Representing and Scoring Track Hypotheses for Multitarget groups

March 13, 1998

Charles H. Hastings, Langhorne P. Withers, Jr. and Ronald N. DeWitt
Pacific-Sierra Research Corporation
Arlington, VA 22209

ABSTRACT

This paper presents an economical method for representing track sets for observations of possibly unstable groups of vehicles, and an algorithm to create and update the multi-vehicle track association hypotheses. We represent bundles of tracks to avoid redundant computing, but we remember to multiply our track scores by how many individual target tracks are being represented.

This paper is divided into two main parts. The first part describes representing these bundles of tracks efficiently and the related logic of track extension for multi-target observations. To begin, we consider the combinatorial explosion of association hypotheses for tracking groups of target objects that may split or merge. For a population of 10 targets, if five are observed in one spot and then three are observed in another spot, there are 9, 072,000 track combinations represented and covered by a single update. However, as we will show, all of these equivalent possibilities can be represented as a single track or track bundle.

We count the combinations represented by one multitrack, then we multiply the single-thread track score times the number of threads represented to get the total probability represented. We will derive a formula for the multitrack score update.

The second part of the paper describes an algorithm for extending and initiating multitracks. The algorithm for extending and initiating multitracks is an enhancement of the PSR Tracker Multi-Hypothesis Manager algorithm with pruning. However, where the old algorithm constructed all valid hypotheses for a new observation and then pruned, the new algorithm uses a least cost branch and bound technique to create only those hypotheses likely to survive the pruning process. This results in a significant reduction in the number of hypotheses generated.

1. Introduction

This paper presents an approach to solving the problem of tracking unstable groups of targets in a multiple hypothesis framework. It is assumed that the groups of targets may split apart or merge at any time; that is, no simplifying assumptions can be made as to the groups' maintaining stable configurations or group proximity relations as they move. Because the targets are ground vehicles, they are able to either move or park. Observation data about the groups report locations and vehicle counts, and possibly target types. A group may contain different target types, or only one type.

The PSR Multi-Hypothesis Tracker (referred to as Tracker) receives a series of observations of targets from an input source. Typically, the observations arrive with some prior processing from a Single Input Correlator (SICOR) or a Multiple Input Correlator (MICOR). These prior processing steps reduce the volume of input observations by correlating multiple observations of a target in the same location and time window performing the same activity. Tracker then attempts to correlate observations of targets that may have moved around the battlefield and through the various activity states the target may perform.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (FROM - TO)
13-03-1998	Conference Proceedings	xx-xx-1998 to xx-xx-1998

4. TITLE AND SUBTITLE
Representing and Scoring Track Hypotheses for Multitarget groups
Unclassified

5a. CONTRACT NUMBER
5b. GRANT NUMBER
5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Hastings, Charles H. ;
Withers, Langhorne P. ;
DeWitt, Ronald N. ;

5d. PROJECT NUMBER
5e. TASK NUMBER
5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME AND ADDRESS
Pacific-Sierra Research Corporation
Arlington, VA22209

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME AND ADDRESS
Director, CECOM RDEC
Night Vision and Electronic Sensors Directorate, Security Team
10221 Burbeck Road
Ft. Belvoir, VA22060-5806

10. SPONSOR/MONITOR'S ACRONYM(S)
11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APUBLIC RELEASE
,

13. SUPPLEMENTARY NOTES
See Also ADM201041, 1998 IRIS Proceedings on CD-ROM.

14. ABSTRACT
This paper presents an economical method for representing track sets for observations of possibly unstable groups of vehicles, and an algorithm to create and update the multi-vehicle track association hypotheses. We represent bundles of tracks to avoid redundant computing, but we remember to multiply our track scores by how many individual target tracks are being represented. This paper is divided into two main parts. The first part describes representing these bundles of tracks efficiently and the related logic of track extension for multi-target observations. To begin, we consider the combinatorial explosion of association hypotheses for tracking groups of target objects that may split or merge. For a population of 10 targets, if five are observed in one spot and then three are observed in another spot, there are 9, 072,000 track combinations represented and covered by a single update. However, as we will show, all of these equivalent possibilities can be represented as a single track or track bundle. We count the combinations represented by one multitrack, then we multiply the single-thread track score times the number of threads represented to get the total probability represented. We will derive a formula for the multitrack score update. The second part of the paper describes an algorithm for extending and initiating multitracks. The algorithm for extending and initiating multitracks is an enhancement of the PSR Tracker Multi- Hypothesis Manager algorithm with pruning. However, where the old algorithm constructed all valid hypotheses for a new observation and then pruned, the new algorithm uses a least cost branch and bound technique to create only those hypotheses likely to survive the pruning process. This results in a significant reduction in the number of hypotheses generated.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE	Public Release	20	Fenster, Lynn lfenster@dtic.mil
Unclassified	Unclassified	Unclassified			19b. TELEPHONE NUMBER International Area Code Area Code Telephone Number 703767-9007 DSN 427-9007

