Finding Tree Structures by Grouping Symmetries

Hiroshi Ishikawa[†]
hi@nsc.nagoya-cu.ac.jp

Davi Geiger[‡]
geiger@cs.nyu.edu

Richard Cole[‡]
cole@cs.nyu.edu

[†]Department of Information & Biological Sciences
Nagoya City University
Nagoya 467-8501, Japan

[‡]Courant Institute of Mathematical Sciences
New York University
New York, NY 10012, U.S.A.

Abstract

The representation of objects in images as tree structures is of great interest to vision, as they can represent articulated objects such as people as well as other structured objects like arteries in human bodies, roads, circuit board patterns, etc. Tree structures are often related to the symmetry axis representation of shapes, which captures their local symmetries. Algorithms have been introduced to detect (i) open contours in images in quadratic time (ii) closed contours in images in cubic time, and (iii) tree structures from contours in quadratic time. The algorithms are based on dynamic programming and Single Source Shortest Path algorithms. However, in this paper, we show that the problem of finding tree structures in images in a principled manner is a much harder problem. We argue that the optimization problem of finding tree structures in images is essentially equivalent to a variant of the Steiner Tree problem, which is NP-hard. Nevertheless, an approximate polynomial-time algorithm for this problem exists: we apply a fast implementation of the Goemans-Williamson approximate algorithm to the problem of finding a tree representation after an image is transformed by a local symmetry mapping. Examples of extracting tree structures from images illustrate the idea and applicability of the approximate method.

1. Introduction

The problem of finding various structures in images has been a focus of major activities in computer vision. Generally, local features such as edges are relatively easily found but by themselves are not enough for producing useful description of the image. One of the ways to extract useful information is trying to find certain larger structures by grouping local features. Detecting open image contours was the first of such efforts. Here, implicitly or explicitly, the problem is placing a known structure (a string of points) on the image plane so that certain quantity is optimized. Detecting closed contours is also important since a closed curve gives a segmentation of a region in the image. This problem can also be formulated as an optimization problem explicitly.

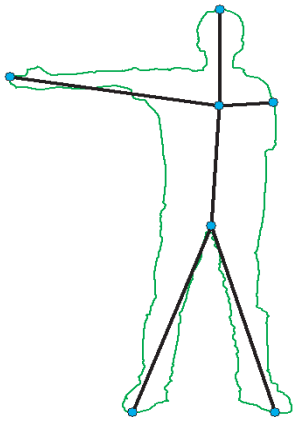


Figure 1. A silhouette of a sketch of person and a possible tree structure representing the silhouette.

As a structure to be found in images, tree structure is also of great interest to vision, as it can represent articulated objects such as people, as well as other structured objects like arteries in human bodies, roads, circuit board patterns, etc (see Figure 1). Tree structures are often related to the symmetry axis representation of shapes, which captures their local 2D symmetries. The importance of symmetry in locating useful information has been pointed out, and detection of local symmetry, sometimes called the symmetry transformation, has been pursued by many. Thus, tree structures are to local symmetries what contours are to local edges.

However, in contrast to the situation in contour detection,

tree-finding has not been characterized as grouping of local clues of 2D symmetry. Instead, existing work on finding tree structures have focused on first finding a closed contour and then computing the symmetry-axis of the shape, although in most applications detecting the outline contour itself is part of the problem. This inevitably loses most information in the image in the early stages of the process: when the outline is being detected, the importance of the local edge for the purpose of finding tree structure is not being considered. Thus, directly linking local features and more global structure is important and, should be the first formulation to consider at any rate, as in general it is a good idea to consider keeping all information available as late in the process as possible.

We suggest in this paper that it is because of the inherent complexity of finding tree structures by grouping local features that prevented clear formulation of this problem as an optimization problem. The optimization problem of finding tree structures in images is essentially equivalent to a variant of the Steiner Tree problem, which is NP-hard. Nevertheless, an approximate polynomial-time algorithm exists, and we use a fast implementation of the algorithm to illustrate an implementation.

