Static Data Association with a Terrain-Based Prior Density

Allen L. Barker
Donald E. Brown
Worthy N. Martin

IPC-TR-94-001
Jan. 15, 1994
(minor revisions Jan. 28, 1998)

Institute for Parallel Computation
School of Engineering and Applied Science
University of Virginia
Charlottesville, VA 22901

This research was sponsored in part by the Jet Propulsion Laboratory under grant number 95772.

Abstract

We consider the problem of estimating the states of a static set of targets given a collection of densities, each representing the state of a single target. We assume there is no a-priori knowledge of which of the given densities represent common targets, but that a prior density for the target locations is available. For a two-dimensional location estimation problem we construct a prior density model based on known features of the terrain. For a simple Gaussian association-estimation algorithm using a prior density we consider when the prior is most effective in data association, or correlation, and when it is most effective in state estimation. We present some simulation results and discuss some issues involved in measuring algorithm performance and in the algorithm implementation. We briefly discuss extensions to higher dimensional state spaces and non-static models.

1 Introduction

In this paper we consider the static data association problem as a special case of the more general problem of tracking multiple targets without a-priori identification. The multitarget tracking problem has been extensively researched and has a wide variety of applications from air traffic control to robotics and image processing. We refer the reader to [BSF88, BB90, MCTW86] and the references contained therein.

We use the term static to refer to the fact that the target states are constant. This is nonetheless an important special case which, for example, can often adequately model seldom-moving or slow-moving targets. The static case also arises in the solution of non-static cases, when a motion model is used to project all densities to a common time instant.

We consider how a prior density for the target locations can be used to aid in data association and target state estimation. The equations and algorithm we present do not assume any particular state dimensionality, but the specific problem we look at is a target location estimation problem in a two-dimensional region. We use known terrain features of the region to construct the prior density for target locations. While the volume of research on target tracking is large, few authors have focused on using terrain information to construct a prior density. A notable exception is [RB79], which we mention further in Section 6.

One may also view the problem as a clustering problem with a prior density. Like target tracking, the clustering problem has received extensive research; see e.g. [Eve93] and the references therein. We work from a Bayesian framework to facilitate generalization to more complex Bayesian models, though similar or identical results can be obtained by a variety of methods.

In Section 2 we describe our problem more formally. Section 3 discusses creating a prior density from terrain information. In Section 4 we present a simple association-estimation algorithm. In Section 5 we look at some one-dimensional examples of association and estimation with the algorithm. In Section 6 we describe some simulations we have carried out, and discuss some related issues. In Section 7 we discuss extensions to non-static models, and in Section 8 we list some conclusions. Some details of the simulation implementation are described in Appendix A, Appendix B contains a discussion of statistical comparisons of algorithm performance, and Appendix C contains code for generating some of the graphs in the paper.

2 Problem Description

The problem we consider is as follows. There exists a set of random n-vectors, or targets, $\tilde{X} = \{\tilde{x}_1, ..., \tilde{x}_{\tilde{N}_T}\}$. Notationally, we write the tilde symbol above random quantities, and take the variable name without the tilde to refer to a member of the random variable's range. Thus we write the set of targets as X when it is considered non-random. The targets are characterized by their states $\tilde{x}_i \in \tilde{X}$, and are assumed to be independent and identically distributed (i.i.d.), continuous random vectors. We write the number of targets in $\tilde{X}$ as $\tilde{N}_T$. We seek to produce an estimate of the set $\tilde{X}$ based on a set of sensor measurements. At each discrete time[1] k we have the set $Z(k) = \{(z(1), t_1), \ldots, (z(k), t_k)\}$ of observations, or sensor reports. At each time k a new sensor report is produced by selecting a target uniformly from X (with replacement) and sampling from a Gaussian distribution with the target state as the mean and a known covariance matrix $V(k)$.[2] Physically these sensor reports correspond to measurements made by sensors with known Gaussian error distributions. Since we assume a known, Gaussian, sensor model these observations can be taken to provide not point estimates, but densities for the states of the selected targets.

We are given the prior density $p(N_T)$ for the number of targets, which is assumed Poisson with known mean μ_{N_T}. We generally use the symbol p to denote the density function associated with any random variable, and assume these densities to exist. Thus $p(\tilde{y} = y)$ is the density function of random variable $\tilde{y}$, and we will simply write $p(y)$ when the meaning is clear. Similarly $p(y|x)$ is the conditional density function of $\tilde{y}$ given that $\tilde{x} = x$. In addition to the prior for the number of targets we are given a prior density $p(x) \equiv r(x)$ for the i.i.d. target states. In our case, $p(x)$ is a function of the terrain, and we consider states to be locations only.

We write the number of *detected* targets at time k, i.e., the number of targets which correspond to some sensor report in $Z(k)$, as $\tilde{N}_D(k)$. We wish to produce and dynamically maintain the set

$$\hat{X}(k) = \{\hat{p}(y_1(k)), \ldots, \hat{p}(y_{\hat{N}_D}(k))\} \tag{1}$$

of densities for the elements of a hypothesized random set $\tilde{Y}(k) = \{\tilde{y}_1(k), \ldots, \tilde{y}_{\hat{N}_D}(k)\}$ of target states which "best" estimates the true set of target states X. We

[1]Here k is shorthand for time t_k, i.e., k is a subscript on a real-valued time instance. Time is not really important in the static case, but is included for generality.

[2]We will interchangeably say a Gaussian density is given, rather than a sample and a covariance. In this case we mean the normalized likelihood.

use the hat symbol to to denote estimates; these estimates correspond to actual data structures in a computer implementation. Note that the estimated number of detected targets at time k is $\hat{N}_D(k)$; we omit the time argument when the meaning is clear.

The algorithm we use to compute the estimates $\hat{X}(k)$ is an *association-estimation algorithm.* Each sensor report is associated with exactly one hypothesized target[3] $\tilde{y}_i$, and the estimated density $\hat{p}(y_i)$ for this hypothesized target is a function only of those reports which are associated with it. This is equivalent to first clustering all the reports so each cluster corresponds to a set of reports assumed to be associated with a common target, and then forming independent state estimates from each cluster. By "best" estimate $\hat{X}(k)$ we mean the set of densities formed by the maximum a-posteriori data association given the observed data, followed by the maximum a-posteriori estimate assuming the data association is correct. That is, letting $\tilde{\Lambda}(k)$ be the unknown data association function at time k, choose the data association $\Lambda_{MAP}(k)$ which maximizes $p(\Lambda|Z)$, and then form the estimate $\hat{X}(k)$ which maximizes $p(\tilde{Y}(k) = X \mid \tilde{\Lambda}(k) = \Lambda_{MAP}(k), Z(k))$. Other definitions of "best", such as maximizing $p(\tilde{Y}(k) = \hat{X}|Z(k))$, are possible[4].

We can write the system above more compactly as

$$\begin{aligned} \tilde{x}_i(k+1) &= \tilde{x}_i(k), \quad 1 \le i \le \tilde{N}_T && (2) \\ \tilde{z}(k+1) &= \tilde{x}_{\tilde{j}_{k+1}}(k+1) + \tilde{v}(k+1), && (3) \end{aligned}$$

where the state $\tilde{x}_i$ of each target is a random n-vector with prior density $p(x)$, the measurements $\tilde{z}(k)$ are random n-vectors formed from the selected target's location with zero mean additive, independent Gaussian noise $\tilde{v}(k+1)$ having known covariance matrix $V(k+1)$. The target chosen at time $k+1$ is indexed by $\tilde{j}_{k+1}$, which is discrete uniform in $[1, N_T]$. The elements of the known set $Z(k) = \{(z(1), t_k), \ldots, (z(k), t_k)\}$ of observations are realizations of the random variables $\tilde{z}(k)$, and from this given data we seek to estimate the unknown number of targets $\tilde{N}_T$ and the unknown states $\tilde{x}(k)$ of the targets. Equation (2) is the static target assumption, and equation (3) describes the sensor model. These types of sensor reports are sometimes called "occasional measurements", as opposed to radar-scan type measurements [Rei79]. In our model each measurement corresponds to an actual target so there are no "false alarms" or "clutter" measurements.

[3] Or *track* in the non-static case.

[4] Note that the "=" sign in $p(\tilde{Y} = \hat{X}|Z)$ is *set equality.* Maximizing this expression is still analogous to a point estimate; ideally we would like a more complete characterization of the posterior density.

In terms of conditional densities we can write the system (2) and (3) as

$$\begin{aligned} p(x_i(k+1)|x_i(k)) &= \delta(x_i(k), x_i(k+1)), \quad 1 \leq i \leq N_T \quad (4) \\ p(z(k+1) \mid j(k+1), X, N_T) &= G(x_j(k), V(k+1), z(k+1)) \quad (5) \\ p(j|N_T) &= 1/N_T, \quad (6) \end{aligned}$$

where $\delta(a,x)$ is the Dirac delta function with "spike" at $a = x$, and

$$G(a, A, x) = (2\pi)^{-n/2} \det(A)^{-1/2} e^{(-1/2)(x-a)'A^{-1}(x-a)}$$

is the n-dimensional Gaussian density function with mean vector a and covariance matrix A.

3 Modeling the Prior Density

In order to use knowledge of terrain features to improve association and estimation we first use known terrain features to produce prior density $p(x) \equiv r(x)$ for the location of any individual target. In building an operational system, this would be one of the most difficult tasks. An expert's domain knowledge, combined with past data, would be used to construct the prior. Since we use artificially generated data, though, we are able to create a fairly simple model-based prior density *which is the true prior density for the data.* That is, we have $\hat{p}(x) = p(x)$ so our *estimate* of the prior is identical to the true prior density. Robustness of the prior under modeling errors is another important issue we do not consider, but which would need consideration when the prior is not known exactly.

Recall that the target locations are i.i.d., and that the number of targets is independently Poisson distributed with density $p(N_T)$. We assume that for any location x we know the vector $Features(x)$ which contains the values for a finite number of terrain attributes. In our case

$$Features(x) = \begin{pmatrix} Water(x) \\ Obstacles(x) \\ Slope(x) \\ SurfMaterial(x) \\ Vegetation(x) \\ RoadDist(x) \end{pmatrix}, \quad (7)$$

where the elements are categorical variables except for *RoadDist*, which is the distance to the nearest road.

