GRAPH DECOMPOSITION BASED ON DEGREE CONSTRAINTS

ZOÉ HAMEL

March 3, 2016

1. Introduction

Let $G = (V(G), E(G))$ be a graph G (loops and multiple edges not allowed) on the set of vertices $V(G)$ and the set of edges $E(G)$. The *degree of a vertex* i in $V(G)$, $deg_G(i)$, is the number of edges of G incident with i. A *subgraph* F of G is a graph $F = (V(G), E(F))$ such that $E(G) \subseteq E(F)$. This paper focuses on the following problem: given a graph G on n vertices, does there exists a subgraph F of G such that the degree of the i^{th} vertex is given by the function $f : V(G) \to \mathbb{Z}$ via $f(i) = deg_F(i)$ for all $i = 1, 2, ..., n$. This problem is called the *f-factor problem*, and the subgraph of G satisfying the function $f(i) = deg(i)$ is an *f-factor* of G. We generalize these concepts to a *general graph*; a graph on which multiple edges and loops are allowed. For example, a loop at i counts for 2 towards the degree of i.

Tutte's f-factor theorem provides a characterization of general graphs for which a given f-factor exists. Anstee gave an algorithmic proof of Tutte's theorem to solve the f-factor problem [2]. One goal of this paper is to give the details of Anstee's paper in which he combines a network flow approach and a graph theory approach in order to either find a given f-factor or show that none can exist. Section 3 provides basic theory and results about network flows necessary to understand how network flows are used.

For the purpose of this paper, it will be useful to think of a general graph G in term of its adjacency matrix. We define $\lambda(e)$, the *multiplicity of an edge* $e = (i, j)$, as the number of edges joining i and j in G. The *adjacency matrix* of G, denoted A_G, is a matrix with rows and columns labelled by the vertices of the graph, with entries $\lambda(i, j)$ in position (i, j) off the diagonal, and with entries $2\lambda(i, i)$ in position (i, i) on the diagonal. Note that any general (undirected) graph has a symmetric adjacency matrix. Therefore, the adjacency matrix of an f-factor F of G is a symmetric matrix with integral entries, i^{th} row and column sums equal to $f(i)$, even entries on the diagonal, and satisfying $0 \leq A_F \leq A_G$.

Let $D(G)$ be the directed graph associated to a general graph G such that $A_G = A_{D(G)}$ as in the example in Figure 1. We define a *directed f-factor* to be a subgraph B of $D(G)$. Therefore, the adjacency matrix of a directed f-factor B has i^{th} row and column sums equal to $f(i)$ for all $i = 1, ..., n$ and satisfies $0 \leq A_B \leq A_G$.

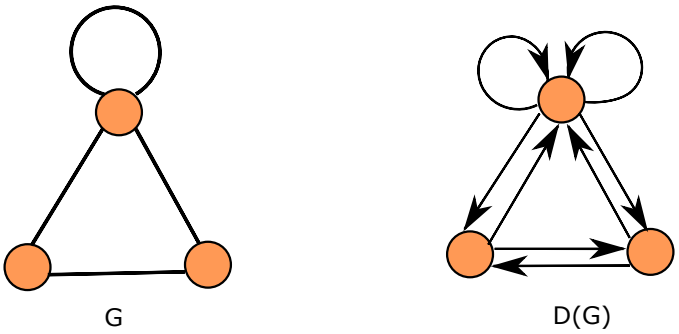


FIGURE 1. General graph G and its associated directed graph $D(G)$.

Starting with a general graph G on n vertices and given a function f, we want to determine whether G has an f-factor or not. We will solve this problem in two phases described in details in Section 4. The first phase consists in looking for a directed f-factor in $D(G)$. In this first phase, network flows will be used in order find a directed f-factor or prove that none exists. We will show that if $D(G)$ doesn't have a directed f-factor, then G cannot have an f-factor. On the other hand, if $D(G)$ has a directed f-factor, G may have an f-factor.

The second phase of the problem consists in trying to transform this directed f-factor B into an f-factor of G. As noted earlier, A_B already has i^{th} row and column sum equal $f(i)$ and $0 \leq A_B \leq A_G$. We need to transform A_B into (1) a symmetric matrix (we will call this a fractional f-factor), (2) with integral entries off the diagonal and (3) even entries on the diagonal. During the transformations, we will have to preserve the row and column sums and the inequality $0 \leq A_B \leq A_G$. While making the matrix symmetric is straight-forward and can always be done, obtaining integral entries on the diagonal and even entries on the diagonal involves a relatively complex algorithm which will be specifically discussed in Section 5. If using this algorithm we are unable to obtain an f-factor, we will prove in Section 6 that no f-factor can exist using Tutte's characterization.

Section 7 generalizes our theorem to the (g, f)-factor problem and then specializes to the 1-factor problem. Section 8 describes the sparse substitute idea of Gabow. Section 9 gives two applications of (g, f)-factors: one is related to the edge colouring problem and one about graph decomposition.

2. Notations and definitions

Some of these definition will become clear when explained in context throughout the paper.

General graphs.

Graph $G = (E(G), V(G))$**:** A graph on the set of edges $E(G) : \{e : e \in E(G)\}$ and the set of vertices $V(G) = \{v : v \in V(G)\}$.

General graph: A graph in which multiple edges and loops are allowed.

Loop $e = (v, v)$: An edge joining a vertex v to itself.

Multiple edges: Several edges joining the same two vertices.

Degree of a vertex v in G, $deg_G(v)$: The number of edges in G incident with v. Note that loops are special cases which add 2 to the degree, each connection to v counting as 1.

Multiplicity of the edge $e = (i, j)$, $\lambda(e)$: The number of edges of G joining i to j.

Empty edge: An edge e such that $x(e) = 0$.

Full edge: An edge e such that $\lambda(e) - x(e) = 0$.

Partition of $V(G)$: A collection of subsets V_i of the vertices such that

$$\bigcup_i V_i = G(V) \text{ and } V_j \bigcap V_i = \emptyset \text{ for } i \neq j.$$

Adjacency matrix of G, A_G: A matrix with rows and columns labelled by the graph vertices, with entries $\lambda(i, j)$ in position (i, j) off the diagonal, and with entries $2\lambda(i, i)$ in position (i, i) on the diagonal.

Connected graph: A graph where there is a path from any vertex to any other vertex in the graph.

Spanning subgraph F of G: A graph $F = (V(G), E(F))$ such that $0 \leq A_F \leq A_G$ where for all $e \in E(F)$, $x(e) \in \mathbb{Z}$. If we drop the requirement $x(e) \in \mathbb{Z}$, we say that F is a fractional subgraph.

Subgraph G_n of G: A subgraph of G such that $V(G_n) = \{v \in V(G) : deg_G(v) = n\}$.

Underlying subgraph of G: A subgraph obtained by removing the loops from G and replacing the multiple edges by a single edges.

Independent vertex set in G: Subset of $V(G)$ such that no two vertices of this subset are joined by an edge of G.

Bipartite graph G: A (general) graph for which the set of vertices $V(G)$ is partitioned into two disjoint independent vertex sets X_1 and X_2; that is, $X_1 \cup X_2 = V(G)$

and $X_1 \cap X_2 = \emptyset$.

Subgraph of G induced by the vertex set $X \subseteq V(G)$: A subgraph of G with vertex set X and edge set consisting of the edges of G joining vertices of X.

Network flows (See more details in Section 3).

Directed graph, $D = (N(D), A(D))$: A directed graph with the set of nodes $N(D) = \{n : n \in N(D)\}$ and the set of arcs $A(D) = \{a : a \in A(D)\}$, where each are an ordered pair of nodes.

Directed graph $D(G)$: A directed graph associated to any undirected general graph G such that $A_{D(G)} = A_G$; in other words, every edge $e = (i, j) \in G$ corresponds to two arcs in $D(G)$, $a_1 = (i, j)$ and $a_2 = (j, i)$.

Network $Q = (N(Q), D(Q))$: A directed graph with data and two special nodes: a source s and a sink t.

Network $Q(G)$: A network associated to any undirected general graph G by a special construction (see Section 4.1).

Capacity function of Q: The function $u : A(Q) \to \mathbb{R}$, acting as an upper bound on the flow function of Q.

Lower bound function of Q: The function $l : A(Q) \to \mathbb{R}$, acting as a lower bound on the flow function of Q.

Flow function of Q: The function $x : A(Q) \to \mathbb{R}$ satisfying the capacity constraint: $l(a) \leq x(a) \leq u(a)$.

Empty arc: An arc a such that $x(a) = 0$.

Full arc: An arc a such that $u(a) - x(a) = 0$.

Flow conservation condition:

$$\sum_{i:(i,j)\in A} x(i, j) = \sum_{j:(i,j)\in A} x(i, j) \quad \forall i \neq s, t.$$

Size of the flow: The total amount of flow leaving the source s.

Maximal flow of Q: A flow in Q which has the maximal size among all flows in the network Q.

Network flow problem: Finding the maximum flow which can be pushed from s to t.

st-cut: A partition of the nodes in two sets Y and Z such that the source s is in the set Y and the sink t is in Z.

Net flow across cut: (flow leaving Y) - (flow entering Y).

Capacity of a cut: The sum of the capacity of each arc going from Y to Z.

Minimum cut of G: A cut which has the minimum capacity among all cuts in a network Q.

The excess capacity of a cut, $t_{Y,Z}$: (capacity of the cut) - (size of the flow).

Residual network of Q, Q': A network formed based on Q and a given flow x.

Residual capacity of Q': The capacity function of the residual network Q', $r : A(Q') \to \mathbb{R}$.

Augmenting path: A directed path P from s to t inside the residual network such that $r(a) > 0$ for all $a \in P$.

Ford-Fulkerson algorithm: An algorithm to find a maximum flow for any given network associated with an initial flow.

Walks and Trees.

Walk: A sequence of vertices and edges $v_1, e_1, v_2, e_2, v_3, ..., e_{n-1}, v_n$ in a general graph G which may pass through the same edge or vertex more than once.

Path: A walk with no repeated vertices.

Trail: A walk with no repeated edges.

Euler trail: A trail going through every edge of a given graph exactly once.

Length of a walk: The number of edges of a walk.

Closed walk: A walk such that $v_0 = v_n$.

Cycle: A closed trail such that the only repeated vertex is $v_0 = v_n$.

Tree: A connected graph without cycles.

Leaf: A vertex v in the tree such that $deg(v) = 1$.

Matchings (See more details in Section 5.1).

Matching M: A matching M in G is a set of edges without common vertices in G.

Size of a matching M: The number of edges $e \in M$.

Maximal matching: A matching M such that if we add an edge $e \notin M$, then it is no longer a matching.

Matching problem: Given a graph G, finding a maximum matching.

Alternating path in a matching M: A path whose edges are alternatively in and out the matching M.

Augmenting path in a matching M: An alternating path in M satisfying the additional condition that its two end vertices aren't incident to any edges of M.

Perfect matching: A matching M in G such that for all $i \in G$, $deg_M(i) = 1$.

f-factors (See more details in Section 4).

f-factor of G: A subgraph F of G satisfying $f(i) = deg_F(i)$ for all $i \in V(G)$, given a function f. If F is a fractional subgraph, it is a fractional f-factor.

f-factor problem: Given a general graph G and a function f, determining whether G has an f-factor or not.

(g, f)-factor of G: A subgraph F of G satisfying $g(i) \leq deg_F(i) \leq f(i)$ for all $i \in V(G)$, given the functions f and g. If F is a fractional subgraph, it is a fractional (g, f)-factor.

Directed f-factor of $D(G)$: A subgragh B of $D(G)$ where A_B has i^{th} row and column sums equal to $f(i)$ and satisfying $0 \leq A_B \leq A_G$.

Directed (g, f)-factor of $D(G)$: A subgragh B of $D(G)$ where A_B has row and column sums, $r(i)$ and $c(i)$ repectively, satisfying $g(i) \leq r(i) \leq f(i)$ and $g(i) \leq c(i) \leq f(i)$.

f-odd component C of G: In Tutte's f-factor Theorem 4.1, for a given a partition S, T, U of $V(G)$, the components of the graph $G - (S \cup T)$ satisfying

$$\sum_{x \in V(C)} f(x) + e_G(C, T) \equiv 1 \pmod 2.$$

where $e_G(C, T)$ counts all edges (i, j) in G such that $i \in C$, $j \in T$, according to multiplicity.

f-barrier: A partition of $V(G)$ which upon examination reveals that G has no f-factor.

Auxiliary graph H: A subgraph H (with a special construction) of the general graph G for which we want to solve the f-factor problem.

Alternating Walks (See more details in Section 5).The algorithm to find alternating walk in a graph involves tree growing creating blossoms.

Alternating walk: A walk $v_1, e_1, v_2, e_2, v_3, ..., e_{n-1}, v_n$ such that we can add 1 to entry $x(e_1)$, subtract 1 from entry $x(e_2)$, etc. (or 2 if e_i is a loop) while preserving $0 \leq A_F \leq A_G$; or the same series of operations starting with subtraction.

Reachable: A vertex i is reachable from j if there is an alternating walk from i to j.

Dual of a statement: A statement for which we replace X by $\overline{X}$, $x(e)$ by $\lambda(e) - x(e)$, label $(+)$ by $(-)$, addition by subtraction, etc., from the initial statement.

Unscanned vertex: A leaf of the tree from which we haven't tried to grow the tree yet.

Auxiliary graph G^*: A subgraph of G with a specific construction used in the algorithm to find alternating walks.

Base of the tree: The vertex from which we start growing the tree.

Blossom: A cycle formed in growing the tree.

Pseudovertex: A vertex of the tree representing a blossom or a nesting of blossoms.

Simple vertex: A vertex of the tree which is not a pseudovertex.

Nesting of blossoms: A blossom such that one of its vertices (or more) is a pseudovertex itself.

Base of a blossom: The vertex closest to the base of the tree.

True base of a blossom: If the base of a blossom is not a pseudovertex, then it is the true base; otherwise the true base is inductively the true base of the base of the blossom.

Shrinking a blossom: Replacing the blossom and all of its vertices by a single pseudovertex representing all the vertices in the blossom.

Expanding a blossom: Replacing the pseudovertex back to an alternating path inside the blossom.

Fake loop: A loop in G^* which does not correspond to a loop in G.

Applications (See more details in Section 9).

Berge's substitute S: A complete bipartite graph used to replace each vertex of a graph G in order to transform an f-barrier problem into a matching problem. It composed of two independent sets of vertices: $V_1(S)$ has $(d - f(i))$ *internal vertices* and $V_2(S)$ has d *external vertices*.

Gabow's sparse substitute, S': A graph used to replace each vertex of a graph G in order to transform an f-barrier problem into a matching problem. It is more efficient than Berge's substitute S.

Decomposition of G: Set of k subgraphs $F_1, F_2, ..., F_k$ such that $E(F_1), E(F_2), ..., E(F_k)$ partition $E(G)$.

Edge colouring of G: A decomposition of $E(G)$ into matchings, called colour classes.

Colour class of G: A matching in the edge colouring of G.

Chromatic index of G, $\chi'(G)$: The minimum number of colour classes in any edge colouring of G.

$\triangle(G)$**:** The maximum degree of any vertex in G.

3. Basic results about network flows

A *network* is a directed graph $Q = (N(Q), A(Q)) = (N, A)$ with data and two special nodes: a source node s and a sink node t (Figure 2). In our case, we will consider three type of data: the flow, the capacity (or upper bound on the flow), and the lower bound on the flow. The *capacity function* u and the *lower bound function* l, where $u, l : A \to \mathbb{Z}$, respectively assign a fixed value to each arc $a \in A(Q)$ such that $l(a) \leq u(a)$. The *flow function* $x : A \to \mathbb{R}$ is required to satisfy $l(a) \leq x(a) \leq u(a)$ for all $a \in A$; it is called the

capacity constraint. Note that the lower bound function is sometimes omitted and that the flow is usually non negative, so $x : A \to \mathbb{R}^+$ such that $0 \leq x(a) \leq u(a)$. An arc $a \in A$ is *empty* if $x(a) = 0$ and it is *full* if $u(a) - x(a) = 0$. For all nodes $n \in N$, the inflow equals the outflow; this is called the *flow conservation* and can be written as

$$\sum_{i:(i,j)\in A} x(i,j) = \sum_{j:(i,j)\in A} x(i,j) \quad \forall i, j \neq s, t. \tag{1}$$

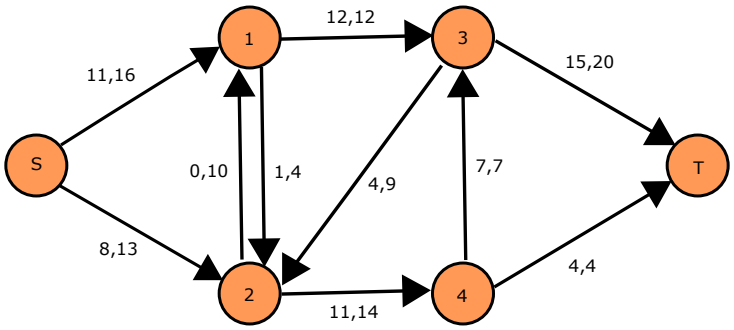


FIGURE 2. Network Q with flow x and capacity u (x, u).

The *size of the flow* is the net amount of flow leaving the source (equals to the net amount of flow entering the sink)and the *maximum flow* of a network is a flow of maximum size among all possible flows. The *network flow problem* consists in finding the maximum flow which can be pushed from s to t under the conditions imposed by the capacity constraint and the flow conservation.

A partition of the nodes of a directed graph G is a collection of pairwise disjoint subsets N_i of the nodes such that $\bigcup_i N_i = G(V)$. A cut (or an *st-cut*) is a partition of the nodes in two sets Y and Z such that the source s is in the set Y and the sink t is in Z. The *net flow across the cut* (Y, Z) is the difference between the flow from Y to Z and the flow from Z to Y. The *capacity of a cut* is the sum of the capacity of each arc going from Y to Z. The *minimum cut* of a directed graph is a cut which has the minimum capacity among all cuts. The *excess capacity of a cut* (Y, Z), denoted $t_{Y,Z}$, is the difference between the capacity of the cut and the size of the flow. In our investigation, no flow enters s so we can express $t_{Y,Z}$ as:

$$t_{Y,Z} = \sum_{\substack{i\in Y \\ j\in Z}} u(i,j) - \sum_{i\in Y} x(s,i).$$

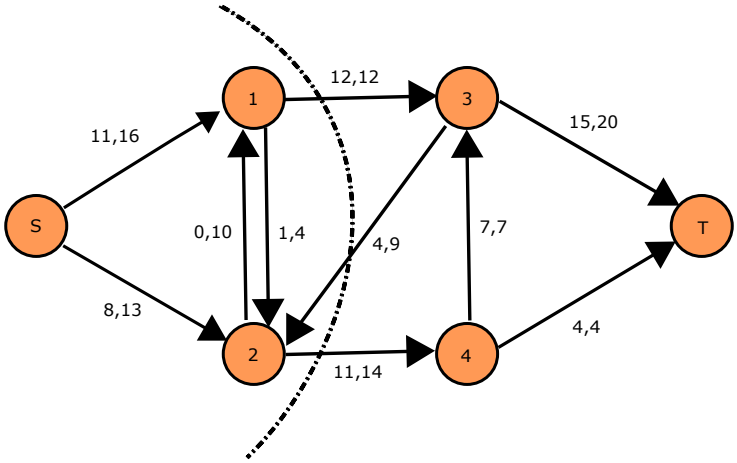


FIGURE 3. An st-cut where $Y = \{s, 1, 2\}$ and $Z = \{3, 4, t\}$.

