Network approaches to two-dimensional phase unwrapping: intractability and two new algorithms

Curtis W. Chen and Howard A. Zebker

Department of Electrical Engineering, Stanford University, Stanford, California 94305-9515

Received June 18, 1999; revised manuscript received October 12, 1999; accepted October 14, 1999

Two-dimensional (2-D) phase unwrapping, that is, deducing unambiguous phase values from a 2-D array of values known only modulo 2π, is a key step in interpreting data acquired with synthetic aperture radar interferometry. Noting the recent network formulation of the phase unwrapping problem, we apply here some well-established ideas of network theory to formalize the problem, analyze its complexity, and derive algorithms for its solution. It has been suggested that the objective of phase unwrapping should be to minimize the total number of places where unwrapped and wrapped phase gradients differ. Here we use network constructions to show that this so-called minimum L^0-norm problem is *NP*-hard, or one that complexity theory suggests is impossible for efficient algorithms to solve exactly. Therefore we must instead find approximate solutions; we present two new algorithms for doing so. The first uses the network ideas of shortest paths and spanning trees to improve on the Goldstein *et al.* residue-cut algorithm [Radio Sci. **23**, 713 (1988)]. Our improved algorithm is very fast, provides complete coverage, and allows user-defined weights. With our second algorithm, we extend the ideas of linear network flow problems to the nonlinear L^0 case. This algorithm yields excellent approximations to the minimum L^0 norm. Using interferometric data, we demonstrate that our algorithms are highly competitive with other existing algorithms in speed and accuracy, outperforming them in the cases presented here. © 2000 Optical Society of America [S0740-3232(00)01303-X]

OCIS codes: 280.6730, 120.3180, 350.5030.

1. INTRODUCTION

Two-dimensional (2-D) phase unwrapping is the process of attempting to recover unambiguous phase values from a 2-D array of phase values known only modulo 2π. Much recent work on phase unwrapping, including ours, has been motivated by the increasing use of synthetic aperture radar (SAR) interferometry. In this technique the phase of an interferogram formed from multiple complex, coherent radar images of the same scene can be used to make highly accurate measurements of surface topography,[1,2] deformation,[3–5] or velocity.[6,7] For many applications, however, these wrapped phase values must be unwrapped, and improper unwrapping is often the most significant source of measurement error. Consequently, a great deal of effort has been put into improving the overall reliability of phase unwrapping algorithms.

Following the leads of Costantini[8] and Flynn,[9] we use the powerful framework of network theory to formalize and interrelate several phase unwrapping concepts. After reviewing and refining the network model, we examine the motivations for one of the problem's frequently suggested goals,[10–13] minimizing the so-called L^0 norm. We show that this minimization problem is theoretically intractable, suggesting that our efforts are best spent developing algorithms to approximate rather than exactly optimize its objective. We next develop two such approximate algorithms based on network techniques, and we compare them with several existing algorithms by using interferometric synthetic aperture radar data. The results demonstrate that our algorithms are highly competitive in speed and accuracy and as such are significant on both practical as well as theoretical levels.

Network theory is studied in a wide variety of fields because of its generality and applicability to different types of problems. With its application to the phase unwrapping problem, we derive rewarding insights from the extensive work others have done on seemingly unrelated subjects. In tailoring those ideas to fit the specifics of the problem at hand, we briefly review network concepts as necessary. For detail, though, the reader is directed to texts such as the one by Ahuja *et al.*[14]

Notationally, we adopt the language of networks containing nodes and arcs rather than that of graphs containing vertices and edges. We use the former to describe both, neglecting their subtle differences for simplicity's sake. We use the term "approximate algorithm" to denote any algorithm that finds an approximate rather than an exact solution. Our colloquial use of the word "approximation" should not be confused with its use in some fields to suggest algorithm performance guarantees. We use the term "gradient," in keeping with the phase unwrapping literature, to describe scalar differences between discrete neighboring pixels. Strictly, a gradient is a continuous vector quantity, but our notation should be clear in context.

All commonly used phase unwrapping algorithms are based on the assumption that the true unwrapped phase field varies slowly enough that in most places, neighboring phase values are within one-half cycle (π rad) of one another. If this assumption were true everywhere, the

0740-3232/2000/030401-14$15.00

phase could be unwrapped by simply integrating wrapped phase differences, or gradients, along any path from pixel to pixel throughout the interferogram. While this assumption is indeed correct for most parts of an interferogram, even the most mundane of interferograms may contain many phase gradients that are greater than one-half cycle. Moreover, as we integrate from pixel to pixel, the inclusion of an erroneous phase gradient value causes all subsequent pixels to be in error. Thus, if handled improperly, phase gradients greater than one-half cycle may cause large-scale errors that affect whole regions of the interferogram.

The task of a phase unwrapping algorithm, then, reduces to locating gradients that are greater than one-half cycle: these gradients are called discontinuities. Most algorithms locate discontinuities by posing the phase unwrapping problem as a constrained optimization problem whose solution exactly or approximately minimizes the value of some objective function. Such a function maps particular unwrapped phase fields to scalar values according to some criterion, called the objective, for comparing the desirability of possible solutions. Least-squares algorithms implicitly locate discontinuities by minimizing the squared difference between unwrapped and wrapped phase gradients.[12,15–17] On the other hand, the residue-cut algorithm of Goldstein *et al.*[10] and the minimum cost flow (MCF) algorithms suggested by Costantini[8] and Flynn[9] use "path-following" approaches that expressly identify and accommodate discontinuities during direct integration of phase gradients. Such approaches are sometimes referred to as local rather than global because, as described below, they use only local information at each step to approach globally optimal solutions for their respective objective functions. We draw heavily from these algorithms, so we briefly outline them here.

In a properly unwrapped phase field, the integral of unwrapped gradients around a closed loop should always be zero. In a wrapped phase field, however, a true gradient greater than one-half cycle will be wrapped to a different value, causing the results of some closed path integrals to be nonzero integers. This property was used by Goldstein *et al.*[10] to locate discontinuities. Wrapped phase gradients are integrated in loops for each 2×2 square of pixels in the interferogram, resulting in 1, 0, or -1. Nonzero results are called residues or charges, and they indicate the presence of inconsistencies with respect to the assumption that wrapped phase gradients are less than one-half cycle. Moreover, integration paths encircling unbalanced numbers of positive and negative residues also contain such inconsistencies. To avoid such paths, "cuts" are grown in a treelike manner throughout the interferogram such that every charge is on a neutral tree, where a tree is simply a set of connected cuts. Phase integration paths are then disallowed from crossing over any cuts, so no paths encircle unbalanced charge. Consequently, along these paths, phase gradients may be integrated under the assumption that they are less than one-half cycle. Cuts therefore represent possible locations of discontinuities. Although an attempt is made to minimize the total length of cuts in the scene, the exact minimum of this quantity generally cannot be attained. Still, the residue-cut algorithm is fast and often quite accurate. In its original implementation, however, cuts are able to close on themselves, so a solution may suffer as large areas of the interferogram are closed off from other areas, causing the unwrapping to be incomplete.

MCF algorithms are similar to the residue-cut algorithm in that they connect positive and negative residues, but they do so guided by a different minimization objective. The first MCF approach can be attributed to Flynn,[9] although the framework suggested by Costantini[8] generalizes considerably the formulation of the phase unwrapping problem. Under this framework, the phase unwrapping problem itself is equated with a general network flow problem, allowing the interchangeable use of fast existing network optimization routines as well as many other ideas in the rich and well-developed area of network theory. Here we treat Flynn's algorithm as a specific implementation of the general MCF approach.

With both the MCF and the residue-cut approaches, the unwrapped phase, when rewrapped, is identical to the original wrapped phase; indeed, phase unwrapping may be viewed as the process of finding the integer numbers of cycles which, when added to the wrapped phase values, result in a residue-free unwrapped solution. Equivalently, one may find the numbers of extra cycles added to the wrapped phase gradients rather than to the phase values themselves; it is the total number of these extra gradient cycles that an MCF algorithm seeks to minimize, not the total cut length as in the residue-cut algorithm. The two objectives differ because cuts may have zero or multiple extra cycles of phase across them.

The general MCF minimization problem is solved with network flow techniques with use of a model like the one in Fig. 1. Each 2×2 residue loop integral is represented by a node whose surplus equals the value of the integral (1, 0, or -1, with negative surpluses being demands). Directed arcs, or possible flow paths, connect neighboring nodes and are consequently associated with phase gradients in the original data. Flow on a particular arc physically represents the difference in cycles between its associated unwrapped and wrapped phase gradients. The problem is constrained so that flow must be conserved at all nodes, meaning that the net flow out of a node (flow out minus flow in) must be equal to the node's

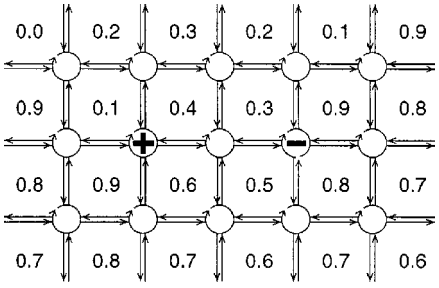

Fig. 1. Example network equivalent of the phase unwrapping problem. The numbers represent the 2-D array of phase samples (normalized to one cycle). Each 2×2 clockwise loop integral of wrapped phase gradients is a node in the network, and positive and negative residues result in supply and demand nodes. Neighboring nodes are connected by arcs, or possible flow paths. The amount of flow on an arc represents the difference (in cycles) between the unwrapped and the wrapped phase gradients associated with that arc. The net amount of flow out of a node must be equal to the node's surplus.

surplus. These constraints are equivalent to the property that an unwrapped phase field must be residue free: Summing the flow out of a node is equivalent to integrating differences between unwrapped and wrapped phase gradients around a loop. Given the original wrapped phase, then, any unwrapped phase field can be completely specified by a flow meeting the network constraints; such a flow is called feasible. Moreover, the objective functions of most algorithms can be embodied in network cost functions that map the amount of flow on each arc to a cost. Hence one can think of different algorithms sharing the common goal of finding the feasible flow with the smallest total cost, where differences in algorithm objectives are reflected by differences in arc cost functions. The network model thus provides an elegant, general way of relating unwrapped solutions to their original wrapped phase fields. This is helpful because flow in a network may be more easily conceptualized than differences between unwrapped and wrapped phase gradients.

