Algorithms for Multiple-Target Tracking

Tracking many moving objects at once is a task whose difficulty grows out of all proportion to the number of objects. A new method works for thousands of targets

Jeffrey K. Uhlmann

When a major-league outfielder runs down a long fly ball, the tracking of a moving object looks easy. Over a distance of a few hundred feet, the fielder calculates the ball's trajectory to within an inch or two, and times its fall to within milliseconds. But what if an outfielder were asked to track 100 fly balls at once? As it turns out, even 100 fielders trying to track 100 balls simultaneously would likely find the task an impossible challenge.

Problems of this kind do not arise in baseball, but they have considerable practical importance in other realms. The impetus for the studies described in this article was the Strategic Defense Initiative, the plan conceived a decade ago for defending the U.S. against a large-scale nuclear attack. According to the terms of the original proposal, an SDI system would be required to track tens or even hundreds of thousands of objects, including missiles, warheads, decoys and debris, all of them moving at speeds of up to eight kilometers per second. Another application of multiple-target tracking is to air-traffic control, where the goal is to maintain a safe separation between the hundreds of aircraft that might be operating near a busy airport. In particle physics, multiple-target tracking is needed to make sense of the hundreds or thousands of particle tracks emanating from the site of a high-energy collision. There is a similar requirement in studies of molecular dynamics.

The task of following a large number of targets turns out to be surprisingly hard. If tracking a single baseball or warhead or aircraft requires a certain measurable level of effort, then it might seem that tracking 10 similar objects would require at most 10 times as much effort. Actually, for the most obvious methods of solving the problem, the difficulty is proportional to the square of the number of objects; thus 10 objects demand 100 times the effort, and 10,000 objects increase the difficulty by a factor of 100 million. This combinatorial explosion is the crux of the multiple-target-tracking problem.

Consider how you might go about following the motion of a single object. You receive a series of position reports from a sensor of some kind, such as a radar system. To reconstruct the object's trajectory, you plot the successive positions in sequence and then draw a line through them. Extending this line yields a prediction of the object's future position. Now suppose you are tracking 10 targets simultaneously. At intervals you receive 10 new position reports—but the reports do not come with labels that would tell you which targets they represent. On the contrary, when you plot the 10 new positions, each report could in principle be associated with any of the 10 existing trajectories. It is this need to consider every possible combination of reports and tracks that makes the difficulty of an n-target problem proportional to n^2.

Over the years there have been many attempts to devise an algorithm for multiple-target tracking with better than n^2 performance. Some of the proposals offered significant improvements in special circumstances or for certain instances of the multiple-target-tracking problem, but they retained their n^2 worst-case behavior. Now my colleagues and I at the Naval Research Laboratory in Washington have developed a new class of algorithms for the crucial step of associating reports with tracks; if a set of objects can be tracked at all, these algorithms can be guaranteed to do it in less than n^2 steps. Practical applications are under way. Even with the new methods, multiple-target tracking remains a complex task, which strains the capacity of the largest and fastest supercomputers, but important problem instances are now within reach.

Keeping Track

The modern need for tracking algorithms began with the development of radar during World War II. By the 1950s radar was a relatively mature technology. Systems were installed aboard military ships and aircraft, and at airports. The tracking of radar targets, however, was still done by manually drawing lines through blips on a display screen. The first attempts to automate the tracking process were modeled closely on human performance. For the single-target case the resulting algorithm was straightforward: The computer accumulated a series of positions from radar reports, and estimated the velocity of the target in order to predict its future position.

Even single-target tracking presented certain challenges connected with the uncertainty inherent in all position measurements. A first problem is de-

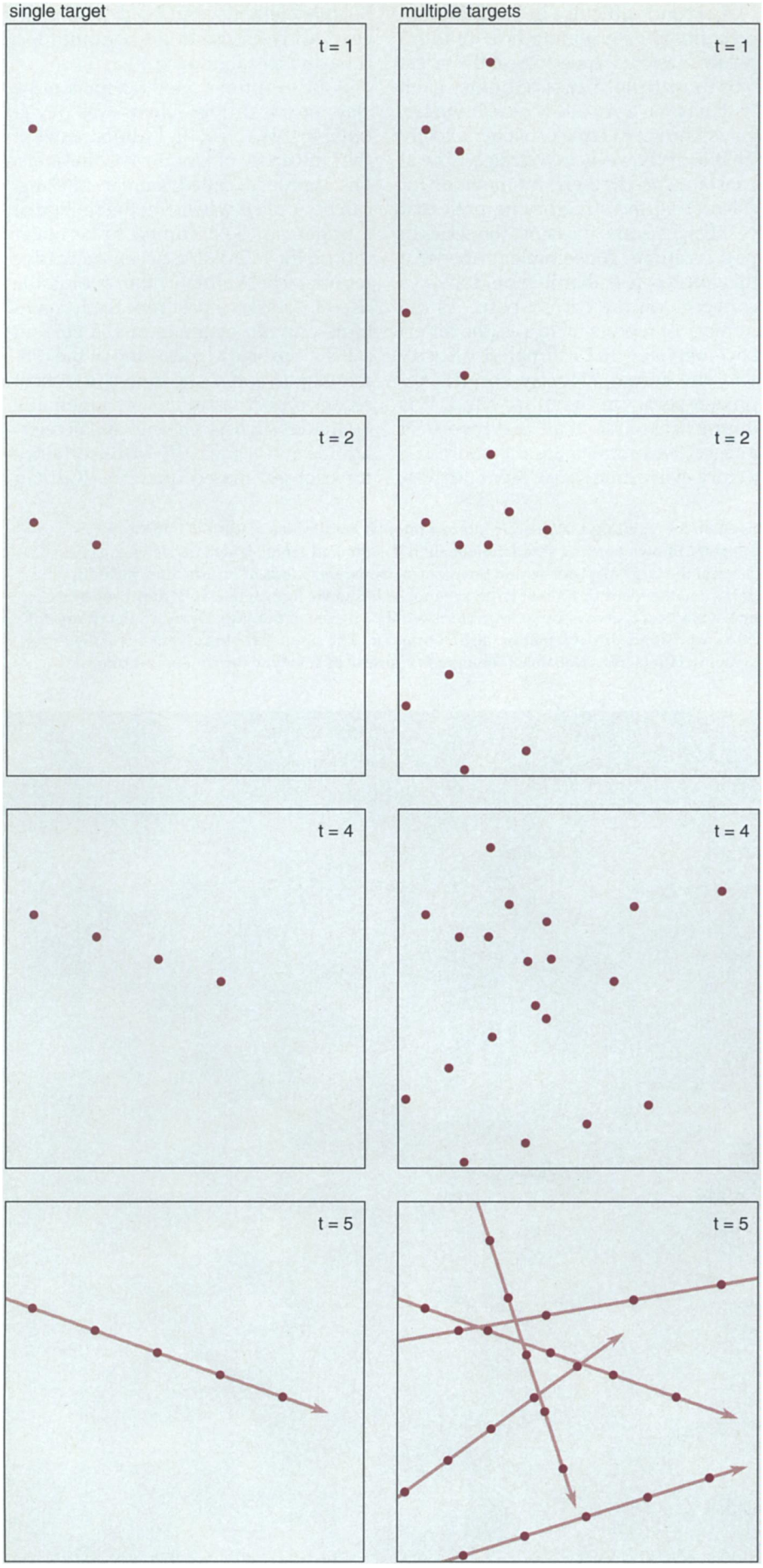
Jeffrey K. Uhlmann is at the Naval Research Laboratory in Washington. He performed both his undergraduate and his graduate work at the University of Missouri-Columbia, where his areas of emphasis included philosophy, computer science and discrete mathematics. His present research interest is in the area of multidimensional search and correlation. Since 1987 he has collaborated with other investigators at NRL on applications of computationally efficient data structures to problems in multiple-target tracking. In addition to journal publications, this work has resulted in five patents and patents pending. His hobbies include writing for film and television, and music composition. A sample of his musical work will be released in September on Rockit Records. Address: Code 5570, Naval Research Laboratory, Washington, DC 20375-5000. Electronic mail: uhlmann@excalibur.itd.nrl.navy.mil.

ciding how to represent this uncertainty. A crude approach is to define an error radius surrounding the position estimate. This practice implies that the probability of finding the target is uniformly distributed throughout the volume of a three-dimensional sphere. Unfortunately, this simple approach is far from optimal. The error region associated with many sensors is highly nonspherical; radar, for example, tends to provide accurate range information but has poorer radial resolution. Furthermore, one would expect the actual position of the target to be closer on average to the mean position estimate than to the perimeter of the error volume, which suggests in turn that the probability density should be greater near the center.

A second difficulty in handling uncertainty is determining how to interpolate the actual trajectory of the target from multiple measurements, each with its own error allowance. If the target is known to have constant velocity (that is, if it travels in a straight line at constant speed), there are methods for calculating the straight-line path that best fits, by some measure, the series of past positions. A desirable property of this approach is that it should always converge on the correct path: As the number of reports increases, the difference between the estimated velocity and the actual velocity should approach zero. On the other hand, it is impractical to retain all past reports of a target and recalculate the entire trajectory every time a new report arrives. Such a method would eventually exceed all constraints on computation time and storage space.

A near-optimal method for a large class of tracking problems was developed in 1960 by R. E. Kalman, now of the University of Florida at Gainesville. His approach, called Kalman filtering, can be applied whenever the motion of a target can be assumed to be linear during the interval between successive reports. The Kalman filter avoids the need to preserve past reports. It maintains only an estimate of the current state of the object, consisting of the estimated position at the time of the most recent report, estimates of kinematic quantities such as velocity and acceleration, and a measure of uncertainty for each estimated quantity. Kalman

Figure 1. Strategic Defense Initiative, the proposed system for repelling a large-scale intercontinental missile attack, was the impetus for extensive investigation of the automated tracking of multiple targets. Shown here is a simulation of an SDI command center, where the defense against an attack would be coordinated. The individual consoles and the large displays behind them can show various kinds of information, including the tracks of incoming missiles. The plotting of these tracks and the identification of targets cannot be left to the human operators, however, because the system might have to cope with tens of thousands of warheads, decoys and other objects within a period of minutes. Devising algorithms for tracking vast numbers of targets has turned out to be a surprisingly difficult part of the SDI program. The simulated control center is at the Army Strategic Defense Command Advanced Research Center in Huntsville, Alabama. (Photograph courtesy of Teledyne Brown Engineering.)



showed that if a few basic conditions are satisfied, the state estimate can be updated with each incoming report in such a way that the calculated trajectory of the target converges to the actual trajectory.

