Genetic Optimization of the Parameters of a Track-While-Detect Algorithm

David J. Montana

Bolt Beranek and Newman Inc.
10 Moulton Street, Cambridge, MA 02138

ABSTRACT

We have developed an algorithm to detect the presence of narrowband signals and track the time evolution of their center frequencies. This algorithm has 35 parameters whose optimal values depend on (among other things): (1) the expected dynamics of the signals, (ii) the background statistics, and (iii) the clutter (i.e., the number of simultaneous signals). Manually optimizing these parameters is a difficult task not only because of the large number of parameters but also because of the interdependence of their effects on performance. We have therefore devised an automated method for optimizing the parameters. It has three basic components: (i) a "truth" database with a graphical interface for easy manual entry of "truth", (ii) a scoring function which is a linear combination of six subscores (three evaluating detection performance and three evaluating tracking performance), and (iii) a distributed genetic algorithm which optimizes the parameter values for a particular truth database. We have used this procedure to optimize the parameter values to a variety of signal types and environmental conditions. The results have been improved performance as well as the ability to make the algorithm adaptive: as the system detects changes in the environmental conditions, it can switch to a different set of parameters.

1. INTRODUCTION

1.1 Detection and tracking of narrowband signals

A spectogram is a two-dimensional image with one axis corresponding to frequency and the other to time. Narrowband signals appear in a spectogram as curves whose widths are often not more than a few pixels. Detecting the existence of narrowband signals and tracking how their center frequencies evolve in time is an important problem in many fields including sonar, radar, geophysics and astrophysics.

A variety of algorithms have been developed for this problem, and there are different ways of classifying these algorithms. One distinction is between detect-before-track algorithms and track-while-detect algorithms. The detect-before-track algorithms separate the problems of detection and tracking, detecting signals in individual time frames and then stringing together these individual detections to form tracks[1,2]. The track-while-detect algorithms intertwine the problems of detection and tracking, forming trial tracks in every place where a signal could possibly be and making detection decisions based on the cumulative evidence from an entire track[3,4,5]. An advantage of the track-while-detect algorithms is that they can detect and track signals of signal-to-noise ratio (SNR) too low to be detected in an individual time frame and hence to be detected by the detect-before-track algorithms.

A second distinction is how restrictive are the assumptions underlying the algorithm. Two special-purpose algorithms are the following: (i) a Generalized Hough transform approach which assumes that all the tracks are from some well-characterized low-dimensional set[3] and (ii) a dynamic programming approach which assumes that all the tracks

are nearly straight lines[4]. The MAPLE (Maximum A Posteriori Line Extractor) algorithm[5] also makes restrictive assumptions about the data; after all its assumptions there are only two free parameters to fit to the data, an a priori estimate of the noise variance and an a priori estimate of the signal track variance. (However, MAPLE is not a special-purpose algorithm insofar as it may still work effectively if the assumptions are not met.) At the other extreme, a Hidden Markov model (HMM) approach to tracking[2] makes few assumptions about the data and hence in theory can work well on a greater range of data than more restrictive approaches. The consequence of this extra flexibility is the need to tune a large number of system parameters (in particular, the transition probabilities of the HMM) to the data of interest. To effectively tune these parameters requires both a large database of signals and a procedure for optimizing the parameters to the data. For the HMM approach, the Baum-Welch algorithm is this procedure. Below, we will discuss a more general approach to optimizing the parameters of a detection and tracking algorithm. However, we first describe the particular track-while-detect algorithm to which we have applied this procedure.

1.2 Our track-while-detect algorithm

We have developed an algorithm for detection and tracking which is both effective and computationally inexpensive[6]. Its basic functionality follows the outline for a tracker described by Blackman[7]. While we choose not to discuss the details of the algorithm in this paper, we will give an overview of how it operates. It has two components, a main component and a postprocessor. The main component has four basic functions, which using the terminology of Blackman[7] are: (1) track initiation: decide where to start new tracks (in bins (i) with SNR high enough above the noise background and (ii) not too close to other tracks); (2) track update: decide how existing tracks proceed (find the bin which is most likely a continuation of the current track and update the track state accordingly; this update includes smoothing using a filter with situation-dependent parameters); (3) track confirmation: decide which sections of which tracks to consider to be true signals rather than noise (based on the criteria of track length, average SNR, and other characteristics of the track); and (4) track deletion: decide where the tracks end (because either (i) the underlying signal has disappeared, (ii) the track has "lost" the actual signal, or (iii) two tracks are tracking the same signal). The postprocessor has two functions: (1) joining multiple tracks from the main component which correspond to one signal into a single track, and (2) stricter validation of the tracks after joining.