Prior to the development described in this paper, Tracker assumed that observations received contained one vehicle. If an observation was received which contained more than one vehicle, the observation was split into a set of observations with an implied negative correlation. That is, Tracker would not attempt to correlate the observations that came from the same original observation together. Unfortunately, this approach led to a variety of anomalies in the results. For example, an observation of three targets at a location followed by three targets at a second location would be split up into six "independent" observations (see Figure 1). These permutations of the one unique solution clutter the solution space and have the effect of diluting the probability assigned to each hypothesis. In addition, because Tracker was typically configured to prune hypotheses, with a sufficiently large set of multi-vehicle observations the pruning window would become filled with permutations of the same hypothesis, each with equivalent probability.

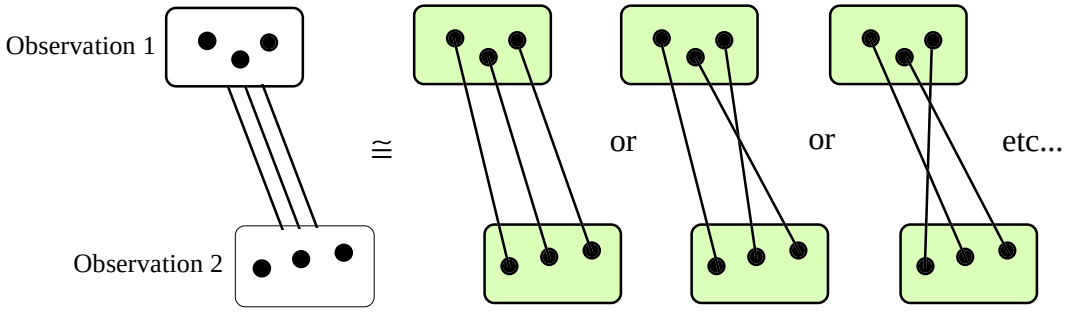

Figure 1. Multi-vehicle track example. Here a 3-track hypothesis (on the left side) represents 6 equivalent possible ways to map 3 vehicles in Observation 1 into the 3 vehicles in Observation 2.

An additional problem with the original approach was that the software would compute track extensions from prior tracks to the new observation unnecessarily. It should only have been necessary to extend prior tracks to this location once, regardless of how many vehicles were found in the new observation. Because Tracker split the observation up, these track extensions would be recalculated for each generated observation. This greatly increased the computational cost of processing the new observation.

To address these concerns, a new approach was needed. The basic idea of the new approach was to modify Tracker's hypothesis manager to accept multi-vehicle observations. By treating the observation as a unit, the computational concerns went away. Only one track extension would be necessary for all prior observations. In addition, the Tracker's hypothesis manager could use the multi-vehicle observation to eliminate the permutations of the unique track combinations.

With this in mind, we will describe how we represent the multitracks, how the multitrack multiplicity is computed, how the multitrack probability is calculated, and finally how Tracker's Hypothesis Manager generates these multitrack hypotheses efficiently.

2. Representing Multitracks efficiently

Our goal is to represent tracks with the minimum amount of redundancy. If the computations (Markov-Bayes update, track-set probability score) are the same for several tracks, we only need to do them once, and store them once, using a single track representation for the entire bunch. Because the track-probability score is the same for every track set in a family of T track sets, we multiply the score for one of them by the multiplicity of T to get the net probability for the track sets in that family.

A track is a sequence of observations assigned to a unique vehicle. For each target type i, there is a "motorpool" of $M^{(i)}$ vehicles to assign to data tracks. For simplicity, assume there are three vehicle types: type A (i=1) and type B (i=2), as well as OTHER (false targets such as large trucks, i=0). We label each track with a vehicle id (a "license plate number"). This brings us to the first redundancy: everything is the

same. Our calculations are unchanged for any vehicle of a given type that is assigned to a track. If we have $N^{(i)}$ tracks of the same vehicle type i, there are

$$\left[\begin{matrix} M^{(i)} \\ N^{(i)} \end{matrix}\right] = N^{(i)}!\binom{M^{(i)}}{N^{(i)}} = M^{(i)} \cdot (M^{(i)} - 1) \cdots (M^{(i)} - (N^{(i)} - 1))$$

ways to label them with unique ids.