A particularly important feature of the Kalman filter is its use of a Gaussian probability distribution to model uncertainty. The Gaussian distribution has a single point of maximum probability, called its mean; moving away from the mean, probability diminishes exponentially (although it never falls to zero). Thus the distribution can be imagined as a cloud, densest at the center and wispy at the edges. A graph of the probability density along any axis passing through the mean is a symmetrical bell curve. But the bell curves along different axes can have different widths, so that the cloud does not have to be spherical. The cloud is described as an error ellipse (or ellipsoid in three dimensions) because contours of equal probability form an ellipse or ellipsoid. Hence a sensor report is simply a mean position with an associated error ellipse, and the projection of a track to a new position can also be represented by a mean point and an ellipse.

Kalman's work revolutionized the field of target tracking. By the mid-1960s, Kalman filtering was a standard methodology. It is just as central to multiple-target tracking as it is to single-target tracking.

Figure 2. Correlation problem is at the heart of what makes multiple-target tracking a difficult task. The information available for plotting a track consists of position reports (shown here as colored dots) from a sensor such as a radar system. In tracking a single target *(left)*, one can accumulate a series of reports and then fit a line or curve to these data points in order to estimate the object's trajectory. With multiple targets *(right)* there is no obvious way to determine which object has generated which report. Here five reports appear initially, at $t = 1$; then five more reports are received at $t = 2$. But it is not immediately apparent to the eye which of the later dots goes with which of the earlier ones, and a computer program would find the correlation task just as difficult. (In fact, there need not *be* a correlation; the five reports at $t = 2$ could represent five new targets.) As further reports arrive, coherent tracks do begin to emerge. The tracks from which these reports were derived are shown in the final panels, at $t = 5$. Here and in subsequent illustrations all targets move in a straight line at uniform speed and are confined to two dimensions; the more realistic task of following curved paths in three-dimensional space is appreciably more difficult.

Nearest Neighbors

What multiple targets add to the tracking problem is the need to assign each incoming position report to a specific target track. The earliest mechanism for classifying reports was the nearest-neighbor rule. The idea of the rule is to estimate each object's position at the time of a new position report, and then assign the report to the nearest such estimate. This intuitively plausible approach is especially attractive because it decomposes the multiple-target-tracking problem into a set of single-target problems.

Kalman's formulation provided the foundation for a statistically meaningful version of the nearest-neighbor rule. Specifically, from Kalman's theorems one can derive an equation that gives the probability that a report at time t and a track projected to time t are associated with the same object. Accordingly, the nearest-neighbor rule can be redefined to state that a report should be assigned to the track with which it has the highest probability of association. In this way a multiple-target problem can still be decomposed into a set of single-target problems.

The nearest-neighbor rule has strong intuitive appeal, but doubts and difficulties connected with it soon began to emerge. For example, early implementers of the method discovered problems in creating initial tracks for multiple targets. In the case of a single target, two reports can be accumulated to derive a velocity estimate, from which a track can be created. For multiple targets, however, there is no obvious way to deduce such initial velocities. The first two reports received could represent successive positions of a single object or the initial detection of two distinct objects. Every subsequent report could be the continuation of a known track or the start of a new one. To make matters worse, almost every sensor produces at least occasional spurious reports, which give rise to spurious tracks. Thus the tracking system needs an additional mechanism to recognize and delete tracks that are never assigned reports with a high probability of association.

Another difficulty with the nearest-neighbor rule becomes apparent when reports are misclassified, as will inevitably happen from time to time if the tracked objects are close together. A misassignment can cause the Kalman-filtering process to converge very slowly, or it may even fail to converge alto-

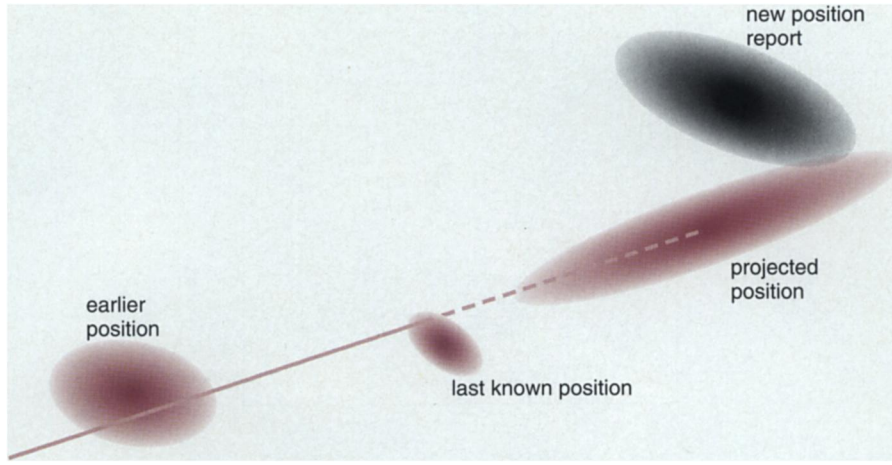


Figure 3. Uncertainty in position reports and in estimates of a target's trajectory add to the difficulty of the correlation problem in multiple-target tracking. No sensor provides an absolutely accurate or reliable measure of target motion; there is always some uncertainty in position and velocity. The uncertainty can be represented by a Gaussian distribution, which falls off smoothly with distance from the mean position. The amplitude of the Gaussian distribution at any point—indicated here by darkness of shading—gives the probability of finding the target at that point. The distribution is often nonsymmetrical, since sensors may have better resolution in some directions than in others; hence the positional uncertainty is often described in terms of an error ellipse (or, in three dimensions, an error ellipsoid). The extent of overlap between error ellipses determines the likelihood that a track and a position report are correlated.

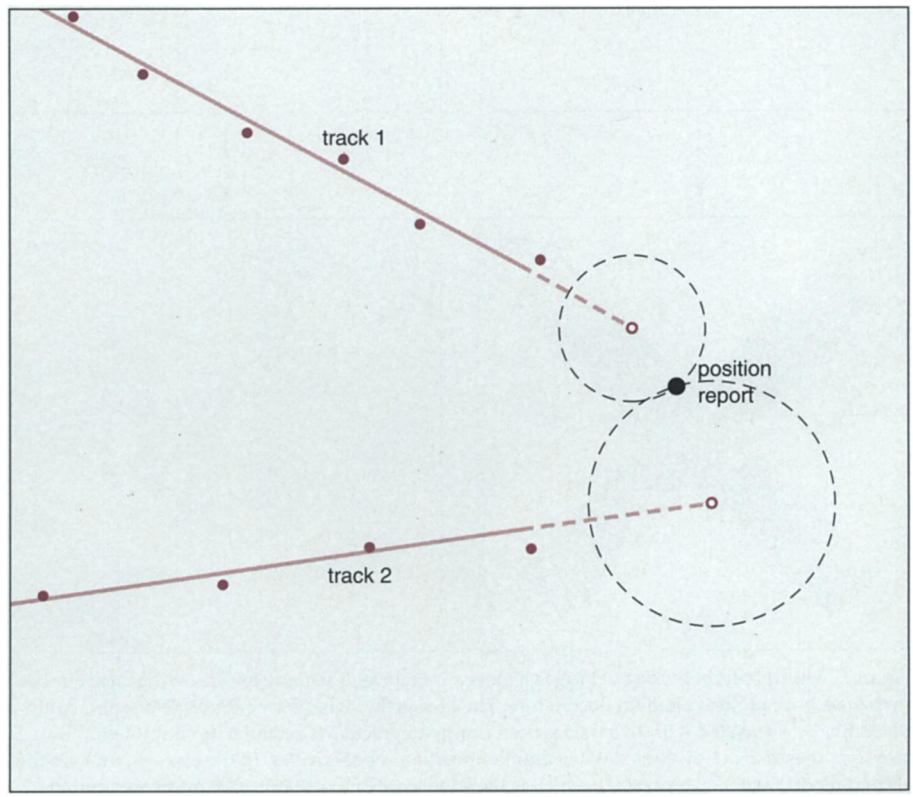


Figure 4. Nearest-neighbor rule is perhaps the simplest approach to the correlation problem. When a new position report arrives, all existing tracks are projected forward to the time of the new measurement. (In this diagram, earlier target positions are indicated by colored dots and the projected positions by colored circles; the new position report is a black dot.) Then the distance from the report to each projected position is calculated, and the report is associated with the nearest track. In the situation shown, the report would be assigned to track 1. The main drawback of this scheme is that if the assignment is erroneous, subsequent analysis is severely disrupted.

