Decomposition of graphs: some polynomial cases

Cristina Bazgan, Zsolt Tuza, Daniel Vanderpooten

Decomposition of graphs: some polynomial cases

Cristina Bazgan* Zsolt Tuza† Daniel Vanderpooten*

Abstract

We study the problem of decomposing the vertex set V of a graph into two parts (V_1, V_2) which induce subgraphs where each vertex v in V_1 has degree at least $a(v)$ and each vertex v in V_2 has degree at least $b(v)$. We investigate several polynomial cases of this *NP*-complete problem. We give a polynomial-time algorithm for graphs with bounded treewidth which decides if a graph admits a decomposition and gives such a decomposition if it exists. We also give polynomial-time algorithms that always find a decomposition for the following two cases : triangle-free graphs such that $d(v) \geq a(v) + b(v)$ for all $v \in V$ and graphs with girth at least 5 such that $d(v) \geq a(v) + b(v) - 1$ for all $v \in V$.

Keywords: Graph, decomposition, degree constraints, treewidth, girth, complexity, polynomial algorithm.

1 Introduction

For a graph G, we denote by $V(G)$ and $E(G)$ the vertex set and the edge set, respectively. Given a set $S \subseteq V(G)$, the subgraph of G induced by S is denoted by $G[S]$; and we write $d_S(x)$ for the degree of a vertex x in $G[S \cup \{x\}]$ (i.e., $x \in S$ may or may not hold).

* LAMSADE, Université Paris-Dauphine, Place du Marechal de Lattre de Tassigny, 75775 Paris Cedex 16, France. `Email: {bazgan,vdp}@lamsade.dauphine.fr`

† Computer and Automation Institute, Hungarian Academy of Sciences, Budapest; and Department of Computer Science, University of Veszprém, Hungary. `Email: tuza@sztaki.hu` Research supported in part by the Hungarian Scientific Research Fund, grant T-32969.

We consider the following general problem:

DECOMPOSITION
Input: A graph $G = (V, E)$, and two functions $a, b : V \to \mathbb{N}$ such that $a(v), b(v) \leq d(v)$, for all $v \in V$.
Question: Is there a nontrivial partition (V_1, V_2) of V such that $d_{V_1}(v) \geq a(v)$ for every $v \in V_1$ and $d_{V_2}(v) \geq b(v)$ for every $v \in V_2$?

A partition satisfying the previous property is said to be *satisfactory* and is called *decomposition.*

DECOMPOSITION is *NP*-complete. Indeed the special case where $a = b = \lceil \frac{d}{2} \rceil$ has been shown *NP*-complete in [BTV03b].

In this paper we study polynomial instances of this problem. These instances may arise when restricting the structure of the graph, or imposing constraints on a and b, or both.

We are not aware of any previous result on the first case. We show here that, for graphs with bounded treewidth, one can decide in polynomial time if a graph is decomposable and give in polynomial time a decomposition when it exists.

Concerning the second case, Stiebitz [Sti96] proved that, when a and b are such that $d(v) \geq a(v) + b(v) + 1$ for all $v \in V$, any graph admits a decomposition. His result is not constructive. A polynomial-time algorithm that finds such a decomposition is given in [BTV03a].

In the third case, Kaneko [Kan98] showed that any triangle-free graph such that $d(v) \geq s + t$ for all $v \in V$, where s and t are positive integers, admits a decomposition. Diwan [Diw00] showed that any graph with girth at least 5 such that $d(v) \geq s + t - 1$ for all $v \in V$, where s and t are positive integers ≥ 2, admits a decomposition. These two results are not constructive and hold for constants s and t instead of functions a and b. We present here algorithms that give a decomposition in polynomial time for the general case of functions.

The paper is organized as follows. In Section 2, we give a polynomial-time algorithm for graphs with bounded treewidth. The polynomial-time algorithms for triangle-free graphs and graphs with girth at least 5 are presented in Section 3.

2 Decomposition of graphs with bounded treewidth

Many graph problems, including a very large number of well-known *NP*-hard problems, have been shown to be solvable in polynomial time on graphs with treewidth bounded by a constant k [Arn85, Bod88].

Definition A *tree representation* $\mathcal{T} = (T, \mathcal{H})$ of a graph $G = (V, E)$ consists of a tree $T = (X, F)$ with node set X and edge set F, and a set system $\mathcal{H}$

over V whose members $H_x \in \mathcal{H}$ are labeled with the nodes $x \in X$, such that the following conditions are met.

