Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

L.M. Kirousis

On effectively labeling planar projections of polyhedra

Computer Science/Department of Algorithmics & Architecture Report CS-R8715 March

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

On Effectively Labeling Planar Projections of Polyhedra

Lefteris M. Kirousis

Department of Mathematics, University of Patras, Patras, Greece
and
Center for Mathematics and Computer Science, Amsterdam, The Netherlands

A well-known method for interpreting planar projections (images) of 3-dimensional polyhedra is to label their lines by the Clowes-Huffman scheme. However, the question of whether there is such a labeling has been shown to be NP-complete. In this paper a linear in time algorithm is given that answers the labelability question under the assumption that some information is known about those edges of the polyhedron whose both faces are visible. In many cases, this information can be derived from the image itself. Moreover, the algorithm has an effective parallel version, i.e., with polynomially many processors it can be executed in time polynomial in $\log n$.

1980 Mathematics Subject Classification (1985 Revision), Primary: 68T10 *Secondary:* 51M20, 68Q20, 68Q25, 68R10.
CR Classification System (1987 Version): I.2.10, F.2.2, G.2.2.
Keywords and Phrases: Polyhedron, Labeling, 2-dimensional image of a polyhedron.

1. Introduction.

Interpreting 2-dimensional figures as 3-dimensional objects is a central problem in Artificial Intelligence some aspects of which are studied in this paper.

The 3-dimensional objects considered are restricted to *solid, trihedral, opaque polyhedra.* For our purposes, a polyhedron is a closed and bounded subset of the three dimensional euclidean space $\mathbf{R}^3$, whose topological boundary is a union of finitely many subsets (the faces), each lying on a plane and bounded by finitely many segments of lines (the edges). The faces intersect at the edges. The endpoints of the edges are the vertices of the polyhedron. Notice that according to our definition, a polyhedron may contain a number of smaller polyhedra disconnected from each other. Solid means that no 'hanging' faces or edges are allowed. In other words, 'origami' constructions are excluded. Formally, that amounts to requiring that for every point on the surface (boundary) of the polyhedron there exists a topologically three dimensional set that contains this point and is contained in the polyhedron. A trihedral polyhedron is one all the vertices of which are intersections of exactly three faces lying on three different planes. This excludes the possibility of touching polyhedra or cracks.

Without loss of generality, we consider polyhedra situated in the first of the octants defined by the three cartesian axes of the three dimensional euclidean space. If for such a polyhedron, we project the parts of its edges visible from the xy plane onto this plane, we get a graph called the *image* of the polyhedron (Figure 1). Formally, a point (x, y, z) of the polyhedron is called *visible* (from the xy plane) if there is no other point (x, y, z') in the polyhedron such that $z' < z$. Only the visible parts of the edges are projected because of the opaqueness assumption. The projected

Report CS-R8715
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

polyhedron is refered to as the *scene*, or the 3D *realization* of the image. The projections of the vertices and the edges of the polyhedron are called the *nodes* (or *points*) and *lines*, respectively, of the image.

If a face of the polyhedron occludes part of an edge, then the projections of the boundary edge of the occluding face and the visible part of the partly occluded edge intersect in the image, forming a T-shaped figure. Although such a point of intersection is not the projection of a vertex, we count it among the nodes of the graph. Thus, the image is a plane graph, with nodes of degree 2 or 3.

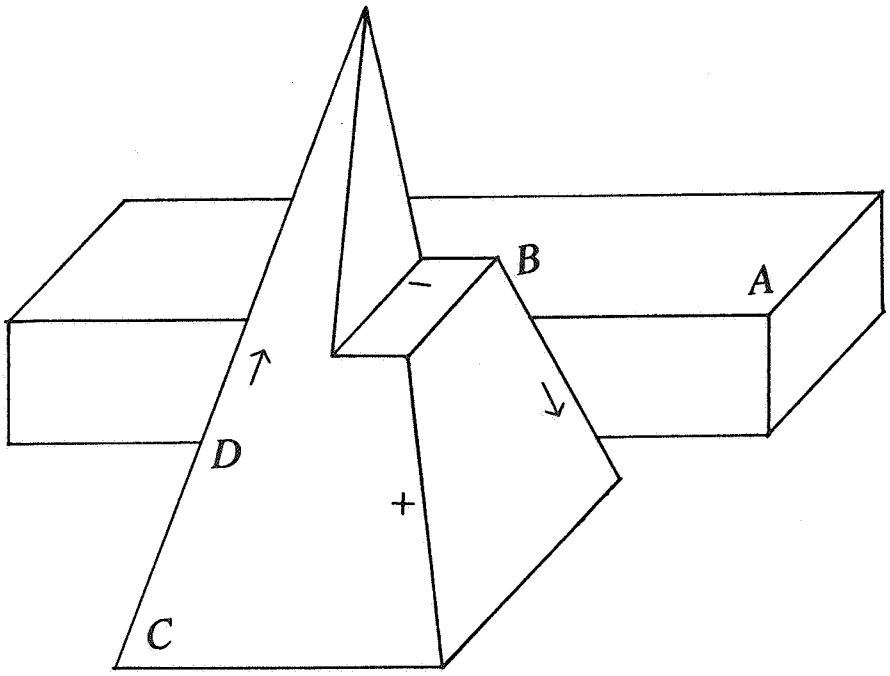

Figure 1. The image of a polyhedral object.

For reasons of clarity, we consider only orthographic projections, although the results can be almost verbatim extended to central (perspective) projections.

There is a natural classification of the nodes of an image according to their shape: *Y-nodes* are those where the three incoming lines form three angles of less than 180°, *E-nodes* are those where one angle is greater than 180°, *L-nodes* are those that have degree two, finally *T-nodes* are those where two of the three incoming lines are collinear (nodes A, B, C and D in Figure 1 are examples of each classification group, respectively). The T-nodes, as we have seen, are not projections of vertices.

