An Efficient Algorithm for Data Association in Multitarget Tracking

B. ZHOU, Member, IEEE
N. K. BOSE, Fellow, IEEE
The Pennsylvania State University

An efficient algorithm is developed to compute the *a posteriori* probabilities of the origins of measurements in the joint probabilistic data association filter (JPDAF). The inherited parallelism of this algorithm enables it to be suitable for implementation in a multiprocessor system. In this algorithm, the *a posteriori* probability of the origin of each measurement in the JPDAF is decomposed into two parts. The computation of one part becomes trivial and the algorithm developed here is implemented on the other part, which is shown to be related to permanents. The computational complexity of this algorithm is analyzed in the worst case as well as in the average case. An analysis of this algorithm enables us to conclude that this algorithm is more efficient than other existing ones in the average case.

Manuscript received September 1, 1992; revised December 10, 1993.

IEEE Log No. T-AES/31/1/08014.

This work was supported by a grant from SDIO/IST and managed by the Office of Naval Research under Contracts N00014-86-0542 and N00014-92-J-1755.

Authors' current addresess: B. Zhou, 102 Malott Hall, Cornell University, Ithaca, NY 14853; N. K. Bose, The Spatial and Temporal Signal Processing Center, Department of Electrical and Computer Engineering, Pennsylvania State University, University Park, PA 16802.

I. INTRODUCTION

In a cluttered environment, the received measurements may not all arise from the targets of interest. Some of them may be from clutter or false alarm. As a result, there always exist ambiguities in the association between the previous known targets and measurements. To solve the problem of data association between targets and measurements, three typical approaches have been reported in the literature. The first is called a *target-oriented* approach in which each measurement is assumed to have originated from either a known target or clutter, as in the *joint probabilistic data association filter* (JPDAF) [1, 2]. The second is called a *measurement-oriented* approach in which each measurement is hypothesized to have originated from either a known target, a new target, or clutter [3]. The third approach is referred to as *track-oriented* where each track is hypothesized to be either undetected, terminated, associated with a measurement, or linked to the start of a maneuver [4–6]. In these approaches, the number of data association hypotheses could increase rapidly with the increase in the number of targets and the number of measurements. Therefore, in a multitarget tracking algorithm, the computational cost in generating the data association hypotheses would be excessive when the number of targets and the number of measurements are large.

In recent years, a lot of attention has been given to the task of improving the computational efficiency of a multitarget tracking algorithm. Fitzgerald [7] proposed an *ad hoc* formula to approximately compute the β_j^ts in the JPDAF. For $j > 0$, β_j^t is the *a posteriori* probability that measurement j originated from target t. For $j = 0$, β_0^t is the *a posteriori* probability that no measurement originated from target t. Fitzgerald's formula breaks down when four targets are required to be tracked [8]. Sengupta and Iltis [8, 9] developed an analog neural network to emulate the JPDAF. They showed that the neural network is capable of handling two to six targets and three to twenty measurements. However, the heuristic nature of their approach makes implementation difficult [10]. Alternatively, Nagarajan, et al [11], arranged the hypotheses in the measurement-oriented approach [3] in a special order so that the probabilities of the hypotheses are proportional to the product of certain probability factors already evaluated. The algorithm locates the N globally best hypotheses without evaluating all of them. In [12], Fisher and Casasent developed a fast joint probabilistic data association (JPDA) algorithm. In their algorithm, the computation of the β_j^ts was implemented using enormous number of vector inner product operations. Since the vector inner product operation could be easily realized on an optical processor, they proposed a specialized optical processor to approximately implement the fast JPDA algorithm. Recently, Zhou and Bose [13] proposed a depth-first search (DFS) approach to efficiently compute the β_j^ts in the JPDAF. Their algorithm requires much less computation than the fast JPDA algorithm in the average case, when there are, on average, two to three measurements inside the validation gate of a target. The validation gate of a target is a region with a prescribed size around the predicted position of the target.

