A Graph-Based Method for Face Identification from a Single 2D Line Drawing

Jianzhuang Liu and Yong Tsui Lee, *Member*, *IEEE Computer Society*

Abstract—The faces in a 2D line drawing of an object provide important information for the reconstruction of its 3D geometry. In this paper, a graph-based optimization method is proposed for identifying the faces in a line drawing. The face identification is formulated as a maximum weight clique problem. This formulation is proven to be equivalent to the formulation proposed by Shpitalni and Lipson in [1]. The advantage of our formulation is that it enables us to develop a much faster algorithm to find the faces in a drawing. The significant improvement in speed is derived from two algorithms provided in this paper: The depth-first graph search for quickly generating possible faces from a drawing and the maximum weight clique finding for obtaining the optimal face configurations of the drawing. The experimental results show that our algorithm generates the same results of face identification as Shpitalni and Lipson's, but is much faster when dealing with objects of more than 20 faces.

Index Terms—3D object reconstruction, depth-first search, face identification, graph algorithms, line drawing interpretation, maximum weight clique problem.

1 Introduction

An important research area in computer vision is developing algorithms that can interpret a single 2D line drawing of an object as humans do and can reconstruct its 3D geometry. One application of such research is in CAD, where it is highly desirable to convert a design sketch into a 3D model. An object consists of faces. If the face configuration of an object is known before reconstructing it, the complexity of the reconstruction problem will be reduced significantly [1], [2], [3]. Our work in this paper is to find the faces in a 2D wireframe drawing which are the ones humans would most likely identify. Here, a line drawing should be able to be represented by a single edge-vertex graph and is a 2D projection of an object from a general viewpoint that reveals all the edges and vertices of the object (no edges or vertices coincide). Fig. 1 shows two such line drawings together with their individual faces. The object in Fig. 1a has six faces, while that in Fig. 1c has three.

In recent years, several papers addressed 3D reconstruction from 2D single wireframe drawings. Marill [4] presented a very simple optimization approach that can duplicate human perception in recovering the 3D shape of a simple 2D line drawing. Leclerc and Fischler [3] improved Marill's results significantly by first finding the face configuration of a wireframe object and then combining a planarity constraint into Marill's objective function, but Leclerc and Fischler's method of finding the planar faces of a drawing is only suitable for simple objects (such as those without holes) and cannot deal with multisolution cases. Shpitalni and Lipson took the work further and presented impressive results in face identification from wireframe objects [1] and in reconstruction of 3D objects from line drawings based on the face configurations found [2]. Their method can handle a large range of objects that may be manifold or nonmanifold. However, their algorithm for face identification is inefficient. In our experiments, when dealing with an object consisting of more than 20 faces, Shpitalni and Lipson's algorithm requires such a long time (for example, more than one hour on a Pentium II PC for an object with 27 faces) that it cannot be applied in practice because of the exponentially increasing combinatorial search. In this paper, we revisit the problem tackled by Shpitalni and Lipson in [1] proposing, for face identification, a graph-based optimization method that is different from, but equivalent to, Shpitalni and Lipson's method. Our formulation allows us to develop a more efficient algorithm to find the faces of a wireframe object.

The rest of this paper is organized as follows: In Section 2, we briefly describe Shpitalni and Lipson's method and point out why it is so time-consuming. We present our method in Section 3. The equivalence between these two methods is proven in Section 4. Section 5 provides two separate algorithms that carry out the main computation in our method: Depth-first graph search for quickly generating the minimal set of potential faces of a drawing and maximum weight clique finding for choosing the best face configurations from these potential faces. Section 6 presents the experimental results and the comparisons of computational time between our and Shpitalni and Lipson's algorithms. Finally, our conclusions are given in Section 7.

- *J. Liu is with the Department of Electronic Engineering, The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong. E-mail: jzliu@ee.cuhk.edu.hk.*
- *Y.T. Lee is with the School of Mechanical and Production Engineering, Nanyang Technological University, Nanyang Avenue, Singapore. E-mail: mytlee@ntu.edu.sg.*

Manuscript received 3 Feb. 2000; revised 29 Nov. 2000; accepted 9 Apr. 2001. Recommended for acceptance by S. Dickinson, M. Pelillo, and R. Zabih. For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 111376.

2 Shpitalni and Lipson's Method

Shpitalni and Lipson's work in [1] is closely related to our work in this paper: finding faces in a single 2D drawing.

0162-8828/01/$10.00 © 2001 IEEE

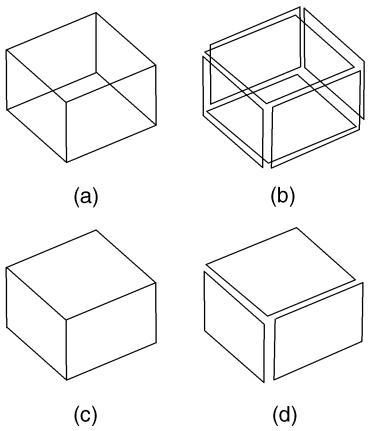

Fig. 1. (a) A drawing and (b) its six faces. (c) Another drawing and (d) its three faces.

They proposed two methods of face identification, respectively, for manifold objects of genus zero and for general objects. Their first method is simple and efficient, but only suitable for a limited type of objects. For general manifold or nonmanifold objects, face identification becomes much more difficult. Shpitalni and Lipson's main effort is in developing the second method. This method can successfully find the faces of a general object despite the inefficiency of their algorithms when dealing with objects of many faces. Here, our work is to develop a more efficient method for face identification for general objects. Below, we briefly describe some terms (also used in this paper) and Shpitalni and Lipson's method for general objects. Recall that a line drawing is represented by an edge-vertex graph.

- $d(v)$: The degree of a vertex v denoting the number of edges meeting at v.
- $R(v)$: The rank of a vertex v denoting the number of faces with boundaries passing through v.
- $R(e)$: The rank of an edge e denoting the number of faces with boundaries passing through e.
- A circuit: A closed trail (in a graph) where all its vertices except the end vertices are distinct.
- A non-self-intersecting circuit: A circuit without edges intersecting.
- A potential face: A non-self-intersecting circuit.
- A minimal potential face: A potential face without a smooth (noncreased) edge (in the drawing) connecting two of its nonadjacent vertices.

Fig. 2 shows a simple drawing and all its potential faces, among which the last four are minimal potential faces. Circuit $(1, 2, 3, 7, 6, 4, 1)$ is not a minimal potential face because there exists an edge connecting vertices 3 and 4 in the drawing. The last three circuits are real faces.

Shpitalni and Lipson's face identification method for general objects is built upon a basic theorem, the face adjacency theorem, which states that two adjacent planar faces may coexist in the same object if and only if their common edges are collinear. This theorem can be expanded to allow nonplanar, smooth faces. The expanded one states that two adjacent smooth faces may coexist in the same object if and only if their common edges are smooth [1]. Their method can be summarized in these steps:

1. Find all the potential faces from a given wireframe line drawing using the circuit space method.
2. Obtain a smaller set of minimal potential faces from the set of the potential faces.
3. For n minimal potential faces, calculate the binary matrix $B = [b_{ij}]_{n\times n}$ according to the face adjacency theorem, where $b_{ij} = 1$ denotes that faces i and j can coexist in the same object, whereas $b_{ij} = 0$ denotes that they cannot.
4. Calculate the upper bounds of the ranks (also called the maximum ranks) of all the edges and vertices from the drawing through an iterative procedure.
5. Use the A* algorithm to search for the optimal solutions of this optimization problem:[1]

$$\text{minimize: } g(x) = \sum [R^+(e) - R(e)] + \sum [R^+(v) - R(v)] \quad (1)$$

$$\text{subject to: } R(e) \le R^+(e), \forall e \quad (2)$$

$$R(v) \le R^+(v), \forall v \quad (3)$$

$$b_{ij} = 1, i \ne j,\ i,\ j \in x, \quad (4)$$

   where x is a subset of the minimal potential faces, $R(e)$ and $R(v)$ are the respective actual edge and vertex ranks corresponding to x, and $R^+(e)$ and $R^+(v)$ are the respective upper bounds of edge and vertex ranks of the drawing (see Section 4.1 for how to derive $R^+(e)$ and $R^+(v)$).
6. When there is more than one solution, use image regularities [1] to select the most plausible one.

The kernel of the method is the formulation of face identification presented in (1), (2), (3), and (4). This Shpitalni and Lipson's optimization problem is called SLOP in what follows. Steps 1 and 2 generate the input of the A* algorithm for solving SLOP and Steps 3 and 4 are used to obtain $R^+(e)$, $R^+(v)$, and B for building SLOP. The algorithms in Steps 1 and 5 consume most of the computation time and are inefficient. The reasons for the inefficiency are given below. First, we consider the algorithm in Step 1.