- $\bigcup_{x \in X} H_x = V$.
- For each $uv \in E$ there is an $x \in X$ with $u, v \in H_x$.
- For each $v \in V$, the node set $\{x \in X \mid v \in H_x\}$ induces a subtree of T.

The third condition is equivalent to assuming that if $v \in H_{x'}$ and $v \in H_{x''}$ then $v \in H_x$ holds for all nodes x of the (unique) x'–x'' path in T. The *width* of a tree representation $\mathcal{T}$ is

$$w(\mathcal{T}) = \max_{x \in X} |H_x| - 1$$

and the *treewidth* of G is defined as

$$tw(G) = \min_{\mathcal{T}} w(\mathcal{T})$$

where the minimum is taken over all tree representations $\mathcal{T} = (T, \mathcal{H})$ of G.

The '-1' in the definition of $w(\mathcal{T})$ is included for the convenience that trees have treewidth 1 (rather than 2).

The determination of the treewidth of a graph is *NP*-hard [ACP87]. However, for constant k, Bodlaender [Bod96] gave a linear-time algorithm that determines whether the treewidth of G is at most k, and if so, finds a tree-decomposition of G with treewidth at most k.

As indicated in [Bod97], any tree representation $\mathcal{T} = (T, \mathcal{H})$ of a graph can be transformed in linear time into a *nice* tree representation $\mathcal{T}' = (T', \mathcal{H}')$ with $w(\mathcal{T}') = w(\mathcal{T})$, with linear size in $|T|$ and $H'_x \neq \emptyset$, for all $H'_x \in \mathcal{H}'$, where T' is a *rooted* tree satisfying the following conditions:

(a) Each node of T' has at most two children.

(b) For each internal node x with two children y, y', we have $H'_y = H'_{y'} = H'_x$.

(c) If a node x has just one child y, then

$$H'_x \subset H'_y \quad \text{or} \quad H'_y \subset H'_x \qquad \text{and} \qquad \big| |H'_x| - |H'_y| \big| = 1\,.$$

Theorem 1 DECOMPOSITION *can be decided in polynomial time for graphs of treewidth less than k for every fixed $k > 1$. Moreover, a decomposition can be found in polynomial time if it exists.*

Proof: Consider a tree representation of width less than k which can be obtained in linear time by the algorithm proposed in [Bod96]. Let $\mathcal{T} = (T, \mathcal{H})$ be a nice tree representation, rooted in r, obtained from the previous one.

The essential part of the algorithm is dynamic programming, organized as a *postorder* traversal of (T, r). For each node x of T the following data will be calculated:

- a set $\mathcal{P}_x$ of *bipartitions* of H_x,
- for each $P = (A, B) \in \mathcal{P}_x$ a set $I(P)$ of integer vectors $i_1(P), i_2(P), \ldots$ of length $|H_x|$,
- indicators Y or N telling whether P or some of its feasible extensions is a nontrivial one (i.e. with both classes being nonempty),
- if x is not a leaf, then one or two pointers from each $i_j(P) \in I(P)$ to the child(ren) y of x indicating which partition(s) at the node(s) y have been used in creating $i_j(P)$.

The vectors in $I(P)$ are the possible degree sequences of the vertices in H_x, collected for all feasible partitions of the subgraph of G induced by the vertices that occur in the sets H_z, where z runs over the nodes of the subtree of T rooted at x. That is, several vectors may be associated with the same P.

Since $H_x = H_y$ may occur, sometimes we shall use the more precise notation $i(P, x)$ or $i_j(P, x)$ to indicate that the vector belongs to a partition *at the node* x. Analogously, $I(P, x)$ will stand for the set of vectors for P at node x. The coordinate for $v \in H_x$ in $i_j(P, x)$ will be denoted by $i_j(P, x; v)$.

In the trivial case where T consists of just one node, G can have at most k non-isolated vertices, therefore the existence of a decomposition can be decided by brute force in constant time (since k is fixed). Hence, we assume that T has at least one leaf.

Depending on the position of x in T, those P and $i(P)$ are computed as follows.

<u>Leaf</u>. If $x \in X$ is a leaf of T, then $\mathcal{P}_x$ consists of all partitions $P = (A, B)$ of H_x. The coordinates of $i(P)$ are the degrees $d_A(v)$ for $v \in A$ and $d_B(v)$ for $v \in B$. The indicator is N if $A = \emptyset$ or $B = \emptyset$, and it is Y otherwise.

