Finding Trails

Scott Morris Kobus Barnard

Computer Science Department
University of Arizona
{smorris,kobus@cs.arizona.edu}

Abstract

We present a statistical learning approach for finding recreational trails in aerial images. While the problem of recognizing relatively straight and well defined roadways in digital images has been well studied in the literature, the more difficult problem of extracting trails has received no attention. However, trails and rough roads are less likely to be adequately mapped, and change more rapidly over time. Automated tools for finding trails will be useful to cartographers, recreational users and governments. In addition, the methods developed here are applicable to the more general problem of finding linear structure.

Our approach combines local estimates for image pixel trail probabilities with the global constraint that such pixels must link together to form a path. For the local part, we present results using three classification techniques. To construct a global solution (a trail) from these probabilities, we propose a global cost function that includes both global probability and path length. We show that the addition of a length term significantly improves trail finding ability. However, computing the optimal trail becomes intractable as known dynamic programming methods do not apply. Thus we describe a new splitting heuristic based on Dijkstra's algorithm. We then further improve upon the results with a trail sampling scheme.

We test our approach on 500 challenging images along the 2500 mile continental divide mountain bike trail, where assumptions prevalent in the road literature are violated.

1. Introduction

There is a growing need for accurate digital representations of recreational trails and 4x4 roads. Trail maps are not even available for many areas. Further, as trails are constructed, closed or rerouted, maps quickly become out of date. Digital representations are becoming more desirable as technologies such as GPS navigation become more widespread. Being able to monitor trails automatically will support natural resource management, directly, and through research into recreation simulation modeling [1].

In this paper we develop a semi-automatic method for extracting trails from aerial and satellite images. We assume that the two end points of the trail are given. Our task is to find the most likely trail that connects the two points.

To develop and validate our system we exploit the large amount of training and testing data available through GPS technology. This data is collected automatically as people recreate and is becoming increasingly plentiful. However, for our purpose, raw GPS data has a non-negligible amount of error, and thus we automatically lock the GPS data onto the nearby trail using the GPS-snakes algorithm [2].

Thus we can easily acquire a large quantity of aerial image data. We compute relatively simple feature vectors and then train systems to estimate the likelihood that an observed image patch is on a trail. We experimented with three ways to do this: a naïve Bayesian classifier, a support vector machine with soft output, and a multi-modal mixture model. All methods give roughly comparable performance for the raw classification task (trail versus not-trail) when tested on held out data.

Even with high quality training data, trails cannot be found by local methods alone. Trail image data is too varied and too noisy. For example, the trail in Figure 1 is completely obscured by bushes in places. In other cases, trails are mimicked by washes, animal tracks, or random alignment of terrain features. Trails can also be indistinguishable from the background on occasion.

Clearly, what further distinguishes trail pixels from non-trail ones is that the trail pixels can be linked up to go somewhere. In other words, we need to integrate the

global condition of a path with the results from our classifiers. We do this by supposing that the per-pixel probabilities are an estimate of the amount of energy a traveler would have to expend to cross that pixel. We then wish to minimize the total energy along the path. This is attractive because it can be done efficiently, provided we blindly ignore the fact that we are comparing trails of different lengths. In practice this approach has some merit, both in the literature and on our test set. But we have found that taking care to control the length of the path significantly improves performance.

Adding length information is not trivial. We were able to formulate a method similar to Floyd's minimum cost algorithm that can handle a length term, but unfortunately this approach has complexity $O(N^3)$ in time and $O(N^2)$ in space, where N is the number of pixels. Since this is computationally prohibitive, we instead developed a splitting heuristic based on Dijkstra's algorithm. This heuristic works by piecing together portions of shortest paths using the length based global cost function. We then further improve on the heuristic method by randomly sampling trails, since the heuristic is constrained to piecing together shortest paths.

Our results on 500 trail images suggest that adding length information to the energy formulation significantly improves the ability to extract trails over naively using the minimum cost formulation.