It is interesting to note that at the same conference, ICCV'87, in which Kass, Witkin and Terzopoulos published the "snake paper" [12], the same authors also reported the work [26] that addresses the fact that object information contains symmetry information which is richer than just contour information. Somehow that work have not developed further, in part, we suspect, because of this complexity of the problem.

2. Related Work

In detecting open image contours, implicitly or explicitly, the problem is placing a known structure (a string of points) on the image plane so that certain quantity is optimized. It is explicit in the work on snakes [12], which clearly defines the problem as an optimization problem that minimizes an energy that includes both the data terms that favor contours passing through points of high intensity gradient magnitude (local feature) and the regularizing terms that tend to keep the contour short or smooth. The work of Montanari [15] is the first to present the view that these edge grouping computations can be best described by a graph representation and solved by the use of graph algorithms. In Montanari's work, dynamic programming is used to globally optimize the criterion. Many others have followed it (e.g., [3, 9, 20].)

In detecting closed contours, [8] developed a method for finding closed contours using chains of tangent vectors. This problem can also be formulated as an optimization problem explicitly. [7] uses a ratio energy of a generalized area to a generalized length, and use a "pinned ratio" algorithm to optimize it. [11] defines a form of energy functional for the modelling and identification of regions and their boundaries in images, which can be globally optimized using polynomial-time graph algorithms.

Detection of local symmetry, sometimes called the symmetry transformation, has been pursued by many, including [17, 3, 27, 14, 5, 23]. These have not somehow lead to the explicit grouping of these features to comprise a tree structure. For example, roads are usually defined by the pair of boundary lines, with the line in the middle as the symmetry axis. As the road bifurcates, the symmetry axis becomes a tree structure. While the detection of roads through the middle line has been addressed by [3] using dynamic programming, the problem of bifurcation of roads, and therefore the extraction of tree structures, has not been addressed. Indeed, the dynamic programming approach cannot extend to the extraction of trees.

Instead, existing work on finding tree structures have focused on first finding a closed contour. Given an outline contour of a shape, one can apply the symmetry-axis computations to extract tree structures representing the symmetry axis. For instance, [21, 16, 18, 22, 28, 19]. In the work of [24, 25] the awareness for the problem of detecting symmetry structure in images is raised, but not addressed as a graph problem to extract a tree structure.

All the problems above, including finding the symmetry axis of a closed contour, can be solved exactly in a reasonable (i.e., polynomial) time. [15] has shown that detecting open contours can be solved in $O(n^2)$ time and we have shown that this problem can indeed be solved in $O(n \log n)$ time [9], where n is the number of pixels in the image. As we mentioned above, closed contours can be found in $O(n^3)$ time ([11]). Finally, it has been shown that this problem can be solved in $o(L^2)$ where L is the size of the contour ([13]).

In contrast, directly finding tree structures by grouping local features can be directly mapped to a known NP-hard problem, as we show in this paper.

2.1 Our Contribution

The main contribution of this paper is to show that the problem of detecting tree structures in images can be mapped to a variant of the Steiner Tree problem. As this problem is NP-hard, we also introduce an approximate method and illustrate the algorithm with a fast implementation.

In order to formulate the problem, we create a graph $G(V, E)$ where each vertex $v = (p, \theta)$ represents a pixel location p and orientation θ, and graph edges represent geometrical constraints between the vertices. Image responses at each vertex are described by a quantity $S(p, \theta)$, which

possibly represent symmetry information.

In this graph, we define a nonnegative cost function on both vertices and edges. The cost function gives a vertex larger value when the resopnse $S(p, \theta)$ is stronger, i.e., the larger the value is, the more likely it is to be part of the tree solution. (Perhaps it should be called the benefit rather than the cost; but we call it the cost as in "the cost of not including in the tree." See the algorithm below.)

The edges can also have costs. In addition to an optional measure $S(p, \theta)$ (for example representing the symmetry axis) a measure of geometrical constraint/consistency between the two vertices it connects gives rise to this cost. The cost in the case of edges is defined so that edges with larger cost are less likely to be included in the extracted tree.

Once the cost is defined, we would like to extract a tree out of this graph as a subset of vertices and edges, so as to minimize the sum of the cost of the edges in the tree and the cost of vertices *not* included in the tree.