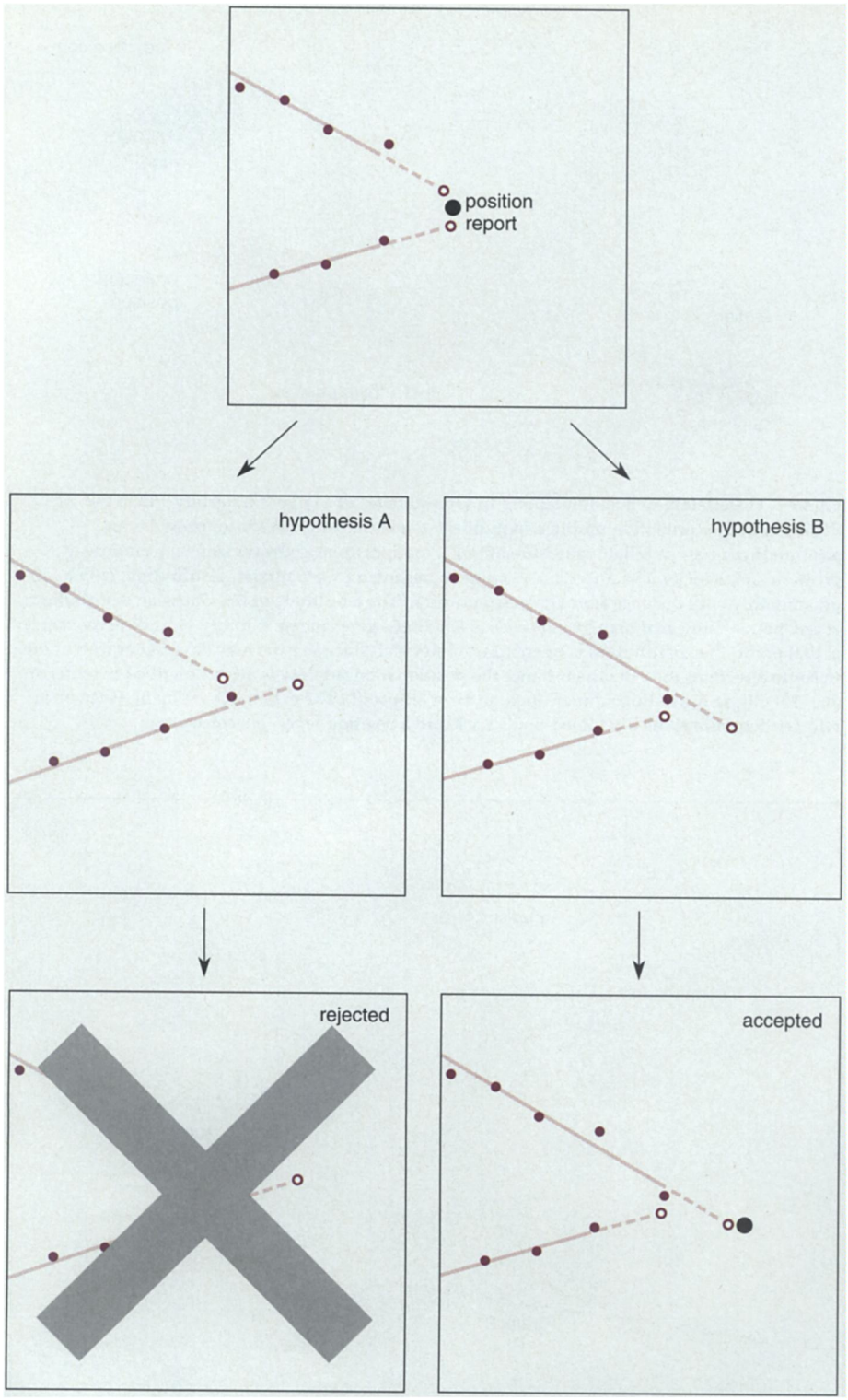


Figure 5. Multiple-hypothesis tracking is a more sophisticated scheme for following target tracks in the presence of measurement uncertainty. The idea is that whenever a report *(black dot)* could plausibly be associated with two tracks, both candidate tracks are retained by the system. The two hypotheses are re-evaluated when another position report arrives. In the case shown here the second report favors hypothesis *B*, which is therefore confirmed. A program implementing the multiple-hypothesis algorithm requires a mechanism for identifying and deleting hypotheses that fail to be supported by subsequent measurements, such as hypothesis *A*.

gether—in which case the track cannot be predicted. Moreover, tracks updated with misassigned reports (or not updated at all) will tend to correlate poorly with subsequent reports and may therefore be mistaken as spurious by the track-deletion mechanism. Mistakenly deleted tracks then necessitate subsequent track initiations and a possible repetition of the process.

Track Splitting

A robust solution to the problem of assignment ambiguities is to create multiple "hypothesis tracks." Under this scheme, the tracking system does not have to commit itself immediately or irrevocably to a single assignment of each report. If a report is highly correlated with more than one track, an updated copy of each track can be created; subsequent reports can be used to determine which assignment is correct. As more reports come in, the track associated with the correct assignment will rapidly converge on the true target trajectory, whereas the falsely updated tracks are unlikely to be correlated with any subsequent reports.

This basic technique is called track splitting. One of its worrisome consequences is a proliferation in the number of tracks that a program must keep tabs on. The proliferation can be controlled with the same track-deletion mechanism used in the nearest-neighbor algorithm, which scans through all the tracks from time to time and eliminates those that have a low probability of association with recent reports. A more sophisticated approach to track splitting, called multiple-hypothesis tracking, maintains a history of track branchings, so that as soon as one branch is confirmed, the alternative branches can be pruned away.

Track splitting in its various forms is widely regarded as the best strategy for handling the ambiguities inherent in correlating tracks with reports from multiple targets. It is even used to minimize the effects of spurious reports when tracking a single target. Nevertheless, some serious difficulties remain. First, track splitting does not completely decompose a multiple-target tracking problem into independent single-target problems, in the way the nearest-neighbor strategy was intended to do. For example, two hypothesis tracks may lock onto the trajectory of a single object. Since both tracks are valid, the standard track-deletion mechanism cannot eliminate either of them; the deletion procedure has to be modified to detect redundant tracks, and so it cannot look at just one track at a time. This coupling between multiple tracks is theoretically troubling, but experience has shown that it can be managed in practice at low computational cost.

A second problem is the difficulty of deciding when a position report and a projected track are correlated closely enough to justify creating a new hy-

pothesis track. If the correlation threshold is set too high, correct assignments may be missed so often as to prevent convergence of the Kalman filter. If the threshold is too low, the number of hypotheses may grow exponentially. The usual practice is to set the threshold low enough to ensure convergence, and then add another mechanism to limit the rate of hypothesis generation. A simple strategy is to select the n hypothesis candidates with the highest probabilities of association, where n is the maximum number of hypotheses that computational resource constraints will allow. This "greedy" method usually yields good performance.

Even with these enhancements, the tracking algorithm makes such prodigious demands on computing resources that large problems remain beyond practical reach. Monitoring the computation to see how much time is spent in various subtasks shows that calculating probabilities of association is by far the biggest expense. The program gets bogged down projecting target tracks to the time of a position report, calculating error ellipses, and then looking for intersections of the ellipses. Since this is the critical section of the algorithm, further effort has focused on improving performance here.

Gating

The various geometric calculations involved in estimating a probability of association are numerically intensive and inherently time-consuming. Thus one approach to speeding up the tracking procedure is to streamline or fine-tune these calculations—to encode them more efficiently without changing their fundamental nature. This is what computer scientists generally have in mind when they speak of "optimizing" a program. Careful optimizing can be important, but in this case it is not enough. It is necessary to actually reduce the number of calculations being performed.

An appealing approach to reducing the number of probability calculations is to do a preliminary screening of tracks and position reports; only if a track-report pair passes a computationally inexpensive feasibility check is there any need to complete the full probability calculation. This process is called gating. Several geometric tests could serve as gating criteria. For example, if on average each track is updated every five seconds, and the targets are known to have a maximum speed of 10 kilometers per second, it is probably safe to assume that a track and report more than 50 kilometers apart are not correlated. (A larger distance may be required to take into account the uncertainty measures associated with both the tracks and the reports.)

Gating algorithms are successful in reducing the numerical overhead of the correlation process and in increasing the number of targets that can be tracked in real time. Unfortunately, the benefits of simple gating diminish as the number of targets increases. Specifically, implementers of gating algorithms have found that increasing the number of targets by a factor of 20 increases the computational burden by a factor of 100. Moreover, it turns out that the largest percentage of computation time is still spent in the correlation process, although now the bulk of the demand is for simple distance calculations within the gating algorithm. Clearly the expense of the correlation process involves more than numerical overhead; there is a combinatorial aspect as well.

A cursory examination of the correlation process reveals that even with elaborate gating techniques, every report still must be compared with every track. If there are n reports and n tracks, then n^2 comparisons are required to determine which pairs are correlated. Simple gating can reduce the average cost of each comparison, but what is really needed is a method to reduce the sheer number of comparisons. Some structure must be imposed on the set of tracks that will allow correlated track-report pairs to be identified without having to compare every report with every track. A corollary is that the long history of attempts to maintain tracks of multiple targets as if they were independent tracking problems must be abandoned.

The new correlation problem is difficult conceptually because it demands that most pairs of tracks and reports be excluded from consideration without ever examining them. At the same time, no track-report pair whose probability of association exceeds the correlation threshold can be disregarded. For some time, these constraints seemed impossible to satisfy simultaneously. Often, the latter constraint was relaxed to say that *few* track-report pairs whose probability of association