In the MCF algorithm, arc cost functions are linear, and flow magnitudes are also restricted to be integers. This network flow optimization problem is called the MCF problem, and efficient algorithms exist for its solution.[14] Once the flow problem is solved, the unwrapped phase can be integrated with extra cycles inserted where dictated by flow.

It has been repeatedly suggested that the goal of phase unwrapping should be to minimize the total length of discontinuities in the unwrapped phase.[10–13] In Section 2 we discuss the motivations for this idea and describe how the use of such an objective makes the phase unwrapping problem *NP*-hard, or theoretically intractable. The proof, based on network constructions and complexity theory, is given in Appendix A. This result does not imply that we should abandon the minimum-discontinuity objective but suggests that we should focus our efforts on designing efficient heuristics and approximate algorithms for it. We develop two such algorithms using network techniques in Sections 3 and 4. The first algorithm is an improvement on the residue-cut algorithm, in which we use the network ideas of shortest paths and spanning trees. Our improved algorithm is very fast and has two main advantages over the original: Complete coverage is guaranteed, and the user is able to guide cut placement by defining weights on the interferogram. With our second algorithm we extend the ideas of linear network flow problems to the nonlinear minimum-discontinuity case. This algorithm uses network cycle-canceling ideas, employing the shortest-path routines from the first algorithm to provide excellent approximations to the minimum-discontinuity objective.

2. PHASE UNWRAPPING OBJECTIVES AND COMPLEXITY

Before describing any algorithms, we shall explore the objectives and theoretical complexity of the phase unwrapping problem itself, using the minimum L^p-norm framework suggested by Ghiglia and Romero.[12] For compactness of notation, we simply use here the term L^p in place of the phrase "minimum L^p norm." L is the difference between a pair of corresponding unwrapped and wrapped phase gradients, and p is a power to which L is raised in order to establish an error metric. In the network model, L is thus equivalent to the magnitude of flow on an arc whose cost is L^p. This leads to the minimization objective

$$\text{minimize}\left\{ \sum_{i,j} w_{i,j}^{(x)} |\Delta\phi_{i,j}^{(x)} - \Delta\psi_{i,j}^{(x)}|^p + \sum_{i,j} w_{i,j}^{(y)} |\Delta\phi_{i,j}^{(y)} - \Delta\psi_{i,j}^{(y)}|^p, \right. \quad (1)$$

where $\Delta\phi^{(x)}$ and $\Delta\psi^{(x)}$ are the x components of the unwrapped and wrapped phase gradients, respectively, and $\Delta\phi^{(y)}$ and $\Delta\psi^{(y)}$ are their y direction counterparts. Wrapped gradients are always assumed to be between $-\pi$ and π. User-defined weights w are assigned to particular gradient differences, and the summations include all appropriate rows i and columns j.

In the limit as p approaches zero (henceforth $p = 0$ or L^0), the objective is to minimize the number of discontinuities, or places where the unwrapped and the wrapped phase gradients differ by any amount. Because it approximately minimizes the total discontinuity length, the residue-cut algorithm can be classified as an L^0 algorithm. On the other hand, the MCF algorithm yields L^1 solutions, and least-squares algorithms yield L^2 solutions. As noted above, many authors suggest that the goal of phase unwrapping should be to minimize the total length of discontinuities in a scene, as in the L^0 metric. This idea has some intuitive appeal, and in the application of measuring surface topography, for example, it corresponds to minimizing the length of physical discontinuities in a surface.

Empirically, L^0 and L^1 algorithms have also tended to be more accurate than L^2 solutions.[13,18] If in fact minimizing the L^0 objective should be the true goal of phase unwrapping, the accuracy of L^1 algorithms might be explained by the observation that L^0 and L^1 optimal solutions are often similar and are identical when the corresponding gradients of the true unwrapped phase and the wrapped phase differ by exactly one cycle or not at all. This may be the case in areas of an interferogram corrupted by noise but with little underlying topography, where residues exist mainly in positive–negative pairs. However, when the topography contains physical discontinuities, as in the case of layover, corresponding unwrapped and wrapped phase gradients may differ by many cycles. In such cases the L^1 objective would assign high costs to the multiple-cycle discontinuities, instead favoring a larger set of discontinuities, each with a smaller number of cycles; such behavior would lead the MCF algorithm away from the correct unwrapped phase. A simplified example of how residues may be arranged for such a case is shown in Fig. 2. The L^0 optimal flow correctly reflects the multiple-cycle discontinuities introduced by layover, whereas the L^1 optimal flow results in a global error.

Still, L^1 solutions are appealing in that they can be calculated exactly and quite efficiently, and our ability to *find* good L^1 solutions may make up, to some degree, for deficiencies in the objective itself. On the other hand,

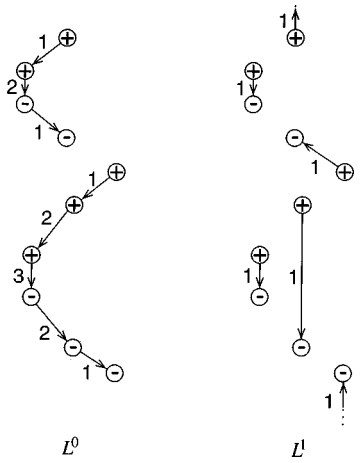

Fig. 2. Residue arrangement and flow in the presence of topographic layover with radar illumination from the left and hence range increasing toward the right. The residue arrangement reflects two peaks in layover. The arrows represent flow in the directions shown, with magnitudes indicated by the numbers. The L^0 optimal solution (left) correctly reflects the multiple-cycle discontinuity, whereas the L^1 optimal solution (right) does not.

finding an exact L^0 solution is very difficult. The residue-cut algorithm approximates L^0 behavior reasonably well in areas of good coherence but very poorly in other areas. The minimum L^p-norm algorithm of Ghiglia and Romero[12] is not guaranteed to find a global minimum when $p = 0$, only a local minimum. Buckland *et al.* claim that their minimum cost matching algorithm[11] finds "the global minimum of total cut length," but this is the case only where cuts are restricted to positive–negative residue pairs (matchings), whereas a global L^0 minimum generally contains extended discontinuities with many residues on them. No known algorithm is able to efficiently compute an exact, global L^0 minimum, and we assert now that in all likelihood, none will ever be able to do so in the general case. To justify this claim, we show in Appendix A that the L^0 problem is *NP*-hard, or that it belongs to a class of intractable problems that are likely impossible to solve in polynomial time.[19] A polynomial-time algorithm is one whose worst-case running time is bounded by some polynomial of the problem size. It is not difficult to see that an algorithm whose execution time increases exponentially or factorially with the number of variables quickly becomes impracticable.

NP-hard problems derive their difficulty from a more restrictive set of problems known as class *NP*-complete. *NP*-complete problems are posed as questions with yes–no answers, and among other interesting characteristics, any *NP*-complete problem can be transformed into any other of the hundreds of *NP*-complete problems in polynomial time, implying that a polynomial-time algorithm solving any one *NP*-complete problem could be used to solve all others. Such an algorithm has never been found to exist, however, and although it remains to be proven that no such algorithm can exist, this is the prevailing assumption in the field. *NP*-hard problems are those that, if solvable exactly, could also yield solutions to *NP*-complete problems through polynomial-time transformations.[19] Therefore, to show that the L^0 problem is *NP*-hard, we show that some *NP*-complete problem may be polynomially transformed into it. That is, we show that if we have a hypothetical black box solving the L^0 problem, we can use it to solve any instance of some *NP*-complete problem. We address the details and subtleties of this transformation in Appendix A, proving that the L^0 problem is *NP*-hard. It is at least as hard as any *NP*-complete problem and is among the most difficult known to complexity theory.

It would be foolish to give up on phase unwrapping because of this, though. Strictly, the phase unwrapping problem itself is also impossible because there is no way to unambiguously recover the information lost when the true phase is wrapped. The hardness of the L^0 problem simply tells us that instead of trying to solve the problem exactly, we ought to focus our efforts on reasonable and efficient heuristics and approximate algorithms. Such algorithms may give excellent solutions in practice. Moreover, there is no physical reason that the exact L^0 minimum must provide the correct unwrapping, nor is there any physical meaning in the threshold of one-half cycle (π rad) for deciding whether a gradient is a discontinuity. Consequently, good L^0 approximations may in general be just as accurate as exact solutions. Two algorithms for finding such approximations are given in the following two sections.