2. Previous Work

We are not aware of previous work focused on finding recreational trails in remote sensed images. However, the problem of finding roads from such data has been well studied, with a variety of approaches surveyed in [3]. We briefly review some of what has been learned from work on roads below. However, we emphasize that trails are more difficult than roads since they are narrower, less predictable, less uniform, more often obscured, and more often indistinguishable from the background. This means that the handling of the global constraint is more critical—simply (virtually) following the trail will fail much more often than following a well maintained road. Features from trails are also less reliable, leading to our emphasis on a probabilistic treatment and learning the relationship between features and category probabilities from data.

Early work on extracting roads focused on local methods, using image processing techniques on local pixel neighborhoods. To determine if pixels are "road" or "background" operators such as edge detectors, ridge finders, crest detectors, morphological operators and even road-specific operators have been tried [4].

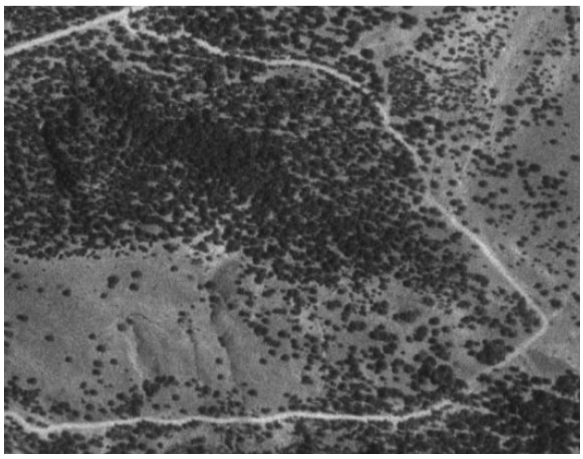


Figure 1: Example USGS aerial photograph, from the Santa Rita mountains

Road tracking [4] extends the local information in plausible directions. However, to keep the search space reasonable, the assumption that roads generally change direction slowly is made—a very reasonable simplification in the road domain, but clearly not prudent in the case of trails.

Our approach is closer to that by Fischler et al [5], which uses a dynamic programming optimization over the pixel lattice. The cost function is based on weights determined by local operators. To extend this work to trail data would require non-trivial extensions. First, it is not clear how to build reliable operators for the case of trails—hence we learn them from data. Second, because even learned trail classifiers can be unreliable, the standard dynamic programming optimization can result in trails that are too long, relative to their total cost. Since trails turn direction more often than roads this especially applies. We instead propose optimizing a cost function with an added penalty term that is function of length, and present heuristics for computing it.

3. Data Collection

In order to build a statistical model of trails we require training data. The most obvious approach is to have a human operator choose example trail points by inspection of an aerial photograph. This approach suffers not only from being manual but it also exhibits a typical problem in many machine learning situations: a lack of large, quality datasets.

Instead, we propose using an already existing and growing data source: GPS track logs. Track logs are sequences of precise locations that a GPS receiver automatically records as a trail user travels. While GPS data is still in some sense manually collected, it is relatively easy to collect, and is being collected for

other purposes, even as a part of everyday recreation. Inexpensive consumer level GPS receivers are increasingly being used to record outdoor trips. Many web sites offer GPS track logs for download, and centralized, user submitted databases are on the horizon.

3.1. Aerial Image data

We use USGS DOQ imagery (Figure 1) because a public domain internet source is available for nearly the entire United States. Microsoft's Terraserver [6] efficiently serves these images, which we are able to download, on-the-fly, using TopoFusion [7] software. Using TopoFusion we are able to seamlessly process GPS data from around the country. As points are processed, aerial images are downloaded via the internet and cached on local disk to speed up further iterations. This greatly simplified our research effort and improves the applicability of the techniques developed. Our approach can be used to extract trails for any location in the United States. We use the highest level detail imagery available, which is at a resolution of one meter per pixel.

3.2. Pre-processing using GPS-snakes

Although using GPS provides an abundance of data, the quality of the data suffers compared to human extracted data. There are two sources of error: the GPS system itself and photo calibration error. The result is that GPS data points sometimes do not lie exactly on corresponding trails in aerial images. These problems can be partially addressed by using the GPS-snakes algorithm [2]. GPS-snakes can be used to pre-process the GPS data, placing it on trails in the neighborhood of the data. The GPS-snakes method draws inspiration from the active contour (snakes) model developed by Kass et al [8], but it exploits properties of GPS data and trail image appearance to correct GPS tracks. In Figure 2, original GPS data is shown along with the GPS-snakes corrected output. Table 1 shows the positive effect of using the GPS-snakes to clean up the data before training.