As shown in [13, 14], the performance of an approximation of the JPDAF degrades drastically when the density of targets is high. Therefore, it is important to implement the JPDAF accurately in a dense target environment. Among the algorithms mentioned above, only the two algorithms proposed in [12, 13] have the potential for accurately computing the β_j^ts in the JPDAF. The fast JPDA algorithm in [12] is suitable for implementation in a multiprocessor system. Nevertheless, the computational cost is very high when the number of targets and the number of measurements are large since the design of the algorithm is based on the worst case scenario. In the worst case, all measurements fall inside the intersection of the validation gates of all targets. Although the DFS algorithm [13] is much more efficient than the fast JPDA algorithm in the average case, it is specifically directed towards implementation through a system with a powerful centralized processor, normally used in ground-based tracking systems. Our objective here is to propose an algorithm which not only performs better than the DFS algorithm in the average case, but which is also suitable for implementation in a tracking system with several spatially distributed microprocessors.

A brief review of the JPDAF is given in Section II, followed by the problem formulation in Section III. The new algorithm is developed in Section IV. In Section V, the computational complexity of the new algorithm is analyzed in the worst case as well as in the average case. Finally, the advantages of the new algorithm are summarized in Section VI.

II. REVIEW OF THE JPDAF

We assume that there are n targets being tracked at time index k. Let the dynamic model for target t be described by

$$x^t(k+1) = F^t(k)x^t(k) + G^t(k)w^t(k) \tag{1}$$

$$z^t(k) = H^t(k)x^t(k) + v^t(k), \qquad t = 1,2,\ldots,n \tag{2}$$

where $x^t(k)$ is the N^t-dimensional state vector, $z^t(k)$ is the M^t-dimensional measurement vector with M^t actually independent of t, $F^t(k)$, $G^t(k)$ and $H^t(k)$ are known model matrices, $w^t(k)$ and $v^t(k)$ are, respectively, the p^t and M^t-dimensional noise vectors, which are assumed to be zero-mean independent identically distributed Gaussian processes with known covariances:

$$E\{w^t(k)(w^t(j))'\} = Q^t(k)\delta_{kj} \tag{3}$$

$$E\{v^t(k)(v^t(j))'\} = R^t(k)\delta_{kj} \tag{4}$$

where $\delta_{kj} = 1$ if $k = j$ and $\delta_{kj} = 0$ otherwise. The prime superscript indicates matrix transposition. The initial target state $x^t(0)$ is assumed to be normally distributed with mean $\hat{x}^t(0 \mid 0)$ and covariance $P^t(0 \mid 0)$. It is also assumed to be independent of $w^t(k)$ and $v^t(k)$ for all $k \geq 0$.

Suppose that m measurements are received at time index k. In a cluttered environment, m does not necessarily equal n and it may be difficult to distinguish whether a measurement originated from a target or from clutter. It is reasonable to denote a measurement at time index k by

$$z(k) = \begin{cases} z^t(k) & \text{if } z(k) \text{ is from target } t \\ z^c(k) & \text{if } z(k) \text{ is from clutter.} \end{cases} \tag{5}$$

The measurement $z^c(k)$ is usually assumed to be uniformly distributed in the surveillance region and the number of clutter is subject either to Poisson distribution or to uniform distribution [1]. In this paper, Poisson distribution is assumed for the purpose.

In the JPDAF, the *a posteriori* probability β_j^t is computed. For $j > 0$, β_j^t is the *a posteriori* probability that validated measurement j originated from target t. β_0^t is the *a posteriori* probability that no measurement originated from target t. A validated measurement is one which is either inside or on the boundary of the validation gate of a target. Mathematically, a validation gate is defined by

$$(z(k) - \hat{z}^t(k))'S^t(k)^{-1}(z(k) - \hat{z}^t(k)) \leq g^2 \tag{6}$$

where $\hat{z}^t(k)$ is the predicted value of $z(k)$ for target t. The error, $(z(k) - \hat{z}^t(k))$, is the innovation generated from $z(k)$ for target t, $S^t(k)$ is its covariance matrix, and g is a selected threshold. As discussed in [2, 15], the choice $g > \sqrt{M^t} + 2$ ensures that the correct measurements will lie within the gate with probability 0.999. The dimension of $z(k)$ is M^t. The inequality given in (6) is said to generate a validation test. The result of the test is kept in what is called a validation matrix Ω. This validation matrix Ω is a $m \times (n+1)$ rectangular matrix defined as [2],