3. AN L^0 MINIMUM SPANNING TREE ALGORITHM: RESIDUE CUTS REVISITED

The speed and accuracy of the residue-cut algorithm make it attractive, despite its inability to provide complete solutions. We present here an improved version of this algorithm that makes use of several network ideas to yield complete coverage and allow user-defined weights.

In the original residue-cut algorithm,[10] a nonzero residue, or charge, is identified in the wrapped data. A cut is then drawn from this charge to the next nearest charge, regardless of sign. This forms the beginning of a tree. If the tree is not neutral, the next nearest charge to the tree is added and a cut drawn to it, and the process continues until the tree becomes neutral. At that point, some other unbalanced charge becomes the root of a new tree. When all charges are on neutral trees, the phase is integrated in a flood-fill fashion such that no cuts (tree branches) are integrated over. This approach is generally accurate in areas of good interferogram coherence, but it has problems in areas of poor coherence where densely placed cuts close on themselves, preventing integration into whole regions of the interferogram.

Since the cuts serve as possible locations of discontinuities, the total tree length is an upper bound on the total discontinuity length. Minimizing the tree length is thus a reasonable objective, but finding its exact minimum is precisely an *NP*-hard problem called the minimum rectilinear Steiner tree problem (see Appendix A). Note, however, that increasing the tree length does not necessarily increase the total discontinuity length, since many tree branches are not discontinuities. Suppose that we make the following modification to the residue-cut algorithm: When a tree becomes neutral, we start the next tree at

the next nearest charge to the current tree, and we actually draw a cut to the new tree. In other words, we build a single tree that contains all the charges. In general this tree will not be a minimum Steiner tree, but the process described does build what is called a minimum spanning tree, which reasonably approximates a minimum Steiner tree (see Fig. 3); we therefore call our algorithm the minimum-spanning-tree (MST) algorithm.

A minimum spanning tree is one that contains all nodes of some set S and minimizes the total tree length, given the important restriction that tree branches are allowed to split only at nodes in S, whereas in a Steiner tree, branches may split anywhere. With S as the set of all charges, the process described above is known as Prim's algorithm.[14] Stated more succinctly, one can build an exact minimum spanning tree by starting with an arbitrary node in S, then adding the shortest path from any node in S on the tree to any node in S not on the tree, and continuing. Note that our minimum spanning tree is not meant to span all nodes of the network, only those of nonzero phase residue. That is, the tree spans all nodes of an implied network that consists of the set of all charges and the shortest paths between these charges. On an unweighted rectilinear network, a minimum spanning tree can be no longer than 1.5 times the length of the minimum Steiner tree.[20] Minimum spanning trees have been used previously in phase unwrapping in a different way to connect reliable areas of a magnetic-resonance-imaging phase field.[21]

The reader may suspect that the inclusion of extraneous tree branches may tend to make even more cuts close on themselves, blocking off even more regions of the unwrapping than in the original algorithm. By definition, however, a true tree should never close on itself. Indeed, an examination of the source code of the original residue-cut implementation shows that cuts are allowed to close on themselves only because of the way they are represented internally: cuts are associated with particular phase *values* as in Fig. 4. A cut, however, physically represents a phase *difference* between two adjacent phase values that may be greater than one-half cycle. Since each phase value is associated with both a row and a column difference, the components of the vector gradient, cut locations cannot be faithfully stored in association with scalar phase values alone. Instead, we associate cuts with phase differences, as in the network model, and we are guaranteed that our tree will not close on itself.

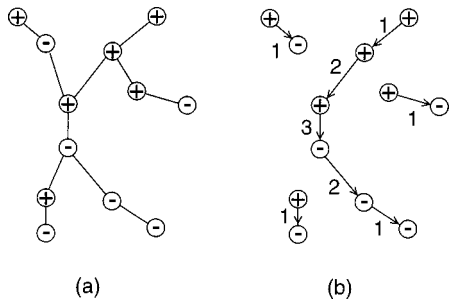

Fig. 3. A minimum spanning tree connects all residues (a). The tree branches are possible locations of discontinuities (b), which occur where flow on the tree is nonzero. The numbers represent flow magnitudes.

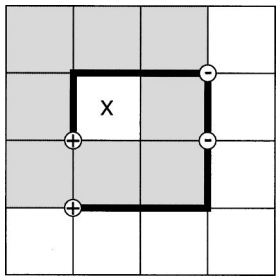

Fig. 4. Residues and cuts are associated with phase *values* (gray boxes) in the original implementation of the residue-cut algorithm. This allows the cuts to close on themselves so that pixel X cannot be unwrapped. When discontinuities are associated with phase *differences* (heavy lines), as in Fig. 1, no pixels are isolated. Note that the tree shown is a spanning tree but not a minimum spanning tree.

It should also be noted that in the original implementation of the residue-cut algorithm, two charges both a distance of d away from a tree might both be added through d distance cuts, even if the addition of the first allows the addition of the second through a shorter cut. This causes some tree branches to be longer than necessary and makes closed cuts more likely. Such behavior can be avoided by following Prim's algorithm more closely. We thus overcome the most significant drawback of the residue-cut algorithm: the closing of cuts.

Further improvement is found in another network idea that allows the specification of weights on the data. Instead of searching for the closest charge to a tree by examining boxes of increasing size, we can use Dijkstra's shortest-path algorithm.[14] This well-known algorithm is used to find the shortest paths from a given source node to all other nodes in some network whose arcs have nonnegative integer lengths or weights. By using Dijkstra's algorithm to find the nearest charge to the tree in each step of Prim's algorithm, the user is able to define weights on phase gradients, thus guiding the placement of tree branches.

Our MST algorithm may be thus summarized. We begin by selecting one arbitrary charge as the source and the beginning of our tree. We find the nearest charge to the tree in the positively weighted network, and we add this new charge to the tree by zeroing the weights of arcs on the path to it. Since all nodes on the tree now have a distance of zero between them, we can locate the nearest charges to any node on the tree. We successively add nearest nontree charges to the tree until all charges are added, resulting in an approximate minimum Steiner tree on the weighted network. In implementation, efficiency can be greatly increased by using special data structures[14] and by recomputing at each stage only the quantities affected by the inclusion of a new tree branch. The boundary of the interferogram can be handled as in the MCF case by connecting all boundary nodes to a single "ground" node that is assigned whatever charge necessary to make the network neutral.[8] This is equivalent to using a superconducting boundary as in the residue-cut case.[10] Note that we do not follow Prim's algorithm exactly in that we allow new tree branches to connect to any node, not just to charges, in the existing tree. This relaxation produces a solution closer to the optimal minimum Steiner tree.

To find the unwrapped phase associated with this tree, either we can use the flood-fill integration method of the residue-cut algorithm since our tree does not close on itself, or we can integrate using our knowledge of the flow on the network as in the MCF algorithm. In the MST algorithm, flow is restricted to arcs on the tree, so a unique flow is defined by the tree and the arrangement of charges. We can easily find this flow by recursively descending the tree from the source, as in a depth-first search, and integrating charge from the tree leaves back up. Doing so verifies that many tree branches do not carry flow and that by combining all of the trees of the residue-cut algorithm into one large tree, we have not added to the overall L^0 discontinuity length.

Since it is derived from the residue-cut algorithm, we would expect the MST algorithm to be just as accurate in the unweighted case, but with the extra advantage that a complete unwrapping is always given. Moreover, if the user specifies weights based on ancillary information, the algorithm can favor cuts in locations where discontinuities are more likely to occur, improving the solution's accuracy. Surprisingly, there is no trade-off in run time for this increase in capability. In fact, our MST implementation runs even more quickly than our residue-cut implementation, probably because the latter is slowed by some redundant searching. Example unwrappings from both are given in Section 5 below.

4. ITERATIVE L^0 IMPROVEMENT: DYNAMIC COST CYCLE CANCELING

The MST algorithm is fast because it approximately minimizes the global L^0 objective by evaluating only local quantities at every step. The drawback of this approach is a loss of accuracy when residues become very dense; for such interferograms, a more complex, iterative algorithm may be more reliable. In this section we develop such an algorithm by extending the network ideas of the linear MCF problem to the concave L^0 case. The technique we use is called cycle canceling, and our adaptation of it may be used to make improvements in the L^0 sense to any unwrapped phase initialization, resulting in an approximate L^0 solution.

The formulation of the L^0 network problem is very similar to that of the L^1 problem solved by the MCF algorithm. In the L^1 case, however, arc cost functions are linear in the amount of flow on the arc. This makes the incremental cost of sending additional flow on an arc independent of the amount of flow already on it. In the L^0 problem, arc costs are zero if there is no flow on the arc or some constant otherwise, so the incremental cost of sending flow on an arc does depend on the existing flow. Despite this important difference, we can still adapt the simple L^1 cycle-canceling strategy from network theory to help us approach L^0 solutions. This cycle-canceling approach is similar to the one used by Flynn in his L^1 minimum weighted discontinuity algorithm.[9] The general L^1 cycle canceling approach is outlined here, and the reader is directed to the references[14] for more detail.