The following Lemma 3.1 relates the size of the flow to the net flow across the cut and is useful to solve network flow problems.

Lemma 3.1. *(**Flow value Lemma**) Let f be any flow and (Y, Z) be any st-cut. Then, the size of the flow equals the net flow across the cut.*

Let us define some general terminology which will be important for the rest of this paper. In a general graph G, a *walk* is a sequence of vertices and edges $v_1, e_1, v_2, e_2, v_3, ..., e_{n-1}, v_n$ which may pass through the same edge or vertex more than once. The *length of a walk* corresponds to its number of edges. A *trail* is a walk that passes through every edge at most once. A *path* is a walk which passes through every vertex at most once. We say that a walk is *closed* if $v_0 = v_n$ and a *cycle* is a closed trail such that the only repeated vertex is $v_0 = v_n$. We say that a graph is *connected* if there is a path from any vertex to any other vertex in the graph. A *tree* is a connected graph without cycles. In the tree, if a vertex v has degree 1, it is called a *leaf*.

Given a network Q and a flow x, we define the *residual network* of Q, denoted Q', to be the network with capacity r (called *residual capacity*). A residual network is formed as follows: for each arc $a = (i, j) \in Q$, we have two arcs $a_1 = (i, j)$ and $a_2 = (j, i)$ in Q' where $r(a_1) = u(a) - x(a)$ and $r(a_2) = x(a)$. In a network flow, an *augmenting path* is a directed path inside the residual network $(s, a_1, n_1, a_2, ..., a_n, t)$ from s to t such that $r(a_i) > 0$ for all $i = 1, ..., n$. If there exists an augmenting path P then the size of the flow can be increased by $min(r(a_i))$ for all $i = \{1, 2, ..., n\}$ on the path P.

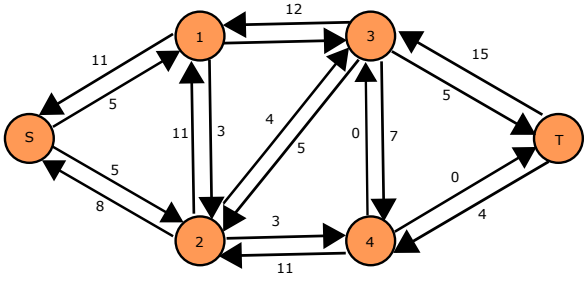


FIGURE 4. Residual network Q' of Q with flow x as shown in Figure 2.

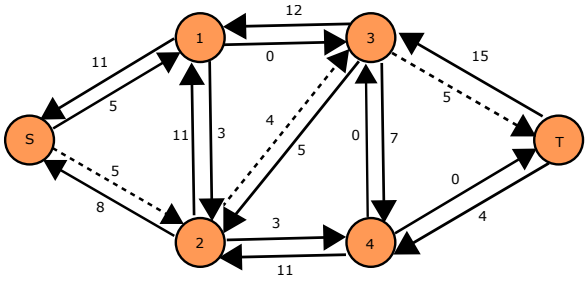


FIGURE 5. Augmenting path in Q' as shown in Figure 4.

Theorem 3.2. (Max-flow Min-cut Theorem, [7]) *For any network Q, the capacity of the minimum st-cut is equal to the size of the maximum flow from s to t.*

To prove the Max-flow Min-cut Theorem 3.2, we will prove that the following statements are equivalent:
(i) f is a maximum flow,
(ii) there is no augmenting path,
(iii) there exist a cut such that the capacity of this cut equals the size of the flow.

Proof. $(i) \Rightarrow (ii)$ We will prove $((\sim ii) \Rightarrow (\sim i))$ Suppose that there exists an augmenting path with respect to a flow f. Then the flow can be increased by the $min(r(a_i))$ where a_i are the arc on the path. Therefore f is not maximum.
$(ii) \Rightarrow (iii)$ Suppose that there is no augmenting path with respect to f. Consider the cut (Y, Z) such that Y consists of the vertices connected to s with no full forward arcs, and no backward empty arcs. Note that $s \in Y$ by definition, and $t \in Z$ since there is no augmenting path. Since backward edges are empty, the capacity of this cut is equal to the net flow across the cut which is equal to the size of the flow, by the Flow value Lemma.
$(iii) \Rightarrow (i)$ Suppose that there exists a cut (Y, Z) such that the capacity of the cut equals the size of the flow f. Let f' be another flow. Then, $f' \leq$ capacity of (Y, Z) = size of f. Therefore f is a maximum flow. □

We can mention the *Ford-Fulkerson algorithm* which is used to find the maximum flow of a network. Starting from any flow in a network, the algorithm allows to increases the initial flow step by step until it reaches its maximum size. It goes as follows:
(1) x is the initial flow
(2) Create the residual network of Q with respect to the flow x
(3) While there exists an augmenting path in the residual network, add this path to the flow increasing the size of the flow by the minimum residual capacity along this path.

For example, consider the network Q as in Figure 2 with initial flow x and its residual network Q' as in Figure 4. We showed in Figure 5 that we can find an augmenting path. Using

this path, we can then augment flow x by the minimum residual capacity along the path, which is 4, resulting in the flow given in Figure 6. Then we look for a new augmenting path in order to do the same thing. In this case, we can't find an augmenting path so the flow is now maximal in Q.

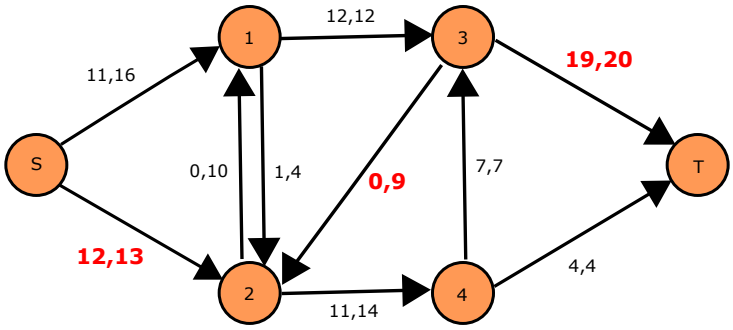


FIGURE 6. Example of the Ford-Fulkerson algorithm on network Q with flow x as shown in Figure 2.

One last fundamental network flow result which will be very important in our theory is the Integrality Theorem 3.3.

Theorem 3.3. (Integrality Theorem) *Let $Q = (N, A)$ be a directed graph for network flow problem. If all capacities are integers, then there exists a maximum flow x such that $x(a)$ is integer for all $a \in A$.*

Remark 3.4. *This idea of augmenting paths won't always work for non-integral capacity. An alternate search strategy for augmenting paths using breadth first search does work.*

4. f-FACTOR PROBLEM

A subgraph F of a general graph G can be defined as a vector $\overrightarrow{x} = (x(e) : e \in E(G))$ such that $0 \leq x(e) \leq \lambda(e)$ and $x(e) \in \mathbb{Z}$. If the requirement $x(e) \in \mathbb{Z}$ is dropped, we say that F is a *fractional subgraph*. In a network flow, $x(a)$ represents the flow, while in a graph, $x(e)$ represents the multiplicity of the edge e in the subgraph F. The adjacency matrix of a subgraph F of G always satisfies $0 \leq A_F \leq A_G$. For example, consider the general graph G on 10 vertices in Figure 7 and its subgraph F in Figure 8. We can express F as $\overrightarrow{x} = (x(e_1), x(e_2), ..., x(e_{16})) = (1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 2, 0, 1, 1)$.

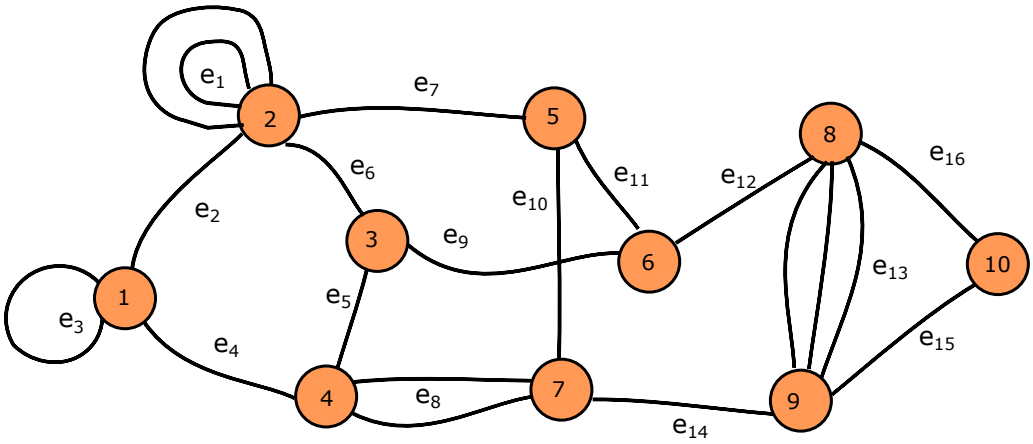


FIGURE 7. General graph G.

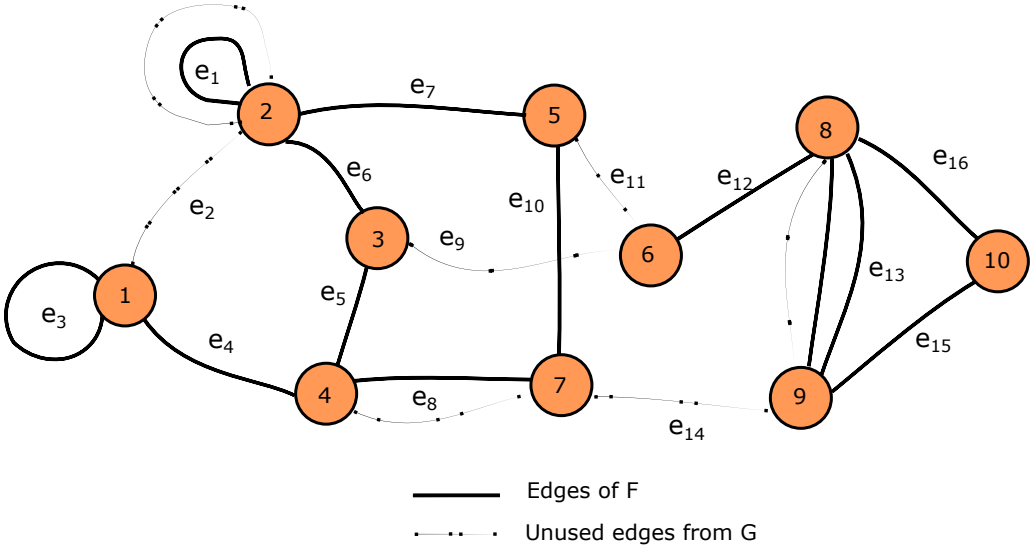


FIGURE 8. Subgraph F of G; an f-factor of G with $f(i) = \sum_{j \in V(G)} x(i,j)$.

Now consider the functions $f : V \to \mathbb{Z}^+$. A general graph G has an *f-factor* if there exists a subgraph F of G given by $\overrightarrow{x} = (x(e) : e \in E(G))$ with $x(e) \in \mathbb{Z}$ and such that for all $i \in V(G)$:

$$\sum_{j \in V(G)} x(i,j) = deg_F(i) = f(i). \tag{2}$$

For example, the subgraph F of G in Figure 8 is an f-factor if we define $f(i) = deg_F(i)$. If the requirement $x(e) \in \mathbb{Z}$ is dropped, then F is *a fractional f-factor*. As mentioned in the introduction, the adjacency matrix of an f-factor F of G corresponds to an integral symmetric matrix A with even entries on the diagonal, i^{th} row and column sums equal to

$f(i)$ for $i = 1, 2, ..., n$ and satisfying $0 \leq A_F \leq A_G$. For the subgraph F (Figure 8), we have the following matrix:

$$A_F = \begin{bmatrix} 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Note that the adjacency matrix of a fractional f-factor will have the same properties with the exceptions that entries off the diagonal may not be integers and entries on the diagonal may not be even integers.

Given a general graph G and an integer function f, the well-known f-factor problem asks whether an f-factor of G exists. Tutte's f-factor Theorem 4.1 is a famous theorem in graph decomposition which gives a characterization for the existence of f-factors in a given general graph G. Recall that $e_g(S, T)$ counts all edges (i, j) such that $i \in S$ and $j \in T$, according to multiplicity.

Theorem 4.1. (Tutte's f-factor Theorem, [12]) *Let G be a general graph and $f : V(G) \to \mathbb{Z}^+$ be a function. Then G has a f-factor if and only if for all disjoint subsets S and T of $V(G)$,*

$$\sum_{i \in S} f(i) + \sum_{i \in T} (deg_G(i) - f(i)) - e_G(S, T) - q(S, T) \geq 0, \tag{3}$$

where $q(S, T)$ denotes the number of components C of $G - (S \cup T)$ such that

$$\sum_{i \in V(C)} f(i) + e_G(C, T) \equiv 1 \pmod 2. \tag{4}$$

For convenience, we will call any components of $G - (S \cup T)$ satisfying equation (4) an *f-odd component* of G.

The objective of this Section is to provide a detailed description of the theory given by Anstee to solve the f-factor problem. We will start by showing how he combines a network flow approach and a graph theory approach and the advantages of doing so.

4.1. **From graphs to network flows and back again.** We start with a general graph G on n vertices for which we want to solve the f-factor problem. We associate two other graphs to G: a directed graph $D(G)$ such that $A_G = A_{D(G)}$ (Figure 1), and a constructed

network flow graph $Q(G)$ (Figure 10).

The network flow graph $Q(G) = (N, A)$ is constructed on $2n + 2$ nodes as follows: a source s, a sink t, and nodes $R_1, R_2, ..., R_n, S_1, S_2, ..., S_n$. There are arcs from s to R_i and from S_i to t such that $u(s, R_i) = u(S_i, t) = f(i)$ for $i, = 1, 2, ..., n$ and arcs from R_i to S_j with capacity $u(i, j) = \lambda(i, j)$. If there is an edge (i, j) in G, it is translated by two arcs (R_i, S_j) and (R_j, S_i) in $D(G)$. Figure 10 represents the network flow graph $Q(G)$ of the general graph G shown in Figure 9. All arcs between R_i and S_j all have capacity 1, except $u(R_1, S_5) = u(R_5, S_1) = 2$ since $\lambda(1, 5) = 2$.

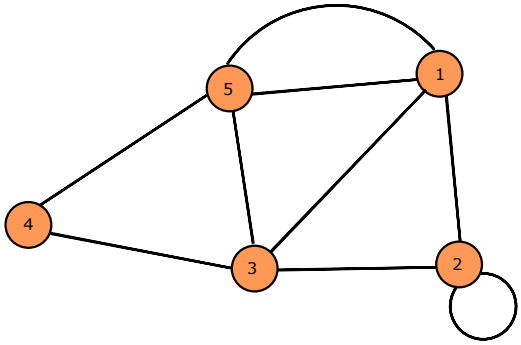


FIGURE 9. General graph G.

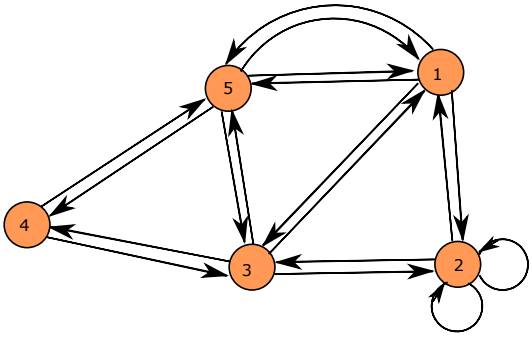


FIGURE 10. Directed graph $D(G)$.

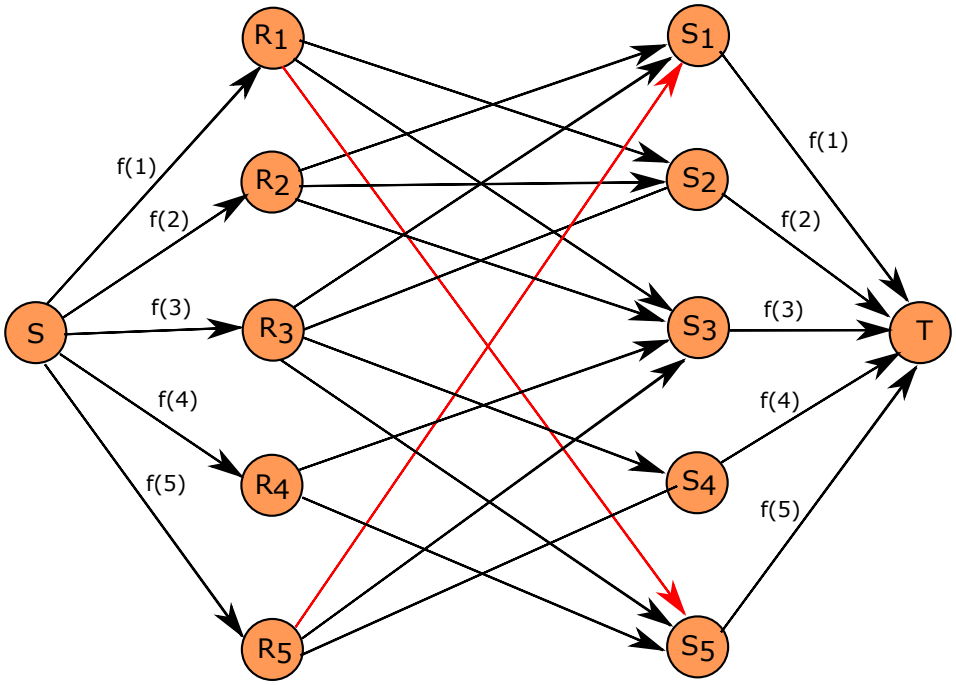


FIGURE 11. Network $Q(G)$.

The first step is to determine if $D(G)$ has a directed f-factor B. A *directed f-factor* can be defined in term of its adjacency matrix as an integral matrix $A_B = (b_{ij})$ with i^{th} row and column sums equal to $f(i)$ for $i = 1, 2, ..., n$ and satisfying $0 \leq A_B \leq A_G$. The key observation and the reason why it is interesting for us to use a network flow approach is that finding a directed f-factor in $D(G)$ is equivalent to finding an integral maximum flow of size $f(1) + f(2) + ... + f(n)$ in $Q(G)$. Indeed, if there exists a directed f-factor B with adjacency matrix $A_B = (b_{ij})$, then there is an integral maximum flow of size $f(1) + f(2) + ... + f(n)$ such $x(R_i, S_j) = b_{ij}$. Now assume that there exists an integral maximum flow of size $f(1) + f(2) + ... + f(n)$ in $Q(G)$. Consider the matrix $A_B = (b_{ij})$ where $b_{ij} = x(R_i, S_j)$. We can make two observations based on the flow conservation property:

$$(1) \quad \sum_{j=1}^{n} x(R_i, S_j) = f(i) \quad \forall i = 1, ..., n;$$

$$(2) \quad \sum_{i=1}^{n} x(R_i, S_j) = f(j) \quad \forall j = 1, ..., n.$$

So A_B has i^{th} row and column sums equal to $f(i)$ while $0 \leq A_B \leq A_G$; therefore, B is a directed f-factor of G.