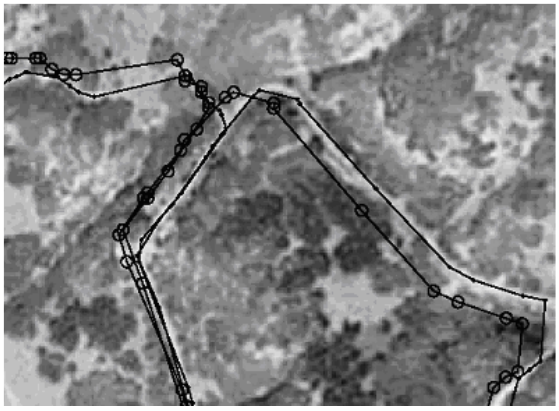


Figure 2: GPS track log corrected using GPS-snakes. Empty circles: original data. Smaller filled dots: corrected data.

3.3. Negative (non-trail) examples

To learn the difference between trails and the background we also need non-trail example points. To generate non-trail points we simply sample random points in the area covered by the trail points. This is based on the observation that most of the world is not a trail. Thus we select non-trail points which are the mean of eight randomly selected trail points. Visual inspection confirmed that this process produced good selections of negative examples. In all experiments we used equal numbers of positive and negative examples.

4. Statistical Learning

Naturally we begin by computing feature vectors for image regions surrounding each pixel. We use feature vectors from the training data to build a classifier that estimates the probability that a pixel is on a trail. To find trails in a new image, we compute analogous feature vectors, and use them as input to the classifier to estimate the trail probabilities for the second part of the algorithm which finds the path through the pixels. While the focus of this work is on the integration of low and high level information, we experimented with three different systems of probabilistic classification.

4.1. Computing Feature Vectors

To capture what trails look like in aerial images we use a characterization of texture in the form of responses to oriented Gaussian filter kernels [9-11]. These function as oriented edge detectors, capturing edges in the direction perpendicular to the direction the trail is traveling. The kernels are centered at the data point and the response at each orientation is used as a component of the feature vector. The final component of the feature vector is a simple grayscale histogram in a small area surrounding the point.

We settled on the following parameter values which gave good performance on preliminary experiments: 12 filter orientations, at a single scale (sigma is 2.5 pixels). The histogram was computed in a 9x9 window around the pixel.

4.2. Naïve Bayes

A naïve Bayes classifier operates under the 'naïve' assumption that all of the feature vector components are independent of each other. This is not the case with our vectors, but even when this assumption is violated, naïve Bayes often performs well [12].

We generate two sets of histograms, one for trail and one for not trail. Among each set there is a histogram for each component of the feature vector. With the histograms computed, the likelihood of a test feature vector being trail or not trail is computed simply by counting the proportion of training data that fell into the bins corresponding to the test vector's components. If the likelihood produced by comparison with the trail histograms is greater than the likelihood from the not trail histograms, the vector is classified as a trail.

One issue for the Bayes classifier is the choice of the number of histogram bins. To determine this number we plotted the performance on some held out data of the classifier with varying bin values. We chose 32 bins, which was maximal for the Tucson Mountain Park dataset. The plot was relatively flat, indicating that performance is insensitive to the number of bins.

4.3. Support Vector Machine

We presented the feature vectors to SVM^{light} [13], a C implementation of Support Vector Machines [14]. The SVM attempts to find a multidimensional cutting plane that separates the positive (trail) and negative (non-trail) examples. We experimented with the four available kernel options: linear, polynomial, radial basis and sigmoid. We found that the linear cutting plane consistently outperformed the other kernel options on training and test data. The SVM^{light} software provides soft classification which we normalized to use as a crude estimate of probability.