The current state of an iterative MCF (L^1) algorithm may be described by a concept known as the residual

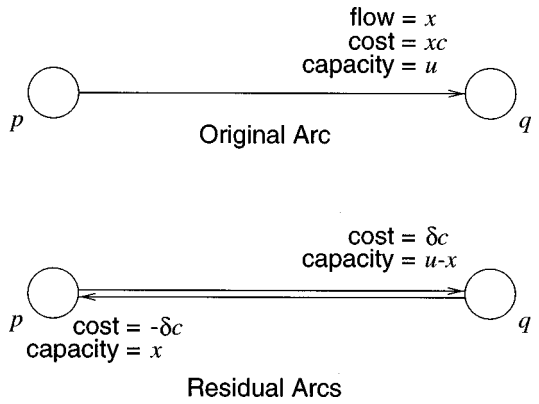

Fig. 5. Original and residual arcs for an L^1 network problem. The residual arcs represent the costs and possible flow paths for an incremental amount of flow δ given the existing flow.

network[14] (here, the word residual is an unfortunate overlap in notation and is unrelated to phase residues). The residual network keeps track of the amount of flow on each arc, and it provides a compact way of determining the costs of additional flows. The nodes of the residual network are the same as the nodes of the original network, and the arcs of the residual network reflect possible paths and incremental costs for additional flow given the existing flow. Suppose the original network contains some arc (p, q), going from node p to node q. This arc has a cost of c units per unit of flow and a capacity, or upper bound, of u units of flow (see Fig. 5). If at some stage of the MCF algorithm there are x units of flow on (p, q), the residual network contains an arc (p, q) with a cost of c units per unit of flow and a capacity of $u - x$. The residual arc reflects the fact that an incremental amount of flow δ of up to $u - x$ units can be sent on the original arc for a total additional cost of $c\,\delta$. The residual network also contains an arc (q, p) in the direction opposite to (p, q) because we can effectively send flow from q to p by letting δ be negative, decreasing the flow on the original arc. Doing so decreases the total cost, so the reverse residual arc has a negative cost of $-c$. We can decrease only as much flow as is on the original arc, though, so the capacity of the reverse residual arc is x.

Next, observe that if we have a feasible flow, its feasibility is preserved when we add flow on a closed, directed cycle as long as capacity constraints are met on the cycle. So, if we have a feasible flow and we augment flow on a cycle with a net negative cost, we obtain a new feasible flow with a smaller total cost (see Fig. 6). In fact, it can be shown that for the MCF objective, a solution is optimal if and only if the residual network contains no negative-cost cycles. These negative cycles can be found by using a variant of Dijkstra's algorithm. Because of negative arc lengths, minor modifications are made, but Dijkstra's algorithm still attempts to find the shortest paths (lengths closest to $-\infty$) from the source to all other nodes in the network. If the network contains a negative cycle, however, the path length to any node in the cycle can be made infinitely negative by repeatedly looping around the cycle. Since Dijkstra's algorithm would become stuck in such loops, it can be used to locate negative cycles. A simple but exact MCF algorithm might thus consist of the

following steps: (1) initialize the network with any feasible flow, (2) use Dijkstra's algorithm to find some negative cycle, (3) augment flow on this cycle so that it is no longer negative, (4) continue, canceling negative cycles until none remain. This approach is easy to explain and implement, but in practice, other techniques are much faster for the MCF problem.[14]

With a few modifications, we now adapt this L^1 cycle-canceling strategy to the L^0 problem. Our approach, which is *not* based on solving the L^1 problem, redefines residual network costs so that they reflect the L^0 objective, allowing the cycle-canceling process to make explicit L^0 improvements to any feasible flow. Because of local minima, however, we cannot guarantee the L^0 optimality of our solution even if it is free of negative cycles. Furthermore, L^0 residual costs depend on the current flow, so we must continually recalculate these costs as we update the solution. For this reason, we call our L^0 algorithm the dynamic cost cycle canceling (DCC) algorithm.

Suppose that the cost of arc (p, q) is zero if there is no flow on it, or a constant c if flow exists (see Fig. 7). Its capacity is u. If there are $x \neq 0$ units of flow on the arc, the residual network will have a forward arc (p, q) with a capacity of $u - x$ for which any amount of additional flow δ incurs no cost. There will also be a reverse arc (q, p) with a capacity of x and a cost of $-c$ if exactly x units of flow are added (removed from the original arc so $\delta = -x$), and zero cost otherwise. If $x = 0$, the residual network has only a forward arc (p, q) with a capacity u and a cost c for any positive increment of flow.

Because of this nonlinearity, we must look for negative cycles of specific increments of flow; that is, we must find cycles for which the augmentation of exactly δ units of flow results in a negative cost. To do so, we calculate residual arc costs for a specific δ and then search for negative cycles. For example, the cycle in Fig. 6(a) has a negative cost with respect to the L^0 metric for $\delta = 2$, but not for $\delta \neq 2$. When we find negative cycles, we simply augment δ units of flow on them, recalculate those costs, and continue. Weights may be assigned to the network by varying the arc costs c. After augmenting δ units of flow on as many negative cycles as possible, we reset the flow increment δ to equal $\delta + 1$, recalculate costs in the residual network, and continue, iterating on δ. Since augmenting δ_1 units of flow on a negative cycle may create other cycles that have negative costs for $\delta_2 \neq \delta_1$, we must keep looping over δ until we are satisfied with the solution. As all network quantities are restricted to be integers, each canceled negative cycle decreases the L^0 objective value by an integer amount, so this value is monotonically nonincreasing, and we need not worry about instabilities or convergence failures.

One more modification must be made to the MCF (L^1) cycle-canceling algorithm before we can use it for the L^0 problem. In the case above, where we have $x \neq 0$ units of flow on (p, q), we have a trivial negative cycle in the residual network: We can send another $\delta = x$ units of flow on (p, q) at zero cost and then send x units of flow back on (p, q) for a cost of $-c$. Physically, this cycle does nothing to the solution since we simply send flow on an arc and immediately cancel it. If these cycles are allowed, however, our cycle-finding routine becomes continually trapped in them. Moreover, the cost of the cycle is not actually negative, since flow on the forward arc changes the cost of the reverse arc. Therefore we modify the cycle-finding routine so that it does not backtrack as described, avoiding the problem of such trivial negative cycles. The unfortunate consequence of this restriction is the fact that other nontrivial negative cycles may also be overlooked. To reduce the number of overlooked cycles, we start our cycle-finding routine from different sources spread throughout the network for each δ. We may still miss some negative cycles, but this is not too troublesome given the knowledge that an optimal solution is not guaranteed even if all negative cycles are found. Furthermore, as pointed out above, an optimal L^0 solution does not guarantee a correct unwrapping.

Any phase unwrapping algorithm can be used to initialize the DCC algorithm, as the cycle-canceling process may proceed from any feasible flow. Good results are obtained both qualitatively and in the L^0 sense when approximate L^0 initializations from the MST algorithm are used. The cycle-canceling process preserves the accuracy of the MST algorithm in areas where the MST algorithm unwraps well, while in residue-dense regions, the DCC algorithm's L^0 improvements correct local errors in the initialization. Other experiments using exact L^1 solutions to initialize L^0 cycle canceling result in DCC unwrap-

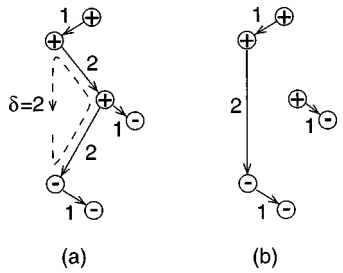

Fig. 6. A nonoptimal flow in the L^1 sense (a) may be improved on by augmenting flow on a closed, directed cycle (dashed line) with a net negative residual cost. The result (b) is a feasible solution with a smaller total cost. The numbers indicate flow magnitudes. The cycle shown also has negative cost with respect to the L^0 metric for $\delta = 2$ but not for $\delta \neq 2$.

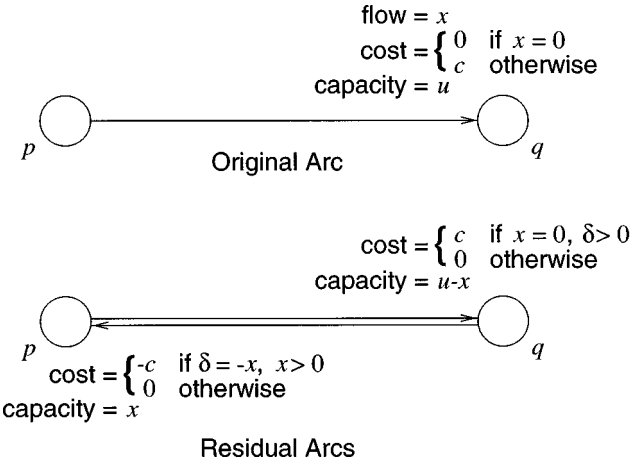

Fig. 7. Original and residual arcs for the L^0 network problem. The nonlinearity of the L^0 objective makes arc costs in the residual network dependent on the flow in the current solution.

pings that are quite similar to their L^1 initializations. Presumably, L^1 solutions are often very near local L^0 minima, so L^0 cycle canceling has little effect on them.