Each of these functions has its own set of associated parameters, and hence the algorithm has a large number of parameters. Originally, some of these parameters were chosen based on assumptions about the signals and the noise (e.g., uncorrelated, Gaussian noise), while the rest were chosen to fit the data by people familiar with the algorithm. However, neither of these approaches to parameter selection have proven effective. The analytic approach has been unsuccessful because its assumptions are invalid. The manual approach is very difficult because the effects of the parameters are greatly intertwined and the combined effect of changing the values of multiple parameters is unforeseeable even to people experienced with the algorithm. Plus, evaluating a change of parameter values requires running the algorithm with these parameter values on a large amount of data and judging the overall results, a process which is both time-consuming and subjective. Therefore, we have investigated an empirical approach to parameter selection, one which allows the parameters to be determined by (i.e., to adapt to) the data.

1.3 Automatic parameter selection

The rest of the paper examines our approach to selection of parameter values fit to the data. This approach has three basic components: (1) the truth tracks database, (2) the scoring function, and (3) the genetic optimizer. The truth tracks database (described in Section 2) is essentially a collection of data along with a characterization of the ideal performance for a detection and tracking performance on that data. Such a collection of training data is essential for any algorithm with data-dependent parameters. To facilitate the collection of sufficient quantities of data, we have developed a graphical user interface to allow sonar analysts to easily enter "truth".

The scoring function (described in Section 3) assigns a numerical score to a detection and tracking algorithm based on its performance on the current truth tracks database. The score is a linear combination of six subscores corresponding to different ways of deviating from the ideal. (Three subscores evaluate detection performance and three evaluate tracking performance.) The developer selects the weights on these subscores based on the relative importance of

each performance criterion to overall system performance.

The genetic optimizer (described in Section 4) finds a set of parameter values which provide nearly optimal performance with respect to the scoring function. We have used a genetic algorithm because of its ability to efficiently search large and complex spaces to find nearly global optima. A particularly interesting characteristic of our genetic algorithm is how the computation is distributed across multiple machines to speed up the optimization process. Selection of an appropriate generation size for the genetic algorithm is critical to the appropriate utilization of the distributed resources.

The results (described in Section 5) confirm that this is an effective method for selecting parameters. The process has derived parameter values better in all respects than those hand-tuned on the same data. Furthermore, we have found parameter sets tuned to different environmental conditions and different system goals, thus allowing adaptation to both external and internal stimuli.

2. THE TRUTH TRACKS DATABASE

2.1 The database structure

The truth tracks database is the training data to which we tune the parameters of the algorithm. Hence, it must contain all the information needed to score the algorithm's performance. The primary information is the spectograms along with the "true" frequency trajectory of each signal in the spectograms. However, additional information is needed to allow the scoring function to know which signals and pieces of signals are particularly important, so it can assess a greater penalty for mistakes in these areas.

The database structure has three levels: the spectogram level, the track level, and the pixel level. The database consists of a list of truthed spectogram regions. Each truthed spectogram region consists of the pathname of a spectogram; a minimum and maximum time and a minimum and maximum frequency, which define a rectangle of interest within this spectogram; and a list of truth tracks in that region of the spectogram. Each truth track consists of a start time; a length; a characterization of the signal; an array of frequencies, one for each frame of the track; and a list of the events for that track and their times. The track characterization and the list of events is what allows the scoring functions to assign importance.

2.2 The truth tracks editor

The truth tracks database should also have the following two properties. First, it should contain enough data to determine the parameter values with good statistical significance. Since our algorithm has many (35) parameters, we need a relatively large amount of data. (It is tough to determine exactly how much data is enough because, as explained in Section 5, some parameters are used less often than others and thus require more data.) Second, the database should be representative of the full range of data that the algorithm will encounter. Hence, it should contain examples of each type of signal it will encounter under each type of environmental conditions. (As described in Section 5, we sometimes split the database into multiple subdatabases, one for each category of environmental conditions. However, the full database still needs to be fully representative.)