There is a standard way [1, 3] to label the lines of an image depending on the way the corresponding edges of the scene are seen from the projection plane: if an edge is convex as seen from the projection plane, its projection is labeled by a '+', if it is concave, its projection is

labeled by '–', and if it is a *contour* edge, i.e. an edge whose one face occludes the other, its projection is labeled with an '→'. The direction of the arrow is such as to leave to the right the occluding face. So, there are three labels and four possible ways to label a line. Non-contour edges (lines) are called *connecting* edges (lines). In Figure 1, some labels are given that correspond to the natural realization of the image depicted.

Having a consistent labeling of an image obviously reveals important information about its 3D realization. Combinatorially, though, there is a very large number of possible labelings of the lines of an image.

In their pioneering works, Clowes [1] and Huffman [3] have shown that there is a very small number of ways to label the lines of a node so that it can be possibly realized as the projection of a vertex of a polyhedron.

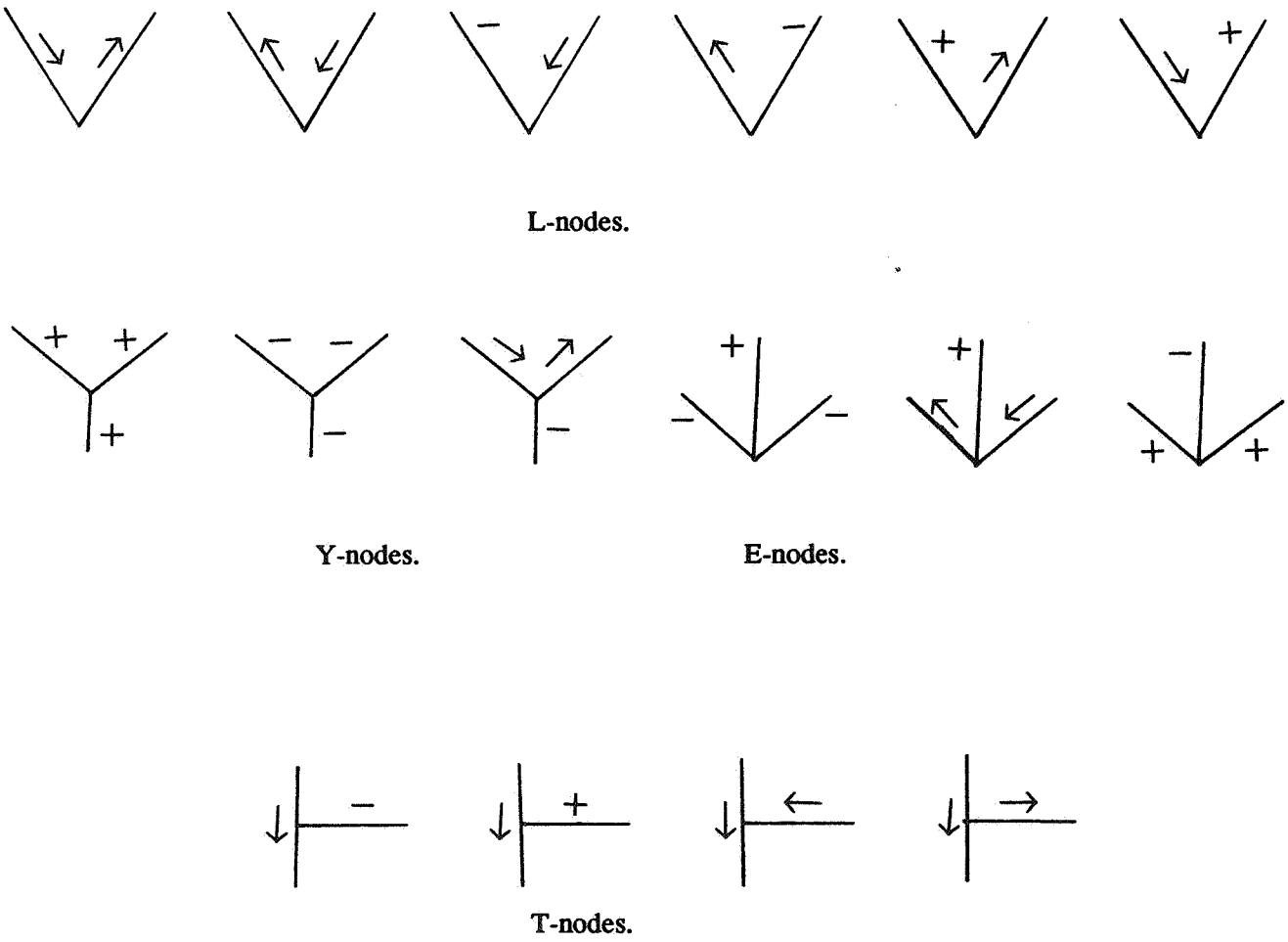

Figure 2. Legal labelings of a node (Clowes-Huffman labeling scheme).

These *legal* labelings for each type of node are given in Figure 2. Obviously, they impose severe

restrictions on the number of ways an image can be labeled. Winston [9] gives an excellent account for the work in the area. In this paper, we show how to further restrict the possible labels of an image, again under the sole assumption that the image is realizable.

Despite these restrictions, we are still faced with the combinatorial problem of consistently labeling the lines of an image so that no line receives different labels from its two endpoints. Images with such a labeling are called *labelable*. Non-labelable images are just impossible objects. The converse is known not to be true. For example, Figure 3 depicts a labelable image which is not realizable. The non-realizability is due to the fact that the two different faces A and B intersect in two non-collinear edges, namely a and b. Nevertheless, Sugihara [7] has given a polynomial algorithm that checks the realizability of a legally labeled image.