$$\Omega = [\omega_{jt}] = \begin{array}{c} \overbrace{\begin{array}{ccccc} 0 & 1 & 2 & \cdots & n \end{array}}^{t} \\ \begin{pmatrix} \omega_{10} & \omega_{11} & \omega_{12} & \cdots & \omega_{1n} \\ \omega_{20} & \omega_{21} & \omega_{22} & \cdots & \omega_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \omega_{m0} & \omega_{m1} & \omega_{m2} & \cdots & \omega_{mn} \end{pmatrix} \end{array} \left.\begin{array}{c} 1 \\ 2 \\ \vdots \\ m \end{array}\right\} j \tag{7}$$

where $\omega_{j0} = 1$ for $j = 1,2,\ldots,m$, $\omega_{jt} = 1$ if measurement j is inside the validation gate of target t and $\omega_{jt} = 0$ otherwise for $j = 1,2,\ldots,m$, and $t = 1,2,\ldots,n$. Based on the validation matrix Ω, data association hypotheses (or feasible events $\mathcal{E}$s [2]) are generated subject to the following two restrictions:

1) Each measurement can have only one origin (either a specific target or clutter).

2) No more than one measurement originates from a target.

This leads to a combinatorial problem where the number of data association hypotheses increases exponentially with n and m.

Each feasible event $\mathcal{E}$ is represented by a hypothesis matrix $\hat{\Omega}$ in [2]. $\hat{\Omega}$ has the same size as the validation matrix Ω. A typical element in $\hat{\Omega}$ is denoted by $\hat{\omega}_{jt}$ where

$$\hat{\omega}_{jt} = \begin{cases} 1 & \text{if } \omega_{jt} = 1 \text{ and measurement } j \text{ is hypothesized to be from clutter for } t = 0 \\ 1 & \text{if } \omega_{jt} = 1 \text{ and measurement } j \text{ is hypothesized to be associated with target } t \text{ for } t \neq 0 \\ 0 & \text{otherwise.} \end{cases} \quad (8)$$

Corresponding to the two restrictions above, in $\hat{\Omega}$, there is at most one unit element in each column except for $t = 0$ and there is exactly one unit element in each row.

After each $\hat{\Omega}$ is obtained, the conditional probability of the corresponding data association hypothesis or feasible event is calculated by a formula given in [2]. A simplified version of this formula is given as

$$P(\mathcal{E}(\hat{\Omega}) \mid \mathcal{Z}) = \frac{1}{c}(P_0)^{\min(n,m)-m_d} \prod_{j:\hat{\omega}_{jt}=1} P_j^t \quad (9)$$
$$\text{for} \quad j = 1,2,\ldots,m, \quad \text{and} \quad t = 1,2,\ldots,n$$

where $\mathcal{Z}$ is the set of all measurements received up to current time index k, c is a normalizing constant, m_d is the number of targets detected in this feasible event $\mathcal{E}$, $\hat{\omega}_{jt} = 1$ indicates that measurement j is associated with target t in the event, and

$$P_j^t = \begin{cases} N(\tilde{z}_j^t; 0, S^t)P_D & \text{if} \quad \omega_{jt} = 1 \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

$$P_0^t = \lambda(1 - P_D) = P_0$$
$$\text{for} \quad j = 1,2,\ldots,m, \quad \text{and} \quad t = 1,2,\ldots,n. \quad (11)$$

In (10) and (11), λ is the clutter density, P_D is the probability of detection, and $N(\tilde{z}_j^t; 0, S^t)$ is a normal density function having zero mean and a covariance matrix S^t with

$$\tilde{z}_j^t(k) = z_j(k) - H^t(k)\hat{x}^t(k \mid k-1). \quad (12)$$

In (12), $H^t(k)$ is the observation matrix for target t as defined in (2) and $\hat{x}^t(k \mid k-1)$ is the one-step prediction of the state estimate of target t. It is understood that the normalizing constant c in (9) is obtained by the summation, $\sum_{\mathcal{E}(\hat{\Omega})} P(\mathcal{E}(\hat{\Omega}) \mid \mathcal{Z})$. Therefore, c is omitted hereafter. The *a posteriori* probability β_j^t is computed from the conditional probabilities in (9) by

$$\beta_j^t = \sum_{\mathcal{E}(\hat{\Omega})} P(\mathcal{E}(\hat{\Omega}) \mid \mathcal{Z})\hat{\omega}_{jt} \quad (13)$$

$$\beta_0^t = 1 - \sum_{j=1}^{m} \beta_j^t$$
$$\text{for} \quad j = 1,2,\ldots,m, \quad \text{and} \quad t = 1,2,\ldots,n. \quad (14)$$

The equations of the JPDAF are the same as those of a standard Kalman filter with the following two changes.

