Chapter 2
Graph-Theoretic Foundations

In this chapter, we present some important concepts and algorithms from graph theory for the benefit of a general reader. These concepts and algorithms constitute the theoretical underpinnings of a variety of techniques used to solve the various problems that arise in the context of virtual craniofacial surgery. The focus of this chapter is to present the necessary graph-theoretic foundations for the research presented in the remainder of the monograph. The details of the graph-theoretic modeling of specific problems in the context of virtual craniofacial surgery, along with appropriate justifications, will be provided accordingly in the later chapters. This chapter contains an introductory section, followed by three separate sections on graph matching, graph isomorphism, and network flow. We state most of the important theoretical results without proof. The interested reader is referred to reputed textbooks on graph theory such as [31–33], and [34] for a more formal and detailed treatment of the subject.

2.1 Some Basic Terminology

In this section, we discuss some common terminology in graph theory such as the order and the size of a graph, and a walk, a path, and a cycle within a graph following [32]. We also provide the definition of a bipartite graph.

Definition 2.1 A *graph G* is an ordered pair of disjoint sets (V, E) such that E is a subset of the set $V^{(2)}$ of unordered pairs of V. The set V is the set of *vertices*, and E is the set of *edges*.

Definition 2.2 The *order* of G is the number of vertices in G, and the *size* of G is the number of edges in G.

Definition 2.3 A graph $G(V_1 \cup V_2, E)$ is *bipartite* if the two vertex sets V_1 and V_2 are disjoint and every edge in the edge set E joins a vertex in V_1 to a vertex in V_2. The graph G is said to have a bipartition (V_1, V_2).

Definition 2.4 A *path P* in a graph *G* is denoted by a sequence of vertices $(v_0, v_1, \ldots, v_n)$ that it connects. In this case, *P* is said to be a path from v_0 to v_n.

Definition 2.5 A *walk W* in a graph *G* consists of an alternating sequence of vertices and edges, such as $(v_0, e_1, v_1, e_2, v_2, \ldots, v_{n-1}, e_n, v_n)$, where $e_i = \{v_{i-1}, v_i\}$, $0 < i \leq n$. In this case, *W* is termed as a $v_0 - v_n$ walk of length *n*. The difference between a path and a walk is made clear by context in spite of the notational similarity.

Definition 2.6 A graph C_n constitutes a *cycle* of order *n* if its vertices v_i, $0 < i < n$, are distinct from each other, $n \geq 3$, $v_0 = v_n$, and there exists a $v_0 - v_n$ walk.

Consider the graph shown in Fig. 2.1. It has an order 5 as it contains five vertices, namely $\{A, B, C, D, E\}$. The size of the graph is 7 as it contains seven edges, namely $\{AB, AD, AE, BC, BD, BE, DE\}$.

Figure 2.2 shows a bipartite graph. Note that the two vertex sets V_1 and V_2 are disjoint. V_1 consists of the vertices $\{A, B, C\}$, and V_2 consists of the vertices $\{D, E\}$. The edges $\{AD, AE, BD, BE, CD, CE\}$ are such that each edge connects one vertex in V_1 to another vertex in V_2.

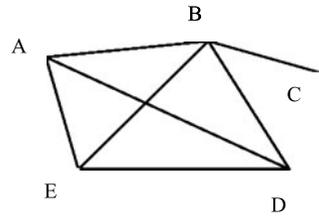The graph in Fig. 2.3 constitutes a cycle of order 4.

Fig. 2.1 A graph
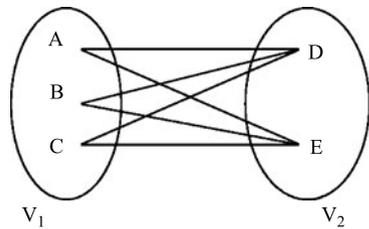

Fig. 2.2 A bipartite graph
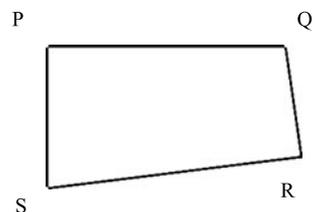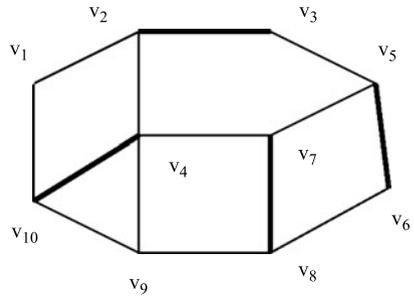