PUT SOMEWHERE $x(i, j) = 1/2(x(R_i, S_j) + x(R_j, S_i))$.

Remark 4.2. *In order to claim that there exists an integral maximum flow of size $f(1) + f(2) + ... + f(n)$, and therefore that G has a directed f-factor, it is sufficient to find a maximum flow of this size, by the Integrality Theorem 3.3*

The following theorem reveals the significance of the existence of a directed f-factor in G for the f-factor problem.

Theorem 4.3. (Anstee, [2]**)** *Let G and f be given. Then G has a fractional f-factor if and only if $D(G)$ has a directed f-factor.*

Proof. ($\Rightarrow$) Assume that G has a fractional f-factor. By definition, equation (2) holds. Translating into the network flow graph $Q(G)$ for the network flow problem, we get $x(s, R_i) = f(i)$, $x(S_j, t) = f(j)$, and $x(R_i, S_j) = x(R_j, S_i) = x(i, j)$ implying that

$$\sum_{i:(R_i,S_j)\in A} x(R_i, S_j) = f(i); \qquad \sum_{j:(R_i,S_j)\in A} x(R_i, S_j) = f(j)$$

This flow x is a maximum flow of this size $f(1) + f(2) + ... + f(n)$. By the Integrality Theorem 3.3 , there exists an integral maximum flow x' of this size and therefore $D(G)$ has a directed f-factor.
($\Leftarrow$) Assume that $D(G)$ has a directed f-factor B. Consider the general graph corresponding to the matrix $(A_B + A_B^T)/2$. It may not have integral entries off the diagonal or even entries on

the diagonal but it is a symmetric matrix with i^{th} row and column sums $f(i)$ for $i = 1, 2, ..., n$ and it satisfies $0 \leq (A_B + A_B^T)/2 \leq A_G$. Therefore, it is a fractional f-factor of G. □

Lemma 4.4. *If G has a directed f-factor, then G has a half integral fractional f-factor.*

Proof. Assume that $D(G)$ has a directed f-factor B, then $(A_B + A_B^T) \in \mathbb{Z}$ and so $(A_B + A_B^T)/2$ is a fractional f-factor of G such that $2x(e) \in \mathbb{Z}$ for all $e \in E(G)$. □

It is sometimes useful to view the matrix operation $\frac{1}{2}(A_B + A_B^T)$ as a flow operation $x(i, j) = \frac{1}{2}(x(R_i, S_j) + x(R_j, S_i))$.

Using a network flow approach allows us to use well known methods such as the Ford-Fulkerson algorithm to quickly determine whether a graph G has a half-integer fractional f-factor. On one hand, if $D(G)$ has no directed f-factor, then, by Theorem 4.3, G doesn't have a fractional f-factor and it cannot have an f-factor. On the other hand, if $D(G)$ has a directed f-factor, then, by Theorem 4.3, G has fractional f-factor B. The next phase of Anstee's theory is to figure out if B can be transformed into an f-factor. We provide an algorithm that either succeeds in transforming the half integral fractional f-factor into an f-factor or finds an *f-barrier*, a partition of $V(G)$ which upon examination reveals that G has no f-factor.

4.2. **Existence of a directed f-factor.** The following theorem about the existence of a directed f-factor is an adapted version of the Max-flow Min-cut Theorem 3.2.

Theorem 4.5. (Anstee, [2]**)** *Let G be a general graph. There is a directed f-factor in G if and only if*

$$\sum_{i \in T} f(i) \leq \sum_{j \in S} f(i) + \sum_{\substack{i \in T \\ j \in (T \cup U)}} \lambda(i, j) \tag{5}$$

for all partitions S, T, U of $\{1, 2, ...n\}$.

Proof. By Remark 4.2, it is sufficient to show that there is an integral maximum flow of size $f(1) + f(2) + ... + f(n)$ if and only if equation (5) holds for all partitions S, T, U of $1, 2, ..., n$. Equation (5) can be deduced from Theorem 3.2. Let $I, J \subseteq \{1, 2, ..., n\}$. Observe that for any arbitrary cut between $I \cup \bar{J}$ and $\bar{I} \cup J$ such that the source $s \in I \cup \bar{J}$ and the sink $t \in \bar{I} \cup J$, the capacity of the cut equals

$$\sum_{i \in \bar{I}} f(i) + \sum_{\substack{i \in I \\ j \in J}} u(i, j) + \sum_{j \in \bar{J}} f(i). \tag{6}$$

By Max-flow Min-cut Theorem 3.2 , we obtain

$$\sum_{i \in \bar{I}} f(i) + \sum_{\substack{i \in I \\ j \in J}} u(i, j) + \sum_{j \in \bar{J}} f(i) \geq \sum_{i=1}^{n} f(i). \tag{7}$$

We now show that the condition can be restricted to a more specific cuts, where $I \subseteq J$. Note that:

$$\sum_{i\in\bar{I}} f(i) = \sum_{i\in\bar{I}\cup\bar{J}} f(i) - \sum_{i\in I\setminus J} f(i)$$

$$\sum_{i\in\bar{J}} f(i) = \sum_{i\in\bar{J}\setminus I} f(i) + \sum_{i\in I\setminus J} f(i)$$

$$\sum_{\substack{i\in I\\ j\in J}} u(i,j) = \sum_{\substack{i\in I\cap J\\ j\in I\cup J}} u(i,j) + \sum_{\substack{i\in I\cap J\\ j\in I\setminus J}} u(i,j) + \sum_{\substack{i\in I\setminus J\\ j\in J}} u(i,j)$$

Moreover, since $u(i,j) = u(j,i)$, we can write

$$\sum_{\substack{i\in I\cap J\\ j\in I\setminus J}} u(i,j) + \sum_{\substack{i\in I\setminus J\\ j\in J}} u(i,j) = \sum_{\substack{i\in I\setminus J\\ j\in J\setminus I}} u(i,j).$$

Therefore:

$$\sum_{i\in\bar{I}} f(i) + \sum_{\substack{i\in I\\ j\in J}} u(i,j) + \sum_{i\in\bar{J}} f(i)$$

$$= \sum_{i\in\bar{I}\cup\bar{J}} f(i) - \sum_{i\in I\setminus J} f(i) + \sum_{\substack{i\in I\cap J\\ j\in I\cup J}} u(i,j) + \sum_{\substack{i\in I\setminus J\\ j\in J\setminus I}} u(i,j) + \sum_{i\in\bar{J}\setminus I} f(i) + \sum_{i\in I\setminus J} f(i)$$

$$= \sum_{i\in\bar{I}\cup\bar{J}} f(i) + \sum_{\substack{i\in I\cap J\\ j\in I\cup J}} u(i,j) + \sum_{\substack{i\in I\setminus J\\ j\in J\setminus I}} u(i,j) + \sum_{i\in\bar{J}\setminus I} f(i).$$

Finally note that $I \barcap J = \bar{I}\cup\bar{J}$, $I \barcap J = \bar{J}\setminus I$, and $u(i,j) \geq 0$. Thus, if the following inequality holds,

$$\sum_{i\in I\barcap J} f(i) + \sum_{\substack{i\in I\cap J\\ j\in I\cup J}} u(i,j) + \sum_{i\in I\barcap J} f(i) \geq \sum_{i=1}^{n} f(i), \tag{8}$$

so does equation (7). Therefore, in order to be true for all subsets $I, J \subseteq \{1,2,...,n\}$, it is sufficient to be true for all I, J such that $I \subseteq J$. In order to complete the proof, we simply have to set $S = \bar{J}$, $T = I$, and $U = J \setminus I$. □

We can relate this theorem to the characterization given in Tutte's f-factor Theorem 4.1. Consider the partition S, T, U of $V(G)$ such that $\bar{S} = U \cup T$ and $\bar{T} = S \cup U$. It is worth noting that in Tutte's Theorem 4.1 the set U does not explicitly appear and is simply seen as $\overline{S\cup T}$; otherwise, Tutte uses the same partition of $V(G)$. First, assume that G has no directed f-factor. Then, by Theorem 4.5, there exists a partition S, T, U of $\{1,2,...,n\}$ such that equation (5) doesn't hold. By rearranging this inequality, we have:

$$\sum_{j\in S} f(i) - \sum_{i\in T} f(i) + \sum_{\substack{i\in T\\ j\in(\bar{S})}} \lambda(ij) \leq 0. \tag{9}$$

Notice that in equation (3),

$$\sum_{i\in T} deg_G(i) - e_G(S,T) = \sum_{i\in T} deg_{(G-S)}(i) = \sum_{\substack{i\in T \\ j\in(T\cup U)}} \lambda(ij). \tag{10}$$

Since $q(S,T) \geq 0$, if equation (9) holds, equation (3) doesn't hold and there is no f-factor. Therefore, as desired, if there is no directed f-factor our conclusion that there cannot be an f-factor agrees with the characterization given in Tutte's f-factor Theorem 4.1.

We can also compute $t_{T,\bar{S}}$ for an st-cut formed by the source s, R_i for $i \in T$, and S_j for $j \in S$ as in Figure 12. Recall that $t_{T,\bar{S}}$ equals the capacity of the cut minus the size of the flow, and so we have:

$$t_{T,\bar{S}} = \left(\sum_{i\in\bar{T}} f(i) + \sum_{\substack{i\in T \\ j\in\bar{S}}} \lambda(i,j) + \sum_{i\in S} f(i) \right) - \sum_{i} f(i)$$

So

$$t_{T,\bar{S}} = \sum_{j\in S} f(i) + \sum_{\substack{i\in T \\ j\in\bar{S}}} \lambda(ij) - \sum_{i\in T} f(i). \tag{11}$$

For any given directed f-factor, $B = (b_{ij})$ where $b_{ij} = x(i,j)$, another expression of $t_{T,\bar{S}}$ will be useful in proving that when $t_{T,\bar{S}}$ is relatively small for some pair $(T,\overline{S})$ we cannot find an f-factor. By Lemma 3.1, $t_{T,\bar{S}}$ can also be defined as the capacity of the cut minus the net flow across the cut:

$$t_{T,\bar{S}} = \sum_{i\in\overline{T}} f(i) + \sum_{\substack{i\in T \\ j\in\overline{S}}} \lambda(i,j) + \sum_{i\in S} f(i) - (\sum_{i\in\overline{T}} f(i) + \sum_{\substack{i\in T \\ j\in\overline{S}}} b_{ij} + \sum_{i\in S} f(i) - \sum_{\substack{i\in\overline{T} \\ j\in S}} b_{ij}).$$

After cancellation, we obtain the following equality:

$$t_{T,\bar{S}} = \sum_{\substack{i\in\bar{T} \\ j\in S}} b_{ij} + \sum_{\substack{i\in T \\ j\in\bar{S}}} (\lambda(i,j) - b_{ij}). \tag{12}$$

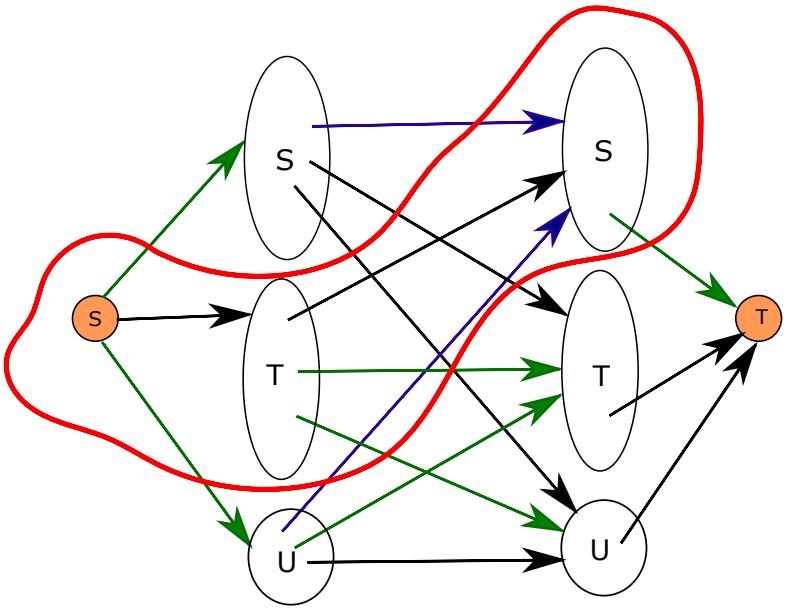


FIGURE 12. Partition S, T, U with st-cut $(T, \overline{S})$.

Now we need to examine the case where $D(G)$ has a directed f-factor and describe how to either find an f-factor or an f-barrier.

4.3. **From fractional f-factor to integral f-factor.** Assume that $D(G)$ has a directed f-factor B. By Theorem 4.3 , $A_F = (B + B^T)/2 = (a_{ij})$ is the adjacency matrix of a half integral fractional f-factor F of G such that $a_{ij} = x(e)$ where $e = (i, j)$ in F and $2x(e) \in \mathbb{Z}$. The purpose of this section is to transform F so that as many as possible fractional entries off the diagonal become integer entries and as many as possible odd integer entries on the diagonal become even integer entries. This transformation has to be done while preserving $0 \leq A_F \leq A_G$.

Consider the general graph H on the vertices of G such that there is an edge $e = (i, j)$ when $x(i, j)$ is not an integer and a loop $e = (i, i)$ when $x(i, i)$ is an odd integer. So H is a subgraph of F consisting only of the edges e such that $x(e)$ needs to be transformed in order to obtain an f-factor. For each entry $x(e)$ where $e \in E(H)$, we want to add or subtract $1/2$ (1 for loops) to get integer entry in A_H (even entry for loops) while preserving the degree of the vertices of H. When such transformation is successfully executed on some edges $e \in H$, we say that those edges can be *removed* from H and the respective entries a_{ij} of A_H then become the entries a_{ij} of A_F. The ultimate goal is to remove all the edges from H in order to have the desired f-factor in G.

The following Lemma 4.6 partitions the edges of any general graph into three sets depending the type of walk it belongs to. In order to transform the edges of H, we will use this

partition and proceed with a specific strategy for each of these different type of walks. We will first give a method to remove the even length closed trails in and then to remove the vertex disjoint odd cycles. Note that loops are odd cycles of length 1.

Lemma 4.6. *The edges of any general graph G can be partitioned into: (i) even length closed trails; (ii) paths joining two vertices of odd degree in the graph G; (iii) vertex disjoint odd cycles.*

Remark 4.7. *Since row and column sums are integers in A_F, the degree of each vertex is an integer and therefore must have an even number of half edges at each vertex. It implies that every vertex in H has even degree. Hence, H doesn't have any path joining two vertices of odd degree.*

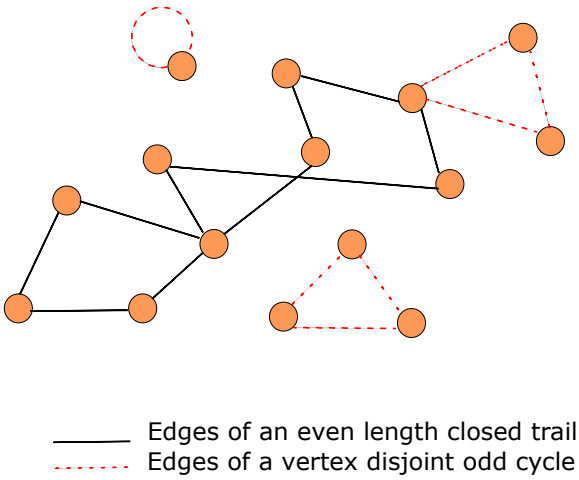


FIGURE 13. Structure of the general graph H:
even length closed trails and vertex disjoint odd cycles.

Removing even length closed trails in H

Consider any even length closed trails in H. We can transform A_F by alternatively add $1/2$ and subtract $1/2$ from the adjacent edges of the trail (1 for a loop) in order to remove it from H as in Figure 14. Therefore, no matter what the fractional f-factor F is, we can always remove every even length closed trails of H. Now only the vertex disjoint odd cycles remain (a loop is an odd cycle of length 1). Note that if H doesn't have any vertex disjoint odd cycles, we have transformed our fractional f-factor F into a desired f-factor; we have solved the f-factor problem.

Proposition 4.8. *If H has only even length closed trails (no vertex disjoint odd cycles), then the directed f-factor B can always be transformed into an f-factor.*

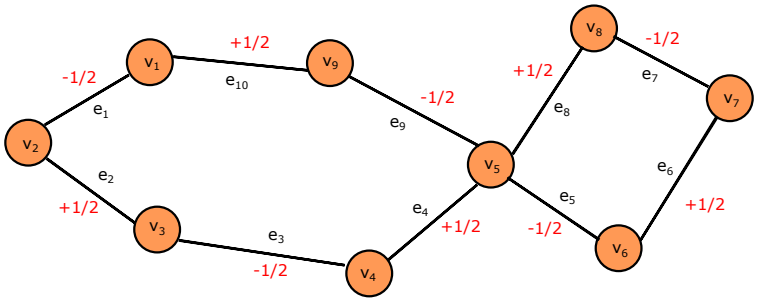


FIGURE 14. Removing an even length closed trail of H.

Removing the vertex disjoint odd cycles in H

Remark 4.9. *The number of vertex disjoint odd cycles in H is even if and only if $\sum f(i)$ is an even integer.*

Proof. Note that for $\sum f(i)$ to be an even integer is an easy necessary condition for an f-factor to exist since for any general graph we have $\sum deg(i) = 2|E|$. Hence, if there is an f-factor, $\sum f(i) = \sum deg(i) = 2|E|$ is always even. Now consider the following edge partition of F: $E = \{e \in F : e \notin E(H)\}$ and $E' = \{e' \in F : e' \in E(H)\}$. By construction, each edge in E counts for 2 in the total sum of the degree of the vertices of F, $\sum deg_F(i)$, while each edge in E' counts for 1. Hence $\sum deg_{F-H}(i)$ is always even, and so the parity of $\sum deg_F(i)$ depends on whether $\sum_{i \in F} deg_H(i)$ is odd or even. However, since each edge of E' counts for 1 in $\sum deg_F(i)$, it also counts for 1 in $\sum deg_H(i)$. Therefore, $\sum deg_H(i)$ corresponds to the number of edges in H. By consequence, $\sum f(i)$ is even if and only if the number of edges in H is even; that is, if and only if the number of vertex disjoint odd cycles in H is even.