Getting sufficient quantity and variety of truth data involves two tasks: (1) obtaining spectograms and (2) assigning truth. Obtaining spectograms involves mastering the logistics of data collection and distribution, a problem of no technical interest. However, there are some technical issues in truth assignment. In real sonar data (unlike in simulated data), there is no a priori knowledge of the underlying truth. The best approximation to truth is the opinion of an expert sonar analyst. To get the information from the sonar analyst into the database requires some sort of user interface. We have developed a tool called the truth tracks editor to provide this interface. (This is analogous to the windowing interface for rule threshold optimization described in an earlier paper[8].)

The truth tracks editor is a graphical, menu-driven tool for entering and editing data in the truth tracks database.

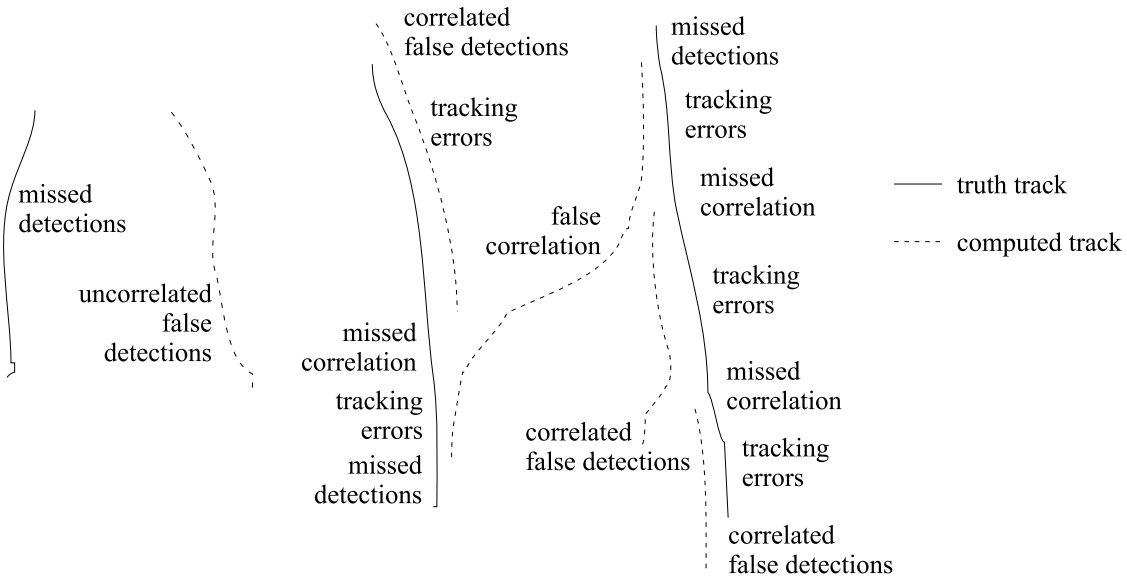


Figure 1: Examples of contributions to the different performance measures.

It is designed to make the entry of large amounts of truth data relatively easy and hence to ensure sufficient quantity and quality of such data. It operates at each of the three levels of the truth tracks database. At the spectogram level, it allows (i) creation and deletion of truthed spectogram regions and (ii) changing of the boundaries of the rectangle of a truthed spectogram region. At the track level, it allows (i) creation and deletion of truth tracks, (ii) joining of two truth tracks into one, (iii) cutting of one truth track into two, and (iv) selection of characterizations for a truth track. At the pixel level, it allows (i) zooming and panning (to allow better views of the individual pixels), (ii) creation and deletion of events at points of the track, (iii) selection of the track frequency for a given time, and (iv) interpolation between selected points on the track.

3. THE SCORING FUNCTION

To evaluate a particular detection and tracking algorithm, the scoring function first runs the algorithm on the spectogram regions in the truth tracks database, producing output we refer to as "computed tracks". It then computes a single score based on the differences between the computed tracks and the truth tracks.

The score is computed in two stages. The first stage is to compute a correspondence map between pixels of truth tracks and pixels of computed tracks. This map is defined as follows. A pixel of a truth track corresponds to a pixel of a computed track if and only if (1) the computed track pixel is the closest such pixel to the truth track pixel, (2) the truth track pixel is the closest such pixel to the computed track pixel, and (3) the computed track is tracking the same signal as the truth track at some point. This leaves some truth track pixels unassociated with computed track pixels and vice versa. We call a (truth or computed) track unassociated if all its pixels are unassociated. We define an association transition as two associated pixels of a track (i) whose associated pixels are not from the same track and (ii) which are either contiguous or are separated only by a block of unassociated pixels.