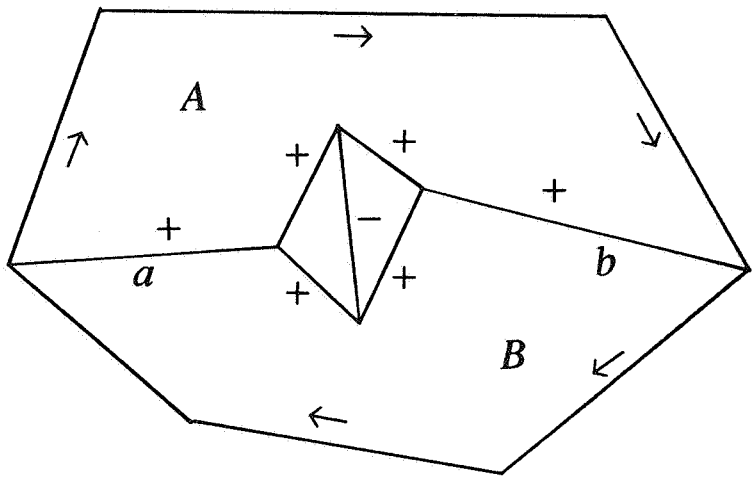

Figure 3. A labelable but non-realizable image.

Waltz [8] developed a filtering algorithm for labeling an image. Waltz' algorithm empirically had good running time in most applications [9]. Nevertheless, Kirousis and Papadimitriou [4] proved that the labelability (and realizability) problems are NP-complete. On the positive side, they gave a polynomial algorithm for the labelability problem of images that are taken to be projections of *orthohedral scenes*, i.e. scenes where all faces are perpendicular to one of the three cartesian axes. Orthohedral scenes, however, are a very limited special case of the trihedral world.

On the other hand, Freuder [2] investigates the question of how many lines of the image must be a priory known to be contour or connecting in order to insure that the image has a unique labeling.

2. Statement of result.

In this paper, we give a linear (in time) algorithm that answers the labelability question of a general image, under the assumption that a certain minimal set M of connecting lines is given. A set of lines M of an image is called a *minimal set of connecting lines*, if there is no legal labeling of the image whose set of connecting lines is properly contained to M. Notice that, in general, the

number of legal labelings compatible with a minimal set of connecting lines can be anywhere from zero to an exponential expression of the number of the nodes of the image.

This result possibly explains the success of Waltz's algorithm, despite the fact that its complexity is exponential in general. This is so because the knowledge of the set M can often be directly derived from the image.

To be more specific, the set M that we have to be given must contain at least one line for each Y-node. In other words, for each Y-node we have to know a priory at least one line that must be labeled with a '+' or a '–'. It is clear from the set of legal labelings of Y-nodes that every Y-node contains at least one connecting line. Therefore, such a set M is a minimal set of connecting lines.

Notice that in many cases such an a priori specification of a line as a connecting one can obtained from the type of one of its nodes. For example, if a line of a Y-node is also the middle-line of an E-node, then this line must be a connecting one and so there is no reason to require any information at all about the lines of this Y-node. The image of Figure 1 is such that no information is needed about the connecting lines to answer the labelability question. It seems that for most 'natural' images, the number of Y-nodes for which we cannot specify a connecting line (using only information supplied by the image) is very small, if not zero. So, our algorithm effectively labels most images without any outside information at all.

3. A linear algorithm for labeling an image.

We start by making explicit an assumption about the scene which practically states that small movements of the camera 'looking at' the scene do not affect the information content of the picture (image). This assumption is made in every work in the area. Our definition and lemma 1 below formalize what is informally assumed in other works, e.g., [3] or [9].

The scene is assumed to be in *general position* with respect to the projection plane. Informally, that means that if we perturb the polyhedron (or equivalently, the projection plane) a little bit, no new lines or nodes will appear on the image. In other words, the projective geometry of the image will not change. Formally, that means that no plane or line geometrically defined by edges or vertices of the scene is perpendicular to the projection plane.

As a consequence of the general position assuption we have the following:

Lemma 1. If two lines in the image are parallel, then they are projections of parallel edges. Also, if two lines are collinear then they are projections of collinear edges.

Proof. Indeed, suppose that two edges were skew to each other and had parallel projections. These skew edges uniquely define a line in space that is perpendicular to both. Consider the plane which is perpendicular to this uniquely defined line and contains the first of the two skew edges. This plane, which is geometrically defined in a unique way from the elements of the image, is perpendicular to the projection plane, a contradiction. For the second part of the lemma, consider two collinear lines. By the first part of the lemma, those are projections of parallel edges. If these edges are not collinear, then the plane that they define would be perpendicular to the projection plane, a contradiction again.□

In the complexity considerations of the algorithms in the sequel, unless we state differently or unless what we mean is clear from the context, we take as input size the number of the nodes of the image (denoted by n). If e is the number of edges of the image, then, because all nodes are

of degree 2 or 3, $e = \Theta(n)$.

The procedures of our algorithm involve the location of various types of subgraphs of the image (these types will be formally defined in the sequel). We assume, without explicitly mentioning it each time, that the location of such a subgraph is done in linear time by one of the standard graph searching techniques. Moreover, we suppose, without loss of generality, that the image is connected. Indeed, otherwise we only have to label its connected components separately.

Finally, we suppose that for each Y-node of the image we know at least one line which is connecting. These lines are marked by c and comprise a minimal set of connecting lines.

To check the labelability of the image, we first check the labelability of certain types of subgraphs of it, which we now define.

An *L-chain* is a subgraph of the image that comprises all the lines incident on a maximal succession of adjacent L-nodes (Figure 4). We suppose, without loss of generality that L-chains are open, like the one in Figure 4, i.e., they do not form a cycle of lines. This is so, because a cycle of L-nodes constitutes a connected component by its own, which obviously is legally labelable (and also realizable as an object with only one visible face).

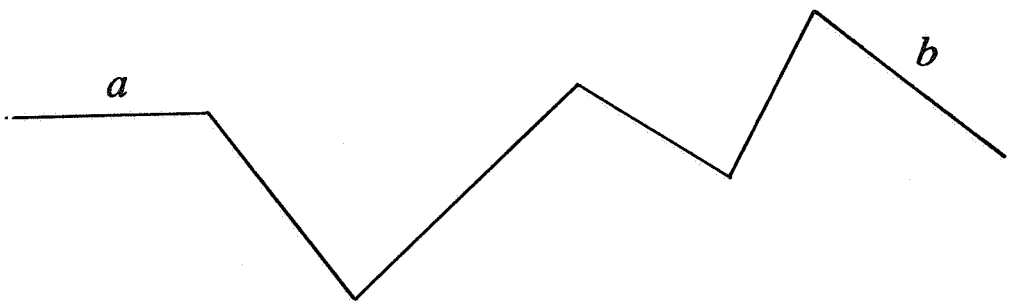

Figure 4. An L-chain. The lines a and b are called *end-lines*.