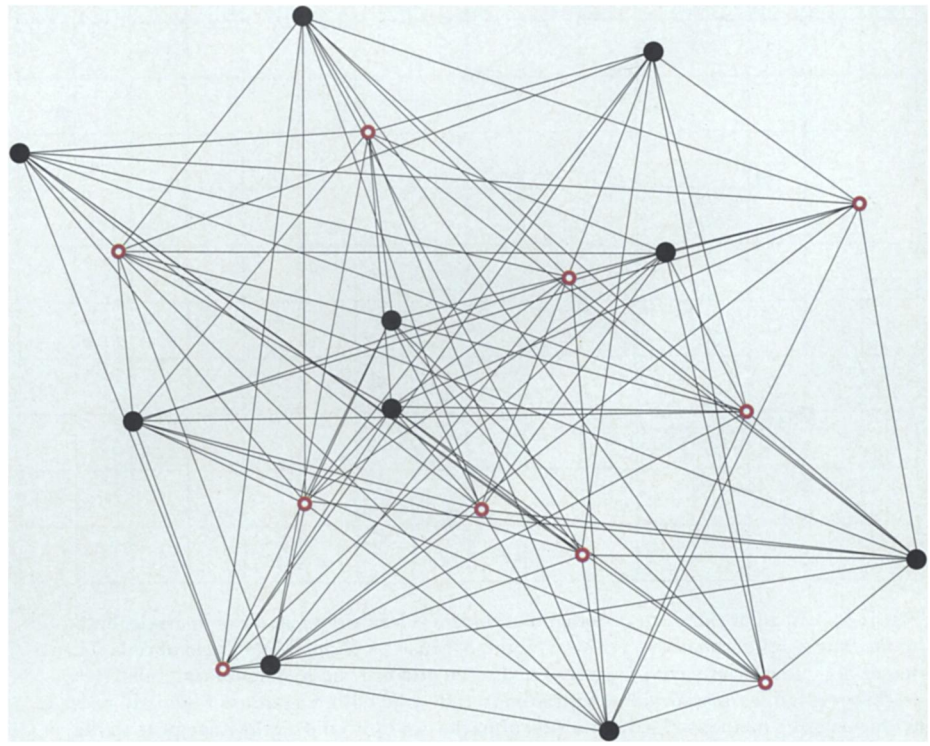


Figure 6. Combinatorial bottleneck afflicts all of the most straightforward algorithms for multiple-target tracking. In these algorithms every incoming position report must be compared with every candidate track, in order to see whether or not they might be associated. Here there are 10 reports *(black dots)* and 10 tracks *(colored circles, representing new projected positions)*; thus 100 comparisons are needed. More generally, the effort that must be expended to solve a correlation problem for n tracks and n reports is proportional to n^2. For large values of n, these n^2 steps are the dominant factor in determining the performance and the maximum capacity of a tracking program.

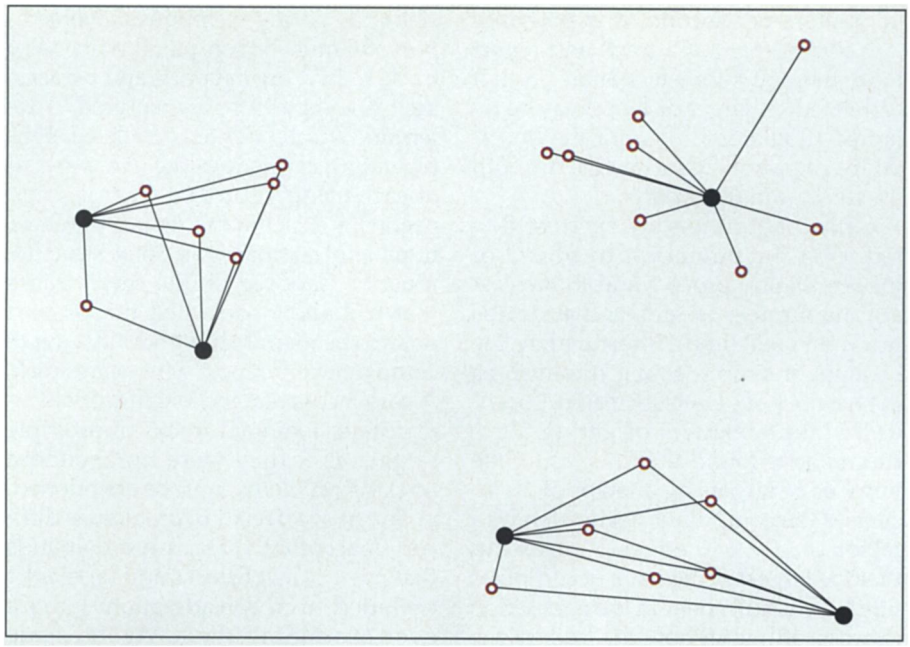


Figure 7. Clustering methods attempt to avoid the n^2 bottleneck by grouping tracks into clusters. Then an incoming report can be compared only with the tracks in the nearest cluster. Such methods work well only if the data fall readily into well-separated clusters. If the targets are evenly dispersed, they may form either n clusters of one track each or one giant cluster of n tracks; in either case, the algorithm reverts to the original n^2 performance.

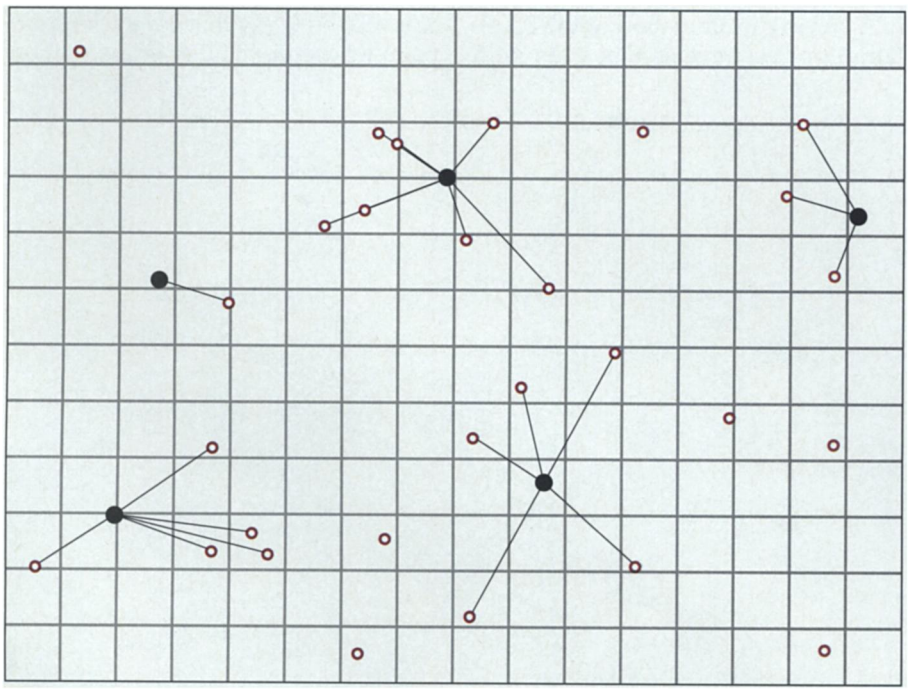


Figure 8. Grid algorithms offer another mechanism for confining attention to tracks in the immediate neighborhood of a position report. All space is divided into a grid of cells. Then a report is compared only with the tracks in its own grid cell and in neighboring cells. Here tracks are eligible for association with a report if they lie within a radius of two grid cells. As with clustering methods, the distribution of tracks can confound a grid strategy. If all the tracks are in a single cell, or if all the cells must be examined to be sure of assigning a report to the correct track, the algorithm bogs down.

exceeds the threshold could be mistakenly disregarded. This seemingly reasonable compromise, however, led to numerous ad hoc schemes that either failed to adequately limit the number of comparisons or failed to adequately limit the number of missed correlations. Some approaches were susceptible to both problems.

Most of the ad hoc strategies depend heavily on the distribution of the targets. A common approach is to identify clusters of targets that are sufficiently separated that reports from targets in one cluster will never have a significant probability of association with tracks from another cluster. This allows the correlation process to determine from which cluster a particular report could have originated and then to compare the report only to the tracks in that cluster. The problem with this approach is that the number of properly separated clusters depends on the distribution of the targets and therefore cannot be controlled by the clustering algorithm. If n tracks are partitioned into n clusters, each consisting of a single track, or into a single cluster of n tracks, the method still results in a computational cost equivalent to the comparison of every report to every track. Unfortunately, most real-world tracking problems tend to be close to one of these extremes.

A correlation strategy that avoids some of the distribution problems associated with clustering is to partition the space in which the targets reside into grid cells. Each track can then be assigned to a cell according to its mean projected position. In this way the tracks that might be associated with a given report can be found by examining only those tracks in cells within close proximity to the report's cell. The problem with this approach is that its performance depends heavily on the size of the grid cells as well as on the distribution of the targets. If the grid cells are large and the targets are densely distributed in a small region, every track will be within a nearby cell. Conversely, if the grid cells are small, the algorithm may spend as much time examining cells (most of which may be empty) as would be required to simply examine each track. In a three-dimensional grid, there are 26 cells immediately adjacent to each cell. If the search is extended to a radius of two cells, the number of cells that must be examined on each query rises to 124.

Binary Search

The deficiencies of grid methods suggest the need for a more flexible data structure. The main requirement imposed on the data structure has already been mentioned: It must allow all proximate track-report pairs to be identified without having to compare every report with every track (unless, of course, ev-

ery track is within the prescribed proximity to every report).

A clue to how this trick might be accomplished comes from one of the best-known algorithms in computer science: binary search. Suppose you are given a sorted list of n numbers and asked to find out whether or not a specific number, q, is included in the list. The most obvious search method is simply to compare q with each number in sequence; in the worst case (when q is the last number or else is not present at all), the search requires n comparisons. There is a much better way. Because the list is sorted, if you find that q is greater than a particular element of the list, you can exclude