In our implementation, then, we begin the DCC algorithm by initializing the flows with the MST algorithm. We set the flow increment δ to one, locate and cancel as many negative cycles as we can find, and repeat this process several times using different sources for the cycle-finding routine. We then increment δ and begin again. We continue in this manner, resetting δ to one when it becomes larger than the maximum flow in the network. The algorithm terminates when no more negative cycles can be found for any δ or when some limit on the number of iterations has been reached. We can then integrate the phase, using knowledge of the flow.

5. RESULTS

We now present results from the application of our algorithms to interferometric test data, and for comparison we also include solutions from a few other notable algorithms. These are the residue-cut, MCF, and L^p-norm (with $p = 0$) algorithms, described above, and the preconditioned conjugate-gradient algorithm,[15,13] which computes an L^2 least-squares (LSQ) solution. The L^p-norm and LSQ implementations used here also include a final congruence step to make the unwrapped phase differ from the wrapped phase by only integer numbers of cycles.[13]

Because the nuances of real interferometric data are difficult to capture fully in simulation, we test the algorithms on real data and use an available digital elevation model (DEM) as ground truth. The flattened interferogram shown in Fig. 8(a) was acquired by two passes, 105 days apart, of the European Space Agency's ERS-1 satellite over an area of Nevada, north of Death Valley, California. The 1250×830 pixel interferogram is formed from five looks in azimuth and a single look in range, with range shown increasing toward the right. The ground pixel spacing is approximately 20 m in both dimensions. The interferogram magnitude is shown in gray-scale brightness and the phase is shown in color, as indicated by the color bar. The placement of phase residues is shown in Fig. 8(b). The spatial density of these residues corresponds loosely to the magnitude of the interferogram

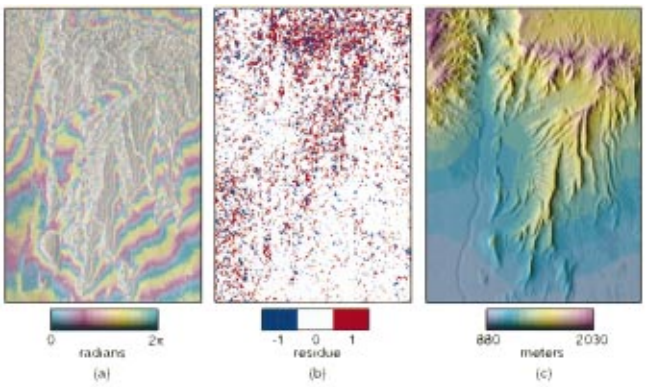

Fig. 8. Data used to test the algorithms: (a) wrapped interferogram with magnitude in gray-scale and phase in color, (b) locations of phase residues, (c) reference DEM with shaded relief in gray-scale and elevation in color.

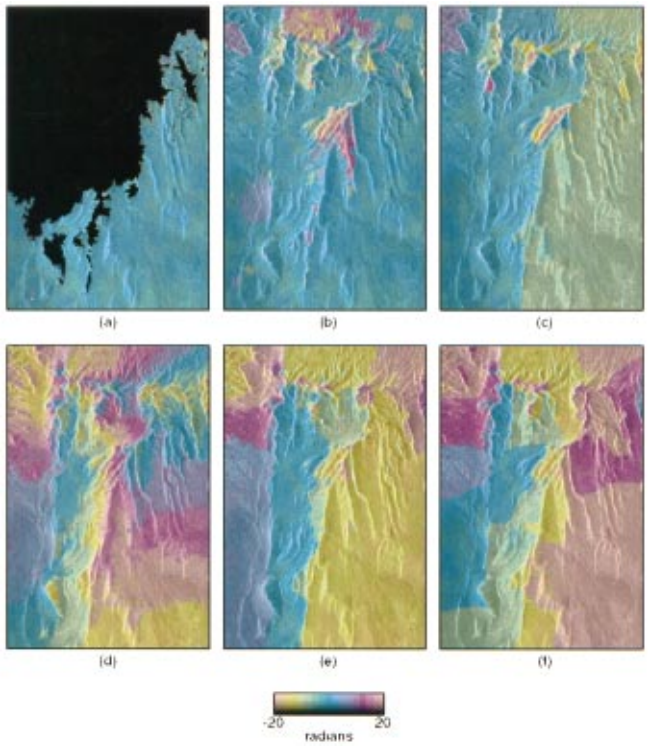

Fig. 9. Algorithm results with uniform weights: (a) residue-cut, (b) MST, (c) DCC, (d) LSQ, (e) MCF, (f) L^p-norm. The color represents relative phase error calculated with the reference DEM.

complex correlation coefficient[22] ρ. Our topographic reference is the 1:24,000 scale U.S. Geological Survey DEM shown in Fig. 8(c), where the gray scale is a shaded relief image generated from the DEM and the color represents elevation in meters. The stated accuracy of the DEM is equal to or better than 15 m.

Since different algorithms fare differently in the presence of different terrain types observed with interferometric radar, we have selected a data set that includes a variety of typical features. In the center of the interferogram are long discontinuities caused by foreshortening and layover. The upper part of the interferogram contains areas of rough topography, while the lower part is relatively flat, though it is not without areas of dense residues.

Figure 9 shows the results from the different algorithms used with uniform weights (i.e., unweighted). For the residue-cut unwrapping [Fig. 9(a)], areas where no solution is produced are shown in black. To make unwrapping errors apparent, the DEM-derived unambiguous phase has been subtracted from the unwrapped solutions so that differences of integer numbers of cycles between areas represent phase unwrapping errors. Some smaller differences of less than one cycle are apparent but do not result from phase unwrapping, as all algorithm implementations used here add only integer numbers of cycles to the wrapped phase. Such discrepancies between the interferogram and the DEM may be attributable to atmospheric artifacts; the differences are consistent with changes on the order of a few percent in relative humidity.[23] Other possibilities include inaccuracies in

the DEM, noise in the interferogram, changes of the topography over time, or inaccuracies in the DEM-to-radar coordinate transformation and registration process. In any case, it is easy to distinguish the patchlike, integer-cycle unwrapping errors from other errors. Of course, there is always an unknown constant offset in interferogram phase, so we can determine error only in a relative sense. Different colored patches thus represent relative errors, where we have chosen a somewhat arbitrary reference phase for presentation purposes. Consequently, the quality of an unwrapped solution may be evaluated by its degree of color segmentation.

The residue-cut algorithm [Fig. 9(a)] does well where it unwraps in the flatter areas, but it cannot unwrap through the residue-dense regions. On the other hand, the MST algorithm [Fig. 9(b)] provides a complete unwrapping that is reasonably accurate in spite of some localized errors in areas of rough topography and dense residues, where the residue-cut algorithm fails to provide a solution. It is important to note that the MST algorithm does not introduce any large-scale global phase errors; that is, its solution is not improperly segmented. The DCC algorithm [Fig. 9(c)] corrects many of the local errors in the MST solution, but it improperly segments the scene into two large regions which differ by one cycle. This segmentation follows the discontinuities formed by layover and foreshortening running in azimuth through the middle of the scene. Individually, the two segments are reasonably accurate. Notably, by nature of the algorithm, the DCC solution is better than the MST solution in an L^0 sense, but in this case the MST solution is qualitatively superior since it is free of global errors. The LSQ [Fig. 9(d)], MCF [9(e)], and L^p-norm [9(f)] algorithms all do very poorly in the unweighted case, segmenting the image into many patches that differ by several cycles of phase. The errors of the LSQ and MCF solutions may be due to the unreliability of the L^2 and L^1 metrics when discontinuities have multiple cycles of phase across them, as illustrated in Fig. 2. Although the L^0 objective may be generally more immune to these types of errors, in this case the L^p-norm algorithm converges on a bad local L^0 minimum.

Uniform weights may be seldom used in practice, but the unweighted results here are illustrative of algorithm behavior in something of a natural state. Knowledge of this behavior is important because our ability to specify proper weights is in general limited and imperfect. The understanding of how an algorithm works without weights may give an idea of its sensitivity to changes in weighting scheme parameters, and it may furthermore suggest altogether new weighting schemes. Various methods have been suggested before,[16,13,18] but this area is neither theoretically nor empirically developed in a definitive way.