A *YE-chain* is a subgraph of the image that comprises all the lines incident on a maximal succession of adjacent Y- or E-nodes (Figure 5).

In the following lemmata, we prove certain properties that a labeling of a graph must satisfy. These properties restrict the possible labelings of an image even further than the Clowes-Huffman scheme (Figure 2). The properties are proved under the sole assumption that the image must have a 3D realization with the given labeling. We then give a linear algorithm that checks the labelability of an image when a connecting line is given for each Y-node. Of course, when developing the algorithm we do not assume that the image is realizable, as we do when proving properties of the labeling. This distinction, although is not always explicitly stated, is clear from the context. It should be pointed out that the NP-completeness of the general labelability problem remains valid even under the extra restrictions on the legal labelings that are described below. This is so, because in [4] the NP-completeness is proved by reducing the

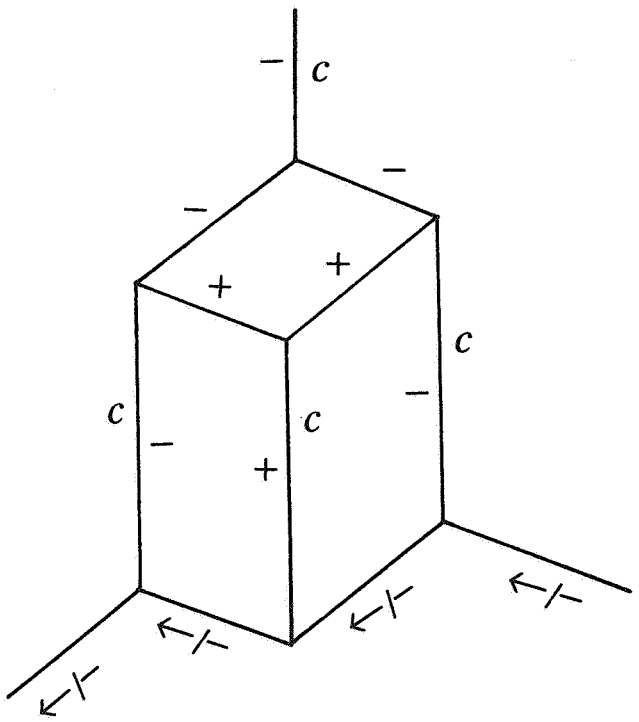

Figure 5. A YE-chain.

boolean satisfiability question to the labelability question of an image which if labelable is also realizable.

Lemma 2. If a line of an L-chain has a connecting label ('+' or '−'), then the labels of all other lines are uniquely determined.

Proof. Call a the line whose connecting label is given. If no two lines of the L-chain are collinear, then the labels of all other lines are arrows. This is so because there is at most one visible face on each side of the L-chain, and two faces cannot intersect on non-collinear edges. Moreover, it easily follows from the legal labelings of L-nodes in Figure 2 that these arrows have uniquely defined direction. Therefore, in this case, we have a uniquely defined label for all lines. If, on the other hand, some lines are collinear, then by the general position assumption, they correspond to collinear edges. Also, again because there is at most one pair of visible faces, for any two collinear edges, if one has a connecting label ('+' or '−') the other must have the same label. Therefore, the lines that are collinear with a must have the same label as a. Moreover, the ones that are not collinear with a must be labeled with an arrow. It can be easily seen now that each of the lines collinear with a determines a unique direction for these arrows. If these directions are incompatible, then the L-chain is not labelable. If they are compatible, we have a

uniquely defined label for the lines of the L-chain.□

Notice that the above lemma cannot be proved assuming only the restrictions on the labeling imposed by the Clowes-Huffman scheme.

Lemma 3. If a line of an L-chain is labeled with an arrow, then the possible labelings of any other line on that L-chain are at most three. These are the two arrows of opposite directions and one connecting label. In other words, an arrow label on a line of an L-chain excludes one connecting label from each of the other lines (the excluded connecting label need not be the same for all lines).

Proof. Itis easy to see that if a '+' is changed to a '−' (or vice versa) then all arrow labelings (which are uniquely determined by the previous lemma) must change direction. Therefore, an arrow labeling cannot be compatible with both a '+' and a '−' on some other line.□

Let us turn now to the YE-chains. Every Y-node of such a chain has at least one line marked by c, meaning that it must be labeled by either a '+' or a '−'. Also, the middle-lines of E-nodes are connecting lines. Moreover, any line in a YE-chain has at most three labelings. This is so because a) from the side-lines of E-nodes, arrows forming a 'left turn' are excluded and b) from the two lines of a Y-node that are not marked by c, arrows forming a 'right turn' are excluded. So, for a line of a YE-chain an arrow-label has only one possible direction. But there is a lot more that can be said about the possible labelings of a YE-chain. First we give a definition.

We generalize the notion of a label by introducing the notion of a *type*. Types are to be assigned to lines of YE-chains only (for those, an arrow-label has only one possible direction). There are four distinct types. The first three, the types '+', '−' and '→' are identical to the corresponding labels. It is only for reasons of convenience that we also call them types. The fourth type we introduce is denoted by '−/→'. Its interpretation is that a line with this type must have label '−' or '→', but it is not known which of the two is the case. For example, if we know that the middle-line of an E-node has label '+', then the legal labelings of Figure 2 do not provide sufficient information to determine the label of the side-lines. However, the type of them is uniquely determined to be '−/→'. In other words, this last type may be viewed as a set of possible labels. We call the type '−/→' the *ambiguous* type. Its introduction is due to C.H. Papadimitriou [5]. We say that the ambiguous type is *more general* from the types '−' and '→'.

