Stable-marriages algorithm for preprocessing phase maps with discontinuity sources

J. A. Quiroga, A. González-Cano, and E. Bernabeu

A new algorithm is proposed for solving the problems associated with discontinuity sources in phase maps. It is based on the stable-marriages algorithm and is implemented as a recursive procedure.

With this technique, discontinuity sources of opposite sign are connected by a set of cut lines that fulfills a stability criterion and possesses the minimum cut length of the stable sets. The algorithm is fast and easy to implement and has proved efficient, as experimental results show.

Key words: Interferometry, phase unwrapping.

1. Introduction

Over the past several years, many algorithms for phase unwrapping have been suggested.[1–4] These algorithms are designed to extract continuous phase maps that must be path independent, i.e., produce logically consistent phase maps.

In a phase map the number of 2π jumps between two given points A and B can be calculated as

$$n = \sum_i \left[\frac{\phi(i) - \phi(i-1)}{2\pi} \right], \qquad (1)$$

where brackets denote rounding to the nearest integer, $\phi(i)$ is the phase value at pixel i, and index i runs over the pixels of a path that joins A and B.

For a logically consistent phase map this number n must be independent of the chosen path. This implies that, for any closed path, n must be zero. However, real phase maps, owing to noise, sampling problems, or characteristics of the object, can become logically inconsistent; that is, phase unwrapping can become path dependent. These inconsistencies arise from the existence of discontinuity sources in the phase map.[4] In this way, a closed path containing or enclosing one or more discontinuity sources will generally have a nonzero value for n.

J. A. Quiroga and E. Bernabeu are with the Departmento de Óptica, Facultad de Ciencias Físicas, Universidad Complutense, Ciudad Universitaria, Madrid 28040, Spain. A. González-Cano is with the Escuela Universitaria de Óptica, Universidad Complutense, Arcos de Jalón, Madrid 28037, Spain.

Received 6 September 1994; revised manuscript received 1 March 1995.

0003-6935/95/235029-10$06.00/0.

Let us consider the phase map of Fig. 1. The two paths A and B that join points P_1 and P_2 contain a different number of 2π jumps. We find five jumps that follow B, so the phase difference between P_1 and P_2 is taken to be of the order of 10π rad. However, owing to noise or sampling problems, one of the jumps does not occur when we compute the phase difference along path A; thus we obtain a phase difference of 8π. This implies that the phase difference between P_1 and P_2 is path dependent, and the phase map of Fig. 1 is logically inconsistent.

To detect the discontinuity sources, we calculate the value of n by following the smallest closed path starting in every point, defined by the corners of a 2×2 square. When the value of n for this path is not zero, the starting point is a discontinuity source. It has been proved[5] that, for this type of path, n will always be either -1, 0, or 1. When $n = -1$, the discontinuity source is called a negative pole, and when $n = 1$, it is called a positive pole.

In general, the area to be processed is defined by use of a processing mask of arbitrary shape. This implies that, for the points immediately adjacent to the mask, the 2×2 square can include masked points. If this happens, the point for which we are computing n is declared a point of the mask.

Physically, these discontinuities correspond to a break of the fringes in the phase map and appear naturally as pairs of poles of opposite sign (called dipoles). Sometimes, however, an isolated discontinuity (monopole) may appear near the boundaries of the phase map or when a fringe ends abruptly because of sampling problems or the characteristics of the object. The abrupt ending of a fringe may produce not only a single isolated monopole but also a group of poles, the sum of whose residues is 1 or -1.

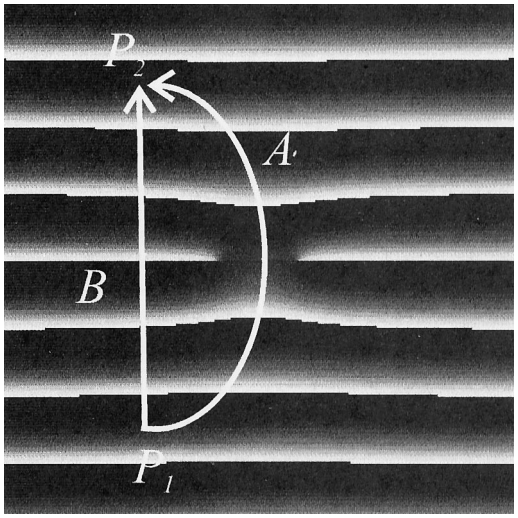


Fig. 1. Illustration of the problems caused by discontinuity sources in phase maps. The phase difference between P_1 and P_2 is different when calculated by path A than by path B.

To solve this problem, some researchers have proposed algorithms capable of automatically dealing with this type of error.[6,7] Others have proposed a preprocessing of the phase map.[4,5,8,9] The algorithm that we present here is of the second type; that is, it detects the discontinuity sources and places cut lines that act as barriers for a further phase unwrapping.

2. Description of the Algorithm

A. Foundations

Once the discontinuity sources of a phase map have been detected, we must prevent the phase-unwrapping algorithm from passing through a fringe break, delimited by two poles of opposite sign. We can do this by connecting poles of opposite sign by cut lines. Also, monopoles near the boundary must be connected by a cut line to a point on the boundary. The simplest way of placing the cut lines is to take each pole in turn and to join it to either the nearest unpaired pole of opposite sign or to the boundary, whichever is closest. This is called the nearest-neighbor method. But, if we proceed in this way, cut lines can be placed in a wrong way, as Huntley *et al.* show.[9] To obtain an optimum cut-lines set, a global selection criterion must be defined. The usual criterion is that the total length of the cut lines must be minimum for a given set of discontinuity sources. Following this criterion, Huntley *et al.* have proposed an algorithm based on simulated annealing.[9] However, the computational cost of this method is high.