4.4. Multi-modal mixture model

We also trained a generative multi-modal mixture model [15] [16] to classify image points. Here we assume that an image pixel and its label ("trail" or "not_trail") are concurrently generated as follows. First, a hidden factor, or node, is chosen according to a prior distribution. Then the visual features for that pixel and its label are generated conditionally independent given the node. The label is generated according to a simple frequency distribution, and since there are only two labels, this reduces to a single number, namely the probability of "trail". The image features are generated according to a Gaussian distribution with a diagonal covariance matrix. The model for the joint distribution for the "trail" label, t, and the feature vector, **v**, is given by:

$$P(t,\mathbf{v}) = \sum_{n}^{N} P(t \mid n) P(\mathbf{v} \mid n) P(n) \quad (1)$$

where n indexes over nodes, $P(t \mid n)$ is the probability of trail given the node, $P(\mathbf{v} \mid n)$ is a Gaussian distribution for that node, and $P(n)$ is the node prior. Model parameters are learned from the training data using the expectation maximization algorithm [17]. We verified that the performance of the model is relatively robust to a wide range for the number of nodes, N. For the results in this paper we used 200 nodes.

5. Trail Extraction

In the final step we construct connected pixel sequences that represent probable trails, given a classified probability image. The process is semi-automatic, meaning we assume that we are given a start and end point. The algorithm then connects the points with an optimal path based on the probability image.

5.1. Trail objective function

We consider P(not_trail) for a given pixel to be an estimate of the energy required to cross that pixel. The idea being that a pixel that is high in trail probability (and thus low in non-trail probability) will be easy to cross. We can then compute the global minimum for the objective function:

$$f(path) = \sum_{1}^{N} P(not\_trail) \quad (2)$$

If we let N vary we can find the above minimum efficiently with a shortest path algorithm such as Dijkstra's [18], and doing so leads to a baseline algorithm which we test below. However, this favors longer paths since we are, in effect, maximizing P(trail) along the path. Figure 3 is an example of an image where minimizing (2) resulted in a path that is too long.

There are situations when a bias for short paths is desirable. For example, in areas where there is a genuine lack of information (e.g. trees obstructing the trail) the shortest path is a reasonable guess. There is also a notion that trails, though they do not follow the shortest Euclidean distance between points, are still "going somewhere," rather than roaming around aimlessly. Therefore, we introduce an empirically determined crude length bias for the cost function:

$$P(path) \propto length^2 / d(v1, v2)^2 \quad (3)$$

where $d(v1, v2)$ is the Euclidean distance between the start and end nodes (constant for a given trail image),

and *length* is the total distance of the path which varies among hypotheses. (2) then becomes:

$$f(path) = \frac{length^2}{d(v1, v2)^2} \sum_{1}^{N} P(not\_trail) \quad (4)$$

Computationally, adding a length term to the objective function is difficult. Known fast minimum cost methods do not apply since dynamic edge weights are introduced into the pixel lattice.

We have formulated a dynamic programming algorithm to compute the minimum path, with length prior, that runs in $O(N^3)$ time, where N is the number of pixels in the image. It is based on enumerating optimal paths of a particular length (up to a reasonable cut-off) in turn. Besides being computationally expensive, the algorithm also requires $O(N^2)$ space, making it wholly impractical for anything but very small images. We instead develop alternative methods.

5.2. Recursive Splitting Heuristic

Since finding the global minimum of (4) is computationally prohibitive, we have devised a more feasible heuristic method based on the following observation. The optimal path is likely to be composed of portions of nearly optimal sum paths. That is, for very short distances, the minimal cost and length adjusted cost paths are the same.

Our heuristic proceeds as follows. Let *v1* and *v2* be the start and end nodes of the path. We first compute the shortest path between *v1* and all other nodes using Dijkstra's algorithm. The cost function used is the sum of P(not_trail) as in (2). We then do the same for *v2* to all nodes. Now we find the intermediate node, v_{int}, which minimizes (4) along the path ($v1 \rightarrow v_{int} \rightarrow v2$), where the paths used are the optimal P(not_trail) paths computed by Dijkstra's algorithm. The resulting path has the lowest objective value, subject to the constraint that only the shortest paths from nodes *v1* and *v2* through an intermediary point can be followed. Under this greedy assumption we then proceed by recursing on the two halves of the path. Shortest paths and intermediate nodes are computed just as before. The recursion runs to a specified depth (we used a recursive depth of two in this work).