Obviously, if the ambiguous type is assigned to a line marked by c or to the middle-line of an E-node, then a label (non-ambiguous type) for this line is uniquely determined (namely, '−'). In general, the legal labelings of Figure 2 imply the following observations about the types that can be assigned to the lines incident to a node of a YE-chain.

a) If a type is assigned to a line marked by c or to the middle-line of an E-node, then a label for this line is uniquely determined.

b) If a type is assigned to a side-line of an E-node, then a label for its middle-line and a type for the second side-line are uniquely determined.

b) If a type is assigned to the middle-line of an E-node, then types for all its lines are uniquely determined (the type for the middle-line is non-ambiguous).

c) If a type is assigned to a line of a Y-node, then types for all its lines are uniquely determined (the type for the line marked by c is non-ambiguous).

Of course, the above are true under the assumption that the original assignment of a type is legal (e.g., is not the assignment of '→' to a line marked by c). Observe that the c marks are

essential to obtain the above. Otherwise, assigning e.g., the ambiguous type to a line of a Y-node, would leave three possibilities for the other lines: the '–' and two arrows of opposite direction.

If we are given the type of one line of a YE-chain, we can uniquely assign types to all of its lines, following the above rules for the propagation of types. Specifically, we first visit the node for which the type of one of its lines is given. We assign types to all its lines following the rules above. We visit the other nodes in a breadth-first fashion. In other words, after assigning types to all lines of a node A, we visit sequentially all nodes B such that the line AB was assigned a new type during this last visit to A (a new type is either the first type assigned to a line or a type less general than the previous one). When visiting the node B, a type is determined for all its lines utilizing the above rules and the new type of AB that made us visit B. If a line of B had already another type, we assign to it the less general one (if, of course, the two types are compatible). Notice that we may be forced to return to A, immediately after the visit to B is completed. The procedure continues as long as we can assign a new type to a line. If we ever hit a contradiction, i.e., if we are forced to assign incompatible types to the same line, then we stop and we conclude that there is no assignment compatible with the type originally given. The time complexity of this procedure is linear, since we can assign a new type to a line at most twice. The procedure, so far, is similar to the filtering algorithm of Waltz [8], only with much fewer possible cases. In Figure 5, an assignment of types is given under the assumption that the line at the top marked with c was given to have label '–'.

A line which at the end of the above procedure has the ambiguous type is called an ambiguous line. Also, we call *maximal ambiguous set* a maximal succession of adjacent (i.e., sharing a node) ambiguous lines. In Figure 5, the four lines at the bottom form a maximal ambiguous set. It can be seen now that if following the above procedure, we succeed to uniquely assign types, then we can also assign a legal labeling to the chain. Indeed, for a maximal ambiguous set, we can either label all of its lines with a '–' or with a '→'. Arbitrarily specifying in this way the label of the lines of a maximal ambiguous set does not pass any information outside the set, therefore it can be safely done. This is so because any line adjacent to an element of a maximal ambiguous set (and not itself an element of the set) must be a connecting line whose label is uniquely determined just from the type of the elements of the set. A similar fact does not hold true for the possible types the Waltz' filtering algorithm gives. In that case, to check for labelability, we have to try all exponentially many combinations we might be possibly left with at the end of the algorithm.

The above considerations show how to label YE-chains. The problem, however, is to label the whole image. Difficulties arise if we try to check whether joining together labelable YE-chains through L-chains gives a labelable graph. To be more specific, when joining YE-chains and L-chains the following situations may arise.

First, suppose that we have a subgraph of the image that consists of a YE-chain and an L-chain so that the YE-chain shares an end-line a with the L-chain and moreover, the second end-line b of the L-chain is the middle-line of a T-node of the image (Figure 6). We call such L-chains *hanging* L-chains. It is trivial to check that the labelability of the YE-chain in this case implies the labelability of the subgraph consisting of the YE-chain and the L-chain. In other words, we can safely ignore hanging L-chains when resolving the labelability question of a graph. Even more obvious is the fact that we can also ignore L-chains whose both end-lines are middle-lines of two T-nodes of the image. Indeed, those are always labelable (e.g., by arrows going 'along' the L-chain) in a way that does not interfere with the other labels of the graph.

For a case a little more interesting from the above, consider a subgraph consisting of two

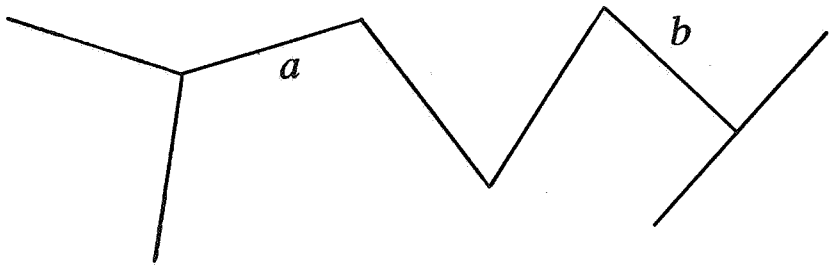

Figure 6. A hanging L-chain (with end-lines a and b).

YE-chains connected through an L-chain and suppose moreover, that the two end-lines a and b of the L-chain, which belong to the YE-chains, are collinear (Figure 7).

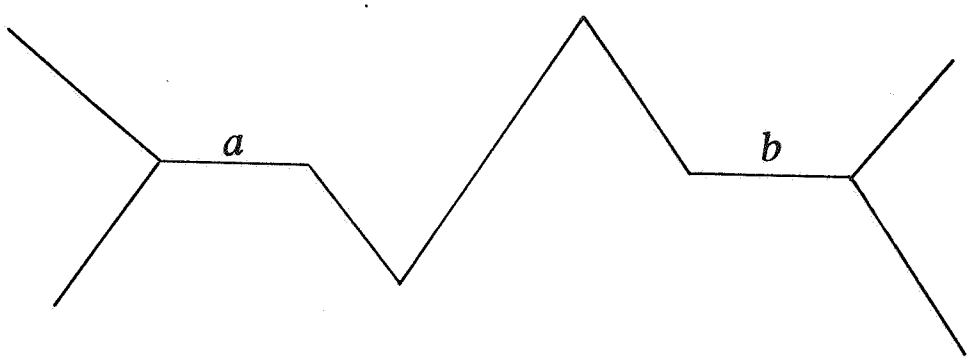

Figure 7.