We propose an algorithm that achieves full optimization of the placing of cut lines in an easier way and with much lower computational cost. It is based on the so-called stable-marriages algorithm,[10] an example of the programming techniques called general resolution of problems.[11]

These techniques define algorithms to solve specific problems not by using prefixed calculation rules but by following a systematic trial-and-error method. The general procedure is to divide the calculation process in partial tasks. These tasks are frequently expressed in a recursive form and are constituted by the execution of a finite number of subtasks. The general procedure can be considered a method of search that gradually constructs and checks a basic task tree. This task tree can be reduced by the use of heuristic techniques to decrease the computational cost of the algorithms.

In the case of the stable-marriages algorithm, two sets P and N with equal numbers of elements M are given. The goal is to find M pairs (p, n), with $p \in P$ and $n \in N$, that satisfy certain restrictions and preferences. Usually there will exist several sets of pairs that satisfy the conditions. From this set of possible solutions of the problem one must be chosen by use of an *a priori* selection criterion.

In our case the disjoint sets P and N are the positive and negative poles, respectively. The preferences are established as functions of the distance between poles, and the *a priori* criterion to be satisfied is that the total length of cut lines must be minimum with respect to all other stable solutions.

In general, in a given phase map, the number of positive and negative poles may not be the same because, as we have said, there can exist isolated discontinuity sources (monopoles) that may appear by two causes:

(1) Monopoles exist near the boundary of the phase map, whose counterparts of opposite sign would be placed beyond the boundary.
(2) Monopoles exist that are caused by an abrupt ending of a fringe that may be far from the boundary.

First, we describe the algorithm for an ideal set of discontinuities with no boundary and no isolated monopoles. In this case it is ensured that the number of positive and negative poles is the same, and any positive pole has a corresponding negative pole. Later, we show how we can deal with monopoles of the two mentioned types and with the problems associated with the boundary.

B. Algorithm for an Ideal Case

1. *Construction of Preference Matrices*

In the ideal case the number of positive and negative poles is the same, and we denote it by M. To join the positive and negative poles in the optimum way (i.e., to establish perfect marriages between them), we must know their preferences. To do this, we construct a distance matrix, given by

$$D[p, n] = \mathrm{dist}(p, n), \tag{2}$$

where $p \in P$ and $n \in N$ and dist means the Euclidean distance between the positions of those poles. As we can see, D is a square $M \times M$ matrix.

Now, for each row of matrix D (i.e., the distances from a given positive pole p_0 to all the negative poles

n) we consider the values of the elements $D[p_0, n]$ and we establish the order between them. This order determines a list of preferences for each positive pole p_0. Obviously, the preferred negative pole for p_0 to join is the nearest one. Using this, we construct a so-called preference matrix $NP[p, r]$. This is done in the following way: if the distance between p_0 and a certain negative pole n_0 is the rth in magnitude among all the distances $D[p_0, n]$, we assign to the element $NP[p_0, r]$ the value n_0. This means that $NP[p, r]$ corresponds to the negative pole that occupies the position r in the list of preferences of p; that is, the pole is the rth closest negative pole to p. If two points are at the same distance of a given one, the ordering algorithm used (in our case a bubble algorithm) decides which one to place first in its ordering process. No rule is imposed for those cases. The ordering in these cases has no effect in the performance of the stable-marriages algorithm.

In the same way we construct another preference matrix for negative poles, $PN[n, r]$, by using the columns of $D[p, n]$ instead of the rows. $PN[n, r]$ corresponds to the positive pole that occupies the position r in the list of preferences of n; that is, the pole is the rth-closest positive pole to n. This is shown in the following numerical example; let us consider a matrix D, given by

$$D = \begin{bmatrix} 10 & 4 & 8 \\ 4 & 16 & 7 \\ 3 & 1 & 14 \end{bmatrix}; \tag{3}$$

the preference matrices are

$$NP = \begin{bmatrix} 2 & 3 & 1 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \end{bmatrix}, \tag{4}$$

$$PN = \begin{bmatrix} 3 & 2 & 1 \\ 3 & 1 & 2 \\ 2 & 1 & 3 \end{bmatrix}. \tag{5}$$

For instance, element $NP[1, 1]$ is 2 because the negative pole number 2 is the nearest one to positive pole 1 because $D[1, 2]$ is the smallest of the first row of matrix D.

2. *Stable-Marriages Algorithm*

Once we have defined the preference matrices, the algorithm begins. The algorithm is implemented recursively[11] by a procedure [called Try(p)] that is calling itself continuously, thus establishing different sets of marriages between poles (called solutions, denoted by S) in a trial-and-error process that seeks an optimum solution S_{opt}. The marriages are formed attending to two factors:

(1) The preferences of the poles, according to the preference matrices defined above.

(2) The feasibility of the marriages, according to some imposed conditions.

The algorithm tries to marry in a successive way all the positive poles. The selected partner for any positive pole is in principle the negative pole that occupies the first place in its list of preferences, that is, $NP[p, 1]$. However, this marriage may not be possible because the desired negative pole may have been assigned to a previous positive pole. In this case the algorithm tries the second, third, etc., preferences, until it finds an unmarried one. The marriages established in this way must be stable. The stability is an important concept that is treated in an extensive way in Subsection 2.B.3.

Proceeding in this way, we obtain a first solution, S_1. We characterize this solution by the total length of its cut lines L_1, and we consider S_1 as the current optimum solution. Next, we systematically explore the whole space of stable solutions S. To do this, we break the marriage of the last positive pole p_M, and we try to marry it with its next feasible (stable and unmarried) preference. Obviously, for this pole there does not exist any other feasible partner, so we must go one step backward and break the marriage of the preceding pole p_{M-1} and look for its next feasible partner.

We systematically proceed in this way until we obtain a second stable solution S_2. For this solution we evaluate its total cut length L_2. If $L_2 < L_1$, we take S_2 as the current optimum solution. If $L_2 > L_1$, S_1 remains as the current optimum solution. We seek any other stable solutions in the same way. For each stable solution S we compare its cut length L with the cut length of the current optimum solution L_{opt}.