Another redundancy comes into play when the observations report multiple vehicles (at the same location, activity, and type). We naturally want to represent a "braid" of *n* single-target tracks that go from one observation to the next together as an "*n*-track." For example, this simple 3-track segment represents 3! equivalent track sets:

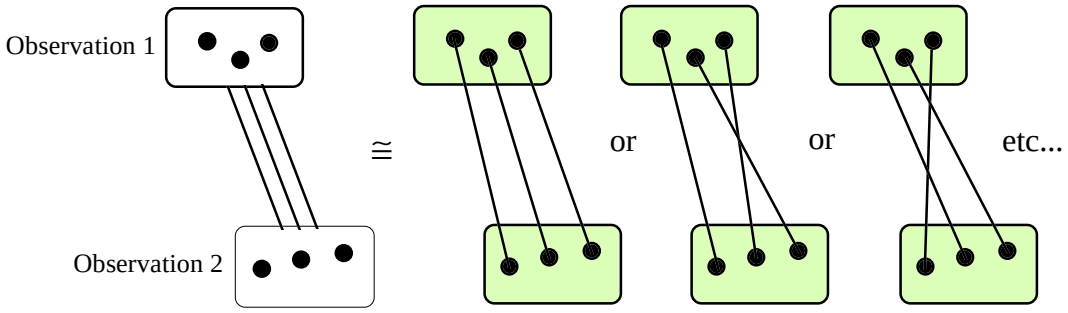

Figure 2. Multi-vehicle track representation. Here a 3-track hypothesis (on the left side) represents 6 equivalent possible ways to map 3 vehicles in Observation 1 into the 3 vehicles in Observation 2.

More generally, an *N*-track can split. The basic split operation occurs when some of its tracks through one observation (1) continue to the next observation (2), but some don't, they just end for now (and may continue on to a later observation, after the next one). For example, here a 5-track is split into a 3-track segment and a 2-track stop or pause. This can happen in $\left[\begin{matrix} 5 \\ 3 \end{matrix}\right] = 3!\binom{5}{3} = 5 \cdot 4 \cdot 3$ ways (5 ways to choose the predecessor of the first vehicle in Observation 2, then 4 ways to choose that of the second vehicle, 3 ways to choose that of the third vehicle). A few of the 60 ways are pictured on the right below:

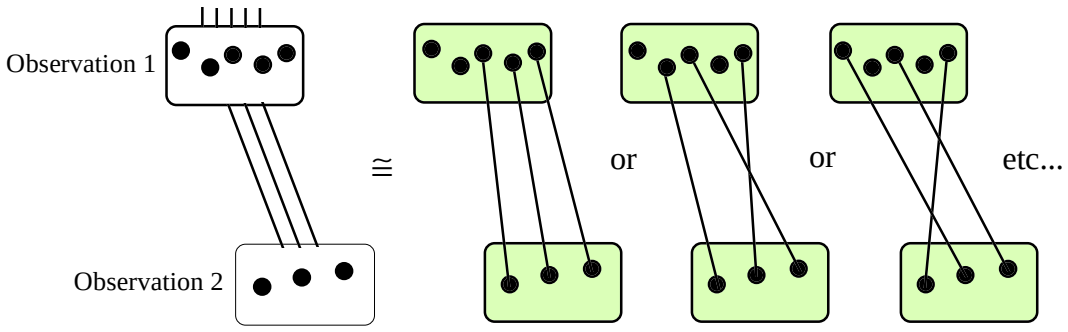

Figure 3. Basic track split. Here is a 5-track that splits, into a 3-track and a 2-track stop. The scheme on the left depicts 60 equivalent ways to map 3 of 5 vehicles in Observation 1 into the 3 vehicles in Observation 2.

How many sets of unique vehicle tracks does the scheme on the left represent in all? Let's assume the five tracks all begin with observation 1, and that the vehicles are assigned to them from a population of *M*=10.

Then our total is $\begin{bmatrix}10\\5\end{bmatrix}\begin{bmatrix}5\\3\end{bmatrix} = 5!\binom{10}{5}3!\binom{5}{3} = 10\cdot 9\cdot 8\cdot 7\cdot 6\cdot 5\cdot 5\cdot 4\cdot 3 = 9{,}072{,}000$ equivalent track sets represented by one diagram!