In order to model the prior density from the *Features* function we assume the functional form

$$p(x) = r(x) = r(Features(x)) \propto \prod_i r_i(Features_i(x)), \tag{8}$$

where $Features_1(x) = Water(x)$, etc. Thus in our model we assume each terrain feature is independent. We assign likelihood values for each of the categorical variable values and take the likelihood for *RoadDist* to be Gaussian with mode a constant. Because of the independence assumption (8) we then multiply these likelihoods together and normalize. For example, the water feature can take on two possible values, defined as $Water(x) = true$ for water covered regions and $Water(x) = false$ otherwise. We decided a priori that targets will never locate at water regions, so we assign $r_1(true) = 0$, and $r_1(false) = 1$. Thus we have defined a likelihood function $r_1(Water(x))$ over the entire terrain region by associating a likelihood value with each possible $Water(x)$ value. Similarly, we decided a priori that the targets we are modeling will tend to locate on the sides of roads. The roads provide access to the location, but the targets then pull to the side of the road and become approximately static. We model this situation by assigning

$$r_6(RoadDist(x)) = (1 - mv)e^{-((RoadDist(x)-rmd)/rmd)^2/2} + mv$$

where rmd is the mode distance from the road, i.e., the most likely distance from a target to a road, and mv is the minimum value the function decays to as the distance from a road increases. Once all the r_i are defined they are multiplied together to form the final likelihood, which can be normalized if desired.

In a real implementation the additional independence assumption of the functional form (8) would probably not be acceptable and combinations of features would need to be considered. When r can be taken to be a function only of the terrain features, though, generalization from past data is greatly simplified since we know the terrain features at each point. If we were given a collection of past data then the actual numerical values to plug into the model could be determined by fitting the parameters of the selected density model to maximize the likelihood of the past data. In our case, the parameters of our model are the likelihood values assigned for the $r_i(Features_i(x))$ functions.

In Figure 1 we show the road, water, and obstacle features for a 14 km by 14 km region in Killeen, Texas. Water and obstacles are black, roads

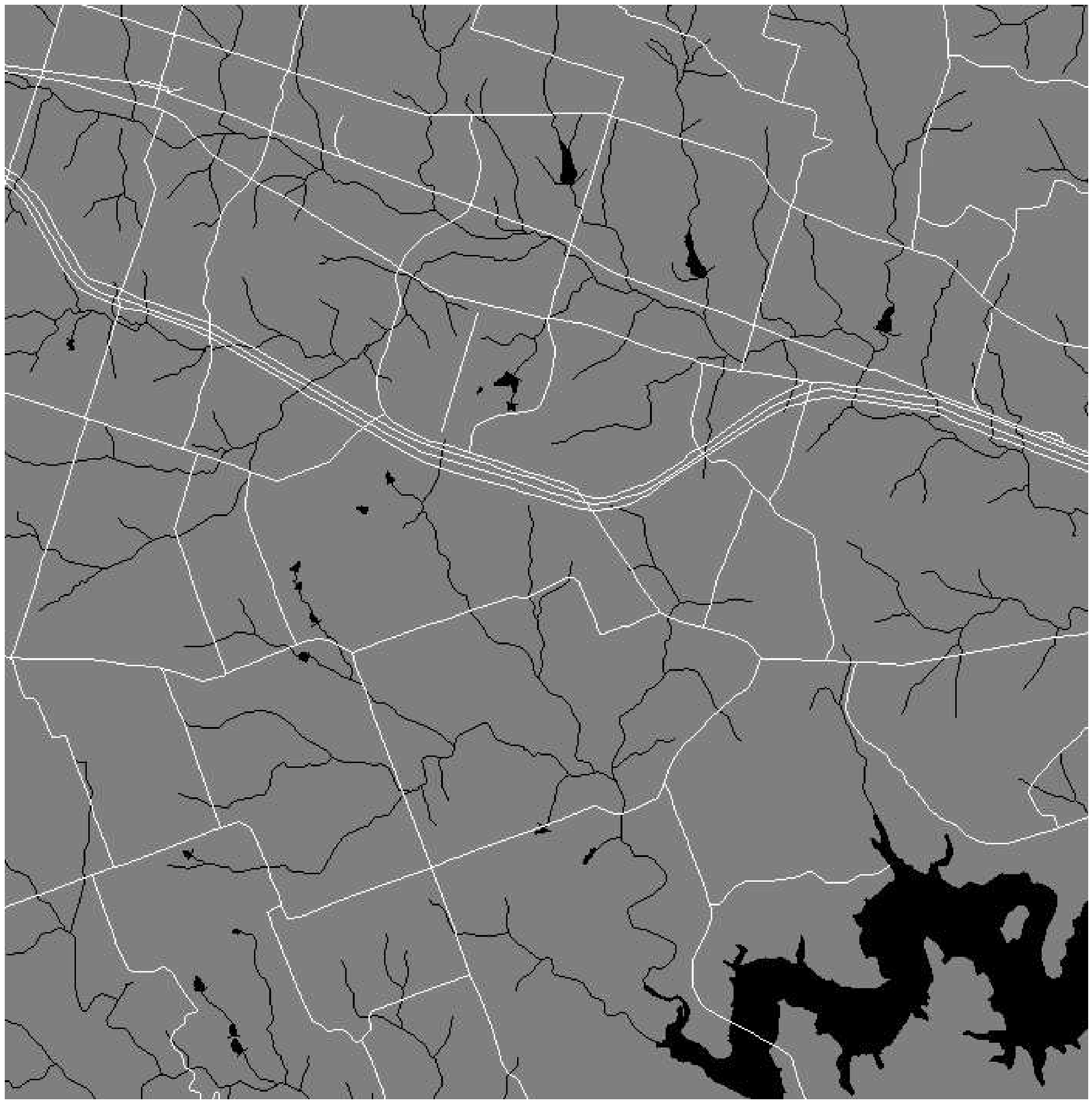
Figure 1: Road, water and obstacle features.

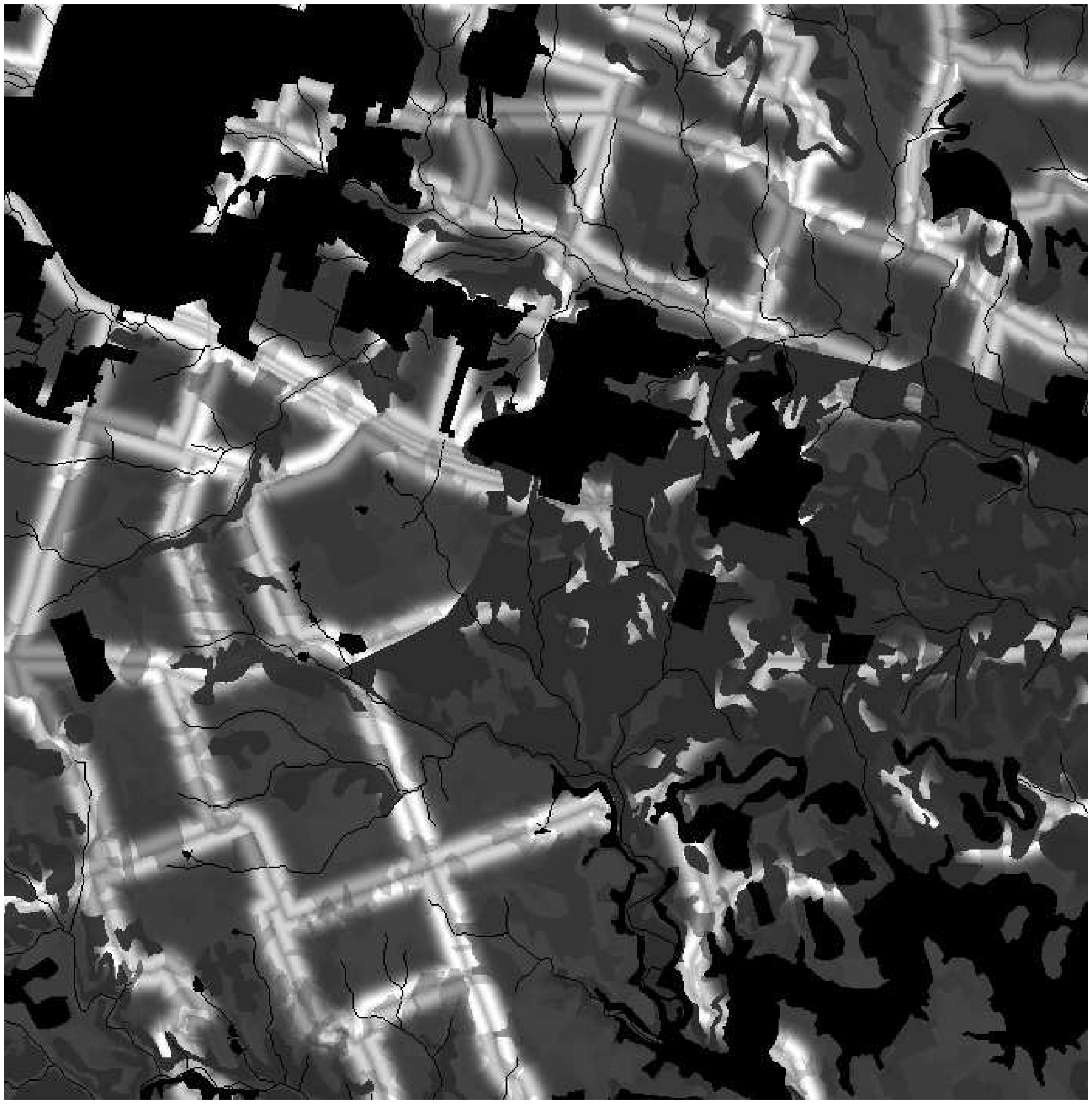

Figure 2: The terrain-based prior density.

are white, and the other features are not shown. Using actual terrain information and our simplified model, the prior density for target locations was calculated. This density is shown pictorially in Figure 2 for the same region of Figure 1. Lighter areas correspond to higher probability. The scaling is nonlinear to emphasize the black *zero probability regions.* Note the Gaussian ridges which follow along both sides of roads, interrupted somewhat by the effects of the other terrain features. This density was used in the simulation described in Section 6, in conjunction with the algorithm described in the following section.

4 A Simple Association-Estimation Algorithm Using Priors

In this section we present a simple data association and estimation algorithm which can be derived using the assumptions of Section 2 and the general hypothesis evaluation formulation of [MCTW86]. We first present the algorithm, followed by some notes on the algorithm. The algorithm is a "greedy" (or "nearest neighbor" [BB90] or "zero scan" [Rei79]) algorithm. This is because for each incoming report $z(k)$ the algorithm either associates $z(k)$ with a hypothesized previously seen target $\tilde{y}_i(k) \in \tilde{Y}(k)$ and updates the corresponding density $\hat{p}(y_i(k))$ with the new data to form the new estimate $\hat{X}(k+1)$, or else it assumes $z(k)$ is associated with a previously undetected target $\tilde{y}_{\hat{N}(k)+1}(k)$ and inserts a new density into $\hat{X}(k)$ to form $\hat{X}(k+1)$. The association is made which is *locally* optimal. Sensor reports are never again considered by the algorithm, thus a hard, irretrievable decision is made, unlike in batch algorithms [Mor77] and multiple hypothesis algorithms [Rei79]. The algorithm proceeds as if all previously made decisions were correct.

