since M is maximal, no edges connect X with Y; but G may have several edges from X to $B \setminus Y$, from Y to $A \setminus X$, and also edges different from those in M which connect $A \setminus X$ with $B \setminus Y$. We introduce the following important notions.

Definition 5.1 *With respect to a matching M, an **alternating path** is a path in which every second edge belongs to M and every other edge does not belong to M. An **augmenting path** is an alternating path whose two ends are not contained in the edges of M.*

[2] For example, if G is the graph with vertex set $\{a_1, a_2, b_1, b_2\}$ and edge set $\{a_1b_1, a_1b_2, a_2b_2\}$ (which means the path $b_1a_1b_2a_2$ of length three), and we begin with the edge a_1b_1, then greedy selection will find the matching $M = \{a_1b_1, a_2b_2\}$ of two edges, which is largest; but if we begin at a_1 with the other edge, a_1b_2, then we end up with the single-edge matching $M = \{a_1b_2\}$, which is non-extendable but certainly is not of maximum cardinality.

For example, if $ab \in M$ is an edge of the matching for some vertices $a \in A$ and $b \in B$, and xb is an edge of G for some $x \in X$, then xb is an alternating path of length one and xba is an alternating path of length two. Note that the notion of alternating path heavily depends on the choice of M.

For our purpose, alternating paths from X to Y, that is $x_1x_2 \dots x_m$ with $x_1 \in X$ and $x_m \in Y$ (for any m) are of great importance. The reason why such a path is said to be augmenting is as follows. In this situation the number m of vertices in the path is even, and the length of the path is odd. We can then modify M to obtain a larger matching, by switching M-edges and non-M-edges along the path:

$$M := (M \setminus \{x_{2i}x_{2i+1} \mid 1 \leq i \leq m/2 - 1\}) \cup \{x_{2i-1}x_{2i} \mid 1 \leq i \leq m/2\}.$$

This transformation is illustrated in Figure 5.1.

During this transformation, the number of M-edges contained in the path increases from $m/2 - 1$ to $m/2$, while X and Y shrink to $X \setminus \{x_1\}$ and $Y \setminus \{x_m\}$, respectively. Observe further that M remains a matching, because we removed from the original M all the edges incident with the internal vertices of the path, and the two ends were not covered by M originally.

Suppose that no more improvements of this kind are possible; i.e., none of the alternating paths with respect to the current M is augmenting. We now find a transversal set T in the form $T = T_A \cup T_B$, where $T_A \subseteq A$ and $T_B \subseteq B$. If M covers the entire vertex class A, then $T_A = A$ and $T_B = \emptyset$. Otherwise, if $X \neq \emptyset$, we define T_B as the set of those vertices in B which can be reached from X along alternating paths. Then, let T_A be the set of those vertices in $A \setminus X$ which *cannot be reached* from X along alternating paths.

Claim: We have $|T| = |M|$.

Proof: It suffices to prove — as $T \cap (X \cup Y) = \emptyset$ — that we have selected precisely one vertex for T from each edge of M. This can be seen as follows. Since G is bipartite, every B-vertex of an alternating path is reached along a non-M-edge, and every A-vertex is reached along an M-edge. This is obvious for the first edge, and then follows by the requirement of alternation. Hence, if there exists an alternating path from X to the B-vertex of an edge $e \in M$, then the path can be continued to the A-vertex of e, too (and this is the only way to continue, i.e. the corresponding A-vertex did not occur in the path before). Consequently, if some $e \in M$ is reachable from X along alternating paths, then precisely the B-vertex is selected for T; and otherwise its A-vertex is selected. This proves the claim. $\Diamond$

Moreover, we have:

Claim: The set T is a transversal.

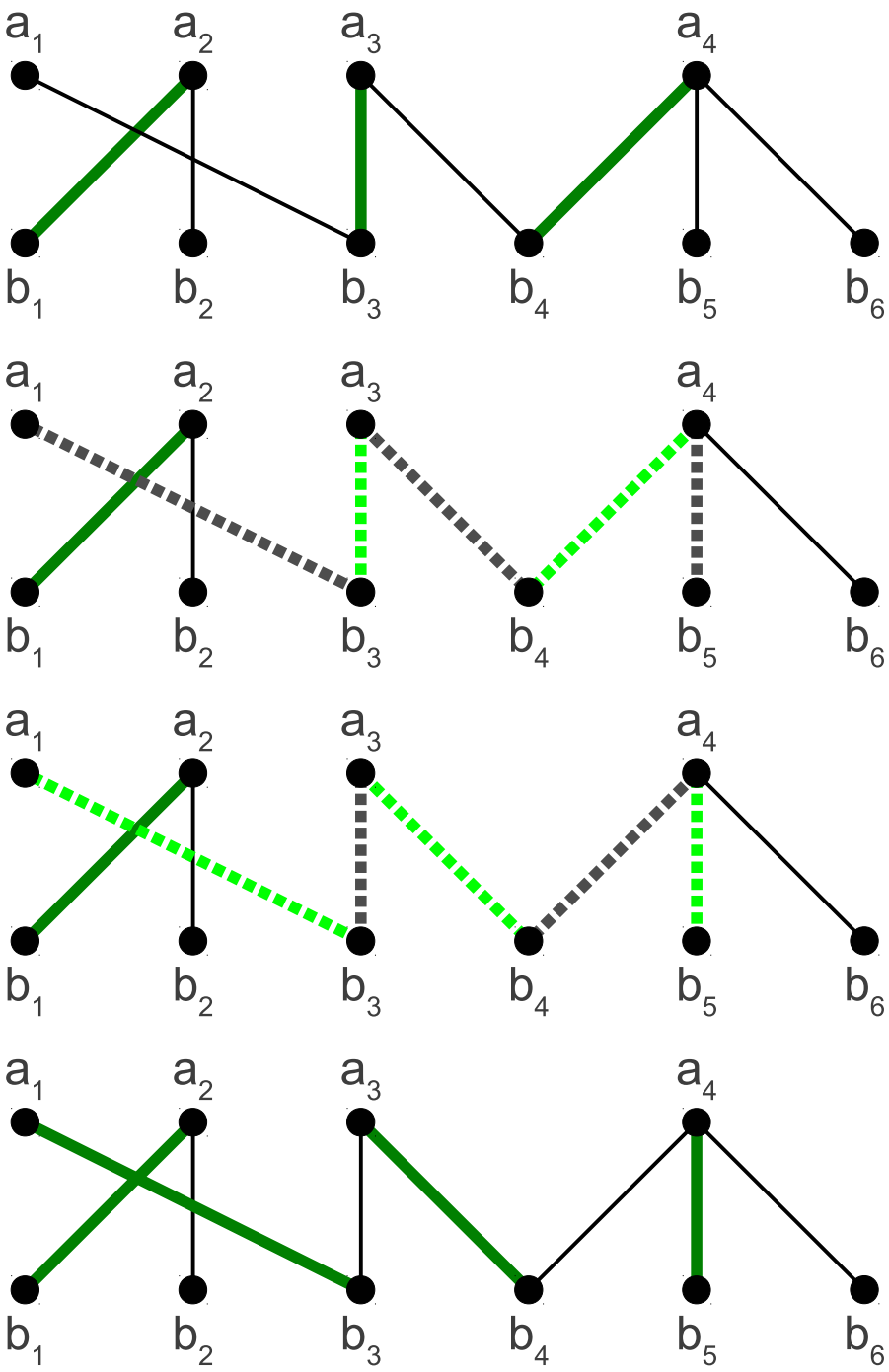


Figure 5.1: *A matching* $M_1 = \{a_2b_1, a_3b_3, a_4b_4\}$ *with an augmenting path* $a_1b_3a_3b_4a_4b_5$ *and the enlarged matching* $M_2 = M_1 \setminus \{a_3b_3, a_4b_4\} \cup \{a_1b_3, a_3b_4, a_4b_5\}$. *(For* M_1 *we have* $X = \{a_1\}$ *and* $Y = \{b_2, b_5, b_6\}$.*)*

Proof: We have to show that every edge of G has a vertex in T_A or in T_B (or both). Let us use the temporary notation $A' = A \setminus (X \cup T_A)$ and $B' = B \setminus (Y \cup T_B)$. Four kinds of edges have to be excluded:

- From X to Y:

  Such edges have been eliminated already at the end of the first phase, when M became non-extendable.

- From X to B':

  Such edges would be alternating paths of length one, therefore their B-vertices would belong to T_B rather than to B'.

- From A' to Y:

  A vertex $a \in A$ belongs to A' because there exists an edge $ab \in M$ such that its other end $b \in B$ is reachable from X along some alternating path P. But then, extending P with ab and with an edge from a to Y we would obtain an augmenting path from X to Y, which is impossible because the improving (second) phase of the algorithm terminated.

- From A' to B':

  As in the previous case, some vertex $a \in A$ belongs to A' because there exists an edge $ab \in M$ such that b is reachable from X along some alternating path P. If there was an edge from a to some $b' \in B'$, then extending P with the edges ab and ab' we would obtain an alternating path to b', therefore b' should belong to T_B rather than to B'.

$\Diamond$

We are now in a position to complete the proof of the theorem. The algorithm has found a matching M, its size cannot be larger than the matching number $\nu(G)$; and we also determined a transversal set T, its cardinality cannot be smaller than the transversal number $\tau(G)$. Thus, recalling that $\tau \geq \nu$ always holds, we obtain

$$\tau(G) \leq |T| = |M| \leq \nu(G) \leq \tau(G).$$

Since the two ends of this sequence of inequalities are equal, all numbers occurring in it must be equal; that is,

- $|T| = \tau(G)$, hence T is a transversal of minimum cardinality;

- $|M| = \nu(G)$, hence M is a matching of maximum size;

- $\tau(G) = \nu(G)$.

One can also observe that all steps of the algorithm can be implemented efficiently. This completes the proof of the theorem. □

By the proof above a matching in a bipartite graph is maximum if and only if no augmenting path exists. It is known that the analogue of this condition is also necessary and sufficient for the general non-bipartite case.

Matchings which cover all vertices, and also graphs having such matchings, are of special interest. For illustration see Figure 5.2.

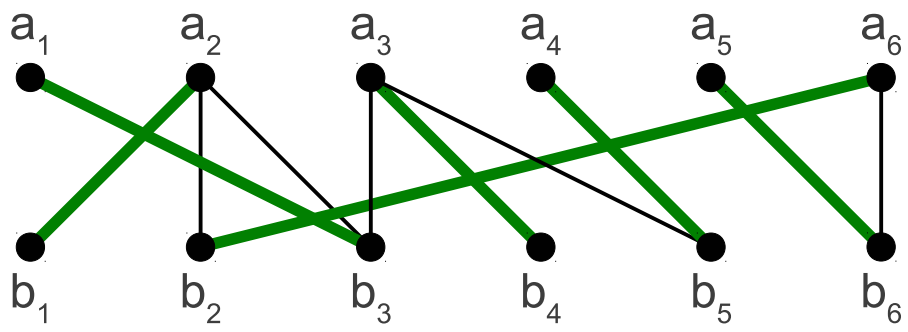


Figure 5.2: *An example for perfect matching in a bipartite graph*

Definition 5.2 *A **perfect matching** in a graph G is a set of edges such that each vertex is incident with precisely one edge.*

Clearly, a perfect matching is always a matching of maximum cardinality. Moreover, a necessary condition for admitting a perfect matching is that the graph has even number of vertices. But this condition is far from being sufficient; even if the graph is bipartite with partition classes of equal sizes, it may not have a perfect matching. For the existence of perfect matchings in this restricted class of graphs, the following famous condition will be shown to be necessary and sufficient.

Hall's Condition (HC): Let G be a bipartite graph with vertex classes A and B. We say that G satisfies Hall's Condition for the partite class A, if for every subset X of A

$$|N(X)| \geq |X|$$

holds, where $N(X)$ denotes the neighborhood of X, that is

$$N(X) = \bigcup_{x \in X} N(x).$$

Theorem 5.4 (Hall's Theorem, Hall's Marriage Theorem) *Let G be a bipartite graph with vertex classes A and B of equal size $|A| = |B|$. Then, G admits a perfect matching if and only if Hall's Condition holds for A.*

Proof: If G has a perfect matching M and $S = \{a_1, a_2, \ldots, a_k\}$ is a subset of A, we have k pairwise different vertices $b_1, b_2, \ldots, b_k$ in B with $a_i b_i \in M$ for every $1 \leq i \leq k$. Therefore, $N(S) \supseteq \{b_1, b_2, \ldots, b_k\}$ and $|N(S)| \geq k = |S|$ holds.

The other direction is proved by contradiction. We assume that G has no perfect matching and then prove that it cannot satisfy Hall's Condition. By Kőnig's theorem, $\tau(G) = \nu(G)$ holds, and by our present condition there exits no perfect matching, i.e.

$$\tau(G) = \nu(G) < |A|.$$

Consider a minimum transversal T and introduce the notations $T_A = T \cap A$ and $T_B = T \cap B$. Since $|T| = \tau(G) = |T_A| + |T_B| < n$, we have $|T_B| < n - |T_A|$. On the other hand, T contains at least one vertex from each edge and consequently, if a vertex x is not in T, all the edges incident with x have their other ends in T. In particular, for the set $X = A \setminus T_A$ every vertex from the neighborhood $N(X)$ of X belongs to $|T|$. This implies $|N(X)| \leq |T_B|$, and we conclude

$$|X| = n - |T_A| > |T_B| \geq |N(X)|,$$

which proves that G does not satisfy Hall's Condition for A. □

In fact, the proof above also verifies the following statement:

Theorem 5.5 *A bipartite graph G with partition classes A and B has a matching which covers all vertices of A, if and only if G satisfies Hall's Condition for A.*

As we observed in the previous proof, the existence of a matching covering all vertices of A equivalently means $\nu(G) = |A|$.

5.2 Systems of distinct representatives

In this section we turn back to set systems and consider the problem of system of distinct representatives.

Definition 5.3 *For set system $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$, where some members are allowed to be identical, a* ***system of distinct representatives (SDR)*** *is a set $\{x_1, x_2, \ldots, x_m\}$ of m pairwise different elements which satisfies $x_i \in S_i$ for each $1 \leq i \leq m$.*

For example, an SDR of system $\mathcal{S} = \{\{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c, d\}, \{a, c, d, e, f\}\}$ is $\{a, c, b, d, e\}$. But $\mathcal{S}' = \{\{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}, \{a, c, d, e, f\}\}$ has no SDR.

The problem of deciding whether there exists an SDR of a set system or determining one if exists, may seem quite different from the problem of maximum matching discussed in the previous section. Nevertheless via the following notion we will find a close connection between them.

Definition 5.4 *Given a set system $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ with underlying set $X = \bigcup_{1 \leq i \leq m} S_i = \{x_1, x_2, \ldots, x_n\}$, its incidence graph is the bipartite graph G with partite classes $A = \{a_1, a_2, \ldots, a_m\}$ and $B = \{b_1, b_2, \ldots, b_n\}$, where $a_i b_j \in E(G)$ if and only if $x_j \in S_i$.*

In other words, the incidence graph has one vertex for each member of the system and one for each element of the underlying set; moreover, adjacency in the incidence graph expresses incidence in the set system. An example is shown in Figure 5.3.

Now, we are ready to reformulate Hall's Theorem to give a necessary and sufficient condition for the existence of an SDR.

Theorem 5.6 *A set system $\mathcal{S}$ admits a system of distinct representatives if and only if the union of any k members of $\mathcal{S}$ contains at least k elements, for every $1 \leq k \leq |\mathcal{S}|$.*

Proof: Necessity is obvious: if the union of k sets contains fewer than k elements, we cannot select k different representatives for them.

To prove sufficiency, consider the incidence graph G of a set system $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$, where also repetitions of sets are allowed. Let $A = \{a_1, a_2, \ldots, a_m\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ be the partite classes of G, where a vertex a_i stands for the set S_i and a vertex b_j corresponds to the element x_j of the underlying set of $\mathcal{S}$. Then, consider a k-element subset X of A. By definition, $N(X)$ contains a vertex $b_j \in B$ if and only if b_j is adjacent to at

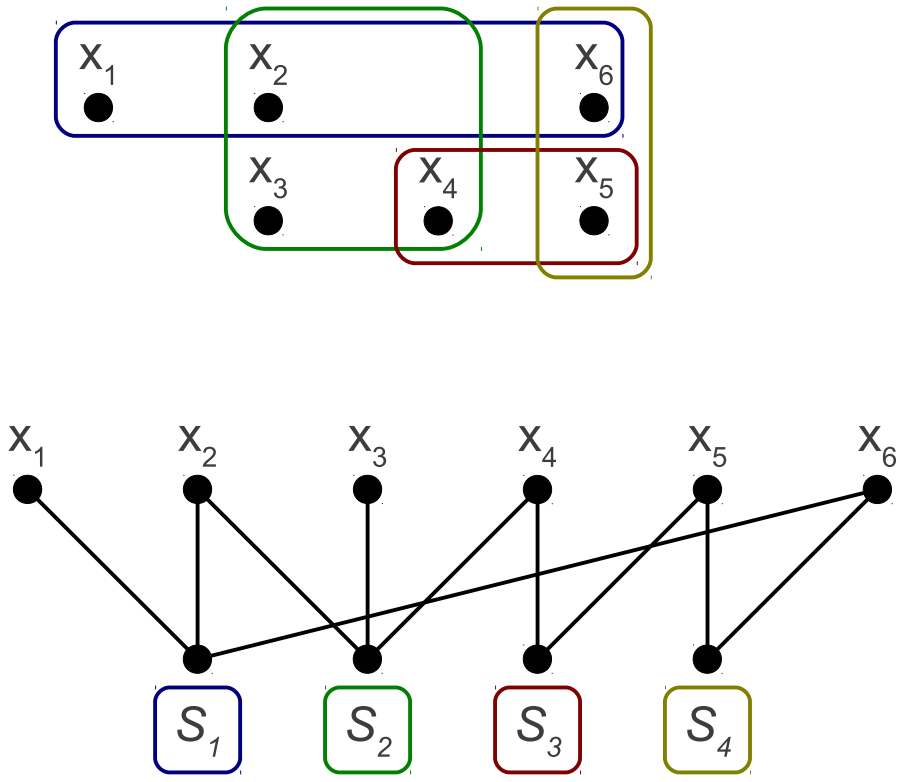


Figure 5.3: *A set system and its incidence graph*

least one $a_i \in X$. In the system $\mathcal{S}$ this means $x_j \in S_i$, or in other formulation $x_j \in \bigcup_{a_i \in X} S_i$. Thus,

$$|N(X)| = |\bigcup_{a_i \in X} S_i|.$$

Then, together with the condition given in the theorem, we have

$$|X| = k \leq |\bigcup_{a_i \in X} S_i| = |N(X)|,$$

which is exactly Hall's Condition for A in G. Hence, Theorem 5.5 implies that there exists a matching M which covers all vertices in A. This matching naturally determines a system of distinct representatives, just choose for every S_i the representative element x_j, if M contains the edge $a_i b_j$. □

5.3 Consequences of the Kőnig-Hall theorem

5.3.1 Edge colorings and factorizations of bipartite graphs

Recall that edge coloring of a graph G is a color assignment to the edges such that any two edges incident with a common vertex must have distinct colors. Equivalently, the edges sharing a fixed color form a matching. As observed in Chapter 1, $\Delta \leq \chi'$ holds for every graph. Vizing's theorem says that this trivial lower bound gives a good approximation, as the chromatic number can exceed Δ by at most 1.

Theorem 5.7 (Vizing's Theorem) *For every graph G,*

$$\chi'(G) \leq \Delta(G) + 1.$$

Hence, for every graph G either $\chi'(G) = \Delta(G)$ or $\chi'(G) = \Delta(G) + 1$ is valid. Still, deciding which one of these two equalities holds is an algorithmically hard problem in general.[3] On the class of bipartite graphs, however, the decision problem becomes easy to solve as for these graphs χ' always equals the maximum degree. In the first step, we prove the statement for regular bipartite graphs, then the result is extended to bipartite graphs in general. Recall that G is k-regular if each vertex $v \in V(G)$ is of degree k.

Theorem 5.8 *For every positive integer k and for every k-regular bipartite graph G,*

$$\chi'(G) = k.$$

[3] Furthermore, already on the class of non-bipartite graphs with all vertex degrees equal to 3, it is NP-hard to decide whether $\chi' = 3$ or $\chi' = 4$ holds.

Proof: Let A and B denote the two partite classes of G. As G is k-regular, for the number of its edges $|E(G)| = k|A| = k|B|$ holds implying $|A| = |B|$.

Further, for any subset $X \subseteq A$, the number of edges between X and B i.e. that of edges between X and $N(X)$ is precisely $k|X|$. Since each vertex $b_j \in N(X)$ is incident with at most k such edges, we have

$$k|X| \leq k|N(X)|.$$

So, we conclude that $|X| \leq |N(X)|$ is valid for every $X \subseteq A$, that is, Hall's Condition holds for G. Then, by Theorem 5.4, graph G must have a perfect matching.

From now, we proceed by induction on k. If $k = 1$, the edge set is a matching and all edges can have the same color 1; $\chi' = 1$. If $k \geq 2$, take a perfect matching M of G — which exists as proved above — and color the edges in M with color k. Deleting the edges in M from G, each vertex degree decreases by exactly 1. In this way the $(k-1)$-regular bipartite graph G' is obtained. By the induction hypothesis, G' can be edge-colored with colors $1, 2, \ldots, k-1$, which yields a proper edge coloring of G. Extending G' by the edges in M (all of them were assigned to color k), a proper edge coloring of G using k colors is obtained. This completes the proof. □

As a maximum matching in a bipartite graph can be found in polynomial time, the proof above also shows that a k-coloring of edges can be determined in polynomial time in any k-regular bipartite graph. We shall see that such an algorithm can be designed also without assuming regularity.

Theorem 5.9 *If G is a bipartite graph, then*

$$\chi'(G) = \Delta(G).$$

Proof: Since $\chi' \geq \Delta$ holds for every graph, it suffices to prove that G has an edge coloring with Δ colors. The crucial point is the extension of G by some edges and vertices to obtain a Δ-regular bipartite graph G'. Then, by Theorem 5.8 there is an edge coloring of G' with Δ colors. Finally, we delete some appropriately chosen edges and vertices, and we obtain a proper edge coloring of G with Δ colors.

The extension of G can be done in several ways. We propose the following procedure.

1. If $|A| > |B|$, extend B with $|A| - |B|$ new vertices. If $|B| > |A|$, extend A with $|B| - |A|$ new vertices.

2. While we have nonadjacent vertex pairs (a_i, b_j) with $a_i \in A$, $b_j \in B$ and with degrees $d(a_i) < \Delta$ and $d(b_j) < \Delta$, extend G with the edge $a_i b_j$.

3. If Step 2 cannot be applied, the vertices with degree smaller than Δ (if there exist some) induce a complete bipartite graph whose partite classes are, say, S_A and S_B. It is clear that $|S_A| < \Delta$ and $|S_B| < \Delta$. Then, put Δ new vertices into A and B each (denote the newly inserted vertex sets by N_A and N_B, respectively) and create some edges between S_A and N_B such that every vertex in S_A has degree Δ and the degrees in N_B differ by at most one. A similar procedure is executed for S_B and N_A.

4. Up to now, $N_A \cup N_B$ is an independent set. To complete the construction we can order the vertices of B according to non-decreasing degrees, say $b_1, b_2, \ldots, b_\ell$. Viewing this order cyclically (i.e., the successor of b_ℓ is b_1), we take the vertices $v \in A$ one by one and draw edges from v to the next $\Delta - d(v)$ vertices of B in the fixed cyclic order. When all the degrees in N_A become equal to Δ, then also the degrees in N_B equal Δ, and we have a Δ-regular bipartite graph G' with subgraph G.

□

The statement of Theorem 5.8 can be expressed in a different context as well.

Definition 5.5 *For an integer $k \geq 1$, a **k-factor** of a graph G is a k-regular spanning subgraph of it. A **k-factorization** of G is a decomposition into mutually edge-disjoint k-factors. In particular, a **1-factor** is just a perfect matching and **1-factorization** means a decomposition into perfect matchings.*

Of course, not all graphs have 1-factors. A 1-factorization always determines an edge coloring with minimum number of colors, but this is not true conversely. One can see that if G has a 1-factorization, then it is k-regular and $\chi' = k$ must hold. In fact, this is true conversely as well: If G is k-regular and has some edge coloring with k colors, this coloring naturally determines a k-factorization of G.

Then, Theorem 5.8 immediately implies:

Corollary 5.1 *Every regular bipartite graph has a 1-factorization.*

5.3.2 Orientations and out-degrees

Theorem 5.10 *Let t be a positive integer, and $G = (V, E)$ a graph. If any vertex subset $Y \subseteq V$ induces at most $t|Y|$ edges, then there exists an orientation of G in which each vertex has out-degree not greater than t.*

Proof: First, we reformulate the condition. If any vertex set Y spans at most $t|Y|$ edges, then any collection of k edges covers at least k/t vertices. Then, construct a bipartite graph F in which the partite class A contains one vertex for each edge of G, while the other class B contains exactly t copies for each vertex of G. Let a vertex $a \in A$ be adjacent to a vertex $b \in B$ if and only if the edge $e(a) \in E(G)$ corresponding to a is incident with the vertex $v(b) \in V(G)$ one of whose copies is b. (This construction is similar to the incidence graph of G, the only difference is that every vertex of G is represented by t vertices in F.)

We will prove that F satisfies Hall's Condition with partite class A. Indeed, take any subset $X \subseteq A$ and let us denote the number of vertices in X by k. These k vertices correspond to k edges of G. By the reformulated condition, these k edges together cover at least k/t vertices in G. These at least k/t vertices are represented with at least $t \cdot \frac{k}{t} = k$ vertices in F and by construction, each of them is adjacent to a vertex in X. In fact, the neighborhood $N(X)$ consists of exactly these vertices. To sum up, $|X| = k$ implies $|N(X)| \geq k$ for any $X \subseteq A$. This corresponds to Hall's Condition and therefore we have a matching M in F which covers all vertices of A.

Now, define the orientation of G as follows. An edge $uv \in E(G)$ is orientated from u to v if and only if in matching M above the vertex representing edge uv is paired up with one of the copies of u. Since u has t copies in B and each one is covered by at most one edge in matching M, any vertex $u \in V(G)$ has out-degree at most t. □

5.4 Stable matchings

A matching in a bipartite graph expresses a pairing between the elements of two sets. Up to this point we discussed problems where the purpose is to find a matching which is of maximum cardinality, now we consider the case when preferences are given and the goal is to find a matching which is stable.

- In this section, a bipartite graph G is meant to be given together with preference lists. That is, for every vertex $v \in A \cup B$, we have a bijective mapping $\{1, \ldots, d(v)\} \to N(v)$ which orders the neighbors due to preferences.

The preference list of v assigns 1 to the most preferred neighbor and then a neighbor x is preferred to a neighbor y if x is assigned to a smaller integer than y; in this case we also say that at u, neighbor x has higher preference than y, or simply u prefers x to y. In our discussion tie is not allowed, the preference mappings must be bijective.

In this section we discuss matchings with the following property:

Definition 5.6 *A **stable matching** in a graph G is a matching M such that for every edge $uv \in E(G) \setminus M$ either*

(i) u has a neighbor u' such that $uu' \in M$ and u prefers u' to v, or

(ii) v has a neighbor v' such that $vv' \in M$ and v prefers v' to u.

Related to a matching M, an edge $uv \in E(G) \setminus M$ which fails to satisfy both (i) and (ii) is called a ***blocking edge***. Hence, M is stable if and only if it has no blocking edge.

By definition, a stable matching is necessarily maximal, as an edge uv whose both ends are uncovered by M satisfies neither (i) nor (ii). Roughly speaking, a stable matching is stable in the sense that there are no two vertices (people, institutions) u and v such that for both of them it is advantageous to conspire and to give the new edge uv to M and to drop out the edges originally incident with u or v in M (if exist).

As a small example, consider the bipartite graph G with partite classes $A = \{a_1, a_2, a_3, a_4\}$ and $B = \{b_1, b_2, b_3\}$, where the preference lists are (the neighbors ordered due to decreasing preferences)

a_1: b_2, b_1;

a_2: b_2, b_3, b_1;

a_3: b_3, b_2;

b_1: a_2, a_1;

b_2: a_1, a_2, a_3;

b_3: a_2, a_3.

Then, matching $M_1 = \{a_1b_2, a_2b_3\}$ is stable as it can be checked. For G, this is the only stable matching but later we will see examples where more than one stable matching exists.

On the other hand, $M_2 = \{a_1b_2, a_2b_1, a_3b_3\}$ is a maximum matching in G but it is not stable as the edge a_2b_3 is blocking: a_2 prefers b_3 to b_1 and also, b_3 prefers a_2 to its present pair a_3.

Now, we prove an existence theorem by designing an algorithm which determines a stable matching for every bipartite graph.