We also cannot allow the resulting path to intersect itself. Checking for intersection is the slowest part of the method, taking $O(N^2)$ time, worst case. Dijkstra's itself runs very quickly when implemented with a Fibonacci heap and is run a constant number of times.

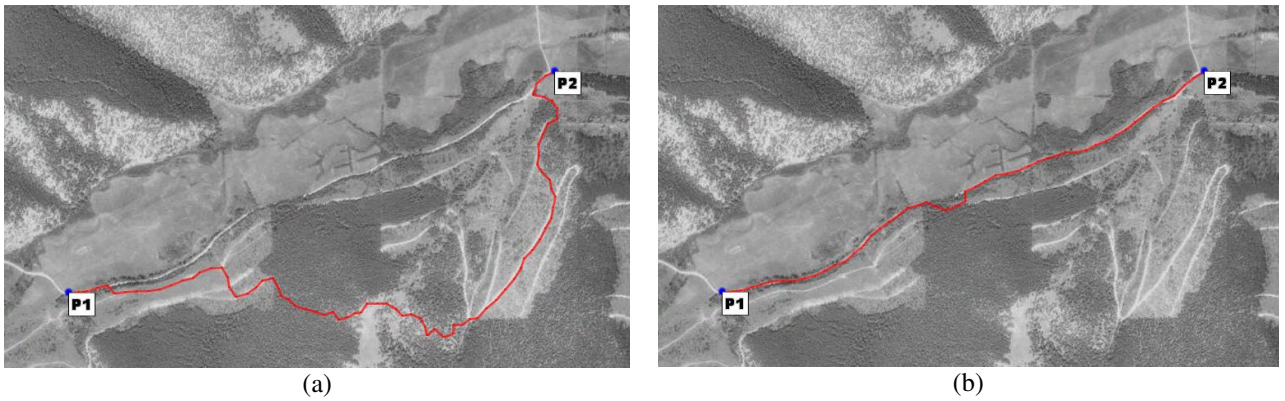
Figure 3 provides an example where the recursive splitting heuristic correctly extracted the trail where naïve minimum cost failed.

5.3. Trail Sampling

The splitting heuristic introduced in the previous section suffers from the limitation that it can only choose minimal cost paths between chosen intermediate points. An example of this is shown in Figure 4(a). We propose to further lower the objective function (4) by sampling random trails in the image.

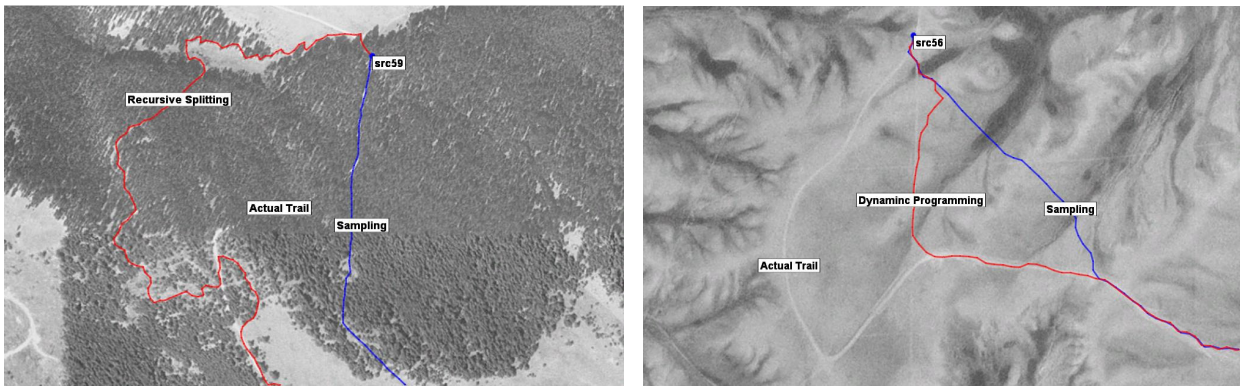
We seed the sampler with the lowest cost trail as output by the splitting heuristic procedure. The current sample is altered by the following proposals:

(a) Trail lengthening. A random point on the trail is chosen. N points are chosen according to a Gaussian distribution centered at the chosen trail point. The point with the maximal value of P(trail) among these N points is chosen to



(a) (b)

Figure 3: A image from the test set from the Great Divide Mountain Bike Route. Seed points P1 and P2 were taken from the GPS track of the trail. Resulting extracted trails are shown, comparing the difference between minimum cost and length adjusted paths. (a) shows the effect of the trail using minimum cost; since portions of the trail are obscured, the optimal path follows longer side trails (b) shows the correct trail extracted using the recursive splitting heuristic. The Hausdorff distance between (b) and the ground truth GPS track is 49.5. This is just less than the threshold for correctness. The only portion "off" trail can be seen in the middle portion of (b), where the trail is obscured.



(a) (b)

Figure 4**:** Trail Finding comparisons. In (a) the recursive splitting method suffers from the constraint that it can only use strict shortest paths to construct global paths. In this case dynamic programming without a length prior causes the trail finder to follow spurious noise. The sampling technique extracts more of the trail. (b) shows an example of a poorly visible trail. All methods fail to extract the correct trail, with the sampling technique causing short cutting due to weak trail evidence in the image.

"pull" the trail to. The trail is cut at half the "pulling" distance away from the chosen point, and connected with straight lines.

(b) Trail shortening. A random point on the trail is chosen. A normally distributed distance to shorten by is chosen. The trail is then walked until this distance is met, and the current trail is replaced by a straight line.

Both proposal types are subject to the constraint that intersecting paths are not allowed. If the proposal introduces an intersection, it is rejected.

Each type of proposal is equally likely to be chosen. At each iteration, the objective function (4) is evaluated for both the current sample, x^t, and the proposed sample, x'. Since we want to explore the trail space, we allow acceptance of worse proposals according to the ratio:

$$\alpha < \frac{f(x^t)}{f(x')} \qquad (5)$$

Where α is drawn from U(0,1). Though similar to the Metropolis Hastings algorithm [19] [20], our objective function (4) is not a probability distribution, so we are not using Metropolis Hastings as such. The ratio (5) is reversed since we seek the minimum of our objective function, rather than the maximum probability. The sampler keeps track of the minimum trail as it explores the trail space.

6. Results

Pixel classification. We present results from two GPS datasets, one from the Tucson Mountains and another from the Santa Rita range. The Tucson Mountains are lower Sonoran desert terrain while the Santa Ritas are higher elevation chaparral. Negative training points for each set were generated using the heuristic proposed in §3.3. Each dataset contains roughly 4000 trail points (as well as 4000 non- trail points).

Each of the three classification models were trained on 90% of the data, while a randomly chosen 10% were held as a test set. Table 1 presents the accuracy rates on the test sets for each of the three methods. There is some variation between the methods, but all seem capable of generating the correct answer, trail or not trail, roughly 75% of the time. Table 1 shows that the effect of pre-processing the data using GPS-snakes is significant. In some cases the performance gain is over 10%.

Due to similar performance of the three pixel classification methods, we chose Naïve Bayes since it has the shortest runtime.

Finding trails. We test our techniques using a large and challenging dataset. The Great Divide Mountain Bike Route (GDMBR) is a bicycle path that covers 2500 miles along the continental divide of North America, from Canada to Mexico. Using GPS data collected along the GDMBR, we train and test a general purpose trail finder.

We trained the Naïve Bayes classifier using GPS points collected on the southern half of the GDMBR. This corresponds to the states of New Mexico and Colorado.

The northern half (Montana, Idaho and Wyoming) was used for testing. 500 trail sections were chosen, at random, from the GPS track. Each trail is associated with a 2000x2000 aerial image, whose pixels were then classified with Naïve Bayes. The average trail length is 3000 meters, with a standard deviation of 820 meters. The average straight line distance between start and end

Table 1: Accuracy on held-out data for three classification methods. Using GPS-snakes to pre-process improves the classification results significantly.