In principle, there exists $M!$ possible solutions. This means that we must direct our search for the optimum solution in order to avoid an enormous waste of computing time. This can be done by use of the mentioned stability condition (hence its importance). This means that we must not consider every possible partner for a given pole, but only those that generate stable marriages, and we thus cut branches in the solution tree. On the other hand, as we are only interested in the optimum solution, we can evaluate the cut length of a solution in every partial step of its formation; i.e., when a new couple is declared stable, we evaluate the partial cut length L_p. If $L_p > L_{\text{opt}}$, we stop searching for new couples for this solution and we explore a new branch of the solution tree. We can represent the algorithm by this pseudocode:

```
procedure TRY(p)
begin
  for r = 1 to M do
  begin n = NP(p, r)
    if n unmarried AND (p, n) stable then
    if L_p < L_opt then
      begin annotate (p, n); declare n married;
```

```
        if p < M then TRY(p + 1)
        else if L < L_opt then annotate optimum
        solution;
        declare n unmarried
      end
    end
end
[main program]
begin initialize of starting sets and matrices;
      TRY(1);
      draw cut lines corresponding to optimum solu-
      tion;
end
```

3. *Stability*

Imposing a stability condition is very important to avoid considering clearly wrong couples. The stability is a consequence of the comparisons between preferences. In the standard stable-marriages algorithm[11] there are two symmetric causes that make a couple (p, n) unstable:

(1) There exists a married negative pole n_p preferred to n by p that also prefers p to its partner.
(2) There exists a married positive pole p_p preferred to p by n that also prefers n to its partner.

This definition of stability is not advisable for our case because it does not ensure that the stable marriages produce a smaller cut length than the unstable, as Fig. 2 shows. Figure 2(a) shows a stable solution according to this criterion. Figure 2(b) is an unstable solution (for instance, the marriage B2 is unstable because B prefers 3, and 3 also prefers B, but 1 is married to A), but it has a smaller cut length than Fig. 2(a).

Also, with this stability criterion the algorithm does not cut the branches of the solution tree fast enough. This means that for $M \approx 50$ the computing time is too long (~15 min. for a 486 66-MHz microcomputer).

For these reasons we modified the stability criterion for our particular case and adopted a new one that tries to reproduce in a small scale what we try to achieve in a full scale: minimizing the total cut length. The criterion is, given a couple (p, n), as follows:

(a) We consider the first unmarried negative pole different than n in the list of preferences of p, which we denote by n^*.
(b) We consider the first unmarried positive pole different than p in the list of preferences of n, which we denote by p^*.
(c) We consider the following distances:

$$d_1 = D[p, n],$$
$$d_2 = D[p^*, n^*],$$
$$d_3 = D[p, n^*],$$
$$d_4 = D[p^*, n]. \quad (6)$$

(d) We compare $d_1 + d_2$ with $d_3 + d_4$. If $d_1 + d_2 \leq d_3 + d_4$, the couple is declared a stable marriage. If $d_1 + d_2 > d_3 + d_4$, the couple is unstable.

This is illustrated in Fig. 3.

This stability criterion cuts the branches of the solution tree very fast and makes possible a processing time of a few seconds for $M \approx 150$. With this criterion the solution of Fig. 2(b) becomes stable,

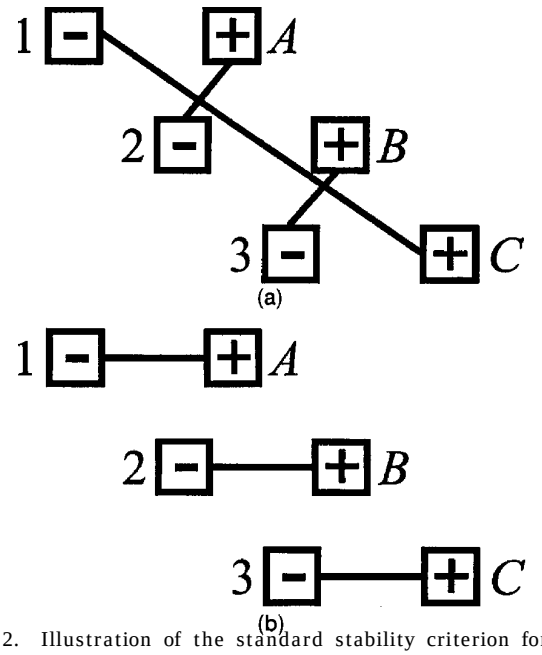


Fig. 2. Illustration of the standard stability criterion for the stable-marriages algorithm. According to this criterion, solution (a) is the stable one. However, the total cut length of solution (b) is evidently smaller, and therefore solution (b) is better for our purposes.

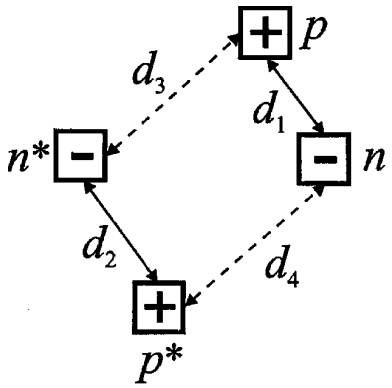


Fig. 3. Construction for the stability criterion adopted by us. n^* is the first unmarried negative pole in the list of preferences of p different than n. Similarly, p^* is the first unmarried positive pole in the list of preferences of n different than p. The marriage between p and n is stable if $d_1 + d_2 \leq d_3 + d_4$.