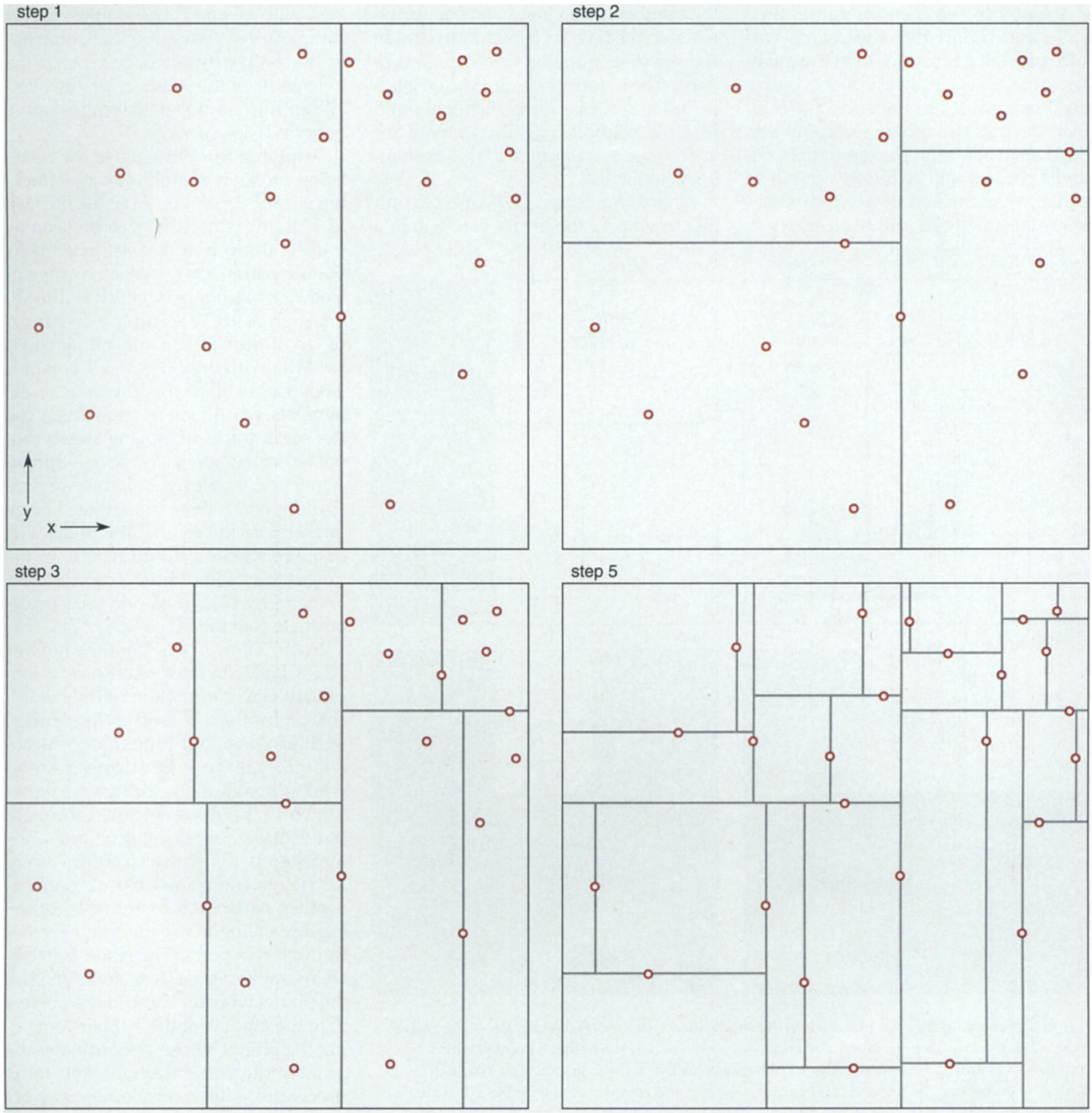


Figure 9. Multidimensional search tree divides space in a way that allows a correlation problem to be solved in fewer than n^2 steps for any reasonable distribution of tracks. Shown here is a method of building the data structure required for such a search; Figures 10 and 11 illustrate actual search procedures. The first step is to identify the track whose x coordinate is the median value of all the x coordinates; in this case there are 25 tracks, and so the median track is the 13th one counting from the left. A vertical line is drawn through the median track, partitioning the space into left and right rectangles. In step 2 the same procedure is applied in each of the left and right rectangles, except that now each subspace is divided horizontally through the track with the median value of the y coordinate. In step 3 the resulting rectangles are further subdivided, using a vertical partition through the track with the median x coordinate in each rectangle. The procedure continues in this way, alternating x and y coordinates, until the subspaces are empty. For the 25 tracks, five iterations are needed. In three dimensions, the algorithm would cycle repetitively through x, y and z coordinates.

from further consideration not only that element but all those that precede it in the list. This principle is applied optimally in binary search. The algorithm is recursive: First compare q to the median value in the list of numbers (by definition, the median will be found in the middle of a sorted list). If q is equal to the median value, then stop, and report that the search was successful. If q is greater than the median value, then apply the same procedure recursively to the sublist greater than the median; otherwise apply it to the sublist less than the median. Eventually either q will be found—it will be equal to the median of some sublist—or a sublist will turn out to be empty, at which point the procedure terminates and reports that q is not present in the list.

The efficiency of this process can be analyzed as follows. At every step, half of the remaining elements in the list are eliminated from consideration. Thus the total number of comparisons is equal to the number of halvings, which in turn is equal to the base-two logarithm of n. In the standard notation of computer science, the performance of the algorithm is said to scale as $O(\log_2 n)$. If n is a million, for example, only 20 comparisons are needed to determine if a given number is in the list.

Binary search can also be used to find all elements of the list that are within a specified range of values, (*min*, *max*). Specifically, the above procedure can be applied to find the position in the list of the largest element less than *min* and the position of the smallest element greater than *max*. The elements between these two positions then represent the desired set. Finding the positions associated with *min* and *max* requires $\log_2 n$ comparisons. Assuming that some operation will be carried out on each of the k elements of the solution set, the overall scaling law for satisfying a range query is $O(\log_2 n + k)$.

Applying this principle to the correlation problem in multiple-target tracking is very appealing. After all, the task of a gating algorithm can be seen as finding the subset of a list of n tracks that lie within some plausible range of a new position report, which is directly analogous to searching a list for a range of numbers. If the gating could be done with only $\log_2 n + k$ comparisons rather than n comparisons, the benefits would be appreciable. The trouble is, tracks of moving targets cannot be sorted into a simple one-dimensional list; they exist in a three-dimensional space. A three-dimensional range query must identify all the points in a data set whose x coordinate is in the range (x_{min}, x_{max}), whose y coordinate is in the range (y_{min}, y_{max}) and whose z coordinate is in the range (z_{min}, z_{max}).

In 1975 Jon Louis Bentley, now of AT&T Bell Laboratories, devised a simple but clever extension of the binary-search method to solve the general multidimensional range-query problem. To see how Bentley's method works, imagine a set of tracks represented by points distributed throughout a three-dimensional volume. The first step is to list the x coordinates of the points and choose the median value; then partition the volume by drawing a plane perpendicular to the x axis through this point. The result is to create two subvolumes, one containing all the points whose x coordinates are less than the median and the other containing the points whose x coordinates are greater than the median. The same procedure is then applied recursively to the two subvolumes, except that now the partitioning planes are drawn perpendicular to the y axis, and they pass through points that have median values of the y coordinate. The next round uses the z coordinate, and then the procedure returns cyclically to the x coordinate. The recursion continues until the subvolumes are empty.

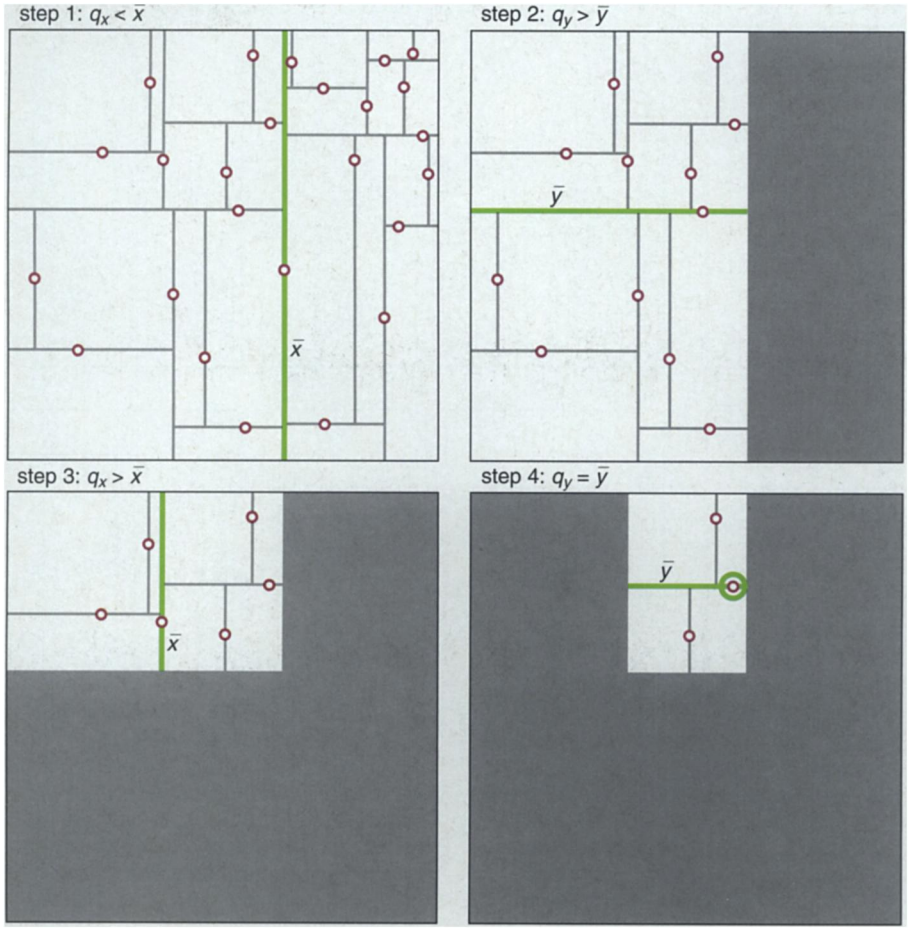


Figure 10. Searching for a specific track in the subdivided space proceeds through a progressive narrowing of focus. The question to be answered by the search is whether a point q, with coordinates q_x and q_y, is present in the set of tracks recorded in the data structure. The search begins by examining the highest-level partition *(step 1)* and comparing q_x with the x coordinate of the partition (which is, of course, the median x coordinate, designated here $\bar{x}$). If q_x and $\bar{x}$ are equal, the search ends, since q has been found: It is the track that lies along the partition. If q_x is greater than $\bar{x}$, the subspace to the right of the partition is examined next, and the left subspace is excluded from further consideration; if q_x is less than $\bar{x}$ (as is the case here), attention turns instead to the left subspace. Next *(step 2)* q_y is compared with the median y value in the remaining subspace, and a similar decision is made: Either the median value is returned as the result of a successful search or else one of the subspaces is eliminated. The next bisection *(step 3)* is based on the value of q_x again; finally in step 4 q_y is found to be equal to $\bar{y}$, and the circled point is identified as the subject of the search. With this procedure the number of steps needed to find one of n points is no greater than $\log_2 n$; for the 25 points shown, at most five steps are needed.