<u>Two children</u>. Let $x \in X$, its two children y' and y''. Consider any partition $P = (A, B)$ of H_x. If $I(P, y') = \emptyset$ or $I(P, y'') = \emptyset$, we also define $I(P, x) = \emptyset$. Otherwise from each pair $i_{j'}(P, y') \in I(P, y'), i_{j''}(P, y'') \in I(P, y'')$ a vector $i_j(P, x) \in I(P, x)$ is obtained by the rule

$$i_j(P, x; v) = i_{j'}(P, y'; v) + i_{j''}(P, y''; v) - d_A(v) \qquad \forall\, v \in A$$

$$i_j(P, x; v) = i_{j'}(P, y'; v) + i_{j''}(P, y''; v) - d_B(v) \qquad \forall\, v \in B$$

In this case we also introduce pointers from each $i_j(P,x) \in I(P,x)$ to the corresponding $i_{j'}(P,y')$ and $i_{j''}(P,y'')$. The indicator for $i_j(P,x)$ is Y if and only if so is at least one of those for $i_{j'}(P,y')$ and $i_{j''}(P,y'')$. If the same $i_j(P,x)$ has already been obtained from a previous pair, then we keep the earlier pointers unless the new pair would change the indicator from N to Y.

Larger child. Assume $H_x = H_y \setminus \{v\}$, where y is the child of x. For each $P=(A,B)$ at y and for each $i(P,y)$ we check whether $i(P,y;v) \geq a(v)$ if $v \in A$ or $i(P,y;v) \geq b(v)$ if $v \in B$. If so, then we maintain the corresponding partition $(A \setminus \{v\}, B)$ or $(A, B \setminus \{v\})$, omit the v-coordinate from $i(P)$, introduce a pointer from $i(P-v,x)$ to $i(P,y)$, and keep the Y/N indicator for $i(P-v,x)$ the same as the one for $i(P,y)$.

(The same partition (A,B) of H_x may be obtained from $(A \cup \{v\}, B)$ and $(A, B \cup \{v\})$ of H_y. If they yield the same vector, only one of them is kept for (A,B), with just one pointer.)

Smaller child. Assume $H_x = H_y \cup \{v\}$, where y is the child of x. From each partition $P=(A,B)$ of H_y we generate two partitions $P' = (A \cup \{v\}, B)$ and $P'' = (A, B \cup \{v\})$ of H_x. The indicator remains Y if it was Y for P, and is changed from N to Y for P' or P'' if $A = \emptyset$ or $B = \emptyset$, respectively. Otherwise it remains N.

¿From each $i(P)$ the corresponding $i(P')$ is obtained by increasing the coordinates at the neighbors of v in A by 1, and introducing a new v-coordinate whose value is equal to $d_A(v)$. The computation of $i(P'')$ is analogous. For both of them the pointer specifies $i(P,y)$ for $i(P \cup \{v\}, x)$.

Root. Graph G has a decomposition if and only if there exists a partition $P=(A,B)$ at the root r and a vector $i(P)$ such that

- $i(P,r;v) \geq a(v)$ for all $v \in A$ and $i(P,r;v) \geq b(v)$ for all $v \in B$, and
- P has indicator Y.

These requirements are easily tested for each $i(P)$. Having found one affirmative case, from $i(P,r)$ one can trace back a sequence of vectors down to all the leaves of T. This sequence determines a vertex partition of the entire G, in which the degree conditions are satisfied.

Correctness. The two trivial partitions keep indicator N all along T, also at r, therefore they will not be considered as solutions. Suppose next that a nontrivial partition P^* is not satisfactory. We show that the algorithm does not output P^* as a solution. By assumption, P^* contains a vertex v whose degree in A or B is less than $a(v)$ or $b(v)$, respectively. Let us consider the subtree T_v of T, at the nodes of which v is listed. Let y be the highest node of T_v, and x the parent of y if $y \neq r$. (If this x exists, it cannot have two children.) We denote by $P=(A,B)$ the partition of H_y generated by P^*.