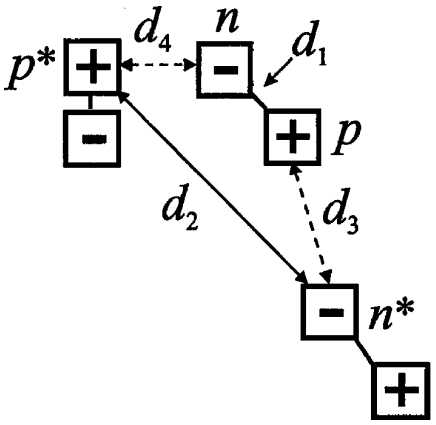


Fig. 4. Problems of the stability criterion adopted by us. If p is the first positive pole to seek a partner, no marriage for it is stable. The logical marriage between p and n is unstable because $d_1 + d_2 > d_3 + d_4$, so the desirable solution, represented by solid arrows, is unstable. This solution is, however, stable with our criterion if p is not the first processed pole.

although the solution of Fig. 2(a) is unstable, as we desired.

However, sometimes couples that clearly should be stable are declared unstable according to this criterion. This is shown in Fig. 4. The logical solution, to join p with n, is unstable because $d_1 + d_2 > d_3 + d_4$. This occurs only when the first positive pole to be considered is p, which may happen. In this case there does not exist any stable marriage for p. Situations of this type can block the algorithm. The practical solution adopted for these cases is to assign to the unmarriable pole its first unmarried preference, even though this marriage is unstable. In this way, the algorithm can continue. With this criterion the solution represented by the solid arrows of Fig. 3 is accepted, as is desirable. This way of proceeding is adopted only when no stable global solution has been found. Once a first stable solution is found, we strictly apply our stability criterion.

C. Algorithm for Real Cases: Monopoles

1. General Method

In a real phase map the area to be processed is limited by a boundary of arbitrary shape. The boundary may break a dipole if one of the poles lies within the processed area while the other one lies outside this area. In this case the phase map presents a monopole placed near the boundary. In principle, this type of pole should not be joined to another pole but to the point of the boundary that is closed to it. On the other hand, owing to sampling problems or to the characteristics of the object from which the interferogram is produced, there may exist isolated monopoles far from the boundary.

In both cases the existence of monopoles causes some problems to our algorithm because the initial quantities of positive and negative poles (i.e., the cardinals of the sets P and N) may be different. The standard stable-marriage algorithm works properly only when these quantities are equal. Even if the cardinalities of P and N are the same, no pole is an adequate partner for the conflictive poles mentioned. Because our algorithm joins only pairs of poles, we need to create partners for these monopoles in order to give them the chance to marry. These are the so-called image poles.

Because it is very difficult to predict which poles are really monopoles, we can generate a virtual image pole for any real pole. This image pole is of opposite sign to the real pole and is placed at the point on the boundary that is closest to it.

However, this way of proceeding is not very efficient because we assume the risk of creating too many image poles when only a few monopoles exist. The increase in the number of poles implies a corresponding increase in the computing time and the memory requirements when we establish the marriages between them. To avoid unnecessary image poles, we impose a condition for any potential image pole to be really generated. This condition is as follows:

(1) For each positive real pole p we consider its first preference, n.

(2) If the first preference of n (i.e., its natural partner) is not p, we generate an image pole for p.

(3) If the first preference of n is p, we consider the points on the boundary closest to each of them,

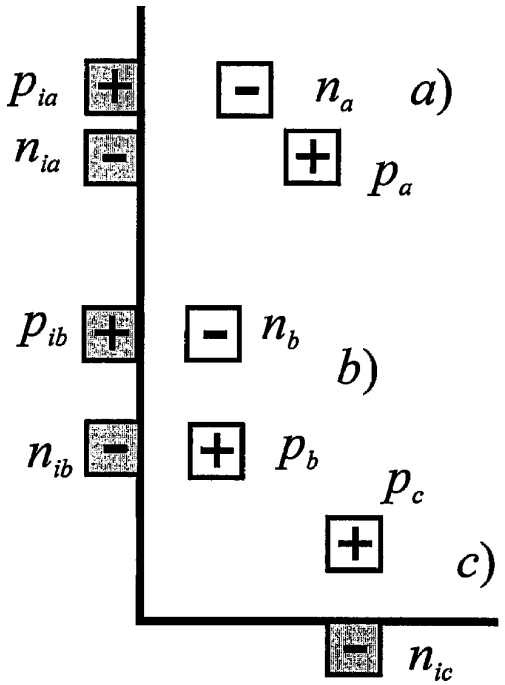


Fig. 5. Illustration of the criterion used for generation of image poles. According to the criterion, image poles p_{ib}, n_{ib}, and n_{ic} are created for n_b, p_b, and p_c, respectively, while no image poles are generated for poles p_a and n_a (that is to say, p_{ia} and n_{ia} are not created).

namely, p_i and n_i. We compare two sums of distances: $D[p, n] + D[p_i, n_i]$ and $D[p, n_i] + D[p, n]$.

(4) If $D[p, n] + D[p_i, n_i] < D[p_i, n] + D[p, n_i]$, no image pole is generated for p or n.

(5) If $D[p, n] + D[p_i, n_i] \geq D[p_i, n] + D[p, n_i]$, two image poles are generated, one for p and one for n; that is, p_i and n_i are accepted as image poles.