The algorithm is as follows. We maintain the set

$$\hat{X}(k) = \{\hat{p}(y_1(k)), \ldots, \hat{p}(y_{\hat{N}_D}(k))\} \tag{9}$$

of densities representing the current estimates for the states of all hypothesized targets $\tilde{Y}(k) = \{\tilde{y}_1(k), \ldots, \tilde{y}_{\hat{N}_D}(k)\}$ in the environment at time k. Recall that $\hat{N}_D$ is the hypothesized, or estimated, number of targets which have been detected. We refer to $\hat{X}(k)$ as the *estimate at time k.* Initially $\hat{X}(0)$ is the empty set. Because of the form of the updating, and because sensor errors are Gaussian, we can use the fact that the product of Gaussians is again Gaussian to represent the elements of $\hat{X}(k)$ as the normalized product of a single Gaussian and the terrain density:

$$\hat{p}(y_i(k)) = c_i^{-1}(k)\ G(a_i(k), A_i(k), y_i(k))\ r(y_i(k)), \tag{10}$$

where $a_i(k)$ is an n-vector, $A_i(k)$ is an $n \times n$, positive definite, symmetric matrix, and $c_i(k)$ is a normalizing constant. Recall that r is the terrain-based prior density function for target locations. We will simply write c^{-1} for normalizing constants when the meaning is clear.

As each sensor report $z(k+1)$ is received, we update $\hat{X}(k)$ to $\hat{X}(k+1)$ by performing the *association step* followed by the *estimation step.* These steps are described in detail in Figure 3. Some general notes on the algorithm follow and, in particular, the scalar-valued function S occurring in the association step is described in note (1).

1. The term $S(n,k)$ occurring in the algorithm is given by

   $$S(n,k) = \sum_{i=n}^{\infty} \frac{i!\, p(\tilde{N}_T = i)}{i^k (i-n)!}. \tag{16}$$

   This term uses the uniform sampling of targets assumption to modify the probability of seeing a new target based on the number of targets seen up to the present time, the number of reports received, and the prior density for the number of targets. (Because the algorithm is a greedy algorithm it takes the number of detected targets to be equal to the *estimated* number of detected targets). As the S term presents some computational difficulties we simply take $S(n,k) = e^{-n\tau}$. Here τ is a constant which functions similarly to the Poisson mean, and works out to give a simple thresholding of the log ratio of $bestPrevious$ and $previouslyUnseen$. This term illustrates how even simple models can give rise to complicated expressions in their optimal greedy algorithms, a point we will return to in Section 6.

2. The function combining densities into a posterior estimate, or "fusing" the data, is simply multiplication followed by renormalization because of the independence assumptions.

3. In the algorithm *description* we maintain the set $\hat{X}$ of densities giving estimates for the locations of each hypothesized target. In the *implementation*, though, we take $\hat{X}$ to be a set of generally *unnormalized* Gaussian densities. This is possible because of the Gaussian sensor error assumption, the reproducing properties of the Gaussian

association step There are three substeps:

1. Find the density $\hat{p}(y_q(k)) \in \hat{X}(k)$ which maximizes

$$S(\hat{N}_D(k), k+1) \int dy_q(k)\ G(z(k+1), V(k+1), y_q(k))\ \hat{p}(y_q(k)), \tag{11}$$

   or equivalently, using (10), the density which maximizes

$$\begin{aligned} S(\hat{N}_D(k), k+1) \int dy\ [G(z(k+1), V(k+1), y) \\ \cdot[c_q^{-1}(k)\ G(a_q(k), A_q(k), y)\ r(y)]], \end{aligned} \tag{12}$$

   where the (multiple) integrals are taken over all space. Call the maximum value *bestPrevious* (*bestPrevious* $= 0$ for $\hat{X} = \emptyset$).

2. Compute the "score value" for the report having been sensed from a target not hypothesized in $\tilde{Y}(k)$, given by

$$\begin{aligned} &previouslyUnseen = \\ &\quad S(\hat{N}_D(k)+1, k+1) \int dy\ G(z(k+1), V(k+1), y)\ r(y). \end{aligned} \tag{13}$$

3. Report $z(k+1)$ is assumed to represent a previously unseen target if *previouslyUnseen* $>$ *bestPrevious*. Otherwise it is assumed to be associated with the hypothesized target $\tilde{y}_q$, whose density $\hat{p}(y_q) \in \hat{X}(k)$ maximized *bestPrevious* in step 1.

estimation step In this step we update $\hat{X}(k)$ to $\hat{X}(k+1)$, i.e., we update our current estimate *assuming* the association step produced the true association. If z represents a previously unseen target then $\hat{N}_D(k+1) = \hat{N}_D(k) + 1$, and the estimate $\hat{X}$ is updated as

$$\begin{aligned} \hat{X}(k+1) &= \hat{X}(k) \cup \{\hat{p}(y_{\hat{N}_D}(k+1))\} \\ &= \hat{X}(k) \cup \left\{c^{-1}\ G(z(k+1), V(k+1), y_{\hat{N}_D})\ r(y_{\hat{N}_D})\right\}, \end{aligned} \tag{14}$$

where c^{-1} is the normalizing constant. Otherwise, $\hat{X}(k+1) = \hat{X}(k)$ except we replace the density chosen in step 1 of the association step by

$$\begin{aligned} \hat{p}(y_q(k+1)) = c^{-1} G(a_q(k), A_q(k), y_q(k+1)) \\ \cdot G(z(k+1), V(k+1), y_q(k+1))\ r(y_q(k+1)). \end{aligned} \tag{15}$$

Figure 3: The association-estimation algorithm.

density, and the fact that we know and save a single ("pixelmap") representation for $p(x)$, which was assumed common to all targets. The Gaussians are generally unnormalized, but are scaled so their products with the terrain, i.e., the posterior densities, *are* normalized.

4. The integrals, as in *bestPrevious*, of the product of two densities over all space can be considered as a similarity measure between the densities: the larger the value the "more similar" the densities are.

5 A One-Dimensional Example

In this section we look at a simple one-dimensional example and consider the question of when the terrain density $p(x)$ makes a significant contribution in association, and when its contribution is significant in estimation of densities for hypothesized targets in $\hat{X}$. As an example in one dimension we consider the case of a sensor report with position estimate 0 and sensor error variance 1. Thus the likelihood given the sensor report is Gaussian with mean 0 and variance 1. For the purposes of this example the terrain density is given by the shifted, normalized cosine wave

$$p(x) = \begin{cases} \frac{1+\cos(2\pi x/\lambda+\delta)}{\int_{-4}^{4} dx \; (1+\cos(2\pi x/\lambda+\delta))} & \text{if } -4 \leq x \leq 4 \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

Here λ is the wavelength and δ is the phase. The two densities are shown in Figure 4, with $\lambda = 1$ and $\delta = 0$. In Figure 5 the posterior density for target location is shown. This density, obtained by normalizing the product of the terrain density and the sensor report Gaussian, would be added to $\hat{X}$ if it were determined that the sensor report represents a previously unseen target. Figures 5 and 6 show the same situation except the phase of the terrain is shifted to $\delta = \pi$.

Note that without knowing the model for generating the prior density from the terrain features we cannot conclude anything about the terrain itself based the knowledge that the prior density is a cosine function. If, for example, the probability of a target locating at any point were known to be proportional to the elevation (or the continuous-valued slope) at the point then we could conclude that the terrain had hills and valleys modeled by a sine/cosine wave function.

We now consider when using the cosine terrain prior results in significantly different data association results versus using the 1-D *uniform* prior.

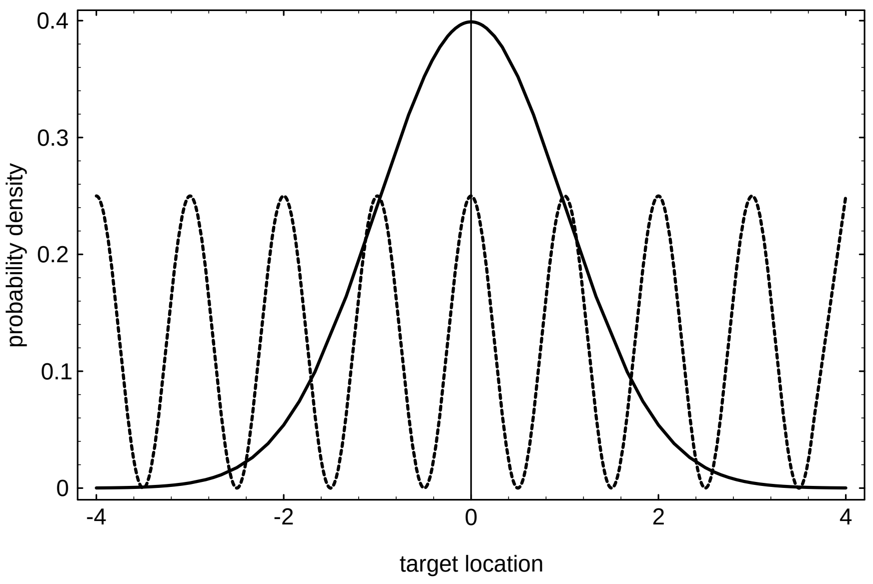

Figure 4: A 1-D example of a sensor report and cosine terrain with $\delta = 0$.