	Data set 1 – Tucson Mountain Park	Data set 1 – Pre-processed Snakes	Data set 2 – Santa Ritas	Data set 2 – Pre-processed Snakes
Naive Bayes	73.9%	76.1%	73.1%	78.5%
Support Vector Machines	74.6%	83.0%	71.3%	81.2%
Multi Modal Mixture Model	75.2%	79.2%	71.4%	81.5%

points is 1920 meters, suggesting that the trail segments are far from straight.

Since the trail follows and crosses the continental divide it often traverses steep mountains, which leads to frequent turns and switchbacks. Elevations range from 2500 ft to 12,000 ft, introducing a wide variety of vegetation and soil types. Figure 3 shows one of the test images with seed points P1 and P2.

Table 2 summarizes the results on the test images. To measure performance we use the Hausdorff distance between the GPS data and the extracted trail. Intuitively this means we are measuring the maximum distance the extracted trail strays from the truth. The values in the table are the average Hausdorff distances for each run of the test set.

The results show that a significant improvement can be gained by penalizing trails by length. The splitting heuristic shows considerable (9.5%) improvement and sampling trails seeded with that answer gives further improvement (17.7%).

At first glance, an average Hausdorff distance of over 100 meters may seem large. But we argue that these are solid results. Given that these trails are, on average, 3000 meters long, only straying from the trail by at most 100 meters means the majority of the trail was correctly extracted.

We define a Hausdorff threshold of 50 meters to determine whether a particular trail is "correct." If the most a solution strays from the ground truth is 50 meters, it has correctly identified the trail (see Figure 3 for an example trail at a Hausdorff of 49.5). Using this measure, the sampling technique gets 60% of the images correct. This is impressive given the difficulty and high variability of the dataset. We have observed a significant number of trail images for which the correct answer is highly unlikely to be extracted by any process. This is either because the trail is not visible to the human eye or because multiple trails are visible in the image, and the GDMBR followed a less likely (even to human eyes) path. Figure 4(b) gives an example where all techniques failed to extract the correct trail due to weak trail evidence.

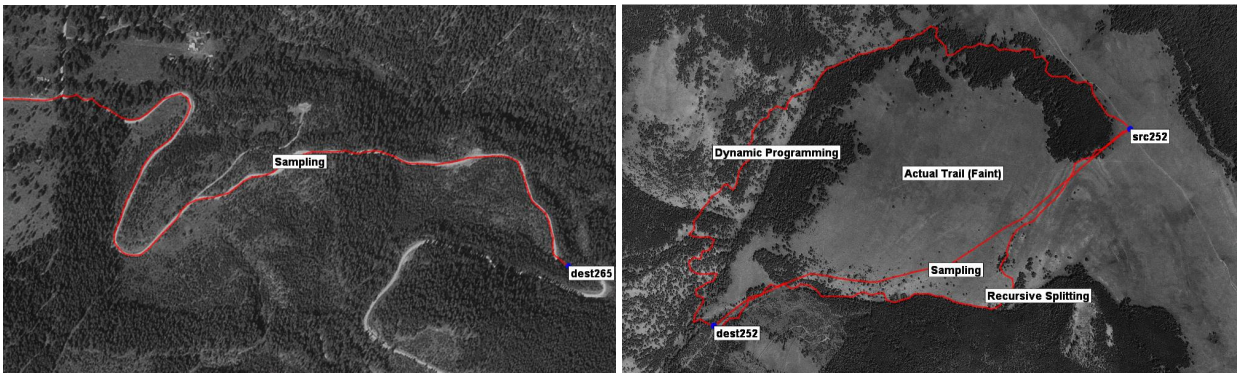
7. Discussion

We have presented results showing that our approach is capable of extracting trails in a wide variety of terrain. Key points include using GPS tracks to obtain significant quantities of training data, a snakes based method to improve the training data, statistical models to estimate the probability that pixels are associated with trails, and a global constraint that trail pixels must link together to form a path.

We have demonstrated that a naïve implementation of minimal cost across the pixel lattice does not correctly deal with the length of the path, and that formulating the problem with a length prior significantly improves performance. Though a global

Table 2: Finding trails results. The table lists the average Hausdorff distance, in meters, between the true trail (GPS data) and the extracted trail for 500 random, 2000x2000 images along the Great Divide Mountain Bike Route. Significant improvement is shown when the length penalty of the splitting heuristc method is added. Sampling trails using the same objective function gives even further improvement. Error estimates are provided in parentheses.