(6) When this process is completed for all the

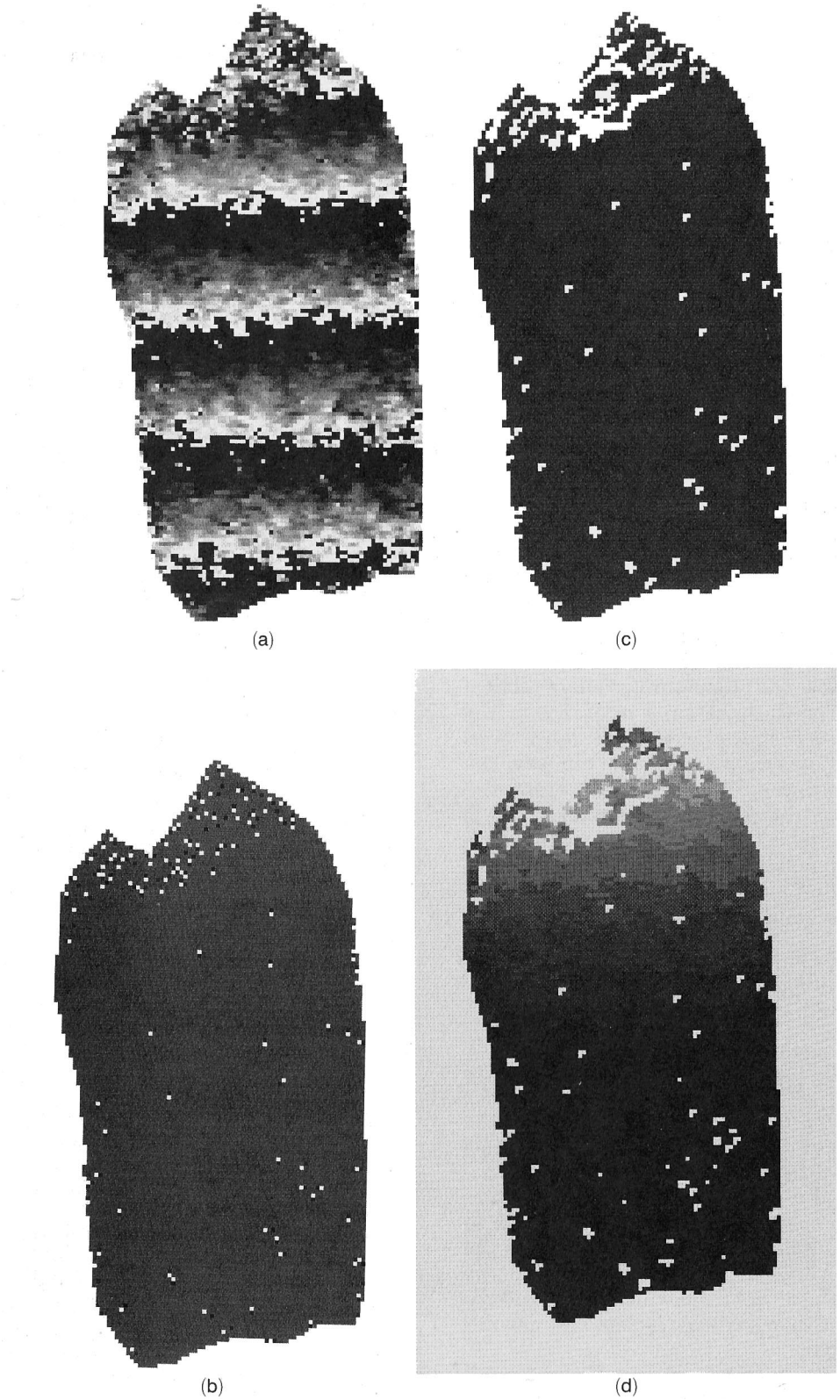


Fig. 6. (a) Real phase map. (b) Poles of the phase map. Positive poles are represented by black dots and negative poles by white dots. (c) Cut lines obtained by the algorithm. (d) Resulting unwrapped phase when the cut lines are used.

positive poles, we repeat it only for the negative poles for which no image pole has been previously generated.

This is illustrated in Fig. 5.

When all the necessary image poles have been generated in this way, we have two new sets of poles, P_I and N_I, that contain the positive and negative image poles, respectively. We only have to add these poles to the real ones and complete the starting sets for the algorithm; that is,

$$P = P_R \cup P_I,$$
$$N = N_R \cup N_I, \tag{7}$$

where P_R and N_R denote the sets of real positive and negative poles. Then, we proceed with the algorithm in the way that it is depicted in Subsection 2.B, using as starting sets P and N and rebuilding the matrices D, NP, and PN.

Only some minor changes must be made in the algorithm so that it can deal with image poles. These are the following:

(1) The real pole from which an image pole is generated is the sole real pole to which the image pole can be joined; that is, the marriage's real image can be declared stable only when the image pole has been generated by the real pole.

(2) The length of the lines joining marriages formed by two image poles (which are possible) is not computed for the evaluation of the total cut length, and these cut lines are not represented in the diagrams.

2. *Use of a Safety Margin*

The above-depicted way of proceeding has proved adequate, as is shown below. However, some problems may arise when limitations on computer memory exist because the generation of image poles produces an increase in the cardinality of sets P and N and the corresponding increase in the size of the matrices D, NP, and PN. When the memory requirements are very restrictive, we may be interested in another strategy.

This strategy is the use of a safety margin, given by a user-selected value, Δ. We generate image poles only for real poles within this safety margin, that is, poles whose distance to the boundary is smaller than Δ, while keeping the condition for generation of image poles stated in Subsection 2.C.1. We do this because we may expect that almost all the monopoles are near the boundary because they correspond to breakings of dipoles by the boundary. The isolated monopoles far from the boundary do not produce, however, any image pole when this strategy is applied. This fact and the fact that the selected safety margin may not include all the monopoles produced by the boundary can make the number of total (real plus image) positive and negative poles unequal, so it is not guaranteed that the stable-marriages algorithm works properly. For this reason it is advisable to use this technique only for phase maps from which we may expect that there do not exist too many monopoles far from the boundary or if we are obliged by strong restrictions on computer memory.

If the cardinals of P and N remain equal after applying this method of generation of image poles, we may apply the stable-marriages algorithm in the usual way. If they are different, we must proceed in the following way:

(1) We use as set P the set of poles with the smaller cardinality, independently of the sign of the poles; that is, we interchange the sets if there are more positive than negative poles.

(2) We establish the marriages by following the usual algorithm. Of course there will always remain some unmarried negative poles for any solution. These are not taken into account when the total cut length of the solutions is evaluated.