Fig. 2.3 A cycle graph

Fig. 2.4 Matching in a
general graph

2.2 Matchings in Graphs

In this section, we first provide some important definitions and then present some
key results on matchings in general graphs and bipartite graphs.

Definition 2.7 A *matching M* of a graph $G = (V, E)$ is a subset of the edges with
the property that no two edges of M share the same node.

Definition 2.8 Edges of a graph in a matching are called *matched* edges; the other
edges are called *free*. Similarly, the vertices that are not incident upon any matched
edge are called *exposed*; the remaining vertices are called *free*.

Definition 2.9 A path $P = (v_1, v_2, \ldots, v_k)$ is called *alternating* if:

edges $\{v_1, v_2\}, \{v_3, v_4\}, \ldots, \{v_{2j-1}, v_{2j}\}, \ldots$ are free,
whereas edges $\{v_2, v_3\}, \{v_4, v_5\}, \ldots, \{v_{2j}, v_{2j+1}\}, \ldots$ are matched.

Definition 2.10 An alternating path $p = (v_1, v_2, \ldots, v_k)$ is called *augmenting* if
both v_1 and v_k are exposed vertices.

Example 2.1 Consider Fig. 2.4 which shows a matching M in a graph G with ver-
tices $\{v_1, \ldots, v_{10}\}$. We can then write the following:

Matched edges: $\{v_2, v_3\}, \{v_4, v_{10}\}, \{v_5, v_6\}, \{v_7, v_8\}$.
Free edges: $\{v_1, v_2\}, \{v_1, v_{10}\}, \{v_2, v_4\}, \{v_3, v_5\}, \{v_4, v_7\}, \{v_4, v_9\}, \{v_5, v_7\}$,
$\{v_6, v_8\}, \{v_8, v_9\}, \{v_9, v_{10}\}$.
Exposed vertices: $\{v_1, v_9\}$.
An alternating path: (v_9, v_8, v_7, v_5).
An augmenting path: $(v_9, v_8, v_7, v_4, v_{10}, v_1)$.

Definition 2.11 If the edge weights of a graph are all unity, the matching problem
is essentially a *Cardinality Matching* problem. A *Maximum Cardinality Matching*
is a matching with a maximum number of edges.

Definition 2.12 When the cardinality of a matching is $\lfloor |V|/2 \rfloor$, the largest possible
in a graph with $|V|$ nodes, we say that the matching is *complete* or *perfect*.

Fig. 2.5 Maximum
cardinality matching in a
general graph
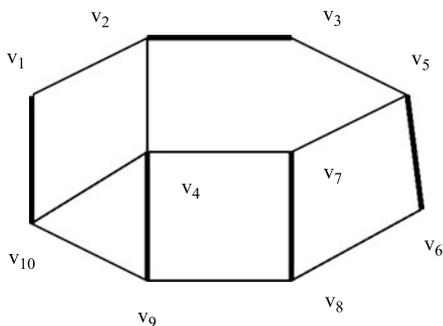
Example 2.2 Consider Fig. 2.5 which shows a maximum cardinality matching M^*
in a graph G with vertices $\{v_1, \ldots, v_{10}\}$.

Matched edges: $\{v_1, v_{10}\}, \{v_2, v_3\}, \{v_4, v_9\}, \{v_5, v_6\}, \{v_7, v_8\}$.

Since the number of vertices is 10 and the cardinality of the matching is 5, this
is also an example of complete or perfect matching. Note that there are no exposed
vertices and hence no augmenting paths in this figure (as opposed to Fig. 2.4).

Definition 2.13 If the edge weights are given by a function $w : E \to \Re_+$, the weight
of a matching is defined as $w(M) = \sum_{e \in M} w(e)$. The *Maximum Weight Matching*
problem is to determine a matching M in G that has maximum weight.