2.1 The equivalent track multiplier for track extensions

Since we update the track score recursively, what we really need is just the multiplier for extending the tracks by one new observation. That is, we have a track set that represents several equivalent track sets. When we extend these tracks by adding one more multi-vehicle observation, we will multiply the number of equivalent track sets by some factor. This is the number we actually need to update the track score. In the last example, this "extension multiplier" was just $3!\binom{5}{3} = 5\cdot 4\cdot 3 = 60$. (The other factor $5!\binom{10}{5} = 10\cdot 9\cdot 8\cdot 7\cdot 6\cdot 5 = 151{,}200$ would have been the initial multiplier, the number of ways to initiate the 5-track of the first observation from a motorpool of 10 vehicles.) We now show how to get this multiplier in more complex cases, such as the one in Figure 4:

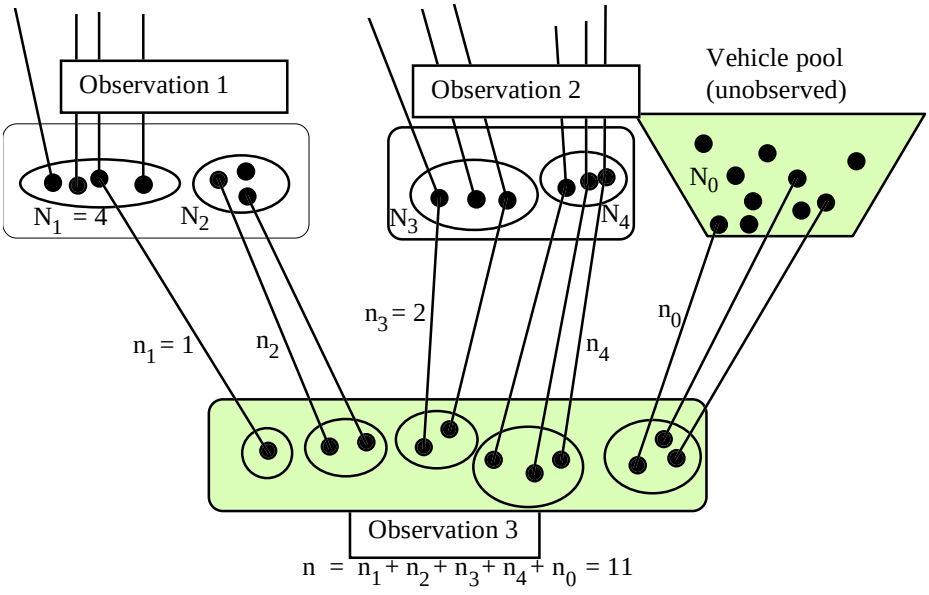

Figure 4. A typical track hypothesis extension. The existing tracks end with observation (1) or (2) and are extended to a new observation (3) with 11 vehicles of the same type. Each *N*-track (target group) can contribute *n* of its vehicles to account for some of the 11 vehicles. Also, the "motorpool" of vehicles not observed before can contribute new vehicles; these represent track initiations or new tracks.

In this example, we start with track hypotheses that end in observations 1 and 2, and we want to extend them by adding the new observation 3 with *n*=11 vehicles of this type. Note there are a total of 18

vehicles represented in Figure 4. (If more than 18 are observed, our initial motorpool population estimate of 18 must be too low.) There are many ways to account for the 11 vehicles observed. They can proceed from up to 4 of the track groups so far, or from the "motorpool" of 10 vehicles remaining to be observed. So if we fix the new track group counts as shown (making sure they add up to *n*), there are $\binom{N_k}{n_k}$ ways to select n_k vehicles from the N_k in each parent group. This provides the *n* vehicles to map into the *n* vehicles observed. Also, there are *n*! ways to map them into the n vehicles observed. So the extension multiplier is

$$n! \cdot \binom{N_1}{n_1} \cdot \binom{N_2}{n_2} \cdot \binom{N_3}{n_3} \cdot \binom{N_4}{n_4} \cdot \binom{N_0}{n_0} = 11! \cdot \binom{4}{1} \cdot \binom{3}{2} \cdot \binom{3}{2} \cdot \binom{3}{3} \cdot \binom{10}{3}.$$

The picture in Figure 4 applies to vehicles all of the same type. Assume we have partitioned all the vehicles that were in the new observation by types, including the OTHER (false target case). For example, suppose there are 30 vehicles observed and that their types are not definite. For one extension hypothesis, we may divide them into $n^{(1)} = 11$ of type A (as pictured above), $n^{(2)} = 7$ of type B, and $n^{(0)} = 12$ OTHERs. Then to handle all three vehicle types, we could simply draw a separate picture like Figure 4 for each type.

2.2 The Multitrack update probability score

We are now ready to give the update formula for the track-set hypothesis probability or "score". There is one missing ingredient, the dot product. We will introduce this and put together the track-set score update.

To extend each track group to the next observation, one Markov-Bayes update has to be computed. The usual Bayes update of the vehicle's distribution over states *x* folds in a new stationary observation (*data*) after the Markov projection *P(x/H)* this way:

$$P(x \,/\, data, H) = \frac{p_d \cdot P(data \,/\, x) \cdot P(x \,/\, H)}{\sum_x p_d \cdot P(data \,/\, x) \cdot P(x \,/\, H)}, \qquad (1)$$

where *H* (for history) represents the previous track data, *P(data/x)* is the observation error probability (for location, typically a Gaussian distribution around the center *x*, pictured by its equiprobability ellipses), and p_d is the probability of detecting the target when it is at *x* and observed [1].