A wireframe line drawing such as that shown in Fig. 3 can be represented as a connected undirected graph. Let $G = (V, E)$ be such a graph where V and E are the sets of vertices and edges of G, respectively. Let $T = (V, E')$ be a spanning tree of G, where E' is the edge set of T. Any edge in $E - E'$ constructs one circuit when added to T. Such a circuit is called a fundamental circuit. All the fundamental circuits of G, with respect to T, form a basis for the circuit space of G [5]. Every spanning tree of G contains $|V| - 1$ edges, so there are $\mu = |E| - |V| + 1$ fundamental circuits with respect to any spanning tree of G. An efficient $O(|V|^3)$ algorithm producing the set of fundamental circuits for G can be found in [5].

Any circuit of G can be expressed as a linear combination (called ring-sum) of the fundamental circuits. Thus, all the circuits of G can be generated using ring-sums with the

1. In [1], Shpitalni and Lipson presented the objective function in this form $g(x) = \sum |R^+(e) - R(e)| + \sum |R^+(v) - R(v)|$. The two absolute signs are removed in (1) because of the two constraints in (2) and (3).

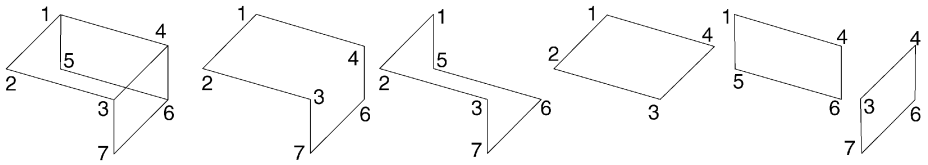

Fig. 2. A drawing and its potential faces.

circuit basis. Enumerating all possible ring-sums (excluding the null element), we obtain $2^{\mu}-1$ vectors, each of which is a circuit or an edge-disjoint union of circuits [6]. Because not every vector is a circuit, a test is required to delete the edge-disjoint unions of circuits. In general, only a small fraction of $2^{\mu}-1$ vectors are circuits. The best time bound of a circuit space algorithm for producing all the circuits of a graph is $O(\mu \cdot 2^{\mu})$ [6]. For face identification, circuits with intersecting edges do not correspond to the faces of an object and can be deleted, leaving behind the potential faces. In fact, the potential faces are also a small fraction of the circuits. Further, the number of potential faces is reduced in Step 2 to obtain a set of minimal potential faces.

For the object in Fig. 3, we have $\mu = 15$ and $2^{\mu}-1 = 32,767$. There are only 3,185 circuits among these 32,767 vectors and 734 non-self-intersecting circuits out of the 3,185 circuits. Among the 734 potential faces, there are only 54 minimal potential faces. The number of vectors generally increases exponentially with increase in the number of faces of an object. The circuit space method needs to enumerate a large pool of vectors before obtaining a much smaller set of minimal potential faces. Obviously, this method is inefficient.

The A* algorithm in Step 5 is also very time consuming; it is used to search for the optimal solutions of SLOP. A* is a state space search approach utilizing certain domain-dependent heuristic information to focus the search for the optimal path. Its efficiency depends critically on how precise the estimate of the heuristic function is and, in general, precise estimates are quite difficult to obtain [7], [8]. Shpitalni and Lipson also employed a heuristic function to help A* speed up its search. The information is the total edge number of the minimal potential faces that have not been assigned and do not conflict with the already assigned minimal potential faces. Note that, when an object has many faces (say, 20), the minimal potential faces (the input of A*) are many more than the actual faces of the object. For the states, except those at deep levels (near the end of the search paths), there are many minimal potential faces that remain to be tested and do not conflict with the already assigned faces. These minimal potential faces, most of which are not actual faces, make the estimate inaccurate. In addition, A* needs to search for possible multiple solutions. Even if it has found an optimal solution with the minimum value of the objective function $g(x) = 0$, it cannot be stopped. Therefore, the heuristic information in Shpitalni and Lipson's A* algorithm is insignificant in improving the search. Instead, the constraints in (2), (3), and (4) play a more important role in pruning the size of the huge search tree. However, our experimental results, given in Section 6, show that the A* algorithm with pruning by the constraints still consumes a large amount of search time.

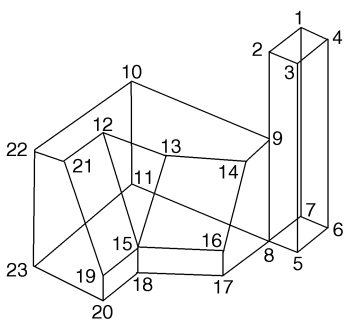

Fig. 3. A graph of 23 vertices and 37 edges.

3 Formulation of Face Identification

For a wireframe drawing with all the edges visible, a human tends to choose a face configuration in which there are as many edges as possible. This is the criterion for our solution to the face identification problem (it is also the criterion for building SLOP in [1]). Besides, the faces selected must not violate the constraint imposed by the face adjacency theorem. With these two points, we define the problem as follows:

Definition 1. *Let $w(i)$ be the number of edges of a face i. Given an edge-vertex graph of an object and the set, denoted by $SMPF$, of the minimal potential faces generated from the graph, the objective of face identification is to find all the sets $x_1, x_2, \ldots, x_s$, where s is the number of sets and $x_k \subset SMPF$, $1 \le k \le s$, such that, for every x_k, the sum of $w(i)$, $i \in x_k$, is maximal and the faces in x_k satisfy the face adjacency theorem. In short, the problem is:*

$$maximize\colon f(x) = \sum_{i \in x} w(i),\ x \subset SMPF \tag{5}$$

$$subject\ to\colon b_{ij} = 1,\ i \neq j,\ i, j \in x. \tag{6}$$

In Definition 1, more than one solution is found when $s > 1$. Comparing this optimization problem with SLOP, we can see that our representation is simpler and does not contain any ranks. Actually, we will prove in the next section that these two representations are equivalent. Now, we show that the problem presented in (5) and (6) corresponds to the maximum weight clique problem (MWCP).

MWCP is an extension of the maximum clique problem (MCP) in graph theory. A clique of a graph G [2] is a complete subgraph of G, a subgraph in which any two vertices are adjacent. A maximum clique of G is the largest complete subgraph of G. MCP is to find the maximum clique of G. If

2. Here, G may not be an edge-vertex graph of an wireframe object.

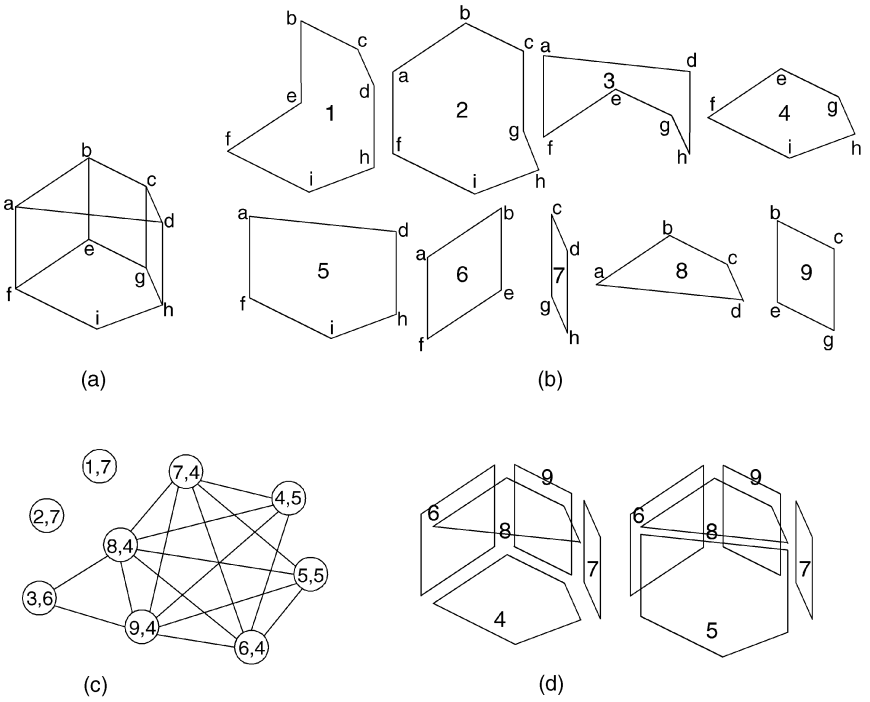

Fig. 4. MWCP for face identification. (a) A drawing of five faces. (b) Nine minimal potential faces of the drawing. (c) The weighted graph corresponding to the minimal potential faces, where the two digits inside a circle (vertex) denote the label and the weight of the vertex, respectively. (d) Two possible solutions with the maximum clique weight 21.

the vertices of G are associated with weights (which should not all be the same), G is called a weighted graph. The weight of a clique is the sum of all the weights of the vertices of the clique. MWCP is the problem of finding a clique of greatest weight in a weighted graph. To see how the problem in Definition 1 corresponds to MWCP, we can create a weighted graph in which a minimal potential face is represented by a vertex with the number of edges of the face being its weight and vertex i and vertex j are adjacent if $b_{ij} = 1$. It is obvious that the problem in Definition 1 is actually MWCP if the objective of MWCP is to find all the cliques of greatest weight. This is the case in our present problem. Thus, we will be solving MWCP.