The second stage is to calculate from the correspondence map six performance measures (analogous to $(1 - P_d)$ and P_f for a pure detection algorithm) which can then be combined using a weighted sum to give a single score (analogous to the way $(1 - P_d)$ and P_f are combined into a single score in an earlier paper on optimization of the thresholds of detection rules[8]). These measures are

- Missed detections: the sum over *unassociated pixels of all truth tracks* of the pixel's importance divided by the sum over *all pixels of all truth tracks* of the pixel's importance.

- Correlated false detections: the sum over *unassociated pixels of associated computed tracks* of the pixel's importance divided by the sum over *all pixels of all truth tracks* of the pixel's importance.
- Uncorrelated false detections: the number of *all pixels of unassociated computed tracks* divided by the number of *all pixels of all spectograms*.
- Missed correlations: the sum over *all association transitions of all truth tracks* of the maximal importance of the pixels contained in the transition divided by the sum over *all pixels of all truth tracks* of the pixel's importance.
- False correlations: the sum over *all association transitions of all computed tracks* of the maximal importance of the pixels contained in the transition divided by the sum over *all pixels of all truth tracks* of the pixel's importance.
- Tracking errors: the sum over *associated pixels of all truth tracks* of the square of the pixel's distance from its corresponding computed track pixel times its importance divided by the sum over *all pixels of all truth tracks* of the pixel's importance.

Figure 1 presents examples of contributions to each of these scores. We now provide intuitive explanations of these measures. The first three scores measure detection performance, while the last three measure tracking performance. The missed detections score is just a weighted version of $(1 - P_d)$. There are two measures of false detections, one for the false detections which are part of (i.e, correlated with) true tracks (i.e., tracks which at some point are tracking a signal) and one for the false detection which are part of false tracks. (The different normalizing factors for these two scores reflect the difference in opportunity for error.) We have divided the false detections into two scores because these two types of false detections have different system-level effects. For instance, we have used our detection and tracking algorithm as part of a larger system for signal interpretation[6,8], and correlated false detections cause many more problems for this system than uncorrelated false detections. The missed correlations score measures the rate of breaks in the tracking continuity. The false correlations score measure how often tracks jump from one signal to another. The tracking errors score measures how well the track frequencies match the "true" center frequencies.

The overall score is a weighted sum of these six subscores. The developer selects the weights before the start of an optimization run. The weights are chosen so as to achieve the desired tradeoff between performance in these six areas. The developer can also select some of the values used in determining importance and thus determine the relative importance of different signal types.

4. THE GENETIC OPTIMIZER

4.1 An overview of genetic algorithms

Genetic algorithms are algorithms for optimization and machine learning based loosely on several features of biological evolution[9]. They require five components[10]:

1. A way of encoding solutions to the problem on chromosomes.
2. An evaluation function which returns a rating for each chromosome given to it.
3. A way of initializing the population of chromosomes.
4. Operators that may be applied to parents when they reproduce to alter their genetic composition. Standard operators are mutation and crossover (see Figure 2).
5. Parameter settings for the algorithm, the operators, and so forth.

Given these five components, a genetic algorithm operates according to the following steps:

1. Initialize the population using the initialization procedure, and evaluate each member of the initial population.

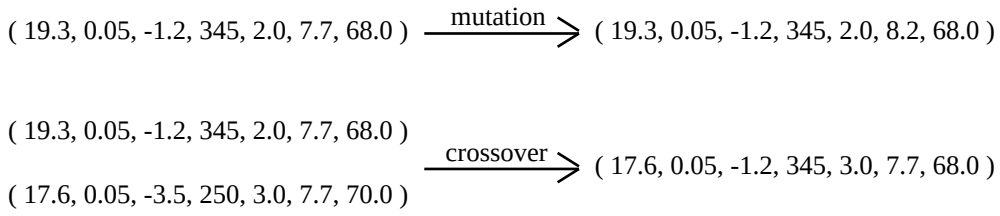


Figure 2: The mutation and crossover operators.

2. Reproduce until a stopping criterion is met. Reproduction consists of iterations of the following steps:
   (a) Choose one or more parents to reproduce. Selection is stochastic, but the individuals with the hightest evaluations are favored in the selection.
   (b) Choose a genetic operator and apply it to the parents.
   (c) Evaluate the children and accumulate them into a generation. After accumulating enough individuals, insert them into the population, replacing the worst current members of the population.