□

If two vertex disjoint odd cycles are joined by an edge e, we can remove those two cycles from H since either $x(e) \geq 1$ so we can subtract 1 from $x(e)$, or $\lambda(e) - x(e) \geq 1$ so we can add 1 to $x(e)$. We consider such edges as alternating walks of length 1, and the vertex disjoint odd cycles can be removed by pair as shown in Figure 15:

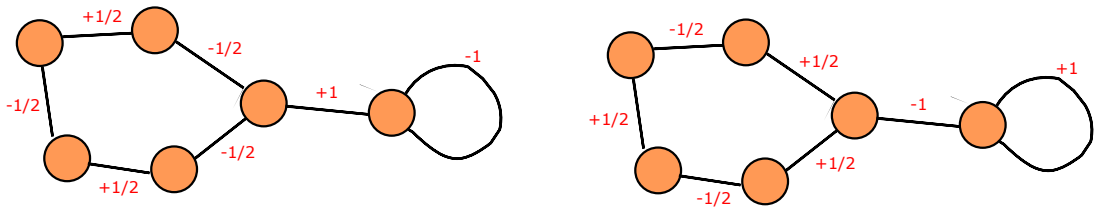


FIGURE 15. Two cases of removing 2 odd cycles of H when they are joined by an edge.

A general graph has the *extended odd cycle property* if any pair of vertex disjoint odd cycles (including loops) is joined by an edge e. The following theorem is a simplified version

of the f-factor theorem given by Anstee using the extended odd cycle property and the observation made in Remark 4.9.

Theorem 4.10. (Anstee, [2]) *Let G have the extended odd cycle property. Then G has an f-factor if and only if G has a directed f-factor and $f(1) + f(2) + ... + f(n)$ is even.*

Let $C_1, C_2, ..., C_t$ be the vertex disjoint odd cycles in H. As opposed to the even length closed trails, removing all the C_i's in H is not always possible. It involves the complex concept of alternating walks, which is an adaptation of the alternating paths used in matching problems (see Section 5.1). We define an *alternating walk* as a walk $v_1, e_1, v_2, e_2, v_3, ..., e_{n-1}, v_n$ such that we can add 1 to entry $x(e_1)$, subtract 1 from entry $x(e_2)$, etc. (or 2 if e_i is a loop) while preserving $0 \leq A_F \leq A_G$; or the same series starting with subtraction. We can add 1 to $x(i,j)$ if $\lambda(i,j) - x(i,j) \geq 1$ in G, and we can subtract 1 from $x(i,j)$ if $x(i,j) \geq 1$ in G. In order to remove a vertex disjoint odd cycles of H, say C_1, we need to find an alternating walk from a vertex of C_1 to a vertex of any other C_i still in H. Therefore, we remove the C_i's in pair only, and so if H contains at least one vertex disjoint odd cycles and $\sum f(i)$ is odd, we cannot remove all the C_i's of H, by Remark 4.9. We will show that it implies that an f-factor cannot exist.

In our investigation, alternating walks give labels to vertices of the general graph G for which we want to solve the f-factor problem. To do so, we give alternating walks a direction based on the vertex from which we start the walk, e_i goes from v_i to v_{i+1}. A walk ends in addition (subtraction) if we are adding to (subtracting from) e_{n-1}. We also say that a vertex j is *reachable* from a vertex i if there is an alternating walk from i to j in G. The labelling of the vertices of G is done based on this concept of reachability. Given a starting vertex v_1, a vertex v of G is labelled $(+)$ if it is reachable from v_1 by an alternating walk ending in addition, while it is labelled $(-)$ if it is reachable from v_1 by an alternating walk ending in subtraction. Note that each vertex can have at most 2 labels. This labelling provides sufficient information about whether a vertex is reachable from v_1 and how a walk can be continued (by adding, subtraction, or both) from a given vertex if needed.

We now return to our main problem and generalize the idea of the extended odd cycle property. The goal is to find an alternating walk $v_1, e_1, v_2, e_2, v_3, ..., e_{n-1}, v_n$ joining 2 odd cycles of H with edges $e_i \notin H$. If such walk exists, both odd cycles can be removed from H as shown in Figure 16. If we manage to remove all odd cycles from H, we will have transformed A_F into the desired f-factor. Section 5 describes the algorithm to find an alternating walk between two disjoint vertex odd cycles. If there is an odd cycle C_k of H which cannot be joined to another cycle C_j $(k \neq j)$ by an alternating walk, then we claim that there is an f-barrier; and thus no f-factor in G. This last claim will be proved in section 6.

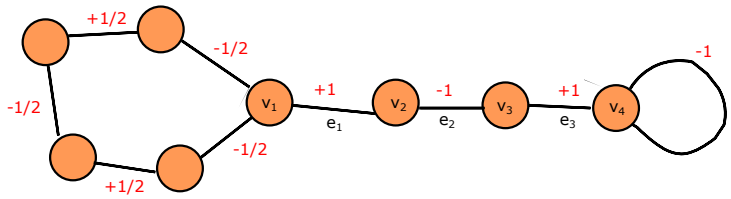


FIGURE 16. Removing 2 odd cycles of H with an alternating walk.

We are ready to state Anstee's alternative version of Tutte's f-factor Theorem 4.1. The proof follows from the steps we have discussed and will be completed in Sections 5 and 6:
(1) Is there a directed f-factor? If no, then there is no f-factor. If yes, continue.
(2) Symmetrize the directed f-factor.
(3) Remove all even length closed trails of H.
(4) Can we remove all vertex disjoints odd cycles of H? If no, there is an f-barrier. If yes, there is an f-factor.

The following statement explicitly describes this process in a version of Theorem 4.1.

Theorem 4.11. (Anstee, [2]) *Let G be a general graph on n vertices. Then G has an f-factor if and only if there is a directed f-factor and there is no f-barrier.*

5. Finding an alternating walk

5.1. **Analogy with Edmond's algorithm for matching problem.** Before giving the details about the algorithm to find an alternating walk, we consider an analogy between the use of alternating walks in an f-factor problem and the use of augmenting paths in a matching problem.

Given a graph G, a *matching* M is a set of edges without common vertices in G and the *size of the matching* M is the number of edges $e \in M$. The *matching problem* consists in finding a matching of maximum size in G, called a *maximum matching*. Given a matching M in a general graph G, an *alternating path* is a path whose edges are alternatively in and out the matching M. Such an alternating path is called an *augmenting path* if it satisfies the additional condition that its two end vertices aren't incident to any edges of M. In other words, it is an odd length path $v_1, e_1, v_2, e_2, ..., e_{n-1}, v_n$ such that $e_{odd} \in M$, $e_{even} \notin M$, and $deg_M(v_1) = deg_M(v_n) = 0$. as shown in Figure 17.

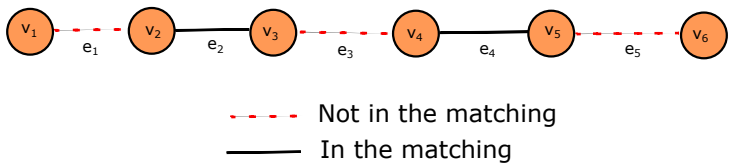


FIGURE 17. Augmenting path P.

If such a path exists, it is possible to modify the initial matching M by letting $e_{even} \in M$ and $e_{odd} \notin M$; we then obtain a new matching M' in G with one more edge than in M as in Figure 18). Hence, using augmenting paths we were able to increase the size of the initial matching M by one. Claude Berge proves the following theorem which indicates the central use of augmenting paths:

Theorem 5.1. (Berge, [6]) *A matching is maximum if and only if there is no augmenting path.*

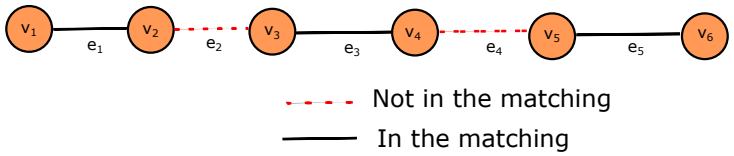


FIGURE 18. Using the augmenting path P from Figure 17 to increase the size of the matching M by one.

Edmond's algorithm is designed to find augmenting paths in a graph. If the algorithm fails at returning an augmenting path P in a graph G, then, by Theorem 5.1, the matching is maximum in G. In the case of an f-factor problem in a general graph instead of a matching problem, alternating walks take the role of augmenting paths. We will show that if the algorithm designed for finding alternating walks between two vertex disjoint odd cycles of H (see Section 5.2) fails at finding such walk for a cycle $C_i \in H$, then there exists an f-barrier.

Edmond's algorithm is based on the concept of blossoms. We will adapt this concept to alternating walks. The algorithm for finding an alternating walk in G roughly consists in growing a tree from a given vertex $v \in G$, called the *base of the tree*, following a specific strategy. A *blossom* is a cycle formed during the tree growing process. The *base of a blossom* is defined as the vertex which is the closest to the base of the tree. *Shrinking* a blossom refers to replacing the blossom and all of its vertices by a new single vertex b, called a *pseudovertex*, such that b will represent all the vertices in the blossom (Figure 20). *Expanding* a blossom refers to replacing the pseudovertex b by an alternating path existing inside the pseudovertex (Figure 20). We define a *simple vertex* as a vertex which is not a pseudovertex. Note that we can have a blossom such that one of its vertices (or more) is a pseudovertex itself. In this case, when we shrink the blossom we have a pseudovertex containing one or more psudovertices; this is called a *nesting of blossoms*. Using induction, we may have a pseudovertex containing pseudovertices which themselves contain pseudovertices, etc. Therefore, it is useful to define the *true base* of a blossom as the following. If the base of a blossom is not a pseudovertex, then it is the true base since it is not contained in another blossom; otherwise the true base is inductively the true base of the base of the blossom.

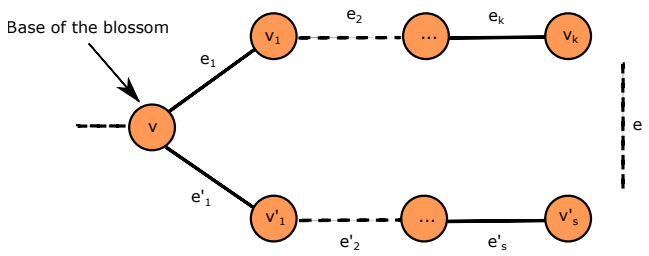


FIGURE 19. Formation of a blossom during the tree growing process.

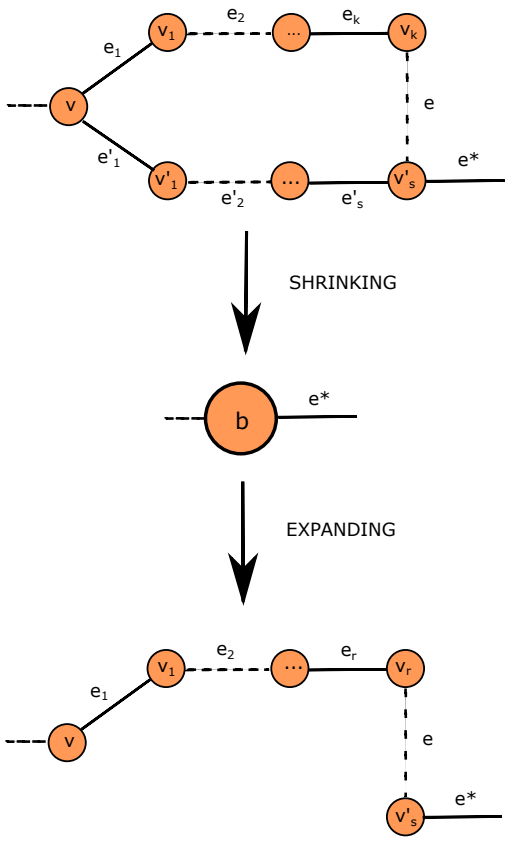


FIGURE 20. Shrinking and expanding a blossom.

Remark 5.2. *In an alternating walk for an f-factor, an edge needs only to be used twice.*

By Berge, in a matching problem, the necessity to use an edge twice in an augmenting path is irrelevant since an edge is either matched or not matched. In other words, $x(e)$ is 0 or 1 (we cannot have $x(e) \geq 2$), and so edge multiplicity does not matter. On the other hand, for f-factor problems, edge multiplicity does matter, making the concept of alternating walks more complex than augmenting paths.

Consider the example illustrated in Figure 21 to understand why it is necessary to allow edges to be used twice in certain circumstances. In Figure 21, let $x(v_1, v_2) \geq 2$ and let the walk have direction as given starting at v. Assume that we did not allow the edge (v_1, v_2) to be used more than once in the walk. Then we wouldn't be able reach w by an alternating walk starting from v using the given edges. Since $e_9 \in \overline{X}$, we need to enter v_1 by subtraction in order to use e_9 from v_1 to w in the alternating walk. This can only be done by going around the cycle and re-entering v_1 by subtraction through e_8. Considering only the edges given, we would fail at giving w label $(+)$ unless we use $e = (v_1, v_2)$ twice. Therefore we would compromise the possibilities for cycles C_i's in H to be removed. However, there are two situations in which using an edges twice may be necessary; when it is used either (1) in opposite direction with the same operation as in Figure 21 or (2) in the same direction in opposite operations. Those are the two cases where using an edge twice will give a vertex a label which it didn't have before. Figure 22 shows how using an edge twice in opposite directions with opposite operations is unnecessary to find alternating walks, since it doesn't give a new label to v_1, and therefore, doesn't allow to leave v_1 with a different operation. In our example, we can directly leave vertex v_1 by e_8 and don't have to go through $e_2, e_3, ..., e_7$; the alternating walk can be reduced to v, e_1, v_1, e_8, w without affecting the label of the vertices. Finally, note that using an edge twice in the same direction with the same operation is obviously unnecessary in an alternating walk for the same reason.

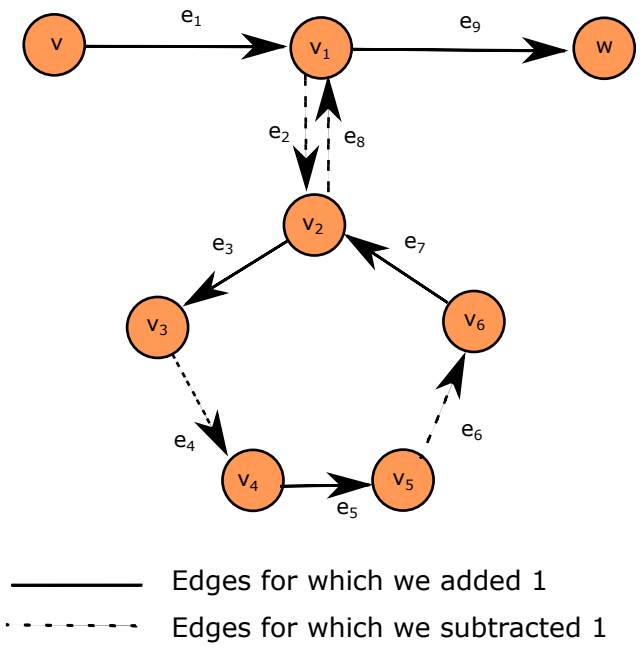


FIGURE 21. Using an edge twice in opposite direction with the same operation.

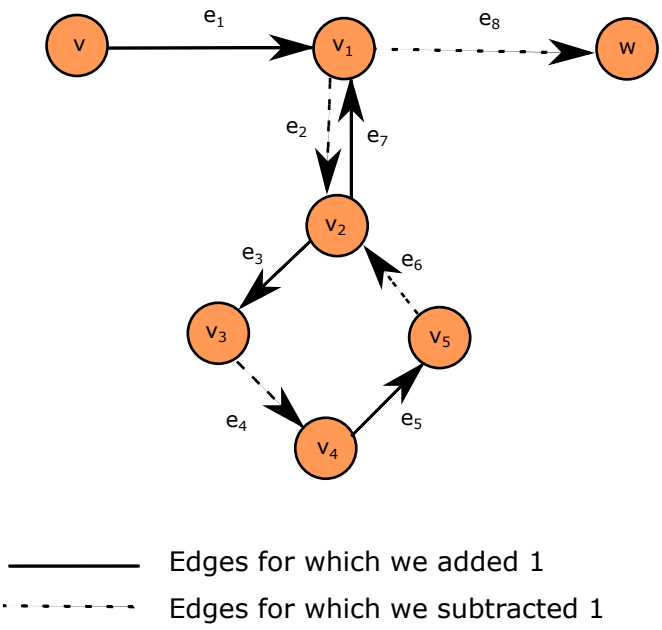


FIGURE 22. Using an edge twice in opposite direction with different operation.

As a consequence of this last remark, we will create a general auxiliary graph G^* on the vertices of G such that it contains the exact number of edges which can potentially be needed in an alternating walk. Edges of $G*$ will be divided into two classes, X and $\overline{X}$. Firstly, for $e \in G$ such that e is not a loop. Entries in G such that $x(e) \geq 1$ will be represented by edges in X: 1 edge $e = (i, j)$ in G^* if $x(e) = 1$ and 2 edges $e = (i, j)$ in G^* if $x(e) \geq 2$. Similarly, entries in G such that $\lambda(e) - x(e) \geq 1$ will be represented by edges in $\overline{X}$: 1 edge $e = (i, j)$ in G^* if $\lambda(e) - x(e) = 1$ and 2 edges $e = (i, j)$ in G^* if $\lambda(e) - x(e) \geq 2$. Secondly, for $e \in G$ such that e is a loop: a loop $e \in X$ is added to G^* if $x(e) \geq 2$, while a loop $e \in \overline{X}$ is added to G^* if $\lambda(e) - x(e) \geq 2$. Therefore, there will be at most 4 edges between each pair of vertices; at most 2 edges in X and at most 2 in $\overline{X}$. In all figures, we will represent edges in X as dot edges (- - -) and edges in $\overline{X}$ as full edges (——).

5.2. **Algorithm.**

Overview of the algorithm

Anstee provides an algorithm for finding an alternating walk between an odd cycle of H, say C_1, and any other odd cycle C_k $(k > 1)$ [2]. The general idea is to grow a directed tree, vertex by vertex, starting from a base vertex r representing C_1, such that a directed path in this tree will correspond to an alternating walk in G. The vertices of the tree correspond to the vertices of G or pseudovertices created during the algorithm. The edges used to grow the tree are edges of G^*. We will use the concept of *duality* in the algorithm. The *dual* of a

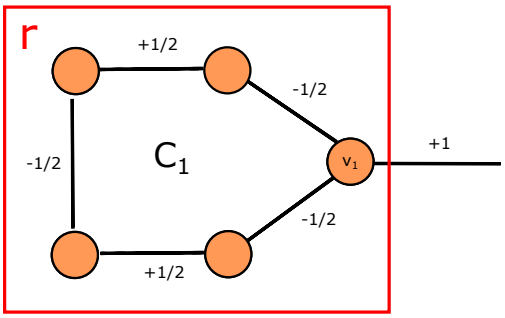
statement refers to the substitution of X by $\overline{X}$, $x(e)$ by $\lambda(e) - x(e)$, label $(+)$ by label $(-)$, addition by subtraction, etc. We also define a vertex v of the tree to be *unscanned* if v is a leaf of the tree from which we haven't tried to grow the tree yet. Note that vertices of the tree can be pseudovertices.