Unfortunately, this is a known NP-hard problem. Nevertheless, there are polynomial-time algorithm that approximates the optimal solution with precise error bounds. In particular we consider the Goemans-Williamson approximate algorithm.

To illustrate how this algorithm may be used for computer vision, we consider the following application: we first apply a symmetry transformation to produce local clues for the presence of 2D symmetries. The symmetry transformation is a map from an image to its "symmetry map," which represents the symmetry strength and orientation at each location:

$$I(p) \rightarrow S(p, \theta)\,, \tag{1}$$

where $S(p, \theta)$ is the strength of the symmetry. A precise description is presented in Section 4.1. Once the symmetry transformation is applied, we create a graph $G(V, E)$ where each vertex $v = (p, \theta)$ represents a pixel location p and orientation θ, and edges represent geometrical constraints between the vertices.

Even though our mapping of the problem of extracting tree structures to the Steinner problem does not require any specific representation of the information at the vertex, it is more intuitive we exemplify with the symmetry transformation. Thus the rest of the paper is organized as follows: The next section is the heart of the work, the approach and graph method to extract tree structures. In section 4 we first discuss an example of local features, the symmetry transformation, in more detail; then we give our initial results of expriment. Section 5 concludes the paper.

3. Extracting Tree Structures

We first assume that we have local features $S(p, \theta)$ computed. As in the case of object boundaries, it is not enough to have local features. We would like to somehow group the features into more structured entity. Here, we group the pixels into a tree structure in the hope that the resulting tree represents the symmetry axis of the object shape.

We build an undirected graph with the vertices corresponding to all pixel-orientation pairs (p, θ). The connectivity of the graph is such that two vertices corresponding to pixel-orientations (p, θ) and (q, ϕ) are connected if (a) p and q are neighboring pixels, or (b) $p = q$ and θ and ϕ are neighboring orientations.

Each vertex has a nonnegative cost. For vertex v corresponding to the pixel-orientation pair (p, θ), we use $S(v) = S(p, \theta)$ for its cost.

Edges shall have the following cost:

$$C(e(u, v)) = C(e((p, \theta), (q, \phi))) = h(|\theta - \phi|)\,, \tag{2}$$

with a function $h(d)$ that decreases with d, so that the pair of vertices at both ends of the edge have consistent orientations and the tree is as smooth as possible.

We then define the total cost of any tree solution, T, in the graph by

$$E(T) = \sum_{e(u,v) \in T} C(e(u, v)) + \sum_{v \notin T} S(v)\,. \tag{3}$$

Note that the sum of the symmetry strength is over the vertices not in the tree solution. The symmetry strength can be called the prize of the vertices. In this way a balance occurs between choosing a large tree T (so that very few remaining vertex prizes contribute to the total cost) and choosing a small tree (so that the cost of the edges is small).

The problem is then defined as how to extract tree structures from this graph that minimizes the total Energy $E(T)$ of the tree. Let us discuss how this problem is formulated as a variant of the Steiner Tree problem.

3.1 Steiner Tree Problem and Variants

Let $G = (V, E)$ be an undirected graph having n vertices and m edges together with nonnegative edge lengths d_{uv}. (We use the word "length" instead of "cost" in accordance with the tradition of the problem; the name also has certain intuitive significance.)

In the **Prize Collecting Steiner Tree Problem**, each vertex u of G has an associated nonnegative penalty p_u. The aim is to find a tree in the graph such that the sum of the length of the edges in the tree plus the penalties of the vertices not in the tree is minimized.

The Goemans-Williamson clustering technique approximately solves this problem in $O(n^2 \log n)$ time and with the approximation factor guaranteed to be at most $2 - \frac{1}{n-1}$ [10]. It is at the core of several approximation algorithms, including those for Generalized Steiner Trees, Prize Collecting Travelling Salesman, 2-Edge Connected Subgraph.

Several improvements have been made since this algorithm was proposed. In our application, the graph is very sparse since each vertex has very few edges compared to the size of the graph. The most efficient algorithm is due to [6]. The implementation gives an approximate solution within a factor $2 + \frac{2}{n^k}$ of the optimal in $O\left(k(n+m)\log^2 n\right)$ time, for any constant k. This time bound is a substantial improvement on other algorithms for graphs which are not too dense. However, it suffers a slight $\frac{1}{n^k}$ additive degradation in the approximation factor, where k can be made as large as required; the running time increases linearly in k.