	Mean Hausdorff Distance	Improvement vs. Baseline
Minimum Cost Dynamic Programming (Baseline technique)	138.7 (9.2)	N/A
Recursive Splitting Heuristic	126.7 (8.1)	9.5%
Sampling	117.8 (7.8)	17.7%



(a) (b)

Figure 5: Further Trail Examples. (a) shows a successfully extracted trail despite occlusion and switchbacks. In (b) a very difficult image is shown. All methods fail to extract the faint trail.

solution that accounts for length is not possible, we have presented a heuristic method, and further improved upon the results by sampling.

The problem underlying our task is quite common. In particular, it is often the case that local features are required for extraction, but are unusable without a global path constraint. One example is the extraction of neuron branching structure from images. Helping neuroscientists do so automatically would increase the scale of the data that they can process. We are pursuing this important alternative application domain.

References

[1] H. R. Gimblett, M. T. Richards, and R. M. Itami, "RBSim: Geographic Simulation of Wilderness Recreation Behavior," *Journal of Forestry*, vol. 99, pp. 36-42, 2001.

[2] S. Morris, "Pathway Extraction using Snakes with GPS Initialization," *Proc. 2nd International Conference on Geographic Information Science*, Boulder, CO, 2002.

[3] M.-F. Auclair-Fortier, D. Ziou, C. Armenakis, and S. Wang, "Survey of Work on Road Extraction in Aerial and Satellite Images," Departement de mathematiques et d'informatique, Universitede Sherbrooke, Technical Report 247, 2000,

[4] D. Geman and B. Jedynak, "An Active Testing Model for Tracking Roads in Satellite Images," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 18, pp. 1-14, 1996.

[5] M. Fischler, J. Tenebaum, and H. Wolf, "Detection of roads and linear structures in low-resolution aerial imagery using a multisource knowlede integration technique," *Computer Graphics and Image Processing*, vol. 15, pp. 201-223, 1981.

[6] T. Barclay, R. Eberl, J. Gray, J. Nordlinger, G. Raghavendran, D. Slutz, G. Smith, and P. Smoot, "The Microsoft TerraServer," MSR-TR-98-17, 1998,

[7] S. Morris and A. Morris, "TopoFusion GPS Mapping," Internet: www.topofusion.com. Dec. 10, 2007.

[8] A. Witkin, M. Kass, and D. Terzopoulos, "Snakes: Active Contour Models," *Proc. IEEE International Conference on Computer Vision*, pp. 259-268, 1987.

[9] J. Malik and P. Perona, "A computational model of texture segmentation," *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 326-332, 1989.

[10] J. Malik and P. Perona, "Preattentive texture discrimination with early visual mechanisms," *J. Opt. Soc. America. A*, vol. 7, pp. 923-932, 1990.

[11] J. Shi and J. Malik., "Normalized Cuts and Image Segmentation," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 22, pp. 888-905, 2000.

[12] P. Langley, W. Iba, and K. Thompson, "An analysis of Bayesian classifiers," *Proc. Tenth National Conference on Artificial Intelligence*, San Jose, CA, pp. 223-228, 1992.

[13] T. Joachims, *Learning to Classify Text Using Support Vector Machines*: Kluwer, 2002.

[14] V. N. Vapnik, *The Nature of Statistical Learning Theory*: Springer, 1995.

[15] K. Barnard and D. Forsyth, "Learning the Semantics of Words and Pictures," *Proc. International Conference on Computer Vision*, pp. II:408-415, 2001.

[16] K. Barnard, P. Duygulu, N. d. Freitas, D. Forsyth, D. Blei, and M. I. Jordan, "Matching Words and Pictures," *Journal of Machine Learning Research*, vol. 3, pp. 1107-1135, 2003.

[17] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, pp. 1-38, 1977.

[18] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*: McGraw-Hill, 1990.

[19] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equations of state calculations by fast computing machines," *Journal of Chemical Physics*, vol. 21, pp. 1087-1092, 1953.

[20] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, pp. 97-109, 1970.