The tree will grow in the following way: if a vertex v has label $(-)$, then we will grow the tree out from v with every edges $e \in \overline{X}$ in G^* which is incident to v. The same is done for the dual. Note that a vertex may have both labels, $(+)$ and $(-)$. In this case, we can grow the tree using edges from both classes X and $\overline{X}$.

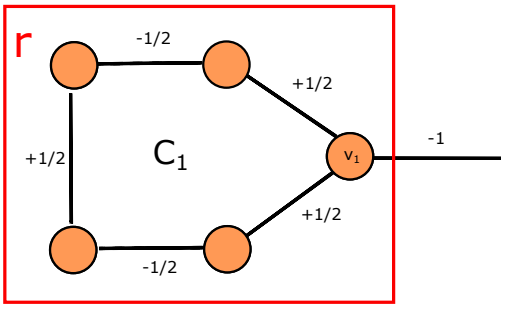
We will store information from the tree using the notation: $p(w) = v$, meaning v is the predecessor of w in the tree. If the tree grows from v to w using an edge in X, then w is given label $(+)$, and if the edge used is in $\overline{X}$, then w is given label $(-)$. In either case, $p(w) = v$. An edge can be used to grow the tree only if it is in G^*; once it is used to grow the tree and give a vertex its label, it is deleted from G^* so that a same edge won't be used more than once.

When the tree grows, cycles may arise. They are the blossoms defined earlier. We will shrink the vertices of the blossom to a pseudovertex which will be given both labels, $(+)$ and $(-)$. When a blossom is shrunken to a pseudovertex, the edges and vertices of the blossom are deleted from the tree (Figure 20). However, we need to keep track of those edges and their original ends independently so that we can expand the blossoms later when restoring the alternating walk in G corresponding to the directed path in the tree (Figure 20).

Finally, it is helpful to study the case of the vertex r, which represents the odd cycle C_1 in the tree, and therefore is defined as the base of the tree. Although we are shrinking C_1 to the unique vertex r, it is different from the other pseudovertices since it has no predecessor. Assume that we have found an alternating walk from C_1 to C_k $(k > 1)$. The first edge of the walk leaving r, say $e_1 = (r, v)$, may either be in X or in $\overline{X}$ since r has both labels. Depending on the class of e_1 and its original ends, $v_1 \in H$ and $v_2 \notin H$, we will be able to find an alternating walk starting at v_1 and leaving C_1 with the appropriate operation to remove H such as in Figures 23 and 24. Note also that in the tree, r may itself be hidden in a pseudovertex if a blossom is created with r as a true base. In that case, the first vertex of the alternating walk, v_1, may be inside a nesting of blossoms.

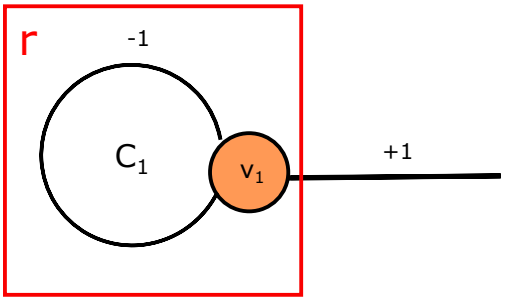


(A) Leaving C_1 with an edge in $\overline{X}$.

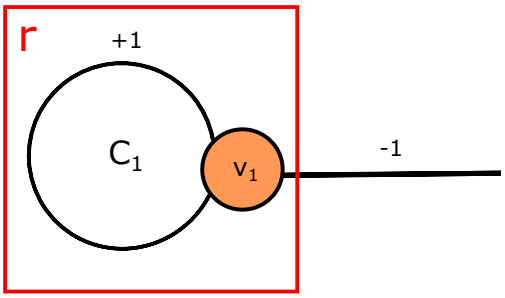


(B) Leaving C_1 with an edge in X.

FIGURE 23. Leaving the odd cycle C_1



(A) Leaving C_1 with an edge in $\overline{X}$.



(B) Leaving C_1 with an edge in X.

FIGURE 24. Special case: leaving the the loop C_1

Pseudo-code of the algorithm

We can now provide the pseudo-code of the algorithm given by Anstee to find an alternating walk from C_1 to any vertex of another odd cycle C_k $(k > 1)$ [2]:

Step 1 (**Shrink** C_1): Consider the odd cycle C_1 and shrink its vertices to a single vertex r. Add r to the list of unscanned vertices with labels $(+)$ and $(-)$. At this point, r is the only vertex in the tree and thus the only vertex in the list of unscanned vertices.

Step 2 (**Create general graph** G^*): Create the general graph G^* on the vertices of G as follows. We divide the edges of G^* into two classes X and $\overline{X}$. For all edges e of G not a loop, put $\min(x(e), 2)$ copies of e in G^* and for all loops e in G, put $\min(x(e)/2, 1)$ copies. Those edges belong to the class X; they are the entries of G from which we can subtract 1 (or 2) while preserving $0 \leq A_F \leq A_G$. On the other hand, for all edges e of G not a loop put $\min(\lambda(e) - x(e), 2)$ copies of e in G^* and for all loops e in G put $\min(\lambda - x(e)/2, 1)$ copies. Those edges belong to $\overline{X}$; they are the edges of G to which we can add 1 (or 2) while

keeping $0 \leq A_F \leq A_G$.

Step 3 (**Pick an unscanned vertex**): If the list of unscanned vertices is empty, then STOP; there is an f-barrier since we are unable to grow the tree any further and we haven't reached a vertex from an odd cycle C_k ($k > 1$) (see results in Section 6). Otherwise select any vertex u from the list of unscanned vertices.

Step 4: If u has label $(+)$, do Step 5 and 6.

Step 5 (**Search for non loop blossom**): If there is an edge $(u, w) \in X$ such that w is already in the tree and has label $(+)$, use it to form a blossom in the tree. Then, shrink the blossom to a pseudovertex. This pseudovertex has label $(+)$ and $(-)$ since each vertex of the blossom can be reached by an alternating walk ending in addition and another alternating walk ending in subtraction through the base of the blossom (see Lemma 5.8). The pseudovertex is added to the list of unscanned vertices. The vertices u and w are deleted from the list of unscanned vertices. All other vertices of the blossom will have already been scanned and deleted from the list of unscanned vertices. Return to Step 3.
Otherwise, if no such edge exists, proceed to Step 6 to grow the tree.

Step 6 (**Grow tree and search for loop blossom**): For all vertices $w \in G^*$ not in the tree do the following. If there is an edge $(u, w) \in X$, then add w to the tree such that $p(w) = u$. Since we leave u by an edge from X, we are reaching w by an alternating walk ending in subtraction and so w gets label $(-)$. Delete this used edge (u, w) from G^*. If $w \in C_k$, then STOP; we have found an alternating walk from C_1 to C_k ($k > 1$). Otherwise, if there is a loop in $\overline{X}$ at w, then w also has label $(+)$ and becomes the pseudovertex of the blossom generated by the loop. In other words, w is a pseudovertex representing only one vertex, which is itself. Add w to the list of unscanned vertices regardless. Now we have grown the tree in every possible direction from u using edges in X.

Step 7: If u has label $(-)$, do the duals of Step 5 and 6.

Step 8: The vertex u is scanned. Delete it from the list of unscanned vertices. Return to Step 3.

- end of the algorithm -

A choice of the algorithm is that the loops appear and are shrunken into pseudovertices in step 6 but it could as well be done in step 5 when we search for blossoms.

Structure of the tree

At any step of the algorithm we will have a tree (see Theorem 5.6) which has the same structure as in Figure 25. Two types of vertices may appear in the tree: (1) simple vertices, which are vertices with only one label, (2) pseudovertices, which are vertices representing a

blossom or a nesting of blossoms and have both labels. Both type of vertices can be preceded and followed by any of the two types and their number of children will be determined by the general graph G on which we solve the problem.

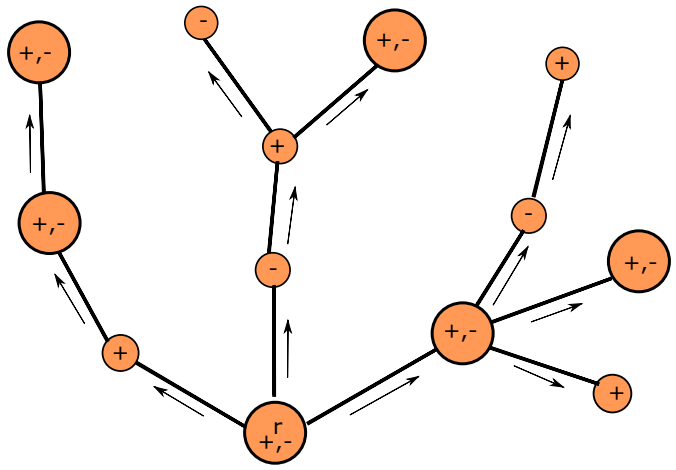


FIGURE 25. Tree structure

Recall that r represents C_1 or the pseudovertex containing it. Since r has both labels, the tree can grow out with edges in X and $\overline{X}$. We now study in more details the structure of the tree by describing separately what may happen when we scan a simple vertex and a pseudovertex.

Case 1: Scanning a simple vertex u with label $(+)$.
Since u has label $(+)$, we can only consider edges in X to leave from u as shown in Figure 26. In view of our results about loops in Section 6, we can assume there is no loop at u in X. First, we search for a blossom using an edge in X joining u to a vertex in the tree with label $(+)$ (see Figure 19). If such edge exists, a blossom is formed and we won't grow the tree from u any further since the vertices of the blossom are replaced by a new pseudovertex and u is assumed to be scanned.
If no blossom is found, then we grow the tree from u to every vertex $w \in G$ where w is not in the tree and such that $(u, w) \in X$. For instance, consider vertex v in Figure 26. On one hand, the edge $(v, w_1) \in X$ gives w_1 label $(-)$ as seen it would appear in the general tree in Figure 25. On the other hand, the edge $(v, w_2) \in X$ first gives w_2 label $(-)$. Perhaps a blossom or nesting of blossoms is later formed with w_2 as its true base and w_2 is then replaced by a pseudovertex, representing all the vertices of the blossom, and only the pseudovertex appears in the tree. The dual is presented in Figure 27.

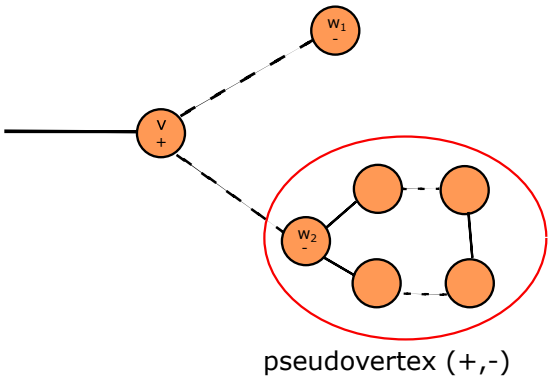


FIGURE 26. Growing the tree from a simple vertex with label (+).

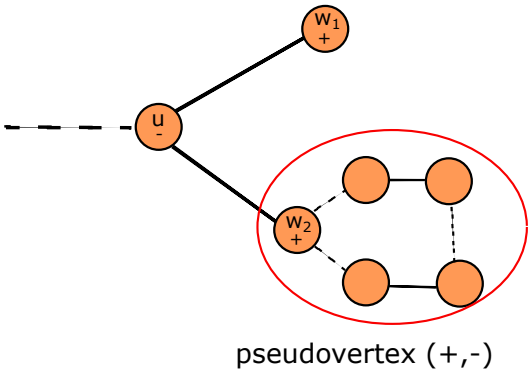


FIGURE 27. Growing the tree from a simple vertex with label (−).

Case 2: Scanning a pseudovertex u with both labels, (+) and (−).
Since u has both labels, we can consider edges in X and $\overline{X}$ to leave u. By choice, we always start by using the edges in X before the edges in $\overline{X}$. If we can form a blossom using an edge in X, it creates a nesting of blossoms which is directly replaced in the tree by a new pseudovertex w, as shown in Figure 28. We stop here; u is considered to be scanned and so it is deleted from the list of unscanned vertices. If no blossom is formed, we grow the tree from u, using every edge $(u, w) \in X$, where $w \in G$ is not in the tree.

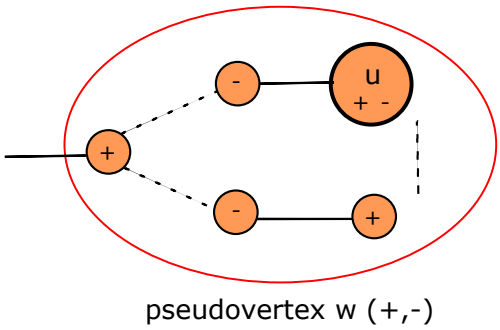


FIGURE 28. Nesting of blossoms when scanning pseudovertex u.

Then, we look for blossoms again, but this time, using edges in $\overline{X}$. In addition to the blossoms and nesting of blossoms we have seen, a special type of blossom may appear as in Figure 29: $e = (u, w) \in X$ has already been used when growing the tree from v using edges in X and now $e' = (u, w) \in \overline{X}$ is being used. We are joining two vertices from the tree which both have label (−) using an edge in $\overline{X}$, which is consistent to the description in Step 6 when we search for a blossom. It is a special type of blossom as it only consists of two vertices, but it still forms a nesting of blossoms which is then replaced by a pseudovertex. If no blossom is formed, we grow the tree from u now using edges in $\overline{X}$ as in Figure 30.

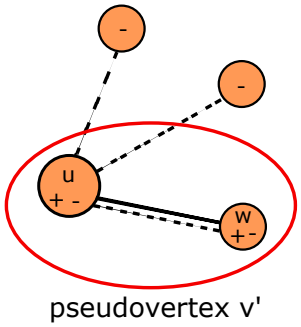


FIGURE 29. Special type of blossom.

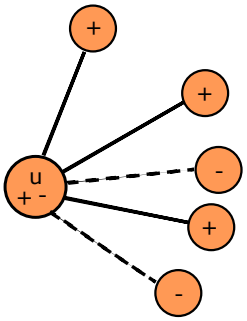


FIGURE 30. Growing tree from a pseudovertex.

5.3. **Proof of the algorithm.** To complete our work, we need to prove that the algorithm terminates (Lemma 5.3), that it creates a tree (Lemma 5.6), and that it finds an alternating walk between two odd cycles of H if one exists (Theorem 5.11).

Lemma 5.3. *Let n be the number of vertices in G, the general graph for which we want to solve the f-factor problem. The algorithm terminates after at most n-(# of vertices in C_1) applications of step 3.*

Proof. There are two instances in which the algorithm can terminate: in Step 6 if there exists an alternating walk from C_1 to C_k or Step 3 if the list of unscanned vertices is empty. We need to prove that the list of unscanned vertices will ultimately be empty if we never reach a vertex from C_k $(k > 1)$. Observe that n is finite and that once a vertex is scanned and deleted from the list of unscanned vertices, it doesn't reappear in the list later. Every time a vertex u is being scanned, we end up with two cases:
Case 1: We can find a blossom containing u as a vertex. Let $e = (u, w)$ be the edge that closes the blossom. Note that w has to be in the list of unscanned vertices since it is in the tree but it has not been scanned yet, otherwise e would have already been used from w to u. A new pseudovertex z is created and added to the list of unscanned vertices while u and w are deleted from the list. Although we add a new vertex to the total number n of vertices, we will be deleting 2 vertices from G.
Case 2: We grow the tree out from u. At the end of step 7, u is deleted from the list of unscanned vertices while other vertices from G may be added.
As a result, regardless of whether we find a blossom or grow the tree, the number of vertices which can appear in the list is reduced by 1 every time we return to Step 3. The list of unscanned vertices will be empty after at most n - (# of vertices in C_1) scanning operations. □

Remark 5.4. *Note that each vertex of G can get at most two labels, $(+)$ and $(-)$. Each time we grow the tree to a new vertex, it gets at least one new label. Therefore, an easy upper bound on the number of relabelling to terminate the algorithm is $2n$.*

The next two lemmas prove how nesting of blossoms appear in the graph constructed from our algorithm and that this graph is indeed a tree.

Lemma 5.5. *Let B, B' be two blossoms with pseudovertex b and b' respectively. If B and B' have a vertex in common then we will have one of the two pseudovertices, say b, contained in the other one, b'.*

Proof. Let B and B' be two blossoms with b and b' as their respective pseudovertex. Note that in the algorithm, blossoms are created one at a time so we can assume, without loss of generality, that B is formed first. Then all the vertices of B are represented by its pseudovertex b. Since B and B' have at least one vertex in common, say u, then u is now represented by b. Consequently, when B' is formed, it is shrunk to its pseudovertex b' which represents all the vertices of B' including the pseudovertex b. Therefore, the pseudovertex b is now contained in b' and does not appear in the shrunken tree when the algorithm terminates. □

Lemma 5.6. *The algorithm creates a tree.*

Proof. Recall that a tree is a connected graph with no cycles (loops being considered as cycles). We need to check that the graph generated by the algorithm growing from vertex r is a tree and therefore will not contain any cycles when the algorithm terminates. It is sufficient to prove that every cycle generated by the algorithm is a blossom and becomes a pseudovertex.
First consider the case where the cycle is a loop. In the algorithm, loops can only be created at step 6. However, the vertex where the loop appears, say w, is directly replaced by a pseudovertex. Therefore, the graph created will not contain any loops.
Now, let's consider the case where we have a cycle of length greater than 1. Such cycle appears when we join two vertices u and v such that they already both belong to the tree; this only happens when we search for blossoms in step 5. Hence, since blossoms are reduced to pseudovertices, no cycle formed by the algorithm will survive. □

Now we prove some results about alternating walks in our tree. Those results will be needed to prove that the algorithm works in Theorem 5.11.

Lemma 5.7. *Let u be a pseudovertex with true base b, where $b \neq r$. Every alternating walk from b to $v \in u$ starts with the same operation.*

Proof. Let u be a pseudovertex with true base b, where $b \neq r$. By contradiction, assume that there exist an alternating walk from b to $v \in u$ starting in addition and another alternating walk from b to $v' \in u$ starting in subtraction. It implies that b has both labels and therefore is itself a pseudovertex. This is a contradiction since b is the true base of u. If u is the pseudovertex representing r this lemma will not hold since r does not have a defined true base. □

Lemma 5.8. *Let u be a pseudovertex with true base b, where $b \neq r$, and v be any vertex in u. In the initial general graph G^*, there exists at least two alternating walks from b to v: one ending in addition and another one ending in subtraction.*

Proof. Consider the blossom B represented by the pseudovertex u with true base b. We will use induction on the steps of the algorithm. It is trivially true at step 1. It is sufficient

to consider step 5 when u is formed. Following the algorithm, the blossom B is formed by joining two directed paths in the tree which have b as a common vertex. Let the two paths P_1 and P_2 respectively be $b, e_1, u_1, e_2, u_2, ..., e_l, u_l$ and $b, e'_1, v_1, e'_2, v_2, ..., e'_k, v_k$ and let $e \in X$ be the edge joining u_l to v_k as in Figure 31. By induction hypothesis, there exists an alternating walk from b to u_l and another one from b to v_k such that they both end in addition. Now consider an arbitrary vertex $u_z \neq b$ of u. Without loss of generality, let u_z be in P_1. We will be looking at the two walks W_1 and W_2 respectively defined by $b, e_1, u_1, e_2, u_2, ..., e_z, u_z$ and $b, e'_1, v_1, e'_2, v_2, ..., e'_k, v_k, e, v_l, e_l, ...e_{z+1}, u_z$. The same argument holds for the dual.