If no member of $I(P, y)$ corresponds to P^*, then we will not get P^* as a solution. Suppose that $i(P, y)$ is generated by P^*. If $y = r$, then v violates the condition at the 'Root' step; and if $y \neq r$, then $H_x = H_y \setminus \{v\}$ and the coordinate $i(P, y; v)$ violates the degree constraint in the step 'Larger child', consequently no pointer can lead to $i(P, y)$ from $i(P - v, x)$. Thus, the partition generated by the algorithm is satisfactory.

Time analysis. Let $n = |V|$ denote the number of vertices. The key point we are going to show is that for each node a polynomially bounded number of data is maintained.

Every H_x has at most 2^k partitions, which yields just a constant number of possible P. Then $i_j(P, x)$ has at most k coordinates, each representing vertex degree and hence being in the range $[0, \dots, n-1]$. Consequently, the number of partition/vector combinations at x is at most $(2n)^k$, polynomial in n. If x has at most one child, the computation for each $i_j(P, x)$ obviously requires a polynomial number of steps only. Similarly, if x has two children y' and y'', then $\max(|I(P, y')|, |I(P, y'')|) \leq n^k$, therefore $I(P, x)$ is generated by at most n^{2k} pairs of degree vectors. Each of them requires a polynomial number of steps. □

3 Decomposition of triangle-free graphs and graphs with girth at least 5

We first introduce some basic definitions.

For a graph $G = (V, E)$, a subset $X \subseteq V$, and a function $h : V \to \mathbb{N}$,

- X is an *h-satisfactory subset* if $d_X(v) \geq h(v)$ for all $v \in X$
- X is a *minimal h-satisfactory subset* if it is an h-satisfactory subset and for every $Y \subset X$, there exists a vertex $v \in Y$ such that $d_Y(v) \leq h(v) - 1$.
- X — or the subgraph $G[X]$ — is *h-degenerate* if every $Y \subseteq X$ contains a vertex v such that $d_Y(v) \leq h(v)$
- assuming that X is h-degenerate, an *h-elimination order* on X is a permutation $v_1, v_2, \dots, v_{|X|}$ of the vertices of X such that each v_i $(1 \leq i < |X|)$ is adjacent to at most $h(v_i)$ vertices v_j with larger subscript, $i < j \leq |X|$.

It is decidable in polynomial time if a set X is h-degenerate (Proposition 4 of [BTV03a]). Moreover, if X is h-degenerate, an h-elimination order on X can be obtained by the following polynomial-time algorithm. Let v_1 be a vertex of X of degree $\leq h(v_1)$. Once $v_1, \dots, v_i$ are defined, let v_{i+1} be a vertex of $X - \{v_1, \ldots, v_i\}$ of degree $\leq h(v_{i+1})$. The existence of this vertex is guaranteed since X is h-degenerate.

We also recall the following procedure from [BTV03a], which is the algorithmic analogue of Stiebitz's Lemma [Sti96].

EXTEND(A, B)
Input: two disjoint nonempty subsets $A, B \subseteq V$ such that A is not $(a-1)$-degenerate and B is not $(b-1)$-degenerate.
Output: a decomposition (V_1, V_2).

Find A', an a-satisfactory subset of A by removing iteratively vertices v from $G[A]$ of degree less than or equal to $a(v)-1$ while it is possible. Find B', a b-satisfactory subset of B in a similar way. Let $V_1 = A'$ and $V_2 = B'$. While there is a vertex v in $V \setminus (V_1 \cup V_2)$ such that $d_{V_1}(v) \geq a(v)$, add v in V_1. While there is a vertex v in $V \setminus (V_1 \cup V_2)$ such that $d_{V_2}(v) \geq b(v)$, add v in V_2. At the end, if $C = V \setminus (V_1 \cup V_2) \neq \emptyset$, then $d_{V_1}(v) < a(v)$ and $d_{V_2}(v) < b(v)$ for any $v \in C$. Since $d(v) \geq a(v) + b(v)$ (in the case of triangle-free graphs) or $d(v) \geq a(v) + b(v) - 1$ (in the case of graphs with girth at least 5), we have, for any $v \in C$, $d_{V_1 \cup C}(v) \geq a(v)$ and $d_{V_2 \cup C}(v) \geq b(v)$. Thus we can add all vertices of C either in V_1 or in V_2, forming a decomposition.

Theorem 2 DECOMPOSITION *has always a solution for triangle-free graphs $G = (V, E)$ such that $d(v) \geq a(v) + b(v)$ for all $v \in V$. Moreover, a decomposition can be found in polynomial time.*

Proof: We present an algorithm that finds the required decomposition.