Searching Bentley's subdivided volume for the presence of a specific point, with given x, y and z coordinates, is a straightforward extension of standard binary search. As in the one-dimensional case, the search proceeds as a series of comparisons with median values, but now attention alternates among the three coordinates. First the x coordinates are compared, then the y, then the z, and so on. At the end either the chosen point will be found lying on one of the median planes, or else the procedure will come to an empty subvolume.

Searching for all the points that fall within a specified interval is somewhat more complicated. The search proceeds as follows: If x_{min} is less than the median-value x coordinate, then the left subvolume must be examined. If x_{max} is greater than the median value of x, the right subvolume must be examined. Since the median could well lie between x_{min} and x_{max}, it is possible that neither subvolume is eliminated from consideration; this possibility does not come up in the one-dimensional search. At the next level of recursion, the comparison is done using y_{min} and y_{max}, then z_{min} and z_{max}.

A detailed analysis of the algorithm reveals that for d dimensions (provided that d is greater than 1), the number of comparisons performed during the search can be as high as $O(n^{1-1/d} + k)$; thus in three dimensions the performance is proportional to $O(n^{2/3} + k)$. In the task of matching n reports with n tracks, the range query must be repeated n times, and so the performance is given by $O(n \times n^{2/3} + k)$, or $O(n^{5/3} + k)$. This scaling is better than quadratic but not nearly as good as the logarithmic scaling observed in the one-dimensional case, which works out, for n range queries, to $O(n \log_2 n + k)$. The reason for the penalty in searching a multidimensional tree is the possibility at each step that both subtrees will have to be searched. In practice, however, this seldom happens, and the worst-case scaling for Bentley's technique is rarely seen. A variety of enhancements have been devised over the years that further improve its average-case behavior. (Other data structures have been developed that can satisfy multidimensional range queries in $O(\log_2{}^d n)$ time. But due to their high computational overhead, these data structures are of more theoretical interest than practical value.)

The most natural way of implementing Bentley's search strategy is with a data structure called a binary tree, a structure made up of nodes linked by "pointers," which are computer memory addresses specifying the location of other nodes. In a binary tree each node can include an item of data as well as exactly two pointers, which specify the left and right children of the node. At the root of the tree is a node whose data ele-

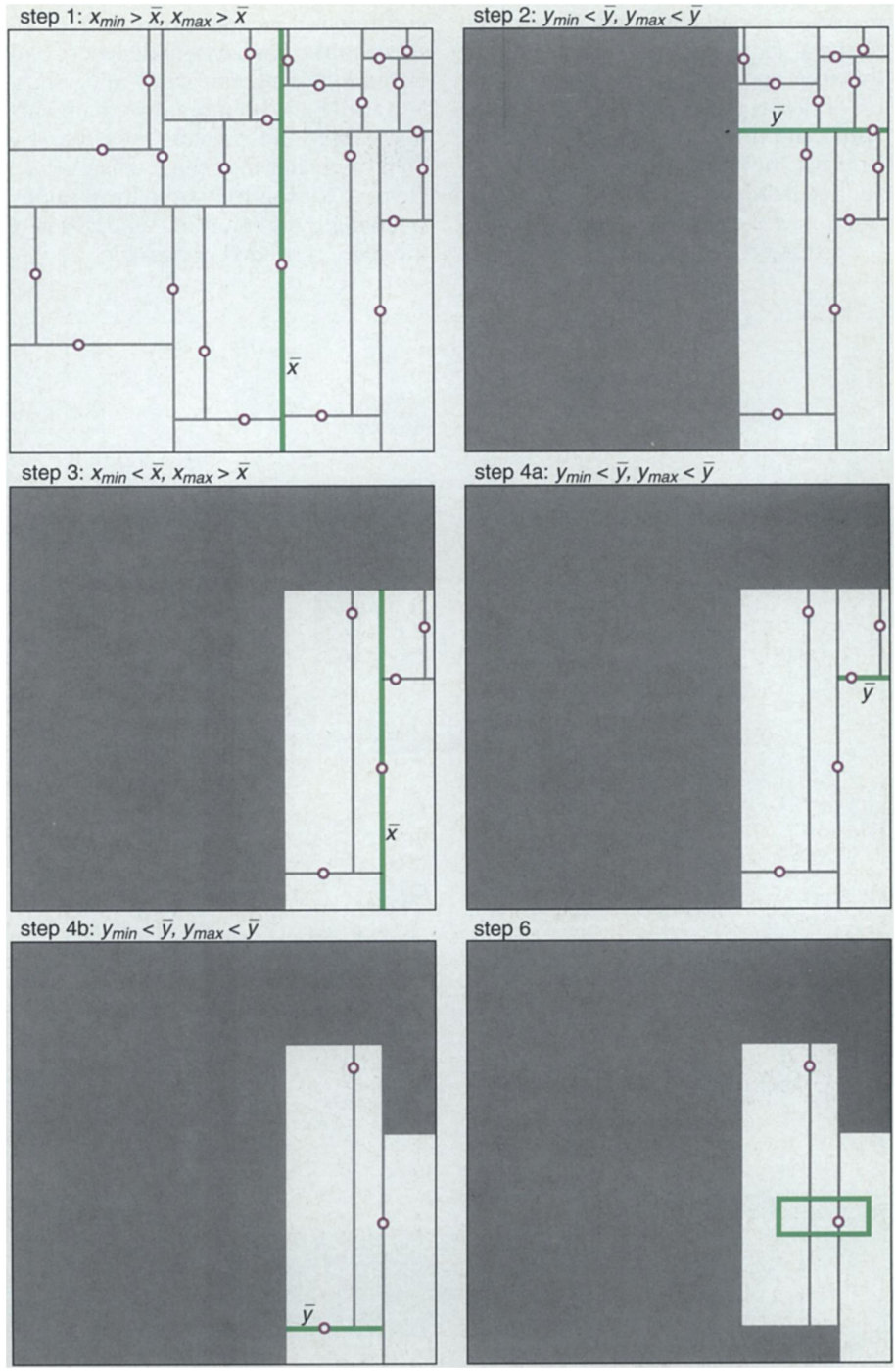


Figure 11. Searching for all points that lie within a range of coordinates is somewhat more complicated and less efficient than searching for a single specified point. The object of the search is to find all the tracks within a rectangle bounded by the coordinates x_{min}, y_{min}, x_{max}, y_{max}. Step 1 compares x_{min} and x_{max} with $\bar{x}$, the median value of x; it turns out that both x_{min} and x_{max} are greater than $\bar{x}$, and so the left subspace is eliminated. Similarly in step 2 the upper subspace is excluded. In step 3, however, x_{min} and x_{max} are found to straddle $\bar{x}$, and so neither subspace can be neglected in further search operations. The left and right subspaces defined in step 3 are examined separately in steps 4a and 4b; after yet another stage of refinement in step 5 *(not shown)*, the area defined by the range query is exhibited in step 6. A range query of this kind can be used in multiple-target tracking to find all the tracks that lie within a certain range of a specified position report.

ment is the point with the median value of the x coordinate; the left pointer of this root node leads to a subtree made up of nodes representing all the points with x coordinates less than the median; similarly the right pointer designates the subtree in which all points have x coordinates greater than the median. The data structure, like the corresponding algorithm, is a recursive one: Each of the subtrees also has a root node whose data element is a median value and whose left and right pointers designate subtrees where all the values are either less than or greater than the median. At the "leaves" of the tree are empty nodes, signaling that no further subdivision of the space is needed or possible.

Multidimensional Trees in Gating

In the late 1980s, my colleagues and I at the Naval Research Laboratory began studying ways of applying advances in computer science to various problems in tracking and correlation. Experiments with multidimensional tree structures produced our most encouraging results. Unlike methods based on