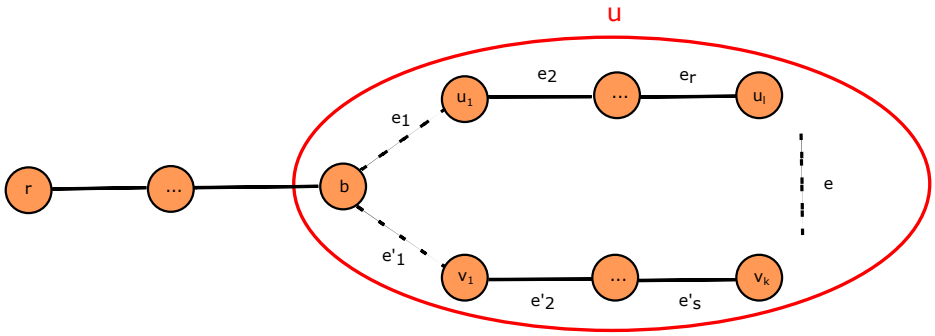


FIGURE 31. Nesting of blossoms when scanning pseudovertex u ($b \neq r$).

Firstly, assume that every vertex in the pseudovertex u is simple. Then, by the algorithm, $b, e_1, u_1, e_2, u_2, ..., e_l, u_l$ and $b, e'_1, v_1, e'_2, v_2, ..., e'_k, v_k$ are two alternating walks. Moreover, since u_l and v_k have label $(+)$ and can only have one label, both alternating walks given above must end in addition. The walk W_1, defined by $b, e_1, u_1, e_2, u_2, ..., e_z, u_z$, is obviously an alternating walk since it is the first part of the given alternating walk from b to u_l. Without loss of generality assume that $e_z \in X$, so W_1 ends in subtraction. Now, let's look at walk W_2. Since $b, e'_1, v_1, e'_2, v_2, ..., e'_k, v_k$ is an alternating walk ending in addition and $e \in X$, $b, e'_1, v_1, e'_2, v_2, ..., e'_t, v_t, e, u_n$ is an alternating walk ending in addition. Moreover, $b, e_1, u_1, e_2, u_2, ..., e_l, u_l$ is an alternating walk itself, with $e_l \in \overline{X}, e_{l-1} \in X, e_{l-2} \in \overline{X}$,...,$e_{z+1} \in X$. Those edges can thus be added in this specific order at the end of the alternating walk $b, e'_1, v_1, e'_2, v_2, ..., e'_k, v_k, e, u_l$ to obtain the desired alternating walk W_2 ending in subtraction. So we have two alternating walks in u from b to u_r: W_1 ending in addition and W_2 ending in subtraction. The dual holds.

Now assume that one of the vertices in the pseudovertex u is itself a pseudovertex, say u_s. Without loss of generality, let u_s be in W_1 to give $b, e_1, u_1, e_2, u_2, ..., e_s, u_s, e_{s+1}..., e_z, u_z$. Observe that e_s hits u_s at its true base, say b_s, and e_{s+1} hits u_s at w_s, any vertex of G in the blossom represented by u_s. By induction hypothesis there is an alternating walk from b to b_s and, by Lemma 5.8, we can choose to extend this alternating walk from b_s to w_s such that it starts and ends with the appropriate operation to continue the alternating walk at e_{s+1}. In other words, we can expand the pseudvertex u_s and replace it by a particular alternating walk in W_1. This induction can be done at each pseudovertex of the walks W_1 and W_2. Note

that by induction hypothesis, by expanding the pseudovertices in W_1 we have an alternating walk from b to u_z. Without loss of generality, assume that it ends in addition. For W_2, expand all the pseudovertices of the walk $u_r, e_{r+1}, ..., e_n, u_n$ in the same manner to get an alternating walk with only simple vertices. Add the expanded version in the opposite order to the alternating walk from b to v_k joining v_k to u_l with e. Similarly to the case with simple vertices, we obtain the second alternating walk from b to u_z such that it ends in subtraction. The dual holds.

Finally, consider the case where $v_z = b$ and $b \neq r$. Then by Lemma 5.7, we know that e_1 and e'_1 belong to the same class, say X. Since b is a true base, it initially is a simple vertex so it can only have label $(+)$, and only can leave with vertex in X when we grow the tree before forming the blossom. However, we know that v_1 is reachable from b by an alternating walk ending in addition, and e'_1 cannot have been used. We can therefore extend this alternating walk by adding the edge $e'_1 = (v_1, b)$. Therefore, b is now reachable by an alternating walk starting from b and ending in subtraction, and thus, b also obtain both labels $(+)$ and $(-)$. Consequently, b can also be left using both type of edges x and $\overline{X}$ as all the other vertices of the blossom. If $b = r = C_1$, then studying the blossom to show that we can leave r using both class of vertices is irrelevant since r originally has both labels already. □

Lemma 5.8 proves that it is reasonable to replace a blossom by a pseudovertex and giving it both labels since every vertex can be reach by an alternatinng walk ending in the operation of our convenience. In other words, the tree can be grown by using both type edges from any vertex inside the pseudovertices. Hence, in the tree, it is simpler to have one unique pseudovertex. Now that we have explored the structure inside the pseudovertices, we are ready to prove that the algorithm accomplishes what we want: if there is an alternating walk between C_1 and another odd cycle C_k in G, we will find a directed path from C_1 to a vertex of C_k in the tree.

Lemma 5.9. *Let $v \in V(G)$ such that $v \notin C_1$. If v has label $(+)$, then there exists an alternating walk from C_1 to v ending in addition. The dual holds.*

Proof. Assume that $v \notin C_1$. Since v has label $(+)$, it belongs to the tree and so there is a directed path from r to v, where r is the vertex representing C_1, the true base of the base of the tree. Let $r, e_1, v_1, e_2, v_2, ..., e_k, v$ be a specific directed path in the tree. If there is no pseudovertex, then this path directly corresponds to the alternating walk in G from r to v ending in addition, using the exact same vertices. Now, assume that the path contain a pseudovertex, say v_l. Then, by Lemma 5.8, we know that there exists an alternating walk ending in the appropriate operation from the true base of the blossom hitting by e_{l-1} to the vertex $w_t \in v_s$ of G which is hit by e_l. That is, if $e_l \in X$, we choose the alternating walk ending in addition, while if $e_l \in \overline{X}$ we choose the one ending in subtraction. We can then replace v_l by the appropriate alternating walk and repeat this process for all pseudovertices on the path. Once every pseudovertex is expanded and replaced by the appropriate alternating walk, we obtain an alternating walk in G from r to v ending in

addition. Note that r and v might themselves be inside pseudovertices and that r needs to be replaced by the vertex of C_1 corresponding to the original end of the edge e_1. □

Lemma 5.10. *Let $v \in V(G)$ such that $v \notin C_1$. If there exists an alternating walk in G from r to v in G ending in addition, then v has label $(+)$ when the algorithm terminates at Step 3. The dual holds.*

Proof. Let the alternating walk W in G from r to v ending in addition be $v_1, e_1, v_2, e_2, v_3, ..., e_n, v$. We want to show by contradiction that v will obtain the label $(+)$ if the algorithm terminates at Step 3. We say that a vertex u in G has the appropriate label if it indeed corresponds to the fact that u is reachable from r by addition if it has label is $(+)$ and by subtraction if it has label $(-)$. Assume that v_p and all previous vertices in W have the appropriate label. Without loss of generality, say that v_p has label $(+)$, implying that $e_p \in X$ in W. Therefore, by the assumption we made, v_{p+1} is reachable by an alternating walk ending in subtraction and should have label $(-)$. Suppose, by contradiction, that v_{p+1} does not have its appropriate label $(-)$. Then, v_{p+1} does not have any label or it has label $(+)$.

First, assume that v_{p+1} does not have any label. Note that when the algorithm terminates at Step 3, every vertex of the tree has been scanned. Hence, v_p has been scanned. If $e_p = (v_p, v_{p+1}) \in X$ is still in G^* when we grow the tree from v_p, it would be used to give v_{p+1} label $(-)$. This is a contradiction. Therefore, e_p has been used and deleted from G^*.

We can now assume that $e_p \in X$ has already been used to grow the tree, but v_{p+1} still doesn't have label $(-)$. Thus, the only way that e_p may have been used was in growing the tree is from v_{p+1} to v_p giving v_p label $(-)$. Thus v_{p+1} has to have label $(+)$ and v_p is therefore inside a pseudovertex, say $v_{p'}$, since it has both labels $(+)$ and $(-)$. We get $p(v'_p) = v_{p+1}$ in the tree. Note that $v_p \notin C_1$ as the vertex representing C_1 does not have a predecessor. Moreover, v_p has to be the true base of $v_{p'}$ since v_{p+1} only has label $(+)$ and $p(v'_p) = v_{p+1}$.

Since $v_{p'} \notin C_1$, there is a vertex $v_q \in W$ such that v_q is not in $v_{p'}$, but $v_{q+1}, v_{q+2}, ...,$ and v_p are in $v_{p'}$. By assumption, since it precedes v_p in W, v_q has its appropriate label, say $(+)$, and so $e_q \in X$. Assume that e_q has not been used. Since v_{q+1} belongs to the pseudovertex $v_{p'}$, it has both label $(+)$ and $(-)$, and therefore e_q would be used in Step 5 to create a blossom. However, this is a contradiction since, by assumption, v_q does not belong to the same pseudovertex as v_{q+1}. Therefore, the edge e_q has been used in the tree.

If e_q has been used in the tree to give v_{q+1} label $(+)$, then $p(v'_p) = v_q$, which is a contradiction since $p(v'_p) = v_{p+1}$. Note that, by construction, we cannot have $v_q = v_{p+1}$ since v_q precedes v_p in W while v_{p+1} comes after v_p, and v_{p+1} cannot be in a blossom. The only case in which we could have $v_q = v_{p+1}$ is if it was the vertex of an edge being used twice, implying that it would be inside a blossom.

Therefore, e_q has been used to give v_q label $(-)$. We then get that v_q is now itself the base of another pseudovertex, say v'_q, and $p(v_{q'}) = v_{p'}$ in the tree. If we repeat this process by looking at the vertex $v_z \notin v'_q$ in W such that $v_{z+1}, v_{z+2}, ...,$ and v_q are in v'_q, we will find that v_z is the base of a pseudovertex, $v_{z'}$ such that $p(v_{z'}) = v_{q'}$. By induction, we will eventually have to either hit v_1 in W or the pseudovertex containing it in the tree. But then, by repeating this process, we would conclude that v_1 is in a pseudovertex which has a

predecessor as well. This is our final contradiction (Figure 32). Therefore, v_{p+1} has to have its appropriate label and, by induction, v will have the appropriate label S. □

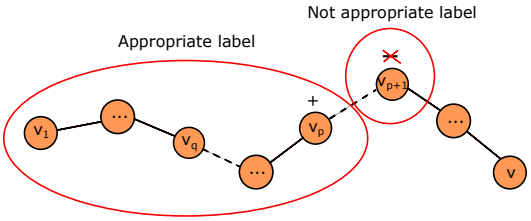


FIGURE 32. The alternating walk in G.

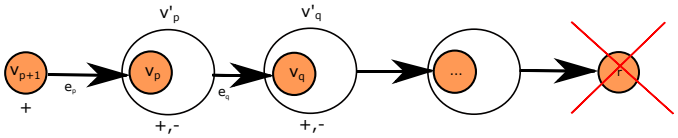


FIGURE 33. Contradiction in the structure of the tree.

From those two Lemmas 5.9 and 5.10, we can deduce Theorem 5.11 to prove that the algorithm works:

Theorem 5.11. *There is an alternating walk from C_1 to any vertex in C_k ($k > 1$) if and only if there is a directed path in the tree from the root r, a pseudovertex of C_1, to any vertex of C_k.*

In order to remove every vertex disjoint odd cycle of H, we need to find $t/2$ alternating walks, where t is the number of vertex disjoint odd cycles. If we are able to do so, we have an f-factor. Recall that if t is odd, by Remark 4.9, we know that G cannot have an f-factor without using the algorithm. However, as desired, the algorithm will claim the same result as we will be unable to remove all vertex disjoint odd cycles of H since we remove them in pairs. There will always be a cycle C_i for which the algorithm terminates at Step 3. In the next section, we will prove that when there is an odd cycle of H for which we failed at finding an alternating walk joining to another vertex disjoint odd cycle still in H - the algorithm terminates at Step 3 - we indeed have an f-barrier.

6. f-BARRIER

Let H consists of the disjoint odd cycles $C_1, C_2, ..., C_t$. Assume that there is an odd cycle in H, say C_1, for which there is no alternating walk joining one of its vertices to any cycle C_k ($k > 1$). We then claim that there is no f-factor in G. The goal of this section is to define the f-barrier, the partition of $V(G)$ which contradicts Tutte's characterization, equation (3).

According to Anstee, we can look at the partition S, T, U of $V(G)$ defined as follows. Let S be the set of vertices reachable from C_1 by an alternating walk ending in addition only (label $(+)$). Let T be the set of vertices reachable from C_1 by an alternating walk ending in subtraction only (label $(-)$). Let U be the set of the remaining vertices of G and decompose U into two subsets: W and L. Let W be the set of vertices reachable from C_1 by an alternating walk ending in addition and by an alternating walk ending in subtraction (labels $(+)$ and $(-)$). Finally, let L be the set of vertices not reachable from C_1 (no label). By assumption, there is no alternating walk between C_1 and C_k for all $k > 1$, therefore $C_2, ..., C_t \in L$. This partition has the following properties [2]:

Property 1 For all edges joining a vertex of S to a vertex of $(S \cup L)$, $x(e) = 0$.

Proof. Let $e = (i, j)$ such that $i \in S$ and $j \in (S \cup L)$. By contradiction, suppose that $x(e) \geq 1$ in G and so $e \in X$ in G^*. There are two cases: either $j \in S$ or $j \in L$. If e has not been used, since $i \in S$, then we can use e to reach j from C_1 with an alternating walk ending in subtraction, giving j label $(-)$. This is a contraction since $j \in S$ or $j \in L$. If e has been already been used, it would have given label $(-)$ to either i or j , which is a contradiction. □

Property 2 For all edges joining a vertex of T to a vertex of $(T \cup L)$, $x(e) = \lambda(e)$.

Proof. Similar to the proof of Property 1. □

Property 3 There are no edges in G joining a vertex in W to a vertex in L.

Proof. By contradiction, let $e = (i, j)$ be an edge such that $i \in W$ and $j \in L$. Then either $e \in X$ or $e \in \overline{X}$ in G^*. Since $i \in W$, if e has not been used, it will be used to grow the tree from i and give j label $(-)$ or $(+)$ depending on the class of e. This is a contradiction since $j \in L$. Note that since $j \in L$, the edge cannot have been used from j to i since the vertex j will never be part of the tree, as it is not reachable from C_1. □

We say that a subgraph of G is induced by the vertex set $X \subseteq V(G)$ when its vertex set is X and its edge set consists of the edges of G joining vertices of X. Consider the subgraph of G induced by the vertices of W and denote its components $U_1, U_2, ..., U_l$. Note that a set U_k corresponds to the set of vertices represented by a pseudovertex in the tree or the union of such sets if a pseudovertex is the predecessor of another one in the tree. Every vertex of C_1 is in W, and so C_1 has to be a subset of one of the components U_i; let $C_1 \subseteq U_1$. We can derive a few more properties of the partition [2]:

Property 4 For all edges joining a vertex in U_1 to a vertex in S, $x(e) = 0$.

Proof. Let $e = (i, j)$ such that $i \in U_1$ and $j \in S$. By contradiction, suppose that $x(e) \geq 1$ in G and so $e \in X$ in G^*. Since $i \in U_1 \subseteq W$, then i is reachable from C_1 by an alternating walk ending in addition. Thus, if e has not already been used when we grow the true from i, it would be used to give j label $(-)$; so j has label $(+)$ and $(-)$. This is a contradiction since $j \in S$. Therefore, e has been used before to give i label $(-)$. But this is a contradiction,

because it would imply that j is a predecessor of a pseudovertex in U_1, but by construction every predecessor of a pseudovertex in U_1 is in W. □

Property 5 For all edges joining a vertex in U_1 to a vertex in T, $x(e) = \lambda(e)$.

Proof. Similar to the proof of Property 4. □

Property 6 The same is true for each U_k $(k > 1)$ with the exception of exactly one edge for each component. Either there is an edge $e = (i, j)$ with $i \in U_k$, $j \in S$ such that $x(e) = 1$ or there is an edge $e = (i, j)$ with $i \in U_k$, $j \in T$ such that $\lambda(e) - x(e) = 1$.

Proof. We can use a similar argument as the one used for Property 4 and 5. Assume that there is an edge $e = (i, j)$ with $i \in U_k$, $j \in S$ such that $x(e) \geq 1$, and so $e \in X$ in G^*. Then, when we grow the tree from the pseudovertex containing i, if e has not been used, it would be used to give j label $(-)$. This is a contradiction. Now assume that $j \in T$. Similarly, if e has not been used, it would be used to give j label $(+)$ and reach a contradiction. Therefore, we can assume that e has been used before to give i label $(-)$. Since U_k is the set of vertices represented by one pseudovertex or several pseudovertices appearing consecutively in our tree, there exists exactly one edge (i, j) where $i \in U_k$ and $j \in S$ or T entering the component U_k. Therefore, this edge is the exceptional edge for each component U_k $(k > 1)$ which doesn't satisfy Property 4 or 5 depending whether $j \in S$ or $j \in T$. □

Table 1 summarizes the properties of the entries of the the adjacency matrix of the fractional f-factor B obtained when the algorithm ends.

<table>
<tr><td></td><td>S</td><td>T</td><td>L</td><td>U_1</td><td>U_2</td><td>...</td><td>U_l</td></tr>
<tr><td>S</td><td>empty</td><td>*</td><td>empty</td><td>empty</td><td colspan="3">mostly empty</td></tr>
<tr><td>T</td><td>*</td><td>full</td><td>full</td><td>full</td><td colspan="3">mostly full</td></tr>
<tr><td>L</td><td>empty</td><td>full</td><td>*</td><td>0</td><td colspan="3">0</td></tr>
<tr><td>U_1</td><td>empty</td><td>full</td><td>0</td><td>*</td><td colspan="3">0</td></tr>
<tr><td>U_2</td><td rowspan="3">mostly empty</td><td rowspan="3">mostly full</td><td colspan="2">0</td><td>*</td><td colspan="2">0</td></tr>
<tr><td>...</td><td colspan="3">0</td><td>*</td><td>0</td></tr>
<tr><td>U_l</td><td colspan="4">0</td><td>*</td></tr>
</table>

TABLE 1. Summary of the Properties 1-6.