3.2 Goemans-Williamson Clustering Technique

Although the following algorithm is essentially the same as the one in the reference above, it is specialized for the Prize Collecting Steiner Tree Problem for the convenience of the reader.

The algorithm proceeds in two steps. The first, Clustering Step, has several rounds, each of which identifies one new edge; the step returns the set of the edges, which is a forest. The second step, called the Pruning Step, considers for each vertex in V a tree that contains the vertex as the root by discarding some of the edges in the forest; then it returns the best tree.

Clustering Step. In each round, the algorithm maintains a partition of V into disjoint subsets; some of these subsets are *active* and the rest are *inactive*. We denote the set containing vertex u by S_u. The algorithm also maintains a *residual potential* $P(S)$ for each subset S, a length d_u for each vertex u, and a set F of edges that will be the output of this step. A subset S is active if and only if $P(S)$ is positive.

At the beginning of the first round, the subsets and values are as follows. Each vertex u is in a singleton subset: $S_u = \{u\}$. The residual potential is defined as $P(S_u) = p_u$. The length d_u is initially set to 0 for all vertices u. The set F begins empty.

In a general round, the lesser of the following is found:

1. The least value of $\frac{d_{uv}-d_u-d_v}{f_u+f_v}$ for an edge $e = (u, v)$ such that $S_u \neq S_v$, where $f_u = 1$ if and only if S_u is active in this round.

2. The least value of $P(S)$ for an active subset S.

We denote the resulting value by ϵ. Then, we decrease the residual potential $P(S)$ by ϵ for each active subset S, and increase the length d_u for each vertex u contained in an active subset by ϵ. Next, if the value ϵ has been found by the case 1 above, the edge e is now added to the set F. Also, S_u and S_v are now replaced by $S_u \cup S_v$, which is given the sum of the residual potentials of S_u and S_v as its residual potential and designated as active for the next round according to the new residual potential. If ϵ has rather come from the case 2 above, we make the subset S inactive. The rounds continue until there are no more active subsets remaining.

Definition. Given a forest F and a partition of the vertices V into subsets, the forest induced in F by this partition is defined as the forest obtained from F by shrinking each subset into a single vertex and removing self-loops.

Pruning Step. For each vertex r in V, a tree that contains r is produced by discarding some of the edges in the forest F as follows.

For each edge e in F, we decide whether or not to delete it as follows, in reverse order in which it was added. Consider the subsets into which V is partitioned just before e was added. Consider the forest induced in F by this partition. This forest has a vertex for each subset at the time e was added and has e and other edges that are added after e in Step A. Now we remove those edges that have already been deleted in Step B. Edge e is removed if and only if one of its endpoints in this forest is an inactive leaf (inactivity is with respect to the round which added e) that does not contain r. Finally, after each edge is considered, the connected component containing r is returned as the tree for r.

After we have performed the above procedure for each r, we evaluate the sum of tree-edge lengths and the sum of non-tree vertex penalties for each tree for r. After all the vertices are considered, we choose the tree with the minimum sum.

3.3 The implementation details

The most time-consuming part of the algorithm is in the Clustering Step. In the original Goemans-Williamson implementation [10], it took $O(n^2 \log n)$ time. This is shortened in the implementation by [6] using a technique called the Dynamic Edge Splitting. Consult [6] for the reasoning behind the algorithm details in the following.

The algorithm keeps a heap $heap(S_u)$ maintaining for each active subset S_u the edges that has an end in the subset, with the key value $d_{uv} - d_u$ for an edge (u, v). We denote the heap structure as a whole by H. The heaps in H can be melded when the corresponding subsets are united. The heap supports a Findkey operation that gives the key value for an edge. It also supports an Offset operation that reduces the key values for all edges at the same time. We also keep a Union-Find Structure to maintain the active subsets and the value of residual potential for the subset. In the course of the algorithm, edges are split by adding a vertex, which is called the s-vertex and has no penalty.