When the components of the genetic algorithm are chosen appropriately, the reproduction process will continually improve the population, converging finally on solutions close to a global optimum. Genetic algorithms can efficiently search large and complex (i.e., possessing many local optima) spaces to find nearly global optima. That is why we have used a genetic algorithm for our optimization problem.

4.2 Our genetic algorithm

The core of our genetic algorithm is a software package called OOGA (Object-Oriented Genetic Algorithm)[10]. It implements the standard operation of a genetic algorithm described above along with some non-standard features such as exponential normalization of evaluations and adaptive operator probabilities. The user selects the five components described above appropriately for the problem of interest. We have chosen the five components as follows:

1. An individual is a string of real-valued parameter values. For each parameter, we select a range and step size, thus restricting it to a finite number of possible values. We choose the ranges and step sizes based on knowledge of the role of each parameter. (Failure to choose these appropriately can cause the genetic algorithm to work slower or not at all.)

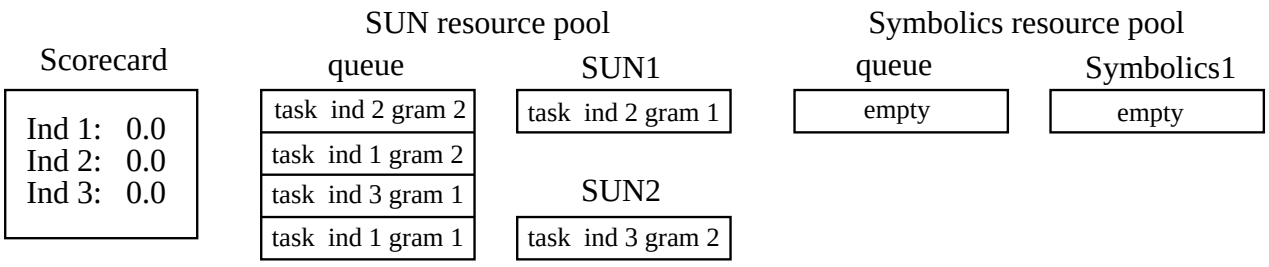
2. The evaluation function runs our track-while-detect algorithm with the parameters specified on the regions of spectograms in the truth database. It then evaluates the algorithm's performance using the scoring function described in Section 3. The weights to use in the scoring function are obtained from the user prior to the run.

3. We initialize the population with individuals chosen randomly with uniform probability across the set of possible parameter values.

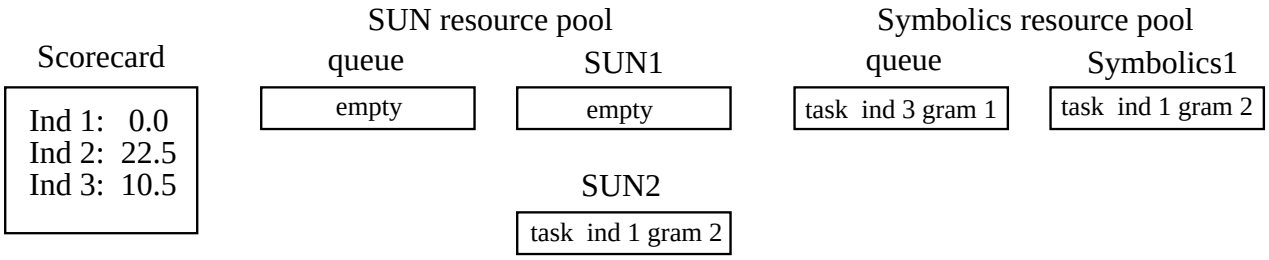
4. The genetic operators are mutation and uniform crossover (see Figure 2). Mutation creates a child by changing certain parameters of the parent to different values selected randomly from the possible values for those parameters. Uniform crossover selects for the value of each parameter of the child the value of the same parameter from one of the two parents; which parent to inherit from is randomly chosen for each parameter.

5. Some key parameters are population size and generation size, whose values are 50 and 5 respectively. A population size of 50 is standard, and Section 4.4 explains the benefit of a generation size of 5.

We now discuss some of the key issues in getting the genetic algorithm to work properly.