As a by-product of the Bayes update to fold in the new observation, we have the "track-extension probability" (often referred to as the dot product). The dot product is the probability that the vehicle would actually be observed at the new location and time, given its track history so far (the previous data assigned to the vehicle). The dot product is the denominator of the Bayes update (1),

$$P(data \,/\, H) = \sum_x p_d \cdot P(data \,/\, x) \cdot P(x \,/\, H). \qquad (2)$$

Putting together the extension multiplier with the track extension probability, the dot product, gives us the track score update,

$$P(\text{extended track set}) = P(\text{track set}) \cdot n! \cdot \prod_{i=0:2} \prod_{k=0:r^{(i)}} \binom{N_k^{(i)}}{n_k^{(i)}} \left\{ P(data / H_k^{(i)}) \right\}^{n_k^{(i)}} / S, \qquad (3)$$

where S is the normalizing sum of the numerators over all possible track set extensions (for all possible partitions of the n targets just observed among the three target types and track groups). Note that the dot product does not have to be redone for all the partitions though; it depends only on the sequence of observations that belong to a given track.

Since *n!* is common to every extension, it cancels out when dividing by S and can be dropped from the calculation. Note that for the OTHER vehicles (false targets, type i=0), since the population $N^{(0)}$ (of large trucks, for example) is very large, we do not attempt to sort out tracks. We also assume the population is so large that it is not depleted by our observations, and so we set $N_0^{(0)} = N^{(0)}$ as an approximation. The "track count" (less one) $r^{(0)}$ for the OTHER vehicles is 0 (so the only value of k is 0, meaning that we are dealing with just with one multi-track, the new one drawn directly from the OTHER population). $H_0^{(0)}$ (the previous track history) is an empty track, and the dot product is that from taking the *data* against the OTHER's latent or steady-state distribution.

3. Least Cost Branch and Bound Algorithm for extension

Now that we've laid the groundwork for the multi-track representation and provided the formulas for calculating the probabilities, we need to generate the unique hypotheses without cluttering the solution space with permutations of these unique solutions. The original Tracker algorithm exhaustively generated all possible hypotheses which could be constructed from the new observation, and then pruned this list to keep the top k solutions. This was a manageable task with observations of one vehicle. The number of new hypotheses is bounded by the number of hypotheses saved from the prior observation (at most k) and the number of tracks that could be extended from the prior hypotheses. If there are P targets in the system, then the most hypotheses that can be generated for the new observation will be $O(k * P)$.

Unfortunately, with multi-vehicle observations the number of unique hypotheses which can be generated is $O(k \sum_{i=0}^{N} \binom{P}{i})$. P can be quite large (perhaps 100) for even a small number of target types being tracked through the system. The example described above in Figure 2 had 9,072,000 hypotheses, and that would be from one prior hypothesis. Each prior hypothesis would generate approximately the same number of hypotheses. Some of the test data we used would have generated billions of hypotheses. The old algorithm would generate all of these hypotheses so that it could keep the top k.

In order to reduce the number of hypotheses which need to be generated a new approach is needed. To arrive at this approach, the problem needs to be cast differently.

To start, let us define some notation for describing the problem. At any given point in the system, there are a set of hypotheses that are the top k hypotheses derived based on all observations through the last observation seen. Let us refer to these as $\text{Hyp}_1^{(i)}$ through $\text{Hyp}_k^{(i)}$. Each hypothesis consists of a set of multitracks. If there are P total targets in the system, there are up to P multitracks in each hypothesis. Let us refer to the multitracks as $\text{Mt}_I^{(i)}$.

To describe a new hypothesis, we need to define which multitracks are present in the new hypothesis and their multiplicity (i.e. how many vehicles in the multitrack are hypothesized to have moved to the location of the new observation). A simple way of viewing this is as a multiplicity vector. Each multitrack is assigned to a position within the vector. The vector contains additional slots for the possibility that vehicles within the observation are new targets (i.e. targets that have not been seen yet within the system). There are also slots for the OTHER case described earlier. We refer to this multiplicity vector as the vehicle count (or vc) vector.

For example, the vc vector would contain: ($Mt_1^{(i)}$, …, $Mt_m^{(i)}$, New $Target_1^{(i)}$, …, New $Target_j^{(i)}$, OTHER), where *m* represents the number of multitracks, *j* represents the number of valid target types for this observation.

There is a maximum value that can be assigned to each multitrack in the vc vector. This is described in a max_vc vector. The value for each multitrack is the smaller of the multitrack vehicle count and the observation vehicle count. Since the vc vector completely defines the hypothesis, the hypothesis probability can be generated based on the vc vector value, so we can describe this as P(vc).