Theorem 5.11 (Stable Marriage Theorem) *For any bipartite graph G with any preference lists on its vertices, there exists a stable matching in G.*

Proof: Consider a graph G with partite classes A and B and with preference lists on its vertices. Each phase of the algorithm consists of two steps

1. Every uncovered vertex $a \in A$ marks the edge connecting it to the neighbor due to highest preference.

2. If there is more than one marked edge incident with a vertex $b \in B$, we keep the one which is most preferred by b and the remaining ones are rejected (unmarked). The choice of b is only temporary, as in a later phase b can get a more preferred marked edge, and in this case b chooses the new one.

In the new phase, every vertex from A whose proposal was rejected marks the neighbor next on the preference list (if exists). The procedure is repeated while there is a rejected vertex in A whose list contains a further neighbor.

In every phase, when the set of marked edges is changed, the preferences on the vertices of B do not decrease, and on at least one vertex the preference necessarily increases. On the contrary, the preferences on the vertices of A do not increase and on at least one vertex it decreases.

When the algorithm terminates, we have some edges marked. Let M be the set of these edges. It is clear that M is a matching, we prove that this is stable. We have two cases for an edge $a_ib_j \notin M$

- If a_ib_j was not marked in any phases and the algorithm has terminated, then a_j is paired with a vertex b_k which has higher preference than b_j on the list of a_j. Hence, a_ib_j is not a blocking edge.

- If a_ib_j was marked in some phase but then was rejected by b_j, then b_j has a more preferred pair and again, a_ib_j is not a blocking edge.

Therefore, the algorithm produces a stable matching for every bipartite graph and its running time is proportional to the number of edges (plus the number of vertices). □

5.5 Perfect graphs

Here, in the last section of this chapter, we study a class of graphs which had a great influence on the development of structural graph theory. It contains several previously discussed types of graphs as subclasses. Some of the observations will be based on Kőnig's theorem, this is the reason why we include the topic here. It might be a separate section, too, but we do not wish to discuss it in that much detail.

Our starting point is the trivial inequality $\chi \geq \omega$, which is valid for every graph. The graphs in the focus of our present interest satisfy this relation with equality, but this alone would not be a very strong condition, since for every graph $G = (V, E)$ we have $\chi(G \cup K_{|V|}) = |V| = \omega(G \cup K_{|V|})$. For this reason, we assume in addition that the equality $\chi = \omega$ remains valid for every induced subgraph as well.

Definition 5.7 *A graph G is called **perfect** if for every induced subgraph $G' \subseteq G$ the equality*

$$\chi(G') = \omega(G')$$

holds.

It is easy to find examples for perfect graphs and for not perfect (i.e., imperfect) ones as well. First, we mention some simple ones.

- Every complete graph is perfect as $\chi(K_n) = n = \omega(K_n)$ and its induced subgraphs are also complete graphs. Similarly, every empty graph is perfect as $\chi(E_n) = 1 = \omega(E_n)$ holds for all $n \geq 1$.

- A cycle of even length is always perfect, as $\chi(C_{2k}) = 2 = \omega(C_{2k})$ and the equality $\chi \geq \omega$ remains valid for its non-empty induced subgraphs, too. (Obviously, if the subgraph is edgeless, both parameters are equal to 1.)

- A cycle of odd length not smaller than 5 is always imperfect as $\omega(C_{2k+1}) = 2$ and $\chi(C_{2k}) = 3$ hold for every $k \geq 2$. On the other hand, the cycle of length 3 is isomorphic to K_3 and therefore it is perfect.

- One can check that the complement $\overline{C_{2k+1}}$ of an odd cycle of length not smaller than 5 is also imperfect since $\omega(\overline{C_{2k+1}}) = k$ and $\chi(\overline{C_{2k+1}}) = k + 1$, if $k \geq 2$.

Also, the following wider classes belong to perfect graphs. They are commonly considered to be classical examples of fairly large perfect graph classes. The next theorem can be proved by using some coloring constructions and Kőnig's Theorem with its consequences.

Theorem 5.12 *Each of the following conditions implies that G is a perfect graph:*

(*i*) *G is bipartite;*

(*ii*) *G is the complement of a bipartite graph;*

(*iii*) *G is the line graph of a bipartite graph;*

(*iv*) *G is the complement of the line graph of a bipartite graph;*

(*v*) *G is a chordal graph;*

(*vi*) *G is the complement of a chordal graph.*

Proof: First, recall that $\chi = \omega = 1$ holds for empty graphs. From now on, we assume that G contains at least one edge. Moreover, observe that if G is contained in a class from (*i*)–(*vi*), then the deletion of some vertices always yields an induced subgraph which belongs to the same class. This property is valid also for graph classes derived from line graphs: If $G = L(F)$, then every vertex v of G corresponds to an edge $e = e_v$ of F, and so $G - v = L(F - e)$ holds. Therefore, although the definition of perfect graphs puts a requirement on every induced subgraph, it suffices to prove $\chi(G) = \omega(G)$ for each case of (*i*)–(*vi*) for the graph G itself.

(*i*) By definition, $\chi(G) = 2$ if G is bipartite. On the other hand, if G is bipartite, it cannot contain K_3 (which is the same as the 3-cycle), hence the largest clique in G is K_2. Then, $\chi(G) = \omega(G) = 2$ holds for every bipartite G.

(*ii*) If G is the complement of a bipartite graph, then its complement $\overline{G}$ is bipartite. A clique in G corresponds to an independent vertex set in $\overline{G}$ and hence

$$\omega(G) = \alpha(\overline{G}) = n - \tau(\overline{G}) = n - \nu(\overline{G})$$

holds, where n denotes $|V(G)|$ and the last equality is due to Kőnig's Theorem. Since $\chi(G) \geq \omega(G)$ is necessarily true, it is enough to prove that there exists a coloring of G which uses $n - \nu(\overline{G})$ colors. To do this, choose a maximum matching M in the bipartite $\overline{G}$, and then define a coloring in which any two vertices have different colors except if they are the two ends of the same matching edge from M. No edge from M belongs to graph G, hence no two vertices with the same color are adjacent in G. This implies that the coloring is proper and additionally, this is a coloring with precisely $n - |M| = n - \nu(\overline{G})$ colors. This verifies $\chi(G) = \omega(G)$.

(*iii*) Assume that $G = L(F)$ where F is a bipartite graph. The vertex colorings of the line graph $L(G)$ are in one-to-one correspondence with the edge colorings of F and hence $\chi(G) = \chi'(F)$. Moreover, by Theorem 5.9, the equality $\chi'(F) = \Delta(F)$ is valid as F is bipartite. A further consequence of bipartiteness is that F contains no cycle C_3 and hence, any three pairwise intersecting edges of F have a vertex in common. As regards the line graph $G = L(F)$, this implies $\omega(G) = \Delta(F)$. Now, we conclude $\chi(G) = \omega(G)$.

(*iv*) Let $G = \overline{L(F)}$ for a bipartite F. By definition of complement and that of line graph, we have

$$\omega(G) = \omega(\overline{L(F)}) = \alpha(L(F)) = \nu(F).$$

To prove $\chi(G) = \omega(G)$ we show a vertex coloring of G with $\nu(F)$ colors. Let T be a minimum transversal of G. Each edge of F can be assigned to a vertex $v \in T$ which covers it. Then, the edge set of F is decomposed into $\tau(F)$ intersecting subsystems. These intersecting subsystems are represented by $\tau(F)$ cliques in the line graph $L(F)$ and hence, we have $\tau(F)$ independent vertex sets in $G = \overline{L(F)}$. Taking these independent sets as color classes, a proper vertex coloring of G with $\tau(F)$ colors is determined. By Kőnig's Theorem, $\tau(F) = \nu(F)$ holds and consequently, $\chi(G) = \omega(G)$ is valid, too.

(*v*) For chordal graphs, $\chi(G) = \omega(G)$ was proved in Chapter 4.

(*vi*) Suppose that $G = \overline{F}$ and that F is chordal. A side product of the algorithm given in Section 3.3.1 is that $\alpha(F) = \theta(F)$ holds. Then, by the complementarity of ω and α, and by that of χ and θ, we obtain the validity of $\chi(G) = \theta(F) = \alpha(F) = \omega(G)$. This completes the proof of $\chi(G) = \omega(G)$.

□

Finally, we mention two famous theorems concerning perfect graphs. Both of them were stated as conjectures in the early 1960's. The first one was proved in an elegant way ten years later, but it took about three further decades until a (rather long) proof of the second one was found.

Theorem 5.13 (Perfect Graph Theorem) *A graph G is perfect if and only if its complement $\overline{G}$ is perfect.*

Theorem 5.14 (Strong Perfect Graph Theorem) *A graph G is perfect if and only if neither G nor $\overline{G}$ contains any induced odd cycles of length greater than 3.*

The Perfect Graph Theorem tells us a good reason why the six graph classes in Theorem 5.12 are in three complementary pairs. On the other hand, the Strong Perfect Graph Theorem gives a characterization of perfect graphs in terms of forbidden induced subgraphs.

Of course, Theorem 5.13 immediately follows from Theorem 5.14. Nevertheless, due to the substantial difference between their proofs, the former definitely remains of independent interest.

Chapter 6

The Max-Cut problem

In this chapter we consider the problem of finding a large bipartite subgraph in a graph. A more general version of this optimization problem is called ***Max-Cut***, therefore we shall use the word 'cut' in our terminology.

Definition 6.1 *Let $G = (V, E)$ be a graph and $X \cup Y = V$ a partition of its vertex set into two classes. The **cut** generated by (X, Y) is the set $F \subseteq E$ of all edges which have one end in X and the other end in Y. We denote the largest possible cut size $|F|$ in G by $mc(G)$.*

Notation 6.1 *For any two disjoint sets of vertices, we denote by $e(X, Y)$ the number of edges with one end in X and the other end in Y. Hence, $mc(G)$ is the maximum value of $e(X, Y)$ taken over all pairs $X, Y \subseteq V$ with $X \cap Y = \emptyset$.*

Keeping the usual meaning of cut in mind, it would be reasonable to assume that both X and Y are non-empty. Nevertheless, it is meaningful to say that the partition $(V, \emptyset)$, one of whose two classes is empty, generates the edge set $F = \emptyset$. Apart from this degenerate case, the graph

$$G - F = (V, E \setminus F)$$

obtained by the removal of the edges of the cut (i.e., the graph with the same vertex set V as G, and with the edge set $E \setminus F$) surely is disconnected. We note further that $F = \emptyset$ may occur also if $X \neq \emptyset \neq Y$ holds; namely, if G is disconnected, then splitting its connected components into two non-empty classes we obtain a non-trivial vertex partition which generates an edgeless cut.

Remark 6.1 *The value of $mc(G)$ is equal to the maximum number of edges in a bipartite subgraph of G. Indeed, it is immediate by definition that the*

largest number, say $b(G)$, of edges in a bipartite subgraph of G is at least $mc(G)$ because the edges in every cut form a bipartite subgraph. That is, $b(G) \geq mc(G)$. To prove that $mc(G) \geq b(G)$ also holds, consider a bipartite subgraph $H \subseteq G$, say with vertex classes A and B, such that H contains exactly $b(G)$ edges. Let $X := A$ and $Y := V \setminus A$. Then $B \subseteq Y$, hence all edges of H are contained in the cut (X, Y). Consequently, $b(G) = |E(H)| \leq e(X, Y) \leq mc(G)$.

Remark 6.2 *An alternative way to think about cuts of $G = (V, E)$ is to select a vertex subset $X \subset V$ and consider the set of edges which have exactly one end in X. Equivalence with the original definition is established by setting $Y := V \setminus X$.*

Remark 6.3 *The equality $mc(G) = |E|$ holds if and only if the entire edge set E can be chosen for F, which happens precisely when G is bipartite. Since bipartite graphs can be recognized by an efficient algorithm, we can also determine the optimum value and the optimal solution on this restricted class of graphs, and decide efficiently in general whether or not $mc(G) < |E|$.*

Algorithmically it is a hard problem to determine the exact value of $mc(G)$ on an unrestricted graph G if it has many vertices. In fact, even to approximate $mc(G)$ beyond a certain precision is hard.[1] More formally, there exists a constant c Here we prove the following general lower bound:

Theorem 6.1 *For every graph $G = (V, E)$, we have*

$$mc(G) \geq \frac{|E|}{2}.$$

Equivalently, the theorem states that every graph can be made bipartite by the removal of at most half of its edges.

It is remarkable that this result can be proved by quite different methods. In the following sections we present three important techniques.

6.1 First approach: Searching local optimum

The idea of this approach is to reach a solution via a sequence of improvements. We start with an arbitrary cut and make small local modifications as

[1] It is known that there exists a constant $c > 0$ such that finding a lower bound, say $L = L(G)$ on $mc(G)$ which satisfies the inequality $L(G) \geq (1 - c) \cdot mc(G)$ for all graphs G is NP-hard.

long as the size of the cut can be increased. Once a non-improvable situation is reached, we verify that it satisfies the inequality to be proved.

Consider an arbitrary vertex partition

$$V = X \cup Y.$$

If some vertex $x \in X$ has more neighbors in X than in Y, we move it from X to Y, i.e. modify the partition to

$$X' := X \setminus \{x\}, \qquad Y' := Y \cup \{x\}.$$

Then the edges joining x with its neighbors in Y get removed from the cut, while the edges joining x with its neighbors in X become members of the cut. The assumption on x implies that the number of the latter is larger than the number of the former; thus, if such a vertex x exists in the original partition (X, Y) then we obtain a cut (X', Y') of larger size. The same idea applies if there is a vertex $y \in Y$ having more neighbors in Y than in X. After not more than $|E|$ steps of this kind the procedure surely stops and no more improvements are possible in this way. Let us denote the two vertex classes in this final cut by X^* and Y^*.

In this situation every vertex v has at least half of its neighbors in the cut, which means at least $d(v)/2$ where $d(v)$ denotes the degree of v. Since each of those edges has precisely one vertex in X^* and one in Y^*, we have

$$e(X^*, Y^*) \geq \sum_{v \in X^*} \frac{d(v)}{2}$$

and also

$$e(X^*, Y^*) \geq \sum_{v \in Y^*} \frac{d(v)}{2}.$$

Thus, taking the average of these two inequalities, we obtain

$$e(X^*, Y^*) \geq \frac{1}{2} \sum_{v \in X^* \cup Y^*} \frac{d(v)}{2} = \frac{1}{4} \sum_{v \in V} d(v) = \frac{1}{4} \left(2 \cdot |E|\right) = \frac{|E|}{2}$$

because the sum of vertex degrees is the double of the number of edges in any graph. By definition, every cut has at most $mc(G)$ edges, and consequently

$$mc(G) \geq e(X^*, Y^*) \geq \frac{|E|}{2}$$

follows, proving the theorem.

The algorithm is illustrated step-by-step in Section A.4 through an example.

Algorithm 6.1 Algorithm to find locally maximal max cut

$G := (V, E)$, $X = V$, $Y = \emptyset$
$A := \{v \in X \mid |N(v) \cap X| > |N(v) \cap Y|\} \cup \{v \in Y \mid |N(v) \cap X| < |N(v) \cap Y|\}$
while $A \neq \emptyset$ **do**
  Select v from A arbitrary
  if $v \in X$ **then**
    $X := X \setminus \{v\}$, $Y := Y \cup \{v\}$
  else
    $X := X \cup \{v\}$, $Y := Y \setminus \{v\}$
  end if
  $A := \{v \in X \mid |N(v) \cap X| > |N(v) \cap Y|\} \cup \{v \in Y \mid |N(v) \cap X| < |N(v) \cap Y|\}$
end while
Cut size of the partition (X, Y) cannot be improved by switching the position of one vertex.

6.2 Second approach: Finding a solution online

In this model we consider the vertices in a sequence $v_1, v_2, \ldots, v_n$ (according to an arbitrarily chosen order) one by one, and make a decision concerning v_i $(i = 1, 2, \ldots, n)$ according to the subgraph induced by $\{v_1, \ldots, v_i\}$ without using any information about vertices of larger index.

We shall maintain and sequentially update a pair (X, Y) of disjoint sets. Initially we set $X = Y = \emptyset$. Then, for $i = 1, \ldots, n$ we modify (X, Y) as follows:

- If v_i has at least as many neighbors in Y as in X, then re-define $X := X \cup \{v_i\}$; otherwise re-define $Y := Y \cup \{v_i\}$.

Having processed all vertices, eventually (X, Y) is a partition of V. We prove that the number of edges in this cut is sufficiently large.

The algorithm is illustrated step-by-step in Section A.5 through an example.

For the proof, each edge $v_i v_j$ will be counted at its vertex of larger subscript, $\max(i, j)$. (This view may as well be considered as an edge decomposition of G into stars $S^2, S^3, \ldots, S^n$ where for $2 \leq j \leq n$ the center of S^j is v_j and its edge set is $\{v_i v_j \in E \mid 1 \leq i < j\}$.) Denoting by d_j^- the number of edges $v_i v_j$ with $i < j$, we see that $\sum_{j=2}^{n} d_j^- = |E|$. Moreover, at each v_j, at least half of the edges ending there belong to the cut (X, Y), that is at least

Algorithm 6.2 Algorithm to find cut with the online aproach

$G := (V, E)$, $X = Y = \emptyset$
Fix a vertex order $v_1, v_2, \ldots, v_n$ arbitrarily
for $i = 1$ to n **do**
  if $|N(v_i) \cap X| > |N(v_i) \cap Y|$ **then**
    $Y := Y \cup \{v_i\}$
  else
    $X := X \cup \{v_i\}$
  end if
end for
Vertex set of G is partitioned into X and Y with the online approach

$d_j^-/2$. Consequently,

$$mc(G) \geq \sum_{j=2}^{n} \frac{d_j^-}{2} = \frac{|E|}{2}.$$

6.3 Third approach: The probabilistic method

Among the three proofs, perhaps this is the most unexpected one. Contrary to the other two, it is not constructive. We choose a partition $X \cup Y = V$ at random, by what we mean that we apply the rule

$$\mathbb{P}(v \in X) = \frac{1}{2}$$

for each vertex $v \in V$, where $\mathbb{P}(\,.\,)$ denotes probability. Of course, this condition equivalently means that $\mathbb{P}(v \in Y) = 1/2$ holds for each $v \in V$. We apply this rule for each v independently of all the other vertices; i.e., no matter how (X, Y) partitions $V \setminus \{v\}$, vertex v still has probability $1/2$ to be in X or in Y.

An edge $v_iv_j \in E$ belongs to the cut if and only if its two ends have been placed into distinct classes, one of them into X and the other into Y. By our rule on random choice, independently of whether v_i has been placed into X or Y, v_j goes into the other class with probability $1/2$. Consequently,

$$\mathbb{P}(v_iv_j \text{ is in the cut}) = \frac{1}{2}$$

for every single edge. Hence, for the expected number of edges in the cut we obtain

$$\mathbb{E}(e(X, Y)) = \sum_{v_iv_j \in E} \mathbb{P}(v_iv_j \text{ is in the cut}) = \frac{|E|}{2}.$$

Expected value of a random variable expresses a weighted sum, namely the possible outcomes of the experiment weighted by the probabilities of the outcomes. In other words, it is a weighted average, always being between the minimum and maximum possible value of the random variable in question. For this reason, at least one of the events has value not smaller than the expectation of the random variable. In our case this implies

$$mc(G) \geq \mathbb{E}(e(X,Y)) = \frac{|E|}{2}.$$

6.4 Notes

We have stated and proved $mc(G) \geq \frac{1}{2}|E|$ for all graphs $G = (V, E)$. In fact, except for empty graphs, strict inequality holds here. This can be seen from the online approach, because the first edge appearing in the fixed order surely belongs to the cut, and at least half of the later edges also do so. Strict inequality can be read out from the probabilistic proof, too: The empty cut with zero edges has positive probability, and since its size is smaller than average (the expected value of cut size), there must exist a cut whose size is larger than average. Applying a more complex argument one can prove that

$$mc(G) \geq \left\lceil \frac{|E|}{2} + \frac{|V|-1}{4} \right\rceil$$

holds for all *connected* graphs $G = (V, E)$. This lower bound is tight whenever G is a complete graph.

6.4.1 Online algorithms

It is appropriate to say a few words about online algorithms here. The context becomes clear by making a comparison with offline problems. In the ***offline*** case of an optimization problem, complete information about the entire problem instance is available already at the beginning, before starting the computation.

The online scenario is different. The input is revealed piece by piece, and we have to make a decision about the recently received piece before knowing anything about later pieces, and it is not even known whether there will be any further pieces; this becomes known only after the processing of the current piece.

In this sense the second proof of Theorem 6.1 can be viewed as an online algorithm because when we decide about the position of v_j, we need not know anything about the adjacencies from and between vertices of higher index.

In practice, we often encounter "semi-online" problems, in which the input arrives piece by piece, and we have to decide about the current piece while we have partial information about later pieces.

6.4.2 Probabilistic methods

An interesting aspect of the third proof is that we conclude the existence of a suitable cut without actually constructing it. From this proof we know for sure that a solution with the required properties exists, but we cannot see from the argument where it is in G. Such a situation often means that in fact there exist many good solutions, because even their average is good enough.

Taking a decision with probability 1/2 is like flipping a (fair) coin. We obtain the same result if we throw a dice and choose one decision if the result is odd and choose the other decision if the result is even. As we have seen, this works nicely for the general lower bound on $mc(G)$. In some other situations, however, it may be useful to distribute probabilities unevenly, for example to take a particular decision only if the dice gave 6, or throwing the dice two times and making a step only if the sum of the two is at least 11, etc. (These variants correspond to probability 1/6 and 1/12, respectively.) In other words, we flip a biased coin, for which head has probability p and tail has probability $1 - p$. Here p can be any real number between 0 and 1, according to the nature of the problem to be solved. In some situations it may even be useful to choose non-constant p that gets close to 0 or to 1 as the input gets large.

Beside non-constructive existence proofs, a very important application of non-deterministic selection is in the area of *randomized algorithms*. In that way, the efficiency of some algorithms can substantially be improved with high probability.

Chapter 7

Locally restricted colorings

The title of this chapter does not completely describe the contents; it just refers to the type of problems we consider, and there are many variants of coloring which belong to this category but we do not touch them here. On the other hand, from our previous chapters proper vertex coloring is already a local restriction: it means that if a 2-vertex subgraph contains an edge, then two distinct colors have to occur in it. Similarly, proper edge coloring requires that any two edges in a vertex triple must get two colors. Below we consider more restricted variants of these notions.

7.1 Precoloring extension

Extending a partial solution is an idea that occurs in many kinds of context. Precoloring extension, traditionally abbreviated as PrExt, means the following algorithmic decision problem:

Input: Graph $G = (V, E)$, color bound $k \in \mathbb{N}$, partial coloring $\varphi_W : W \to \{1, \ldots, k\}$ that is a proper vertex coloring of the subgraph $G[W]$ induced by the precolored set $W \subset V$ in G.

Question: Does G have a proper vertex coloring φ with at most k colors, which extends φ_W, i.e. $\varphi(v) = \varphi_W(v)$ holds for all $v \in W$?

Example 7.1 *Let G be the path $uvxy$ of length 3, and assume that the color bound is 2. We consider $W = \{u, y\}$. If $\varphi_W(u) = 1$ and $\varphi_W(y) = 2$, then the precoloring is extendable to the optimal coloring with $\varphi(v) = 2$ and $\varphi(x) = 1$. But if $\varphi_W(u) = \varphi_W(y) = 1$, then the precoloring is not extendable within the prescribed color bound.*

There is a natural 'search version' of the problem, which, instead of just asking whether the precoloring is extendable, also requires a proper k-coloring of G as a solution if it exists.

It is clear that the answer to PrExt is 'no' whenever k is smaller than the chromatic number of G. Hence, it is reasonable to assume that k is at least as large as $\chi(G)$. But it should be emphasized that we do not require equality, k may be any larger than $\chi(G)$.

Remark 7.1 *If $W = \emptyset$, or W induces a complete graph in G, then the answer to* PrExt *is 'yes' if and only if $\chi(G) \leq k$. In other words,* PrExt *in these restricted cases is equivalent to the graph k-colorability problem.*

Hence, in some cases PrExt and k-colorability are equally hard algorithmically. On the other hand, in some other situations PrExt is provably harder. We list some results for comparison.

- On bipartite graphs, with $k = 2$, PrExt is decidable efficiently.
- On bipartite graphs, with any $k \geq 3$, PrExt is hard to decide.
- On complements of bipartite graphs, the search version of PrExt with a general k is exactly as hard as finding largest matchings in bipartite graphs; that is, solvable efficiently.
- On interval graphs, PrExt is hard.
- On interval graphs, if no color occurs more than once in the precolored set W, then PrExt is solvable efficiently. The same is valid on chordal graphs, too.
- On interval graphs, PrExt is hard even if we restrict the input to precolorings in which each color occurs on at most two precolored vertices.

For some classes of well-structured graphs, beside deciding PrExt (or solving its search version) is not only doable efficiently but also transparent necessary and sufficient structural conditions can be given for the positive/negative instances of the problem.

7.2 List coloring

A more expressive phrase for the problem considered here would be something like "graph coloring from lists"; but the official term is the one given in the subtitle.

Definition 7.1 *Let $G = (V, E)$ be a graph, and let $\mathcal{L} = \{L_v \mid v \in V\}$ be a collection of sets which specify the colors allowed for every vertex v. A **list coloring** of G is a color assignment $\varphi : V \to \bigcup_{v \in V} L_v$ such that*

- *$\varphi(v) \in L_v$ for all $v \in V$;*
- *$\varphi(u) \neq \varphi(v)$ whenever $uv \in E$.*

*If such a φ exists, we say that G is **list colorable**, or that G has an $\mathcal{L}$-coloring. The set L_v is termed the **list** of vertex v (although it is just a set, no ordering is assumed among its elements).*

Remark 7.2 *Precoloring extension can be viewed as a subproblem of list coloring: for vertices $v \in W$ we have a list consisting of a single prescribed color $L_v = \{\varphi_W(v)\}$, while for vertices $v \in V \setminus W$, the allowed set $L_v = \{1, 2, \ldots, k\}$ is the set of all colors.*

Example 7.2 *Let $G = K_n$ be a complete graph, with a given list assignment $\mathcal{L} = \{L_1, \ldots, L_n\}$ on its vertices. Then G is list colorable if and only if the sets in $\mathcal{L}$ satisfy Hall's Condition. The reason is that no two vertices of K_n can be assigned to the same color, hence its list colorings are in one-to-one correspondence with the systems of distinct representatives of the set system $\mathcal{L} = \{L_1, \ldots, L_n\}$.*

One can see, in particular, that if the lists are very long — e.g., if all of them have at least as many colors as the number of vertices — then the graph surely is list colorable. This leads to the following notions and a graph invariant which is the list coloring analogue of chromatic number.

Definition 7.2 *A k-**assignment** on a graph $G = (V, E)$ is a list assignment $\mathcal{L} = \{L_v \mid v \in V\}$ in which $|L_v| = k$ for all $v \in V$. The **choice number** of G — also called **list chromatic number** in some part of the literature — is the smallest k such that G is $\mathcal{L}$-colorable for every k-assignment $\mathcal{L}$. We shall denote the choice number of G by $\chi_\ell(G)$. (Its other standard notation is $\mathrm{ch}(G)$.) We also say that G is k-**choosable** if it is list colorable for every k-assignment.*

The basic chain of inequalities for the graph invariants related to coloring can be written as follows.

Proposition 7.1 *Every graph G satisfies*

$$\omega(G) \leq \chi(G) \leq \chi_\ell(G) \leq \mathrm{col}(G).$$

Proof: We have to prove that χ_ℓ is sandwiched between χ and col. If $\chi_\ell(G) = k$, then G has to be list colorable for every k-assignment. In particular, setting $L_v := \{1, 2, \ldots, k\}$ for every vertex v, $\mathcal{L}$-coloring precisely means proper k-coloring. Its existence implies $\chi(G) \leq k$, as claimed.

For the rightmost inequality we consider a vertex order $v_1, \ldots, v_n$ of G that attains $\max_{1 \leq i \leq n}(d^-(v_i) + 1) = \text{col}(G)$, where d^- denotes backward degree in the given order. Assume that each of the lists of the vertices contains at least col(G) colors. We can then color the vertices "from left to right" in the order $v_1, v_2, \ldots, v_n$. While treating v_i, we see that only at most $d^-(v_i)$ colors are excluded from its list because no vertices of higher index are colored at that moment. Since the list is longer, there is a free color in L_{v_i} that we can assign to v_i without creating a monochromatic edge. Hence, G is list colorable. □

Many graphs have larger choice number than chromatic number.