Here we examine algorithm performance with two different weighting schemes. The first scheme, whose efficacy has been demonstrated by Pritt[16] and Ghiglia and Pritt,[13] uses a binary mask generated by thresholding the magnitude of the complex correlation coefficient ρ. In our experiments the mask threshold of 0.25 provided the best solutions. These results are shown in Fig. 10, where panel (a) shows the weight mask, and unwrapping results

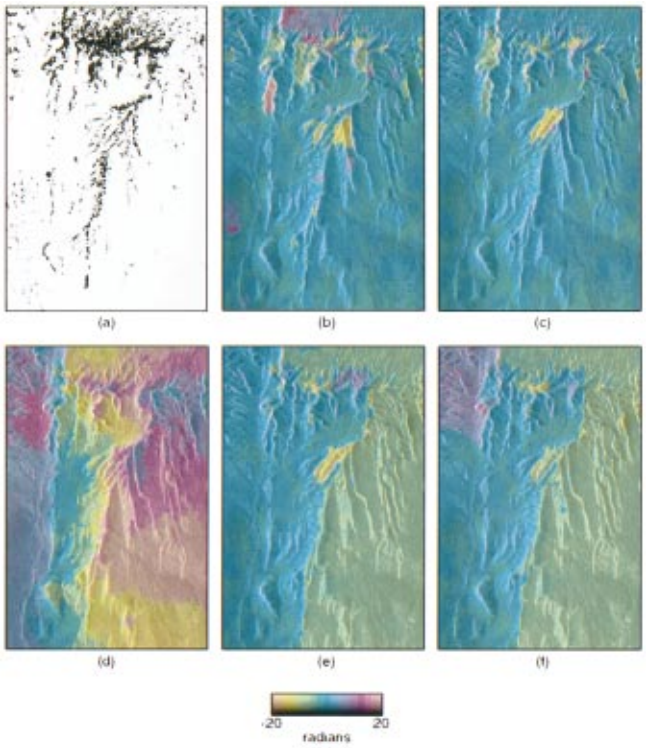

Fig. 10. Algorithm results with interferogram coherence used as weights: (a) weight mask, (b) MST, (c) DCC, (d) LSQ, (e) MCF, (f) L^p-norm. The color represents relative phase error.

are shown analogously to the unweighted case. The residue-cut algorithm does not accept weights, and no result is shown for it. For consistency, we have applied the same weight mask to all algorithms. It should be noted, however, that different algorithms may be more reliable with customized weights. For example, the LSQ and L^p-norm algorithms may give better results when instructed to "fatten" their weight masks by a few pixels,[13] although we do not do so here.

The MST solution [Fig. 10(b)] does not change very much with the introduction of correlation-based weights. Since the algorithm builds its tree by adding nearest residues, discontinuities tend to follow channels of dense residues. These tend to coincide with areas of low interferogram coherence, so in effect, the MST algorithm generates its own correlation-based weights even when none are provided. This helps explain the success of the residue-cut and MST algorithms in the unweighted case. Weighting schemes not based on correlation seem to have a more significant effect on the MST algorithm (see below). The correlation-weighted DCC solution [Fig. 10(c)] is fairly good and is not segmented. There are some local errors, but fewer than in the MST case; the DCC algorithm thus preserves the large-scale accuracy of the MST solution while reducing the number of small-scale errors. The MCF [Fig. 10(e)] and L^p-norm [10(f)] algorithms also produce relatively few small-scale errors, but they both again incorrectly segment the scene. The weights offer a significant improvement over the unweighted cases for these two algorithms, but large-scale errors are nevertheless apparent. The LSQ algorithm [Fig. 10(d)] still performs very poorly, with the weights having only a marginal effect.

Our experiments also include a set of algorithm trials that use edge detection to provide weights. For topographic applications, one may reasonably expect phase discontinuities to coincide with the large changes in image brightness that occur in regions of foreshortening or layover. Our weighting scheme is therefore based on the thresholded output of an edge detection routine applied to the interferogram magnitude. We use the ratio-of-exponentially-weighted-averages operator applied in range for edge information.[24] For the MST, DCC, and MCF algorithms, which accept weights on phase differences rather than phase pixels, the weighting scheme also favors discontinuities where the wrapped gradients are near $\pm\pi$. Results are shown in Fig. 11. Our work with this weighting scheme is preliminary and will be addressed more fully in future work.

The weighted MST solution [Fig. 11(b)] is again accurate globally but still contains localized, small-scale errors in areas of rough topography. The DCC solution [Fig. 11(c)] is quite good and has fewer small-scale errors than the MST solution, though there is a noticeable error near the top of the image. Some errors near the boundaries of the interferogram may be unavoidable, however, especially when large physical discontinuities run out of the scene as is the case here (see the DEM in Fig. 8). The MCF solution [Fig. 11(e)] is also very good and is nearly identical to the DCC solution. When the edge-weighted MCF solution is compared with its unweighted counterpart, the dramatic effect of weights becomes quite apparent. The L^p-norm solution [Fig. 11(f)] also improves a great deal with the inclusion of edge information, but it remains incorrectly segmented. The LSQ algorithm [Fig. 11(d)] once again performs poorly.

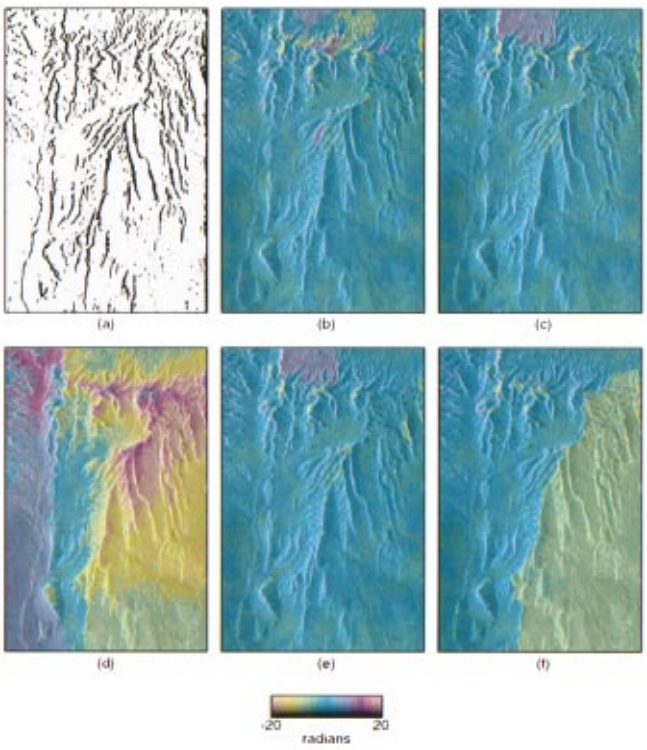

Fig. 11. Algorithm results with edge detection used for weights: (a) weight mask, (b) MST, (c) DCC, (d) LSQ, (e) MCF, (f) L^p-norm. The color represents relative phase error.

Table 1 summarizes different quantitative error measures for each of the unwrappings presented above. However, for evaluating solution quality and algorithm performance, the information available in Figs. 9–11 is more illustrative and complete. Hence, by examining how the table data relate to the errors apparent in Figs. 9–11, we may evaluate the weighting schemes and error metrics themselves. The error metrics listed are the fraction c of pixels unwrapped correctly, the relative error standard deviation σ, and the L^0 and L^1 measures for each of the different weighting schemes. Each column in the table represents one of these error metrics, and each row represents a specific unwrapped solution. For the residue-cut algorithm, the σ and L^p measures are based only on regions where a solution is given and are scaled by the fractional area of coverage. The L^p measures are normalized to the minimum value for each column (residue-cut values are not considered for this minimum).

Without regard to solution accuracy, Table 1 indicates that the DCC and MCF algorithms yield the solutions with the smallest L^0 measures. Interestingly, the MCF algorithm actually aims to optimize the L^1 objective, but as discussed above, L^0 and L^1 solutions may be quite similar.

However, the L^p measures are not very precise indicators of a particular solution's accuracy with respect to qualitative evaluations of Figs. 9–11. These measures are also highly dependent on the weights specified. Ideally, better solutions in a given column would have lower L^p measures, but in the table this is only sometimes the case. For example, inspection of Fig. 9 shows that the MST algorithm provides the best unweighted solution qualitatively, yet all L^p metrics shown suggest it to be among the worst. Such discrepancies lend credence to our earlier assertion that we sacrifice little in computing approximate rather than exact L^0 solutions.

Measures of c and σ agree more closely with subjective quality assessments, but even they are imperfect. For example, a solution that improperly segments the scene into two locally correct halves differing by one cycle might be more desirable than a solution that improperly segments the scene into a checkerboard pattern where alternating squares differ by one cycle. However, the measured values of c and σ would be identical for the two solutions. Moreover, such calculations require *a priori* knowledge of the true topography—knowledge that is usually unavailable.

Table 2 summarizes the approximate execution times and memory requirements of the algorithms on a Hewlett–Packard C-180 workstation. We must emphasize that these parameters are telling only for the implementations used in our tests; the computational costs of any or all algorithms may be reduced with cleverer implementations. As noted above, the MST algorithm is very fast. The LSQ algorithm is also quite fast in the unweighted case when only a single iteration is required, but it runs more slowly when weights are used. Other LSQ techniques promise improvements in speed.[16,17] The MCF algorithm has a moderate run time, but our implementation requires an excessive amount of memory. The DCC and L^p-norm algorithms are both rather slow owing to their nested, iterative structures. The efficiency

Table 1. Summary of Error Measures for Unwrapped Solutions Presented

			Error Metric					
			Unweighted		ρ Weights		Edge Weights	
Algorithm	c	σ (rad)	L^0	L^1	L^0	L^1	L^0	L^1
Unweighted								
Residue-cut[a]	0.51	1.47	0.81	0.80	1.12	1.12	1.07	1.08
MST	0.78	6.12	1.14	1.42	1.19	1.35	1.35	1.64
DCC	0.48	4.66	1	1.08	1.07	1.10	1.16	1.21
LSQ	0.15	18.46	1.73	1.72	2.04	2.04	2.41	2.41
MCF	0.43	11.01	1.01	1	1.09	1.09	1.18	1.18
L^p-norm	0.32	11.97	1.06	1.06	1.15	1.15	1.17	1.17
ρ weights								
MST	0.83	4.05	1.14	1.36	1.09	1.15	1.34	1.54
DCC	0.88	2.87	1.06	1.16	1.01	1.02	1.22	1.30
LSQ	0.22	13.06	1.96	1.95	2.03	2.03	2.65	2.65
MCF	0.47	4.02	1.07	1.07	1	1	1.22	1.23
L^p-norm	0.42	4.62	1.33	1.33	1.11	1.12	1.48	1.48
Edge weights								
MST	0.88	2.75	1.12	1.28	1.16	1.23	1.14	1.20
DCC	0.90	2.48	1.07	1.16	1.12	1.15	1.02	1.03
LSQ	0.25	9.87	1.90	1.89	2.21	2.20	2.32	2.32
MCF	0.90	2.27	1.09	1.10	1.13	1.14	1	1
L^p-norm	0.53	3.76	1.26	1.25	1.31	1.31	1.11	1.11

[a] Reflects only valid areas. Not considered for L^p metric normalization.