For example, consider a pair of observations, one of 10 vehicles and another with 5 vehicles. Assume that we have set the pruning count *k* to 3. When we process the first observation, we have no prior state. We generate a vc vector which contains slots for the new target and for the false alarm: (New Target, OTHER). The max_vc vector is (10, 10), indicating that all of the vehicles can be attributed to the new target and the false alarm case. The possible vc solutions to this are: (10, 0), (9, 1), (8, 2), (7, 3), (6, 4), (5, 5), (4, 6), (3, 7), (2, 8), (1, 9) and (0, 10). Three of these will survive the pruning process. Let us arbitrarily pick (9, 1), (7, 3) and (2, 8) as the surviving hypotheses.

When the second observation is processed, it contains five vehicles. For each of the prior hypotheses, we need to consider all of the possible ways that the targets could be configured. Therefore, we create a vc vector which contains one multitrack (the prior observation connected to this observation), a slot for the new target, and a spot for the false alarm case: (Multitrack, New Target, False Alarm). The following table shows the prior observation hypothesis, the maximum vc vector, and the list of possible vc vector solutions.

Prior observation hypothesis	Max vc vector	Possible VC solutions
(9, 1)	(5, 5, 5)	(0, 0, 5), (0, 1, 4), (0, 2, 3), (0, 3, 2), (0, 4, 1), (0, 5, 0), (1, 0, 4), (1, 1, 3), (1, 2, 2), (1, 3, 1), (1, 4, 0), (2, 0, 3), (2, 1, 2), (2, 2, 1), (2, 3, 0), (3, 0, 2), (3, 1, 1), (3, 2, 0), (4, 0, 1), (4, 1, 0), (5, 0, 0)
(7, 3)	(5, 5, 5)	(0, 0, 5), (0, 1, 4), (0, 2, 3), (0, 3, 2), (0, 4, 1), (0, 5, 0), (1, 0, 4), (1, 1, 3), (1, 2, 2), (1, 3, 1), (1, 4, 0), (2, 0, 3), (2, 1, 2), (2, 2, 1), (2, 3, 0), (3, 0, 2), (3, 1, 1), (3, 2, 0), (4, 0, 1), (4, 1, 0), (5, 0, 0)
(2, 8)	(2, 5, 5)	(0, 0, 5), (0, 1, 4), (0, 2, 3), (0, 3, 2), (0, 4, 1), (0, 5, 0), (1, 0, 4), (1, 1, 3), (1, 2, 2), (1, 3, 1), (1, 4, 0), (2, 0, 3), (2, 1, 2), (2, 2, 1), (2, 3, 0)

The (2, 8) hypothesis has a maximum of 2 vehicles which can be attributed to the multitrack, since that is the multiplicity of the multitrack. As the number of multitracks increases, the number of vectors that exist in the solution space begins to increase according to the counting formulas described in section 3.

Our goal, then, is to define an algorithm that can generate the *k* best solutions without exhausting over the vc vector space. If we set $k = 1$, the problem can be described as an optimization problem:

- maximize *P*(vc) subject to the constraints that:
    1. Each element of the vc vector is an integer ≥ 0
    2. Each element of the vc vector $\leq$ the corresponding element of the max_vc vector
    3. The vc vector accounts for all vehicles in the new observation

To solve for an arbitrary *k*, we simply want to generate the *k* best solutions to this maximization problem.

One of the common approaches to solving a maximization problem with integer values is to use a branch and bound technique. The approach we took was a least cost branch and bound algorithm. The derivation of the solution as a least cost branch and bound algorithm will be described in more detail.

3.1. Least-Cost Branch and Bound

The least cost branch and bound algorithm is one of a family of related algorithms that include Backtracking and Branch and Bound approaches. Problems that are suited to this family of algorithms are problems that can be described as a search through a finite vector or tuple-space. The goal of these algorithms is to limit the number of tuples considered in finding a solution.

The Branch and Bound algorithms are special cases of the Backtracking approach. Backtracking algorithms construct a tree of partial solutions according to a defined rule. When an undesirable state is achieved within the tree, the algorithm "back tracks" to a prior state and continues from there. The crucial step in defining a backtracking algorithm is to define a good tree-construction rule that will cover the solution space.

The next step in defining the backtracking algorithm is to define the rules the algorithm will use to decide which node to expand next. Branch and bound algorithms require that when a node in the tree is expanded, all children of the node being considered will be generated before another node will be expanded. In particular, the least cost branch and bound algorithm requires that we be able to score each node according to the likelihood of it leading to a solution. This is, in fact, one of the points that brought the least cost branch and bound technique into consideration. Each leaf node is a hypothesis for which we can score the probability. Our goal is to generate the *k* highest probability leaf nodes, so this probability is the basis for scoring the nodes. The scoring function is based on the formulas derived in section 2 and will be discussed more fully later. The score of a node will be an upper bound for the score of nodes that can be derived from this node. The score of a leaf node will be its hypothesis probability.