Example 7.3 *On six vertices there are two complete bipartite graphs (hence having $\chi = 2$) which are not 2-choosable:*

- $K_{2,4}$ — *Let the vertices in the two partite sets be a_1, a_2 and b_1, b_2, b_3, b_4, respectively. We construct the following 2-assignment:*

$$L_{a_1} = \{1, 2\}, \quad L_{a_2} = \{3, 4\}$$

$$L_{b_1} = \{1, 3\}, \quad L_{b_2} = \{1, 4\}, \quad L_{b_3} = \{2, 3\}, \quad L_{b_4} = \{2, 4\}$$

- $K_{3,3}$ — *Let the vertices in the two partite sets be a_1, a_2, a_3 and b_1, b_2, b_3, respectively. We construct the following 2-assignment:*

$$L_{a_1} = L_{b_1} = \{1, 2\}, \quad L_{a_2} = L_{b_2} = \{1, 3\}, \quad L_{a_3} = L_{b_3} = \{2, 3\}$$

In either of them, no matter how we color the vertices a_i from their lists, there will be some b_j the two allowed colors of which are already used, so that we cannot complete the coloring on the entire graph.

In fact, χ_ℓ can take any large value on bipartite graphs.

7.3 Kernels in directed graphs

In this section we consider a special type of vertex sets in *directed* graphs.

Definition 7.3 *Let $D = (V, A)$ be a digraph with vertex set V and arc set A. A **kernel** of G is a set $M \subset V$ satisfying the following two properties:*

1. *M is independent;*

2. *for every vertex $u \in V \setminus M$ there is a $v \in M$ such that $uv \in A$.*

In other words, no arcs occur inside M, but from each vertex outside M there is a direct arc into M.

Some digraphs have kernels, some others don't.

Example 7.4 *Consider the directed cycle $\vec{C}_n = v_1v_2 \dots v_n$, whose arcs are $v_1v_2, v_2v_3, \dots, v_{n-1}v_n, v_nv_1$ ($n \geq 2$). If n is even, then $\vec{C}_n$ contains kernels: we can take the $n/2$ vertices of the same parity (all with odd indices or all with even indices). If n is odd, however, then $\vec{C}_n$ does not have any kernels. Indeed, since a kernel M should be independent, we can select at most $(n-1)/2$ vertices into it. Then there must occur two non-selected ones which are consecutive along the cycle, say v_i and v_{i+1}. But then v_i violates condition 2 because it is outside M, and its unique out-neighbor v_{i+1} is not in M either.*

Remark 7.3 *If M is a kernel in D, and v is a vertex of out-degree zero, then*

- *$v \in M$;*
- *$u \notin M$ for all $u \in V$ with $uv \in A$.*

The reason is that v cannot be outside M because of condition 2, and once we have $v \in M$, the in-neighbors of v cannot be inside M because of condition 1.

This simple property determines some steps when searching a kernel under some circumstances:

Theorem 7.1

1. *If $\vec{T}$ is an oriented tree (i.e., no cycles occur in it, not even those of length two), then $\vec{T}$ has a kernel, its kernel is unique, and can be found by an efficient algorithm.*

2. *More generally, every bipartite directed graph has at least one kernel (not always unique), and a kernel can be found efficiently.*

Proof: 1. If every vertex had positive out-degree, then the (finite) graph would contain a directed cycle. Consequently, if $\vec{T}$ is an oriented tree, then it contains some vertex of out-degree zero. Certainly, such a vertex, say v, can be found efficiently. In the construction of M we can perform the following steps:

- Insert v into M.
- Remove all vertices u from $\vec{T}$ such that uv is an arc.

Due to Remark 7.3, these steps are compatible with all kernels of $\vec{T}$, hence the kernels of $\vec{T}$ are in one-to-one correspondence with those of the reduced graph. Indeed, the vertices u satisfy condition 2 because we have put v into M. Moreover, the remaining vertices are non-adjacent to v, therefore any kernel M' of the reduced graph yields a kernel $M = M' \cup \{v\}$ of $\vec{T}$.

After the removal of v and its in-neighborhood, each connected component of the graph remains an oriented tree. Thus, we can repeat the steps above until we find the unique kernel of $\vec{T}$. (If $\vec{T}$ has just one vertex, then this vertex itself is the unique kernel.)

2. Let D be a bipartite digraph. It suffices to consider the 'weak components' of D, which are the connected components of the undirected graph obtained from D by omitting the orientations (i.e., replacing ordered vertex pairs with unordered pairs).

Hence, consider a connected bipartite digraph, say with vertex classes A and B. If there is a vertex with zero out-degree, then we perform the steps displayed above, and so reduce D to a smaller bipartite digraph. Then the existence of a kernel follows by induction on the number of vertices.

On the other hand, if all vertices have positive out-degree, then all $u \in B$ have at least one out-neighbor in A, and similarly, all $u \in A$ have at least one out-neighbor in B. Since both vertex classes are independent sets, either of A and B is a kernel in D. □

The algorithm is illustrated in Figure 7.1 in a concise way; instead of selecting one vertex of out-degree zero at a time, all such vertices are marked simultaneously, and then all their neighbors and the incident edges (indicated with grey afterwards) are deleted before the next step.

The importance of kernels in connection with list coloring is shown by the following result.

Theorem 7.2 *Let $D = (V, A)$ be an orientation of the undirected graph $G = (V, E)$, such that every induced subgraph has a kernel. If $\mathcal{L} = \{L_v \mid v \in V\}$ is a list assignment on G where the list L_v of each vertex $v \in V$ contains more colors than the out-degree $d^+(v)$ of v in D, then G is list colorable.*

Proof: Select an arbitrary color c, and consider the sub(di)graphs $G_c \subset G$ and $D_c \subset D$ induced by precisely those vertices whose lists contain c. By assumption, D_c contains a kernel M. We make the following modifications:

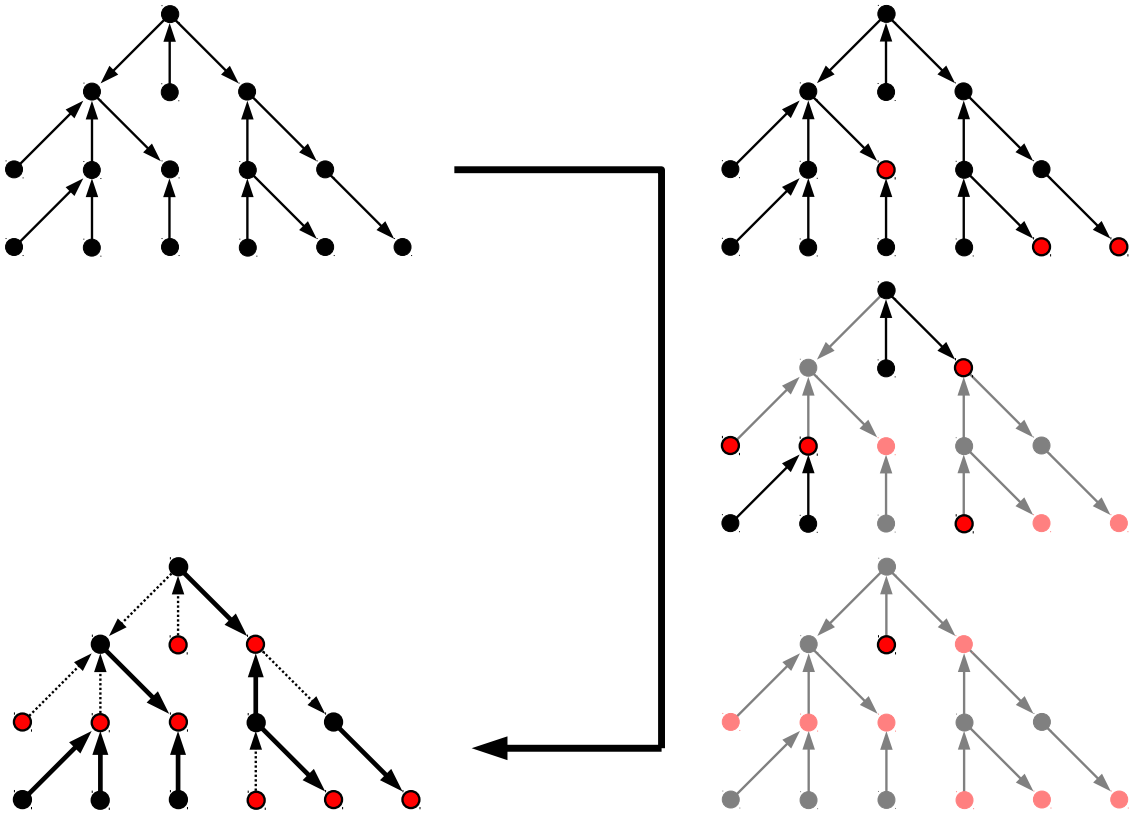


Figure 7.1: *Finding the kernel of an oriented tree*

- Assign color c to the vertices of M.
- Delete color c from the vertices of $G_c - M$.

Since the kernel M is independent, we did not create any monochromatic edges in this step. In the reduced lists of the smaller graph $G' := G_c - M$, color c does not occur anymore, therefore no matter how we shall color its vertices later, they will never create monochromatic edges with vertices of M. Thus, the proof will be done if we show that G' is list colorable.

We are going to verify that the conditions of type $|L| > d^+$ concerning the reduced lists and the out-degrees remain valid in the reduced digraph $D' := D - M$, too. Then the theorem follows by induction on the number of vertices. No out-degrees have been increased, and the lists have not been modified outside G_c, therefore the condition remains valid for all vertices in $G - G_c$. Moreover, the list of a vertex in $G_c - M$ has been reduced by precisely one color. But M is a kernel in G_c, and consequently if we remove M, the out-degrees of all vertices in $G_c - M$ decrease by *at least* 1. Thus, the conditions $|L| > d^+$ remain valid for all vertices in the reduced graph. This completes the proof of the theorem by induction. □

We note that the proof also leads to an efficient algorithm to find a list coloring, provided that a kernel can be found efficiently in each induced subgraph of G.

It is important to emphasize that the conditions $|L_v| > d^+(v)$ alone do not imply list colorability. For example, the complete graph K_{2s+1} admits an orientation with maximum out-degree s for any $s \in \mathbb{N}$, but its choice number is $2s + 1$. The presence of kernels is essential in this method.

7.4 Line graphs of bipartite graphs

In the main result of this section we will prove that the line graphs of bipartite graphs satisfy the equality $\chi_\ell = \chi = \omega$. In fact we know $\chi = \omega$ from the chapter on bipartite graphs: we have proved that these graphs are perfect. So the result mentioned first can also be stated in the form that bipartite graphs are list edge-colorable from any lists whose size is equal to the maximum degree.

Before proving the main theorem, we formulate an assertion; although it looks quite different, it will turn out to be equivalent to the Stable Marriage Theorem. Among the assumptions of the next lemma, the exclusion of 3-cycles equivalently means that every 3-cycle is oriented transitively, like the triangle with vertices a, b, c and arcs ab, bc, ac.

Lemma 7.1 *Let H be a bipartite graph, G be the line graph of H, and D be any orientation of G such that no (cyclically) directed 3-cycle occurs in D. Then every induced subgraph of D has a kernel.*

Proof: We first note that the *induced subgraphs* of D are precisely the line graphs of the *subgraphs* of H, because vertex removal from G (as well as from D) means edge removal from H. Consequently, if we prove that the entire D itself has a kernel for a generic bipartite graph H, then we automatically obtain the result for all induced subgraphs of D, too.

Now, consider any orientation of D in which all triangles are transitively oriented. Since H is bipartite, there is only one way to obtain a triangle in its line graph: to take three edges of H which are incident with the same vertex. This is because any edge vv' of the line graph corresponds to two edges e, e' sharing a vertex in H. Then a third edge can intersect both of them only in their common vertex, otherwise H would not be bipartite.

Looking from the other side, the edges incident with a vertex of H correspond to a complete subgraph of G and to an oriented complete subgraph of D. The latter cannot contain *any* directed cycles: should $\vec{C}$ be any directed cycle, we would choose one of minimum length, and then it should be a triangle because inserting a diagonal of any orientation in a directed cycle always yields a shorter directed cycle. This implies that every complete subgraph of D contains a vertex of zero out-degree, from which it follows that the complete subgraphs are oriented transitively in a linear order.

This corresponds to an ordering of the edges incident with a vertex in H. We interpret the ordering as a preference order, where orientation $e \to e'$ means that e' has higher preference than e. By the Stable Marriage Theorem (Theorem 5.11) there exists a stable matching M' in H with respect to any collection of such preference orders at the vertices. The edges occurring in M' are represented with a set M of vertices in both G and D. Since the edges in the matching M' are mutually vertex-disjoint, the vertices in M are independent. Moreover, stable matching means that every edge outside M' meets some edge of M' such that the latter has higher preference at their common vertex. Translating this to line graphs, every vertex outside M has an out-neighbor in M. Hence, the conditions for a kernel are satisfied by M, implying the validity of the lemma. □

Theorem 7.3 *If G is the line graph of a bipartite graph H, then $\chi_\ell(G) = \chi(G) = \Delta(H)$.*

Proof: We know from Theorem 5.9 that $\chi(G) = \Delta(H)$ holds; hence we need to prove that every Δ-assignment on G admits a list coloring.

We begin with an auxiliary coloring, which has nothing to do with the lists but nevertheless will be very useful. We take a proper edge coloring c' of H with the colors $1, 2, \ldots, \Delta$. Then c' corresponds to a proper vertex coloring c of G in the natural way: if vertex v of G represents edge e of H, we define $c(v) = c'(e)$. From this coloring we derive an orientation on the edges of G as follows. Denote by A and B the two vertex classes of H. If vv' is an edge in G, then the corresponding edges e and e' of H meet either in A or in B. If they meet in A, then we orient the edge vv' from smaller color to larger; and if they meet in B, then we orient vv' from larger color to smaller.

Claim: In this orientation, every out-degree is at most $\Delta - 1$.

Proof: Suppose that $c(v) = i$ for vertex v. Then v can have at most $\Delta - i$ out-neighbors with colors higher than i (they correspond to edges of H that meet the edge of e in A), and at most $i - 1$ out-neighbors with lower colors (these are the vertices whose edges meet the edge of e in B). These are at most $\Delta - 1$ neighbors for v altogether. $\Diamond$

Now we can put the pieces together. We see from Lemma 7.1 that every induced subgraph of D has a kernel. So, the kernel precondition in Theorem 7.2 is satisfied, therefore G is list colorable whenever we have $|L_v| > d^+(v)$ for all $v \in V$ in the current orientation. This is ensured by the previous claim and the assumption that the lists have cardinality Δ. $\square$

7.5 Planar graphs

Planar graphs are defined as the graphs which admit a drawing in the Euclidean plane in such a way that the vertices are points and the edges are non-crossing closed curves. If a graph is planar, then it also has a planar drawing where the edges are straight-line segments. ***Euler's formula*** states that every planar embedding of a *connected* planar graph with v vertices, e edges and f faces (regions) satisfies the equality

$$v + f = e + 2.$$

This formula can be applied in many computations; for instance, it implies the following upper bounds.

Lemma 7.2

1. *A planar graph on n vertices can have at most $3n - 6$ edges.*
2. *A planar graph on n vertices and not containing K_3 as a subgraph can have at most $2n - 4$ edges.*

Since the sum of vertex degrees is the double of the number of edges, it follows that every planar graph contains a vertex of degree at most five, and every planar graph without triangles contains a vertex of degree at most three. This implies the respective upper bounds 6 and 4 on the choice number of such graphs. The latter bound is tight, but the former isn't: every planar graph is 5-choosable, and there exist planar graphs which are not 4-choosable.

Here we apply some of the previous results, to prove the following upper bound.

Theorem 7.4 *Every planar bipartite graph is 3-choosable.*

Proof: Let G be a planar bipartite graph. Every subgraph of G, too, is planar; therefore Lemma 7.2 implies that any t vertices of G induce fewer than $2t$ edges, for any value of t. Thus, by Theorem 5.10 we obtain that the edges of G can be oriented in a way that yields a digraph D with maximum out-degree at most 2. We also see from part 2 of Theorem 7.1 that D and all of its subgraphs contain kernels. Consequently, Theorem 7.2 implies that G is list colorable whenever the list sizes exceed the out-degrees of the vertices in D. In particular, we obtain that G is 3-choosable. □

Remark 7.4 *The upper bound 3 in the theorem is tight: we have seen that the planar graph $K_{2,4}$ is not 2-choosable.*

Remark 7.5 *Lemma 7.2 implies that every planar graph admits an orientation with maximum out-degree at most 3. This cannot be applied to prove via Theorem 7.2, however, that all planar graphs are 4-choosable (this is not even true) because many planar graphs have no kernels.*

Chapter 8

Edge decompositions of graphs

Decomposition techniques are widely used; we have already seen the example of tree decomposition which was an extremely useful tool in solving algorithmic problems on a large class of combinatorial structures. The kind of decompositions considered here is quite different, however, requiring a different way of thinking, and a first warning should say that one must not mix the two.

Definition 8.1 *An **edge decomposition** of a graph $G = (V, E)$ is a partition of its edge set into some subgraphs $F_1, \ldots, F_m$ where $F_i = (V_i, E_i)$, $V_i \subseteq V$ for all $1 \leq i \leq m$, the sets E_i are mutually disjoint and their union is E. In other words, each edge of G occurs in precisely one of the subgraphs F_i.*

Example 8.1 *In every proper edge coloring, the color classes (the monochromatic sets of edges) form a decomposition into matchings. In particular,*

1. *The edge coloring of a d-regular bipartite graph with d colors is a decomposition into perfect matchings.*

2. *Putting each edge of any graph G into a singleton class we trivially obtain a decomposition into copies of K_2.*

In the two numbered sub-examples above, the subgraphs of the decomposition are isomorphic (to a perfect matching and to an edge, respectively), while in the general concept of proper edge coloring all subgraphs of the decomposition belong to a given type (matching). Both versions are of interest and we shall see results of both types.

Definition 8.2 *Let F be a fixed graph. An **F-decomposition** of a graph G is an edge decomposition $F_1, \ldots, F_m$ of G such that all subgraphs F_i are*

isomorphic to F. If G admits an F-decomposition, we also say that F ***decomposes*** *G, or G is* ***decomposable*** *into F.*

We shall mostly deal with edge decompositions of ***complete*** graphs; but before that, we mention the following theorem.

Theorem 8.1 *A connected graph is decomposable into paths of length two if and only if it has an even number of edges.*

One way to prove this result is to start with an arbitrarily chosen spanning tree, and to show that it has a leaf such that either it has degree at least two in the original graph, or the unique edge incident to the leaf together with an edge incident to its neighbor induce a P_3 whose edge-removal keeps the rest of the graph connected. Along these lines, the theorem follows by induction on the number of edges.

8.1 Perfect matchings, Hamiltonian subgraphs

The decompositions of complete graphs presented in this section can be described in a geometric way, applying the idea of symmetry. We shall take regular polygons, for which we introduce the following occasional notation.

Notation 8.1 *For $k \geq 3$ we denote the regular k-gon by R_k.*

We give three constructions, the second and third of them being closely related.

Theorem 8.2 *The complete graph on $n \geq 2$ vertices is decomposable into perfect matchings if and only if n is even.*

Proof: Since a perfect matching puts the vertices into pairs, a graph (not only a complete one) can have a perfect matching only if the number of its vertices is even. Hence the parity condition is necessary.

To prove decomposability for K_n with $n \geq 4$ even, we arrange $n-1$ vertices $v_1, \ldots, v_{n-1}$ in the shape of a regular $(n-1)$-gon R_{n-1} and put the last vertex v_n in its geometric center. The vertex degrees are equal to $n-1$, therefore we need to find exactly $n-1$ mutually edge-disjoint perfect matchings.

The idea is to specify one perfect matching M_i for each edge $v_i v_n$ ($i = 1, \ldots, n-1$). The geometric representation will make this transparent: for

v_iv_n we take those diagonals of R_{n-1} which are orthogonal to v_iv_n (also including one side of R_{n-1} for each i). Formally one can write

$$M_i = \{v_iv_n\} \cup \{v_{i-j}v_{i+j} \mid 1 \leq j \leq n/2 - 1\}$$

where subscript addition is taken modulo $n-1$, i.e. $v_{-k} = v_{n-1-k}$ for $k \leq 0$ and $v_{n-1+k} = v_k$ for $k > 0$, wherever applicable. In this way each M_{i+1} is obtained from M_i by a rotation of $\frac{2\pi}{n-1}$ (that is, $\frac{360}{n-1}$ degrees) clockwise or counterclockwise, depending on the direction of labeling the vertices in R_{n-1}.

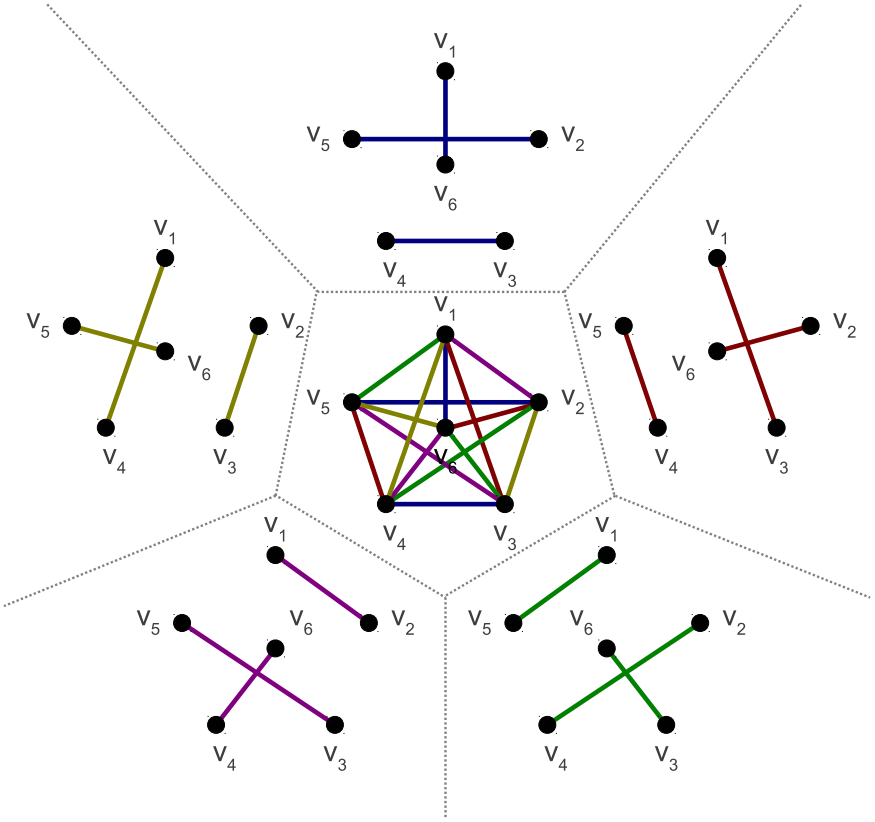


Figure 8.1: *Perfect matching decomposition of* K_6

The only missing even number is $n = 2$, which is trivial because K_2 has just one edge and it forms itself a trivial "decomposition" into one perfect matching. □

Theorem 8.3 *The complete graph* K_n *on* $n \geq 2$ *vertices is decomposable into Hamiltonian paths if and only if* n *is even.*

Proof: Since K_n has $\binom{n}{2} = \frac{n(n-1)}{2}$ edges and each Hamiltonian path contains $n-1$ edges, the decomposition has to contain exactly $n/2$ subgraphs; this number should be an integer, therefore n must be even whenever a P_n-decomposition exists.

Suppose that $n \geq 2$ is even. The case $n = 2$ is trivial because K_2 has just one edge, therefore the graph itself forms a "decomposition" with one class. Hence, from now on we only have to deal with $n \geq 4$.

We now arrange the n vertices $v_1, \ldots, v_n$ to form a regular n-gon R_n. To simplify the formulas, let us write $k = n/2$; we need to find k edge-disjoint Hamiltonian paths. Note that R_n has k long diagonals, and $2k$ diagonals from any other length (including the sides, too). Viewing them geometrically, the diagonals are in $2k$ parallel classes: k of their directions are determined by the sides v_iv_{i+1} $(i = 1, \ldots, k)$, for example the side $v_{k+1}v_{k+2}$ is parallel to v_1v_2; and the other k directions are determined by the short diagonals v_iv_{i+2} (again $i = 1, \ldots, k$). Therefore, it will suffice to show that a suitable coupling of those parallel classes creates a P_n-decomposition. This can be done by the observation that

$$v_1v_2v_nv_3v_{n-1}\ldots v_kv_{k+2}v_{k+1}$$

is a Hamiltonian path composed from the classes of the side v_1v_2 and of the short diagonal v_nv_2. The suitably chosen k positions (rotations) of such paths decompose K_n. Namely, putting the classes of v_iv_{i+1} and $v_{i-1}v_{i+1}$ together $(i = 1, \ldots, k)$ we obtain a P_n-decomposition. Formally, for $i = 1, \ldots, k$ we consider

$$\{v_{i-j}v_{i+1+j} \mid 0 \leq j \leq k-1\} \cup \{v_{i-j}v_{i+j} \mid 1 \leq j \leq k-1\}$$

where subscript arithmetics are now meant modulo n (by the convention $v_{-\ell} = v_{n-\ell}$ and $v_{n+\ell} = v_\ell$ for every $\ell \geq 0$). The first set in the union above is the parallel class containing the side v_iv_{i+1} while the second set is the parallel class containing the short diagonal $v_{i-1}v_{i+1}$. □

Theorem 8.4 *The complete graph K_n on $n \geq 3$ vertices is decomposable into Hamiltonian cycles if and only if n is odd.*

Proof: Since K_n has $\binom{n}{2} = \frac{n(n-1)}{2}$ edges and each Hamiltonian cycle contains n edges, the decomposition has to contain exactly $\frac{n-1}{2}$ subgraphs; this number should be an integer, therefore n must be odd whenever a C_n-decomposition exists.

Suppose that $n \geq 3$ is odd. The case $n = 3$ is trivial because K_3 is isomorphic to C_3, therefore itself is a Hamiltonian cycle, forming a "decomposition"

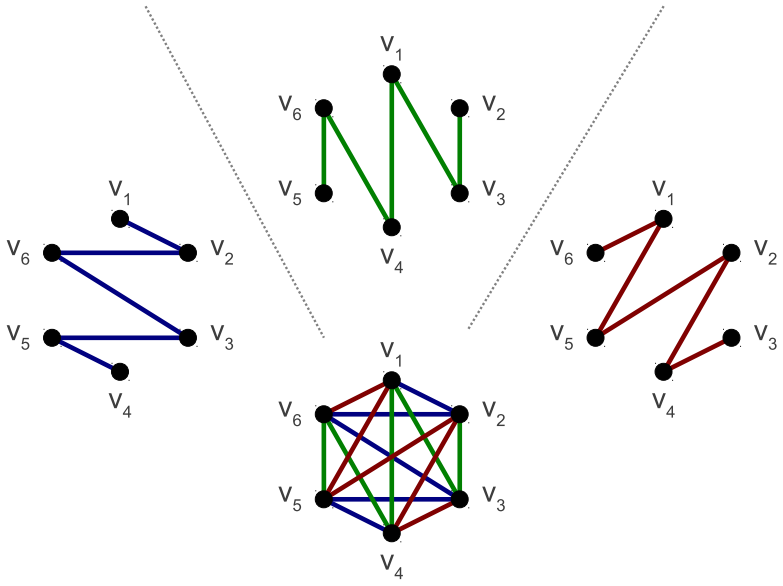


Figure 8.2: *Hamilton path decomposition of* K_6

with one class. If n is larger, we take the previous decomposition into Hamiltonian paths over $n-1$ vertices and extend it with one further vertex v_n. The former construction is composed of paths whose ends are v_i and v_{k+i}, for $i = 1, \ldots, k$. Each of these paths can be extended to a Hamiltonian cycle of K_n by adjoining the edges $v_i v_n$ and $v_{k+i} v_n$. □

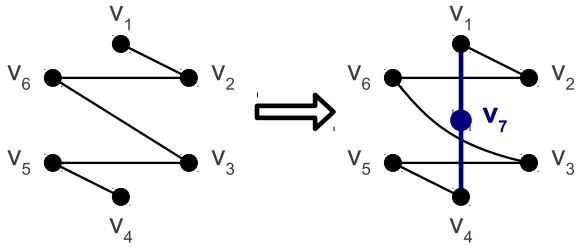


Figure 8.3: *Hamilton cycle decomposition of* K_7

Below we give a more detailed explanation for Hamiltonian cycles, which is self-contained, not using the decomposition into paths. We arrange $n-1$ vertices $v_1, \ldots, v_{n-1}$ to form a regular $(n-1)$-gon R_{n-1} and place v_n into its center. To simplify the formulas, let us write $k = \frac{n-1}{2}$; we need to find

k edge-disjoint Hamiltonian cycles. Now our view is that R_{n-1} has exactly k long diagonals, which are split into two segments by the geometric center. The Hamiltonian cycles to be constructed will be organized along those long diagonals.