In this case, by the general position assumption, the edges a and b must have the same label. Therefore, once we have an assignment of types for one of the two YE-chains, we obtain in a unique way an assignment for the other. Observe also that, by the proof of Lemma 1, if types for the two end-lines of an L-chain are given, then we can, in linear time, (the input size is the length of the L-chain) determine all pairs of labels of the two end-lines that are compatible with the given types and are extendible to a labeling of the whole chain. Of course, it may very well be the case that there is no such pair for a certain L-chain and the given types of its end-lines.

These considerations lead to the following: We call a subgraph of the image a *component* if it is a maximal subgraph comprising a) YE-chains joined together with L-chains in each one of which the two end-lines are collinear and b) hanging L-chains attached to those YE-chains. Given

now a component and the type of one of its lines we can determine, in linear time, whether this component is labelable compatibly with the given type. To do that, we execute the *type assigning* algorithm described below:

1) Assign types to all lines of the YE-chains of the component by the procedure described above. If that cannot be consistently carried out then the component is not labelable (compatibly with the originally given type). Otherwise,

2) for each L-chain of the component with collinear end-lines find, in linear time, all pairs of identical labels of its two end-lines (respectively) that a) can be extended to a legal labeling of the whole L-chain and b) are compatible with the types of the end-lines that have been determined in the previous step. Call these pairs (and the labels they contain) *permissible*. If this step is not succesful, i.e., if there is at least one L-chain with no permissible pair, then the component is not labelable (compatibly with the originally given type). Otherwise,

3) check whether the types found in the first step are compatible with the permissible pairs. Before we show how to do that, let us give a definition: An *M-class* is a maximal sequence of maximal ambiguous sets where each term of the sequence is joined to the next through an L-chain that has collinear end-lines (strictly speaking, the M-classes are the collections of the terms of these sequences, rather than the sequences themselves). It is easy to see that all lines in the union of the sets of an M-class should have the same type. Therefore, we do the following: i) Locate the M-classes. ii) For each M-class M assign to the lines in the union of its elements the (same) most general type which is compatible with the permissible labels of the end-lines of all L-chains that join together elements of M. If for at least one M-class there is no such type compatible with all permissible pairs, then the component is not labelable (compatibly with the originally given type or label).□

It can be seen now that the above algorithm, if it is succesful, then it uniquely assigns types to the lines of the component. Call an M-class whose lines have been assigned the type '$-/\rightarrow$' an *ambiguous* M-class. Observe that both possible labelings of the elements of the sets of an ambiguous M-class are compatible with the permissible pairs. Therefore, we obtain a legal labeling of the component if we specify either as '$-$' or as '$\rightarrow$' the label of all elements in the union of the sets of an ambiguous M-class. Moreover, since each step of the above algorithm imposes on the possible labels restrictions that necessarily follow from the restrictions of the previous steps, the labelings that are thus obtained are the only possible ones. Therefore, we have proved the following:

Lemma 4. Let C be a component of the image and suppose that the type of a line of C is given. Then, in linear time, we can find pairwise disjoint classes $M_1, \ldots, M_k$ (the ambiguous M-classes) such that each M_i consists of maximal ambiguous sets and the following are true:

a) The given type implies that the type of all lines in the union of the sets that appear in any ambiguous M-class is '$-/\rightarrow$'.

b) The labels of all lines that do not belong to a set of an ambiguous M-class are uniquely determined by the given type.

c) All legal labelings of the component that are compatible with the given type can be obtained as follows: (i) for each class M_i, assign to all lines that belong to any of its sets the same label ('$-$' or '$\rightarrow$'), and b) assign labels to the L-chains observing the restrictions imposed by the labels of their end-lines.

There are many ways, other than forming components, that the YE-chains and the L-chains

can join to form larger parts of the image. With regards to the labelability question though, the only remaining interesting case to be examined is that of components joined together through L-chains with non-collinear end-lines. All other cases either reduce to labeling components separately or they can be settled by the methods to be developed for this case. For notational convenience, an L-chain that joins two components and has end-lines that are not collinear and belong to YE-chains of the respective components is called a *component joining* L-chain.

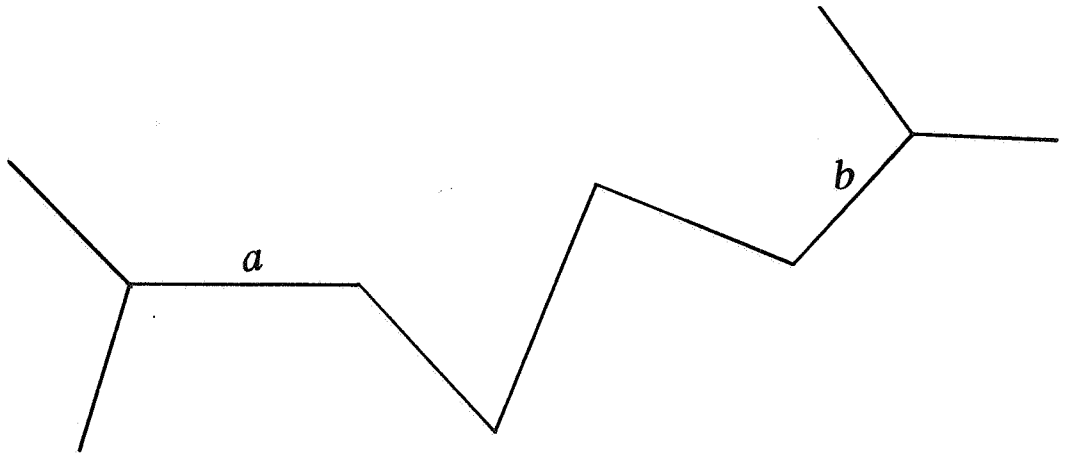

Figure 8. A component joining L-chain.
The end-lines a and b are not collinear and belong to YE-chains of the respective components.