1) The innovation vector is generated by using relation:

$$\tilde{z}^t(k) = \sum_{j=1}^{m} \beta_j^t(k)\tilde{z}_j^t(k). \quad (15)$$

2) The update equation for the covariance matrix $P^t(k,k \mid k)$ is changed to

$$\begin{aligned} P^t(k,k \mid k) = {} & \beta_0^t(k)P^t(k,k \mid k-1) \\ & + (1 - \beta_0^t(k))P^{t*}(k,k \mid k) \\ & + K^t(k)W^t(k)K^t(k)' \end{aligned} \quad (16)$$

where $K^t(k)$ is the Kalman gain for target t,

$$P^{t*}(k,k \mid k) = [I - K^t(k)H^t(k)]P^t(k,k \mid k-1) \quad (17)$$

denotes the covariance update for a single correct return, and

$$W^t(k) = \sum_{j=1}^{m} \beta_j^t(k)\tilde{z}_j^t(k)\tilde{z}_j^t(k)' - \tilde{z}^t(k)\tilde{z}^t(k)'. \quad (18)$$

In general, a covariance matrix $P^t(p,q \mid r)$ is defined as

$$P^t(p,q \mid r) = E\{[x^t(p) - \hat{x}^t(p \mid r)][x^t(q) - \hat{x}^t(q \mid r)]'\} \quad (19)$$

where $\hat{x}^t(p \mid r) = E\{x(p) \mid \text{measurements received up to time index } r\}$.

In the next section, the computation of the β_j^ts is reformulated in such a way that the construction of the hypothesis matrices $\hat{\Omega}$ is not necessary.

III. PROBLEM FORMULATION

In the original JPDAF [2], the computation of β_j^t consists of the following three steps.

1) Construct a validation matrix Ω and compute P_j^t using (10) and (11).

2) Generate data association hypotheses and compute $P(\mathcal{E}(\hat{\Omega}) \mid \mathcal{Z})$ using (9).

3) Compute β_j^t using (13) and (14).

In this section, each β_j^t is decomposed into two parts so that its computations does not require the generation of the data association hypotheses or the construction of the hypothesis matrices as in the original JPDAF [2] and the DFS algorithms [13]. Let $\mathcal{A}$ denote an ordered set of nonzero integers associated with the targets. Let $\mathcal{B}$ denote another ordered set of non-zero integers which are the indices of measurements. The symbol 0 is the index for clutter. Suppose that there are n targets and m measurements are received at a certain time instant. Then,

$$\mathcal{A} = \{1,2,\ldots,n\} \tag{20}$$

$$\mathcal{B} = \{0,1,\ldots,m\}. \tag{21}$$

With the above notation, (13) may be rewritten as

$$\beta_j^t(\mathcal{A},\mathcal{B}) = \sum_{\mathcal{E}(\hat{\Omega})} P(\mathcal{E}(\hat{\Omega}) \mid \mathcal{Z})\hat{\omega}_{jt} \qquad \text{for} \quad j = 1,2,\ldots,m, \quad \text{and} \quad t = 1,2,\ldots,n. \tag{22}$$

Since P_j^t is a factor of the $P(\mathcal{E}(\hat{\Omega}) \mid \mathcal{Z})$s when $\hat{\omega}_{jt} = 1$, β_j^t may be decomposed as

$$\beta_j^t(\mathcal{A},\mathcal{B}) = P_j^t F(\mathcal{A}\backslash t,\mathcal{B}\backslash j) \qquad \text{for} \quad j = 1,2,\ldots,m, \quad \text{and} \quad t = 1,2,\ldots,n \tag{23}$$

where the difference set $\mathcal{C}\backslash l$ is the set derived from $\mathcal{C}$ by deleting element l. In (23), $F(\mathcal{A}\backslash t,\mathcal{B}\backslash j)$ is a function of P_l^τs for $l \neq j$ and $\tau \neq t$. An interpretation of $F(\mathcal{A}\backslash t,\mathcal{B}\backslash j)$ may be obtained using the definition of $P(\mathcal{E}(\hat{\Omega}) \mid \mathcal{Z})$ [2].