Recall that $n-1=2k$, hence v_i and v_{k+i} are antipodal vertices of R_{k-1}. The first cycle will contain the edges v_nv_1 and v_nv_{k+1}, the short diagonal v_2v_{2k} together with all diagonals parallel to it (they are orthogonal to the long diagonal v_1v_{k+1}, but this diagonal is not taken for the cycle), moreover the side v_1v_2 and all diagonals parallel to it, also including the opposite side $v_{k+1}v_{k+2}$.

One can verify that these edges form a Hamiltonian cycle of K_n. We obtain the further $k-1$ cycles by geometric rotation with angle π/k. Formally, for $i=1,\dots,k$ we consider

$$\{v_nv_i, v_nv_{k+i}\} \cup \{v_{i-j}v_{i+j} \mid 1 \le j \le k-1\} \cup \{v_{i-j}v_{i+1+j} \mid 0 \le j \le k-1\}$$

where again subscript arithmetics are meant modulo $n-1$. These are $2+(k-1)+k=2k+1=n$ edges for each cycle.[1]

8.2 Complete bipartite graphs

So far we considered decompositions into one particular graph F. In this section the view will be somewhat different, we specify only the type of graphs into which K_n should be decomposed. That is, we consider the family

$$\mathcal{F} = \{K_{a,b} \mid a \ge 1,\, b \ge 1\}$$

of complete bipartite graphs. We want to find edge decompositions $F_1,\dots,F_m$ of K_n in which every subgraph is isomorphic to some member of $\mathcal{F}$.

Example 8.2 *Let $\{v_1,v_2,\dots,v_n\}$ be the vertex set of K_n. We can get a decomposition into $n-1$ stars, one with i edges for each $i=1,\dots,n-1$, for example with edge set*

$$\{v_iv_j \mid i<j\le n\}.$$

This subgraph is isomorphic to $K_{1,n-i}$.

[1] Each cycle contains one long diagonal, namely $v_{i-k/2}v_{i+k/2}$ if k is even (i.e., if $n-1$ is a multiple of 4) and $v_{i-(k-1)/2}v_{i+(k+1)/2}$ if k is odd (i.e., if $n+1$ is a multiple of 4).

Example 8.3 *More generally, we can apply the following recursive construction. Let $n = n_1 + n_2$, with positive integers n_1, n_2. Partition the vertex set into two subsets V_1, V_2 of cardinalities $|V_1| = n_1$ and $|V_2| = n_2$. Decompose the edge set inside V_i ($i = 1, 2$) into $n_i - 1$ complete bipartite subgraphs. The non-covered edges form a complete bipartite subgraph K_{n_1,n_2} with vertex classes V_1 and V_2. In this way we recursively obtain a decomposition into*

$$(n_1 - 1) + (n_2 - 1) + 1 = n - 1$$

complete bipartite subgraphs. (The previous example is the particular case $|V_1| = 1$, applied recursively in each step.)

Hence the number of decompositions of K_n with exactly $n - 1$ complete bipartite subgraphs grows very fast as n gets large. Interestingly enough, decomposition with fewer subgraphs is not possible. The proof of this result is kind of magic, a very nice application of linear algebra. It is worth noting that no combinatorial proof is known for the theorem so far.

Theorem 8.5 *If $F_1, \ldots, F_m$ is a decomposition of K_n into complete bipartite subgraphs, then $m \geq n - 1$.*

Proof: We represent the vertices of K_n with real variables $x_1, \ldots, x_n$. An edge v_iv_j will be represented with the product x_ix_j. If H is a subgraph of K_n with edge set E, we associate with H the sum of its edges,

$$s(H) = \sum_{v_iv_j \in E} x_ix_j,$$

taking one term for each edge.

Let us compute the s-value for a complete bipartite subgraph. If F_ℓ has vertex classes A_ℓ and B_ℓ, then F_ℓ has the edge set

$$\{v_iv_j \mid v_i \in A_\ell,\, v_j \in B_\ell\}.$$

Hence, the corresponding sum is

$$s(F_\ell) = \sum_{v_i \in A_\ell,\, v_j \in B_\ell} x_ix_j = \left(\sum_{v_i \in A_\ell} x_i \right) \left(\sum_{v_j \in B_\ell} x_j \right).$$

On the other hand, with a little algebraic manipulation from the multinomial theorem, we obtain that the sum associated with the complete graph K_n is

$$s(K_n) = \sum_{1 \leq i < j \leq n} x_ix_j = \frac{1}{2} \left(\left(\sum_{i=1}^{n} x_i \right)^2 - \left(\sum_{i=1}^{n} x_i^2 \right) \right).$$

If $F_1, \ldots, F_m$ is a decomposition of K_n into complete bipartite subgraphs F_ℓ with vertex classes A_ℓ and B_ℓ, then by definition we have

$$s(K_n) = \sum_{\ell=1}^{m} s(F_\ell)$$

from what, by substitution, the observations above yield:

$$\frac{1}{2}\left(\sum_{i=1}^{n} x_i\right)^2 - \frac{1}{2}\left(\sum_{i=1}^{n} x_i^2\right) = \sum_{\ell=1}^{m}\left(\sum_{v_i \in A_\ell} x_i\right)\left(\sum_{v_j \in B_\ell} x_j\right). \tag{8.1}$$

Now the great trick comes. Consider the following system of $m+1$ homogeneous linear equations over n variables:

$$\begin{aligned} x_1 + \cdots + x_n &= 0 \\ \sum_{v_i \in A_1} x_i &= 0 \\ \sum_{v_i \in A_2} x_i &= 0 \\ &\vdots \\ \sum_{v_i \in A_m} x_i &= 0 \end{aligned} \tag{8.2}$$

Look what this means for Equation (8.1) if the real numbers x_i satisfy all equations in (8.2). The right side is zero because the term for each F_ℓ is a product of two sums, the first of them being zero by the assumption that we took a solution of (8.2). Also the first term on the left side of (8.1) is zero, by the first equation of (8.2). Consequently, every solution of (8.2) must satisfy

$$x_1^2 + \cdots + x_n^2 = 0.$$

Thus, we obtain

$$x_1 = \cdots = x_n = 0,$$

i.e. the system (8.2) has the trivial solution only.

We know from the theory of homogeneous linear equations that if nothing but the trivial solution exists then the number of equations has to be at least as large as the number of variables.[2] In our case this means

$$m + 1 \geq n$$

and this is what we had to prove. □

[2] In general: if there are p variables, q linearly independent homogeneous equations, and $p \geq q$ holds, then the space of solutions has dimension $p - q$.

8.3 Complete subgraphs

For complete subgraphs we consider both kinds of decompositions: F-decomposition into copies of a fixed graph F and $\mathcal{F}$-decomposition into members of a family $\mathcal{F}$ of graphs; more precisely, with the complete graph $F = K_p$ with a fixed $p \geq 3$ or with the family

$$\mathcal{F} = \{K_p \mid p \geq 2\}.$$

Of course, K_n "decomposes" K_n into one complete subgraph; we are interested in the other (nontrivial) decompositions.

8.3.1 Complete subgraphs of variable size

We first consider the problem of $\mathcal{F}$-decompositions, where the number of vertices in the complete subgraphs is not fixed.

Example 8.4 *Every complete graph K_n on n vertices can be decomposed into the same number n of complete subgraphs in the following way. Denoting the vertices by $v_1, v_2, \ldots, v_n$ we take the complete subgraph of order $n-1$ on the vertex set $\{v_1, v_2, \ldots, v_{n-1}\}$, and $n-1$ further complete subgraphs of order 2 which are the edges v_1v_n, v_2v_n, $\ldots$, $v_{n-1}v_n$.*

This construction is optimal, as shown by the following result.

Theorem 8.6 *If $F_1, \ldots, F_m$ is a decomposition of K_n into $m \geq 2$ complete subgraphs each having at least two vertices, then $m \geq n$.*

Proof: We shall assume that the inequality is not valid, and derive a contradiction from it. Before that, we need some preparation. Let F_j have vertex set V_j, and let us denote $n_j = |V_j|$ for $j = 1, 2, \ldots, m$. Further, for vertex v_i, let us denote by d_i the number of subgraphs F_j containing v_i.

Claim: If $v_i \notin V_j$, then $d_i \geq n_j$.
Proof: For each $v_\ell \in V_j$, the vertex pair v_iv_ℓ is contained in precisely one subgraph of the decomposition. This subgraph is different from F_j because $v_i \notin V_j$, and also for any two different vertices of F_j these subgraphs are different because the edge connecting the two vertices is already contained in F_j. Hence, at least as many subgraphs contain v_i as the number of vertices in F_j. ◊

Now we are in a position to prove the theorem. Assume, for a contradiction, that $n > m$ holds. If $d_i \geq n_j$ also holds for a pair (i, j) of subscripts

— which is the case by the Claim above if $v_i \notin V_j$ — then we see that $n \cdot d_i > m \cdot n_j$ is valid (because $n_j > 0$). It is a matter of routine to check that this latter inequality is equivalent to the following one:

$$\frac{1}{n(m-d_i)} > \frac{1}{m(n-n_j)}.$$

We sum up this for all pairs (i,j) such that $v_i \notin V_j$. It follows that

$$\sum_{\substack{i,j \\ x_i \notin V_j}} \frac{1}{n(m-d_i)} > \sum_{\substack{i,j \\ x_i \notin V_j}} \frac{1}{m(n-n_j)}.$$

The proof will be done if we show the surprising fact that both sides of this inequality are equal to 1. Once we prove this, the contradiction $1 > 1$ will follow immediately.

For the left side:

$$\sum_{\substack{i,j \\ x_i \notin V_j}} \frac{1}{n(m-d_i)} = \sum_{i=1}^{n} \sum_{\substack{j \\ x_i \notin V_j}} \frac{1}{n(m-d_i)} = \sum_{i=1}^{n} (m-d_i) \cdot \frac{1}{n(m-d_i)} = \sum_{i=1}^{n} \frac{1}{n} = 1$$

because, by definition, d_i of the m subgraphs contain v_i, hence exactly $m-d_i$ do *not* contain it; this fact verifies the equality in the middle.

For the right side:

$$\sum_{\substack{i,j \\ x_i \notin V_j}} \frac{1}{m(n-n_j)} = \sum_{j=1}^{m} \sum_{\substack{i \\ V_j \not\ni x_i}} \frac{1}{m(n-n_j)} = \sum_{j=1}^{m} (n-n_j) \cdot \frac{1}{m(n-n_j)} = \sum_{j=1}^{m} \frac{1}{m} = 1$$

because each F_j has n_j vertices, hence exactly $n-n_j$ vertices are outside it.

Thus, the contradiction $1 > 1$ implies that the assumption $n > m$ was false and the theorem is true. □

8.3.2 Complete subgraphs of fixed size

Here we consider K_p-decompositions of complete graphs. There are some number-theoretic conditions which are necessary for the existence of such decompositions.

Proposition 8.1 (Integrality Conditions) *If K_p decomposes K_n, then*

- $\binom{n}{2}$ *is a multiple of* $\binom{p}{2}$,

- $n-1$ *is a multiple of* $p-1$.

Proof: The first condition follows from the fact that K_n and K_p have $\binom{n}{2}$ and $\binom{p}{2}$ edges, respectively, therefore the decomposition has to contain exactly $\binom{n}{2}/\binom{p}{2}$ subgraphs, which must be an integer. The second condition is obtained by comparing vertex degrees: they are $n-1$ in K_n and $p-1$ in K_p, therefore each vertex has to occur in precisely $\frac{n-1}{p-1}$ of the subgraphs. □

These conditions are not always sufficient for the existence of K_p-decomposition, but they are "almost sufficient", as expressed in the following result.

Theorem 8.7 *For every integer $p \geq 3$ there exists a threshold value $n_0 = n_0(p)$ such that, for every $n > n_0$, K_p decomposes K_n if and only if the Integrality Conditions are satisfied.*

Example 8.5 *For $p = 3$ — that is, $F = K_3$, decomposition into triangles — the Integrality Conditions mean:*

- $\binom{n}{2}$ *is divisible by 3,*
- $n-1$ *is even.*

Simple analysis of cases modulo 6 yields that these conditions are satisfied if and only if n is of the form $6k+1$ or $6k+3$. It can be proved that all those values of n admit decompositions into K_3, i.e. the conditions are not only necessary but also sufficient for $p = 3$. In such a decomposition, the vertex sets of the copies of K_3 yield a 3-uniform set system called ***Steiner Triple System****.*

8.4 Notes

We have seen that both inequalities $m \geq n-1$ and $m \geq n$ (for complete bipartite subgraphs and for complete subgraphs, respectively) are tight, but for the former there has been a much richer family presented which attains equality. Actually, the complete bipartite decompositions with $m = n-1$ have not been characterized so far, this would be an open problem for research. On the other hand, conditions are known which are necessary and sufficient for the complete subgraph decompositions with $m = n$; it turns out that, beside the simple construction given above, there is only one further family of decompositions. We shall see them in the chapter on finite projective planes; they require far more elaborated methods than the previous ones.

8.4.1 Double enumeration

In the proof of Theorem 8.5 we took the sum of inequalities over all pairs (i, j) such that $v_i \notin V_j$, and then we manipulated with the sums on the two sides. The idea behind this way of computation can be represented with a bipartite graph, say B, where the vertices $v_1, \ldots, v_n$ of K_n are put in the first vertex class of B, and the subgraphs $F_1, \ldots, F_m$ correspond to the vertices in its second vertex class. Edges $e_{i,j}$ of B represent the terms in the sum; i.e., $v_i F_j$ is an edge in B if and only if $v_i \notin V_j$. Each edge $e_{i,j}$ has a value say $a_{i,j}$ at its end in the first class and a value say $b_{i,j}$ at its end in the second class; these are the corresponding terms of the inequality on the left and right side, respectively. In our case $a_{i,j} > b_{i,j}$ holds for all pairs (i, j), from what we see immediately that the sum of the a-values is larger than that of the b-values.

One key point in the proof was how to evaluate the two sums. In this bipartite representation, each term corresponds to exactly one edge, and so one may say that the summation is taken over the edges of B. We simplified the computation by grouping the terms in a convenient way. The reader can observe that each group of terms corresponds to the edges incident with a vertex v_i for the left-side sum, and to those incident with a subgraph G_j for the right-side sum. Since each edge has exactly one end in each vertex class of B, this grouping ensures that each term has been counted exactly once on both sides.

Chapter 9

Finite projective planes

In this chapter we consider a very special kind of finite set systems, which have many nice structural properties. The name "projective plane" refers to strong analogy with geometry, which we shall comment at the end of the chapter. We shall use the terms ***point*** for the elements of the underlying set and ***line*** for the sets of the set system.

Definition 9.1 (Axioms of finite projective planes of order q)

A1 Any two points are contained together in exactly one line.

A2 Any two lines intersect in exactly one point.

A3 There is a line with exactly $q+1$ points.

A4 There are four points, no three of which are on the same line.[1]

*A pair $(P, \mathcal{L})$ is a **projective plane of order** q if $\mathcal{L}$ is a set system over the set P, and the elements $p \in P$ as points and the sets $L \in \mathcal{L}$ as lines satisfy the four "axioms" A1–A4 above. As for terminology, if $p \in L$, then we say that point p and line L are **incident**, or p lies on L, or L passes through p.*

Although these four axioms look simple, lots of structural properties can be derived from them. Quantitatively, one can prove the facts below.

Theorem 9.1 *Every projective plane of order q has the following parameters.*

[1] This requirement is often formulated in the way that there is a *quadrangle* in the plane. If no three of a set of points are on the same line, those points are usually said to be in *general position*. Hence, a quadrangle means four points in general position, and A4 requires that such a 4-tuple must occur in every finite projective plane.

1. *The number of points is* $q^2 + q + 1$.

2. *The number of lines is* $q^2 + q + 1$.

3. *Every line has exactly* $q + 1$ *points.*

4. *Every point is incident with exactly* $q + 1$ *lines.*

Sketch of Proof: Choose line L_0 with exactly $q + 1$ points, say $p_1, \ldots, p_{q+1}$ (by A3). If $p \notin L$ is a point outside L, then each p_i $(1 \leq i \leq q+1)$ determines precisely one line with p (by A1) and those lines are mutually distinct (by A2, investigating their intersection with L). No more lines can pass through p because they would meet L (by A2) in points different from the previous $q+1$ ones (by either of A1 and A2). Hence, *all* points outside L — or outside any line with exactly $q + 1$ points — are incident with exactly $q + 1$ lines. Switching the role of "point" and "line" in this argument we obtain that if p is any point incident with exactly $q + 1$ lines and L is any line not passing through p, then L has exactly $q+1$ points. The four points in general position (A4) ensure that there are at least two points p, p' outside L, both contained in exactly $q+1$ lines, only one of which — say, L' — is passing through both of them. All of the $2q$ other lines meeting $\{p, p'\}$ have exactly $q + 1$ points, and no point is contained in all of them. Therefore all points lie on exactly $q + 1$ lines and all lines have exactly $q + 1$ points.

Each p_i $(1 \leq i \leq q+1)$ is incident with q lines different from L, those lines are mutually distinct and together with L they include all lines. Similarly, each of the $q + 1$ lines passing through p contain exactly q points different from p, and their union covers all points. This yields the validity of the assertions concerning $q^2 + q + 1$. □

Remark 9.1 *Since any two points belong to exactly one line, we may also view projective planes as edge decompositions of complete graphs into complete subgraphs. Moreover, by the previous result, the number of points and lines is the same, therefore every finite projective plane provides us with an example of edge decomposition of* K_{q^2+q+1} *in which the number of* K_{q+1} *subgraphs is equal to the number of vertices. Conversely, however, the construction given in Example 8.4 with* $m = n$ *cannot be viewed as a projective plane. Indeed, the large subgraph with* $n - 1$ *vertices in the decomposition meets every 4-tuple in at least three elements and therefore axiom A4 does not hold for this structure, although A1 and A2 are valid. As a matter of fact, the goal of introducing A4 as a requirement is to exclude the non-symmetric decomposition from the class of finite projective planes.*

Example 9.1 *The **Fano plane** is the smallest example of finite projective planes, with $q = 2$. It has 7 points and 7 lines, they can be drawn in a convenient way in an equilateral triangle T. The points are the three vertices, three midpoints of sides, and the geometric center of T. Three of the lines are the sides, three others are the medians, while the seventh line of the Fano plane is represented with the incircle of T.*

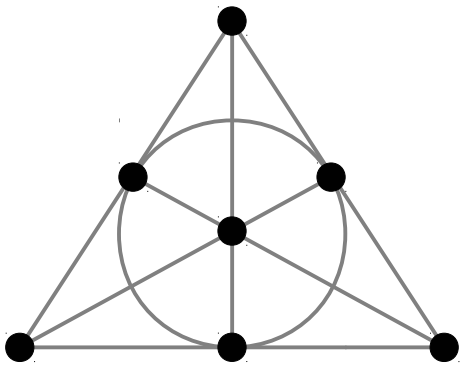


Figure 9.1: *Fano plane with $q = 2$*

It is a major open problem in combinatorics to characterize the values of q for which a projective plane of order q exists. Prime powers $q = p^\alpha$ (where p is a prime number and α is a positive integer) play a central role in this problem.

Conjecture 9.1 *If there exists a projective plane of order q, then q is a prime power.*

Theorem 9.2 *If q is a prime power, then there exists a projective plane of order q.*

We shall prove this fact in the sequel. For small orders we also know that the conjecture is true: while prime powers $2, 3, 4, 5, 7, 8, 9, 11,...$ admit constructions, while there do not exist any projective planes of orders 6 and 10. Non-existence for 6 was known (in a different but equivalent form) already in the 18^{th} century, but a proof for 10 was given only in the 1970's with extensive use of computer.

We shall prove Theorem 9.2 using abstract algebra. The necessary prerequisites are given in the next section.

Remark 9.2 *Some values of q admit a unique projective plane of order q, namely the one which is constructed below. There exist some other values of q, however — the smallest one being $q = 9$ — for which it is known that several different projective planes exists.*

9.1 Finite fields

We recall from earlier studies (first-year course) that the algebraic structure called ***field*** has a set X of elements and is equipped with two algebraic operations, addition and multiplication, for which we shall use the standard symbols $+$ and $\times$. Both operations are required to be commutative ($a+b = b+a$, $a \times b = b \times a$) and associative ($a+(b+c) = (a+b)+c$, $a \times (b \times c) = (a \times b) \times c$) and satisfy the rule of distributivity ($a \times (b+c) = a \times b + a \times c$) for all elements $a, b, c \in X$. The neutral elements of addition and multiplication are traditionally denoted by 0 and 1 ($a + 0 = a$, $a \times 1 = a$); moreover, each $a \in X$ is required to have an additive inverse ($a + x = 0$) and, except for $a = 0$, also a multiplicative inverse ($a \times y = 1$).

We are now interested in fields over *finite* sets. In this area, the fundamental theorem of Galois states:

Theorem 9.3 *A finite field over $q \geq 2$ elements exists if and only if q is a prime power.*

From this very important result, we shall apply the 'if' part; i.e., that for every prime power q there exists a field of order q. This field, which is unique (up to isomorphism) for every q, is called the ***Galois field of order*** q.

Notation 9.1 *The Galois field of order q is denoted by $GF(q)$.*

One way to describe the Galois field of a given order is to list the results of its operations. This can be done, e.g., by giving its *additive table* for the results of addition and *multiplicative table* for those of multiplication. The simplest case, $q = 2$ yields[2]

$+$	0	1
0	0	1
1	1	0

$\times$	0	1
0	0	0
1	0	1

[2] There is a natural interpretation of these binary tables in terms of set-theoretic operations, too: $+$ corresponds to the symmetric difference $(A \setminus B) \cup (B \setminus A)$, whereas $\times$ corresponds to the intersection $A \cap B$.

In general, if we represent the elements of $GF(q)$ with nonnegative integers $0, 1, \ldots, q-1$ then the results of the two operations can be computed as addition and multiplication ***modulo*** q. For example, if $q = 3$ then we have

$+$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

$\times$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

while for $q = 5$ the tables are

$+$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

$\times$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

If q is not a prime, however, then computing modulo q does not work. For example, if $q = 4$ then 2 times 2 modulo 4 would yield zero and we cannot accept this, because then 2 would not have a multiplicative inverse. Also, taking the analogous cyclic table for addition modulo 4 would not allow a multiplicative table that satisfies the rule of distributivity.[3] Hence, for $q = 4$ — which is a prime power but not a prime — the tables are different from the previous examples:

$+$	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

$\times$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

In this case the additive table, viewed as a square matrix, is composed of 2 by 2 submatrices whose arrangement and structure follows that of $GF(2)$.

9.2 Galois planes

In the construction of the projective plane of order q we apply here Galois fields. The plane built upon $GF(q)$ is called the ***Galois plane of order*** q.

We represent the points and lines with ***homogeneous coordinate triples***:

[3] One can observe that the rules $0 \times a = 0$ and $1 \times a = a$ (for all $a \in GF(4)$ uniquely determine the multiplicative table. And then, if the additive table is computed modulo 4, there occur triples a, b, c of elements violating the distributive rule.

- (a, b, c) for a point,
- $[x, y, z]$ for a line,

none of them being the all-zero triple $0, 0, 0$. By 'homogeneous' we mean that, for every nonzero element $\lambda \in GF(q) \setminus \{0\}$, the triples

- (a, b, c) and $(\lambda \times a, \lambda \times b, \lambda \times c)$ mean the same point,
- $[x, y, z]$ and $[\lambda \times x, \lambda \times y, \lambda \times z]$ mean the same line.

Point (a, b, c) and line $[x, y, z]$ are ***incident*** if and only if

$$a \times x + b \times y + c \times z = 0$$

where all operations are calculated in $GF(q)$. Hence, point-line incidence means that the *scalar product* of their coordinate vectors is zero, exactly in the same way as in analytic geometry.

Since $GF(q)$ has q elements, we can compose q^3 ordered triples of coordinates over it, from which we have excluded $(0, 0, 0)$ from the points and $[0, 0, 0]$ from the lines. Homogeneousness means identification of triples in $(q-1)$-tuples. Therefore, the number of points and lines we have defined in this way is equal to

$$\frac{q^3 - 1}{q - 1} = q^2 + q + 1.$$

We have to check that the axioms A1–A4 are satisfied. To do this, we apply linear algebra. Let $(a, b, c), (a', b', c')$ be two distinct points; this means that the two triples are linearly independent over $GF(q)$. A line $[x, y, z]$ passes through these two points if and only if it is incident with both of them, i.e. precisely when it satisfies the following system of linear equations:

$$\begin{aligned} a \times x + b \times y + c \times z &= 0 \\ a' \times x + b' \times y + c' \times z &= 0 \end{aligned}$$

Since the two equations are linearly independent, and we have three variables and two equations, the solution space has dimension 1. That is, from a basic nonzero solution $[x_0, y_0, z_0] \neq [0, 0, 0]$ all solutions are obtained as

$$[\lambda \times x_0, \lambda \times y_0, \lambda \times z_0]$$

where λ runs over the $q-1$ nonzero elements of $GF(q)$. Since we have represented the lines with *homogeneous* triples, those $q-1$ solutions correspond to the same one line. This proves A1. The proof of A2 is completely analogous,

we just begin with two lines $[x, y, z], [x', y', z']$ and search for a point (a, b, c) which is incident with both of them.

If we want to determine the number of lines passing through any one point (a, b, c), we need to compute the number of solutions of the equation

$$a \times x + b \times y + c \times z = 0.$$

Since we have three variables and just one equation, the solution space has dimension 2, that means q^2 triples from which only $q^2 - 1$ remain because $[0, 0, 0]$ is excluded. Multiplying with any $\lambda \neq 0$ the triple represents the same line, therefore the solutions are split into $(q-1)$-tuples. Consequently, the number of lines incident with any point is

$$\frac{q^2 - 1}{q - 1} = q + 1.$$

In a completely analogous way, starting from a line $[x, y, z]$ and searching for triples $[a, b, c]$ whose scalar product with $[x, y, z]$ is zero, we can derive that each line contains exactly $q + 1$ points. This verifies A3.

Finally, a quadrangle required by A4 is, for example, the following quadruple of points:

$$(0, 0, 1), \quad (0, 1, 0), \quad (1, 0, 0), \quad (1, 1, 1).$$

One can check that any three of these vectors are linearly independent over any field. Hence, picking any three of these four points, say (a, b, c), (a', b', c'), (a'', b'', c''), the system of equations

$$\begin{aligned} a \times x + b \times y + c \times z &= 0 \\ a' \times x + b' \times y + c' \times z &= 0 \\ a'' \times x + b'' \times y + c'' \times z &= 0 \end{aligned}$$

has just the trivial solution $x = y = z = 0$. This all-zero triple does not correspond to any lines, therefore the four points above are in general position, as required by A4.

Notation 9.2 *The Galois plane of order q is denoted by $PG(2, q)$.*

9.3 Projective plane, Euclidean plane

The Euclidean plane — which is the geometry of dimension 2 taught in standard curriculum — has its theory built from the undefined basic notions

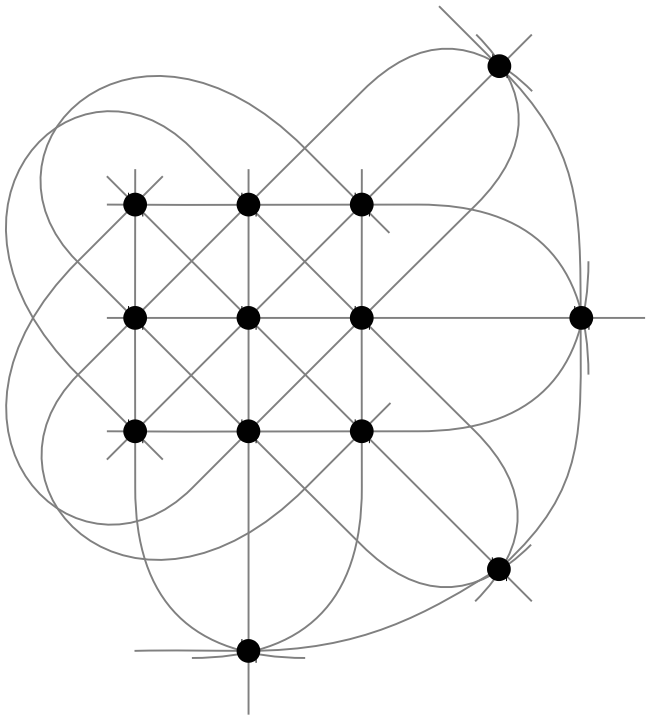


Figure 9.2: *Galois plane of order 3*

point and line, and from the axioms: properties which we assume to be true without proof. All theorems of planar geometry are derived from those axioms.

One of the axioms requires that any two points determine a straight line. It follows that any two lines meet in *at most one* point. There also exist parallel lines, however. The 'parallel postulate' ensures that for every line L and every point p outside L there is a unique line passing through p and not meeting L. Based on this axiom, the lines are in *parallel classes*: two lines belong to the same class if and only if they are parallel.[4]

The *projective plane* is an extension of the Euclidean plane, in the following way. For each parallel class $\mathbf{P}$ we take one further "point", say $x_{\mathbf{P}}$, which we call an "ideal point". Each line $L \in \mathbf{P}$ is extended with this $x_{\mathbf{P}}$, hence any two extended lines meet: in a "real" point if they are not parallel, and in an ideal point if they are parallel. In order to ensure that any two points determine a line, we define the "ideal line" to be the set of all ideal points.