The algorithm begins by splitting each edge into two pieces, making each vertex a singleton subset, each vertex with positive penalty active, and initializing the heap structure H with each edge piece having key value equal to its length.

In each round the algorithm performs the following tasks.

Step 1: Choosing the Next Edge. First, it performs DeleteMins from H repeatedly until an edge piece connecting distinct subsets is found. It also finds the active subset with the least residual potential. If the latter value is smaller, the subset is made inactive, the value is subtracted from the key values of H by the Offset operation, and this round is concluded. Otherwise, this edge piece $e' = (v', w')$ is now added to the forest F. Let $key(e')$ be the key (obtainable using the Findkey operation) of e' in either $heap(S_{v'})$ or $heap(S_{w'})$, whichever is active (if both are active, take the smaller of the two keys) and Offset H by $key(e')$. Then the sets $S'_{v'}$ and $S'_{w'}$ are united.

Step 2: Splitting Edges. At most one of v' and w' can have the property that it is an s-vertex and is contained in an inactive singleton set at the beginning of the current round. If neither v' or w' has this property, then we skip this step. Otherwise, without loss of generality, let v' have this property. Being an s-vertex, v' has exactly one edge piece other than e' incident on it. We will split this edge piece $f = (u', v')$ as follows.

To split f, the value of $d_{u'}$ is needed. This is not directly available, but can be computed using the Findkey operation on f, $S_{u'}$ and then using the relation that the key for f in $heap(S_{u'})$ is $d_{u'v'} - d_{u'}$. Once $d_{u'}$ is determined, f is repeatedly split by adding a sequence of vertices on f at distances $\frac{d_{u'v'}}{2}, \frac{d_{u'v'}}{4}, \ldots, \frac{d_{u'v'}}{2^i}$ from v', where i is the least number such that one of $d_{u'v'} - \frac{d_{u'v'}}{2^i} > d_{u'}$ and $\frac{d_{u'v'}}{2^i} \leq \frac{d_{uv}}{2^k}$ hold, where (u, v) is the original graph edge of which f is a part. (If $i = 0$, we don't split f.) Call the last vertex in the sequence u'''. One new singleton inactive subset $\{u'''\}$ is added to the Union-Find Structure and the heap structure, and the two new edge pieces incident on u''' are added to the heap structure; all other new s-vertices created are made part of $S_{u'}$ and all other new edge pieces created are added to the forest to be output. The edge piece f itself is removed from all the structures.

Step 3: Melding Heaps. If neither v' nor w' have the above property required for edge splitting, then $heap(S_{v'})$ and $heap(S_{w'})$, are melded, the active status of $S_{v'} \cup S_{w'}$ is determined.

4. Illustration: Experiments

We now describe an example of actual local symmetry feature and then the result of tree-extraction using it.

4.1 Symmetry Transformation

As an example of "off the shelf" local symmetry features, here we describe a simple "symmetry transform" that gives a representation of the image that captures symmetry information. In most applications the symmetries are between pairs of intensity edges. However, sometimes the symmetries can be directly related to the image intensity value itself: in the medical imaging domain, for example, arteries are usually darker in the center and brighter as one moves outwards towards the boundaries.

4.2 Intensity representation

In cases such as the artery images where the darker pixel is more likely to be located on the symmetry axis, a function of the form

$$S(p, \theta) = \frac{1}{Z} e^{-\alpha[I(p)+I(p+\delta)]} \tag{4}$$

measures how likely the pixel at p with orientation θ belongs to the symmetry axis. The vector δ brings a pixel location p to its immediate neighbor in the direction of θ. The parameter α controls the relative strength of different greyscale values. The larger α is, the sharper the "symmetrical" difference is among grey value pixels.