(3) Once the optimum stable solution has been found, we assign as partners to the remaining unmarried negative poles the points of the boundary closest to them.

The final results of applying this technique are acceptable in most cases, as shown below.

3. *Experimental Results*

We tested the algorithm with real phase maps using a 486 66-MHz microcomputer. Figure 6 shows a representative result. In Fig. 6(a) we can see a phase map corresponding to a speckle interferogram. Figure 6(b) shows the position of the discontinuity

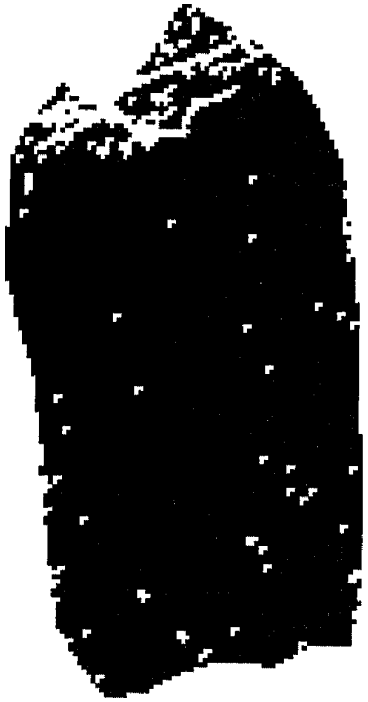


Fig. 7. Cut lines for the phase map of Fig. 6 when a safety margin of two pixels is used.

sources (poles). Positive poles are represented by black dots and negative poles by white dots. The number of poles was 120 real positive poles, 121 real negative poles, 59 image positive poles, and 58 image negative poles, with a total cardinality for P and N of $M = 179$. No safety margin was considered. Figure 6(c) shows the cut lines obtained by the algorithm. The processing time for obtaining the optimum solution was 13 s. Finally, Fig. 6(d) shows the unwrapped phase map by using the processing mask that appears in Fig. 6(c). Cut lines are shown. The value of the phase in the nonprocessed points corresponding to cut lines can be obtained by interpolation.

When a safety margin of two pixels is used to process the poles of Fig. 6(b), the resulting cut lines are the ones shown in Fig. 7. While the number of real poles remains the same, the number of images poles has changed (28 positive and 25 negative image poles). This implies that the cardinals of P and N are different (128 and 126, respectively). Nevertheless, the cut lines are correctly placed, as shown in Fig. 7. Owing to the reduction of the total number of poles, the processing time is shorter (9 s).

The importance of the concept of safety margin in some cases is shown in Fig. 8. Figure 8(a) shows a phase map in which delimited zones exist with problems caused by a wrong sampling. As can be seen in Fig. 8(b), poles are clearly distributed in two separate groups. In each of these groups, dipoles and monopoles exist because of fringe breaks, but within the groups the number of poles of opposite sign is the same. Here the best choice is to set $\Delta = 0$ (no image poles) and to let the algorithm marry the poles within the groups and thus avoid the joining of any pole to the boundary, even if such marriages could produce solutions of smaller total cut length. Figure 8(c) shows the cut lines for these conditions. These cut lines are correctly placed, as one can see, because they prevent an unwrapping process to cross the conflictive areas. Finally, Fig. 8(d) shows the unwrapped phase map obtained with the mask of Fig. 8(c). Again, cut lines are shown. The processing time was 7 s for $M = 68$.

Figure 9 shows the result of applying our algorithm to a phase map that presents some fringe breaks. In this case the set of cut lines provided by a simple