Despite that extension with ideal points and with ideal line may look a bit artificial for first sight, the projective plane has many remarkable properties. One of them is point-line duality, which originates from the fact that its incidence axioms are symmetric: not only any two points determine exactly one line, but also any two lines determine exactly one point.

The theory of finite projective planes took much inspiration from that of the real projective plane. This also includes the way we introduced the coordinate triples for the points and lines over the Galois field. We have also seen some quantitative aspects of point-line duality in the finite case: the number of points and lines is the same, and the number of incidences on a point and on a line is the same.

[4] The relation of being parallel is an *equivalence relation* on the set of lines, therefore its equivalence classes form a partition, its classes consist of the mutually parallel lines.

Chapter 10

Extremal problems

Generally speaking, an extremal combinatorial problem asks for the (global) maximum or minimum of a function on a class of structures. The corresponding value is called extremum, and the structures on which this value is attained are called extremal.

In this chapter we consider extremal problems on graphs in which the number of edges has to be maximized or minimized. One classical kind of these questions are commonly termed as Turán-type problems, giving a tribute to the paper published by Pál Turán in 1941. We shall also discuss a recent optimization problem concerning communication networks, which turns out to be closely related to a case of the problem class just mentioned.

10.1 Forbidden subgraphs

The general question addressed here can be formulated as follows.

Problem 10.1 *Let F be a fixed "forbidden" subgraph. Determine the maximum number $\mathrm{ex}(n, F)$ of edges, as a function of n, in graphs on n vertices which do not contain F as a subgraph.*

As long as $n < |V(F)|$, we certainly have $\mathrm{ex}(n, F) = \binom{n}{2}$. It is also trivial that $\mathrm{ex}(n, K_2) = 0$. Below we disregard from these cases. One can also observe immediately that $\mathrm{ex}(n, P_3) = \lfloor \frac{n}{2} \rfloor$, because in a graph without P_3, any two edges must be vertex-disjoint. The first classical result concerning $\mathrm{ex}(n, F)$ is the following one.

Theorem 10.1 *If a graph $G = (V, E)$ on n vertices contains no K_3 as a subgraph, then $|E| \leq \lfloor n^2/4 \rfloor$. Equality holds if and only if G is the complete bipartite graph $K_{\lceil n/2 \rceil, \lfloor n/2 \rfloor}$.*

This result can be generalized for any $F = K_p$. In order to emphasize the structural aspects, we first state the assertion concerning extremal graphs rather than on extremum.

Theorem 10.2 (Turán's theorem) *For $n \geq p$, the graph having the largest number of edges without containing K_p as a subgraph is obtained by partitioning the vertices into $p-1$ classes $V_1, \ldots, V_{p-1}$ as equally as possible and join two vertices by an edge if and only if they belong to distinct classes V_i, V_j ($1 \leq i < j \leq p-1$) In particular, we have*

$$\mathrm{ex}\,(n, K_p) = \binom{n}{2} - \sum_{i=0}^{p-2} \binom{\lfloor \frac{n+i}{p-1} \rfloor}{2}.$$

Interestingly enough, for every fixed $p \geq 3$, the difference between this expression and the value of $\mathrm{ex}\,(n, G)$ grows more slowly than n^2 when n gets large. Using asymptotic notation one can write this in the form

$$\mathrm{ex}\,(n, G) = \frac{\chi(G) - 2}{2(\chi(G) - 1)}\, n^2 + o(n^2).$$

The formula in Theorem 10.2 may look somewhat complicated. Nevertheless, beside the particular case $p = 3$ of Theorem 10.1 it also gives a nice formula for $p = 4$:

Theorem 10.3 *For every $n \geq 4$ we have* $\mathrm{ex}\,(n, K_4) = \lfloor \frac{n^2}{3} \rfloor$.

We have to note, however, that if G is bipartite, then the numerator $\chi(G) - 2$ and hence also the coefficient of n^2 in the formula above becomes zero. This is called the 'degenerate case' of Turán's problem, and its asymptotic analysis is much harder than that for $\chi(G) \geq 3$. We shall see the interesting example of C_4 later.

10.2 A generalization and some proofs

There are many proofs known for Turán's theorem, and they apply quite different approaches. Hence this area also offers an instructive study for methodology. Below we present some typical arguments.

One way to extend the theorem to a more sensitive formula is to observe that an alternative summary of its message is:

If a graph has many edges, then its clique number is large.

The meaning of 'many' and 'large' can exactly be read out from the numerical formulation of the theorem. Switching to complementary graphs, this summary yields:

> *If a graph has few edges only, then its independence number is large.*

This version offers us the possibility to replace the global condition of 'few edges' with a more sensitive bound which uses the degree sequence.[1]

Theorem 10.4 *In any graph $G = (V, E)$ we have*

$$\alpha(G) \geq \sum_{v \in V} \frac{1}{d(v) + 1}. \tag{10.1}$$

First proof: Greedy selection of vertices. We apply the following algorithm:

- Initialize $S := \emptyset$.
- While $V(G) \neq \emptyset$, select a vertex v of minimum degree in G.
- Update $S := S \cup \{v\}$.
- Update $G := G - v - N(v)$.

That is, we delete all vertices adjacent to a selected vertex of minimum degree. At the end of this procedure we have $|S|$ at least as large as the right-hand side of (10.1). Since S is independent, the theorem follows.

Second proof: Greedy deletion of vertices. We apply the following algorithm:

- While G is not edgeless, identify a vertex v of maximum degree in G.
- Update $G := G - v$.

At the end of this procedure we have an edgeless graph — that is, an independent set — which has at least as many vertices as the right-hand side of (10.1). Hence, the theorem follows.

Third proof: Probabilistic method. We take an ordering $v_1, v_2, \ldots, v_n$ of the vertices at random. That is, all permutations are taken with probability $1/n!$. Define

$$v_i \in S \Longleftrightarrow v_i v_j \notin E(G) \text{ for all } 1 \leq j < i.$$

[1] Recall that the sum of vertex degrees is equal to the double of the number of edges.

We then have the probability

$$\mathbb{P}(v_i \in S) = \frac{1}{d(v_i)+1}.$$

Thus, the expected value of $|S|$ is equal to the right-hand side of (10.1). We can also observe that S is independent. Since expected value means weighted average, the theorem follows. □

10.3 Routing

The problem discussed in this section has its motivation in sending messages between nodes of a communication network. We assume throughout that $G = (V, E)$ is a connected graph (network). Messages have to be sent from each vertex to every other vertex.

Notation 10.1 *We denote by n the number of vertices and by m the number of edges. The vertices of G are denoted by $v_1, \ldots, v_n$.*

Definition 10.1 *A **routing** $\mathcal{R}$ is a collection of $n(n-1)$ paths $P_{i,j}$, one path for each ordered pair (v_i, v_j) of vertices. Path $P_{i,j}$ starts in v_i and ends in v_j.*

Remark 10.1 *If G is a tree, then any v_i, v_j are connected by a unique path, therefore $\mathcal{R}$ is completely determined by G. But if G contains at least one cycle, then there are pairs v_i, v_j of vertices for which $P_{i,j}$ can be specified in more than one possible way.*

Definition 10.2 *Given a routing $\mathcal{R}$ on $G = (V, E)$, the **load** of vertex $v_k \in V$ is the number of paths passing through v_k but not having v_k as an end; that is, the number of ordered pairs (i, j) such that v_k is an internal vertex of $P_{i,j}$. The load of v_k with respect to $\mathcal{R}$ is denoted by $\xi_{\mathcal{R}}(v_k)$.*

We measure the quality of a routing by the largest vertex load occurring in it; the larger this value is, the higher forwarding capacity some vertices of the network must have. For this reason our goal is to find routings in which no vertex has very large load; and to design networks which admit such routings.

Definition 10.3 *The **forwarding index** of G, denoted by $\xi(G)$, is the smallest possible value of the largest vertex load, taken over all routings $\mathcal{R}$ of G:*

$$\xi(G) = \min_{\mathcal{R}} \max_{1 \le k \le n} \xi_{\mathcal{R}}(v_k).$$

We derive a general lower bound in terms of the distances $d(v_i, v_j)$ between the vertices.

Theorem 10.5 *For every connected graph G with n vertices and m edges,*

$$\xi(G) \geq \frac{2}{n} \sum_{1 \leq i < j \leq n} (d(v_i, v_j) - 1) \geq n - 1 - \frac{2m}{n} \tag{10.2}$$

Proof: Let $\mathcal{R}$ be any routing. We first give a lower bound on the total vertex load. If two vertices $v_i, v_j \in V$ are at distance $d(v_i, v_j)$ apart, then every path connecting them — including $P_{i,j}$, too, that has been selected for $\mathcal{R}$ — has at least $d(v_i, v_j) - 1$ internal vertices and hence contributes with 1 to the load of each of them. Moreover, each pair of vertices is connected by two paths in $\mathcal{R}$, one from v_i to v_j and one from v_j to v_i. Thus,

$$\sum_{k=1}^{n} \xi_{\mathcal{R}}(v_k) \geq 2 \sum_{1 \leq i < j \leq n} (d(v_i, v_j) - 1)\,.$$

Each term in the sum of the right side is nonnegative, and the term for (i, j) is zero only if $v_i v_j$ is an edge. Therefore all the $\binom{n}{2}$ unordered pairs i, j but the m edges of G contribute with at least 1 to the sum, hence

$$2 \sum_{1 \leq i < j \leq n} (d(v_i, v_j) - 1) \geq n(n-1) - 2m.$$

These inequalities are valid for all routings, moreover the largest vertex load is not smaller than the average load, which is equal to $1/n$ times the sum of loads. Consequently, lower bounds are obtained on $\xi(G)$ if we divide the expressions above by n. Thus, $\xi(G)$ is at least as large as claimed in the theorem. □

The second inequality in (10.2) holds with equality if and only if G has diameter two. Indeed, otherwise the pairs of vertices at large distance apart contribute to the sum of loads with large numbers.

Disregarding the constant -1, the rightmost side of inequality (10.2) tells us that in order to have the forwarding index as small as about $(1 - c)n$ for some constant $c > 0$ as n gets large, the network has to contain as many as about $cn^2/2$ direct connections, which means a dense structure. We now prove that much fewer edges — namely, a little fewer than $n^{3/2}/2 + n/4$ — still suffice in order to keep $\xi(G)$ under n.

Theorem 10.6 *There exist graphs on n vertices, for infinitely many values of n, which have maximum degree less than $\sqrt{n} + \frac{1}{2}$ and satisfy $\xi(G) < n$.*

Proof: We apply a method very similar to the one which we used in the construction of Galois planes. We consider $n = q^2 + q + 1$ vertices where q is a prime power. We represent the vertices of G with homogeneous coordinate triples $(a, b, c) \neq (0, 0, 0)$ where a, b, c are elements of the Galois field $GF(q)$. For any (a, b, c) and any $\lambda \in GF(q) \setminus \{0\}$, the triples (a, b, c) and $(\lambda \times a, \lambda \times b, \lambda \times c)$ mean the same vertex. Two vertices (a, b, c) and (a', b', c') are adjacent in G if and only if

$$a \times a' + b \times b' + c \times c' = 0$$

holds in $GF(q)$. We prove two properties of G, stated in separate Claims, which are important in connection with the theorem to be proved.

Claim: Every vertex of G has degree q or $q + 1$.

Proof: We know from Chapter 9 on projective planes that the equation

$$a \times x + b \times y + c \times z = 0$$

has precisely $q^2 - 1$ nontrivial solutions in (x, y, z), and they correspond to $(q^2 - 1)/(q - 1) = q + 1$ vertices. If all of them are different from (a, b, c) then the vertex represented by (a, b, c) has degree precisely $q + 1$ in G. Otherwise — i.e., if $a^2 + b^2 + c^2 = 0$ holds in $GF(q)$ — the vertex has degree q. $\Diamond$

Claim: Graph G has diameter two.

Proof: Consider any two distinct vertices (a, b, c) and (a', b', c'). We view them as points in the Galois plane $PG(2, q)$. By axiom A1 of finite projective planes we know that there exists a line $[x, y, z]$ passing through these two points. Then the vertex (a'', b'', c'') defined by $a'' = x, b'' = y, c'' = z$ is adjacent to both vertices (a, b, c) and (a', b', c'). Hence, any two vertices of G are at distance at most two apart. $\Diamond$

Now we are in a position to complete the proof of the theorem. Denote by $v_1, \ldots, v_n$ the vertices of G (which we represented by homogeneous triples). Let v_i, v_j be any two vertices of G If $v_i v_j$ is an edge of G, let the paths $P_{i,j}$ and $P_{j,i}$ be just the edge $v_i v_j$ in both directions; it does not load any vertex. On the other hand, if $v_i v_j$ is not an edge, we can choose a common neighbor v_k of the two vertices and take the paths

$$P_{i,j} = v_i v_k v_j, \qquad P_{j,i} = v_j v_k v_i.$$

These two paths load v_k.

In this way a routing $\mathcal{R}$ has been defined. The load of a vertex is at most the number of ordered pairs of vertices in its neighborhood, thus we have

$$\xi(G) \leq (q + 1)q = n - 1.$$

Moreover, all vertices have degrees at most

$$q+1 < \sqrt{q^2+q+1} + \frac{1}{2} = \sqrt{n} + \frac{1}{2}.$$

This completes the proof of the theorem. □

Example 10.1 *We illustrate the method with the case $q = 2$. The coordinates of the vertices are then taken over $GF(2)$, that means coordinates 0 and 1, with the exclusion of $(0,0,0)$. Two vertices are adjacent if and only if the number of positions where both of them have coordinate 1 is even (i.e., 0 or 2).*[2] *The graph obtained is exhibited in Figure 10.1.*

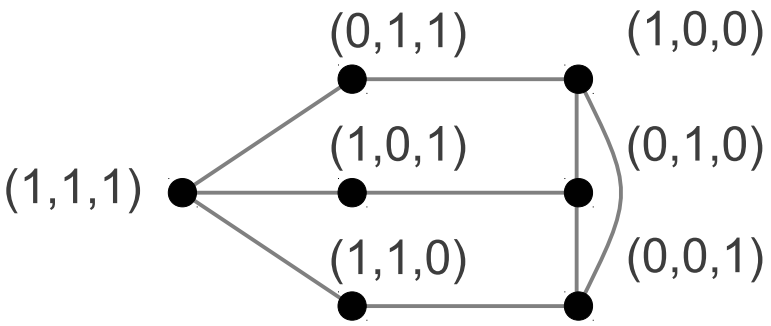


Figure 10.1: *Routing example for the $q = 2$ case*

Remark 10.2 *The routing defined in the proof of Theorem 10.6 for the graph of Figure 10.1 is not optimal: it loads $(1,1,1)$ with 6, its neighbors with 2, and the vertices of the triangle with 4. Hence, if we replace one of the paths of length 2 passing through $(1,1,1)$ with a path of length 3 which then loads two vertices of the triangle, we can reduce the maximum load from 6 to 5. Nevertheless, those routings are nearly optimal for any large n because, by (10.2), the lower bound $\xi(G) \geq n - \sqrt{n} - 3/2$ is valid.*

10.4 The Turán problem for 4-cycles

Interestingly enough, the previous construction gives a strong bound on $\mathrm{ex}\,(n, C_4)$, too. This is remarkable because in routings our goal is to *minimize* the number of edges, whereas in a Turán-type problem one wants to determine the *maximum*.

[2] This is because $0+1=1$ and $1+1=0$ in $GF(2)$.

Theorem 10.7

$$\lim_{n\to\infty} \frac{\mathrm{ex}\,(n, C_4)}{n^{3/2}} = \frac{1}{2}\,.$$

Proof: We have to prove that, for every $\varepsilon > 0$, the quotient $\mathrm{ex}\,(n, C_4)\,/n^{3/2}$ is between $\frac{1}{2} - \varepsilon$ and $\frac{1}{2} + \varepsilon$ if n is sufficiently large.

Upper bound: Let $G = (V, E)$ be any graph on $n = |V|$ vertices and $m = |E|$ edges, not containing C_4 as a subgraph. The upper bound is based on the following simple observation: If two distinct vertices v, w belonged to the common neighborhood of two vertices x, y, then $vxwy$ would be a 4-cycle in G. Since this is excluded, for the number of vertex pairs in the neighborhoods of the vertices we have

$$\sum_{v\in V} \binom{d(v)}{2} \le \binom{n}{2}.$$

In the algebraic manipulations below we apply the well-known facts that $\frac{a_1^2+\cdots+a_n^2}{n} \ge \left(\frac{a_1+\cdots+a_n}{n}\right)^2$ holds for all n-tuples of nonnegative real numbers (the inequality between quadratic and arithmetic means) and that the degree sum in every graph equals the double of the number of edges. Then we obtain

$$\begin{aligned} n(n-1) &\ge \sum_{v\in V} d(v)\cdot(d(v)-1) \\ &= \sum_{v\in V} \big(d(v)\big)^2 - \left(\sum_{v\in V} d(v)\right) \\ &\ge \frac{1}{n}\left(\sum_{v\in V} d(v)\right)^2 - 2m \\ &= \frac{4}{n}\,m^2 - 2m. \end{aligned}$$

This leads to the quadratic inequality

$$m^2 - \frac{n}{2}\,m - \frac{1}{4}\,n^2(n-1) \le 0$$

on m, from which we get the upper bound

$$m \le \frac{1}{2}\sqrt{n^3 - \frac{3}{4}\,n^2} + \frac{1}{4}\,n < \frac{1}{2}\,n^{3/2} + \frac{1}{4}\,n.$$

Lower bound: First we assume that n is of the form $n = q^2 + q + 1$, where q is a prime power. In this case we can take the graph G constructed in the

proof of Theorem 10.6. Every vertex of G has degree at least q, which is larger than $\sqrt{n}-1$. Thus, the number of edges in G is at least $\frac{1}{2}(n^{3/2}-n)$.

Claim: Graph G contains no 4-cycles.

Proof: Suppose for a contradiction that $v_1v_2v_3v_4$ is a 4-cycle in G. Recall that the vertices are represented with homogeneous coordinate triples. In this way we may associate them with points and lines of the Galois plane $PG(2,q)$. Let us assume without loss of generality that v_1 and v_3 are points, while v_2 and v_4 are lines; say,

$$v_1 = (a,b,c) = P_1, \qquad v_3 = (a',b',c') = P_2,$$

$$v_2 = [x,y,z] = L_1, \qquad v_4 = [x',y',z'] = L_2.$$

Due to the assumed adjacencies $v_1v_2, v_2v_3, v_3v_4, v_4v_1$ we can see that the scalar products of all the four pairs of triples belonging to (P_i, L_j) with $i,j \in \{1,2\}$ are equal to zero in $GF(q)$. This implies, however, that the two points P_1, P_2 are contained together in both lines L_1, L_2. This contradicts the axioms of finite projective planes. $\Diamond$

In this way the lower bound on $\mathrm{ex}\,(n, C_4)$ follows if n is of the form q^2+q+1 where q is a prime power. To handle the intermediate values of n we recall a deep result from number theory. It is known that, for some appropriately chosen constant c, every interval $(x, x+cx^{0.525})$ contains a prime. If n is large, we can simplify formulation by putting a somewhat larger exponent and then we can disregard c. So, if q is a sufficiently large prime (or prime power), then the interval $(q, q+q^{3/5})$ contains another prime, say q'. Performing the analogous graph construction for q', we have some number $n' > q^2+q+1$ of vertices, while the number of edges is less than

$$\frac{1}{2}\,(q'+1)(q'^2+q'+1) < \frac{1}{2}\,(q+q^{3/5}+1)((q+q^{3/5})^2+q+q^{3/5}+1)$$

$$< \frac{1}{2}\,q^3 + C\cdot q^{13/5}$$

for some constant C if q is large. Thus, if we add $n'-q^2-q-1$ isolated vertices to G, we still have sufficiently many edges for this larger n'. This completes the proof. $\square$

Further reading

There are many textbooks on graph theory, and also the internet offers a practically infinite source of related material. Here we mention only a couple of books which are entirely free to download.

- J. A. Bondy and U. S. R. Murty, Graph Theory with Applications, North-Holland (New York, Amsterdam, Oxford), 1976.

After some decades the authors made this book freely available and nowadays it can be accessed at several sites, for example

`http://cs.bme.hu/fcs/GTWA.pdf`

(Accessibility of the original French site seems to be restricted at the time of writing these notes.)

A more detailed textbook is

- R. Diestel, Graph Theory, Graduate Texts in Mathematics, Vol. 173, Springer-Verlag (Heidelberg), 2005, 4th edition 2010.

  `http://diestel-graph-theory.com/index.html`

The theory of directed graphs is thoroughly treated in

- J. Bang-Jensen and G. Gutin, Digraphs: Theory, Algorithms and Applications, Springer-Verlag (London), 2008.

  `http://www.cs.rhul.ac.uk/books/dbook/`

Finally we mention the following textbook on combinatorial optimization, which also contains some graph problems discussed in the present work:

- A. Schrijver, A Course in Combinatorial Optimization, 2013.

  `http://homepages.cwi.nl/ lex/files/dict.pdf`

Index

Appendix A

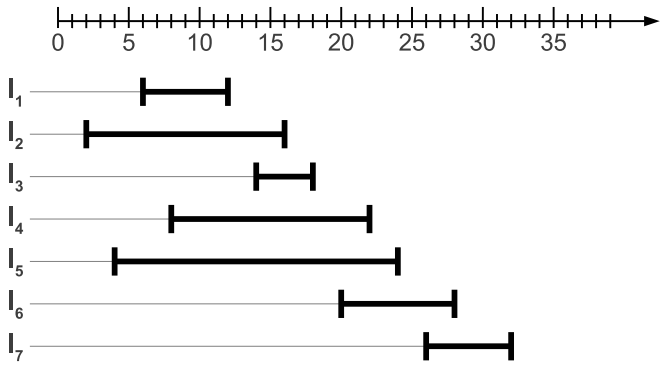
Illustration of algorithms

A.1 Transversal and matching in interval systems

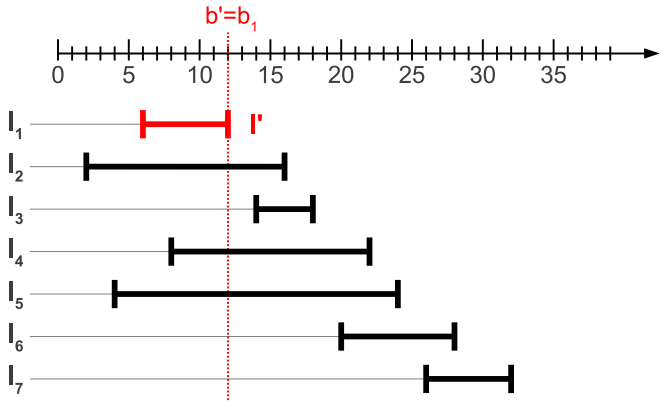
Algorithm

$T := \emptyset$, $\mathcal{M} := \emptyset$, $\mathcal{I} := \{I_1, I_2, \ldots, I_n\}$
while $\mathcal{I} \neq \emptyset$ **do**
  $b' := \min_{I_i \in \mathcal{I}} b_i$,
  $T := T \cup \{b'\}$
  Select I' form $\{I_i \in \mathcal{I} \mid b_i = b'\}$ arbitrary
  $\mathcal{M} := \mathcal{M} \cup \{I'\}$
  $\mathcal{I} := \{I \in \mathcal{I} \mid b' \notin I\}$
end while
$\tau = |T|, \nu = |\mathcal{M}|$, T is a smallest transversal set, and $\mathcal{M}$ is a largest matching.

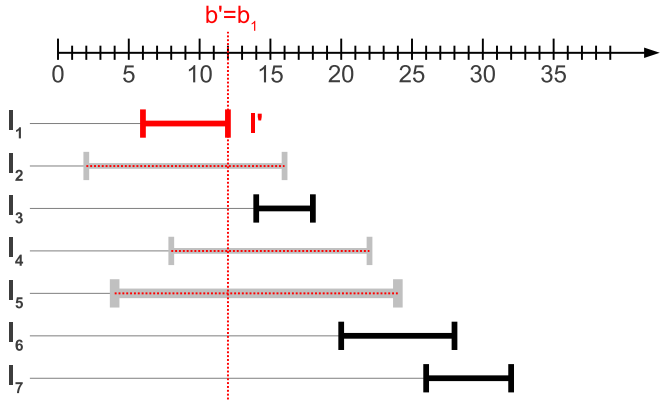
Example



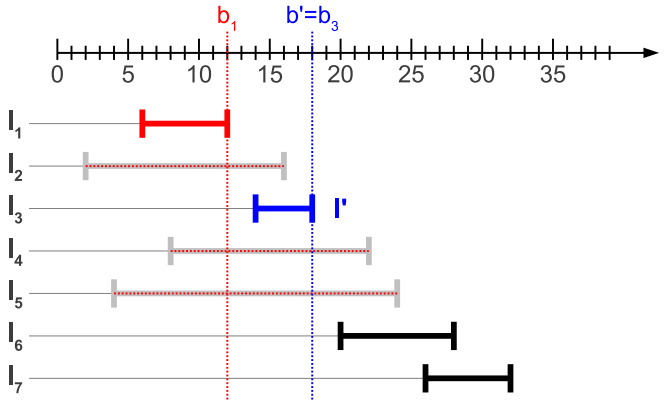
In the example there are 7 intervals in $\mathcal{I}$, as shown above.



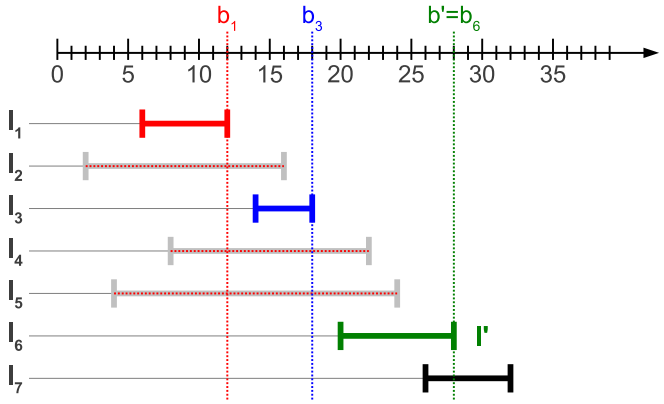
In the first iteration the leftmost right endpoint (b') is selected, which is the right endpoint of the interval I_1. Thus, $b' = b_1$ and $I' = I_1$, which are added to the sets T and $\mathcal{M}$, respectively: $T = \{b_1\}$ and $\mathcal{M} = \{I_1\}$.



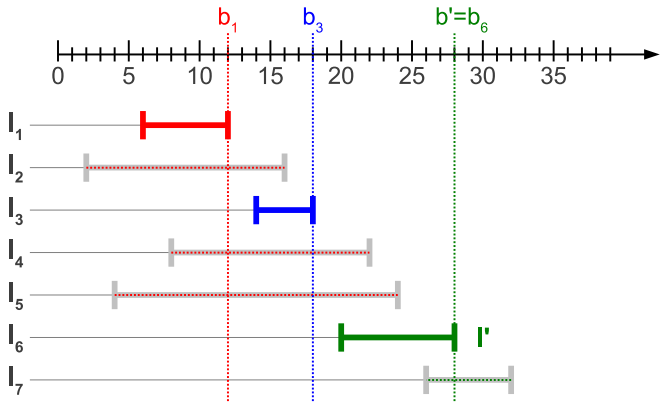
In the last step of the iteration, all of the intervals that contain $b' = b_1$ are removed from $\mathcal{I}$. The only intervals remaining in $\mathcal{I}$ are I_3,I_6, and I_7. As $\mathcal{I}$ is not empty, a new iteration starts.



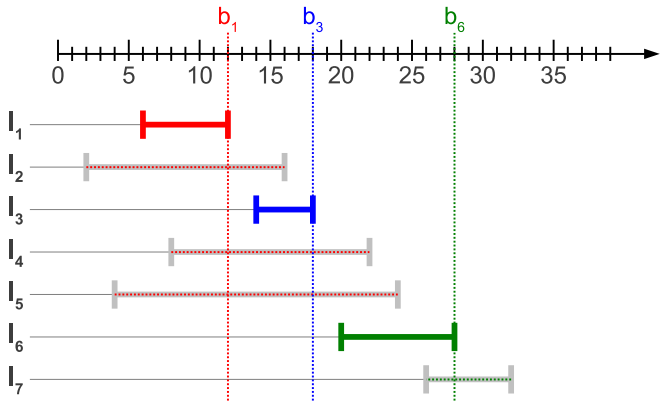
In the second iteration, the leftmost right endpoint is b_3, thus T becomes $\{b_1, b_3\}$ and $\mathcal{M}$ becomes $\{I_1, I_3\}$. b_3 is contained only by I_3 among the intervals still in $\mathcal{I}$, thus, after the last step of the iteration $\mathcal{I} = \{I_6, I_7\}$.