(a) Configuration immediately after starting evaluations



(b) Configuration at a later time

Figure 3: Procedure for distributing evaluations.

4.3 Distribution across multiple machines

An evaluation of an individual (i.e., a set of parameter values) requires running the detection and tracking algorithm on lots of data and then evaluating its performance on this data. In practice, this evaluation requires around 15 minutes. Hence, to evaluate 600 individuals (a typical number needed before convergence) would require almost a week if computed on a single machine. It was therefore imperative for us to spread the computation over multiple machines.

Because the evaluations are independent of each other, we could use distributed rather than parallel processing and avoid the costs in price and complexity of the latter. Distributing the evaluations was complicated by the fact that the detection and tracking algorithm runs on SUN computers while the scoring runs on Symbolics. However, this also provided a means to utilize both the SUN's and Symbolics on our network, thus increasing the degree of parallelism. The distribution process is illustrated in Figure 3 and discussed in more detail in an earlier paper[6].

4.4 The effect of generation size

The generation size is the number of new individuals generated (or, equivalently, the number of old individuals replaced) per generation. The most common choice is to do full generational replacement, i.e. to set the generation size equal to the population size. Syswerda[11] argues instead that it is best to use a generation size of one (thus using what he calls a steady-state genetic algorithm) for two reasons: (1) once the algorithm has evaluated a new individual and found it to be good, it will proceed faster by immediately using this individual as a possible parent rather than waiting, and (2) the algorithm can afford to be exploratory without risking loss of previously gained information. (Note that the second reason really only argues for keeping the best few individuals of each generation.) For these reasons, we have used a generation size of one in genetic algorithms which do not distribute the evaluations across multiple machines.

However, with distributed evaluations, there is an argument for a larger generation size. Before the genetic algorithm

can create the individuals of one generation, it needs to finish evaluating those of the previous generation. During this synchronization period, the algorithm cannot achieve full parallelism. Since the time needed for synchronization does not change significantly with the generation size, the larger the generation size the larger the average amount of parallelism, or equivalently the lower the average time per evaluation. The optimal choice of generation size depends on certain factors including the number of processing units available and the amount of parallelism achievable within an individual evaluation. We have found five to be a good generation size for our problem and computing resources.

Note that the optimal solution (which we have yet to implement) to the tradeoff between using good individuals in the reproduction process as soon as possible and obtaining maximal parallelism is to do away with the concept of generations. Instead, let the reproduction and evaluation processes be asynchronous, that is (1) create a new individual for evaluation as soon as there is a free resource to start the evaluation process and (2) as soon as an individual is finished being evaluated, insert it into the population, eliminating the worst population member to make room. In this case, computing resources are always fully utilized while inidividuals can reproduce as soon as they are evaluated.

5. RESULTS

We have experimented with this automated method of parameter selection on sonar data. Preliminary results have shown this method to be superior to (i.e., to select parameter values which according to the opinion of expert analysts yield detection and tracking performance better than) either the manual approach (i.e., having a human hand-select the parameters) or the analytic approach (i.e., deriving the parameters based on assumptions about the data). However, the success of our approach is dependent on appropriate selection of the weights for the scoring function and the creation of a truth tracks database of sufficient quantity and variety.

Failure to choose appropriate weights for the scoring function causes the score assigned by this function to not accurately represent the performance of the algorithm and hence causes the selection of bad parameter values. One flagrant example of this occurred before we introduced into the scoring function the potential for different importances and therefore considered all signals equally important. In most sonar data, the uninteresting signals greatly outnumber the interesting signals. Without explicit importance weightings indicating otherwise, the numerical predominance of uninteresting signals caused the parameter values to be tuned for good performance on these signals at the expense of poor performance on the interesting signals. This caused the expert analysts to disapprove of the parameter values selected and led us to introduce the concept of importance.

A poor selection of parameter values can also occur due to problems with the truth tracks database. As mentioned in Section 2, an insufficient quantity of data can lead to lack of statistical significance in the choice of parameter values and hence to poor generalization (i.e., performance on data other than that on which the parameter values were tuned). This problem manifest itself in two subtle ways in our experiments. First, some parameters of the algorithm are used much more often than others. While the often-used parameters need relatively small amounts of data to ensure good generalization, the lesser-used parameters need much more data for good generalization. So, when we did have problems with the generalization of a set of parameter values, it could usually be traced to problems with the lesser-used parameters. Second, some spectograms have more signals than other spectograms. For the purposes of five of the six subscores (described in Section 3), the number of signals and their duration are the real measures of how much data is in the truth database; only for the uncorrelated false alarms subscore is the amount of data measured by the number of spectograms and their size. This caused problems with the generalization for the sets of parameter values tuned to low-clutter data (i.e., data with few simultaneous signals).