Now, we give an example in Fig. 4 to illustrate MWCP for face identification more clearly. All the minimal potential faces of the drawing in Fig. 4a are shown in Fig. 4b. The weighted graph is created using these faces and the binary matrix $B = [b_{ij}]_{9\times 9}$ (Recall that B is obtained according to the face adjacency theorem). Vertices 1 and 2 are two isolated ones due to $b_{1x} = 0, x = 2, 3, \ldots, 9$ and $b_{2y} = 0, y = 1, 3, 4, \ldots, 9$. The two maximum weight cliques corresponding to two possible solutions are shown in Fig. 4d. In Section 5, we will provide an algorithm for finding all the maximum weight cliques of a weighted graph.

4 Proof of Equivalence

In this section, we prove that MWCP is equivalent to SLOP. But, first, we point out that the iterative procedure in Step 4 of Shpitalni and Lipson's method for finding the maximum ranks [1] is not necessary. Actually, they can be calculated directly with two equations, which leads to the reduction of SLOP to MWCP.

4.1 Calculation of Maximum Ranks

The so-called ranks play a very important role in Shpitalni and Lipson's method of face identification. The maximum edge and vertex ranks calculated from a given drawing impose two of the three constraints on the faces found by A* when searching (see (2) and (3)). For simplicity, the maximum edge and vertex ranks are denoted, respectively, in this section by $R(e)$ and $R(v)$, instead of $R^+(e)$ and $R^+(v)$.

Based on the face adjacency theorem, when no two edges meeting at a vertex are collinear, Shpitalni and Lipson gave the following three inequalities and an equation for finding the maximum edge and vertex ranks of a graph:

$$R(v) \le \{d(v)[d(v)-1]\}/2 \tag{7}$$

$$R(e) \le \min\{d(v_1), d(v_2)\} - 1 \tag{8}$$

$$R(v) = \lfloor \Big[\sum R(e)\Big]/2 \rfloor, \text{for all edges meeting at vertex } v \tag{9}$$

$$R(e) \le \min\{R(v_1), R(v_2)\}, \tag{10}$$

where v_1 and v_2 are two end-vertices of edge e in (8) and (10), $\lfloor a \rfloor$ denotes the largest integer $\le a$, all the ranks are integers. These relations are derived from the local analysis of a general edge and its two end-vertices. Since a face boundary passing through vertex v must pass through two edges meeting at v, the largest number of faces passing through v cannot exceed $C^2_{d(v)}$, the possible edge pair combinations at v, which leads to (7). Similarly, because a face passing through edge e also passes through one of the other edges meeting at its end-vertices v_1 or v_2, we have $R(e) \le d(v_1) - 1$ and $R(e) \le d(v_2) - 1$; thus, (8) follows.

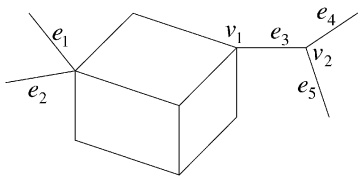

Fig. 5. A graph with two parasitic trees.

After the ranks of all the edges meeting at vertex v have been obtained, (9) follows from the fact that a face passing through v also passes through two of these edges. Since a face passing through an edge e also passes through its two end-vertices v_1 and v_2, we have $R(e) \leq R(v_1)$ and $R(e) \leq R(v_2)$, which lead to (10). Shpitalni and Lipson used (7) and (8) to compute the preliminary estimation of the maximum edge and vertex ranks and then applied (9) and (10) iteratively until all the ranks satisfy (7), (8), (9), and (10).

In the following, we prove that this iterative procedure in Shpitalni and Lipson's method is not necessary in order to find the maximum ranks in a graph. We assume that the degree of any vertex in a graph is larger than one. If there are some vertices with their degrees equal to one, there will be edge subsets that form parasitic trees, as shown in Fig. 5. Obviously, no faces pass through edges e_1 through e_5 and very simple algorithms can be developed to remove these edges. Note that the degree of v_2 is not equal to one at first, but becomes one after e_4 and e_5 are removed, after which e_3 is also removed.

Now, we show that if we use this equation

$$R(e) = \min\{d(v_1), d(v_2)\} - 1 \tag{11}$$

to compute $R(e)$ and then (9) to calculate $R(v)$, all the edge and vertex ranks will satisfy (7), (8), (9), and (10). In the following proofs, it is implied that $d(v) \geq 2$ for any vertex v in a graph; $\lfloor a(a-1)/2 \rfloor = a(a-1)/2$ if a is an integer; $\lfloor a/2 \rfloor = a/2$ if a is even; $\lfloor a/2 \rfloor = (a-1)/2$ if a is odd.

Theorem 1. *If $R(v)$ is obtained by (11) and (9), then $R(v) \leq \{d(v)[d(v)-1]\}/2$.*

Proof. Suppose there are n $(= d(v))$ edges meeting at vertex v (see Fig. 6). From (11), we have $R(e_i) \leq d(v) - 1$, $i = 1, 2, \ldots, n$. Thus,

$$R(v) = \left\lfloor \left[\sum_{i=1}^{n} R(e_i) \right] / 2 \right\rfloor \leq \lfloor \{n[d(v)-1]\}/2 \rfloor = \{d(v)[d(v)-1]\}/2. \quad \square$$

Lemma 1. *In a graph, if $d(v) \geq 2$, $\forall v$, then $R(e) \geq 1$, $\forall e$, where $R(e)$ is obtained by (11).*

This lemma comes directly from (11).

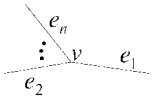

Fig. 6. n edges meeting at vertex v.

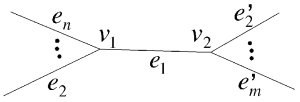

Fig. 7. Edges and vertices in the proof of Theorem 2.

Theorem 2. *If the edge and vertex ranks are obtained by (11) and (9), then $R(e) \leq \min\{R(v_1), R(v_2)\}$ for any edge e and its two end-vertices v_1 and v_2.*

Proof. Let the edges meeting at v_1 be $e_1, e_2, \ldots, e_n$ and the edges meeting at v_2 be $e_1, e_2', \ldots, e_m'$, as shown in Fig. 7. Then, $d(v_1) = n$ and $d(v_2) = m$. Without loss of generality, suppose

$$d(v_1) \leq d(v_2). \tag{12}$$

Then,

$$R(e_1) = \min\{d(v_1), d(v_2)\} - 1 = d(v_1) - 1 = n - 1. \tag{13}$$

We consider two cases.

Case 1. Suppose $R(v_1) \leq R(v_2)$. We now show $R(e_1) \leq R(v_1)$.

Case 1.1. If $\sum_{i=1}^{n} R(e_i)$ is even, then

$$R(v_1) = \left[R(e_1) + \sum_{i=2}^{n} R(e_i) \right] / 2.$$

By Lemma 1, it is easy to see that

$$\sum_{i=2}^{n} R(e_i) \geq n - 1 = R(e_1).$$

Thus, $R(e_1) \leq R(v_1)$.

Case 1.2. If $\sum_{i=1}^{n} R(e_i)$ is odd, then

$$R(v_1) = \left[R(e_1) + \sum_{i=2}^{n} R(e_i) - 1 \right] / 2.$$

By Lemma 1, we know $R(e_i) \geq 1$, $i = 2, 3, \ldots, n$, but we claim that it is impossible here that $R(e_i) = 1$, $i = 2, 3, \ldots, n$. Assume, to the contrary, that

$$R(e_2) = R(e_3) = \cdots = R(e_n) = 1.$$

Then,

$$R(v_1) = [R(e_1) + n - 2]/2 = R(e_1) - 1/2,$$

which indicates that $R(v_1)$ is not an integer and, thus, produces a contradiction. Therefore, there is at least one $R(e_k)$, $2 \leq k \leq n$, such that $R(e_k) \geq 2$, which results in $\sum_{i=2}^{n} R(e_i) \geq n$. Thus, it follows that

$$R(v_1) \geq [R(e_1) + n - 1]/2 = R(e_1).$$

Case 2. Suppose $R(v_1) > R(v_2)$. We now verify $R(e_1) \leq R(v_2)$.

Case 2.1. If $R(e_1) + \sum_{i=2}^{m} R(e_i')$ is even, we have, by Lemma 1,

$$R(v_2) = \left[R(e_1) + \sum_{i=2}^{m} R(e_i') \right] / 2 \geq [R(e_1) + d(v_2) - 1]/2.$$

From (12) and (13), it follows that $R(v_2) \geq R(e_1)$.

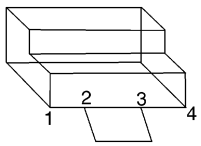

Fig. 8. A smooth entity chain $(1, 2, 3, 4)$, where three faces pass through edge $(2, 3)$.