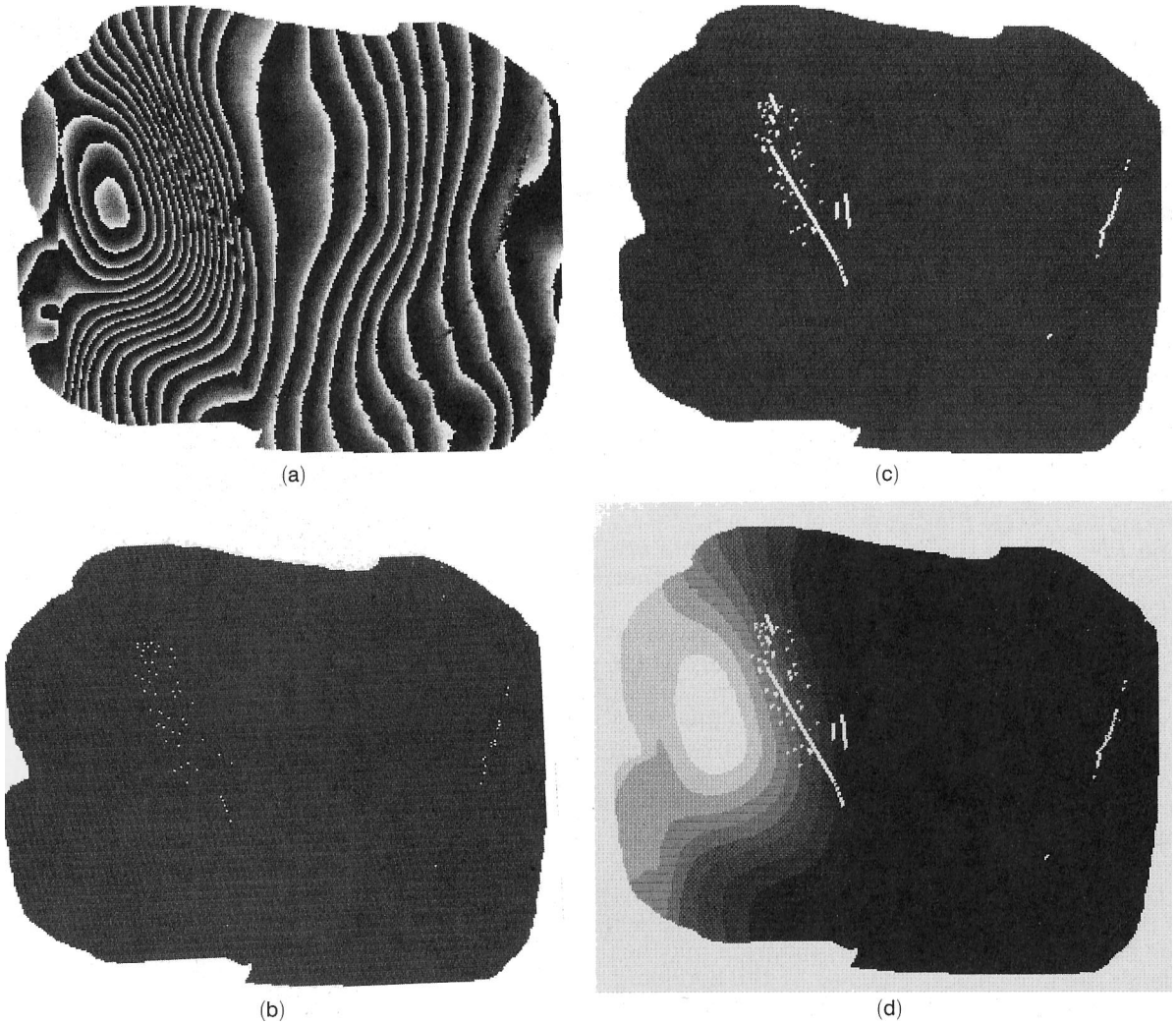
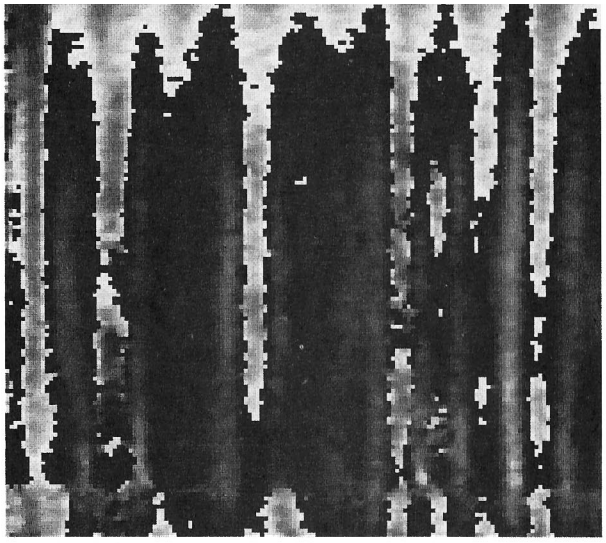
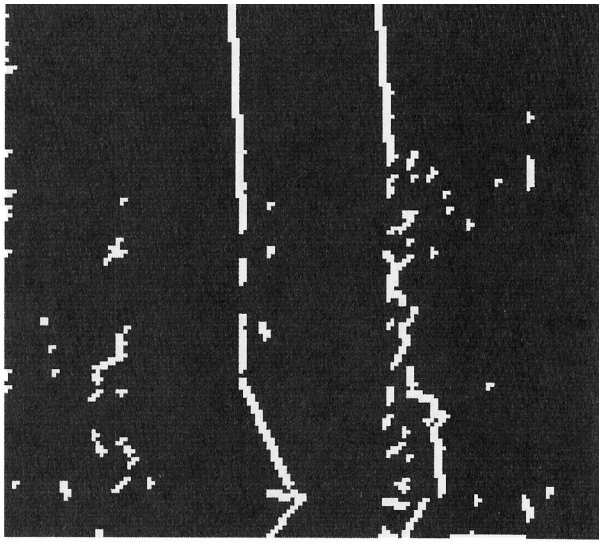


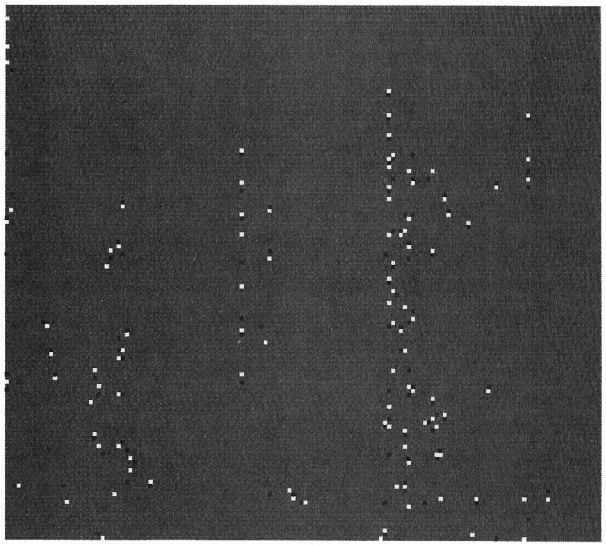
Fig. 8. (a) Real phase map. (b) Poles of the phase map. Positive poles are represented by black dots and negative poles by white dots. (c) Cut lines obtained by the algorithm when a safety margin of 0 is used. (d) Resulting unwrapped phase when the cut lines are used.