According to the least cost branch and bound technique, we select the highest scoring solution thus far and extend it. If we select a leaf node, then this is our best solution, since no other node could possibly generate a solution with a higher score.

To extend this for our *k*-best case, we continue the least cost branch and bound algorithm until we have selected *k* leaf nodes. At this point, we have our *k*-best solutions, since all remaining interior and leaf nodes could not possibly lead to a solution that scores higher than our current *k* solutions.

To apply the least cost branch and bound technique to a problem we need to identify:

1. A way to construct the solution space as a tree of partial solutions which lead to complete solutions, and
2. A cost function that is an upper bound of the probability that can be achieved by a child of the node being scored.

Once these two things have been identified, the implementation of the algorithm is rather straightforward.

3.2. Constructing the solution space as a tree of partial solutions

Recall that the problem is described as defining different values for the vc vector, subject to the constraints of the max_vc vector and the number of vehicles in the new observation. To construct this as a tree of solutions, we need to define a mechanism for constructing the tree.

A couple of ways to generate the solution tree were considered. The chosen generation method takes the prior vc vector, finds the rightmost non-zero vector and generates all partial solutions that can be constructed by assigning targets to any one multitarget track to the right of this vector. For example, if

the partial solution vc vector looks like (0, 2, 0, 0, 0), and the vehicle count for the new observation is 4, the following solutions will be generated:

(0, 2, 2, 0, 0)*
(0, 2, 1, 0, 0)
(0, 2, 0, 2, 0)*
(0, 2, 0, 1, 0)
(0, 2, 0, 0, 2)*

The partial solutions indicated with a * are complete solutions. Note that the last column of the vc vector, if set, must be set to the remaining values, since the vc entries must total the total vehicle count in order to be a valid solution. The partial solution (0, 2, 0, 0, 1) could never result in a legitimate solution, since the vector total is not equal to the total vehicle count and nothing could be added to this vector.

3.3. The cost scoring function

The cost scoring function is the crucial element in defining the performance of the least cost branch and bound algorithm. A good cost scoring function can result in a great reduction of computational cost. A bad cost scoring function can result in generating a large number of unnecessary partial solutions.

The cost scoring function for a partial solution must be an upper bound of the probability that could be found for any solutions below this partial solution on the solution tree. This is the only way to guarantee that the algorithm will find the best solution. Otherwise, the algorithm may inadvertently prune a branch of the tree that contains the best solution.

Therefore, we need to identify a cost scoring function for a partial solution that defines an upper bound of all full solutions that could be generated from this partial solution. Recall equation (3):

$$P(\text{extended track set}) = P(\text{track set}) \cdot n! \cdot \prod_{i=0:2} \prod_{k=0:r^{(i)}} \binom{N_k^{(i)}}{n_k^{(i)}} \left\{ P(data / H_k^{(i)}) \right\}^{n_k^{(i)}} / S, \tag{3}$$

This equation can be expanded using separate factors for existing, new, and false alarm (OTHER) tracks as follows:

$$P(Hyp_k^{(t)}) = P(Hyp_k^{(t-1)}) \cdot n! \prod_{j=1:r} \binom{Mt_j}{vc_j} \left\{ P(track\ ext_j) \right\}^{vc_j} \cdot \prod_{i=1:2} \binom{N_0^{(i)}}{vc'_i} \left\{ P(new\ tar_i \right\}^{vc'_i} \cdot \binom{N^{(0)}}{vc'_0} \left\{ P(OTHER) \right\}^{vc'_0} / S, \tag{4}$$

where there are *r* existing target groups, each group of some type and having vc_j vehicles, whose tracks are being extended into the new observation; and possibly one new target group of each type (two here) in the new observation with vc'_i vehicles each; and the third factor is for false targets.

The cost function will know a partial set of vc_i, but not a full set. The known values can be inserted into this equation. The remaining terms need to be approximated. The first approximation made is to observe that $\binom{n}{k} x^k$ is bounded above by $(nx)^k$. This approximation is used for the unknown portions of this equation. So the largest value of *nx* for the remaining possible vc entries is selected and raised to the *k* power. This then becomes the value of the remaining factors providing an upper bound to the cost.

Some refinements to this cost function will be discussed later.

3.4. Least cost branch and bound implementation

The implementation of the least cost branch and bound is straightforward. An abstract container class implements a priority queue. The priority queue stores members that are en-queued in order using a comparison function. When an element is de-queued from the priority queue, the "best" element is returned, as implemented by the comparison function. For our implementation, the comparison function is a sorting by the highest probability that could possibly be generated from this node.