Case 2.2. If $R(e_1) + \sum_{i=2}^{m} R(e_i')$ is odd, then

$$R(v_2) = [R(e_1) + \sum_{i=2}^{m} R(e_i') - 1]/2.$$

From (12) and (13), we have $h_1 = d(v_2) - R(e_1) \geq 1$ and, by Lemma 1, $h_i = R(e_i') \geq 1$, $i = 2, 3, \ldots, m$. However, we claim that it is impossible that $h_i = 1, i = 1, 2, \ldots, m$. Assume, to the contrary, that $h_1 = h_2 = \cdots = h_m = 1$. Then, we have

$$R(v_2) = [R(e_1) + m - 2]/2 = [R(e_1) + d(v_2) - 2]/2 = R(e_1) - 1/2,$$

which indicates that $R(v_2)$ is not an integer and produces a contradiction. Therefore, there exists at least one h_k, $1 \leq k \leq m$, such that $h_k \geq 2$. Then,

$$\begin{aligned} R(v_2) &= \Big[R(e_1) - 1 + \sum_{i=2}^{m} h_i\Big]/2 \\ &= \Big[R(e_1) - 1 - h_1 + \sum_{i=1}^{m} h_i\Big]/2 \\ &\geq \Big[R(e_1) - 1 - h_1 + m + 1\Big]/2 \\ &= \Big[R(e_1) - d(v_2) + R(e_1) + m\Big]/2 = R(e_1), \end{aligned}$$

which completes the proof. □

Theorems 1 and 2 show that the edge and vertex ranks calculated by (11) and (9) satisfy conditions (7), (8), (9), and (10) and, thus, they are the maximum ranks. We do not need the iterative procedure in [1] to obtain them.

When two edges meeting at a vertex are collinear in a graph, there may be two or more faces sharing the two edges in a smooth entity chain, as the smooth entity chain $(1, 2, 3, 4)$ in Fig. 8, where three faces pass through edge $(2, 3)$. In this case, Shpitalni and Lipson extended (8) to

$$R(e) \leq \min\Big\{\sum d(v_L) - 2n_L, \sum d(v_R) - 2n_R\Big\} + 1, \quad (14)$$

where n_L (n_R) is the number of vertices on the left (right) of edge e and v_L (v_R) denote all the n_L (n_R) vertices along the smooth entity chain on the left (right) of edge e. Then, (14) is used to replace (8) in order to find the maximum edge and vertex ranks in the iterative procedure. Here, we follow Shpitalni and Lipson, using "left" and "right" to denote the two directions along a smooth entity chain.

We can also prove that the iterative procedure is unnecessary in this case if we use this equation

$$R(e) = \min\Big\{\sum d(v_L) - 2n_L, \sum d(v_R) - 2n_R\Big\} + 1 \quad (15)$$

to compute $R(e)$ and then use (9) to calculate $R(v)$.

Theorem 3. *If the edge and vertex ranks are obtained by (15) and (9), then $R(e) \leq \min\{R(v_1), R(v_2)\}$ for any edge e and its two end-vertices v_1 and v_2.*

The proof of Theorem 3 is similar to that of Theorem 2. We do not present it here because of space constraints. The interested reader can find the proof in our technical report [9].

Theorem 3 indicates that the edge and vertex ranks calculated by (15) and (9) are the maximum ranks and, thus, Shpitalni and Lipson's iterative procedure to find them is not necessary. More importantly, the theorems in this section allow us to reduce SLOP to MWCP.

4.2 Reduction of SLOP to MWCP

In this section, the maximum edge and vertex ranks calculated from a graph G are denoted by $R^+(e)$ and $R^+(v)$, respectively, and the edge and vertex ranks computed from some subset x of minimal potential faces are denoted by $R(e)$ and $R(v)$, respectively. Now, we show that SLOP presented in (1), (2), (3), and (4) is redundant and can be replaced by MWCP.

Lemma 2. *Let x be a subset of the minimal potential faces of a graph. If the faces in x satisfy the constraint in (4), then 1) $R(e) \leq R^+(e), \forall e$ and 2) $R(v) \leq R^+(v), \forall v$.*

Proof.

1. Both the constraint in (4) and the inequality in (14) stem from the face adjacency theorem. Shpitalni and Lipson defined the general maximum edge rank inequality in (14) and used the right side of it to be the initial value in their iterative procedure for finding $R^+(e)$ and $R^+(v)$. However, we have proven that the iterative procedure cannot reduce this initial value. Therefore, it is equal to $R^+(e)$, which is the upper bound for allowing the most faces to pass through edge e according to the face adjacency theorem. Therefore, if the constraint in (4) is satisfied by the faces in x, it is impossible that more than $R^+(e)$ faces in x pass through e. So, $R(e) \leq R^+(e), \forall e$.
2. Let all the edges meeting at vertex v be $e_1, e_2, \ldots, e_m$. Since a face in x passing through v also passes through two of these edges, we obtain $2R(v) = \sum_{i=1}^{m} R(e_i)$. From the result in 1, we have $R(e_i) \leq R^+(e_i)$, $i = 1, 2, \ldots, m$. Thus,

$$2R(v) \leq \sum_{i=1}^{m} R^+(e_i). \quad (16)$$

If $\sum_{i=1}^{m} R^+(e_i)$ is even, then from (9),

$$2R^+(v) = \sum_{i=1}^{m} R^+(e_i).$$

So, $R(v) \leq R^+(v)$. If $\sum_{i=1}^{m} R^+(e_i)$ is odd, again from (9), we have

$$2R^+(v) = \sum_{i=1}^{m} R^+(e_i) - 1.$$

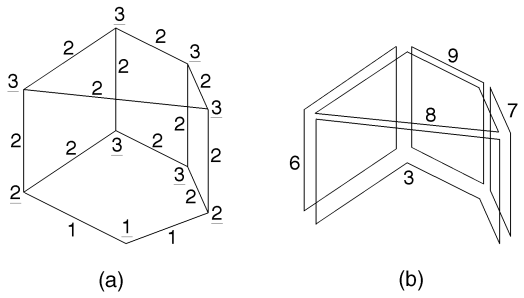

Fig. 9. (a) A drawing with its maximum ranks shown, where the maximum vertex ranks are denoted by underlined numbers and the maximum edge ranks by the other numbers. (b) A subset of the minimal potential faces of the graph in (a).

Since $2R(v)$ is even, it follows from (16) that $2R(v) \leq \sum_{i=1}^{m} R^+(e_i) - 1$. Hence, $R(v) \leq R^+(v)$. □

Theorem 4. *SLOP in (1), (2), (3), and (4) is equivalent to MWCP in (5) and (6).*

Proof. Let x be a subset of the minimal potential faces of a graph G. By Lemma 2 and the definitions of the constraints in SLOP and MWCP, we can see that, for the same x, the constraints in SLOP are satisfied if and only if the constraint in MWCP is satisfied.

Since no two vertices of a face are the same, the edge number and the vertex number of any face are equal. Thus, for some x, $\sum R(e) = \sum R(v)$, and $g(x)$ in (1) can be rewritten as $g(x) = \sum R^+(e) + \sum R^+(v) - 2\sum R(e)$. Here, $\sum R^+(e) + \sum R^+(v)$ is a constant for G and $\sum R(e) = f(x) = \sum_{i \in x} w(i)$. It is easy to see that $g(x)$ is minimum if and only if $f(x)$ is maximum. Therefore, SLOP and MWCP are equivalent. □

The constraint in (4), coming from the face adjacency theorem, implies the constraints in (2) and (3) by Lemma 2. However, that the faces in x satisfy (2) and (3) does not mean that they must satisfy the constraint in (4) too. Fig. 9 gives such an example. The drawing in Fig. 4a with its maximum vertex and edge ranks is shown in Fig. 9a. A subset, $x = \{3, 6, 7, 8, 9\}$, of the minimal potential faces in Fig. 4b are illustrated in Fig. 9b. It can be seen that, for x, the constraints in (2) and (3) are satisfied, but the constraint in (4) is not because $b_{36} = b_{37} = 0$.

5 Two Algorithms in the Face Identification Problem

We provide two algorithms in this section for two key steps in the face identification problem. The first is a depth-first search algorithm for generating the set of minimal potential faces of a drawing. The second is a maximum weight clique finding algorithm for choosing the best face configurations from these minimal potential faces.

5.1 A Depth-First Search Algorithm

Depth-first search on a graph can generate all the circuits of the graph and several such algorithms are available [6]. However, from the example (Fig. 3) given in Section 2, we have seen that the minimal potential faces of interest are much fewer than the circuits. An algorithm enumerating all the circuits first and then finding the minimal potential faces is not efficient for face identification. In this section, we present a depth-first search algorithm with two pruning operations derived from two conditions of being a minimal potential face: a circuit without self-intersecting edges and without any other edge in a drawing which connects any two nonadjacent vertices of the circuit [1].

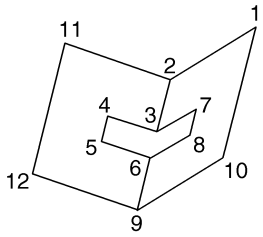

Fig. 10. A wireframe object where circuit $(1, 2, 3, 4, 5, 6, 9, 10, 1)$ encloses circuits $(3, 4, 5, 6, 8, 7, 3)$ and $(1, 2, 3, 7, 8, 6, 9, 10, 1)$, but cannot be deleted.