Consider now a component joining L-chain. We know that each of its end-lines, being an element of a YE-chain, can be labeled in at most three ways: with a connecting label or with an arrow of a determined direction. Also, by the arguments in the proof of Lemma 2, we know that these end-lines cannot both get a connecting label. Moreover, by the proof of Lemma 3, we know that assigning an arrow label to one of these end-lines excludes the possibility of one connecting label from the other. Therefore, the permissible pairs of labelings of these end-lines, i.e., those that can possibly be extended to a labeling of the whole L-chain, are at most three: a pair consisting of an arrow on each end-line, and two pairs consisting of an arrow on one and a connecting label on the other. Also, the above considerations reduce the number of permissible labelings of each end-line to at most two.

For each end-line a that has exactly two permissible labelings, we introduce a boolean variable v_a which is intended to take the value 'true', if the label of a is a prespecified one out of the two possibilities, and is intended to take the value 'false', otherwise. Of course, it is immaterial which of the two labelings corresponds to which of the two boolean values of v_a. The important thing is that we have a different boolean value for each of the two permissible labelings. For example, if the two permissible labels of an end-line are '+' and '–', we may associate the value 'true' with '–' and the value 'false' with '+'.

Observe now that a label on an end-line a of an L-chain may uniquely determine the label of another end-line b . Such restrictions on the labelings of end-lines can be expressed by boolean

formulas of the form (x implies y), where x and y are literals of the variables v_a and v_b, respectively. Observe that all these formulas have two literals, therefore we can check them for satisfiability in linear time by assigning a truth value to a variable and following the implications as far as they would go. These considerations lead to the following algorithm that checks the labelability of the image (given at least one line marked c for each Y-node).

1) Determine the L-chains and the YE-chains of the image and the at most three possible labelings of each line in a YE-chain.

2) Determine the hanging L-chains and the L-chains with collinear end-lines and then determine the components of the image. Determine the component joining L-chains, and find the (at most two) permissible labelings of their end-lines. If this step is not succesful, i.e., if there is an end-line with no permissible labelings, then the image is not labelable. Determine the boolean formulas that express the restrictions on how these labelings can be paired.

3) Introduce a variable *EL* and set it equal to the collection of end-lines of component joining L-chains with exactly one permissible labeling. Let *EL* have the structure of a stack (i.e., a LIFO structure). Also, introduce a boolean variable *abrt*, whose intended meaning is that the step (4) below has been aborted.

4) As long as *EL* is non-empty repeat the two-step procedure described in the following two paragraphs.

i) Set *abrt* = false and and take the top end-line, call it a, out of *EL*. Call C the component that a belongs to. Observe that, since a has only one permissible labeling, the label of one line of C is given. If no assignment of types has been carried out for C, invoke the type assigning procedure for it and determine its ambiguous M-classes $M_1, \ldots, M_k$. Otherwise, if such an assignment has been previously carried out, then further restrict, as necessary, the collection of the ambiguous M-classes of C. For a component already provided with types, such a 'lifting of ambiguity' of some of its M-classes is the only new information that the label of a can provide. If, when assigning types to a component a contradiction arises, i.e., incompatible types are assigned to the same line, then the current labeling cannot be carried out, so a) abort the current step (4) of the algorithm, c) undo all assignments of types as well as all changes on the set *EL* and on the collection of boolean formulas executed during this step and c) set *abrt* = true.

ii) The specification of types done in (i) further restricts the permissible labelings of the end-lines of the component joining L-chains. So, consider those end-lines whose label is by now uniquely determined. Those determine the truth value of their corresponding variables v. Propagate these truth values as far as possible, by following the implications described by the boolean formulas. In this way, the permissible labelings of end-lines are further restricted. Again, if any contradiction arises, abort the current step (4), undo all changes and set *abrt* = true. Otherwise, put in *EL* (in any order) the end-lines whose set of permissible labelings has been reduced to a singletone. From the collection of boolean formulas, retain only those whose both variables correspond to end-lines with exactly two permissible values.

5) If *abrt* = false and if there is an end-line, say a, of a component joining L-chain whose label is not as yet uniquely determined do the following: Arbitrarily specify a truth value to the corresponding variable v_a. Thus, the label of a is uniquely determined. Put a on top of *EL* and go to step (4). If this step is aborted, that means that the chosen truth value leads to a contradiction. Therefore, in this case, assign to the v_a the only remaining second truth value. Thus, again, the number of permissible labelings of a is reduced to one. Put a on top of *EL* and go to (4).□

If the above algorithm ends with *abrt* = true, then there is no legal labeling of the image. If,

on the other hand, the algorithm ends with *abrt* = false then we have succeded to assign types to the lines of the image. From this assignment, we obtain a legal labeling of the image by arbitrarily specifying one label ('–' or '→') to the elements of each M-class that has type '–/→'. To be more specific, we prove the following:

Theorem. If for each Y-node of the image we are given at least one line that must be labeled with a '+' or '–', then, in time linear on the number of nodes (or edges), we can check whether the image is legally labelable.

Proof. Execute the algorithm described above. Observe that according to this algorithm, if we ever assign contradictory types to a line, we try the alternative truth-value assignment only for the last call of step (5). If this also leads to a contradiction, then we conclude that the image is not labelable. In other words no bactracking is ever made to a call of step (5) other than the last one. The only thing that needs to be shown to obtain the correctness of the algorithm is that this is enough.