Table 2. Computational Costs of Algorithms

	Execution Times (s)			
Algorithm	Unweighted	ρ Weights	Edge Weights	Memory (Mb)
Residue-cut	12			10
MST	8	9	11	50
DCC	2040	3887	5640	50
LSQ	38	220	253	50
MCF	199	348	243	400
L^p-norm	1950	1958	1960	70

of the DCC algorithm may improve a great deal, however, as our simple implementation was designed to test the concept rather than to provide great speed.

6. CONCLUSIONS

The two-dimensional phase unwrapping problem has received a great deal of attention in recent years, and significant progress has been made on the problem both in theory and in practice, but the field can hardly be called mature. Rather, as it is attacked in new and different ways, important insights into both the problem and algorithms for its solution are gained. Our work is meant to further this process.

We have shown that the phase unwrapping problem with use of the L^0 metric is *NP*-hard, implying that it is an intractable problem that is likely impossible for any efficient algorithm to solve exactly. Knowing this, we assert that our efforts should not be focused on elegant, exact mathematical solutions but should be geared toward useful approximate algorithms; we have presented two here: The MST algorithm embodies important improvements to the residue-cut algorithm, and the DCC algorithm extends the network ideas of simple linear MCF algorithms to the concave L^0 case.

Our tests indicate that the two algorithms perform as well as or better than existing algorithms on real data with a wide range of topographic features. The MST solutions presented are consistently free of large-scale segmentation errors. Some parts of MST solutions tend to be unreliable where residues become very dense, but overall, the MST algorithm is quite accurate—an especially impressive fact given its great speed advantage over other algorithms. In the cases when the MST algorithm does produce unreliable results, the DCC algorithm, appropriately weighted, is a more flexible and robust alternative. It does well in cleaning up the small, local errors that may occur in an MST unwrapping. Though slow, the DCC algorithm outperforms the LSQ, MCF, and L^p-norm algorithms overall on the test data here. In this light, it may be used in conjunction with the MST algorithm as something of a "big gun" for difficult interferograms.

That said, we should also point out that different algorithms, and perhaps even different implementations of the same algorithm, each have their own unique behavioral traits. For example, different algorithms may be better suited for interferograms with different features, or weighting schemes that work well for one algorithm may do little for another. A wise approach to phase unwrapping, then, would be to maintain a diverse arsenal of unwrapping algorithms and to know how and when to use each.

Finally, our test results underscore the need for a better framework under which we may evaluate the quality

of phase unwrapping solutions. The L^p-norm formulation, though useful for categorizing algorithms, is not a very precise indicator of a particular solution's accuracy. The idea of keeping the number of physical discontinuities small is appealing and is often generally effective, but there is no physical reason that this number should be at an absolute minimum in the true unwrapped phase. As an error measure, the L^0 metric also depends heavily on the weights specified by the user. A good deal of work should thus be centered around formulating weighting schemes that give physical meaning to the weighted norms. The dramatic benefits of appropriate weights are well known, but the subject deserves a more comprehensive treatment both theoretically and experimentally. Perhaps with the proper combination of weights and objective functions, we will be able to better evaluate quantitatively the accuracy of phase unwrapping solutions and from there design algorithms to provide maximally accurate solutions.

APPENDIX A: *NP*-HARDNESS OF THE L^0 PROBLEM

In this proof we first show that any instance of some *NP*-complete problem can be solved by an algorithm that solves the network equivalent L^0 problem. Then we show that any instance of this network problem can be solved by a general L^0 phase unwrapping algorithm. We begin by formally stating an *NP*-complete problem called the rectilinear Steiner tree (RST) problem[19,25]: let $S \subseteq Z \times Z$ be a set of points at integer locations on a plane and let K be a positive integer; does there exist a connected tree composed solely of horizontal and vertical line segments, which contains all points in S and whose total length is no larger than K? Such a tree is called a rectilinear Steiner tree, and we show here that we will always be able to find one of minimum length for any S by using a hypothetical L^0 algorithm. Clearly we can then answer the question posed by the RST problem for any S and K.

The network equivalent of the L^0 phase unwrapping problem is to find a feasible flow on a network such as that of Fig. 1, which minimizes the total length of arcs carrying flow, where all arcs have unit length. Since the nodes of the network lie on a rectilinear grid, we can assume that they are located at integer points of a Cartesian coordinate system. Let n be the number of points in S. We construct an instance of the network problem such that there are n positive residues at nodes corresponding to the points in S, as in Fig. 12. Suppose that the rightmost positive residue is at (x_0, y_0). We line up n negative residues at $(x_0 + 1, y_0)$, $(x_0 + 2, y_0), \ldots, (x_0 + n, y_0)$.

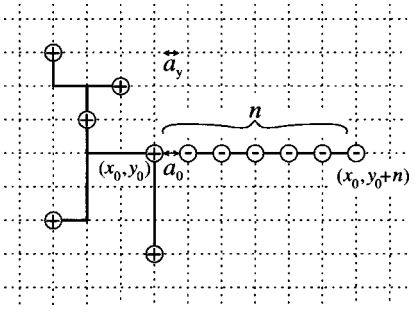

Fig. 12. L^0 network formulation of a RST problem instance. Positive residues are located at the points in S, and negative residues are lined up to the right of the rightmost positive residue. The optimal L^0 solution must be a single minimum Steiner tree, whose length is n units longer than the minimum length for the original RST problem.

Now we prove that the optimal RST length l_{RST} for the points in S will be equal to $l_0 - n$, where l_0 is the total length of the optimal L^0 flow. We first show that the L^0 solution will be a single tree. Let A be the set of all arcs a_y going from (x_0, y) to $(x_0 + 1, y)$. Suppose that there are m disconnected trees in the L^0 solution. Each tree must be neutral, so it must contain at least one distinct arc a_y in order to connect positive and negative residues. Let $y_{\max}$ be the y coordinate of the uppermost of these arcs if any such arcs have $y > y_0$, and zero otherwise. Define $y_{\min}$ similarly for $y < y_0$. Define R as the set of all arcs that are part of the optimal L^0 solution on or to the right of the line $x = x_0$. We can find a lower bound on the total length l_R of these arcs:

$$l_R \geq n + (y_{\max} - y_0) + (y_0 - y_{\min}) + (m - 1). \quad \text{(A1)}$$

The n term on the right-hand side of this equation comes from the fact that at least one horizontal path must connect some positive residue to the rightmost negative residue. Flow must also traverse the vertical distances from $y_{\max}$ and $y_{\min}$ to the negative residues at y_0, explaining the next two terms. The last term includes one unit of length for each of the m arcs a_y used, barring the one included in the first term. Now, observe that we can draw a single tree R' having length $l_{R'}$ and consisting of all arcs between (x_0, y_0) and $(x_0 + n, y_0)$ and all arcs between $(x_0, y_{\max})$ and $(x_0, y_{\min})$. R' contains all of the residues that R contains, so if we substitute R' for R, we still have a feasible flow. Because we assume that the original solution is optimal, l_R can be no greater than $l_{R'}$:

$$l_R \leq l_{R'} = n + (y_{\max} - y_0) + (y_0 - y_{\min}). \quad \text{(A2)}$$

Combining the two inequalities gives us $m = 1$, so the optimal L^0 solution is a single tree that must be a minimum RST. Furthermore, all positive residues are on a connected subtree and all negative residues are on another connected subtree, both of which are minimum Steiner trees for their respective residues.[26] Clearly, the length of arcs from (x_0, y_0) to $(x_0 + n, y_0)$ is n, and the remaining length $l_0 - n$ of the L^0 solution is the length of the minimum RST for the points in S. Knowledge of this length provides us with the answer to the RST problem instance for any K.

So far, we have shown that an L^0 network algorithm can solve the RST problem, so we must now show that any instance of the network problem has a phase unwrapping equivalent. That is, we must be able to generate a wrapped phase field for any instance of the network problem above. Some arrangements of residues, however, do not correspond to realizable phase fields—for example, when many residues of the same sign are all located adjacent to each other. This calls for a slight modification to our formulation of the network RST equivalent. Suppose that the nodes of the network problem are now located at multiples of 0.25 instead of 1 in the Cartesian co-

ordinate system. Residues still lie at their integer coordinates. Since we are using a rectilinear distance metric, the optimal RST length for this problem simply changes by a factor of 4 and we lose no generality.[26] Next, note that a positive residue results from the following 4×4 array of phase values (in cycles):

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0.25 & 0 \\ 0 & 0.75 & 0.5 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}.$$

Therefore to generate the wrapped phase field we simply center a copy of the above array at the locations of all positive residues, and we do the same for negative residues, using the opposite values. We may do this without overlap for all residues because they are now each separated by at least four nodes in the network. After the rest of the array is filled with zeros, the resulting wrapped phase field will have an L^0 solution that corresponds exactly to the solution of the network problem, so any L^0 phase unwrapping algorithm could be used to solve the RST problem for any S and K.