4.3 Edge/Contour representation

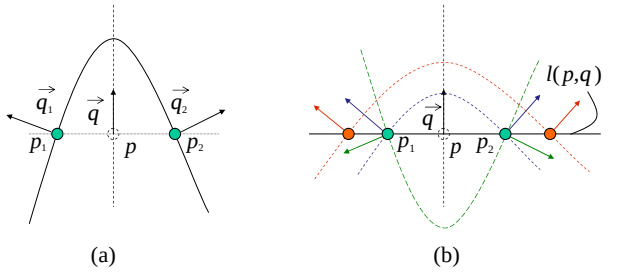
Many images contain intensity edges that form object shape boundaries when they are appropriately grouped. These intensity edges are characterized by their large magnitude of the intensity gradient vector, represented as $\vec{\nabla} I(p)$. The gradient vector is large where there is a large change in intensity and points normal to the intensity boundary. The symmetry axis is the locus of the symmetric points. Often, a good characterization of the symmetry axis location is: a point-orientation pair (p, θ) belongs to the symmetry axis if there is a pair of pixels in the object boundary with their normal vectors mirror symmetric with respect to (p, θ). More precisely, given two points on the object boundary located at p_1 and p_2 with normal vectors $\vec{\theta}_1$ and $\vec{\theta}_2$ respectively (see Figure 2 a.), then the midpoint $p = \frac{1}{2}(p_1 + p_2)$ is on the symmetry axis with orientation $\vec{\theta} = \vec{\theta}_1 + \vec{\theta}_2$, if

$$(p_1 - p_2) \perp (\vec{\theta}_1 + \vec{\theta}_2) \quad \text{and} \quad (p_1 - p_2) \parallel (\vec{\theta}_1 - \vec{\theta}_2)\,. \tag{5}$$

We can define the symmetry transformation via a voting scheme for each pixel p and orientation θ (also defined by the vector $\vec{\theta}$) as the measure

$$S(p, \theta) = \sum_{p_1 \in l(p,\theta)} \sum_{\theta_1} |\vec{\theta}_1 \cdot \vec{\nabla} I(p_1)||\vec{\theta}_2 \cdot \vec{\nabla} I(p_2)| g(|p_1 - p|), \tag{6}$$

where $p_2 = 2p - p_1$, $\vec{\theta}_2 = \vec{\theta} - \vec{\theta}_1$, and $l(p, \theta)$ is the line perpendicular to the vector $\vec{\theta}$ passing through p. Also, we have



(a) (b)

Figure 2. (a) Points on the object boundary that are symmetric. (b) the voting scheme contributors to the strength $S(p,\theta)$.

used the constraint that the pixel $p_2 = 2p - p_1$ with angle θ_2, must be mirror symmetric to θ_1 at p_1, i.e., $\vec{\theta}_1 + \vec{\theta}_2 = \vec{\theta}$. The quantity $|\vec{\theta} \cdot \vec{\nabla}(p)|$ gives the intensity edge response at pixel p and at orientation θ. The line $l(p,\theta)$ can be extended to infinity, i.e., until reaching the end of the image frame. We also include a function $g(|p_1 - p|)$ that decreases with the distance between points, so that the strength of the contributions is diminished as pairs of points are further away from the center of the symmetry (at p). Thus, the measure $S(p,\theta)$ given by (6), or symmetry transform, is given by the intensity edge strength of pair of candidates to object boundaries that are mirror symmetric, factored by a term that decreases with the distance between the symmetric boundary points.

To give an idea of these quantities, Figure 3(b) shows the edge responses of image Figure 3(a) and Figure 3(c) displays the edge symmetry strength function.

4.4 Experiments

In order to illustrate the effect of the algorithm and to show that it is possible to extract objects by extracting the tree structures that best represent the objects, we applied the method to an image. Figure 3 shows the result. Oriented edges were extracted and then $S(p,\theta)$ was computed according to the formula (6). In order to demonstrate the results on an image, we show $S(p) = \max_\theta S(p,\theta)$. This quantity illustrates how the symmetry strength at every pixel can be captured by this transformation.

Once the symmetry transform was obtained and the construction of the graph $G(V,E)$ complete, we assigned the values $S(p,\theta)$ to each vertex u in the graph and assigned the smoothness constraint on the edges $e(u,v)$ of the graph. Then we applied the approximate tree finding algorithm to extract optimal tree structures of the image. The tree structure captures the symmetry axis of the objects, even though no representation of the object was given a priori. It takes about 15 minutes to process the image shown on a 3GHz machine with $k = 1$.