For a circuit with an edge connecting its two nonadjacent vertices, such as the circuit $c_1 = (12, 15, 18, 20, 19, 21, 12)$ and the edge $(15, 19)$ in Fig. 3, c_1 cannot coexist in the same object with the two enclosed smaller circuits $c_2 = (12, 15, 19, 21, 12)$ and $c_3 = (15, 18, 20, 19, 15)$ according to the face adjacency theorem, but c_2 and c_3 can since they only have one common edge. Because the number of edges of c_2 and c_3 is larger than that of c_1, the optimization process in Step 5 of Shpitalni and Lipson's method will tend to select c_2 and c_3 as the object's faces and discard c_1. It is worth noting that not every circuit that encloses two others can be deleted, as in the case shown in Fig. 10, where the circuit $c_1 = (1, 2, 3, 4, 5, 6, 9, 10, 1)$ encloses circuits $c_2 = (3, 4, 5, 6, 8, 7, 3)$ and $c_3 = (1, 2, 3, 7, 8, 6, 9, 10, 1)$. Though c_1 cannot coexist with c_2 and c_3, c_2 and c_3 cannot coexist with each other either. So, all three of them need to be kept in the set of minimal potential faces.

Now, let us see why it is beneficial to combine the two conditions of being a minimal potential face into depth-first search on a graph. For the graph in Fig. 3, consider the circuits starting from and ending at vertex 1:

$$(1, 7, 6, 4, 3, 5, 8, v_1, v_2, \ldots, v_m, 9, 2, 1),$$

where $v_1, v_2, \ldots, v_m \in \{10, 11, \ldots, 23\}$. Obviously, there are many different circuits obtainable by the permutations of some of the vertices $v_1, v_2, \ldots, v_m$. However, with the first condition, all these circuits are not potential faces because edges $(1, 7)$ and $(4, 3)$ intersect. Thus, we can stop the search algorithm when this fact is established. For the second condition, consider the circuits

$$(1, 4, 6, 7, 8, v'_1, v'_2, \ldots, v'_n, 9, 2, 1),$$

where $v'_1, v'_2, \ldots, v'_n \in \{10, 11, \ldots, 23\}$. Since there exists an edge $(1, 7)$ connecting the two nonadjacent vertices 1 and 7 in these circuits, they can be eliminated when the algorithm reaches vertex 7. Our algorithm for finding the set of minimal potential faces is provided below.

Algorithm 1. (Depth-first search with two pruning operations)

[To find the set of the minimal potential faces of a graph $G = (V, E)$ with the vertices numbered from 1 to $|V|$, given

the adjacency lists of G, $AdjList(v)$, $v = 1, 2, \ldots, |V|$, the adjacency matrix of G, and the intersecting matrix denoting whether any two edges of G intersect.]

```
1. index ← 0
2. for i = 1 to |V| do Label(i) ← 0
3. for i = 1 to |V| − 2 do
     begin
4.       start = i
5.       CIRCUIT(i)
6.       Update the adjacency lists AdjList(v),
             v = i + 1, i + 2, ..., |V|
             by deleting vertex i from them
     end
7. procedure CIRCUIT(v)
     begin
8.       index ← index + 1
9.       Label(v) ← 1
10.      Path(index) ← v
11.      for u ∈ AdjList(v) do
12.          if u = start and index ≥ 3 then
                 begin
13.                  if edge (u, v) intersects any edge in
                         the current path then goto 17
14.                  if Path(2) < Path(index) then
                         output the circuit consisting of
                         the vertices in Path followed
                         by u and goto 17
                 end
             else
15.                  if Label(u) = 0 and edge (u, v) does not
                         intersect any edge in the current
                         path and there is no edge in G
                         connecting any two nonadjacent
                         vertices in the current path,
                         then extend the path by calling
16.                      CIRCUIT(u)
17.      index ← index − 1
18.      Label(v) ← 0
19.  end of CIRCUIT
```

Now, we explain the above algorithm. An array $Path$ is used to keep the vertices in the current search path. Line 1 initializes a global variable $index$, the value of which gives the position of the last-added vertex in $Path$ during the search. Line 2 initializes a label $Label(v)$ for each vertex v. $Label(v) = 1$ indicates that vertex v is in $Path$ and 0 otherwise. In Lines 3, 4, 5, and 6, always beginning from the smallest numbered vertex i, the algorithm searches graph G for the minimal potential faces that start and end at that vertex. A global variable $start$ $(= i)$ is used to let the recursive procedure *CIRCUIT* know whether a circuit is found. After all such circuits are found, the adjacency lists of G are updated by deleting vertex i and then the search continues.

The main part of the algorithm is the recursive procedure *CIRCUIT*. In Line 12, the condition $index \geq 3$ guarantees a circuit consisting of at least three edges. The two pruning operations are presented in Lines 13 and 15. Whether an edge intersects any edges in the current path can be determined from the input intersecting matrix and the adjacency matrix of G can be employed to determine if there is an edge in G connecting any two nonadjacent vertices in the path. Note that if we ignore the condition $Path(2) < Path(index)$ in Line 14, we will obtain double the circuits, each of which exists twice in reverse orders, such as the circuits $(1, 2, 3, 4, 1)$ and $(1, 4, 3, 2, 1)$ in Fig. 3.

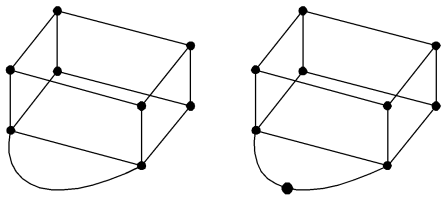

Fig. 11. (a) A drawing with parallel edges. (b) The drawing without parallel edges generated by adding to the left drawing a new vertex at the middle of the curved edge.

The above algorithm cannot be directly applied to a special kind of drawings in which there exist "parallel" edges that join pairs of vertices, as shown in Fig. 11. An adjacency matrix is not suitable for representing such a graph. However, the algorithm works well when we add a new vertex at the middle of the curved edge before search. The efficiency of this algorithm over the circuit space method will be demonstrated in Section 6.

5.2 A Maximum Weight Clique Finding Algorithm

As we have mentioned in Section 3, MWCP is an extension of MCP, the well-known maximum clique problem in graph theory. Unfortunately, MCP is NP-complete, meaning that there are no polynomial algorithms for it so far [10]. If all the weights of the vertices of a weighted graph are equal to one, MWCP reduces to MCP. This indicates that MWCP is also NP-complete. However, for face identification, it is still possible to develop much faster algorithms for MWCP than the A* algorithm for SLOP.

Many algorithms for MWCP have been proposed in the literature. Bomze et al. provided a comprehensive survey of these algorithms in [11]. We have not tested the performance of these algorithms against ours presented in the following. However, our algorithm is easy to understand and implement, can give all the maximum weight cliques when multiple solutions exist in a weighted graph, and is much faster for MWCP than the A* algorithm for SLOP.

Carraghan and Pardalos [12] proposed an exact algorithm for MCP, using "partial enumeration." Their computational results show that the algorithm can solve very large MCP. In addition, they claimed that their algorithm was faster than any previously known method for MCP when tackling randomly generated graphs. Carraghan and Pardalos's algorithm cannot be applied to MWCP, but their scheme of partial enumeration is helpful for our developing an algorithm for MWCP.

Now, we describe how our algorithm works through an example: finding the maximum weight cliques of the weighted graph in Fig. 4c. We emphasize again that in face identification, all the cliques with the maximum weight are

required to be found. Here are several variables used in Table 1 for the description of the algorithm:

- $W(i)$: The weight of vertex i.
- ns: The number of solutions.
- $CMWC(i)$: An array to store the ith current maximum weight clique found so far.
- cmw: The current maximum weight of a clique.
- ccw: The weight of the current clique.
- pvw: The sum of the weights of the vertices that can be added into the current clique (see the following explanation).

At first, cmw is set to 0 and the vertices are listed in the order of increasing degrees. We have observed in our experiments that this ordering makes the algorithm run faster. In the middle column of Table 1, each list of vertices corresponds to a "level." At level 1, the set $S_1 = \{v_1, v_2, \ldots, v_n\}$ of all the n vertices are listed. The algorithm first finds all the maximum weight cliques containing v_1 and then, from the set $S_2 = S_1 - \{v_1\}$, tries to find all the maximum weight cliques that contain v_2 and have larger weights than the cliques found previously, etc. The bold number (vertex) on the list at level 1 indicates that this vertex and those after it will be examined for possible better cliques (ones with larger weights). The list of vertices at level i ($i > 1$) are the vertices adjacent to and listed after the bold vertex at level $i - 1$.

Now, suppose the algorithm reaches level k. This means that it has found a clique of k vertices that are located from level 1 to level k and are marked bold in the table. As defined above, ccw is equal to the sum of weights of these vertices, and pvw is the sum of the weights of the vertices listed after the bold vertex (say, v_j) at level k. For example, at level 3 of Step 4 in Table 1, we have $v_j = 7$, $ccw = W(4) + W(6) + W(7)$, and $pvw = W(8) + W(9)$. If $ccw + pvw < cmw$, the cliques, found by further adding the vertices adjacent to and listed after v_j at level k, will not have larger weights than the current maximum weight clique. In this case, we can stop the algorithm doing such an invalid search. Note that the value of cmw becomes larger when a better clique is found, providing a higher threshold for the algorithm to prune the search. This pruning scheme can save a large amount of computational time when dealing with a large weighted graph. For the example in Table 1, at level 5 of Step 4, the algorithm finds the first maximum weight clique and then the second at level 5 of Step 5. These two cliques are the two solutions to the MWCP.