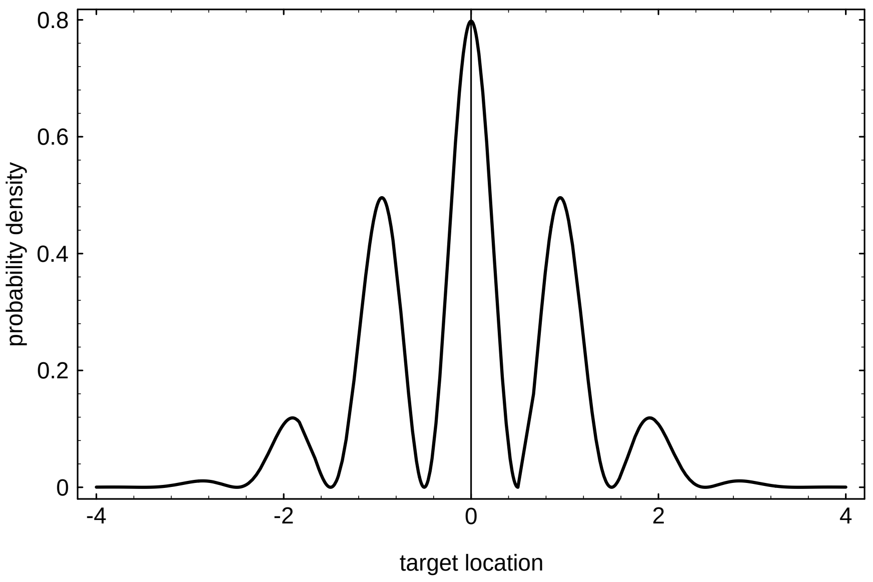

Figure 5: The posterior density for target location.

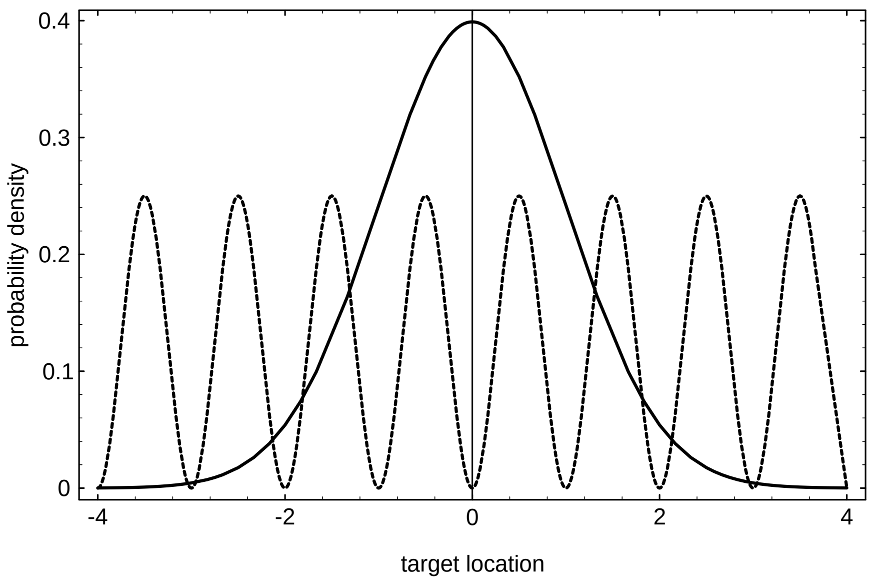

Figure 6: A 1-D example of a sensor report and cosine terrain with $\delta = \pi$.

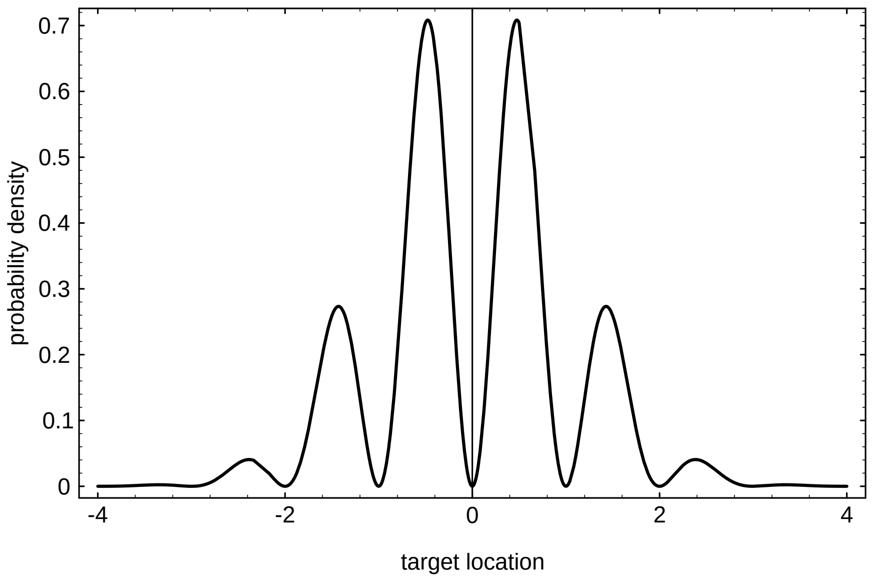

Figure 7: The posterior density for target location.

We first consider the case where the sensor report is assumed to represent a previously unseen target, and we take $S(.,.) = 1$. In this case the relevant variable is $previouslyUnseen$ in (13), i.e. the integral of the product of the Gaussian and the terrain density, versus the integral of the Gaussian times the 1-D uniform density. If the value of $previouslyUnseen$ calculated using the cosine density is approximately equal to the value computed using the uniform density then there is no significant advantage to using the more complicated cosine prior.

A three-dimensional plot of the $previouslyUnseen$ values for the cosine terrain case, as a function of λ and δ, is shown in Figure 8. We have subtracted off the integral of the Gaussian over the uniform terrain, i.e. the constant $previouslyUnseen$ value computed using the uniform terrain, so that using the cosine terrain is equivalent to using the uniform terrain when the function value is zero. The two-dimensional plot in Figure 9 shows the same function with λ varying and δ values of 0 (solid), $\pi/2$ (dotted), and π (dot-dash). Notice that with the phase equal to 0 and the wavelength λ of the terrain prior on the order of six standard deviations of the Gaussian the terrain is maximally effective at *increasing* the value of $previouslyUnseen$ relative to the uniform terrain. As the wavelength increases to infinity *and* as it decreases to zero the value of the integral approaches that of the uniform terrain case.

The nonuniform terrain prior is most effective at *decreasing* the $previouslyUnseen$ value relative to the uniform case when the phase is $\delta = \pi$. In this case, as the wavelength goes to zero, the cosine terrain case approaches the uniform terrain case. If the cosine density had an additive constant to keep it strictly positive it would also approach the uniform terrain case as the wavelength approached infinity. Thus we see that for the prior to be most effective in association its local variation should be on about the same order as that of the sensor reports.

When we consider the effectiveness of the terrain prior verses the uniform prior in estimation, the situation is somewhat different than for association. Consider, say, the integrated squared or absolute difference between the posterior estimate using the cosine terrain versus the posterior estimate using the uniform terrain. In this case the two results converge as the wavelength approaches infinity, but the terrain becomes more and more influential as the wavelength approaches zero. In practice, though, estimates are needed only to some fixed precision, and there is a point past which decreasing the wavelength is no longer helpful relative to the uniform terrain. In Figure 10 we show, for the example given above, the posterior densities

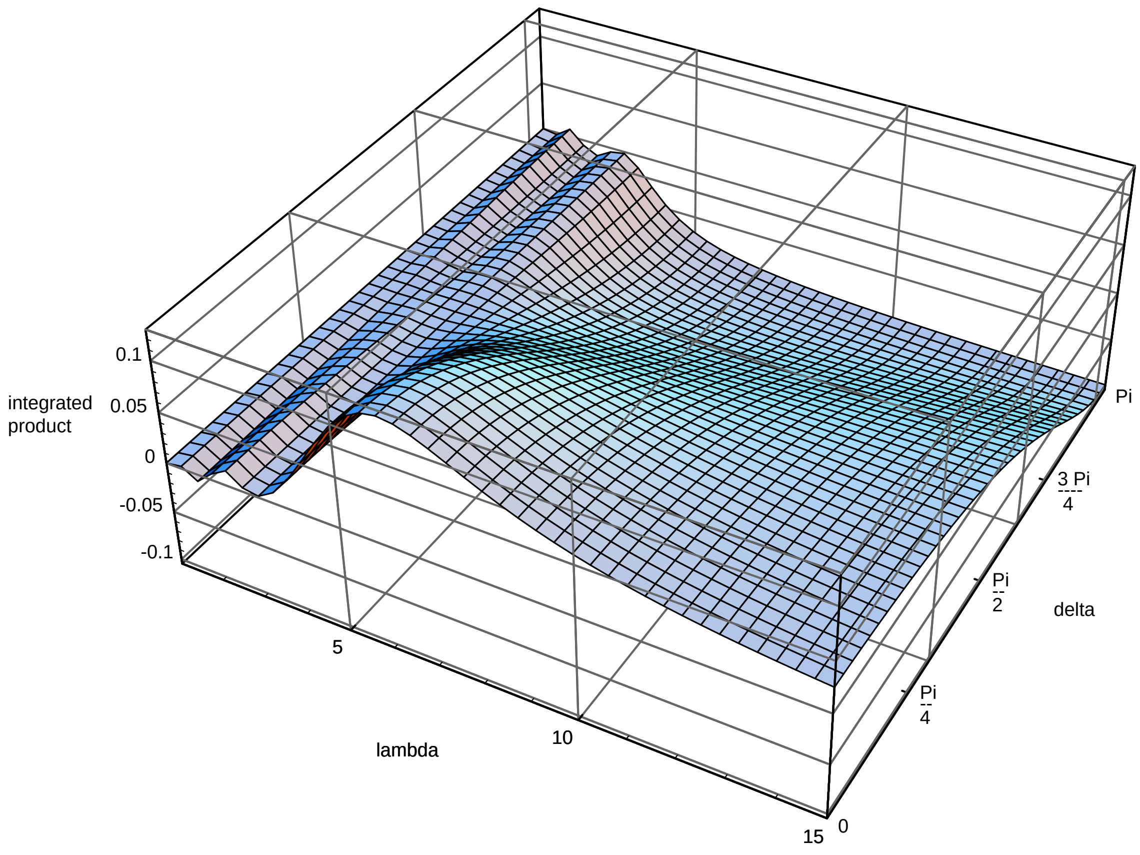

Figure 8: The integral of gr as a function of λ and δ.