Next, we present three theorems on matchings in bipartite and general graphs.
The first theorem discusses the condition for the existence of maximum matching.
The next two theorems give the results for the worst-case time-complexity of two
graph matching algorithms, namely, the Maximum Cardinality Minimum Weight
(MCMW) matching for a bipartite graph and Maximum Weight Graph Matching
(MWGM) for a general graph.

Theorem 2.1 *A matching M in a graph G is maximum if and only if there is no
augmenting path in G with respect to M.*

Theorem 2.2 *The worst-case time complexity of the Maximum Cardinality Mini-
mum Weight (MCMW) matching algorithm for a bipartite graph $G = (V_1 \cup V_2, E)$
with $|V_1| = |V_2| = n$ is $O(n^3)$.*

Theorem 2.3 *The worst-case time complexity of the Maximum Weight Graph
Matching (MWGM) algorithm for a general graph $G = (V, E)$ with $|V| = n$ is
$O(n^4)$.*

For detailed proofs of the above theorems and details of the matching algorithms,
the interested reader can refer to [31, 33].

2.3 Isomorphism and Automorphism of Graphs

We discuss the isomorphism and automorphism of graphs with some examples. Some important results on the time complexity of the graph isomorphism problem and graph automorphism problem are also stated.

Definition 2.14 Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic, denoted by $G_1 \cong G_2$, if there exists a bijection $M \subseteq V_1 \times V_2$ such that, for every pair of vertices $v_i, v_j \in V_1$ and $w_i, w_j \in V_2$ with $(v_i, w_i) \in M$ and $(v_j, w_j) \in M$, $(v_i, v_j) \in E_1$ if and only if $(w_i, w_j) \in E_2$. In such a case, M is a *graph isomorphism* from G_1 to G_2.

Definition 2.15 An *automorphism* of a graph G is a graph isomorphism between G and itself.

   The set of all automorphs of a graph forms a group under the operation of composition. This group is termed the *automorphism group* of the graph. It is a well-known fact that the graph isomorphism problem (i.e., determining whether or not two graphs are isomorphic) $\in NP$. However, it can be solved in *polynomial time* for many special graphs [34]. Now, we state and prove a result on graph automorphisms for cycle graphs.

Theorem 2.4 *The automorphism group of a cycle graph C_n on $n \geq 3$ vertices is a group of order $2n$.*

Proof A cycle graph C_n on $n \geq 3$ vertices is left fixed by exactly n rotations as well as by exactly n reflections. Thus, the resulting automorphism group has order $2n$. $\square$

Example 2.3 The two graphs G_1 and G_2 in Fig. 2.6 are isomorphic. The mapping from G_1 to G_2 is given by the following bijection:

$$M = \big\{(A, 1), (B, 2), (C, 3), (D, 4)\big\}.$$

Fig. 2.6 A pair of isomorphic graphs
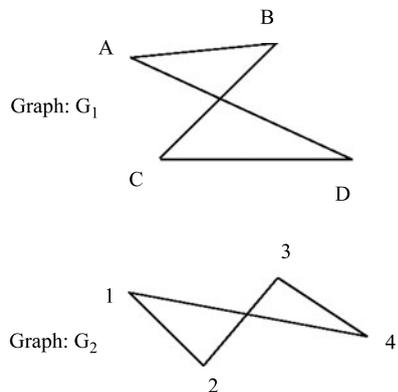
Example 2.4 Let C_4 be a cycle graph of order 4 with vertices $[P, Q, R, S]$, as shown in Fig. 2.3. Then from Theorem 2.4 we conclude that there exist 4 rotational automorphs and 4 reflectional automorphs, i.e., a total of 8 automorphs of C_4. The members of the automorphism group of C_4 are given by:

1. $\{P, Q, R, S\}, \{S, P, Q, R\}, \{R, S, P, Q\}, \{Q, R, S, P\}$ (these are the 4 rotational automorphs)
2. $\{Q, P, S, R\}, \{P, S, R, Q\}, \{S, R, Q, P\}, \{R, Q, P, S\}$ (these are the 4 reflectional automorphs)

2.4 Network Flows

In this section, we describe the basic concepts underlying network flows following [35]. As in previous sections, we state the definitions, provide some illustrative examples, and state key theorems without proofs. For the proofs of the theorems in this section, the interested reader is referred to well-known textbooks such as [35] and [36]. We end the section with a description of the Ford–Fulkerson algorithm for computing the maximum flow in a flow network.