The algorithm is shown in Algorithm 2 in which, besides the variables defined above, l denotes the current level the algorithm is at; $first(l)$ and $last(l)$ are two indexes, denoting the $first(l)$th and $last(l)$th vertices at level l, respectively (the former corresponds to the bold one and the latter to the last one at some level in Table 1).

Algorithm 2. (Maximum weight clique finding)

[To find all the maximum weight cliques of a weighted graph $G = (V, E)$ with the vertices numbered from 1 to $|V|$]

1. Initialization: $cmw \leftarrow 0$ **and** $l \leftarrow 1$ **and** $first(l) \leftarrow 0$ **and** $last(l) \leftarrow |V|$
2. $first(l) \leftarrow first(l) + 1$
3. Calculate ccw and pvw
4. **if** $ccw + pvw < cmw$ **then**
5. **if** $l = 1$ **then** output ns cliques $CMWC(i)$, $1 \le i \le ns$ **and** stop
6. **else** $l \leftarrow l - 1$ **and goto** 2
7. **if** among the vertices listed after the $first(l)$th one at level l, there exist m (> 0) vertices that are adjacent to the $first(l)$th one in G **then** list these vertices at level $l + 1$ **and** $first(l + 1) \leftarrow 0$ **and** $last(l + 1) \leftarrow m$ **and** $l \leftarrow l + 1$ **and goto** 2
8. **if** $ccw > cmw$ **then** $cmw \leftarrow ccw$ **and** $ns \leftarrow 1$ **and** put the $first(1)$th, $first(2)$th, $\cdots$, $first(l)$th vertices in $CMWC(ns)$
9. **if** $ccw = cmw$ **then** $ns \leftarrow ns + 1$ **and** put the $first(1)$th, $first(2)$th, $\cdots$, $first(l)$th vertices in $CMWC(ns)$
10. **if** $l > 1$ **and** $first(l) = last(l)$ **then** $l \leftarrow l - 1$
11. **goto** 2

Before proceeding to the next section, we summarize our face identification method in the following four steps:

1. Find all the minimal potential faces from a drawing using the depth-first search algorithm.
2. Calculate the binary matrix B according to the face adjacency theorem, with the minimal potential faces.
3. Construct the weighted graph, and search for all the maximum weight cliques in this graph using the maximum weight clique finding algorithm.
4. Use image regularities to select the most plausible one if there is more than one solution.

Here, Steps 2 and 4 are, respectively, the same as Steps 3 and 6 in Shpitalni and Lipson's method presented in Section 2. These two steps can be implemented easily. The reader can find more details in [1].

6 Experimental Results

In this section, we present a set of examples to illustrate and compare the face identification by our method and Shpitalni and Lipson's method. As expected, for every drawing, the faces found by the two methods are the same because of the equivalence between these two methods. However, our algorithm runs much faster when dealing with objects of many faces. The comparisons of computational efficiency between the two algorithms are emphasized here.

Shpitalni and Lipson's method consists of six steps, while ours consists of four. Each step in either method is carried out by a separate algorithm. The majority of computational time in Shpitalni and Lipson's method is consumed by the algorithms for generating the minimal potential faces and the A* algorithm. Thus, for each drawing, we give three times taken in generating the minimal potential faces, finding the optimal face configurations, and carrying out all the steps in each method. For conciseness, the algorithms to perform Steps 1 and 2 in Shpitalni and Lipson's method are called "CircuitSpace" and "A*" stands for the A* algorithm in Step 5. The whole algorithm to implement Steps 1, 2, 3, 4, 5, and 6 is called "S&L." Similarly, "DFS," "MWCF," and "L&L" are short, respectively, for the depth-first search algorithm, the maximum weight clique finding algorithm, and the whole

TABLE 1
The Steps and Explanation of Our Algorithm for Finding the Maximum Weight Cliques of the Weighted Graph in Fig. 4c

1.	Level 1: **1** 2 3 4 5 6 7 8 9	Since vertex 1 has no adjacent vertices listed after it, the algorithm completes the search rooted at it. So, $ns \leftarrow 1$, $CMWC(ns) \leftarrow \{1\}$, $cmw \leftarrow W(1)$.
2.	Level 1: 1 **2** 3 4 5 6 7 8 9	Since vertex 2 has no adjacent vertices after it, the algorithm completes the search rooted at it. Since $W(2) = cmw$, do: $ns \leftarrow ns + 1$, $CMWC(ns) \leftarrow \{2\}$.
3.	Level 1: 1 2 **3** 4 5 6 7 8 9	Since $\sum_{i=3}^{9} W(i) \geq cmw$ and vertex 3 has adjacent vertices after it, go down to the next level.
	Level 2: **8** 9	List all the adjacent vertices after vertex 3 at level 1. Since $W(3) + \sum_{i=8}^{9} W(i) \geq cmw$ and vertex 8 has an adjacent vertex after it, go down.
	Level 3: **9**	The algorithm reaches the bottom. Since $ccw = W(3) + W(8) + W(9) = 14 > cmw = 7$, update cmw: $cmw \leftarrow ccw$. Keep the new clique and discard the previous two: $ns \leftarrow 1, CMWC(ns) \leftarrow \{3, 8, 9\}$. Go up.
	Level 2: 8 **9**	Since $W(3) + W(9) < cmw = 14$, go up.
4.	Level 1: 1 2 3 **4** 5 6 7 8 9	Since $\sum_{i=4}^{9} W(i) \geq cmw$ and vertex 4 has adjacent vertices after it, go down.
	Level 2: **6** 7 8 9	List all the adjacent vertices after vertex 4 at level 1. Since $W(4) + \sum_{i=6}^{9} W(i) \geq cmw$ and vertex 6 has adjacent vertices after it, go down.
	Level 3: **7** 8 9	List all the adjacent vertices after vertex 6 at level 2. Since $W(4) + W(6) + \sum_{i=7}^{9} W(i) \geq cmw$ and vertex 7 has adjacent vertices after it, go down.
	Level 4: **8** 9	List all the adjacent vertices after vertex 7 at level 3. Since $W(4) + W(6) + W(7) + \sum_{i=8}^{9} W(i) \geq cmw$ and vertex 8 has an adjacent vertex after it, go down.
	Level 5: **9**	The algorithm reaches the bottom. Since $ccw = W(4) + W(6) + W(7) + W(8) + W(9) = 21 > cmw = 14$, update cmw: $cmw \leftarrow ccw$. Keep the new clique and discard the previous one: $ns \leftarrow 1$, $CMWC(ns) \leftarrow \{4, 6, 7, 8, 9\}$. Go up.
	Level 4: 8 **9**	Since $W(4) + W(6) + W(7) + W(9) < cmw$, go up.
	Level 3: 7 **8** 9	Since $W(4) + W(6) + \sum_{i=8}^{9} W(i) < cmw$, go up.
	Level 2: 6 **7** 8 9	Since $W(4) + \sum_{i=7}^{9} W(i) < cmw$, go up.
5.	Level 1: 1 2 3 4 **5** 6 7 8 9	Since $\sum_{i=5}^{9} W(i) \geq cmw = 21$ and vertex 5 has adjacent vertices after it, go down.
	Level 2: **6** 7 8 9	List all the adjacent vertices after vertex 5 at level 1. Since $W(5) + \sum_{i=6}^{9} W(i) \geq cmw$ and vertex 6 has adjacent vertices after it, go down.
	Level 3: **7** 8 9	List all the adjacent vertices after vertex 6 at level 2. Since $W(5) + W(6) + \sum_{i=7}^{9} W(i) \geq cmw$ and vertex 7 has adjacent vertices after it, go down.
	Level 4: **8** 9	List all the adjacent vertices after vertex 7 at level 3. Since $W(5) + W(6) + W(7) + \sum_{i=8}^{9} W(i) \geq cmw$ and vertex 8 has an adjacent vertex after it, go down.
	Level 5: **9**	The algorithm reaches the bottom. Since $ccw = W(5) + W(6) + W(7) + W(8) + W(9) = cmw = 21$, Keep the new clique: $ns \leftarrow ns + 1$, $CMWC(ns) \leftarrow \{5, 6, 7, 8, 9\}$. Go up.
	Level 4: 8 **9**	Since $W(5) + W(6) + W(7) + W(9) < cmw$, go up.
	Level 3: 7 **8** 9	Since $W(5) + W(6) + \sum_{i=8}^{9} W(i) < cmw$, go up.
	Level 2: 6 **7** 8 9	Since $W(5) + \sum_{i=7}^{9} W(i) < cmw$, go up.
6.	Level 1: 1 2 3 4 5 **6** 7 8 9	Since $\sum_{i=6}^{9} W(i) = 16 < cmw = 21$ and the algorithm is at level 1, the entire search is completed.

algorithm to carry out Steps 1, 2, 3, and 4 in our method. All the algorithms are implemented using Visual C++, running on a 300 MHz Pentium II PC.