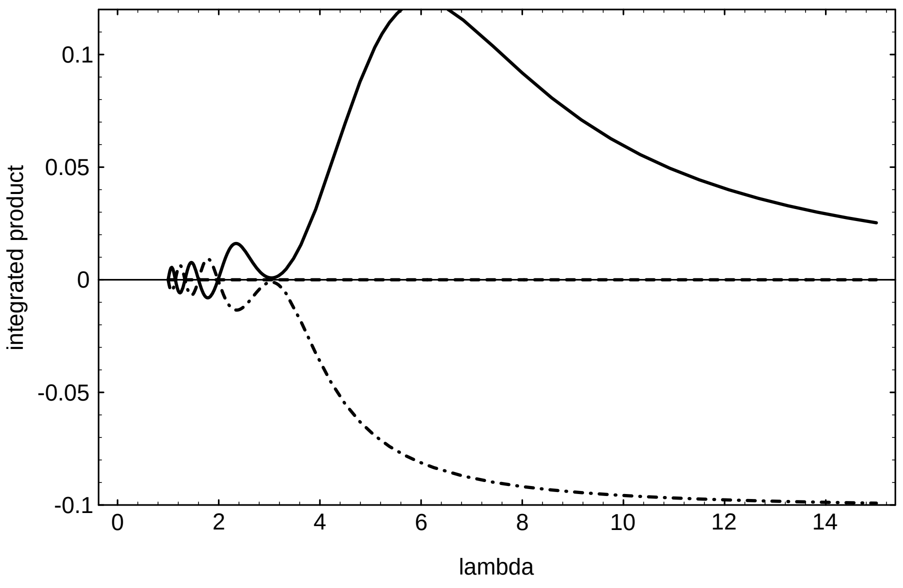

Figure 9: The integral of gr at $\delta = 0$, $\delta = \pi/2$, and $\delta = \pi$ as a function of λ.

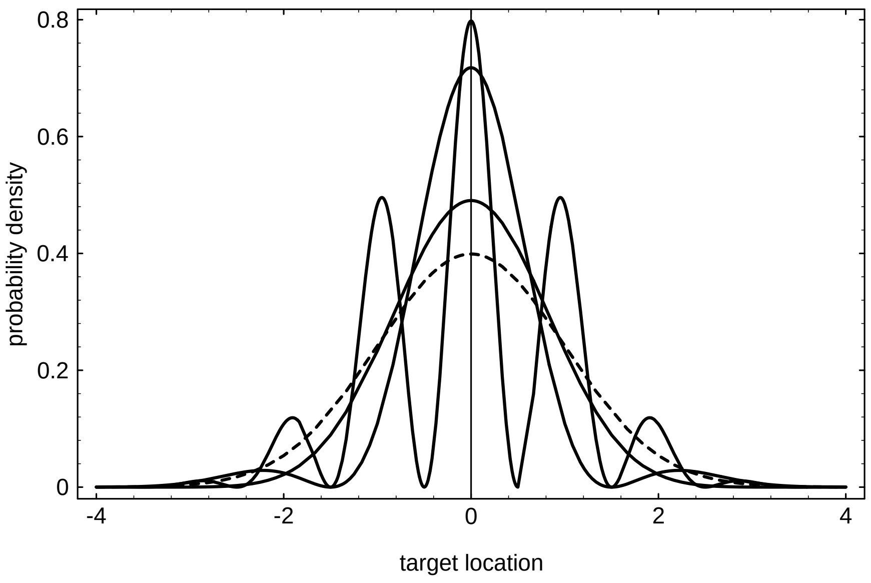

Figure 10: Posterior densities converging to the uniform terrain posterior (dashed).

for uniform terrain and for cosine terrain with $\delta = 0$ and $\lambda = 1$, $\lambda = 3$, and $\lambda = 6.5$. The uniform terrain estimate is given by the Gaussian sensor report density normalized for $-4 \leq x \leq 4$, and is the dashed curve. As λ increases the posterior densities approach the Gaussian uniform prior estimate.

The case where a sensor report is assumed to correlate with a previously seen target is similar to the previous case, except that we must consider the Gaussian which results from multiplying the sensor report density with the density in $\hat{X}$ corresponding to the target it is hypothesized to be associated with. Since we can represent all the estimates in $\hat{X}$ as a product of a Gaussian and the terrain density a qualitative analysis similar to that given above can be carried out.

We end this section with a mention of some other effects which can render the prior ineffective. Perhaps the most important is the number of targets per unit area in a region relative to the spread of the Gaussian sensor reports. If the densities do not tend to overlap in a significant way then a simple Mahalanobis distance threshold will be sufficient. Of course a Mahalanobis distance gating should always be used before performing the more expensive nonuniform prior operations, so a nonuniform prior algorithm could still be employed. If a batch algorithm is being used, then well formed clusters of sensor reports may appear, making association trivial. In this case the terrain may not be especially useful for estimation either, since the Gaussian estimates become more and more peaked as more reports are added, with the prior eventually approaching the large wavelength case relative to the Gaussian.

6 Some Simulation Results

In this section we describe simulation runs we have performed. Using the terrain density described in Section 3 we generated artificial data and tested several variations of the association-estimation algorithm described in Section 4. We first describe the data generation process. We then describe the variations of the algorithm we studied. Next, we define the performance measure we use as a basis for algorithm comparison. Finally, we present the results of some simulations. We note that these simulations are intended only to be illustrative of the general methods employed. Due to the heavy computational burden involved in repeatedly computing integrals of Gaussians over the terrain prior density, we have not attempted more exhaustive tests.

The target locations and sensor reports were generated as follows. First, 200 targets were sampled from the density $p(x) \equiv r(x)$, described in Section 3. Of these targets, only those lying in the central 7 km by 7 km rectangle of the entire 14 km by 14 km region were chosen, and the rest discarded. Note that this reduces edge effects but also violates the model assumed in deriving the association-estimation algorithm. That is, we do not have $\hat{p}(x) = p(x)$ exactly, but we ignore this discrepancy. The target locations were then sampled uniformly with replacement, and for each target sampled a sensor report was generated. Sensor report location estimates $z(k)$ were created by adding zero mean Gaussian noise to the true location. The covariance matrices of this sensor noise were generated according to a model of an airborne sensor platform. See [BPS92] for more details. At time k the current sensor error covariance matrix $V(k)$ is reported without error, along with the location estimate $z(k)$, to make up a sensor report. A total of 28 independent sets of target locations were generated, and for each set of target locations 100 reports were generated. Thus the total collection of data consisted of 28 independent data sets, $W = \{w_1, \ldots, w_{28}\}$, with each data set having 100 reports.

As presented in Section 4, the association-estimation algorithm uses a common prior $p(x)$ in both the association step and the estimation step. We call the algorithm using the uniform prior for both association and estimation UU, and the algorithm using the terrain-based true prior density $p(x)$ for both association and estimation TT. In addition to these algorithms we tested an algorithm we call UT. This algorithm works exactly like UU except that it uses the terrain-based prior for its *reported* estimate, but not for its *saved* estimate. Thus, like UU, its saved estimate is always a set of Gaussians and it performs identically to UU in terms of data association. Its score, however, is based on its reported estimate, which is formed by multiplying each of its saved-estimate Gaussians with the terrain prior and renormalizing (and saving the normalizing values). This has the advantage that since we approximate the uniform density as existing over all space the integrals in the association step can be performed analytically, while the terrain prior is still used in estimation.

In order to compare the performance of the algorithms we define the following measure of algorithm performance. Let $EstimateMixture(A(w, \tau), x)$ be the equally weighted mixture density[5] of all the densities in the estimate $\hat{X}(k)$ produced by running algorithm A on data set w with parameter value

[5] A mixture density is just a weighted sum of densities.

τ. Let $PerfectMixture(w, x)$ be the equally weighted mixture of all the densities in the estimate produced by performing *perfect* data association on data set w followed by the estimation step using the true prior density $p(x)$. The performance measure we use is the root integrated squared difference of the two mixtures:

$$Score(A(w,\tau))^2 = \int dx[EstimateMixture(A(w,\tau),x) - PerfectMixture(w,x)]^2. \tag{18}$$

The lower an algorithm's score the better, and the perfect association algorithm has a score of 0.

In defining the performance measure as we have, we are assuming that the estimate produced with perfect association and using prior density $p(x)$ in estimation can be used as a basis for judging performance. In a situation where the true target locations are known but perfect association information is not available, or where the sensor error densities are only approximations, it may be preferable to use a similar score measure but using the true locations instead of $PerfectMixture$. See [SPB89] for an axiomatic approach to performance measures of a similar type.

In theory, we could simply compute the true probability of each data association hypothesis given Z or compute the true posterior density value for any estimate given Z. We could then use one of these values as the performance measure. This assumes, though, that the model assumed truly matches the data distribution and that the algorithm used to calculate the probabilities is the correct algorithm. The first assumption is generally not satisfied by real-world data. The second assumption may not be satisfiable unless a reasonably efficient algorithm for computing the probabilities can be found. For these reasons we have chosen the less model-dependent performance measure above. Expanding the square in (18) allows the integrals to be computed as sums of integrals over Gaussians times the terrain prior density, or the squared terrain prior density.

Now that we have defined the measure of algorithm performance, we present some simulation results. In Figures 11-13 we show the average score values versus τ for algorithms UU, UT, and TT. Recall that τ is a parameter which functions similarly to the mean number of targets μ_{N_T}; as τ increases the estimated number of detected targets $\hat{N}_D$ tends to decrease. The score values are scaled by $10^{4.5}$, and the error bars show one standard deviation above and below the mean. As expected, algorithm UU performs uniformly worse than than the other algorithms. This is expected because the other

	TT	UT	UU
TT	-	.63	.99
UT	.37	-	.99
UU	.005	0.00	-

Table 1: Estimated probability one algorithm (row) is better than another (column).

algorithms are making use of information it does not have, and the score is based on this information too. Algorithm UT, while identical to UU in terms of data association, uses the terrain prior in its reported estimate to become competitive with algorithm TT. Notice that algorithm TT does not vary as smoothly with τ as the other algorithms; using the terrain in association can increase the sensitivity to parameters. At its best, though, TT outperforms the others. In Figures 14 and 15 we show the average of absolute error in estimating the number of detected targets, i.e., the averages of abs($\hat{N}_D - N_D$) after 100 reports versus τ.

Next, we statistically compare algorithms by computing approximate probabilities for one algorithm to perform better than another. We define one algorithm A1 to perform better than another algorithm A2 iff the best expected score of A1 is less than the best expected score of A2, with best taken over the free τ parameter. We discuss statistical comparisons based on scalar score measures in more detail in Appendix B; see also [BSF88]. The results of comparing UU, UT, and TT are shown in Table 1. The algorithms are compared at their best τ parameter settings so, for example, TT is given parameter setting $\tau = -1.7$. As the table shows, both UT and TT significantly outperform UU, but the differences between TT and UT are too small to be significant.

7 Extensions

There are a variety of possible extensions to the basic model and algorithms we have presented, most of which are still covered by the very general framework of [MCTW86]. Some of these extensions include higher dimensional state vectors, non-static cases with motion models, non-Gaussian sensor reports, more general sensor models, more sophisticated data association and hypothesis management techniques, time dependent priors, and non-independent targets. In this section we consider only the first two.