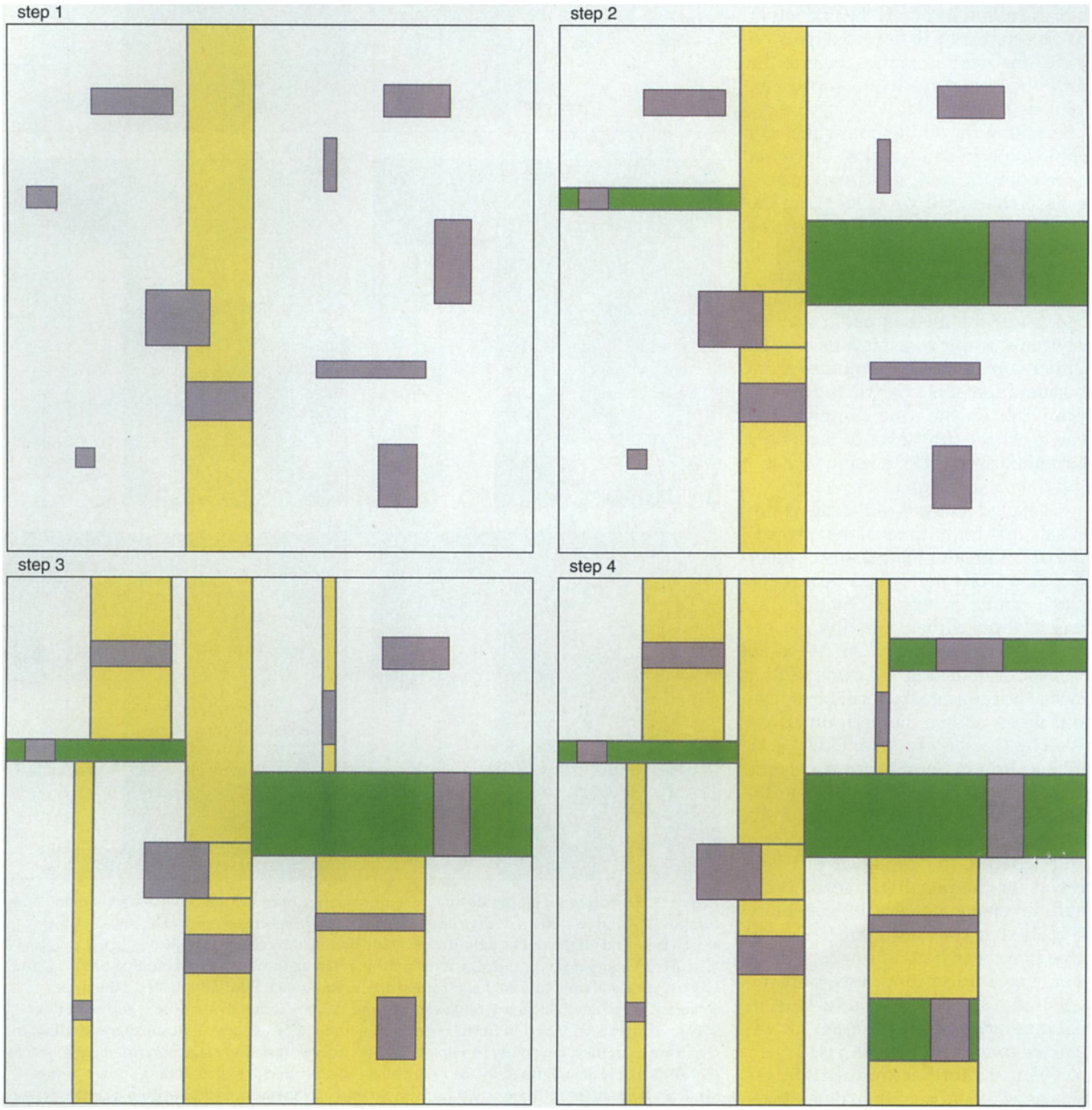


Figure 12. Allowing for uncertainty in both tracks and position reports requires a more elaborate data structure: a multidimensional *ternary* tree, which at each level divides space into three areas instead of two. The ternary tree organizes objects that are not pointlike but instead occupy a rectangular area. Step 1 identifies the object with median x coordinates and subdivides the entire space into three regions: points that lie entirely to the left of this object, points that lie entirely to the right, and points in the range of x coordinates spanned by the median object. In step 2 each of these three regions is subdivided in turn, creating areas entirely above, entirely below and straddling the object with median values of the y coordinate. Another level of vertical divisions is made in step 3. For the 10 objects shown here, the process terminates after the additional horizontal trisections of step 4.

clustering or grids, the performance of techniques based on tree structures is relatively independent of the distribution of the targets. In particular, the computation time required for a given range query depends more on the number of points that satisfy the query than on the extent of the range.

In these experiments we stored a set of projected track positions as points in a multidimensional tree structure. Then for each position report we defined a gating range, and searched the tree for all track points that fell within the range. As noted above, each such search requires at most $n^{2/3} + k$ operations, and in many instances the actual performance is appreciably better. After this gating pass, a full-scale calculation of the probability of association is needed only for the k track-report pairs that pass the range test. With these methods we were able to simulate the correlation process with more than a million targets.

The problem with this scheme is that it cannot readily account for uncertainty in the track projections, because objects in the tree structure are treated as points. Uncertainty in the position reports is accommodated by the use of a range query; that is, the gating range can be adjusted to encompass the entire error ellipse associated with a position report. Indeed, even the shape of the report's error ellipse can be taken into account, since the range query can be performed with different x, y and z ranges. But ignoring the error in the track projections is unacceptable; it could lead to missed correlations and a failure of the tracking algorithm to converge.

One approach to solving this problem is to shift all the uncertainty associated with the tracks onto the reports. The nature of this transfer is easy to understand in the simple case of a track and a report whose error ellipsoids are spherical and just touching. Reducing the radius of the track error sphere to zero while increasing the radius of the report error sphere by an equal amount leaves the enlarged report sphere just touching the point representing the track, so that the correlation is preserved. Unfortunately, when this idea is applied to multiple tracks and reports, the query region for *every* report must be enlarged in all directions by an amount large enough to accommodate the largest error ellipse associated with *any* track. Techniques have been devised to find the minimum enlargement necessary to guarantee that every track correlated with a given report will be found; however, a few tracks with large error ellipses can result in such large query regions that an intolerable number of uncorrelated tracks will also be identified.

The solution we ultimately found was to devise a new multidimensional tree structure that can represent uncertainty in the stored data. The new form of tree still cannot accommodate error *ellipses*, but it allows for error *boxes*, and that has turned out to be an acceptable approximation. A box is defined as the smallest cuboidal shape, with sides parallel to the coordinate axes, that can entirely surround the error ellipse. Since the axes of the ellipse may not correspond to those of the coordinate system, the box may differ significantly in size and shape from the ellipse it encloses.

The new data structure is needed in order to allow searches in which one range of coordinate values is compared with another range, rather than the simpler case where a range is compared with a single point. A binary tree will not serve this purpose because it is not always possible to say whether one interval is greater than or less than another; the intervals may well be intersecting and yet not identical. What is needed is a *ternary* tree, with three descendants per node. At each stage in a search of the tree, the maximum value of one interval is compared with the minimum of the other, and vice versa. These comparisons can potentially eliminate either the left subtree or the right subtree, but whatever their outcome, it will be necessary to examine the middle subtree—the one made up of nodes representing boxes that might intersect the query interval. In three dimensions, the extra search effort demanded by the added subtrees is usually not great unless there are many intersections among the boxes stored in the tree. Although more sophisticated tree variants provide better worst-case scaling, the simplicity of ternary trees tends to translate into relatively low computational overhead and competitive performance in practice.

At this point it is possible to summarize our entire tracking algorithm. Tracks are recorded by storing the information—such as current positions, velocities and accelerations—that a Kalman filter needs to estimate the future position of each candidate target. When a new batch of position reports arrives, the existing tracks are projected forward to the time of the reports. An error ellipse is calculated for each track and each report, and then a box is constructed around each ellipse. Now the boxes representing the track projections are organized into a multidimensional ternary tree. Each box representing a report becomes the subject of a complete tree search; the result of the search is the set of all track boxes that intersect the given report box. Track-report pairs whose boxes do not intersect are excluded from all further consideration. Next the set of track-report pairs whose boxes *do* overlap is examined more closely to see whether the inscribed error ellipses also overlap. Whenever this calculation indicates a correlation, the track is extended to the position of the new report. Tracks that consistently fail to be associated with any reports are eventually deleted; reports that cannot be associated with any existing track initiate new tracks.

The Strategic Defense Initiative

The birth of SDI in 1983 established the most challenging and focused set of engineering and scientific funding priorities since the space program of two decades before. The goal was to develop a space-based system to defend against a full-scale missile attack against the United States. Some of the most critical problems were the design and deployment of sensors to detect the launch of missiles at the earliest moment possible in their 20-minute flight, and the design and deployment of weapons systems capable of destroying the detected missiles. Although an automatic tracking facility would clearly be an integral component of any SDI system, it was not generally considered a "high risk" technology. Tracking, especially of aircraft, had been widely studied for more than 30 years, and so the tracking of nonmaneuvering ballistic missiles seemed to be a relatively simple engineering exercise. The principal constraint imposed by SDI was that the tracking be precise enough to predict a missile's future position to within a few meters, so that it could be destroyed by a high-energy laser or a particle-beam weapon.

The high-precision tracking requirement led to the development of highly detailed models of ballistic motion that took into account the effects of atmospheric drag and various gravitational perturbations over the earth. By far the most significant source of error in the tracking process, however, resulted from the limited resolution of existing sensors. This fact reinforced the widely

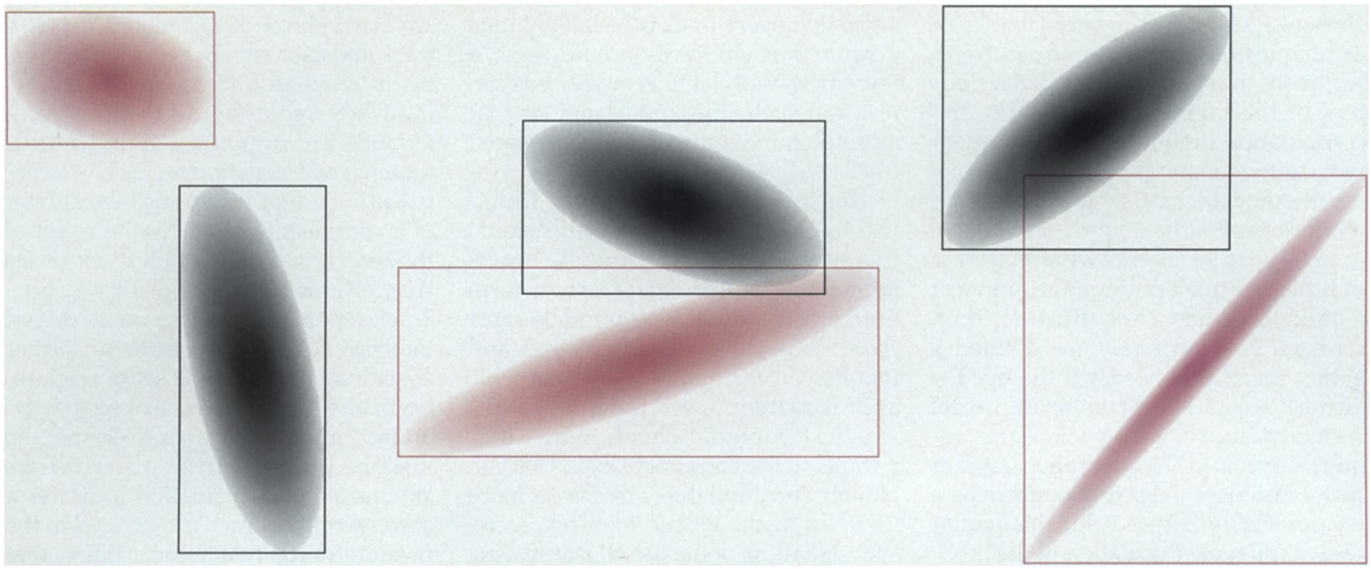


Figure 13. Intersection of error boxes offers a preliminary indication that a track and a report are probably correlated. A definitive test of correlation requires a computation to determine the extent to which the error ellipses overlap, but this computation is a time-consuming one. Furthermore, there is no convenient way to store ellipses in a multidimensional search tree. A useful alternative strategy is to draw a box circumscribing each ellipse; if the sides of the boxes are parallel to the coordinate axes, the boxes are readily stored in a search tree. If two boxes do not intersect *(left)*, it is safe to assume that the inscribed ellipses do not intersect either. When the boxes do overlap, in most instances the ellipses also touch *(center)*. There are a few cases where the boxes overlap but the ellipses do not *(right)*; these false positives must be weeded out in subsequent processing.

held belief that the main obstacle to effective tracking was the poor quality of sensor reports. The impact of large numbers of targets seemed manageable: One had only to build larger, faster computers. Although many in the research community thought otherwise, the prevailing attitude among funding agencies was that if 100 objects could be tracked in real time, then it should be little trouble to build a machine 100 times faster—or simply have 100 machines run in parallel—to handle 10,000 objects.