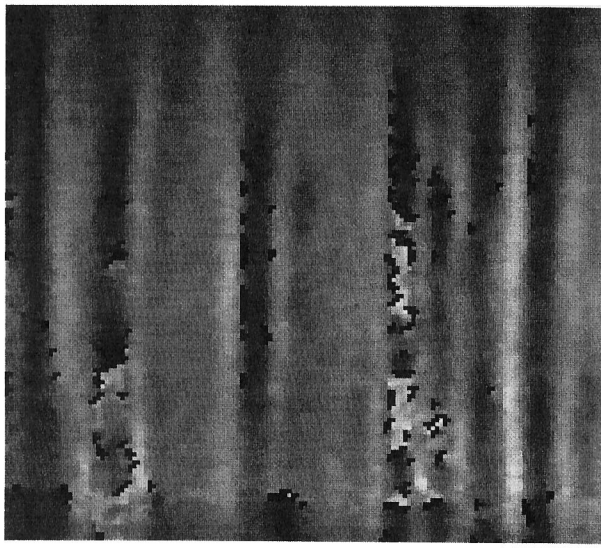
(a)



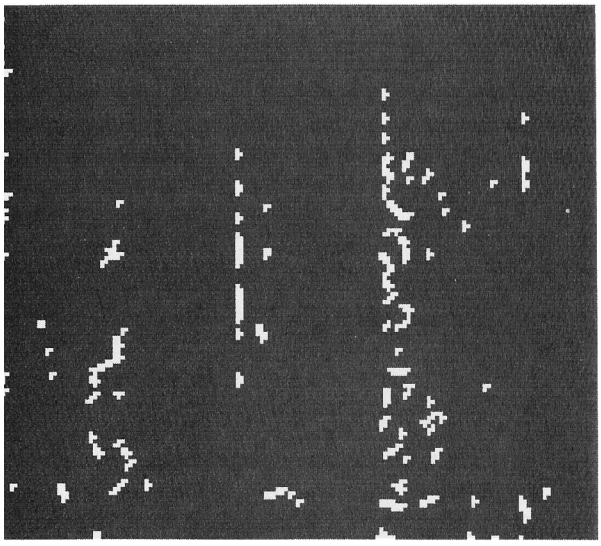
(d)



(b)



(e)



(c)

Fig. 9. (a) Real phase map. (b) Poles of the phase map. Positive poles are represented by black dots and negative poles by white dots. (c) Cut lines obtained by our algorithm. (d) Cut lines obtained by a nearest-neighbor method. (e) Resulting unwrapped phase when the cut lines obtained by our algorithm are used.

nearest-neighbor method is clearly less adequate than the set provided by our algorithm. Once the poles have been detected, image poles are created according to the rules described in Subsection 2.C.1. No security margin is considered (i.e., $\Delta = \infty$). In this way, for the phase map of Fig. 9(a), the number of positive poles is 103 real poles plus 50 image poles, and the number of negative poles is 104 real poles plus 49 image poles. That is, the cardinalities of P and N are equal to 153. These two sets (real plus image) of poles are the ones used by both algorithms (our own and a nearest-neighbor method). Figure 9(b) shows the positive of the discontinuity sources (poles). Positive poles are represented by black dots and negative poles by white dots. The result of applying our algorithm to the phase map is shown in Fig. 9(c), and the result obtained by a nearest-neighbor method is shown in Fig. 9(d). The time employed by the nearest-neighbor method to establish cut lines was 10 s, while our algorithm needed 13 s. One can see that a more convenient set of cut lines is obtained by our algorithm. Figure 9(e) shows the unwrapped phase map obtained with the mask of Fig. 9(c).

4. Conclusions

We have depicted an efficient and fast algorithm for the preprocessing of phase maps with discontinuities based on the stable-marriages algorithm. This general-purpose algorithm has been modified by us in order to adapt it to the usual problems of real phase maps. In particular, we have introduced the concepts of image poles and security margins. The main advantage of the algorithm is its short processing time, even when implemented on a conventional microcomputer. It is especially efficient, as are other branch-cut algorithms, for processing noisy phase maps. Problems such as geometrical features of the phase map (such as cracks, aliasing, etc.) usually cannot be solved by this type of algorithm.

We thank Hans Steinbichler for the images of Fig. 6, 8, and 9. This work was partially supported by Tecnologias Avanzadas de la Producción project TAP92-0087.

References

1. K. A. Stetson, "Phase-step interferometry of irregular shapes by using an edge-following algorithm," Appl. Opt. **31,** 5320–5325 (1992).
2. D. P. Towers, T. R. Judge, and P. J. Bryanston-Cross, "A quasi heterodyne holographic technique and automatic algorithms for phase unwrapping," in *Fringe Pattern Analysis*, G. T. Reid, ed., Proc. Soc. Photo-Opt. Instrum. Eng. **1163,** 95–119 (1989).
3. H. A. Vrooman and A. M. Mass, "Image processed algorithms for the analysis of phase-shifted speckle interference patterns," Appl. Opt. **30,** 1636–1641 (1991).
4. J. M. Huntley, "Noise-immune phase unwrapping algorithm," Appl. Opt. **28,** 3268–3270 (1989).
5. D. J. Bone, "Fourier fringe analysis: the two-dimensional phase unwrapping problem," Appl. Opt. **30,** 3627–3632 (1991).
6. P. Andrä, U. Mieth, and W. Osten, "Strategies for unwrapping noisy interferograms in phase-sampling interferometry," in *Industrial Applications of Holographic and Speckle Measuring Techniques*, W. P. Jueptner, ed., Proc. Soc. Photo-Opt. Instrum. Eng. **1508,** 50–60 (1991).
7. M. Servin, R. Rodríguez-Vera, and A. J. Moore, "A robust cellular processor for phase unwrapping," J. Mod. Opt. **41,** 119–127 (1994).
8. R. M. Goldstein, H. A. Zebker, and C. L. Werner, "Satellite radar interferometry: two-dimensional phase unwrapping," Radio Sci. **23,** 713–720 (1988).
9. J. M. Huntley, R. Cusack, and H. Saldner, "New phase unwrapping algorithms," *Fringe '93, Proceedings of the Second International Workshop on Automatic Processing of Fringe Patterns*, W. Jueptner and W. Osten, eds. (Akademie-Verlag, Berlin, 1993), pp. 148–153.
10. D. Gale and L. S. Shapley, "College admissions and the stability of marriage," Am. Math. Mon. **69**(1), 9–14 (1962).
11. N. Wirth, *Algorithms + Data Structure = Programs* (Prentice-Hall, Englewood Cliffs, N.J., 1976).