This algorithm maintains a vertex partition (A, B) of the input graph $G = (V, E)$, together with an ordering $v_1, \ldots, v_{|A|}$ of the vertices of A, with the following properties:

(1) $|A| \geq 2$ and $|B| \geq 2$

(2) A is a-degenerate but not $(a-1)$-degenerate

(3) $d_A(v_1) = a(v_1)$, $d_A(v_2) = a(v_2)$, and $v_1 v_2 \in E$

(4) $v_1, v_2, \ldots, v_{|A|}$ is an a-elimination order on A

(5) Deleting any one of v_1 or v_2 from $v_1, \ldots, v_{|A|}$, an $(a-1)$-elimination order on $A - v_1$ or $A - v_2$ is obtained, respectively.

Let us note that the assumption $|B| \geq 2$ in (1) follows from (3), because v_1 and v_2 together have at least $b(v_1) + b(v_2) \geq 2$ neighbors in B but they do not have a common neighbor since G is triangle-free. Also, if $A \neq \emptyset$, then (2) implies $|A| \geq 2$ because $a(v) \geq 1$ for every $v \in V$.

PREPROCESSING

Find a *minimal* a-satisfactory subset $A \subseteq V$ in polynomial time applying an algorithm presented in [BTV03a]. Then select v_1 in A such that $d_A(v_1) = a(v_1)$, and find an $(a-1)$-elimination order on $A - v_1$. Finally, set $B = V \setminus A$.

Minimality of A means that there is at least one vertex v_1 with $d_A(v_1) = a(v_1)$ (for otherwise removing any one vertex, the subset would still be a-satisfactory); moreover, $A - v_1$ is $(a-1)$-degenerate. That is, some v_2 has degree at most $a(v_2) - 1$ in $A - v_1$. But A was a-satisfactory, i.e. $d_A(v_2) \geq a(v_2)$. The only possibility is that v_2 has degree $a(v_2)$ in A, and $v_1 v_2$ must be an edge. All conditions (1)–(5) above can be satisfied in this way.

The algorithm will either find a satisfactory partition at the first line of the Main Loop below or perform some modifications in (A, B). At any step, the actual value of the quantity

$$w(A, B) = |E(G[A])| + |E(G[B])| + \sum_{v \in A} b(v) + \sum_{v \in B} a(v)$$

is assigned to (A, B). The key point is that if the first line does not terminate the algorithm, then a modified partition will have a larger $w(A, B)$ value. Since $w(A, B) = O(|V| \cdot |E|)$, the number of rounds where the Main Loop is performed is polynomial.

MAIN LOOP

1. If the set $B = V \setminus A$ is not $(b-1)$-degenerate, then run EXTEND(A, B) to find a satisfactory partition (V_1, V_2) and STOP;

   else select a vertex $x \in B$ with $d_B(x) < b(x)$.

2. If $v_1 x \in E$, then exchange $v_1 \leftrightarrow v_2$.

   // Since G is triangle-free, at least one of $v_1 x$ and $v_2 x$ is a non-edge. //

3. $A := A \cup \{x\}$, $B := B - x$, and put x at the end of the a-elimination order.

   // This remains an a-elimination order, because $v_1 x \notin E$ and $A - v_1 - x$ has been $(a-1)$-degenerate. //

4. If $v_2 x \in E$ and $A - v_1$ is not $(a-1)$-degenerate, then set $A := A - v_1$ and $B := B \cup \{v_1\}$.

   // This ensures $|B| \geq 2$ again, keeping A a-satisfactory. //

5. Find the smallest subscript i such that the set $S_i := \{v_{i+1}, v_{i+2}, \ldots, v_{|A|}\}$ is $(a-1)$-degenerate.

6. Re-define $A := \{v_i\} \cup S_i$, $B := V \setminus A$, and update the a-elimination order on A to ensure the properties (3)–(5).

One can observe that these steps are feasible and can be performed in polynomial time. We should note that $|B| \geq 2$ holds after Line 4 also in the cases where v_1 remains in A. Indeed, if $v_2x \notin E$, then v_1 and v_2 still have at least $b(v_1) + b(v_2) \geq 2$ distinct neighbors in B; and if $A - v_1$ is $(a-1)$-degenerate, then v_1 is adjacent to some $v \in A$ such that $d_A(v) = a(v)$, consequently $|B| \geq b(v_1) + b(v) \geq 2$.