In the third iteration the leftmost right endpoint is b_6, thus, $T = \{b_1, b_3, b_6\}$ and $\mathcal{M} = \{I_1, I_3, I_6\}$.



As b_6 is contained in both I_6 and I_7, thus, after the last step of the iteration $\mathcal{I} = \emptyset$, and the algorithm quits the loop.



Finally, $T = \{b_1, b_3, b_6\}$ is a smallest transversal set, and $\mathcal{M} = \{I_1, I_3, I_6\}$ is a largest matching. Note, that the set of intervals removed at the ends of each iteration form an intersecting subsystem. Thus, a decomposition of $\mathcal{I}$ into minimum number of intersecting subsystems is: $\{\{I_1, I_2, I_4, I_5\}, \{I_3\}, \{I_6, I_7\}\}$.

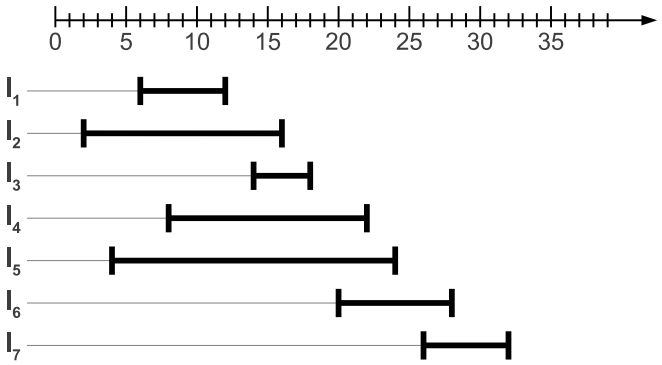
A.2 Decomposition of interval systems into matchings

Algorithm

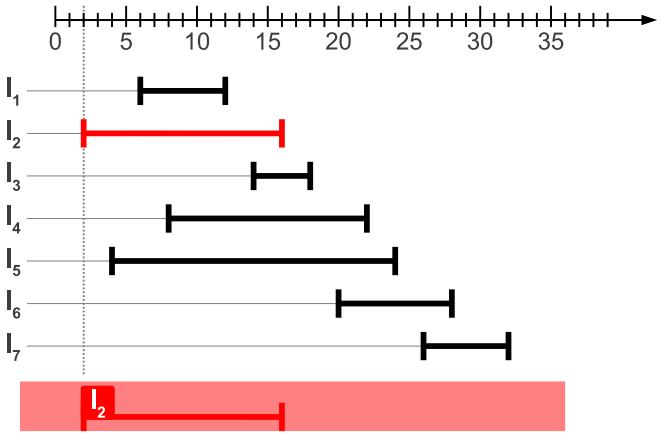
$\varphi := \emptyset$, $\mathcal{I} := \{I_1, I_2, \ldots, I_n\}$
while $\mathrm{Dom}(\varphi) \neq \mathcal{I}$ **do**
  $a' := \min_{I_i \in \mathcal{I} \setminus \mathrm{Dom}(\varphi)} a_i$,

Select I' from $\{I_i \in \mathcal{I} \setminus \mathrm{Dom}(\varphi) \mid a_i = a'\}$ arbitrary
$k := \min_{\substack{k \in \mathbb{Z}^+ \\ \nexists I \in \mathrm{Dom}(\varphi),\, a' \in I,\, \varphi(I) = k}} k$
$\varphi := \varphi \cup \{(I', k)\}$
end while
φ is a coloring of $\mathcal{I}$ with the smallest number of colors

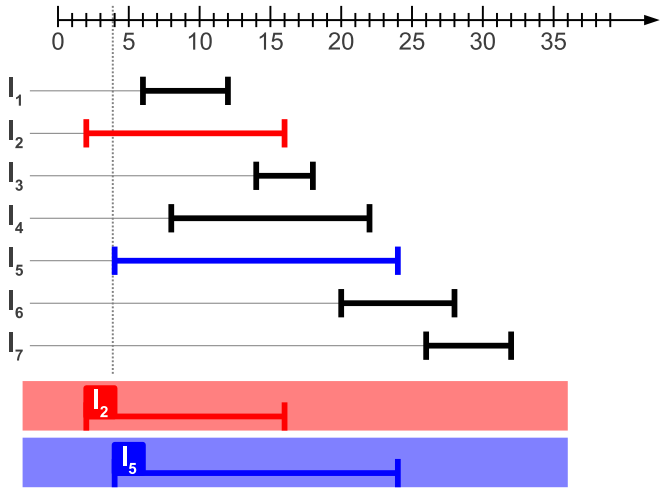
Example



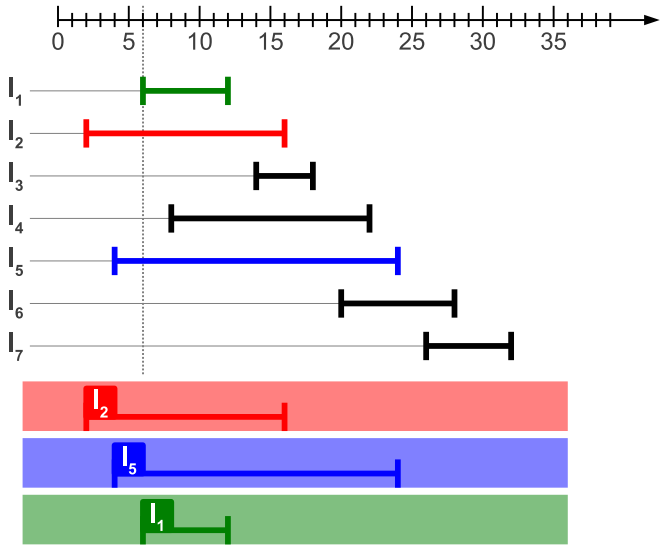
In this illustrative example the interval system has 7 intervals as shown on the figure above.



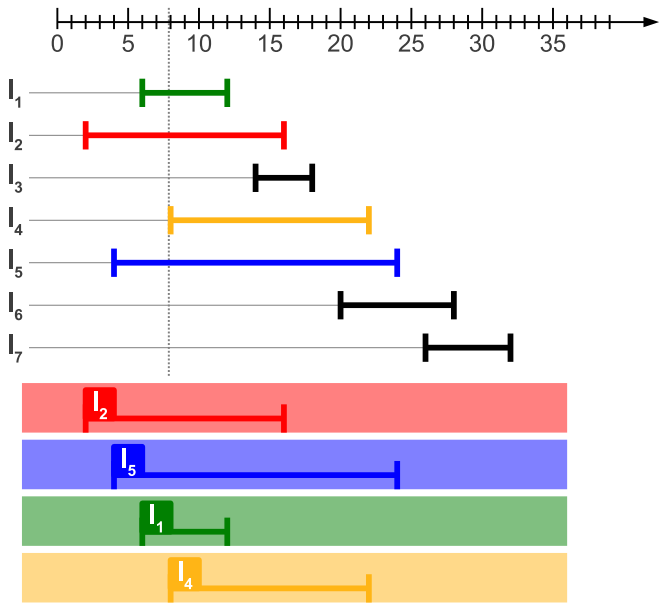
The iteration starts with the increasing order of the left endpoints. Interval I_2 has the smallest endpoint, thus I_2 is added as the first element of the first matching / color class, denoted here by red.



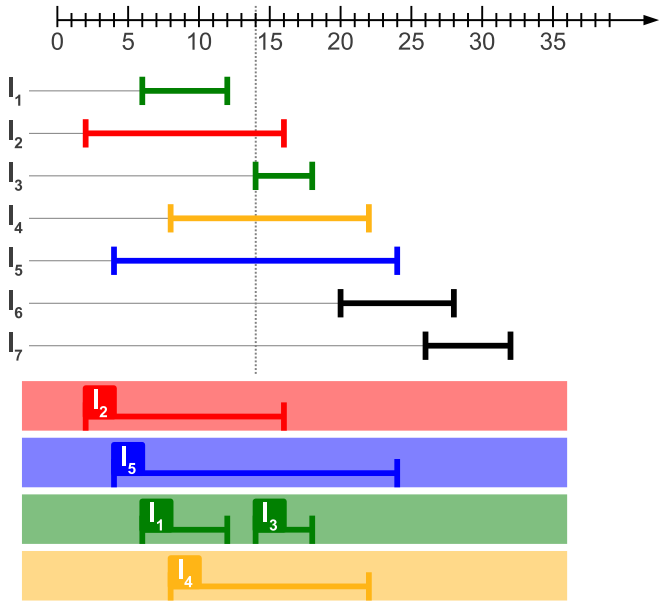
The second smallest left endpoint belongs to I_5. As I_5 has intersection with I_2, it can not be added to the first (red) group, thus a new matching / color is initialized for it. (Denoted by blue.)



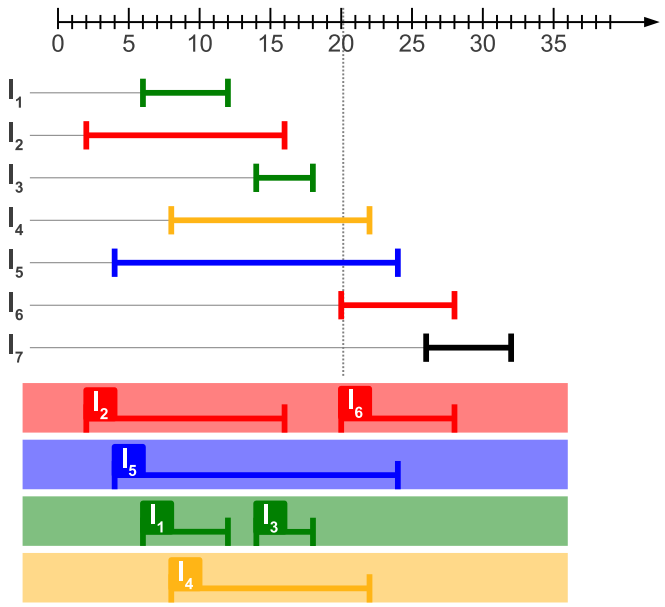
The next smallest endpoint belongs to I_1. It has intersection with both of the previous intervals, thus, a new group is opened for it (green).



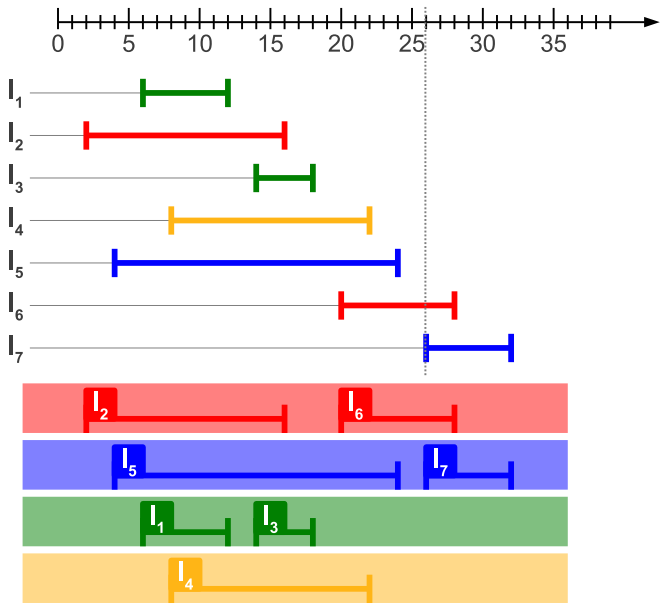
I_4 is selected next and put to a new group, as it has intersection with I_2, I_5, and I_1 as well.



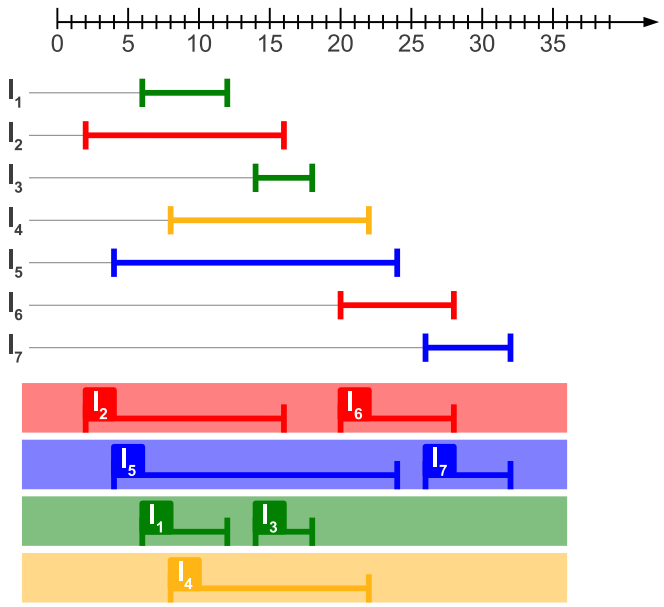
The next smallest left endpoint belongs to I_3. It has intersection with I_2 and I_5, thus it can not belong to the first two matchings. However, it has no intersection with any intervals in the third one, thus I_3 is assigned the green color and put to the third matching.



I_6 is selected next, that has no intersection with I_2, thus it can be put immediately to the first matching.



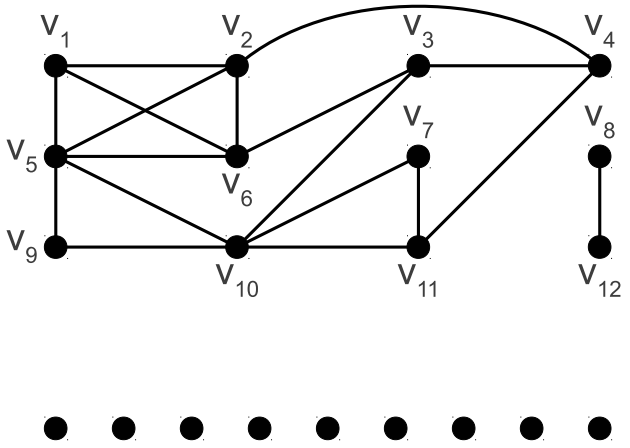
Last, I_7 is selected, that has intersection with I_6, but no intersection with I_5, so it is put to the second matching.



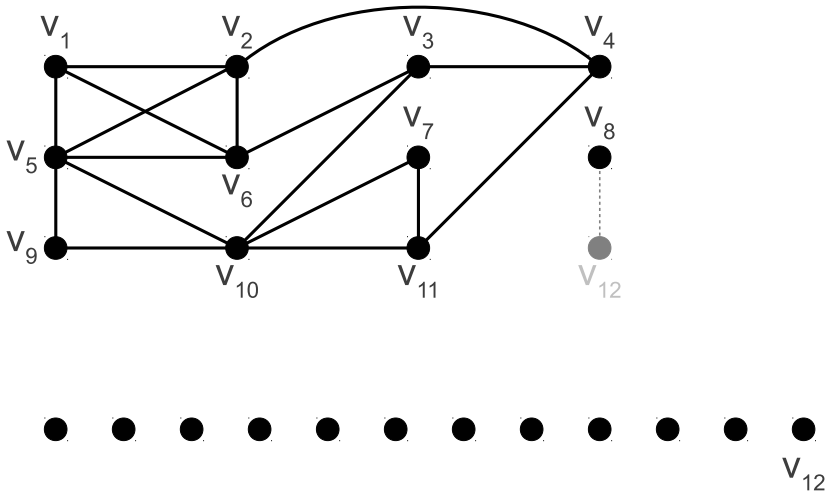
The iteration is over, each interval is put into a matching / assigned a color. The interval system is decomposed into 4 matchings / colored with 4 colors. It is easy to see, that these 4 colors are necessary, as there are 4 intervals which are pairwise intersecting: I_1, I_2, I_4, and I_5. (All of them contain, e.g., 10.)

A.3 Optimal vertex order for coloring number

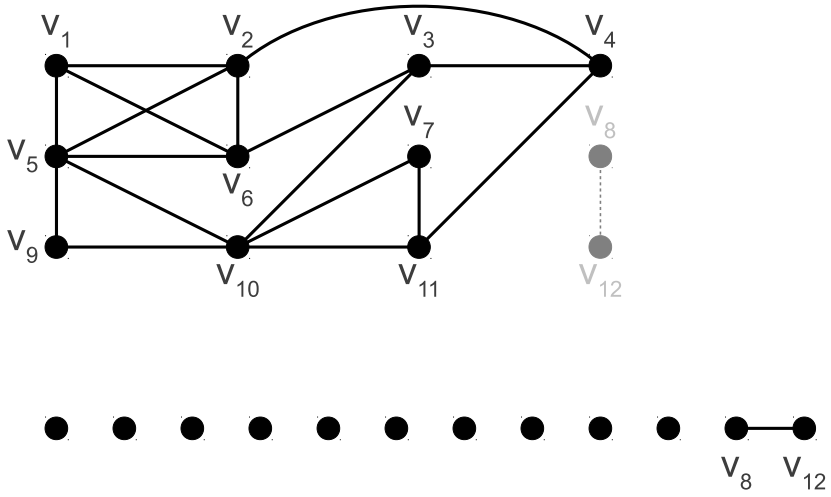
Example



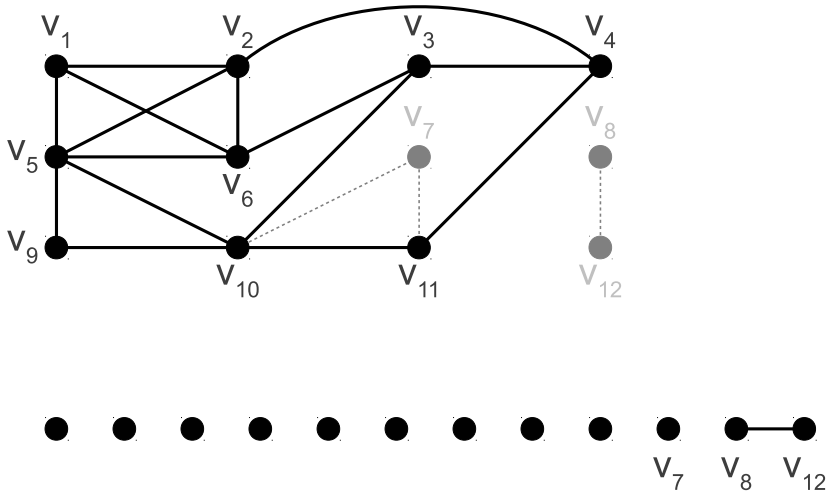
As an example we take the graph from Figure 1. In this graph both v_8 and v_{12} have the minimum degree of 1. We select now v_{12} randomly to be at the end of the optimal vertex order.



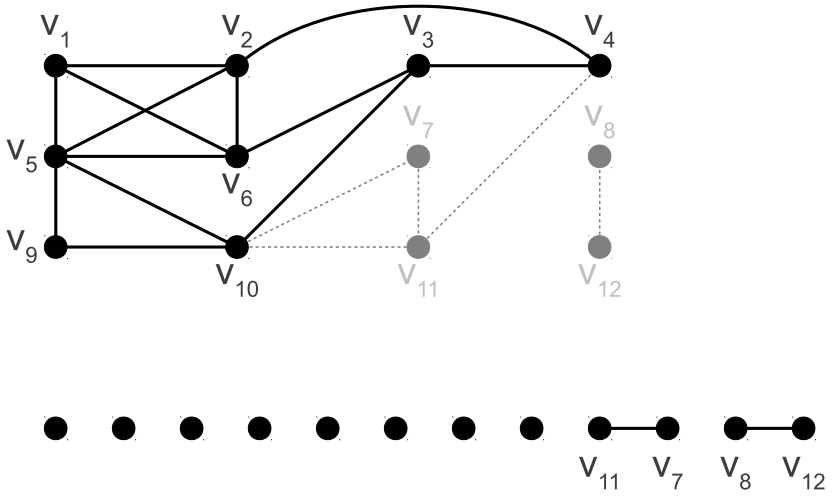
After removing v_{12} from the graph, v_8 has the degree of 0, thus it must be selected next.



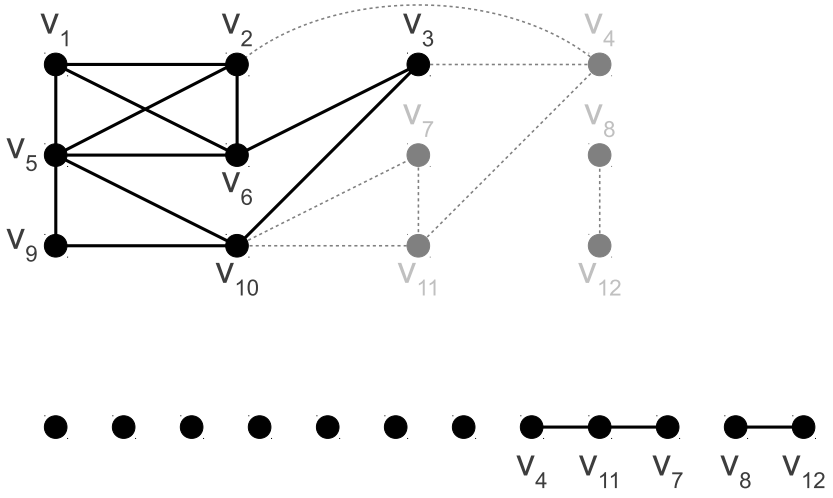
After both v_8 and v_{12} are removed, the edge v_8v_{12} can be also added to the partial graph at the bottom. Now, v_7 and v_9 have the minimum degree of 2, we select v_7.



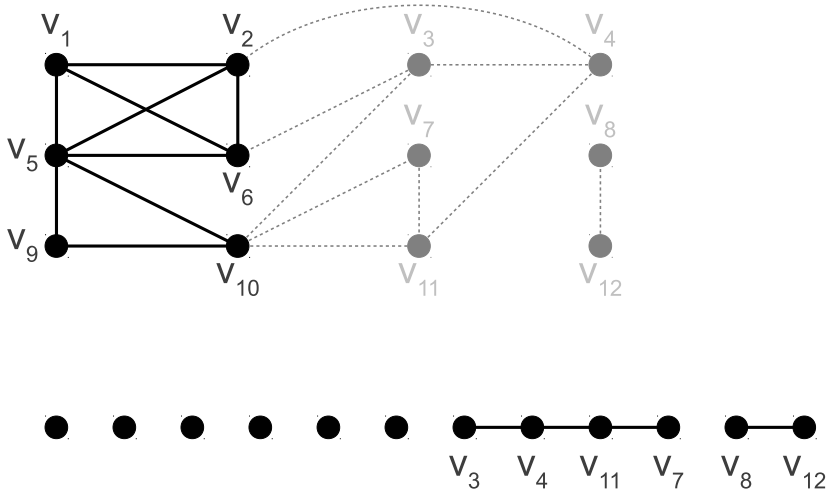
After removing v_7 the degree of v_{11} also decreases to 2, and it is selected as the next vertex.



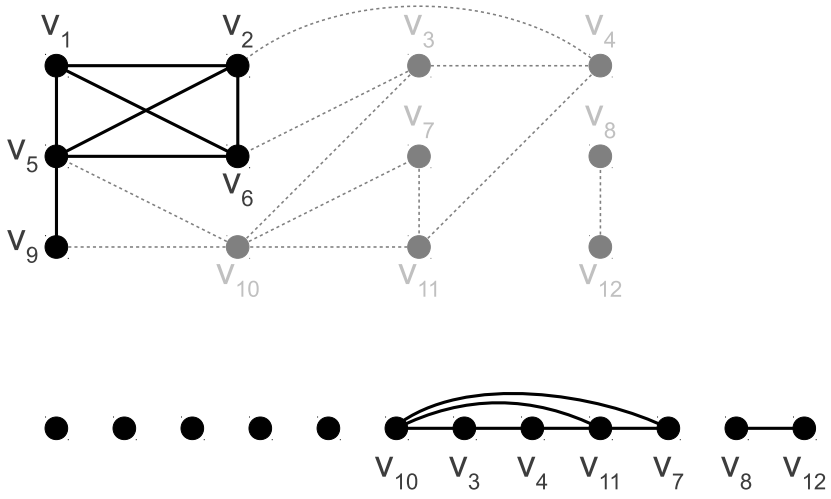
After removing v_{11} the degree of v_4 is also 2, it is selected as the next vertex.



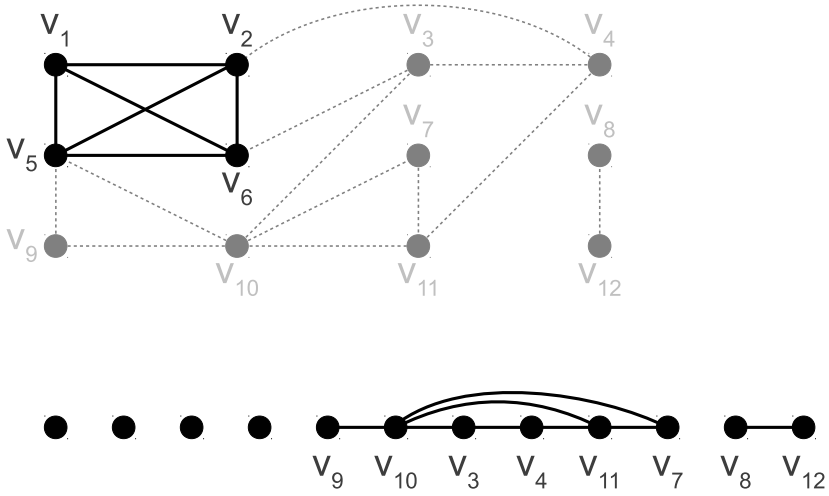
Again, after removing v_4 the degree of v_3 decreases to 2, and it is selected as the next vertex. Note that instead of any of the last three vertices, v_9 could have been selected. The selection between vertices with minimum degree is arbitrary.



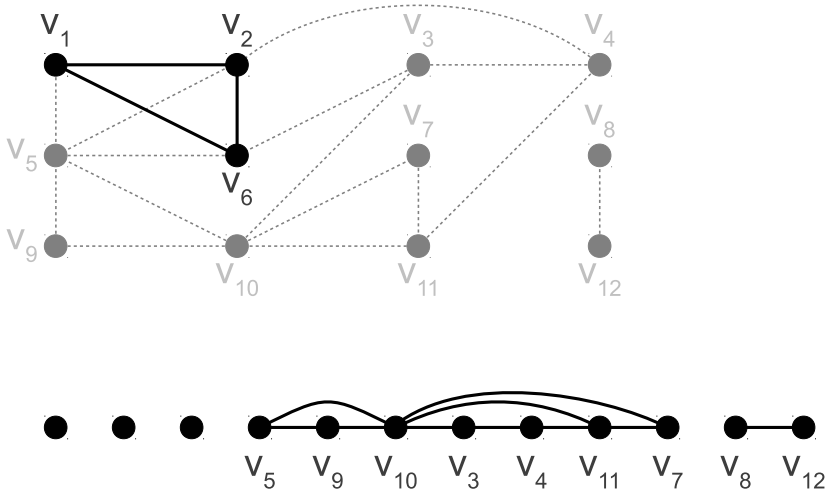
Now, the degree of v_{10} is reduced to 2, and it is selected as the next vertex.



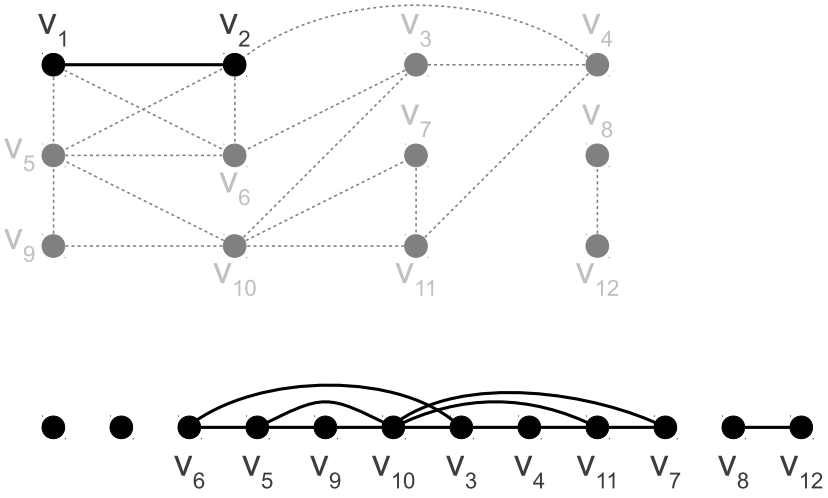
After removing v_{10}, the degree of v_9 is reduced to 1, and each of the remaining vertices has degree either 3 or 4, thus v_9 must be selected for the next vertex in the order.