The actual algorithm then is straightforward:

1. Create the VC vector with all zeros and en-queue it
2. While the queue is not empty and we don't have *k* solutions yet
    a) De-queue an element
    b) If the element is a leaf node, generate a new solution
    c) If the element is not a leaf node, generate child solutions from this node and en-queue them.

3.5. Performance results

The software was run through a standard test-data set developed as part of the DMIF-1 testing work sponsored by DARPA. This test set contains several days worth of multi-vehicle observations of various target types moving around the battlefield. A subset of the results is presented here. This subset was selected around the observations that were most computationally expensive in the old software. The results here are representative of the results seen throughout the data. The value for *k* (the pruning value) is set to 20 for this data.

Number partial solutions considered	Total number of valid solutions
87	80
129	160
87	80
603	58,240
766	33,408
864	20,971,520
864	20,971,520
364	20,971,520
763	51,840
655	5,242,880
92	80
60	80
687	40,320
87	80

There are several interesting things about this data. Most obvious is that the overall number of partial solutions considered is greatly reduced. The algorithm has greatly reduced the amount of work required to generate the top *k* solutions for observations with many valid choices.

A peculiarity within the data is that some of the faster observations require the generation of more partial solutions than exhausting over all solutions would take. This is caused by the fact that the software counts the creation of partial solutions, including partial solutions that are added to the queue and never are expanded. Since the algorithm requires generating all partial solutions from a node, this number can be rather high. However, many of those partial solutions are never processed, they are simply generated and scored.

Also, the cost of doing the extra work in these cases is rather small, and is insignificant when compared to the computational savings found in generating only 864 partial solutions vice 20,971,520 total solutions.

4. Conclusions

The least cost branch and bound algorithm results in a significant reduction in the computational cost in generating solutions for multi-vehicle observations. The algorithm can be slower in some circumstances. Also, there is no guarantee that the reduction of computational cost will always occur. Like most algorithms that are applied to NP-complete problems, special cases exist that result in NP performance. Our testing has yet to run across this scenario in the tracking data. However, we can construct a scenario where the algorithm would result in exhaustion.

A post-development analysis identified some areas for improvement:

1. We should be able to detect the situations when the algorithm will result in exhausting over the solution space. In fact, in this case, there is very little difference in all of the resulting hypotheses, so some kind of short-circuiting of this anomalous condition should be feasible.

2. We can improve the cost function approximation. In fact, a lower upper bound for $\binom{n}{k}x^k$ than $(nx)^k$ is $((n-k)\,x)^k$. In addition, since the cost function (3) is essentially expressible as a multinomial probability, it is unimodal and we should be able to restrict the search for the maximum to a fairly small multidimensional box. The next *k-1* best costs could be found in a neighborhood of the maximum. This research work should improve the performance even further.

The algorithm relies on the fact that we are pruning hypotheses so that we keep only the k best solutions after every observation. The Tracker software, in fact, supports other techniques for keeping the number of hypotheses small. We can prune by hypothesis probability threshold. This could be implemented with a more sophisticated construct on the while loop. It is unclear how feasible this feature would actually be, since the number of hypotheses exceeding the threshold is likely to increase as the probabilities flatten out.

Another technique for managing the hypotheses is combining. Combining takes like hypotheses and combines them to reduce the number of hypotheses saved between observations. It should be feasible to apply a least cost branch and bound technique to a system that is doing combining. However, this is a rather significant research effort, and is currently not planned.

5. Acknowledgments

This work was sponsored by DARPA. Several other individuals at PSR were instrumental in this development, including Kevin Grottle, Serge Lubenec and Steve McKay.

6. References

[De96] DeWitt, R., Rappoport K., and Withers, L., "IBIS/Tracker: A Software Tool for Tracking Ground-based Time Critical Targets Using Military Domain Operations Models," *Proceedings of the 9th National Symposium on Sensor Fusion*, Monterey CA, March, 1996.

[Ho78] Horowitz, Ellis, Sahni, Sartaj, *Fundamentals of Computer Algorithms*, Computer Science Press, Rockville, MD, 1973.

[Ra96] Rappoport, K., "A High-Fidelity Behavior-Based TCT Motion Model with Fast Prediction and Scoring," *Proceedings of the 9th National Symposium on Sensor Fusion*, Monterey CA, March, 1996.

[Wi97] Withers, L., DeWitt, R., Grottle, K., Hastings, C., and Shearer, R., "A Unified Discrete Markov-Bayes Model for Tracking Target Vehicles with Moving and Stationary States," *Proceedings of the 10th National Symposium on Sensor Fusion*, Lexington, MA, April, 1997.