Based on the entries of Table 1, we want to prove that the partition S, T, U is indeed an f-barrier. First, we can compute $t_{T,\bar{S}}$ for the directed f-factor B using equation (12), where

a_{ij} is the entry of A_B:

$$t_{T,\bar{S}} = \sum_{\substack{i\in\bar{T}\\ j\in S}} a_{ij} + \sum_{\substack{i\in T\\ j\in\bar{S}}} (\lambda(ij) - x(i,j)) = l - 1. \tag{13}$$

By equation (11) we can also write

$$t_{T,\bar{S}} = \sum_{i\in S} f(i) + \sum_{\substack{i\in T\\ j\in\bar{S}}} (\lambda(ij)) - \sum_{i\in T} f(i) \tag{14}$$

$$= \sum_{i\in S} f(i) + \sum_{i\in T} (deg_G(i) - f(i)) - e(S,T) \tag{15}$$

$$\tag{16}$$

since $\sum_{\substack{i\in T\\ j\in\bar{S}}}(\lambda(ij)) = e(S,T)$ and, for $i \in T$, $\sum_{\substack{i\in T\\ j\in S\cup\bar{S}}}(\lambda(ij)) = deg_G(i)$.

We want to show that each U_i is an odd components. Then, we will have $q(S,T) \geq l$, and so $\sum_{i\in S} f(i) + \sum_{i\in T}(deg_G(i) - f(i)) - e(S,T) = l - 1 \leq q(S,T)$ contradicting equation (3); hence proving that no f-factor can exist and the the partition described above is an f-barrier. Recall that C is an odd component if $\sum_{i\in C} f(i) + e(C,T) \equiv 1 \pmod 2$, and note that it is the same as $\sum_{i\in C} f(i) - e(C,T) \equiv 1 \pmod 2$.

First, to show that U_1 is an odd component we use the fact that the odd cycle C_1 of half edges is in U_1. Then, we have $\sum_{i,j\in U_1} a_{ij} \equiv 1 \pmod 2$ and by using the properties of the partition proved above we get:

$$\sum_{i\in C} f(i) - e(C,T) \equiv \sum_{\substack{i\in U_1\\ j\in U_1}} a_{ij} + \sum_{\substack{i\in U_1\\ j\in S}} a_{ij} - \sum_{\substack{i\in U_1\\ j\in T}} (\lambda(ij) - a_{ij}) \pmod 2 \tag{17}$$

$$\equiv 1 + 0 + 0 \pmod 2 \tag{18}$$

$$\equiv 1 \pmod 2 \tag{19}$$

$$\tag{20}$$

Hence, U_1 is an odd-component.

Now, to show that U_k for $2 \leq k \leq l$ is an odd-component, we first observe that $\sum_{i,j\in U_k} a_{ij} \equiv 0 \pmod 2$ since there is no half edges, and so every edge inside U_k counts for 2 towards the sum of the degrees. Moreover, we know from Table 1, that either (1) $\sum_{\substack{i\in U_k\\ j\in S}} a_{ij} = 1$, $\sum_{\substack{i\in U_1\\ j\in T}}(\lambda(i,j) - a_{ij}) = 0$ or
(2) $\sum_{\substack{i\in U_k\\ j\in S}} a_{ij} = 0$, $\sum_{\substack{i\in U_k\\ j\in T}} (\lambda(ij) - x(i,j)) = 1$.

Therefore,

$$\sum_{\substack{i\in U_k\\ j\in S}} a_{ij} + \sum_{\substack{i\in U_k\\ j\in T}} (\lambda(ij) - x(i,j)) = 1.$$

Then, we get:

$$\sum_{i\in C} f(i) - e(C,T) \equiv \sum_{\substack{i\in U_k\\ j\in U_k}} a_{ij} + \sum_{\substack{i\in U_k\\ j\in S}} a_{ij} - \sum_{\substack{i\in U_k\\ j\in T}} (\lambda(ij) - a_{ij}) \pmod 2 \tag{21}$$

$$\equiv 0 + 1 \pmod 2 \tag{22}$$

$$\equiv 1 \pmod 2 \tag{23}$$

$$\tag{24}$$

Hence, for $2 \leq k \leq l$, U_k is an odd-component.

The partition of the vertices of G into S, T, U is therefore an f-barrier. It is worth noting that finding an f-barrier using the algorithmic proof given by Anstee is more restrictive than finding a partition contradicting Tutte's characterization only, equation (3). Any f-barrier created using the partition described above will contradict Tutte's characterization, however it is assuming that G has a directed f-factor, and so equation (5) always holds. There may be other partition of the set G contradicting Tutte's characterization, equation (3) but for which equation (5) does not hold.

7. Specialization to 1-factors and generalization to the (g, f)-factors

In this section, we start by investigating a special case of the f-factor problem where $f \equiv 1$ before exploring a general case of the f-factor problem, called the (g, f)-factor problem. In both situations, we want to see how Anstee's technique can be applicable.

7.1. **The 1-factor problem.** Consider the f-factor problem where $f \equiv 1$. First, we can note that a 1-factor F cannot have multiple edges or loops since we have $deg_F(i) = 1$ for all $i \in V(G)$. Therefore, when starting with a general graph G, we can restrict the study to its subgraph G' obtained by removing loops and replacing the multiple edges by a single edges. This graph G' is called the *underlying subgraph of* G. Therefore, a general graph G has a 1-factor if and only if its underlying subgraph G' has a 1-factor. One of the most important results in factor theory is Tutte's 1-factor Theorem 7.1 in which he gives a characterization of graphs for which there exists a 1-factor.

Theorem 7.1. (The 1-Factor Theorem, [11]**)** *A general graph G has a 1-factor if and only if*

$$odd(G - S) \leq |S| \qquad \text{for all } S \subset V(G)$$

where $odd(G - S)$ = the number of odd components of $G - S$.

We now want to see how Anstee's approach can be used to solve the 1-factor problem for a general graph G. Starting with the underlying subgraph G' of G, we look for a directed 1-factor B in $D(G')$. Since there are no loops, if B exists, the entries of A_B are all 0's on the diagonal and either 0, 1/2, or 1 off the diagonal. Next, we symmetrize the matrix A_B to get a symmetric directed f-factor such that $A_F = (A_B + A_B^T)/2$. Observe that the only possible entries are still 0 on the diagonal and 0, 1/2, or 1 off the diagonal. However, since the row and column sums equal 1, each row and column of A_F has either a single entry 1 with the remaining entries all 0's, or two entries 1/2 with the remaining entries all 0's. Therefore, when we form the general graph H, as described in Section 4, each vertex $i \in V(H)$ has degree 2; that is, each component of H is a cycle. Since even cycles are particular cases of even length closed trails, they can be removed as done in Section 4 by alternatively add 1/2 and subtract 1/2 from the edges. Then, only the vertex disjoints odd cycles remain in H.

In order to remove the vertex disjoint odd cycles in H by pair, we look for alternating walks as we did before. We create the general auxiliary graph G^*. However, since for all $e \notin E(H)$, $\lambda(e) = 1$ and $x(e) = 0$ or $x(e) = 1$, then G^* does not contain any multiple edges. Each edge e in G' is translated by exactly one edge in G^*: if $x(e) = 1$, we have $e \in X$ in G^* and if $x(e) = 0$, we have $e \in \overline{X}$ in G^*. Therefore, we will only have alternating paths and no alternating walks; blossoms in which an edge is used twice will not appear in this case. Therefore, Anstee's approach is reduced to an easier case than for the f-factor problem (when $f \not\equiv 1$) as we are looking at a graph without multiple edges or loops making the algorithm a little bit simpler. Note the difference with Edmond's blossom algorithm: we only need alternating path not necessarily augmenting paths.

7.2. **The (g, f)-factor problem.** While a 1-factor is a special case of an f-factor, an f-factor is itself a special case of the (g, f)-factor where $f \equiv g$. We define a *(g, f)-factor* F of a general graph G as a subgraph F of G such that

$$g(i) \leq \sum_{j:(i,j)\in E(F)} x(i, j) \leq f(i) \quad \forall i \in V(G)$$

where $g, f : V \to \mathbb{Z}^+$, $g(i) \leq f(i)$ for all $i \in V(G)$ and $x(e) \in \mathbb{Z}$. If we drop the requirement $x(e) \in \mathbb{Z}$, we say that F is a *fractional (g, f)-factor* of G. In this section, we show that a (g, f)-factor problem can be solved using Anstee's approach by making a few changes in the algorithm. Anstee gives the following theorem [2]:

Theorem 7.2. (Anstee, [2]) *Let G be a general graph on n vertices. Then G has a (g, f)-factor if and only if there is a directed (g, f)-factor and there is no (g, f)-barrier.*

We start with the general graph G, for which we want to solve the (g, f)-factor. We associate the network flow $Q(G)$ to G the same way we did for the f-factor problem except that we impose lower bounds, l, on the flow function x. On the edges from s to R_i and from S_i to t, we have $l(i) = g(i)$ for $i = 1, 2, ..., n$ as in Figure 34.

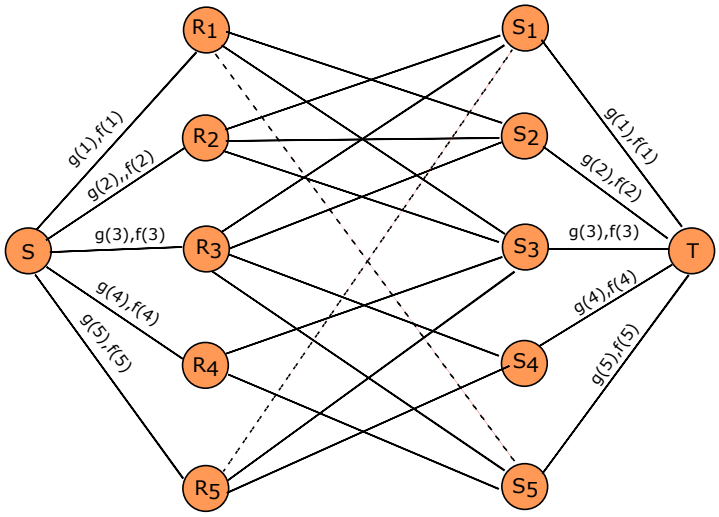


FIGURE 34. Network $Q(G)$ for the (g, f) factor problem.

Consider the directed graph, $D(G)$, associated to G such that $A_G = A_{D(G)}$. We define a *directed* (g, f)*-factor* of $D(G)$ as a subgragh B of $D(G)$ where the adjacency matrix of B, A_B, has row and column sums, $r(i)$ and $c(i)$, satisfying $g(i) \leq r(i) \leq f(i)$ and $g(i) \leq c(i) \leq f(i)$. Similarly to the approach used for the f-factor problem, we want to find a directed (g, f)-factor of $D(G)$ by finding an integral flow of the size $h(1)+h(2)+...+h(n)$ where $g(i) \leq h(i) \leq f(i)$ for all $i = 1, 2, ..., n$. The following theorem about the existence of a (g, f)-factor is a generalization of Theorem 4.5:

Theorem 7.3. *Let G be a general graph. There is a directed (g, f)-factor in $D(G)$ if and only if*

$$\sum_{i \in S} f(i) + \sum_{\substack{i \in T \\ j \in (T \cup U)}} \lambda(i, j) \geq \sum_{i \in T} g(i) \tag{25}$$

for all partitions S, T, U of $\{1, 2, ...n\}$.

If $D(G)$ doesn't have a directed (g, f)-factor, then G doesn't have a (g, f)-factor and the problem is solved. Otherwise, we proceed as we did in the f-factor problem: we transform B into a fractional (g, f)-factor and then we attempt to transform it into a (g, f)-factor.

Firstly, we obtain a fractional (g, f)-factor F by taking $A_F = (A_B + A_B^T)/2$. We then form the same general graph H as before. We need to be a little bit more careful than with the fractional f-factor as the row and column sums of A_F do not have to be integral. If the i^{th} row (and column) sum of A_F is not integral, it means that its number of half integer entries is odd and therefore $deg_H(i)$ is odd. Therefore, by Lemma 4.6, H may now have path joining two vertices of odd degree in addition to even length closed trails and vertex disjoint odd cycles.

We start the transformation of A_F by making every row and column sums integral. This can be done by using the same method as the one used to remove the even length closed trails in H; that is, by alternatively adding $1/2$ and subtracting $1/2$ from the edges of the paths joining two vertices of odd degree. Then, as desired, only the degree of the end vertices will change to become integral. Once this is done, we can remove the even length closed trails as in the f-factor problem. We are only left with the vertex disjoint odd cycles in H.

To remove those vertex disjoint odd cycles with alternating walks, we need to make a slight change in the algorithm given in Section 5. The general auxiliary graph G^* will now have *fake loops*. We define a fake loop as a loop in G^* which doesn't come from the direct translation of a loop in G. In addition to the loops in G^* directly translated from the loops in G such that $e = (i, i) \in X$ in G^* when $x(e) \geq 2$ and $e = (j, j) \in \overline{X}$ in G^* when $\lambda(e) - x(e) \geq 2$, we add fake loops in the following manner:
(1) if $\sum_{j \in V(G)} x(i, j) > g(i)$, then we add a loop $e \in \overline{X}$ at the vertex i;
(2) if $\sum_{j \in V(G)} x(i, j) < f(i)$, then we add a loop $e \in X$ at the vertex i.

Fake loops are necessary to terminate an alternating walk at the vertex i by decreasing (or increasing) the degree of i by 1 when $g(i) < r(i)$ (or $r(i) < f(i)$); that is, when changing the degree of i doesn't affect the inequality $g(i) \leq r(i) \leq f(i)$. They will act as a vertex disjoint odd cycles of H and, when used, will let the algorithm terminate as if an alternating walk was found. However, observe that by using the fake loop $e = (i, i)$, the degree of the vertex i will change by 1 depending on the class of e. When finishing an alternating walk at a vertex i using the fake loop $e = (i, i) \in X$, $deg_G(i)$ increase by 1, but the inequality $g(i) \leq r(i) (\leq f(i))$ is preserved (dual holds). Without using fake loops, the algorithm would fail at removing certain vertex disjoint odd cycles of H. Note that in the case of the (g, f)-factor problem, the vertex disjoint odd cycles are not necessarily removed by pair.

If we are able to remove all the vertex disjoint odd cycles of H, we solved the problem by finding a (g, f)-factor. Otherwise, we can find a (g, f)-barrier with the same partition S, T, U as in Section 6. From the properties proved in Section 6, we deduce the three following properties:

Property 1': For all vertices $i \in S$, $deg_F(i) = f(i)$.

Proof. Let i be a vertex in S. Since $g(i) \leq deg_F(i) \leq f(i)$, assume by contradiction that $deg_F(i) < f(i)$. Then, there is a fake loop $e = (i, i) \in X$ in G^*. Assume that e has not been used. Then, since $i \in S$, when we grow the tree from i we can use e in order to remove the odd cycle C_1. This is a contradiction. Note that e cannot have already been used otherwise the algorithm would have ended, since as soon as a fake loop is used we can terminate the algorithm and remove C_1. Therefore, $deg(i) = f(i)$. □

Property 2': For all vertex $i \in T$, $deg_F(i) = g(i)$.

Proof. Similar to the proof of Property 1′. □

Property 3' For all vertex $i \in U_k$, $g(i) = deg(i) = f(i)$.

Proof. Let i be a vertex in U_k. By contradiction, assume that $g(i) \neq f(i)$. Then, either (1) $deg(i) > g(i)$ or (2) $deg(i) < f(i)$.
Case 1: Suppose that $deg(i) > g(i)$. Then there is a fake loop $e = (i, i) \in \overline{X}$ in G^*. Since $i \in T$, it has label $(-)$ and e can be used to terminate the algorithm and remove the odd cycle C_1. This is a contradiction.
Case 2: Suppose that $deg(i) < f(i)$. Then there is a fake loop $e = (i, i) \in X$ in G^*. Since $i \in S$, it has label $(+)$ and e can be used to terminate the algorithm and remove the odd cycle C_1. This is a contradiction. □

Using those properties, we can prove that the partition S, T, U is a (g, f)-barrier since it can be used to prove that no (g, f)-factor can exist. We use the same approach as for the f-barrier and start by deducing the following result for a fractional (g, f)-factor B:

$$t_{T,\overline{S}} = \sum_{\substack{i \in \overline{T} \\ j \in S}} b_{ij} + \sum_{\substack{i \in T \\ j \in \overline{S}}} (c_{ij} - b_{ij})$$

$$= \sum_{\substack{i \in \overline{T} \\ j \in S}} b_{ij} - \sum_{\substack{i \in T \\ j \in \overline{S}}} b_{ij} + \sum_{\substack{i \in T \\ j \in \overline{S}}} c_{ij}$$

$$= (\sum_{\substack{i \in \overline{T} \\ j \in S}} b_{ij} + \sum_{\substack{i \in T \\ j \in S}} b_{ij} - \sum_{\substack{i \in T \\ j \in S}} b_{ij}) - (\sum_{\substack{i \in T \\ j \in \overline{S}}} b_{ij} + \sum_{\substack{i \in T \\ j \in S}} b_{ij} - \sum_{\substack{i \in T \\ j \in S}} b_{ij}) + \sum_{\substack{i \in T \\ j \in \overline{S}}} c_{ij}.$$

Using the fact that B is symmetric, properties deduced in Section 6 and the ones just proved above on the degree of the vertices of S, T, terms cancel out and we obtain the following equality:

$$t_{T,\overline{S}} = \sum_{i \in S} f(i) + \sum_{\substack{i \in T \\ j \in \overline{S}}} c_{ij} - \sum_{i \in T} g(i) = l - 1. \tag{26}$$

Observe that for B, using the partition constructed based on the label given to each vertex when the algorithm terminates, the expression of $t_{T,\overline{T}}$ is maximize. Therefore, for any symmetric directed (g, f)-factor F such that $A_F = a_{ij}$, we have:

$$\sum_{\substack{i \in \overline{T} \\ j \in S}} a_{ij} + \sum_{\substack{i \in T \\ j \in \overline{S}}} (c_{ij} - a_{ij}) \leq \sum_{i \in S} f(i) + \sum_{\substack{i \in T \\ j \in \overline{S}}} c_{ij} - \sum_{i \in T} g(i). \tag{27}$$

With the same reasoning as in the proof of the f-barrier in Section 6, we also deduce that:

$$\sum_{\substack{i \in \overline{T} \\ j \in S}} a_{ij} + \sum_{\substack{i \in T \\ j \in \overline{S}}} (c_{ij} - a_{ij}) \geq l. \tag{28}$$

But this is a contradiction with equation (26); therefore, no (g, f)-factor can exist and so the partition S, T, U is indeed an f-barrier.