Definition 2.16 A *flow network* $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$. Two vertices in the flow network are distinguished as a source vertex s and a sink vertex t.

Definition 2.17 A *flow* in G is a real-valued function $f : V \times V \to \Re^+$ that satisfies the following properties:

$$\forall u, v \in V, \quad f(u, v) \leq c(u, v); \tag{2.1}$$

$$\forall u, v \in V, \quad f(u, v) = -f(v, u); \tag{2.2}$$

$$\forall u \in V - \{s, t\}, \quad \sum_{v \in V} f(u, v) = 0. \tag{2.3}$$

The quantity $f(u, v)$ is called the flow from the vertex u to vertex v.

**Definition 2.18** The *value* of a flow is defined as

$$|f| = \sum_{v \in V} f(s, v), \tag{2.4}$$

that is, the total flow out of the source.

Definition 2.19 Given a flow network $G = (V, E)$ and a flow f, the *residual graph* of G induced by f is $G_f = (V, E_f)$ where $E_f = \{(u, v) \in V \times V \mid c_f(u, v) > 0\}$. Here, $c_f(u, v)$ denotes the *residual capacity* of (u, v). Let $f(u, v)$ and $c(u, v)$ respectively denote the flow and capacity between u and v. Then, we can write

$$c_f(u, v) = c(u, v) - f(u, v). \tag{2.5}$$

Fig. 2.7 Flow in a graph.
(a) Flow network,
(b) Corresponding residual
network

(a)

(b)
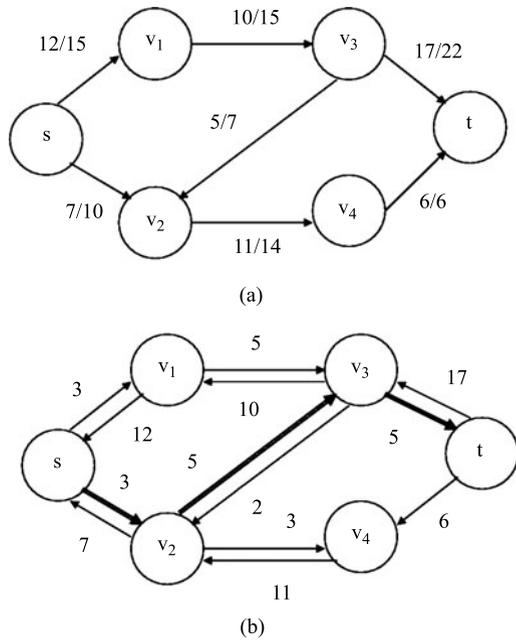
Definition 2.20 Given a flow network $G = (V, E)$ and a flow f, an *augmenting path* p is a simple path from s to t in the residual network G_f.

Definition 2.21 We have previously defined the residual capacity for an edge in a flow network. Now, we define the residual capacity $c_f(p)$ of an augmenting path p as follows:

$$c_f(p) = \min\{c_f(u, v) \mid (u, v) \text{ is on } p\}. \tag{2.6}$$

Thus, $c_f(p)$ is the maximum amount by which a flow can be increased on each edge in the augmenting path p.

Definition 2.22 A *cut* (S, T) of a flow network $G = (V, E)$ is a partition of V into S and $T = V - S$ such that $s \in S$ and $t \in T$.

Definition 2.23 The *capacity* of a cut (S, T) is the sum of the capacities of the edges from S to T.