In order to make a real computer vision application, one needs to further study alternative symmetry maps (there are many variations in the literature) and one needs to further study the parameters of the cost functions to extract the optimal tree. The focus here is only to illustrate that this algorithm can be further developed.

5. Conclusion

We have addressed the problem of extraction of objects in images as tree structures. Tree structures are often related to the symmetry axis representation of shapes, which captures their local symmetries. The applications of this idea to computer vision includes the detection of articulated objects such as people as well as other structured objects like arteries in human bodies, roads, circuit board patterns, etc.

Previously, many authors including us have introduced graph algorithms to detect in a principal manner (i) open contours in images in $o(n^2)$ (n is the number of pixels in the image) (ii) closed contours in images in $O(n^3)$, (iii) tree structures from (closed) contours of size L pixels, in $o(L^2)$.

We here have shown that the problem of finding tree structures in images in a principal manner is comparable to an NP-hard problem. We show that by mapping the tree detection problem to a variant of the Steiner Tree problem.

Despite this result, we introduced an approximate polynomial-time algorithms to the Steiner Tree problem. In particular, we have applied a fast implementation of the Goemans-Williamson approximate algorithm to the problem of finding a tree representation after an image is transformed by a local symmetry mapping. We illustrate the idea and applicability of the approximate method with some simple examples.

Acknowledgments This work was partially supported by NSF ITR grant 25-74100-F5366 (Ishikawa and Geiger) and NSF grants CCR0105678 and CCF0515127 (Cole).

References

[1] A. Agrawal, P. Klein, R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM Journal on Computing*, **24**(3):440–456, 1995.

[2] J. August, K. Siddiqi, and S.W. Zucker. Contour fragment grouping and shared, simple occluders. *Computer Vision and Image Understanding* **76**(2):146–162, 1999.

[3] M. Barzohar and D. B. Cooper. Automatic Finding of Main Roads in Aerial Images by Using Geometric-Stochastic Models and Estimation. *IEEE Transac-*

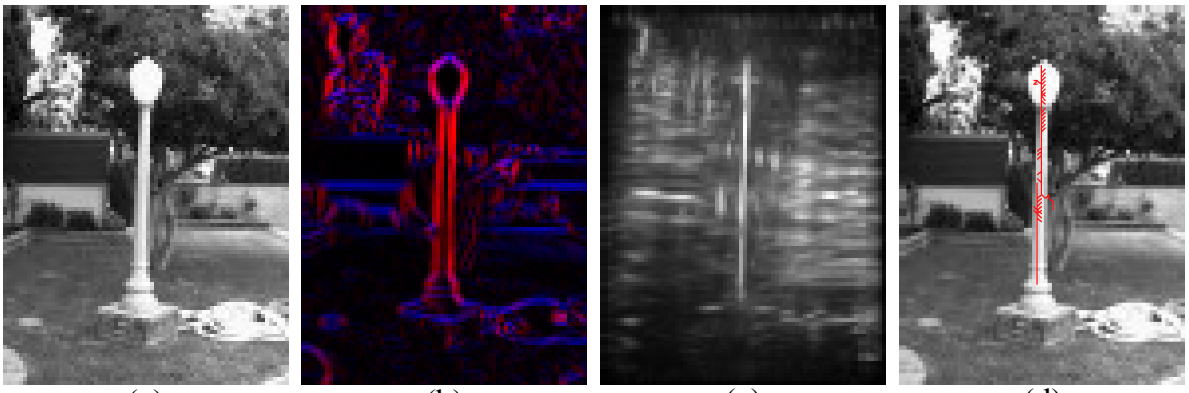


Figure 3. (a) Image example. (b) The oriented edges. We show here the gradient vector colored by the orientation. (c) The application of the symmetry transform to the image. We show the values of $S(p) = \max_\theta S(p, \theta)$, which illustrates the strength of the symmetry at each pixel p. (d) Finally the extraction of the tree structure from images. Note that objects are extracted without any a priori object representation.

tions on Pattern Analysis and Machine Intelligence **18**(7):707–721, 1996.

[4] H. Blum, Biological Shape and Visual Science. *Journal of Theoretical Biology*, **38**:205–287, 1973.