The following theorem follows from Theorem 7.2 and provides three situations where (g, f)-factors directly arise from the existence of a fractional (g, f)-factor. It uses the three properties given above. An *independent vertex set* in a graph, is a subset of the vertices such that no two vertices of this subset are joined by an edge. We define a *(general)bipartite graph* as a (general) graph G for which the set of vertices $V(G)$ is partitioned into two disjoint independent vertex sets X_1 and X_2; that is, $X_1 \cup X_2 = V(G)$ and $X_1 \cap X_2 = \emptyset$. We also denoted G_n, the subgraph of G induced by the set of vertices $\{v \in V(G) : deg_G(v) = n\}$ and $G_{f=g}$, the subgraph of G induced by the set of vertices $\{v \in V(G) : g(v) = f(v)\}$

Theorem 7.4. (Anstee) *Let the general graph G and the functions f, g be given such that it satisfies one of the properties I-III. Then, G has a (g, f)-factor if and only if it has a fractional (g, f)-factor.*
Property I: $G_{g=f}$ is bipartite.
Property II: $g(i) < f(i)$ for $i = 1, 2, ..., n$.
Property III: $f \equiv g$, $\sum_{i \in V} f_i \equiv 0 \pmod 2$ and every pair of vertex disjoint odd cycles are joined by an edge.

Proof. Property I: Assume that G has a fractional (g, f)-factor F and $G_{g=f}$ is bipartite. Assume that the general graph H constructed from F has at least one vertex disjoint odd cycle C_1. Let C_1 be cycle from which we start growing the tree in order to try to remove it from H using the algorithm. By construction of the partition, $C_1 \in U_k$ for some k. Moreover, by Property 3′, every vertex $i \in C_1$ satisfies $g(i) = deg(i) = f(i)$. Hence, $C_1 \in G_{g=f}$. But this is a contradiction since $G_{g=f}$ is bipartite and a general bipartite graph doesn't contain any odd cycles. Therefore, H doesn't have any vertex disjoint odd cycle and so the fractional (g, f)-factor F can be transformed into a (g, f)-factor simply by making all row and column sums integral and removing the even length trails of H.

Property II: Assume that G has a fractional (g, f)-factor and that $g(i) < f(i)$ for $i = 1, 2, ..., n$. Assume that the general graph H constructed from F has at least one vertex disjoint odd cycle C_1. Let C_1 be cycle from which we start growing the tree in order to try to remove it from H using the algorithm. By construction of the partition, $C_1 \in U_k$ for some k. But since $g(i) < f(i)$ for $i = 1, 2, ..., n$, there is not vertex i such that $g(i) = deg_F(i) = f(i)$. Hence, by property 3′, the set $W = U_1 \cup U_2 \cup ... \cup U_l$ is empty. This is a contradiction. Therefore, H doesn't have any vertex disjoint odd cycle and so the fractional (g, f)-factor F can be transformed into a (g, f)-factor.

Property III: Assume that G has a fractional (g, f) factor, $f \equiv g$, $\sum_{i \in V} f(i) \equiv 0 \pmod 2$, and every pair of vertex disjoint odd cycles are joined by an edge. Since $f \equiv g$, we have a

fractional f-factor. Moreover, since $\sum_{i \in V} f_i \equiv 0 \pmod 2$, by Remark 4.9, H has an even number of odd cycles. And finally, since every pair of disjoint odd cycles are joined by an edge, all the vertex disjoint odd cycles of H can be removed.

□

8. Another strategy: reducing an f-factor problem to a matching problem

In this section, we mention another approach to the f-factor problem for a general graph G on n vertices. Instead of having to directly solve the f-factor problem as Anstee did, Gabow proved that it is possible to reduce any f-factor problem into a matching problem. Once in a matching problem situation, well-known methods to solve matching problems can be used.

First, Berge showed that an f-factor problem can be transformed into a matching problem by replacing each vertex $i \in V(G)$ by a *substitute* S. The substitute S, replacing $i \in V(G)$ with $deg_G(i) = d$, is composed of two independent sets of vertices: $V_1(S)$ has $(d - f(i))$ *internal vertices* and $V_2(S)$ has d *external vertices*, as indicated in Figure 35. Note that S is a complete bipartite graph, meaning that any pair of vertices such that one vertex is in $V_1(S)$ and the other in $V_2(S)$ is joined by an edge. We claim that finding a *perfect matching*, which is also a 1-factor, in the substitute S, gives i the right degree; that is, $deg_F(i) = f(i)$. Since it is a complete bipartite graph, if S has a perfect matching M, then exactly $(deg_G(i) - f(i))$ external vertices have their matched edges in S. We are left with $deg_G(i) - (deg_G(i) - f(i)) = f(i)$ external vertices having their matched edges in G. Therefore, M does indeed give i the right degree once the substitute S is replaced by the initial vertex i.

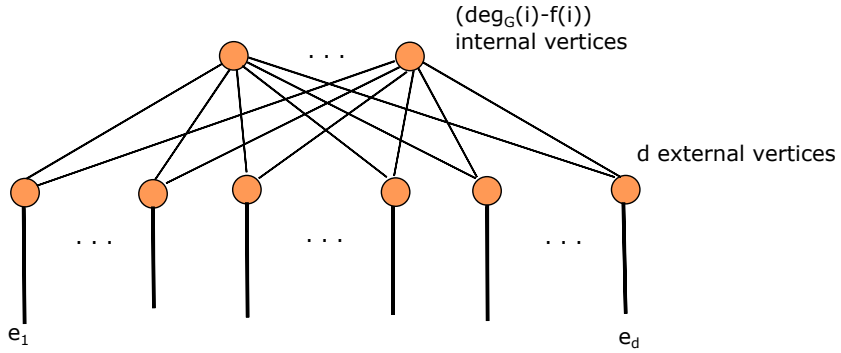


Figure 35. Substitute S for the vertex i.

Although matching problems are generally easier than f-factor problems, we can see here that when using Berge's substitute S, the number of vertices added for each vertex of i is large, making the reduction inefficient. Gabow addressed this issue by proving that an augmenting path passes through the substitute S at most twice. This observation is analogue

to the fact that an edge needs to be used at most twice in an alternating walk (see Remark 5.2). Based on this fact, Gabow gives the method of sparse substitute. It consists in replacing the substitute S introduced by Berge by a *sparse substitute* S', including less edges. Each vertex i in the initial general graph G is replaced by a configuration similar to the one shown in Figure 36. Contrary to the Berge's substitute S, the construction of a sparse substitute is defined with respect to a given matching and will change when the matched edges of i change [8]. Gabow's method of sparse substitute transform an f-factor problem into a matching problem without adding too many extra edges, making the reduction efficient. For more details about the theory of the sparse substitute, see Gabow [8].

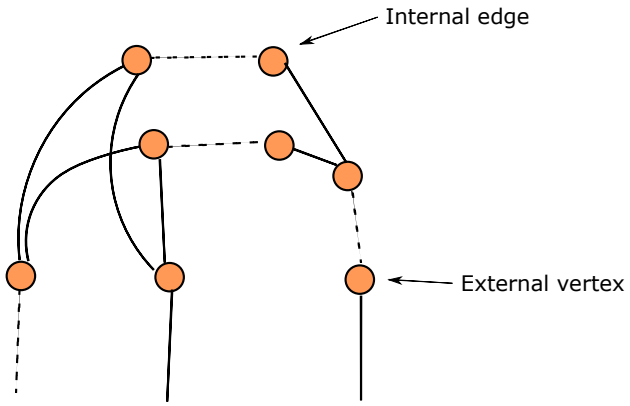


FIGURE 36. Sparse substitute S' for the vertex i.

9. Applications

In this last section, we will mention a couple a applications of (g, f)-factors and particularly how properties deduced from Anstee's algorithmic approach are useful in proving other theorems. The first application deals with edge colouring problems and the second with graph decomposition.

9.1. **Edge colouring.** Let G be a general graph. An *edge colouring* is a partition of $E(G)$ into matchings, called *colour classes*. The *chromatic index*, denoted $\chi'(G)$, is the minimum number of colour classes in any edge colouring of G. The maximum degree of any vertex in G is denoted $\triangle(G)$. Thus, we will always have $\chi'(G) \geq \triangle(G)$. In this section, we look at the upper bound on $\chi'(G)$ in terms of $\triangle(G)$ proved by Shannon: $\chi'(G) \leq \lfloor \frac{3}{2} \triangle(G) \rfloor$. Lemma 9.1 is a direct consequence of Theorem 7.4 (Property I) and will be used in proving Shannon's bound in Theorem 9.2.

Lemma 9.1. *Let G be a general graph without loops and with maximum degree $\triangle$. If $G_\triangle$ is bipartite, then there is a matching in G that meets every vertex in $G_\triangle$.*

Proof. Assume that $G_\triangle$ is bipartite. Let $f(i) = g(i) = 1$ for all vertex $i \in G_\triangle$ and $g(i) = 0$, $f(i) = 1$ otherwise. Then, note that G has a fractional (g, f)-factor such that $x(e) = 1/\triangle$

for all edges $e \in V(G)$. Therefore, by Theorem 7.4 (Property I), there is a (g, f)-factor in G. By construcution, this (g, f)-factor is a matching meeting every vertex in $G_{\triangle}$. □

Theorem 9.2. (Shannon, 1949 [10]**)** *Let G be a general graph without loops. Then G has an edge-colouring with at most $\lfloor \frac{3}{2} \triangle (G) \rfloor$ colours.*

Proof. Let M_1 be a maximal matching in G and $\triangle = \triangle(G)$. Consider the set of vertices $V((G - M_1)_{\triangle}) = \{i \in V(G - M_1) : deg_{G-M_1}(i) = \triangle\}$. Let $i, j \in V((G - M_1)_{\triangle})$ and suppose that there exists an edge $e = (i, j) \in E(G)$. Since $i, j \in V((G - M_1)_{\triangle})$, $i, j \notin M_1$, and so the edge $e = (i, j)$ can be added to the matching M_1. This is a contradiction, since M_1 is a maximal matching. Therefore, there are no edges in G joining a pair of vertices from the set $V((G - M_1)_{\triangle})$, and so $V(G_{\triangle})$ is an independent set of vertices in G. Hence, by definition $(G - M_1)_{\triangle}$ is bipartite. By Theorem 9.1, there exists a matching M_2 in $G - M_1$ such that M_2 meets every vertex of $V((G - M_1)_{\triangle})$. Note that M_2 does not have to be a maximal matching. Thus, if necessary, we add edges to M_2 in order to transform it into a maximal matching in $G - M_1$.
Let $L = G - (M_1 + M_2)$ with maximum degree at most $\triangle - 1$. We want to show that the vertices of L with degree $\triangle - 1$ form a general bipartite graph $L_{\triangle-1}$. Recall that a general graph is bipartite if and only if it has no odd cycles. By contradiction, assume that C is an odd cycle in $L_{\triangle-1}$. Observe that no two consecutive vertices in C can belong to M_2 because $G - M_1$ did not have two adjacent vertices with degree $\triangle$. So we can assume that C has two consecutive vertices, i and j, which do not meet M_2. Thus the edge $e = (i, j)$ can be added to the matching M_2. This is a contradiction since M_2 is maximal. Therefore, $L_{\triangle-1}$ does not have any odd cycles and so it is indeed a general bipartite graph. By Theorem 9.1, there exists a matching M_3 in L that meets every vertex in $L_{\triangle-1}$. Now, the general graph $G - (M_1 + M_2 + M_3)$ has maximum degree $\triangle - 2$.
By repeating this process of removing matchings one by one, we decrease the maximum degree of the general graph G by at least 2 for every 3 matchings removed. Induction now completes the proof, except when we get down to maximum degree 1 for which only one matching will be needed to remove all the edges (since there will be a matching M_n which meets every vertex of $H_{\triangle-(\triangle-1)}$).

□

9.2. **Graph decomposition.** Another application of (g, f)-factors can be found in the decomposition of a general graph G into k subgraphs. We say that a general graph can be *decomposed* into k subgraphs $F_1, F_2, ..., F_k$ if $F_1 \cup F_2 \cup ... \cup F_k = G$ and $F_1 \cap F_2 \cap ... \cap F_k = \emptyset$. Petersen proved that any general regular graph G of degree $2k$ can be decomposed into k regular subgraphs of degree 2 [9]. This theorem is known as the 2-*factorable Theorem*. It was then generalized by Anstee, who proved that a general graph in which all vertices have degree congruent to $2k$ can be decomposed into k subgraphs of degree $\frac{1}{k}$ [3]. A subgraph of G is denoted $\frac{1}{k}G$ if $deg_{(\frac{1}{k}G)}(i) = \frac{1}{k} deg_G(v)$ for all $v \in V(G)$. In his proof, Anstee uses (g, f)-factors and Euler's trails, which are trails going through every edge of a given general graph exactly once.

Theorem 9.3. (Anstee, [3]) *Let G be a general graph satisfying*

$$deg_G(v) \equiv 0 \pmod{2k} \quad \forall v \in V(G). \tag{29}$$

Then G has a subgraph, $\frac{1}{k}G$, satisfying

$$deg_{\frac{1}{k}G}(v) = \frac{1}{k} deg_G(v) \quad \forall v \in V(G). \tag{30}$$

Thus G can be decomposed into k such $\frac{1}{k}$ subgraphs.

Proof. Let G be a general graph satisfying equation (29). Then, each vertex in G has even degree. Since every connected graph with vertices of even degree has an Euler trail, we can find an Euler trail for each component of G. Now we give a direction to the edges of G as they appear in each trail. G is then transformed into a directed graph D such that

$$\sum_{\substack{v:(v,u)\\ \in A(G)}} x(v', u) = \sum_{\substack{v:(u,v)\\ \in A(G)}} x(u, v'') = \frac{1}{2} deg_G(v)' \tag{31}$$

for all $v \in V(G)$. Moreover, note that $\frac{1}{2} deg_G(v) \equiv 0 \pmod{k}$. Form the general bipartite graph $G_B = (V(G), E(G))$ as follows. Let the two independent sets of vertices of $V(G_B)$ be $V' = \{v' : v \in V(G)\}$ and $V'' = \{v'' : v \in V(G)\}$. For each directed edge (u, v) in D, G_B has an edge (u', v''). Therefore, each edge in G correspond to exactly one edge in G_B, and every edge joins a vertex in V' to a vertex in V''. We then have the two following equalities:

$$deg_{G_B}(v') = \sum_{\substack{v:(u,v)\\ \in A(G)}} x(u, v), \qquad deg_{G_B}(v'') = \sum_{\substack{v:(v,u)\\ \in A(G)}} x(v, u). \tag{32}$$

Let $\frac{1}{2} deg_G(v) = nk$ and $g(v) = f(v) = n$ for all $v \in V(G_B)$. Thus, G_B a fractional (g, f)-factor such that each edge of G_B is a fractional edge $\frac{1}{k}$; that is $g(v) = f(v) = \frac{1}{k} deg_{G_B}(v)$. Since G_B is bipartite and $g \equiv f$, by Theorem 7.4 (Property I), G_B has a (g, f)-factor where $g(v) = f(v) = \frac{1}{k} deg_{G_B}(v)$ for all $v \in V(G_B)$. Now going back to our original general graph G and regarding the edges of G_B as edges of G, we have a subgraph F_1 of G such that $deg_{F_1}(v) = \frac{1}{k} deg_G(v)$ for all vertices $v \in V(G)$.

Now, using induction, we prove that G can be decomposed into k such subgraphs. Let $l < k$, where $l \in \mathbb{N}$. Assume that there are l $\frac{1}{k}G$ subgraphs of G, say $F_1, F_2, ..., F_l$. Let $H = G - (F_1 + F_2 + ... + F_l)$. By construction, $deg_H(v) = deg_G(v) - l\,(\frac{1}{k})\, deg_G(v) \equiv 0 \pmod{2(k-l)}$. Therefore, forming the general bipartite graph construction for H as we did before for G, we can find a subgraph F_{l+1} of H such that $deg_{F_{l+1}}(v) = \frac{1}{k-l} deg_H(v)$ for all vertices $v \in V(H)$. And since $deg_H(v) = deg_G(v) - l\,(\frac{1}{k})\, deg_G(v)$, a little bit of algebra proves that $\frac{1}{k-l} deg_H(v) = \frac{1}{k} deg_G(v)$. Therefore, we found another subgraph $\frac{1}{k}G$ and there are $(l+1)$ $\frac{1}{k}G$ subgraphs of G.
By mathematical induction, there are k $\frac{1}{k}G$ subgraphs. Observe that once we have found k

such subgraphs, we have used all the edges of the initial general graph G and therefore we have found a decomposition of G into k $\frac{1}{k}G$. □

The more general theorem, Theorem 9.4, for which no condition is imposed on the degree of the vertices of the general graph G, is also given by Anstee [3]. We omit the proof as it uses (g, f)-factors and Euler trails as for Theorem 9.3 with a few additional twists.

Theorem 9.4. (Anstee, [3]**)** *Let G be a general graph and k an integer. Then G has a $\frac{1}{k}G$ subgraph satisfying the following conditions:*

$$deg_{\frac{1}{k}G}(v) = \frac{1}{k}deg_G(v) \qquad \text{if } deg_G(v) \equiv 0 \pmod{2k}; \tag{33}$$

$$deg_{\frac{1}{k}G}(v) \in \{\frac{1}{k}deg_G(v), \frac{1}{k}deg_G(v) + 1\} \qquad \text{if } deg_G(v) \equiv k \pmod{2k}; \tag{34}$$

$$deg_{\frac{1}{k}G}(v) \in \{\lfloor\frac{1}{k}deg_G(v)\rfloor, \lceil\frac{1}{k}deg_G(v) + 1\rceil\} \qquad \text{if } deg_G(v) \not\equiv \pmod{k}. \tag{35}$$

References

[1] J. Akiyama, and M. Kano, Factors and Factorizations of Graphs, Springer, 2010.

[2] R.P.Anstee, An Algorithmic Proof of Tutte's f-Factor Theorem, *Journal of Algorithms* **6** (1985), 112-131.

[3] R.P. Anstee, Dividing a Graph by Degrees, *Journal of Graph Theory*(1996), 377-384.

[4] R.P. Anstee, and J.R. Griggs, An application of matching theory of edge-colourings, *Discrete Mathematics* **156.1** (1996), 253-256.

[5] R.P. Anstee, Simplified existence theorems for (g, f)-factors, *Discrete applied mathematics* **27.1** (1990), 29-38.

[6] C. Berge, Two theorems in graph theory, *Proceedings of the National Academy of Sciences of the United States of America* **43.9** (1957), 842.

[7] L.R. Ford, and D.R. Fulkerson, Maximal flow through a network, *Canadian Journal of Mathematics* **8** (1956), 399.

[8] H.N. Gabow, An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems, *in* "Proceedings of the fifteenth annual ACM symposium on Theory of computing", Boston, 1983.

[9] J. Petersen, Die Theorie des regulären Graphen, *Acta Math.* **15** (1861), 193-220.

[10] C.E. Shannon, A theorem on colouring the lines of a network, *Journal of Mathematics and Physics* **28.2** (1949), 148-151.

[11] W.T. Tutte, The factorization of linear graphs, *Journal of the London Mathematical Society* **1.2** (1947), 107-111.

[12] W.T. Tutte, The factors of graphs, *Canadian Journal of Mathematics* **4** (1952), 314-328.