Since the initial conditions (1)–(5) are maintained after all, the proof will be done if we show that $w(A, B)$ gets increased whenever the algorithm does not stop at Line 1. We need to investigate those steps where (A, B) is or may be modified, namely the lines 3, 4, and 6.

When x is deleted from B, $|E(G[B])|$ decreases by at most $b(x) - 1$ and $\sum_{v \in B} a(v)$ by exactly $a(x)$. Inserting x into A increases $|E(G[A])|$ by at least $a(x) + 1$ and $\sum_{v \in A} b(v)$ by exactly $b(x)$. Thus, in this step $w(A, B)$ increases by at least 2.

Moving v_1 from A to B does not decrease $w(A, B)$, because we delete exactly $a(v_1)$ edges from $G[A]$ and subtract $b(v_1)$, and then add $a(v_1)$ and extend B with at least $b(v_1)$ edges.

The situation is similar (but may be even better) when the vertices v_j $(j < i)$ are moved from A to B. Since we have an a-elimination order, v_j has at most $a(v_j)$ neighbors with a larger subscript. Hence, if these vertices are moved from A to B sequentially in the order of a-elimination, in each step the corresponding v_j has at least $b(v_j)$ neighbors in the updated set B. Thus, $w(A, B)$ does not decrease.

Summarizing the three cases, the Main Loop increases $w(A, B)$ by at least 2. □

We consider now the case of graphs with girth at least 5. Combining ideas from the proof of [Diw00] with those in the algorithm above, the following generalization of Diwan's theorem can be proved:

Theorem 3 DECOMPOSITION *has always a solution for graphs* $G = (V, E)$ *with girth at least 5 such that* $d(v) \geq a(v) + b(v) - 1$ *for all* $v \in V$ *where* $a, b \geq 2$. *Moreover, a decomposition can be found in polynomial time.*

That is, also in this case, the constant assumptions on vertex degrees can be replaced by arbitrary functions $a(v), b(v) \geq 2$. The corresponding algorithm is more complicated to describe than for the triangle-free graphs, because in some situations the roles of the partition classes A and B have to be switched. In this sense the algorithm is a relative of our previous one in [BTV03a], which worked for all graphs (i.e., without girth considerations), under the condition $d(v) \geq a(v) + b(v) + 1$.

References

[Arn85] S. Arnborg, *Efficient algorithms for combinatorial problems on graphs with bounded decomposability-A survey*, BIT, 25 (1985), 2–23.

[ACP87] S. Arnborg, D. G. Corneil and A. Proskurowski, *Complexity of finding embeddings in a k-tree*, SIAM Journal of Algebraic and Discrete Methods, 8 (1987), 277–284.

[BTV03] C. Bazgan, Zs. Tuza and D. Vanderpooten, *On the existence and determination of satisfactory partitions in a graph*, to appear in Proceedings of ISAAC 2003.

[BTV03a] C. Bazgan, Zs. Tuza and D. Vanderpooten, *On a theorem of Stiebitz about decomposing graphs under degree constraints*, submitted 2003.

[BTV03b] C. Bazgan, Zs. Tuza and D. Vanderpooten, *Complexity of the satisfactory partition problem*, submitted 2003.

[Bod88] H. L. Bodlaender, *Dynamic programming algorithms on graphs with bounded treewidth*, Proceedings of the 18th International Colloquium on Automata, Languages and Programming, 1988, LNCS 317, 105–119.

[Bod96] H. L. Bodlaender, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM Journal of Computing, 25 (1996), 1305–1317.

[Bod97] H. L. Bodlaender, *Treewidth: algorithmic techniques and results*, Proceedings 22nd International Symposium on Mathematical Foundations of Computer Science, 1997, LNCS 1295, 19–36.

[Diw00] A. Diwan, *Decomposing graphs with girth at least five under degree constraints*, Journal of Graph Theory 33 (2000), 237–239.

[GK00] M. Gerber and D. Kobler, *Algorithmic approach to the satisfactory graph partitioning problem*, European Journal of Operation Research, 125 (2000), 283–291.

[Kan98] A. Kaneko, *On decomposition of triangle-free graphs under degree constraints*, Journal of Graph Theory 27 (1998), 7–9.

[Sti96] M. Stiebitz, *Decomposing graphs under degree constraints*, Journal of Graph Theory 23 (1996), 321–324.