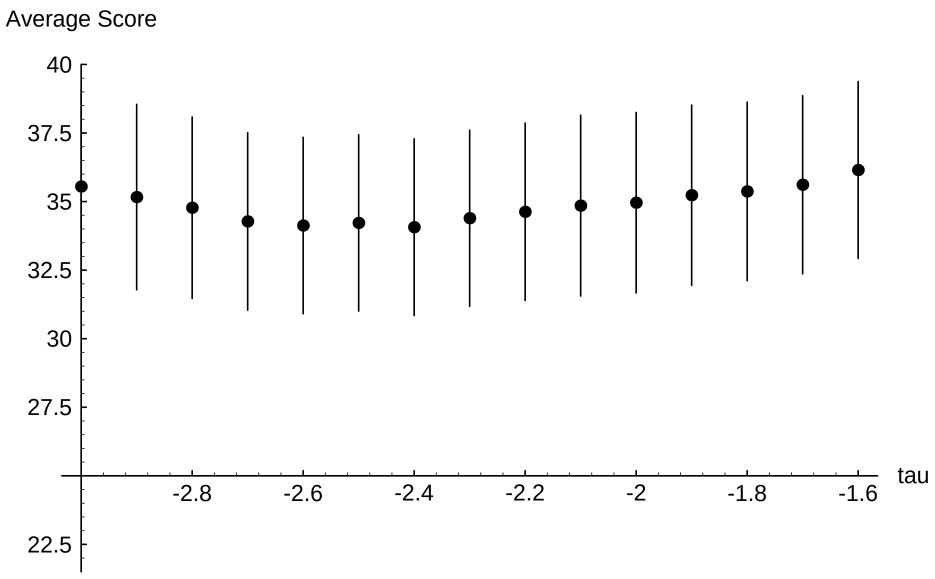

Figure 11: Average scores for algorithm UU vs. τ.

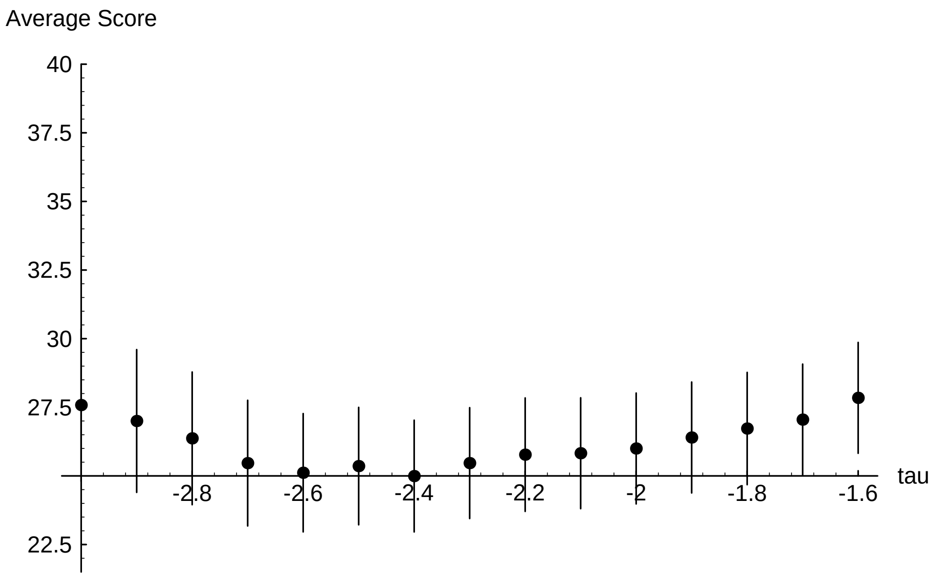

Figure 12: Average scores for algorithm UT vs. τ.

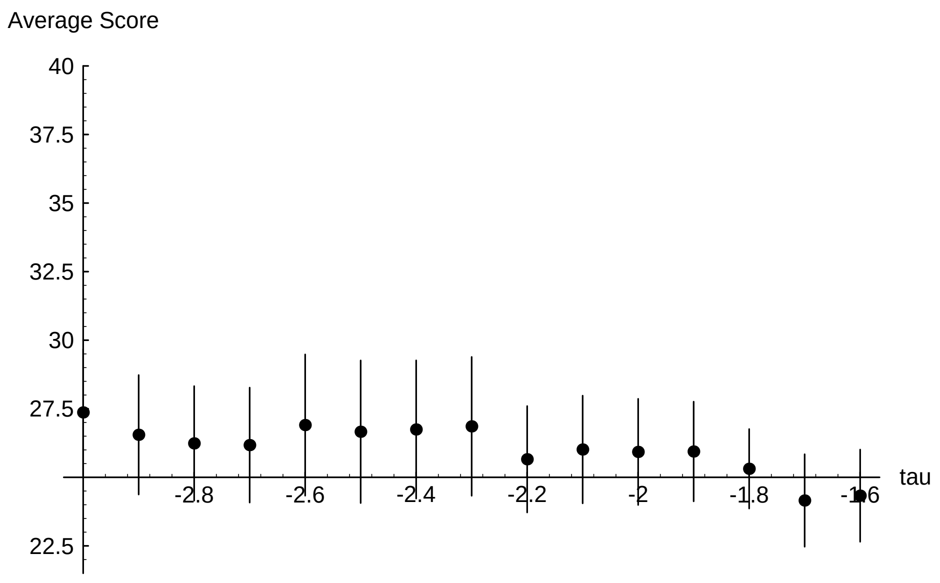

Figure 13: Average scores for algorithm TT vs. τ.

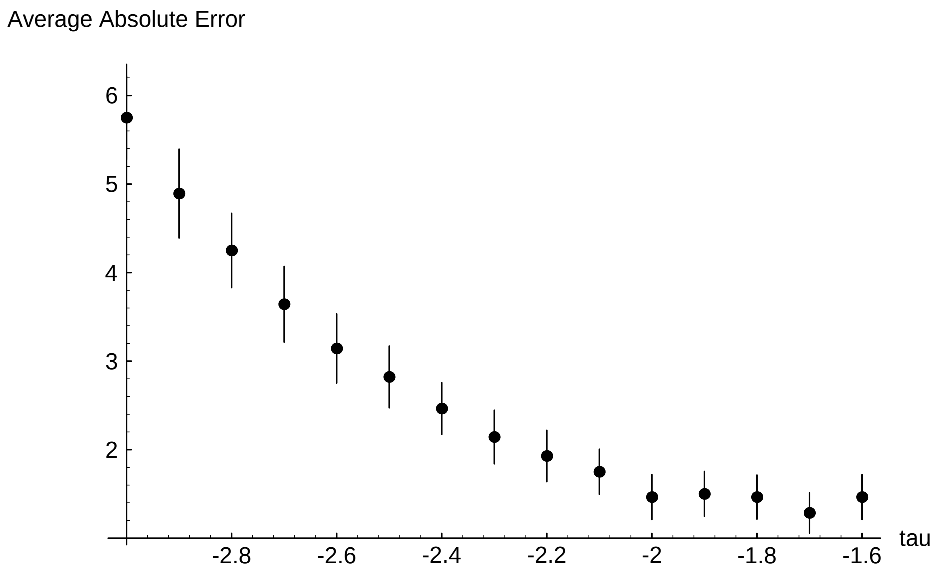

Figure 14: Average absolute error of $\hat{N}_D$, algorithms UU and UT.

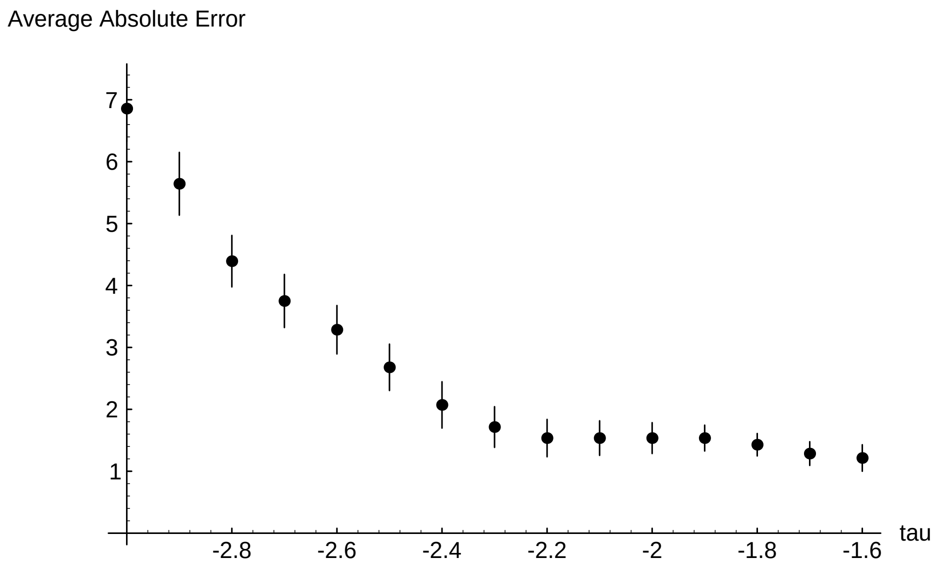

Figure 15: Average absolute error of $\hat{N}_D$, algorithm TT.

The extension to higher dimensional state vectors is straightforward in theory; the algorithms we have presented still hold. Practical problems arise, though, in the representations of the high dimensional densities and in performing the required integrations. These problems become even more severe when we consider the non-static case with a motion model describing the time evolution of states.

By a motion model we mean that we have a model of density $p(x_i(k+1)|x_i(k))$. Using this model we can update, or time-project, all the densities in the database to the time instance of the newest report, thus reducing the problem to the static case. This is done recursively with Bayes' rule as

$$p(x_i(k+1)) \propto \int dx_i(k)\ p(x_i(k+1)|x_i(k))p(x_i(k)). \tag{19}$$

The base case of the recursion is specified by $p(x_i(q)) \equiv r(x_i(q))$, where t_q is the time target i was first hypothesized. The motion model may or may not also depend on terrain. With the standard linear-Gaussian motion model, independent of terrain except for the initial prior, we get an algorithm very much like the static one we have presented except that we time project the Gaussian parts of the estimate as in Kalman filtering. The more realistic case is when the motion model is also dependent on terrain features.