Fig. 12 shows six objects, each together with the faces found by L&L or S&L. Obviously, the faces found accord with human interpretation of the drawings. Each object except the first one in the sequence has two more faces than its previous one. The experiments on these objects can reflect how the computational times taken by the two methods vary with the number of faces of an object.

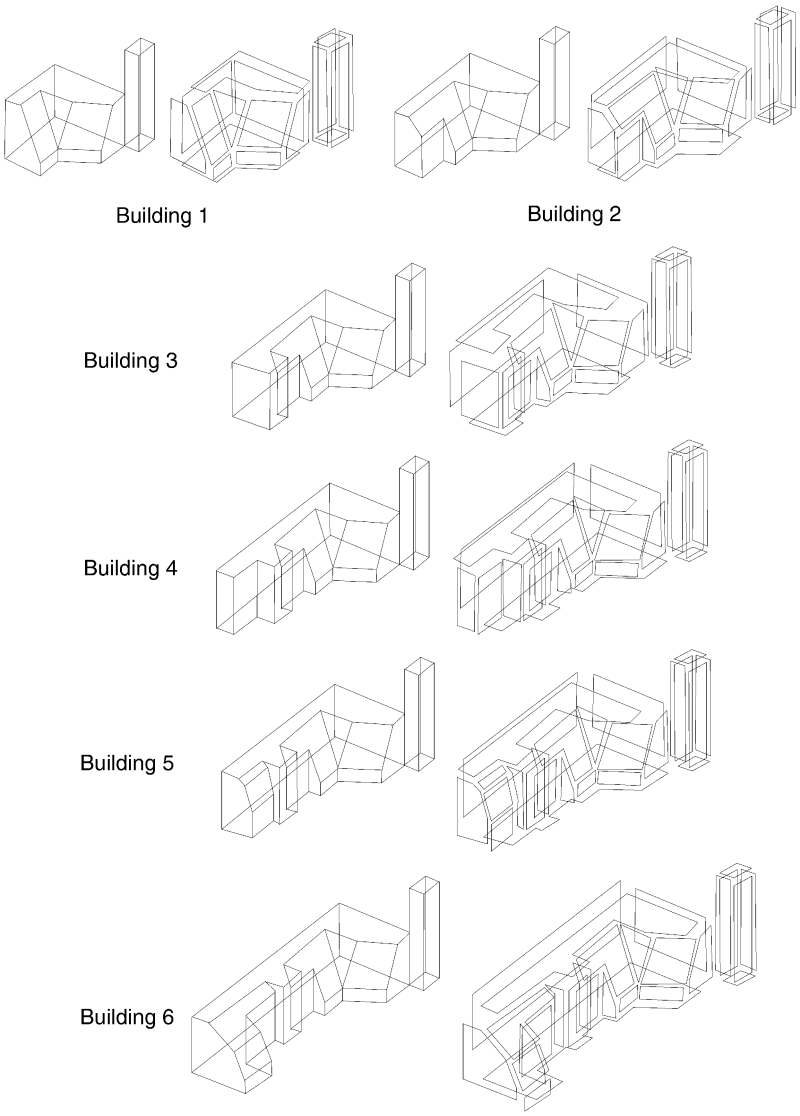

Fig. 12. Six objects each with the faces found by L&L or S&L. One solution is obtained for each object after MWCP or SLOP is solved.

Table 2 gives the results for the six objects in Fig. 12. From it, we can see that the number of circuits (nc) grows exponentially with the number of faces (nf) selected finally. Examining the data in the table, we can derive an approximate relation between nc and nf: $nc \approx 3185 \times 3.3^{(nf-17)/2}$. Fortunately, the increase in the number of minimal potential faces is much slower. In Shpitalni and Lipson's method, when the time taken by CircuitSpace or A* is more than 10 seconds for an object, the time consumed in Steps 3, 4, and 6, if less than 1 second, is ignored. It is shown in the table that both CircuitSpace and A* are time-consuming. For Building 6, S&L requires more than 1 hour, while L&L only takes less than 1 second. It is clear that our algorithm runs much faster.

Fig. 13 shows another three objects and the faces found by L&L or S&L. All three objects have more than 20 faces. Note that "Solid object" has two curved faces. The computational results are presented in Table 3. Again, we can see that our algorithms are much more efficient than Shpitalni and Lipson's.

We also tested the algorithms on all the objects given in Shpitalni and Lipson's experiments in [1]. The results are summarized in Table 4. For three of the objects, MWCF or A* gives more than one solution, but the last step in L&L or S&L selects one face configuration for each object. The faces found accord with human interpretation of the drawings. The drawings and the face configurations selected finally can be found in [1]. From Table 4, we can see that Shpitalni and Lipson's algorithms do not require much time for objects with less than 20 faces.

Comparing Table 4 with Table 1 in [1], the reader may notice that Shpitalni and Lipson's gave fewer potential faces and minimal potential faces for all the objects except the last two. For example, for the object BWP (building with parking), Shpitalni and Lipson gave 194 potential faces and 53 minimal potential faces, while we obtain 3,057 potential faces and 102 minimal potential faces. We believe that Shpitalni and Lipson missed some of them. It is possible that different numbers of potential faces and minimal potential faces are obtained from different

TABLE 2
Results for the Objects in Fig. 12

	Buildings 1–6					
	1	2	3	4	5	6
Vertices/Edges	23/37	27/43	31/49	35/55	39/61	43/67
Circuits	3185	10055	33347	122719	371547	1297089
Potential Faces	734	2053	3056	9355	27918	40500
Minimal Potential Faces	54	89	101	189	332	431
Selected Faces	17	19	21	23	25	27
CircuitSpace	0.95	4.7	22	102	480	2694
A*	1.10	4.4	15	74	347	1642
S&L	2.24	9.3	37	176	827	4336
DFS	0.01	0.01	0.02	0.06	0.12	0.21
MWCF	0.16	0.18	0.18	0.20	0.22	0.24
L&L	0.36	0.38	0.40	0.46	0.66	0.83

The CPU time (seconds) taken by each algorithm is shown in the last six rows.

projections of a 3D object because of different intersections between edges of the projections. However, in our experiments, we made sure that our drawings are as similar to Shpitalni and Lipson's as possible. Consider the object BWP again. If the part "parking" in BWP (see [1]) is added into the lowest vertex in Building 3 in Fig. 12, Building 3 with the parking will present the same edge intersections as BWP. But, we obtain 3,056 potential faces and 101 minimal potential faces from Building 3 (without the parking). In the Appendix, we list all the minimal potential faces of Building 1 for verification. Building 3 has more vertices and edges than Building 1 and is an extension of Building 1. We can see that there are 54 minimal potential faces even in Building 1.

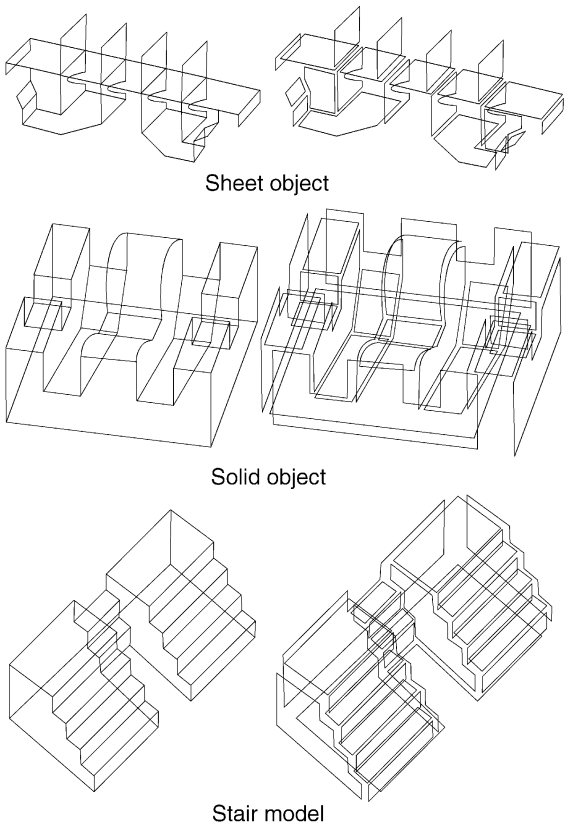

Fig. 13. Three objects and their faces found by L&L or S&L. One solution is obtained for each object after MWCP or SLOP is solved.

7 Conclusions

A graph-based method for face identification from single 2D line drawings has been presented. The method is proven to be equivalent to Shpitalni and Lipson's method. However, our formulation of the face identification enables us to develop a much faster algorithm, L&L, for obtaining faces from drawings. Two separate algorithms, DFS and MWCF contained in L&L, are provided to generate all the minimal potential faces in a line drawing and find the optimal face configurations from it, respectively. DFS uses two pruning operations to speed up the search and is problem-dependent, but MWCF can be applied to any maximum weight clique finding problem.