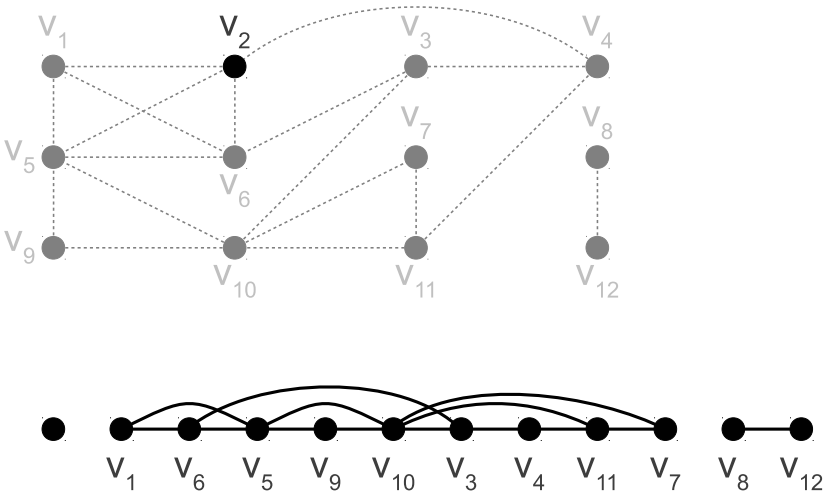
The remaining graph is a K_4, thus all of its vertices have the same degree, and all of the 4! orders of them would be acceptable for the beginning of the optimal vertex order. We first remove v_5,



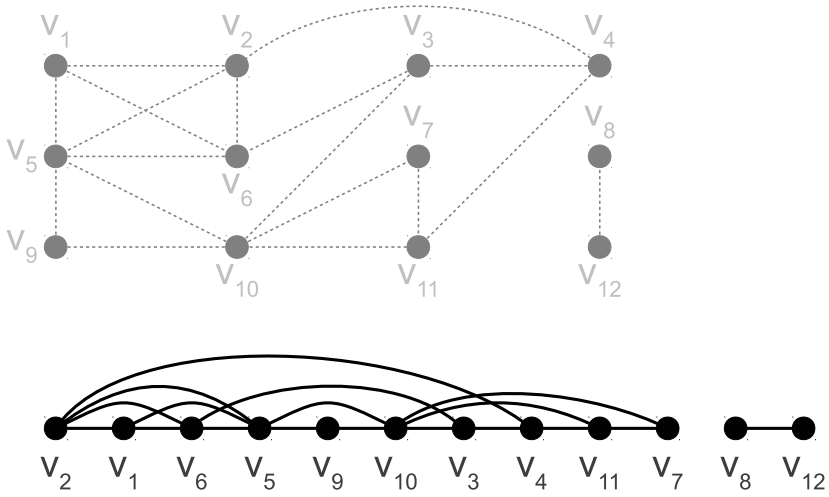
and then, v_6 is the second one.



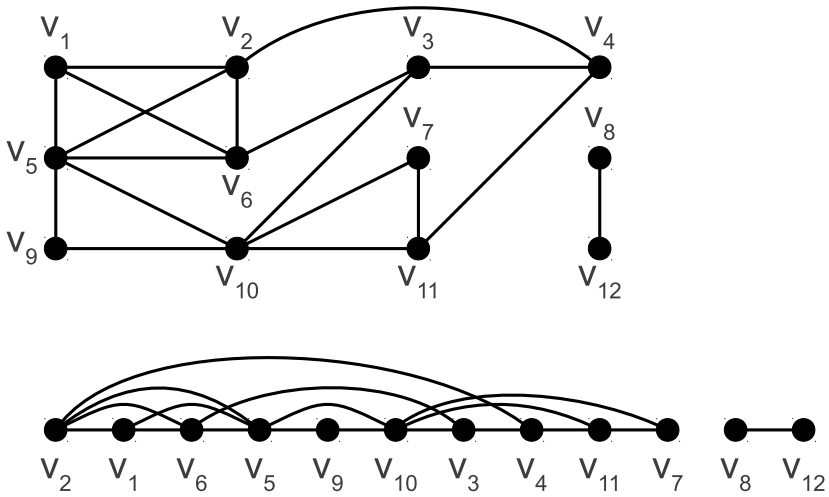
Thirdly, v_1 is removed,



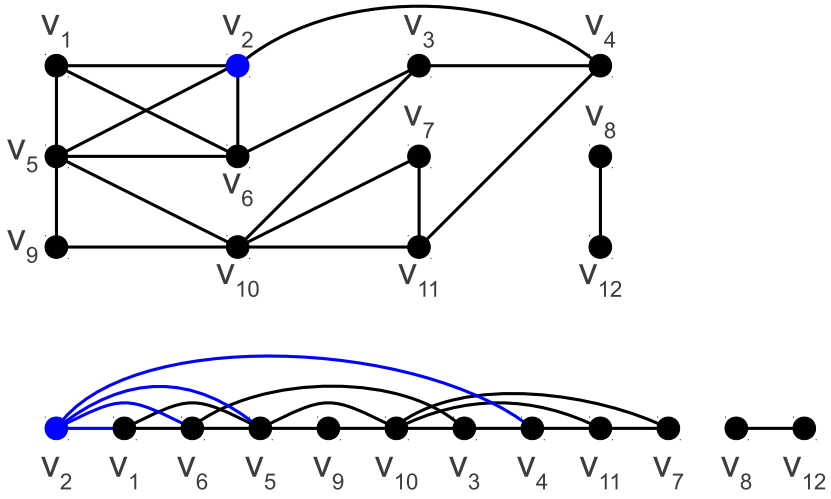
and the last vertex is v_2.



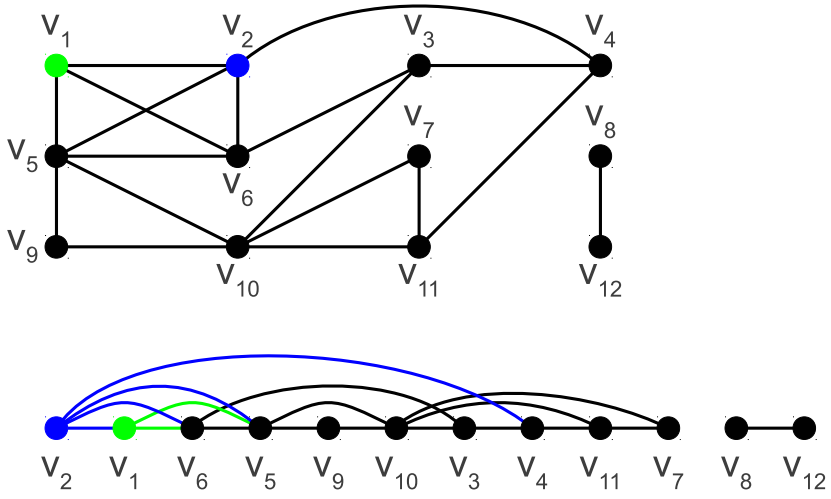
In the optimal ordering of the vertices the largest d^- value is 3, it is attained only by v_5. Therefore, the coloring number equals 4. Moreover, it is sure, that the coloring with the First Fit algorithm will require at most 4 colors. (Since the graph has a K_4 subgraph, the 4 colors are definitely needed, i.e., the coloring obtained by the First Fit approach will be optimal.)



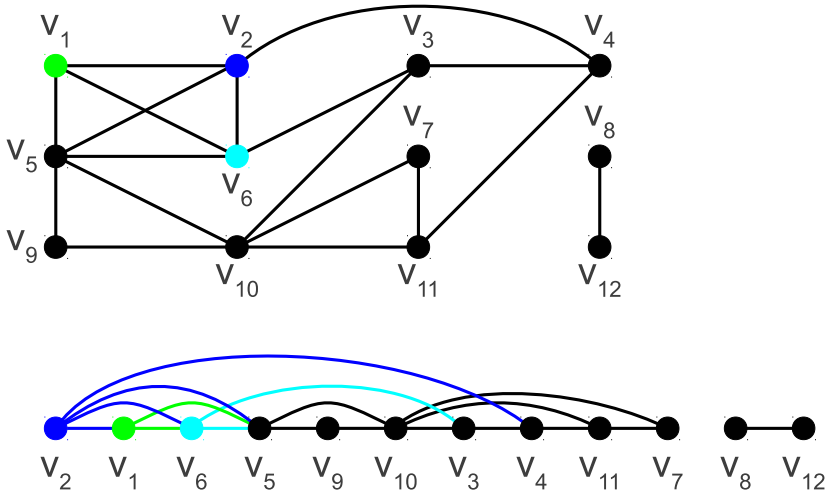
The coloring of the vertices proceeds from left to right, thus first v_2 is colored with the smallest possible color, that is blue in our case. In this example, the order of the colors will be blue $<$ green $<$ cyan $<$ red.



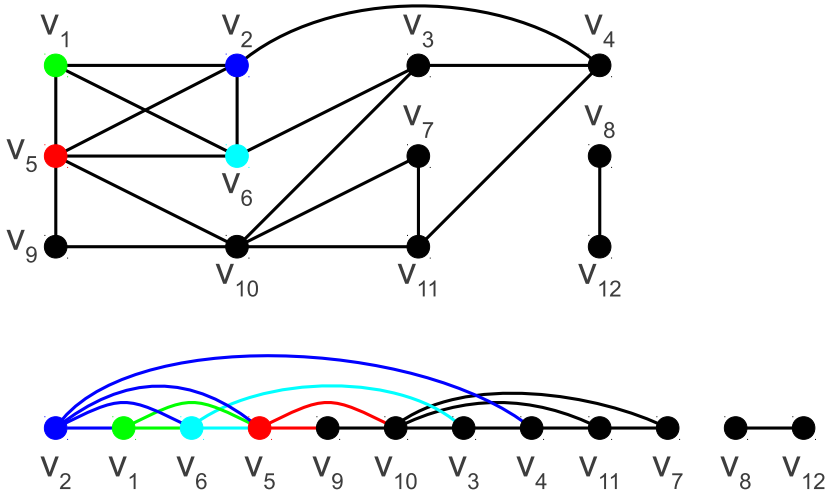
In order to make the evaluation easier, in the bottom graph the edges have the same color as their left endpoint, thus the color of the current vertex can be selected easier, as the prohibited colors appear on the edges ending at that vertex. Vertex v_1 is the next one to be colored, and it has a blue edge (neighbor), thus the green color must be used for it.



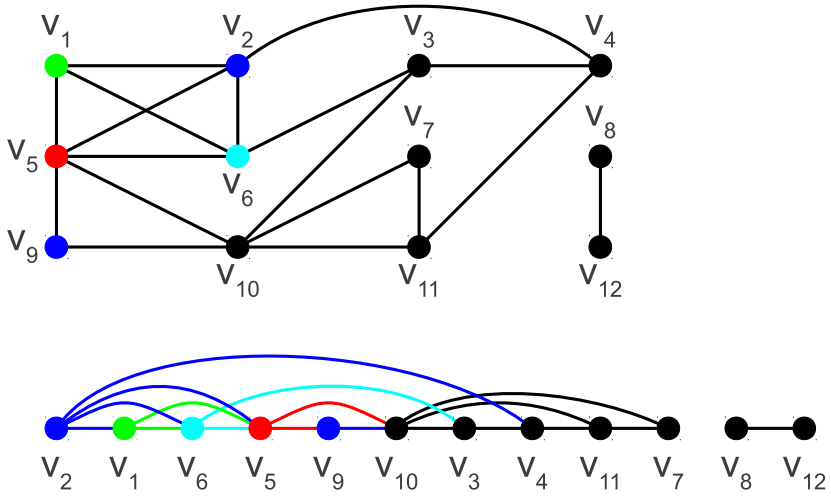
Vertex v_6 has both blue and green neighbors, thus it will need the color cyan.



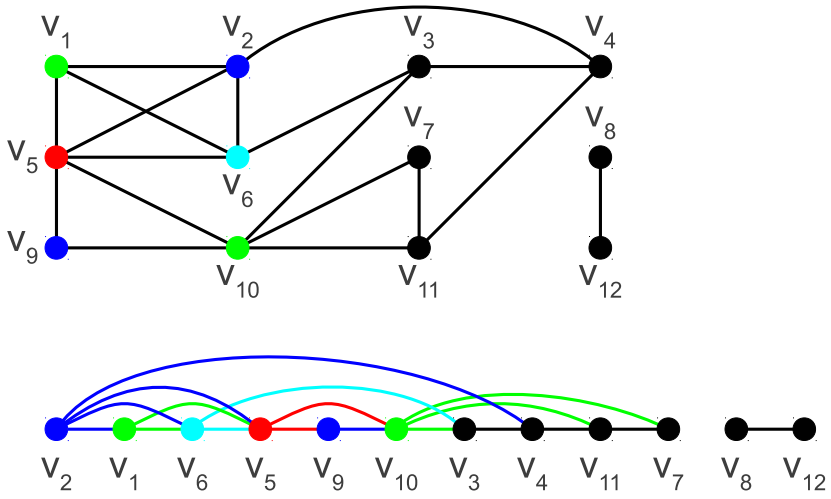
Then, v_5 has blue, green and cyan neighbors, thus it must be colored with the next color, which is red.



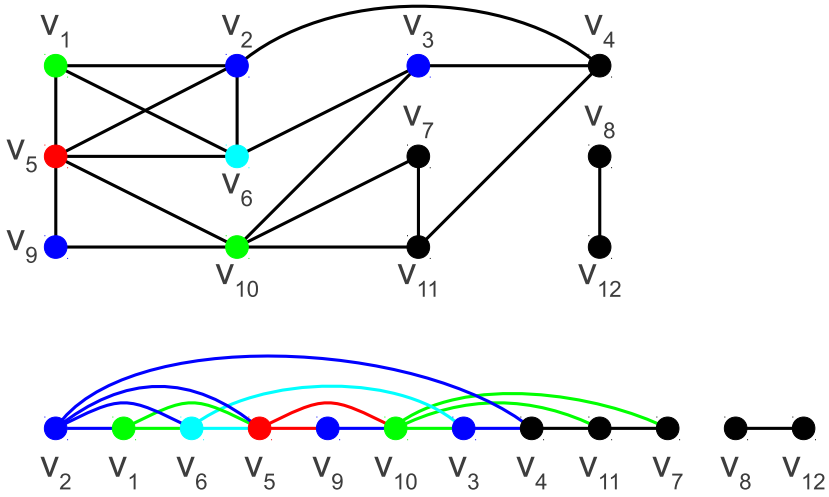
Vertex v_9 has only a red neighbor, thus it can be colored with the smallest color, blue.



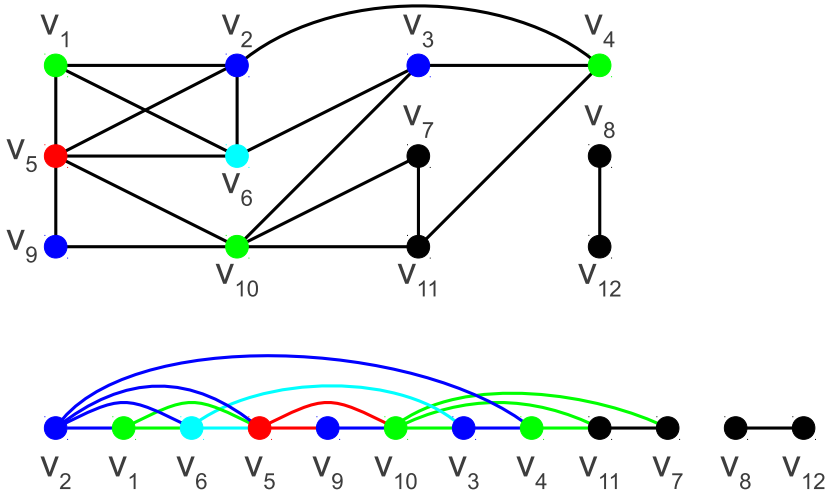
Vertex v_{10} has a blue neighbor, but does not have a green one, so the color green can be used. Note, that having a red neighbor does not influence the selected color until all the smaller colors are used on one of the neighbors.



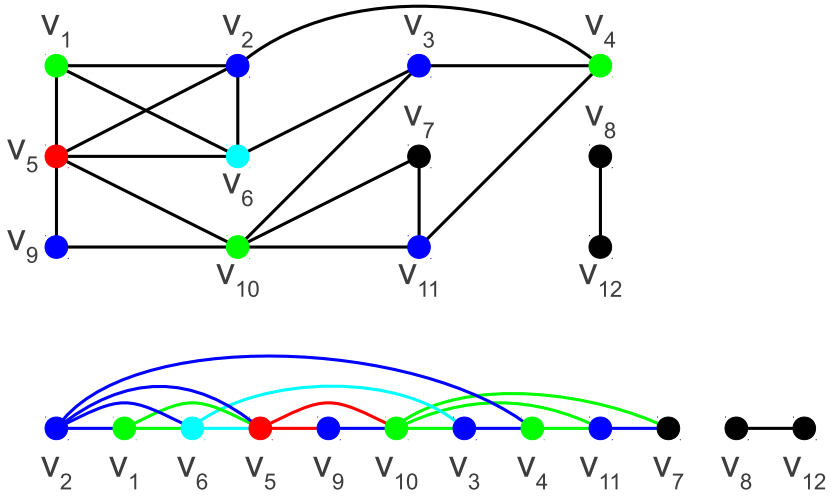
The next vertex v_3 does not have a blue neighbor, thus it is colored with blue.



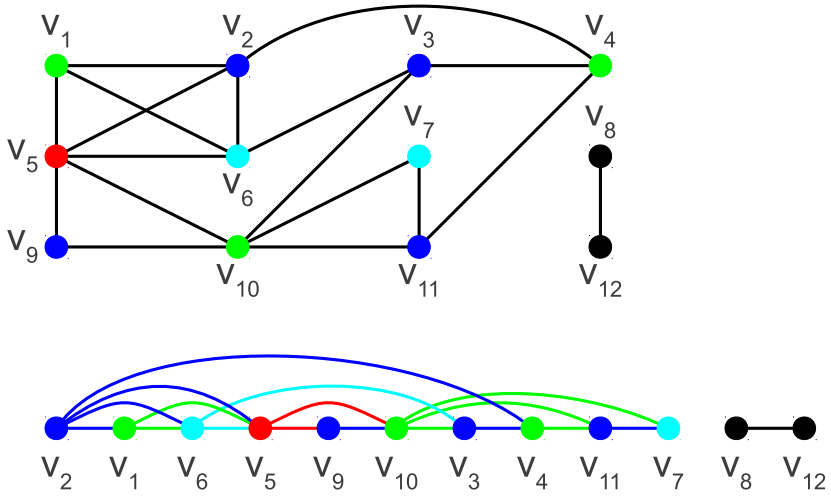
Now, v_4 has two colored neighbors, and both of them are blue, hence v_4 can be colored with green.



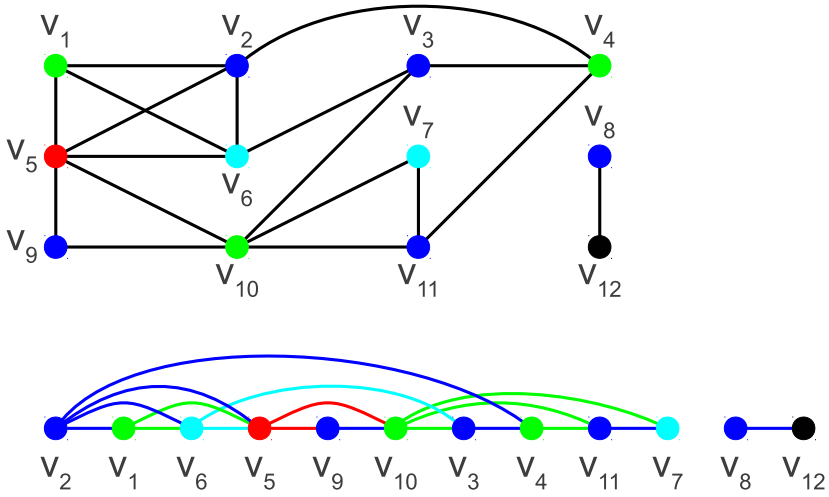
Both of the colored neighbors of v_{11} have green color, none of them is blue, thus v_{11} can be colored with it.



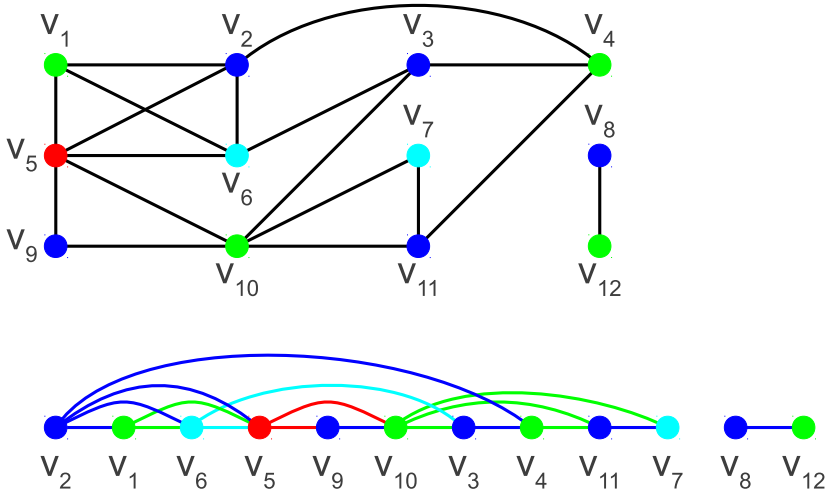
Vertex v_7 has blue and green neighbors, thus the color cyan must be used again.



Vertex v_8 can be colored with blue, since it has no colored neighbors.



Finally, v_{12} has only one blue neighbor, thus it can be colored with green.



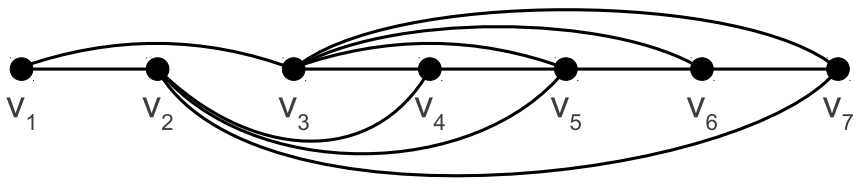
Using the First Fit algorithm and a vertex order optimal for the coloring number, the graph has been properly colored with 4 colors.

A.4 Large cut by local improvements

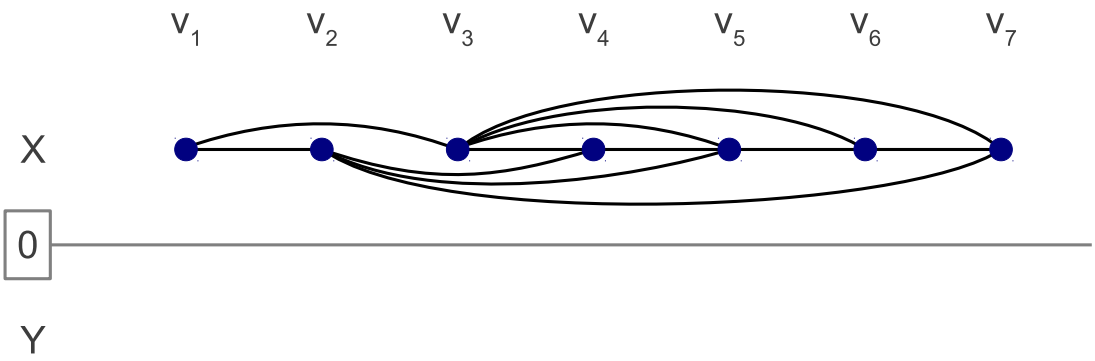
Algorithm

$G := (V, E)$, $X = V$, $Y = \emptyset$
$A := \{v \in X \mid |N(v) \cap X| > |N(v) \cap Y|\} \cup \{v \in Y \mid |N(v) \cap X| < |N(v) \cap Y|\}$
while $A \neq \emptyset$ **do**
  Select v from A arbitrary
  if $v \in X$ **then**
    $X := X \setminus \{v\}$, $Y := Y \cup \{v\}$
  else
    $X := X \cup \{v\}$, $Y := Y \setminus \{v\}$
  end if
  $A := \{v \in X \mid |N(v) \cap X| > |N(v) \cap Y|\} \cup \{v \in Y \mid |N(v) \cap X| < |N(v) \cap Y|\}$
end while
Partition of vertices into sets X and Y which cannot be locally improved.

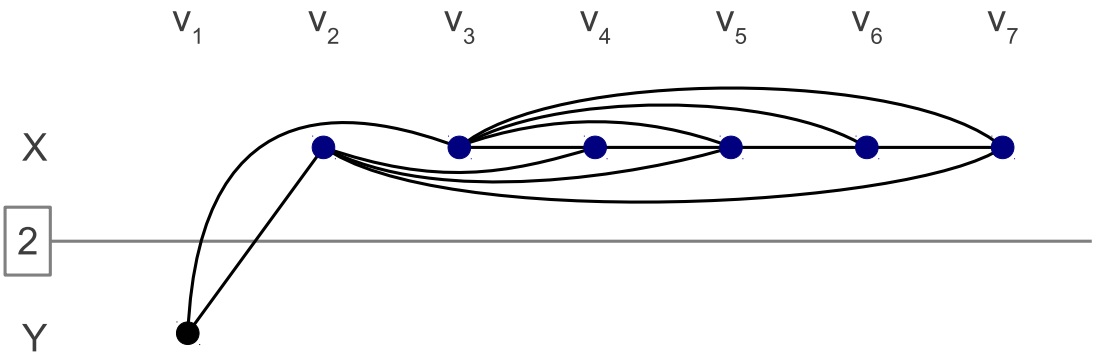
Example



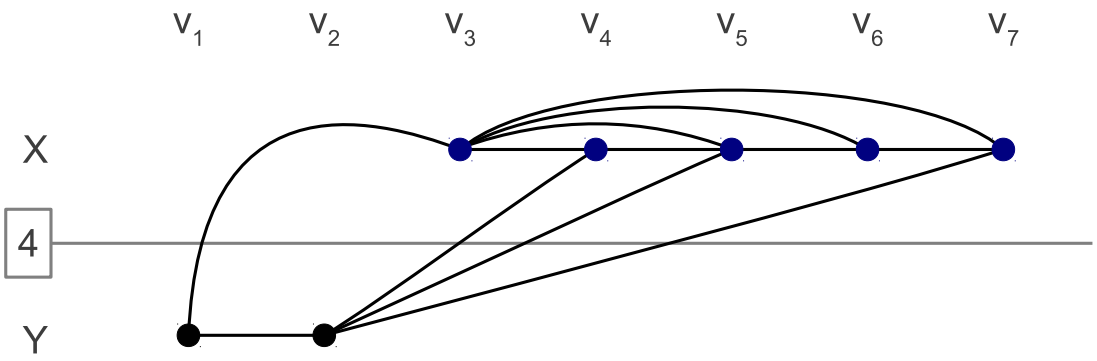
In this illustrative example the graph has 7 vertices and 12 edges



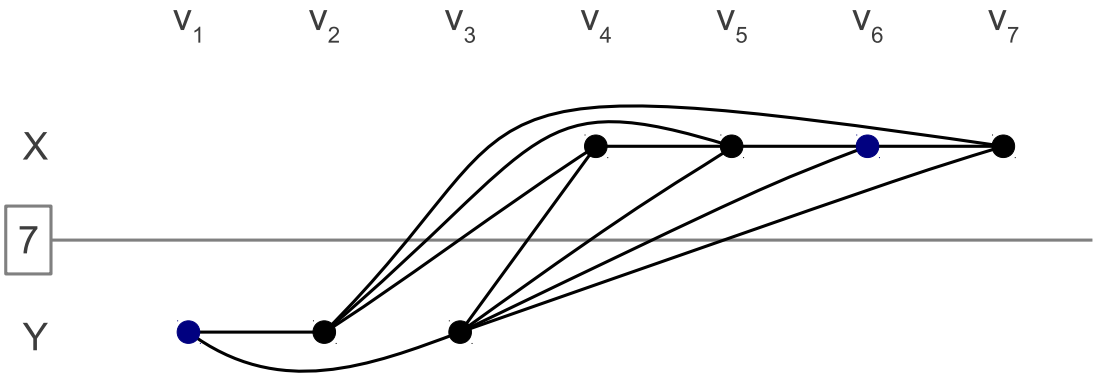
Initially all the vertices are in the partition class X, and all of them have more neighbors in the same class than in the other, thus all of the vertices are in the set A, denoted by blue in this illustration. The size of the cut is 0 at the moment. First move v_1 to Y to increase the size of the cut.