[5] C.A. Burbeck, S.M. Pizer. Object representation by cores: Identifying and representing primitive spatial regions *Vision Research*, 1995.

[6] R. Cole, R. Hariharan, M. Lewenstein, and E. Porat. A Faster Implementation of the Goemans-Williamson Clustering Algorithm. In *Proc. of the 12th annual ACM-SIAM symposium on Discrete algorithms*, pp. 17–25, Washington, D.C., 2001.

[7] I. J. Cox, S. B. Rao, and Y. Zhong, *Ratio regions: a technique for image segmentation*, Proc. Int'l Conf. Patt. Rec., vol. 2, 1996, pp. 557–564.

[8] J. Elder and S. W. Zucker, *Computing contour closure*, Proc. Euro. Conf. Comp. Vis., June 1996, pp. 399–412.

[9] D. Geiger, A. Gupta, L.A. Costa, and J. Vlontzos. Dynamic programming for detecting, tracking and matching elastic contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17**(3):294–302 1995.

[10] M. Goemans, D. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, **24**(2):296–317, 1995.

[11] I. Jermyn and H. Ishikawa. Globally Optimal Regions and Boundaries as Minimum Ratio Cycles. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **23**(10):1075–1088, 2001.

[12] M. Kass, A. Witkin, D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, **1**(4):321–331, 1987.

[13] T.-L. Liu, D. Geiger, and R. V. Kohn. Representation and self-similarity of shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **25**(1):86–99, 2003.

[14] T-L. Liu, D. Geiger, and A. Yuille. Segmenting by Seeking the Symmetry Axis. In *Proc. of Intl. Conf. on Pattern Recognition*. pp.994–998, Sidney, Australia, 1998.

[15] U. Montanari. On the Optimal Detection of Curves in Noisy Pictures. *Communication of the ACM*, **14**(5):335–345, 1971.

[16] M . Pelillo, K. Siddiqi,and S. W. Zucker. Matching Hierarchical Structures Using Association Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **21**(11):1105–1120, 1999.

[17] D. Reisfeld, H. Wolfson, and Y. Yeshurun. Detection of Interest Points Using Symmetry. In *Proc. Intl. Conf. in Computer Vision*, pp.62–65, Osaka, Japan, 1990.

[18] H. Rom and G. Medioni. Hierarchical Decomposition and Axial Shape Description. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **15**(10):973–981, 1993.

[19] T.B. Sebastian, P.N. Klein, and B.B. Kimia. Recognition of shapes by editing shock graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **26**(5):550–571, 2004.

[20] A. Shaashua and S. Ullman. Structural saliency: The detection of globally salient structures using a locally connected network. In *Proc. Intl. Conf. in Computer Vision*, pp. 321-327, Tampa, FL, 1988.

[21] K. Siddiqi and B. B. Kimia. Parts of visual form: Computational aspects. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17**(3):239–251, 1995.

[22] K. Siddiqi, A. Shokoufandeh, S. J. Dickinson, and S.W. Zucker. Shock graphs and shape matching. *Int. Journal of Computer Vision* **35**(1):13–32, 1999.

[23] S.G. Tari, J. Shah, H. Pien. Extraction of shape skeletons from grayscale images. Computer Vision and Image Understanding, 1997.

[24] H. Tek and B. Kimia. Symmetry Maps of Free-Form Curve Segments via Wave Propagation. *International Journal of Computer Vision* **54**(1-3):35–81, 2003.

[25] H. Tek and B. Kimia. Perceptual organization via symmetry map and symmetry transforms. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, Vol. II. pp. 471–477, Fort Collins, CO, 1999.

[26] D. Terzopoulos, A. Witkin, M. Kass. Symmetry-seeking models and 3D object reconstruction. *International Journal of Computer Vision* **1**(3):211–221, 1987.

[27] H. Zabrodsky, S. Peleg, and D. Avnir. Symmetry as a Continuous Feature. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17**(12):1154-1166, 1995.

[28] S.C. Zhu and A.Yuille, FORMS: a Flexible Object Recognition and Modeling System. *International Journal of Computer Vision*, **20**(3):187–212, 1996.