The genetic optimizer proved very reliable in its ability to find nearly global optima in a relatively short time. An average run required between 600 and 800 evaluations to converge. Since there are 35 parameters, this corresponds to only about 20 evaluations per real-valued dimension. Plus, multiple runs of the genetic algorithm on the same data with the same scoring weights would usually produce best sets of parameter values with similar scores. This is a good indication that the genetic algorithm was finding nearly global optima.

Another advantage of our automated approach to parameter selection is the ability to make the algorithm adaptive in real time. Indeed, we divided the training database into two subdatabases, one containing high-clutter data (i.e., data with many simultaneous signals) and the other containing low-clutter data (i.e., data with few simultaneous signals), and we performed automated parameter selection for each database, thus creating two distinct sets of parameter values. We also developed a statistical measure of the amount of clutter in an acoustic scene. Based on the value of this measure, we had the algorithm in real time switch between the low-clutter parameter values and high-clutter parameter values as appropriate.

6. CONCLUSION

Detection and tracking algorithms can in theory achieve good performance over wider ranges of signals and environmental conditions by dropping restrictive assumptions and instead adapting to the data. We have presented a general method for performing this adaptation when the degrees of freedom of the algorithm are represented as parameters. This procedure has three main components: (1) the truth tracks database, which gives the ideal performance of a detection and tracking algorithm on some selected data, (2) the scoring function, which assigns a numerical score evaluating the performance of an algorithm on the truth tracks database, and (3) the genetic optimizer, which selects a set of parameter values which yield a nearly optimal score. Experiments with this optimization procedure have shown the following: if the truth tracks database has sufficient quantity and variety of data and the scoring function has weights which reflect the preferences of expert analysts, then the procedure can produce nearly optimal sets of parameter values. Plus, by segmenting the data according to environmental conditions and finding parameter values optimized to these different conditions, we can create an algorithm adaptive in real-time to the environment.

7. ACKNOWLEDGMENTS

The work described in this paper was funded by SPAWAR under contract N00039-89-C-0300.

Thanks are due to the following people: Steve Milligan, for suggesting this problem; Ron Scott, for his useful discussions; Dave Davis, for the use of his OOGA code; Jeff Morrill, for writing a robust scheduler for distributing the computation; and Roy Westerberg, for creating most of the truth database.

8. REFERENCES

1. Y. Bar-Shalom and T. E. Fortmann, *Tracking and Data Association,* Academic, New York, 1988.

2. X. Xie and R. J. Evans, "Multiple Target Tracking and Multiple Frequency Line Tracking Using Hidden Markov Models," *IEEE Trans. Acoustics, Speech, Signal Proc.,* vol. 39, no. 12, pp. 2659–2676, 1991.

3. D. J. Montana, "Genetic Search of a Generalized Hough Transform Space," *in these proceedings.*

4. Y. Barniv, "Dynamic Programming Solution for Detecting Dim Moving Targets," *IEEE Trans. Aerospace and Electrical Systems,* vol. 21, no. 1, pp. 144–156, 1985.

5. J. J. Wolcin, "Maximum A Posteriori Estimation of Narrow-Band Signal Parameters," *Journal of the Acoustical Society of America,* vol. 68, no. 1, pp. 174–178, 1980.

6. D. J. Montana, "Automated Parameter Tuning for Interpretation of Synthetic Images," in [10], pp. 282-311.

7. S. S. Blackman, *Multiple Target Tracking with Radar Applications,* Artech House, Norwood, MA, 1986.

8. D. J. Montana, "Empirical Learning Using Rule Threshold Optimization for Detection of Events in Synthetic Images," *Machine Learning*, vol. 5, pp. 427–450, 1990.

9. D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison-Wesley, Redwood City, CA, 1988.

10. L. Davis, *Handbook of Genetic Algorithms,* Von Nostrand Reinhold, New York, 1991.

11. G. Syswerda, "Uniform Crossover in Genetic Algorithms," *Proc. Third International Conference on Genetic Algorithms,* pp. 2–9, 1989.