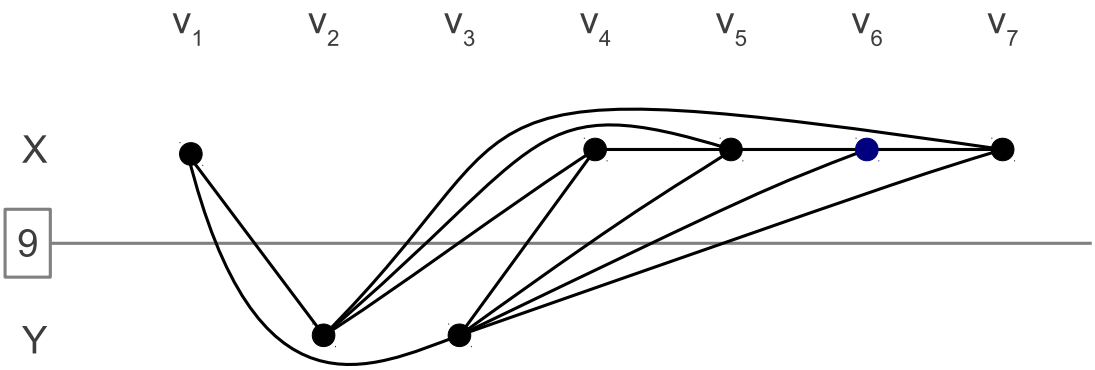
The size of the cut increased to two. All the vertices except v_1 still has more neighbors in the same partition class. Continue the algorithm by moving v_2 to Y, as it has 3 neighbors in X and only one in Y.



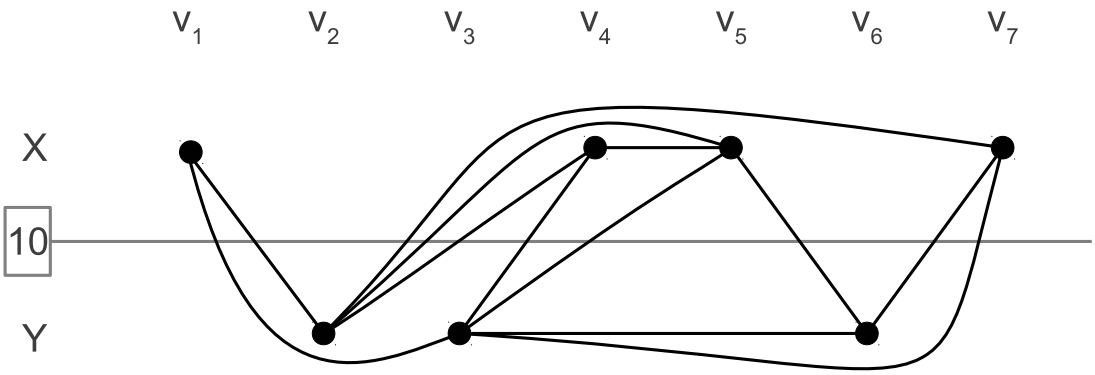
The size of the cut increased to 4, and v_2 is also removed from the set A. Now, we select v_3 as the next vertex to move to Y, as it has 4 neighbors in X and only one in Y.



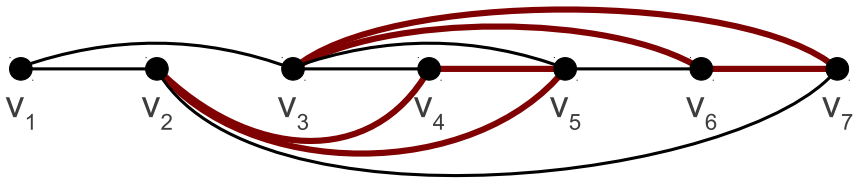
Obviously, v_3 is not in A anymore, as it is the vertex that has been moved last. Moreover, vertices v_4, v_5 and v_7 are also removed from A as they have either equally many or fewer neighbors in X than in Y. Vertex v_6 has two neighbors in X, and only one in Y, thus it still remains in A. Additionally, v_1 is in A again, as it has now 2 neighbors in Y and none in X. Select v_1 again, and move it back to X to increase the size of the cut, which was 7 at this step.



The size of the cut is increased to 9, and the only vertex contained in A is v_6, thus it is moved to Y.



Now, the size of the cut is 10 and A is empty, as each vertex has more neighbors in the other partition class than in its own one. Therefore, the partition with $X = \{v_1, v_4, v_5, v_7\}$ and $Y = \{v_2, v_3, v_6\}$ is locally optimal.



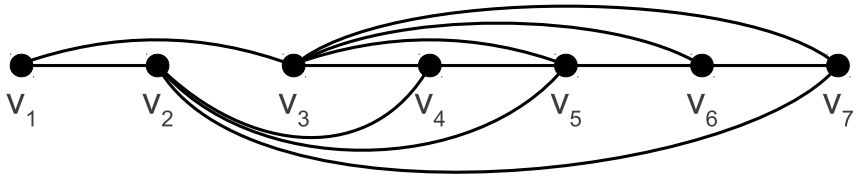
Note that in this case the final cut is also globally optimal, as the graph has two edge-disjoint cycles of length 3 and consequently, the size of the cut cannot exceed $12 - 2 = 10$.

A.5 Large cut by the online approach

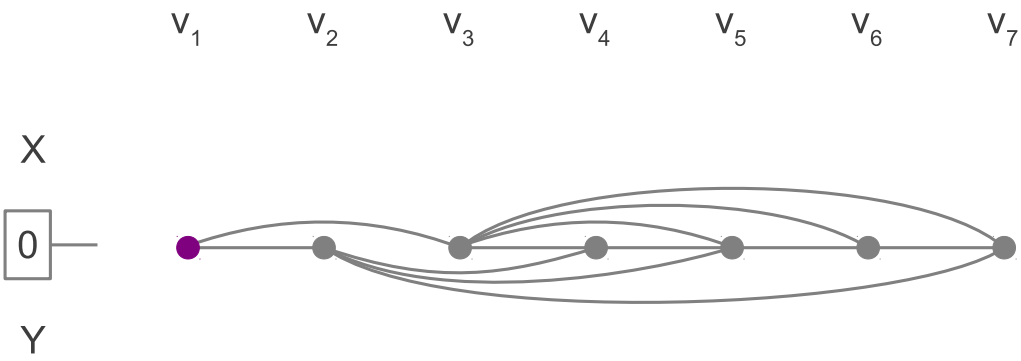
Algorithm

$G := (V, E)$, $X = Y = \emptyset$
while $X \cup Y \neq V$ **do**
  Select v from $V \setminus X \setminus Y$ arbitrary
  if $|N(v) \cap X| > |N(v) \cap Y|$ **then**
    $Y := Y \cup \{v\}$
  else
    $X := X \cup \{v\}$
  end if
end while
Vertices of V are partitioned into X and Y with the online approach

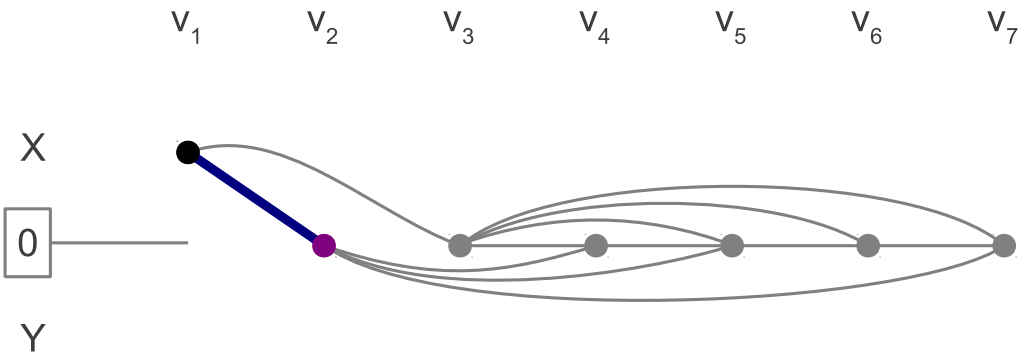
Example



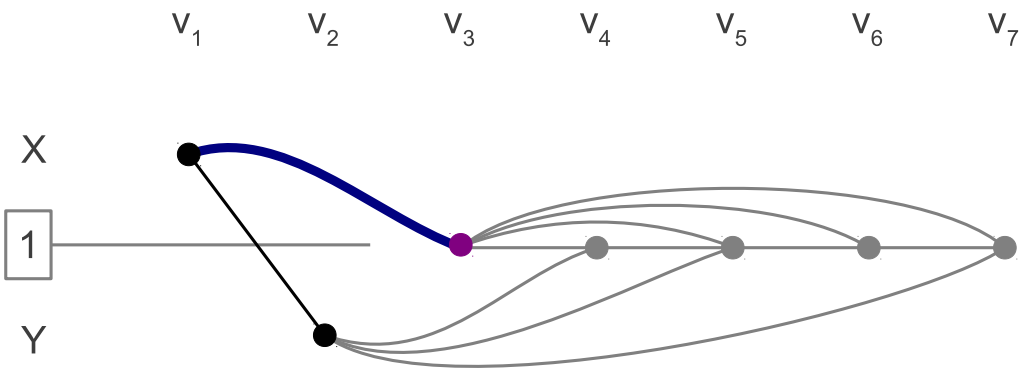
The same example with 7 vertices and 12 edges is used here as in Section A.4.



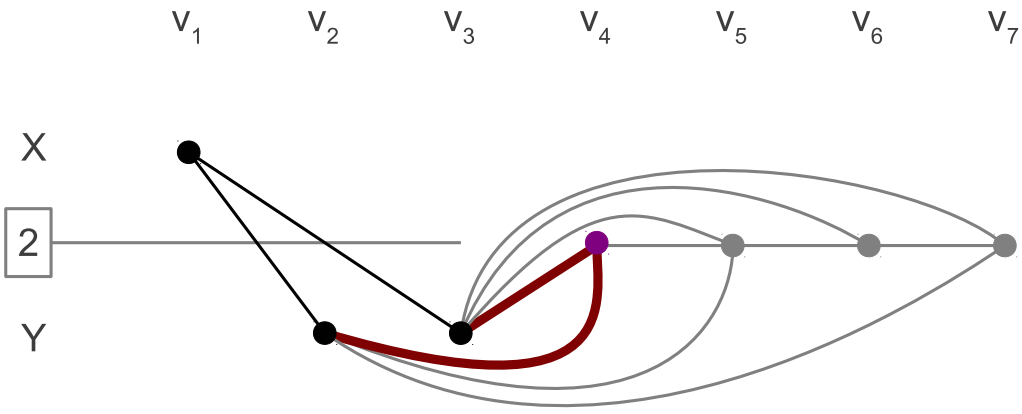
Initially, both X and Y are empty, none of the vertices is put into any of them. First select v_1. As it has neighbors neither in X nor in Y, put it into X.



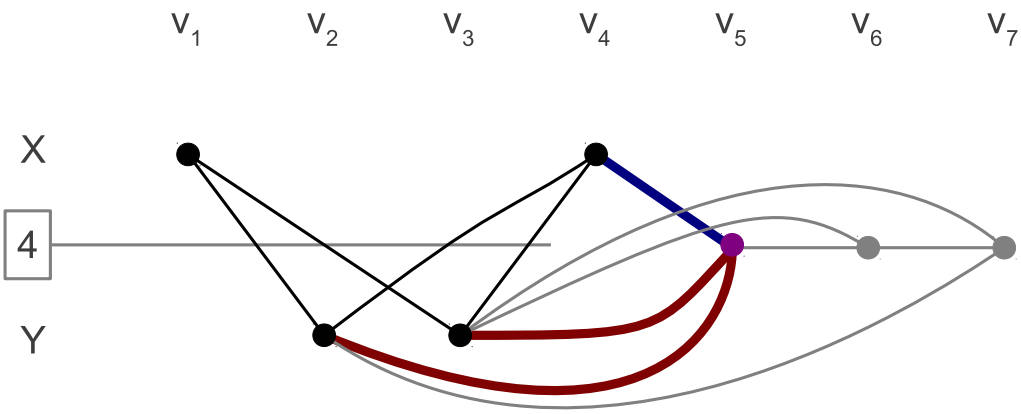
The next vertex selected is v_2, which has one neighbor in X and no one in Y, thus it is added to Y and the size of the cut is increased by 1.



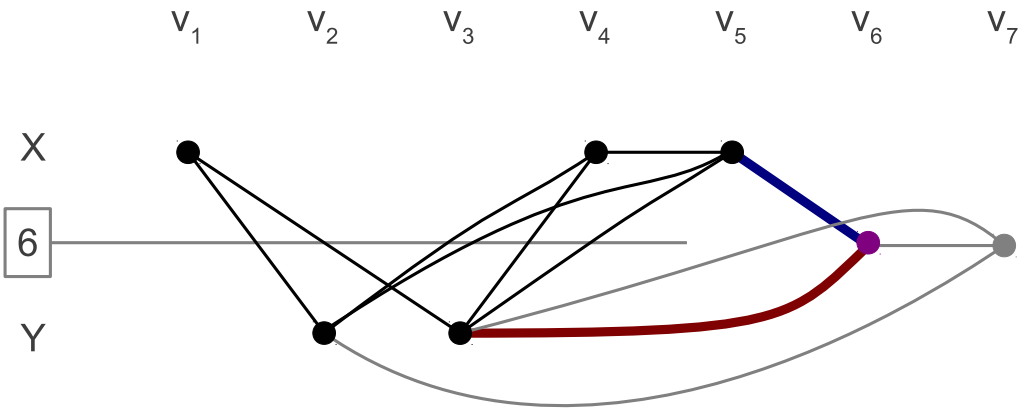
Then, we choose v_3. It has one neighbor in X and none in Y, thus it is put into Y, just as v_2 was in the previous step. Again, the cut is increased by 1.



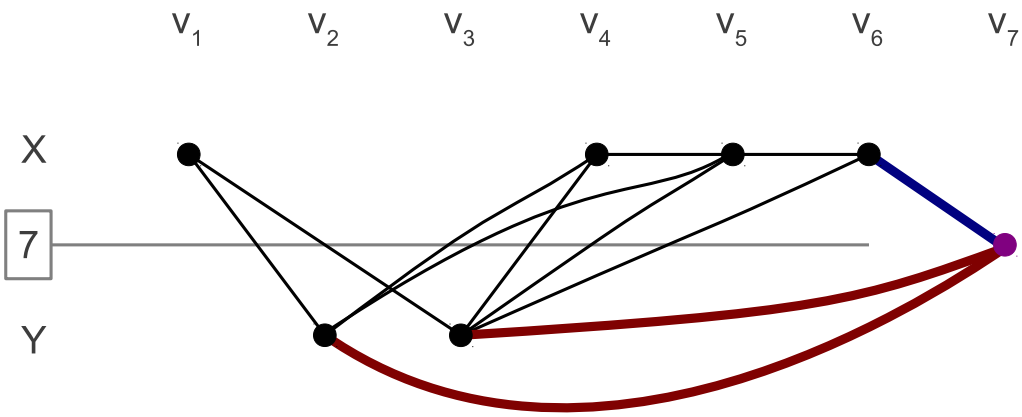
Vertex v_4 has two neighbors, v_3 and v_2, both of them are in Y, hence v_4 is put into X and the cut is increased by 2.



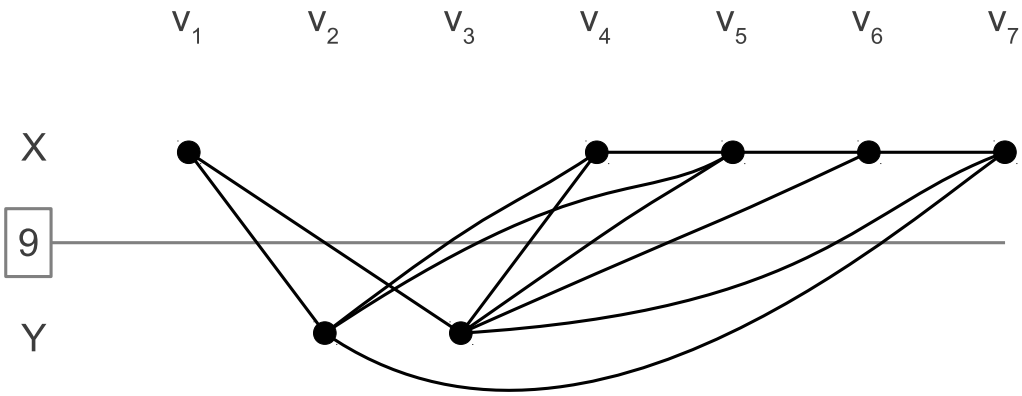
As v_5 has two neighbors in Y and only one in X, it must be put into X, and the size of the cut is increased by 2.



The next vertex v_6 has one neighbor in both of the partition classes, thus the algorithm puts it into X, and the cut is increased by 1.



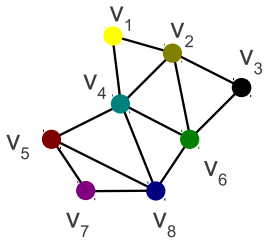
Finally, v_7 has 2 neighbors in Y and only one in X thus we put it into X, and the size of the cut increases by 2.



Now $X \cup Y = V$, i.e., each vertex is added to some partition class and the loop quits. At the end, the size of the cut is 9, that is not optimal, not even locally, as moving v_6 to Y would increase the size by one. Note that the result of this approach is dependent on the order of selection of the vertices. For example, if v_7 is placed before v_6 in the order, then v_7 is added to X. In this case, v_6 has more neighbors in X than in Y, thus it is added to Y. This results in an optimal vertex partition. On the other hand, if the order of the vertices was $v_7, v_6, \ldots, v_2, v_1$, the partition classes obtained would be $X = \{v_7, v_5, v_3, v_1\}$ and $Y = \{v_6, v_4, v_2\}$, and the cut would be of size 8.

A.6 Subtree representation of chordal graphs

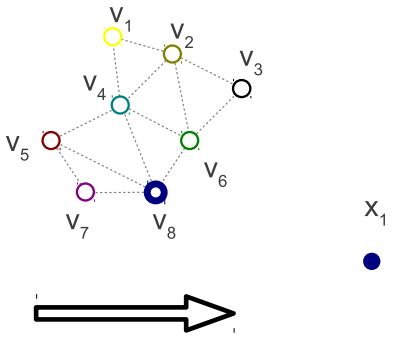
Example



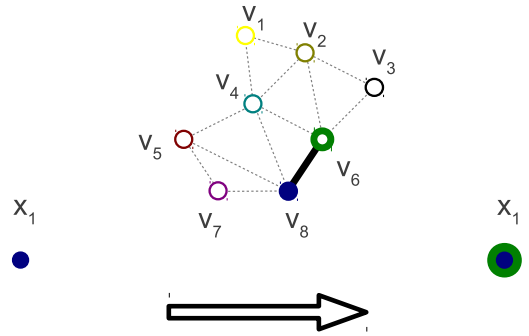
The above graph of order 8 is taken for illustration. The graph has a number of simplicial sequences, the one that we will use is:

$$v_3, v_1, v_2, v_7, v_5, v_4, v_6, v_8.$$

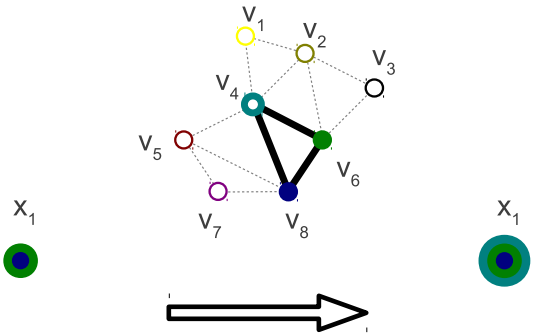
Colors are assigned to each vertex to make the intersection of subtrees more visible. For the construction of the tree the reverse of the simplicial order must be used, thus we start with v_8.



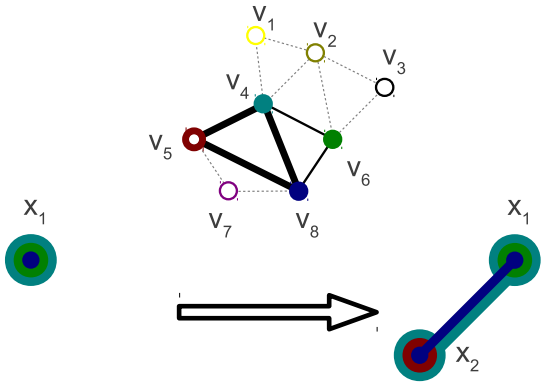
Initially the graph is empty, v_8 has just been added (denoted by thickened border). The construction of the tree starts with introducing node x_1. This node will be present in the subtree representing v_8, thus it is denoted by red. (In the Figures, the left and right trees correspond to the state of the subtree systems before and after adding the current vertex.)



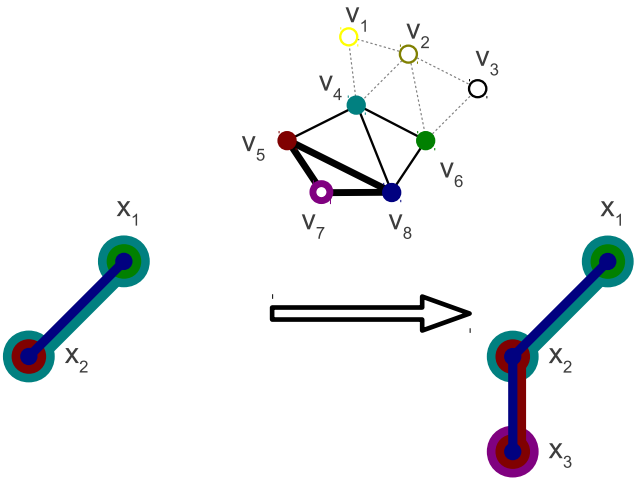
The next vertex to be added is v_6, which is adjacent with v_8. The simplest way to obtain the edge v_6v_8 in the intersection graph of the subtree system is, if that single node in the tree becomes also part of the subtree belonging to the vertex v_6. (Note that other options are also available, e.g., a new node could have been introduced to the tree, that is adjacent with x_1, and the subtree for v_6 would have been the whole tree including both x_1 and x_2.)



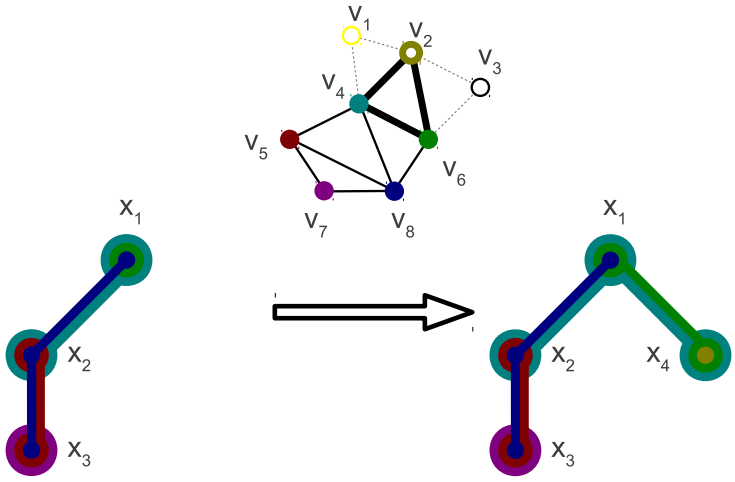
The next vertex is v_4, which is the neighbor of both previous vertices. Hence, the simplest way to extend the subtree system is to include x_1 in the subtree for v_4 as well.



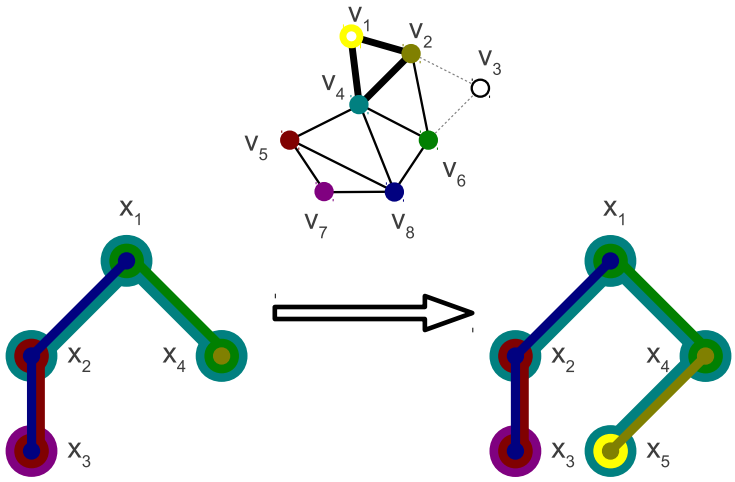
The next vertex in the reverse simplicial order is v_5, that is adjacent to v_4 and v_8, but it is not a neighbor of v_6. As a result, x_1 cannot be a part of the subtree for v_5, and the tree must be extended with a further vertex x_2, where the subtrees belonging to v_8, v_4, and v_5 intersect. However, to maintain the tree properties of the assigned subgraphs, x_1 and x_2 must be adjacent, and the edge between them must be included in the subgraph corresponding to v_4 and v_8 as well.



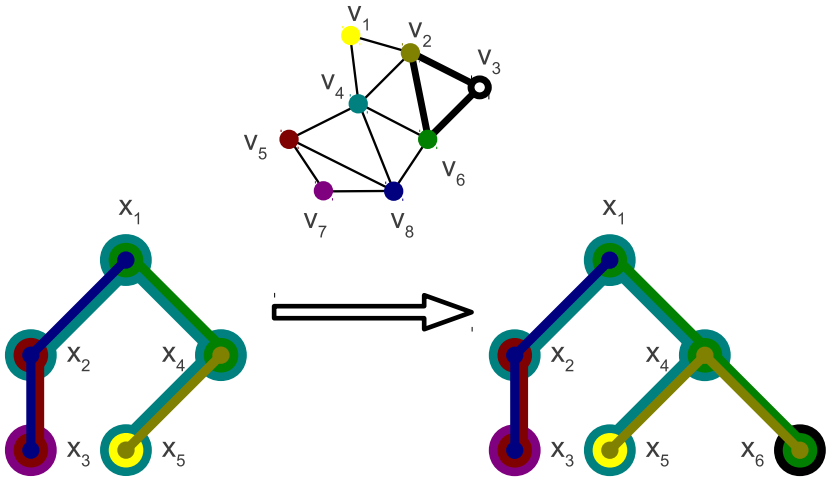
The next vertex v_7 is adjacent with both v_5 and v_8. However, it is not a neighbor of v_4 and therefore, node x_2 cannot be a part of the subtree of v_7. Similarly to the previous case, a new node, x_3 is introduced and added to the subtree of v_7. Moreover, we make x_3 and x_2 to be adjacent, and this edge becomes the part of the subtrees belonging to v_5 and v_8.



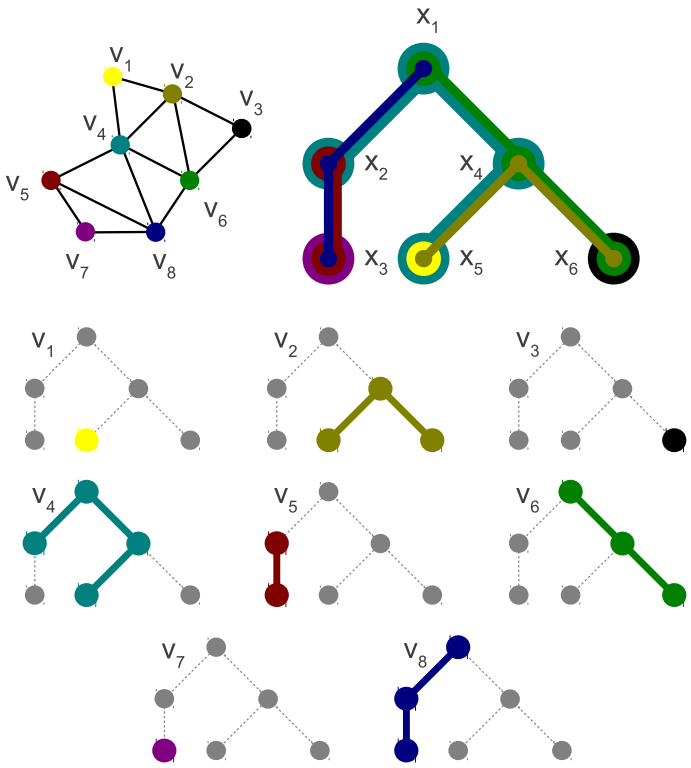
The next vertex, v_2 is the neighbor of v_4 and v_6. The intersection of the subtrees representing v_4 and v_6 contains only the node x_1. However, x_1 also belongs to the subtree for v_8, whilst v_8 is not connected to v_2. Then, as in the previous cases, a new node, x_4 should be introduced in the tree for v_2 and the subtrees of v_4 and v_6 should be extended by x_4.



The next vertex is v_1, the neighbor of v_2 and v_4. As v_1 is not adjacent with v_6, a new node, namely x_5 is introduced for v_1, and the trees of v_2 and v_4 are extended by it.



Finally, v_3 is added to the graph. This vertex is adjacent with v_6 and v_2. We observe that the two subtrees representing v_6 and v_2, respectively, share only the node x_4, but x_4 is also included in the subtree for v_4, which is not adjacent with v_3. Therefore, the tree is extended by x_6, similarly to the previous steps.



The intersection graph of the resulting subtree system yields the original graph.