This nearly completes the proof, but one more subtlety must be addressed. The number of nodes in our constructed network problem is related to the spacing of points in the RST problem that we are given, so as this spacing grows, the number of nodes in the network problem may become very large even though the number points in the RST problem may be relatively small. The number of nodes in the L^0 problem thus cannot in general be polynomially bounded by the number of points in the RST problem. The RST problem, however, was shown to be *NP*-complete through a transformation from another *NP*-complete problem called the node cover problem for planar graphs.[25] We will not discuss this problem here but merely note that any instance of it may be posed as an RST problem with a *constant* bound on point spacing. Consequently, any instance of the node cover problem for planar graphs may be solved by an L^0 phase unwrapping algorithm through a polynomial-time transformation, concluding the proof.

Above, we assume that all arcs are uniformly weighted, but we can easily generalize the proof to the weighted case as well, since an algorithm solving the weighted L^0 problem can always be assigned uniform weights, thereby solving the unweighted problem. We also assume above that our L^0 algorithm adds only integer numbers of cycles to the wrapped phase in obtaining a solution. Again, we can generalize the proof by noting that the L^0 optimum will never require the addition of noninteger numbers of cycles. To see this, recall that the network model remains completely general even when noninteger flows are present. Now, suppose that there is a noninteger flow on arc a in some feasible L^0 solution. Because flow is conserved and all residues are integers, there must be at least one other arc with a noninteger flow connected to each end point of a. Extending this analysis further, a must be part of a closed loop of noninteger flow as in Fig. 13. We can therefore reduce the total discontinuity length of the solution by simply deleting one arc from the loop and reallocating the flows, which then become integers, on the remaining arcs. The new solution still contains the same residues, so it must still be feasible. Since any solution containing noninteger flows can be improved on in the L^0 sense, an optimal L^0 solution will never contain noninteger flows and will therefore never require the addition of noninteger numbers of cycles to the wrapped phase.

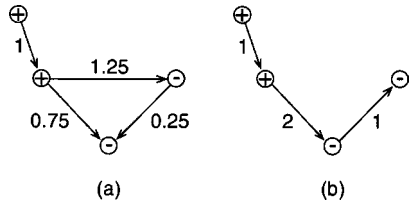

Fig. 13. Part of an L^0 solution, where the numbers indicate flow magnitudes. (a) Noninteger flows must be arranged in closed loops if the net flow out of a node is equal to the node's charge. (b) The loop can always be broken and the flows reallocated to form integer flows constituting a better feasible L^0 solution.

It should be noted in passing that the RST problem with use of a Euclidean distance metric has also been shown to be *NP*-complete.[27] Also, the L^0 network problem is actually a special case of yet another *NP*-hard problem called the minimum concave cost network flow problem.[28] This problem is similar to the MCF problem, but arc cost functions are concave rather than linear. We have shown that even given its special network structure, the L^0 problem remains *NP*-hard.

ACKNOWLEDGMENTS

Our MST and DCC algorithm implementations draw on the SPLIB shortest-path codes written by Boris Cherkassky, Andrew Goldberg, and Tomasz Radzik. Our MCF implementation uses a modified version of the CS2 MCF solver written by Goldberg and Cherkassky and copyrighted by IG Systems, Inc. We use the residue-cut implementation written by Richard Goldstein, Howard Zebker, and Charles Werner, and we use LSQ and L^p-norm implementations written by Dennis Ghiglia and Mark Pritt and copyrighted by John Wiley and Sons, Inc. Moses Charikar provided helpful feedback on complexity issues. The anonymous reviewers are also due thanks for their comments. This work was supported under a contract from the University of Alaska Geophysical Institute.

Corresponding author Howard Zebker can be reached by phone, 650-723-8067; fax, 650-723-9251; or e-mail, zebker@stanford.edu.

REFERENCES

1. H. Zebker and R. Goldstein, "Topographic mapping from interferometric SAR observations," J. Geophys. Res. **91**, 4993–4999 (1986).
2. H. A. Zebker, T. G. Farr, R. P. Salazar, and T. H. Dixon, "Mapping the world's topography using radar interferometry: the TOPSAT mission," Proc. IEEE **82**, 1774–1786 (1994).
3. A. K. Gabriel, R. M. Goldstein, and H. A. Zebker, "Mapping small elevation changes over large areas: differential radar interferometry," J. Geophys. Res. **94**, 9183–9191 (1989).

4. D. Massonnet, M. Rossi, C. Carmona, F. Adragna, G. Peltzer, K. Feigl, and T. Rabaute, "The displacement field of the Landers earthquake mapped by radar interferometry," Nature (London) **364**, 138–142 (1993).
5. H. A. Zebker, P. A. Rosen, R. M. Goldstein, A. Gabriel, and C. Werner, "On the derivation of coseismic displacement fields using differential radar interferometry: the Landers earthquake," J. Geophys. Res. **99**, 19617–19634 (1994).
6. R. M. Goldstein and H. A. Zebker, "Interferometric radar measurements of ocean surface currents," Nature (London) **328**, 707–709 (1987).
7. R. M. Goldstein, H. Englehardt, B. Kamb, and R. M. Frolich, "Satellite radar interferometry for monitoring ice sheet motion: application to an Antarctic ice stream," Science **262**, 1525–1530 (1993).
8. M. Costantini, "A novel phase unwrapping method based on network programming," IEEE Trans. Geosci. Remote Sens. **36**, 813–821 (1998).
9. T. J. Flynn, "Two-dimensional phase unwrapping with minimum weighted discontinuity," J. Opt. Soc. Am. A **14**, 2692–2701 (1997).
10. R. M. Goldstein, H. A. Zebker, and C. L. Werner, "Satellite radar interferometry: two-dimensional phase unwrapping," Radio Sci. **23**, 713–720 (1988).
11. J. R. Buckland, J. M. Huntley, and S. R. E. Turner, "Unwrapping noisy phase maps by use of a minimum-cost-matching algorithm," Appl. Opt. **34**, 5100–5108 (1995).
12. D. C. Ghiglia and L. A. Romero, "Minimum L^p-norm two-dimensional phase unwrapping," J. Opt. Soc. Am. A **13**, 1999–2013 (1996).
13. D. C. Ghiglia and M. D. Pritt, *Two-Dimensional Phase Unwrapping: Theory, Algorithms, and Software* (Wiley, New York, 1998).
14. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications* (Prentice-Hall, Englewood Cliffs, N.J., 1993).
15. D. C. Ghiglia and L. A. Romero, "Robust two-dimensional weighted and unweighted phase unwrapping that uses fast transforms and iterative methods," J. Opt. Soc. Am. A **11**, 107–117 (1994).
16. M. D. Pritt, "Phase unwrapping by means of multigrid techniques for interferometric SAR," IEEE Trans. Geosci. Remote Sens. **34**, 728–738 (1996).
17. G. Fornaro, G. Franceschetti, R. Lanari, and E. Sansosti, "Robust phase unwrapping techniques: a comparison," J. Opt. Soc. Am. A **13**, 2355–2366 (1996).
18. H. A. Zebker and Y. Lu, "Phase unwrapping algorithms for radar interferometry: residue-cut, least-squares, and synthesis algorithms," J. Opt. Soc. Am. A **15**, 586–598 (1998).
19. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, Calif., 1979).
20. F. K. Hwang, "On Steiner minimal trees with rectilinear distance," SIAM J. Appl. Math. **30**, 104–114 (1976).
21. N. H. Ching, D. Rosenfeld, and M. Braun, "Two-dimensional phase unwrapping using a minimum spanning tree algorithm," IEEE Trans. Image Process. **1**, 355–365 (1992).
22. H. A. Zebker and J. Villasenor, "Decorrelation in interferometric radar echoes," IEEE Trans. Geosci. Remote Sens. **30**, 950–959 (1992).
23. H. A. Zebker, P. A. Rosen, and S. Hensley, "Atmospheric effects in interferometric synthetic aperture radar surface deformation and topography maps," J. Geophys. Res. **102**, 7547–7563 (1997).
24. R. Fjørtoft, A. Lopès, P. Marathon, and E. Cubero-Castan, "An optimal multiedge detector for SAR image segmentation," IEEE Trans. Geosci. Remote Sens. **36**, 793–802 (1998).
25. M. R. Garey and D. S. Johnson, "The rectilinear Steiner tree problem is *NP*-Complete," SIAM J. Appl. Math. **32**, 826–834 (1977).
26. M. Hanan, "On Steiner's problem with rectilinear distance," J. SIAM Appl. Math. **2**, 255–265 (1966).
27. M. R. Garey, R. L. Graham, and D. S. Johnson, "The complexity of computing Steiner minimal trees," SIAM J. Appl. Math. **32**, 835–859 (1977).
28. G. M. Guisewite and P. M. Pardalos, "Minimum concave-cost network flow problems: applications, complexity and algorithms," Annals Oper. Res. **25**, 75–100 (1990).