Example 2.5 Consider the flow network in Fig. 2.7(a). Each edge in the flow network is labeled with a flow and a capacity. The residual network is shown in Fig. 2.7(b) with an augmenting path $p = (s, v_2, v_3, t)$. Then, using (2.5) and (2.6), we can write:

$$c_f(s, v_2) = c(s, v_2) - f(s, v_2) = 10 - 7 = 3,$$
$$c_f(v_2, v_3) = c(v_2, v_3) - f(v_2, v_3) = 7 - 5 = 2,$$

Fig. 2.8 Cut in a graph
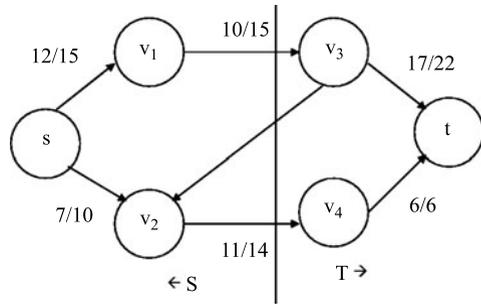
$$c_f(v_3, t) = c(v_3, t) - f(v_3, t) = 22 - 17 = 5,$$
$$c_f(p) = \min\{c_f(s, v_2), c_f(v_2, v_3), c_f(s, v_2)\} = \min\{3, 2, 5\} = 2.$$

Example 2.6 Figure 2.8 depicts the existence of a cut in the same flow network. We can infer following from Fig. 2.8:

A cut (S, T) can be observed in the above flow network where $S = \{s, v_1, v_2\}$, $T = \{v_3, v_4, t\}$.

Capacity of this cut $= c(v_1, v_3) + c(v_2, v_4) = 15 + 14 = 29$.

Note that only edges going from S to T but not the edges in the reverse direction contribute to the capacity of a cut. Hence, only $c_f(v_1, v_3)$ and $c_f(v_2, v_4)$ are considered, and $c_f(v_3, v_2)$ is excluded when computing the capacity of the cut (S, T).

Next, we state some important theorems for flow networks. The first theorem relates the maximum flow and minimum cut in a flow network. The second theorem discusses the nature of capacity functions and the corresponding flow. Once the above definitions are extended to a multisource, multisink flow network, the third theorem establishes the equivalence of a multisource, multisink network flow problem with a single-source single-sink network flow problem.

Theorem 2.5 *For any graph, the maximum flow value from s to t is equal to the minimal cut capacity of all cuts separating s and t. This is known as Max-flow Min-Cut Theorem.*

Theorem 2.6 *If the capacity function c is integer valued, there exists a maximal flow f that is also integer valued. This is called the Integrity Theorem.*

Theorem 2.7 *Maximum flow in a graph with multiple sources $s_1, s_2, \ldots, s_n$ and multiple sinks $t_1, t_2, \ldots, t_n$ is induced by a maximum flow in a simple equivalent graph with an added hypersource s^* and a hypersink t^* with capacities $c(s^*, s_i) = \infty$ and $c(t_i, t^*) = \infty$. This is called the Multisource Multisink Maximum-flow Minimum-cut Theorem.*

The Ford–Fulkerson algorithm is used to find the maximum-flow in a flow network. This algorithm is based on the concepts of a residual graph, augmenting path and cut. Next, we present the basic structure of this algorithm following [35]:

Algorithm 2.1 (Ford–Fulkerson)
Input: A flow network $G(V, E)$ with source s and sink t
Output: A maximum flow f

for each edge in the flow network G
    initialize flow $f := 0$
while there exists an augmenting path p
      do augment flow f along p by adding the residual capacity
      on the augmenting path $c_f(p)$
return f

The time complexity of the Ford–Fulkerson algorithm is $O(E|f^*|)$, where f^* is the maximum flow. Note that the flow value can be increased by at least unity in each iteration of the while loop in the above algorithm, i.e., the loop can be executed at most $|f^*|$ times. The running time of the algorithm increases substantially if $|f^*|$ is quite large. For practical purposes, we often use the Edmonds–Karp algorithm for computation of the augmenting paths. A breadth-first search is employed in such cases where an augmenting path is detected as a shortest path from the source to sink. It can be shown that the Edmonds–Karp algorithm has a time complexity of $O(VE^2)$.