All the drawings in Shpitalni and Lipson's experiments and some new drawings are used to compare our and Shpitalni and Lipson's algorithms. For every drawing, the

TABLE 3
Results for the Objects in Fig. 13

	Sheet object	Solid object	Stair model
Vertices/Edges	56/80	48/72	48/72
Circuits	48928	3735714	3812028
Potential Faces	1652	14579	10791
Minimal Potential Faces	78	657	504
Selected Faces	23	26	26
CircuitSpace	3065	2781	2990
A*	383	693	1012
S&L	3448	3474	4002
DFS	0.08	0.29	0.42
MWCF	0.18	0.24	0.30
L&L	0.46	0.97	1.10

The CPU time (seconds) taken by each algorithm is shown in the last six rows.

TABLE 4
Results for the Objects in Shpitalni and Lipson's Experiments

	BWP	CWPH	SMPWH	NTCM	SMP	AM	OM
Vertices/Edges	49/68	20/30	57/70	30/46	28/39	16/24	16/24
Circuits	33348	1302	1543	22374	702	224	305
Potential Faces	3357	112	525	1127	205	37	21
Minimal Potential Faces	102	52	41	86	42	19	15
Selected Faces	22	10	11	18	11	10	10
Solutions	1	4	4	1	1	2	1
CircuitSpace	50	0.04	1.00	5.00	0.15	0.01	0.01
A*	27	0.22	0.31	1.45	0.21	0.18	0.18
S&L	77	0.42	1.49	6.64	0.52	0.35	0.35
DFS	0.02	0.01	0.01	0.01	0.01	0.01	0.01
MWCF	0.18	0.18	0.16	0.18	0.18	0.18	0.17
L&L	0.40	0.35	0.35	0.38	0.35	0.35	0.34

Here, the object names BWP, CWPH, SMPWH, NTCM, SMP, AM, and OM are short, respectively, for building with parking, cube with piercing hole, sheet metal product with holes, nontrihedral concave manifold, sheet metal product, ambiguent manifold, and orthogonal manifold in [1]. The CPU time (seconds, on a 300 MHz Pentium II PC) taken by each algorithm is shown in the last six rows.

two methods can find the same faces that accord with human interpretation of the drawing, but our algorithm runs much faster when handling an object of more than 20 faces.

The complexities of DFS and MWCF are not polynomial. As the faces of an object increases to a large number (e.g., 50), a nonpolynomial algorithm will suffer from too much computation time. To alleviate this problem, a possible solution is to find an approach that can divide the large graph of an object into several small ones, to each of which our algorithms can be applied efficiently.

TABLE 5
All the Minimal Potential Faces in the Object Building 1

1. **1 2 3 4 1**
2. 1 2 3 5 6 7 1
3. 1 2 9 8 5 6 4 1
4. **1 2 9 8 7 1**
5. **1 4 6 7 1**
6. 2 3 4 6 7 8 9 2
7. **2 3 5 8 9 2**
8. **3 4 6 5 3**
9. **5 6 7 8 5**
10. **8 9 10 11 8**
11. 8 9 10 22 21 12 15 16 17 8
12. 8 9 10 22 21 12 15 18 17 8
13. 8 9 10 22 21 19 15 16 17 8
14. 8 9 10 22 21 19 15 18 17 8
15. 8 9 10 22 21 19 20 18 17 8
16. 8 9 10 22 23 20 18 17 8
17. 8 9 10 22 23 20 19 15 16 17 8
18. 8 9 14 13 12 21 19 20 18 17 8
19. 8 9 14 13 12 21 22 23 11 8
20. 8 9 14 13 12 21 22 23 20 18 17 8
21. 8 9 14 13 15 18 17 8
22. **8 9 14 16 17 8**
23. 8 11 10 22 21 12 15 16 17 8
24. 8 11 10 22 21 12 15 18 17 8
25. 8 11 10 22 21 19 15 16 17 8
26. 8 11 10 22 21 19 15 18 17 8
27. 8 11 10 22 21 19 20 18 17 8
28. **8 11 23 20 18 17 8**
29. 8 11 23 20 19 15 16 17 8
30. 9 10 11 23 20 18 15 13 14 9
31. 9 10 11 23 20 18 15 16 14 9
32. 9 10 11 23 20 18 17 16 14 9
33. 9 10 11 23 20 19 15 13 14 9
34. 9 10 11 23 20 19 15 16 14 9
35. **9 10 22 21 12 13 14 9**
36. 9 10 22 21 12 15 16 14 9
37. 9 10 22 21 19 15 13 14 9
38. 9 10 22 21 19 15 16 14 9
39. 9 10 22 21 19 20 18 17 16 14 9
40. 9 10 22 23 20 18 15 13 14 9
41. 9 10 22 23 20 18 15 16 14 9
42. 9 10 22 23 20 18 17 16 14 9
43. 9 10 22 23 20 19 15 13 14 9
44. 9 10 22 23 20 19 15 16 14 9
45. **10 11 23 22 10**
46. 12 13 14 16 17 18 20 19 21 12
47. 12 13 14 16 17 18 20 23 22 21 12
48. **12 13 15 12**
49. 12 15 18 20 23 22 21 12
50. **12 15 19 21 12**
51. **13 14 16 15 13**
52. **15 16 17 18 15**
53. **15 18 20 19 15**
54. **19 20 23 22 21 19**

APPENDIX

ALL THE MINIMAL POTENTIAL FACES IN THE Object BUILDING 1

The object Building 1 in Fig. 12 is the same as the object in Fig. 3, where the label of every vertex is given. In Table 5, the numbers are vertex labels and each list is a closed loop of vertices that form a minimal potential face. None of the faces contains any other two faces that share only one common edge. The faces found by L&L or S&L are highlighted in bold.

REFERENCES

[1] M. Shpitalni and H. Lipson, "Identification of Faces in a 2D Line Drawing Projection of a Wireframe Object," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 18, no. 10, pp. 1000-1012, Oct. 1996.

[2] H. Lipson and M. Shpitalni, "Optimization-Based Reconstruction of a 3D Object from a Single Freehand Line Drawing," *Computer-Aided Design,* vol. 28, no. 8, pp. 651-663, 1996.

[3] Y.G. Leclerc and M.A. Fischler, "An Optimization-Based Approach to the Interpretation of Single Line Drawings as 3D Wire Frames," *Int'l J. Computer Vision,* vol. 9, no. 2, pp. 113-136, 1992.

[4] T. Marill, "Emulating the Human Interpretation of Line-Drawings as Three-Dimensional Objects," *Int'l J. Computer Vision,* vol. 6, no. 2, pp. 147-161, 1991.

[5] A. Gibbons, *Algorithmic Graph Theory.* New York: Cambridge Univ. Press, 1985.

[6] P. Mateti and N. Deo, "On Algorithms for Enumerating All Circuits of a Graph," *SIAM J. Computing,* vol. 5, no. 1, pp. 90-99, 1976.

[7] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Reading, Mass.: Addison-Wesley, 1984.

[8] B. Zhang and L. Zhang, "The Comparison between the Statistical Heuristic Search and A*," *J. Computer Science and Technology,* vol. 4, no. 2, pp. 126-132, 1989.

[9] J. Liu and Y.T. Lee, "A Graph-Based Method for Face Identification from a Single 2D Line Drawing," Technical Memo DRC-TM01, School of Mechanical and Production Eng., Nanyang Technological Univ., Singapore, 2000.

[10] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Englewood Cliffs, N.J.: Prentice Hall, 1982.

[11] I.M. Bomze, M. Budinich, P.M. Pardalos, and M. Pelillo, "The Maximum Clique Problem," *Handbook of Combinatorial Optimization (Supplement Volume A),* D.-Z. Du and P. M. Pardalos, eds., pp. 1-74. Boston: Kluwer Academic, 1999.

[12] R. Carraghan and P.M. Pardalos, "An Exact Algorithm for the Maximum Clique Problem," *Operations Research Letters,* vol. 9, no. 6, pp. 375-382, 1990.

Jianzhuang Liu received the BE degree from Nanjing Institute of Posts & Telecommunications, China, in 1983, the ME degree from Beijing University of Posts & Telecommunications, China, in 1987, and the PhD degree from the Chinese University of Hong Kong, China, in 1997. From 1987 to 1994, he was a member of the academic staff in the Department of Electronic Engineering, Xidian University, China. From August 1998 to August 2000, he was a research fellow at the School of Mechanical and Production Engineering, Nanyang Technological University, Singapore. He is now a postdoctoral fellow in the Department of Electronic Engineering, the Chinese University of Hong Kong, China. His research interests include image processing, computer vision, pattern recognition, and artificial intelligence.

Yong Tsui Lee received the BSc degree (1977) from the University of Leeds, the MS degree (1980) from the University of Rochester, and the PhD degree (1983) from the University of Leeds. He worked as a CAD system developer in England after completing the PhD degree until 1991, when he joined Nanyang Technological University in Singapore, where he is now an associate professor. His current research interests are in geometric modeling, reverse engineering, sketch input for CAD systems, and, generally, 3D data capture for design. He is also working in the development of rapid prototyping technology. He is a member of the IEEE Computer Society.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** http://computer.org/publications/dlib.