The primary difficulty which arises when the motion model is dependent on terrain features is that the densities involved are in general highly non-Gaussian. Representing these densities and computing integrals over them can be computationally expensive, both in association and estimation and in time-updating the densities. In [RB79] a terrain-based motion model is defined, along with the appropriate association and estimation equations. The approach taken is to discretize all variables: space, time, velocity, and a discrete moving/not moving variable. Then, from a high-level set of basic assumptions about the expected movement of targets, they implicitly define a motion model by specifying an algorithm for computing $p(x_{k+1}|x_k)$ based on this set of assumptions. This algorithm is referred to as the time projection equations. The constant time intervals are taken to be small enough that only local neighborhoods need to be considered in the updating. The updating step is repeated until the time matches that of the current reports, reducing to a static problem.

When the motion model depends on terrain information the use of terrain-based densities (and conditional densities) becomes potentially much more effective than in the static case. In the static case the prior density gradually becomes less and less influential with time as the estimates become more peaked. When the targets' movements depend on the terrain features, though, the terrain information can remain highly influential over time.

8 Conclusions

It is clear mathematically that the use of prior information can improve both association and estimation algorithms. In practice, though, using even moderately complicated prior densities is computationally expensive. In many cases the use of prior information does not result in *significant* improvements, especially when used for data association. With limited computational resources it may be preferable to devote the available resources to other techniques, such as multiple hypothesis algorithms, batch algorithms, or the use of higher-level reasoning.

While using the prior density can be expensive, there are cases where the extra expense is justified and where the prior provides especially valuable information. This suggests a hybrid approach. Dynamically allocating the available resources to trade off between the various approaches available should result in an algorithm with high performance across a broader range of input statistics. The problem then becomes to determine heuristics for effectively making the tradeoff decisions. Analyses such as those we have presented for the static problem can serve as a first step for developing these heuristics.

Another area which needs further research is the representation scheme for the prior densities. The appropriate representation is a function of the class of densities to be represented, the expected class of sensor report and target state distributions, and of the particular association-estimation algorithms being used. While we have simply discretized over a rectangular grid, many other schemes are available such as Gaussian mixtures, vectorized representations, neural nets, etc. In particular, for the cases we have considered, Fourier domain density representations (characteristic functions) might be appropriate, especially if they can be efficiently band-limited to the most useful frequency ranges.

References

[Bar97] Allen L. Barker. *Selection of Distance Metrics and Feature Subsets for k-Nearest Neighbor Classifiers.* PhD thesis, University of Virginia, Computer Science Department, May 1997. ftp://ftp.cs.virginia.edu/pub/dissertations/9703.ps.Z.

[BB90] S. S. Blackman and T. J. Broida. Multiple sensor data association and fusion in aerospace applications. *Journal of Robotic Systems*, 7(3):445–485, 1990.

[BPS92] D.E. Brown, C.L. Pittard, and A.R. Spillane. Asset: A simulation test bed for evaluating data association algorithms. *Computers and Operations Research*, 19(6):479–493, 1992.

[BSF88] Yaakov Bar-Shalom and Thomas E. Fortmann. *Tracking and Data Association.* Academic Press, 1988.

[BT73] G.E.P Box and G.C. Tiao. *Bayesian Inference in Statistical Analysis.* Wiley, 1973.

[Eve93] B.S. Everitt. *Cluster Analysis.* Edward Arnold, 3rd edition, 1993.

[MCTW86] Shozo Mori, Chee-Yee Chong, Edison Tse, and Richard P. Wishner. Tracking and classifying multiple targets without a priori identification. *IEEE Transactions on Automatic Control*, AC-31(5):401–409, 1986.

[Mor77] Charles L. Morefield. Application of 0-1 integer programming to multitarget tracking problems. *IEEE Transactons on Automatic Control*, AC-22(3):302–312, 1977.

[RB79] Donald B. Reid and Robert G. Bryson. A non-gaussian filter for tracking targets moving over terrain. In *Proc. 17th (1978) IEEE Conference on Decision and Control*, pages 112–116, January 1979.

[Rei79] Donald B. Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, AC-24(6):843–854, 1979.

[SPB89] A.R. Spillane, C.L. Pittard, and D.E. Brown. A method for the evaluation of correlation algorithms. Technical Report IPC-TR-89-004, Institute for Parallel Computation, School of Engineering and Applied Science, University of Virginia, April 1989.

[Wol91] Stephen Wolfram. *Mathematica: A System for Doing Mathematics by Computer.* Addison-Wesley, 1991.

A Some Implementation Details of the Simulation

The first step in the simulation process is to create the prior density. The 14 km by 14 km region of interest is broken up into a rectangular grid of 20 by 20 meter cells, giving a 700 by 700 array of cells, where columns run north and south. We are given all the feature values at each cell, except the Euclidean distance to the nearest road $RoadDist(x)$. Instead of $RoadDist$ we are given which cells correspond to roads, and must compute the distances for each cell.

An array of $RoadDist$ values can be computed relatively efficiently as follows. Assume a 700 by 700 array $NorthSouthRoadDist$, initially with zero at all road cells and infinity in all others. Now, for each column do the following. Iterate over the column, replacing all nonzero values by the minimum of the values in the cells to the immediate north or south, plus (in this case) 20 meters. Continue this until no values change; one forward pass followed by one backward pass is sufficient. After this process has been completed for all columns the $NorthSouthRoadDist$ array contains, at each cell, the shortest distance from that cell to a road in the north or south direction. Now square all these values. Next, using these precomputed values, compute the final *RoadDist* array as follows. At each cell (i, j) scan outward in both the east and west directions. Let k be the number of cells from the current cell, with k initially 0. Let $d = (20k)^2$ be the "current squared east-west distance". With k increasing, save the minimum value of d plus either of the precomputed values stored in the $NorthSouthRoadDist$ array at $(i, j + k)$ or $(i, j - k)$, i.e, k cells to the east or west of the current cell. Do this until the saved minimum value is less than d. At this point the saved value is the minimum squared distance to a road from the current cell, by the Pythagorean theorem. Taking the square root of all these values completes the algorithm. Since the maximum value of d at any cell is at most equal to the squared distance to the nearest road, the maximum value of k at any cell is less than or equal to the nearest road distance divided by 20. If we let n be the total number of cells, this algorithm has a runtime bound of $O(nm)$, where m is the average distance from any cell to the nearest road. An alternative algorithm, where an expanding circle of cells around any given cell is examined until a road is encountered, has a runtime bound of order n times the expected *squared* distance from any cell to the nearest road. We note that if Manhattan distances are acceptable then a simple local updating algorithm at each cell can be used. This type of algorithm has the advantage that distances around obstacles can be easily taken into account.

Once we have all the feature values we compute the prior likelihood for each feature type. For each possible value of a given feature a likelihood for a target locating at a point having that feature value is defined. These values are then combined to form, for each feature type, a likelihood function over the entire field. These likelihood functions are stored as a two-dimensional array of bytes. Finally, the likelihoods for each feature type are combined by multiplying and rescaling to fit into another two-dimensional array of bytes, giving the final terrain-based likelihood.

At this level of the implementation the likelihoods are stored as 700 by 700 arrays of bytes. Also at this level we compute the 700 by 700 *cumulative* likelihood array defined so any cell (i, j) contains the *sum* of likelihood values for all cells (k, l) with $k \leq i$ and $l \leq j$. Similarly, we also compute the cumulative squared likelihood array. Note that these are no longer arrays of byte-valued quantities. The cumulative array is useful because with it the sum of the likelihood array values over any rectangular region (aligned with the rows and columns of the array) can be computed in constant time, i.e., with 4 array indexing operations. This operation occurs quite frequently in approximating integrals over the terrain prior.

At the next higher level of abstraction the likelihood and cumulative functions are defined for floating point arguments. The sum of the likelihood over an arbitrary floating point rectangle aligned with the rows and columns of the original array can still be computed in constant time, though several of the lower level cumulative function calls (i.e. array lookups) are required to implement this functionality.

Finally, at the highest level of abstraction for terrain operations we implement a class which approximates the integral of a given Gaussian times the terrain prior. Gaussian densities are defined as a C++ class, with, for example, the multiplication operator overloaded to return the product of Gaussians, which is again Gaussian. To approximate the integral over a Gaussian times the terrain prior the Gaussian is approximated by a fixed number of concentric, hollowed out elliptic cylinders. Thus the smooth bivariate Gaussians are approximated by a series of constant-valued "terraces" or "steps". The ellipses correspond to constant probability contours of the Gaussian, and each "terrace" level has its inner and outer "radius" values chosen so it contains a fixed fraction of the total probability mass of the Gaussian. The height of each terrace is then defined by the probability mass it must contain. The approximate integration algorithm proceeds by calculating the innermost elliptic cylinder, breaking the ellipse up into a fixed number of vertical strips, and summing the terrain density values for each strip using the cumulative function for the terrain density. This sum is then corrected for the difference between total strip area and ellipse area by averaging. This final value is multiplied by the height value and added to the running integral total. Next, the second ellipse from the center is treated in a similar way, with the previously calculated inner ellipse value subtracted out before multiplying by its height value. This process continues outward until a given fraction of the bivariate Gaussian mass, we use 99%, has been covered. Correcting for this factor gives the final integral approximation. The same technique is used to approximate the integrals of Gaussians over the squared terrain density, an operation used in the scoring function. The algorithm can be implemented fairly efficiently, and additional tricks can be used for more speedup. The important point is that the time to perform an integration is not a function of the spread of a Gaussian, since these spreads may be quite large relative to the cell size of the discretized terrain density.

B Statistical Comparisons of Algorithms

In this appendix we discuss the statistical comparison of algorithms based on the expected value of a given score measure. We adopt a Bayesian approach and calculate an estimate of the posterior probability that an algorithm $A1$ is "better than" another algorithm $A2$, given a random sample of data sets. We assume algorithms map data sets to output data structures, and that we are given a score function $Score$ mapping these output data structures to the reals. We also allow an algorithm to take as input a vector of free parameters. Using functional notation we write the score of algorithm $A1$ on data set w with parameter vector τ_1 as $Score(A1(w, \tau_1))$, and similarly for $A2$.

We now define a statistic which captures the notion that the performance of algorithm $A1$ is better than that of another algorithm $A2$ over the distribution of data sets. Informally, we take as a definition that $A1$ is better than $A2$ if and only if the best expected score of $A1$ is less than the best expected score of algorithm $A2$. Thus *lower* score values are better than higher ones. The parameter vectors τ_1 and τ_2 for algorithms $A1$ and $A2$ are assumed fixed at the *optimal* parameter value for their respective algorithms. Of course, many other definitions are possible; this definition was chosen because it seems to agree with an intuitive notion of "better than" while at the same time being relatively straightforward to calculate. If we had distributions on the parameter vectors τ we could take expectations with respect to these vectors instead of optimizing over them, but we assume this information is unavailable.