We first observe that the boolean formulas involved in the algorithm are all of the form (x implies y). We conclude that if any contradiction arises during the propagation of the truth values of the variables v, this contradiction is only due to the assignment of a truth value performed at the last call of step (5). This is proved by the following well-known argument: Suppose that at a stage earlier than the last visit to step (5), the literal y was assigned the truth value 'false'. Suppose, moreover, that the first contradiction during the propagation of truth values is obtained by an implication of the form (x implies y), where x is known to be true. Then, by the contrapositive of the boolean implication, we should have obtained that x is false at the step that y was determined to be false. But then, we would have had an earlier contradiction. So, we proved that no backtracking to calls of step (5) preceding the last one is necessary for contradictions of this type.

Another crucial observation now is that the propagation of types takes place following rules that all have the syntactical form: 'if the type of line a is t, then the type of line b is t'' (again, clauses with two literals which, however, may assume at most four values: the number of possible types). Notice now that the end-lines of component joining L-chains have at most *two* possible labelings. Therefore, the restrictions that the labeling of one end-line imposes on the possible labelings of another can all be expressed by rules that have the syntactical form of two-literal clauses with two-valued variables. Therefore, again, if any contradiction is obtained in assigning labels to end-lines, we should only try to change the truth value assigned at the last call of step (5), and no further backtracking is necessary.

Finally, consider a component C and let a be the end-line in this component that is visited first during the execution of the algorithm. Assigning a label to a leads to the assignment of types to the lines of C, in linear time, by the type assigning algorithm. This assignment is propagated further by step (4). If this is done without any contradictions, then neither any later visits to end-lines of C can lead to contradictory assignments for the lines of C. This is so, because as explained above, no assignment of a label to any end-line can contradict the label assigned to another end-line at an earlier stage. Of course, it is possible that at a later visit to the component C, we could determine the label of an end-line that had been previously assigned—by the type assigning algorithm— the type '–/→'. But, as explained, this only entails the 'lifting of ambiguity' of an ambiguous M-class, and cannot lead to any contradictions inside C. Therefore, a contradictory assignment of types to the lines of C does not make necessary any backtracking

to a call of step (5), other than the last one. But, we have now covered all possible cases of contradiction that could make necessary a backtracking. Since a variable v assumes at most two values we conclude the correctness and the linearity in time of the algorithm. The algorithm though leaves some of the M-classes without a label, but only with an assignment of an ambiguous type. Fortunately, this is no problem: we can arbitrarily assign a '$-$' or '$\rightarrow$' label to all lines appearing in any element of an M-class that is left with an ambiguous type, with the restriction that lines from the same class get the same label. Indeed, this is possible in the context of the whole image, since M-classes that are left with an ambiguous type are not joined via L-chains. Therefore assigning a label to all lines in one of them does not supply any information about the label or type of any line outside the class. The fact that M-classes left with an ambiguous type are not joined follows immediately from the observation that non-collinear end-lines of L-chains are not assigned a type, but rather a label.□

4. Discussion.

It should be mentioned that the constant involved in the complexity formula of the above algorithm is very small. Indeed, the various types of subgraphs of the image can be located in one search of the graph. The determination of the permissible pairs of the labels of the end-lines of the L-chains and the determination of the corresponding boolean formulas all require only one visit to each of the lines of the L-chain involved. Finally, during steps (4) and (5) of the algorithm each line of the graph is visited at most 6 times. In the worst case a line will be assigned at the first visit the type '$-/\rightarrow$', then at the second, the label '$-$' or '$\rightarrow$' and at the third visit a contradiction may arise. The same pattern may be repeated after backtracking to the last call of step (5).

Moreover, the algorithm has an effective parallel version, i.e., it can be carried out with polynomially many processors in time polynomial in $\log n$. Indeed, it is routine to check that all tasks involved in our algorithm, like locating various subgraphs, propagating types, and propagating truth values following the implications of clauses with two literals are all variants of the problem of finding the strongly connected components (in the usual graph-theoretic sense) of a directed graph. We do not give the details, since they are messy and standard. As a sketch, consider the task of propagating types. To carry it out in parallel, we construct a directed graph as follows: For each line of the image, we introduce four nodes of the directed graph under construction. Each of these nodes corresponds to one of the possible four types that the line may have. Now, for each type-propagation rule of the form: 'If the line a has type t, then the line b has type t'', we connect (with a directed edge) the node that corresponds to the line a with label t to the node that corresponds to the line b with type t'. It can be seen that finding the strongly connected components of this directed graph is equivalent to the task of propagating types in the image. It is well known now how to find strongly connected components of a graph in an effective parallel fashion by carrying out matrix multiplication. For a review of all the related results see [6]. As it follows from these results, our algorithm can be carried out with n^3 many processors in $O(\log^2 n)$ time.

Acknowledgement.

The author would like to thank Christos Papadimitriou for numerous inspiring conversations that helped in proving the results of this paper.

References.

[1] M.B. Clowes, "On seeing things," *Artifical Intelligence, 2* (1971), 79-116.

[3] E.F. Freuder, "On the knowledge required to label a picture graph," *Artificial Intelligence, 15* (1980), 1-17.

[3] D.A. Huffman, "Impossible objects as nonsense sentences," in: B. Meltzer and D. Michie (eds.), *Machine Intelligence 6* (Edinburgh University Press, Edinburgh, Scotland, 1971), 295-323.

[4] L.M. Kirousis and C.H. Papadimitriou, "The complexity of recognizing polyhedral scenes," *Proc. 26th Annual IEEE Symposium on Foundations of Computer Science,* 175-185, 1985. To appear also in: *Journal of Computer and System Sciences.*

[5] C.H. Papadimitriou, Personal communication, 1985.

[6] M.J. Quinn and N. Deo, "Parallel graph algorithms," *Computer Surveys, 16*, 3, (1984), 319-348.

[7] K. Sugihara, "Mathematical structures of line drawings of polyhedrons—toward Man-Machine communication by means of line drawings," *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-4,* 5, (1982), 458-469.

[8] D. Waltz, "Understanding line drawings of scenes with shadows," in: P.H. Winston (ed.), *The Psychology of Computer Vision,* (McGraw-Hill, New York, 1975), 19-91.

[9] P.H. Winston, *Artificial Intelligence,* 2nd ed., (Addison-Wesley, Reading, Mass., 1984).