Among the challenges facing the SDI program, multiple-target tracking seemed far simpler than what would be required to further improve sensor resolution. This belief led to the awarding of contracts to build tracking systems in which the emphasis was placed on high precision at any cost in terms of computational efficiency. These systems did prove valuable for determining bounds on how accurately a single cluster of three to seven missiles could be tracked in an SDI environment, but ultimately pressures mounted to scale up to more realistic numbers. In one case, a tracker that had been tested on five missiles was scaled up to track 100, causing the processing time to increase from a couple of hours to almost a month of nonstop computation for a simulated 20-minute scenario. It was later determined that the bulk of the computations took place in the correlation step.

In response to a heightened interest in scaling issues, the Naval Research Laboratory developed prototype systems based on efficient search structures. One of these systems demonstrated that 65 to 100 missiles could be tracked in real time with a program running on a personal workstation. These results were based on the assumption that a good-resolution radar report would be received every five seconds for every missile, which is unrealistic in the context of SDI; nevertheless, the demonstration did provide convincing evidence that SDI trackers could be adapted to avoid quadratic scaling. A tracker developed for the SDI National Testbed in Colorado Springs achieved significant performance improvements after a tree-based search structure was installed in its correlation routine; the new algorithm was superior for as few as 40 missiles. Stand-alone tests showed that the search component could process 5,000 to 10,000 range queries in real time on a modest computer workstation. These results suggested that the problem of correlating vast numbers of tracks and reports had been solved. Unfortunately, a new difficulty was soon discovered.

Up to now I have adopted the simplifying assumption that all position reports arrive in batches, with all the reports in a batch reflecting measurements made at the same instant. A real sensor system would not work this way; reports would arrive in a continuing stream and would be distributed over time. In order to determine the probability that a given track and report correspond to the same object, the track must be projected to the measurement time of the report. If every track has to be projected to the measurement time of every report, the combinatorial advantages of the tree-search algorithm will be lost.

A simple way to avoid multiple projections for each track is to increase the search radius in the gating algorithm to account for the maximum distance an object could travel during the maximum time difference between any track and report. For example, if the maximum speed of a missile is 10 kilometers per second, and the maximum time difference between any report and track is five seconds, then 50 kilometers would have to be added to each search radius to assure that no correlations are missed. For boxes used to approximate ellipsoids, this means that each side of the box must be increased by 100 kilometers.

As estimates of what constitutes a realistic SDI scenario became more accurate, members of the tracking community learned that successive reports of a particular target often would be separated by as much as 30 to 40 seconds. To account for time differences this large would require boxes so immense that the number of spurious returns would negate the

benefits of efficient search. Demands for a sensor configuration that would report on every target at intervals of 5 to 10 seconds were considered unreasonable for a variety of practical reasons. It seemed that the use of sophisticated correlation algorithms had finally reached its limit. Several heuristic "fixes" were considered, but none solved the problem.

A detailed scaling analysis of the problem ultimately pointed the way to a solution: Simply accumulate sensor reports until the difference between the measurement time of the current report and the earliest report exceeds a threshold. A search structure is then constructed from this set of reports; the tracks are projected to the mean time of the reports; and the correlation process is performed with the maximum time difference being no more than half of the chosen time-difference threshold. The subtle aspect of this deceptively simple approach is the selection of the threshold. If it is too small, every track will be projected to the measurement time of every report. If it is too large, every report will fall within the search volume of every track. A formula has been derived that, with only modest assumptions about the distribution of targets, assures the optimal trade-off between these two extremes. Empirical results confirm that the approach essentially solves the time-difference problem.

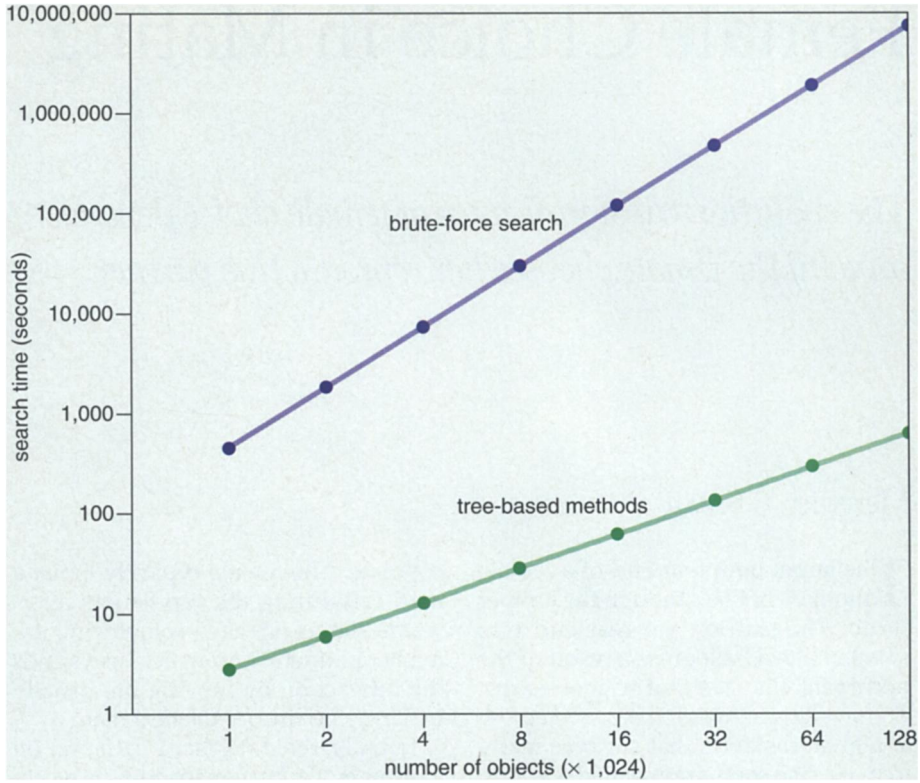


Figure 14. Performance of tree-based association algorithms is clearly superior to that of methods that have n^2 efficiency. Even for as few as 1,000 targets, the tree-based algorithm has a considerable advantage: Execution time on a personal workstation is less than three seconds for the tree-based search, and almost eight minutes for a brute-force search. For 128,000 targets, the tree-based program takes a little more than 10 minutes, whereas the execution time of the brute-force algorithm has grown to 12 weeks. In the test data that generated these results, the density of targets was such that five error boxes, on average, would intersect. (Not all of the tests plotted were actually run; some data were extrapolated or estimated.)

Conclusion

The correlation of reports with tracks numbering in the thousands can now be performed in real time on many currently available computers. More research on large-scale correlation remains to be done, but work has already begun on implementing efficient correlation modules that can be incorporated into existing tracking systems. Ironically, by hiding the intricate details and complexities of the correlation process, these modules give the appearance that multiple-target tracking involves little more than the concurrent processing of several single-target problems. Thus a paradigm with deep historical roots in the field of target tracking is at least partially preserved.

It should be noted that the techniques described in this article are applicable only to a very restricted class of tracking problems. Other problems, such as the tracking of military forces, demand more sophisticated approaches. Not only does the mean position of a military force change, but so also does its shape. Moreover, reports of its position are really only reports of the positions of its parts, and various parts may be moving in different directions at any given instant. Filtering out the local deviations in motion to determine the net motion of the whole is beyond the capabilities of a simple Kalman filter. Other difficult tracking problems include the tracking of weather phenomena and soil erosion. The history of multiple-target tracking suggests that, in addition to new mathematical techniques, new algorithmic techniques will certainly be required for any practical solution to these problems.

Bibliography

Bar-Shalom, Y., and T. E. Fortmann. 1988. *Tracking and Data Association*. San Diego: Academic Press.

Bentley, J. L. 1975. Multidimensional binary trees for associative searching. *Communications of the ACM* 18:509–517.

Blackman, Samuel S. 1986. *Multiple-Target Tracking with Radar Applications*. Dedham, Mass.: Artech House.

Uhlmann, Jeffrey K., and Miguel R. Zuniga. 1991. Results of an efficient gating algorithm for large-scale tracking scenarios. *Naval Research Reviews* 1:24–29.

Collins, Joseph B., and Jeffrey K. Uhlmann. 1990. Efficient gating in data association for multivariate Gaussian distributions. Accepted by *IEEE Transactions on Aerospace and Electronic Systems*.

Uhlmann, Jeffrey K. 1991. Adaptive partitioning strategies for ternary tree structures. *Pattern Recognition Letters* 12:537–541.

Zuniga, Miguel R., J. M. Picone and Jeffrey K. Uhlmann. 1990. Efficient algorithm for improved gating combinatorics in multiple-target tracking. Submitted to *IEEE Transactions on Aerospace and Electronic Systems*.