We can define our meaning of "better than" more formally as follows. Assume we are given a collection $W = \{w_1, \ldots, w_L\}$ of data sets. Let $\mathcal{D}$ be the data set distribution, and assume the data sets w_i, by definition, to be independent random samples from this distribution. Thus the only information we have about the data set distribution $\mathcal{D}$ is that provided by the given collection W of data sets along with the independent random sampling assumption. Let random data set $\tilde{D}$ also have distribution $\mathcal{D}$. The statistic we consider is

$$\tilde{M} = Score(A1(\tilde{D}, \tau_1^*)) - Score(A2(\tilde{D}, \tau_2^*)), \tag{20}$$

where τ_1^* and τ_2^* are the optimized parameter vectors, to be defined next. The expected value of $\tilde{M}$ is given by

$$E[\tilde{M}] \quad = \quad E[Score(A1(\tilde{D}, \tau_1^*))] - E[Score(A2(\tilde{D}, \tau_2^*))], \tag{21}$$

$$\equiv \min_{\tau_1} E[Score(A1(\tilde{D}, \tau_1))] - \min_{\tau_2} E[Score(A2(\tilde{D}, \tau_2))]. \quad (22)$$

We can now make the formal definition

Definition 1 *Algorithm* $A1$ *is* better than *algorithm* $A2$ *iff* $E[\tilde{M}] < 0$.

The problem of estimating the probability $A1$ is better than $A2$ has now been reduced to estimating the probability that $E[\tilde{M}] < 0$. Letting $\mu = E[\tilde{M}]$ we equivalently wish to estimate prob$[\mu < 0]$. Notice that for each data set w_i we can obtain an independent random sample of $\tilde{M}$ by running algorithms $A1$ and $A2$ on data set w_i and computing the score measures. That is,

$$M_i \equiv Score(A1(w_i, \tau_1^*)) - Score(A2(w_i, \tau_2^*)) \quad (23)$$

is a realization of a sample random variable $\tilde{M}_i$ having distribution identical to that of $\tilde{M}$. Thus, assuming τ_1^* and τ_2^* are known, we can apply standard maximum likelihood or Bayesian estimation techniques to estimate the distribution of μ.

If we assume the central limit theorem is valid here, i.e., we assume the distribution of $\tilde{M}$ has mean μ and finite positive variance σ^2, then we can obtain a Gaussian estimate for μ. That is, letting $\tilde{s} = (1/L)\sum_{i=1}^{L} \tilde{M}_i$ be the sample mean, we have

$$p(s|\mu, \sigma^2) \approx G(\mu, (\sigma^2/L), s) \quad (24)$$

for "sufficiently large" sample sizes L. Now, from Bayes' rule we have

$$\begin{aligned} p(\mu|s, \sigma^2) &\propto p(s|\mu, \sigma^2)p(\mu|\sigma^2) && (25) \\ &= p(s|\mu, \sigma^2), && (26) \end{aligned}$$

where we have taken $p(\mu|\sigma^2) = p(\mu) = const$ as the uninformative prior. Note that we are considering σ^2 known; for simplicity we will just replace σ^2 by the sample variance $S^2 = (1/(L-1))\sum_{i=1}^{L}(M_i - s)^2$. More properly we should integrate $p(\mu, \sigma^2|s, S^2)$ over σ^2 and obtain a t distribution [BT73]. Plugging in approximate expressions we get

$$\begin{aligned} p(\mu|s, \sigma^2) &\approx G(\mu, (\sigma^2/L), s) && (27) \\ &\approx G(\mu, (S^2/L), s). && (28) \end{aligned}$$

We can now write

$$\begin{aligned} \text{prob}[A1 \; \textit{better than} \; A2] &\equiv \text{prob}[\mu < 0] & (29) \\ &= \int_{-\infty}^{0} d\mu \; p(\mu | s, \sigma^2) & (30) \\ &\approx \int_{-\infty}^{0} d\mu \; G(s, (S^2/L), \mu). & (31) \end{aligned}$$

Values for this Gaussian integral may be looked up in standard probability tables, or obtained from standard function calls in most computer systems.

Now, we still need to address the minimization with respect to parameter vectors τ_1 and τ_2. Note that from (22) we have

$$\tau_1^* = \arg\min_{\tau_1} E[Score(A1(\tilde{D}, \tau_1))], \qquad (32)$$

and similarly for τ_2^*. Our approach for one-dimensional parameter spaces assumes the mean score varies smoothly with τ. We do the following: 1) run the algorithm on each data set with τ ranging over a fixed set of evenly spaced values, 2) compute the mean score, over all data sets, for each τ value, 3) choose the τ value which maximizes the mean score as optimal, 4) using these optimal values, compute the probabilities for one algorithm to be better than another, as previously described. In higher dimensional parameter spaces the optimization can become much more difficult. In this case adaptive algorithms, which attempt to sequentially select promising new τ values for testing based on the mean scores of previous τ values, may be of use.

As a final note, we could have defined other statistics equivalent to (20), such as

$$\tilde{M}2 = [2 + Score(A1(\tilde{D}, \tau_1^*))]/[2 + Score(A2(\tilde{D}, \tau_2^*))], \qquad (33)$$

where the constant 2 an arbitrary constant > 1 to keep the denominator nonzero and $Score$ is assumed nonnegative. We would then define $A1$ to be better than $A2$ iff $E[\tilde{M}2] < 1$. Even though the total probability mass below and the origin is identical for both statistics, estimates of this probability can differ significantly depending on which statistic we use. (Note that the expected values are generally not equivalent for these different statistics, though the probability mass below the origin is.) One can also think of this as a nonlinear transformation of the data or as a redefinition of the $Score$ function. Generally, the closer the statistic is to being normally distributed the better the estimate will be for a given sample size. See [Bar97] for more on statistical algorithm comparisons.

C Mathematica Code for Some of the Graphs

In this appendix we give the Mathematica [Wol91] code which was used to generate the graphs in Section 5. First we set some options and define some constants.

```
(
SetOptions[Plot, Frame->True];
$DefaultFont = {"Helvetica",4};
dash1 = { Dashing[{0.005, 0.005}] };
dash2 = { Dashing[{0.01, 0.01}] };
dash3 = { Dashing[{0.01, 0.01, 0.001, 0.01}] };
)
```

Next, we define some functions. These include the Gaussian density function and the cosine terrain density function.

```
(
normalize[fun_,var_] :=
      fun / (Integrate[fun, {var,-4,4}] /. var->Unique[var]) ;
normalizeN[fun_,var_] :=
      fun / (NIntegrate[fun, {var,-4,4}] /. var->Unique[var]) ;
SetAttributes[normalize,HoldRest];
SetAttributes[normalizeN,HoldRest];
gauss[x_] := 1/Sqrt[2 Pi] Exp[ -x^2 /2] ;
gaussUniConst=NIntegrate[gauss[x], {x,-4,4}] ;
Print["Defining terrain functions"];
terrNoNorm[x_, lambda_, delta_] := 1 + Cos[2 Pi x / lambda + delta] ;
terrNorm[x_, lambda_, delta_] = normalize[terrNoNorm[x,lambda,delta],x] ;
Print["Defining gauss-terrain functions"];
gaussTerrNoNorm[x_,lambda_,delta_]:= gauss[x] terrNoNorm[x,lambda,delta];
gaussTerrNorm[x_,lambda_,delta_]=
                    normalizeN[gaussTerrNoNorm[x,lambda,delta],x];
gaussTerrNormConst[lambda_,delta_]=NIntegrate[
                     Evaluate[gauss[x] terrNorm[x,lambda,delta]], {x,-4,4}];
)
```

We now make plots in Figures 3-6 and write them to Postscript files.

```
(
Print["Doing the plots now....."];
figure=Plot[Evaluate[{terrNorm[x,1,0], gauss[x]}], {x,-4,4},
       PlotStyle -> { dash1, {Thickness[0.004]} },
       FrameLabel -> {"target location","probability density"}] ;
PSTeX[figure, "plots/gaussNormAndTerrNormPhase0"];
figure=Plot[Evaluate[gaussTerrNorm[x,1,0]], {x,-4,4},
       FrameLabel -> {"target location","probability density"}] ;
PSTeX[figure, "plots/gaussTimesTerrNormPhase0"];
figure=Plot[Evaluate[{terrNorm[x,1,Pi], gauss[x]}], {x,-4,4},
       PlotStyle -> { dash1, {Thickness[0.004]} },
       FrameLabel -> {"target location","probability density"}] ;
PSTeX[figure, "plots/gaussNormAndTerrNormPhasePi"];
figure=Plot[Evaluate[gaussTerrNorm[x,1,Pi]], {x,-4,4},
       FrameLabel -> {"target location","probability density"}] ;
PSTeX[figure, "plots/gaussTimesTerrNormPhasePi"];
)
```

Now we generate Figure 7.

```
(
figure=Plot3D[Evaluate[gaussTerrNormConst[lambda,delta]-gaussUniConst/8],
          {lambda, 1, 15}, {delta,0,Pi},
          AxesLabel -> {"lambda", "delta","integrated\nproduct"},
          DefaultFont -> {"Helvetica",2},
          Ticks -> {Automatic, {0,Pi/4,Pi/2,3Pi/4,Pi}, Automatic},
          FaceGrids -> All,
          PlotPoints -> 40 ];
PSTeX[figure, "plots/intOfGaussCosPlot"];
)
```

Now Figure 8.

```
(
figure=Plot[Evaluate[{gaussTerrNormConst[lambda,0   ] - gaussUniConst/8,
                      gaussTerrNormConst[lambda,Pi/2] - gaussUniConst/8,
                      gaussTerrNormConst[lambda,Pi  ] - gaussUniConst/8}],
          {lambda, 1, 15},
          PlotRange -> {-0.1, 0.12},
          PlotStyle -> { { }, dash2, dash3 },
          FrameLabel -> {"lambda","integrated product"}];
PSTeX[figure, "plots/intOfGaussCosPlotOneD"]
)
```

Finally, we generate Figure 9.

```
(
empty = {  };
Print["making plots"];
figure=Plot[Evaluate[{gauss[x]/gaussUniConst,
                      gaussTerrNorm[x,1,0],
                      gaussTerrNorm[x,3,0],
                      gaussTerrNorm[x,6.5,0]}],
          {x, -4, 4},
          PlotStyle -> { dash2, empty, empty, empty },
          FrameLabel -> {"target location","probability density"}] ;
PSTeX[figure, "plots/convergingDensitiesPlot"]
)
```