For $j = 1,2,\ldots,m$, and $t = 0,1,2,\ldots,n$, let $\mathcal{E}_{jt}$ denote the hypothesis that measurement j originated either from target t if $t \neq 0$ or from clutter if $t = 0$. Then, each feasible event $\mathcal{E}$ may be written as [2]

$$\mathcal{E}(\hat{\Omega}) = \bigcap_{j=1}^{m} \mathcal{E}_{jp(t)}(\hat{\omega}_{jp(t)}) \tag{24}$$

where $p(t)$ denotes an element of the set of permutations of $\{t\} \overset{\Delta}{=} \{0,1,2,\ldots,n\}$. Furthermore, let $\hat{\Omega}'_{(jt)}$ be the hypothesis matrix obtained from $\hat{\Omega}$ after eliminating the jth row and the tth column. $P(\mathcal{E}_{jt}(\hat{\omega}_{jt}) \mid \mathcal{Z})$ is the probability of the hypothesis that measurement j originated from target t subject to the assumption that data association between the remaining measurements and remaining targets is ignored, and $\sum_{\mathcal{E}(\hat{\Omega}'_{(jt)})} P(\mathcal{E}(\hat{\Omega}'_{(jt)}) \mid \mathcal{E}_{jt}(\hat{\omega}_{jt}),\mathcal{Z})$ denotes the probability of all the data association hypotheses among the remaining measurements and targets conditioned on the hypothesis that measurement j is associated with target t. Since all the feasible events $\mathcal{E}$s are mutually exclusive, (22) can be rearranged as below.

$$\begin{aligned}
\beta_j^t(\mathcal{A},\mathcal{B}) &= \sum_{\mathcal{E}(\hat{\Omega})} P(\mathcal{E}(\hat{\Omega}) \mid \mathcal{Z})\hat{\omega}_{jt} \\
&= P\left(\bigcup_{\mathcal{E}(\hat{\Omega}):\hat{\omega}_{jt}=1} \mathcal{E}(\hat{\Omega}) \mid \mathcal{Z} \right) \\
&= P\left(\mathcal{E}_{jt}(\hat{\omega}_{jt}) \Leftrightarrow \left(\bigcup_{\mathcal{E}(\hat{\Omega}):\hat{\omega}_{jt}=1} \mathcal{E}(\hat{\Omega}) \right) \mid \mathcal{Z} \right) \\
&= P(\mathcal{E}_{jt}(\hat{\omega}_{jt}) \mid \mathcal{Z}) \\
&\quad \times P\left(\bigcup_{\mathcal{E}(\hat{\Omega})} \left(\bigcap_{\substack{l\neq j \\ \tau\neq t}} \mathcal{E}_{l\tau}(\hat{\omega}_{l\tau}) \right) \mid \mathcal{E}_{jt}(\hat{\omega}_{jt}),\mathcal{Z} \right) \\
&= P(\mathcal{E}_{jt}(\hat{\omega}_{jt}) \mid \mathcal{Z}) P\left(\bigcup_{\mathcal{E}(\hat{\Omega}'_{(jt)})} \mathcal{E}(\hat{\Omega}'_{(jt)}) \mid \mathcal{E}_{jt}(\hat{\omega}_{jt}),\mathcal{Z} \right) \\
&= P(\mathcal{E}_{jt}(\hat{\omega}_{jt}) \mid \mathcal{Z}) \sum_{\mathcal{E}(\hat{\Omega}'_{(jt)})} P(\mathcal{E}(\hat{\Omega}'_{(jt)}) \mid \mathcal{E}_{jt}(\hat{\omega}_{jt}),\mathcal{Z})
\end{aligned}$$

$$\text{for} \quad j = 1,2,\ldots,m, \quad \text{and} \quad t = 1,2,\ldots,n. \tag{25}$$

Comparing (25) with (23), we have

$$P_j^t = P(\mathcal{E}_{jt}(\hat{\omega}_{jt}) \mid \mathcal{Z}) \tag{26}$$

$$F(\mathcal{A}\backslash t,\mathcal{B}\backslash j) = \sum_{\mathcal{E}(\hat{\Omega}'_{(jt)})} P(\mathcal{E}(\hat{\Omega}'_{(jt)}) \mid \mathcal{E}_{jt}(\hat{\omega}_{jt}),\mathcal{Z}). \tag{27}$$

Therefore, $F(\mathcal{A}\backslash t,\mathcal{B}\backslash j)$ in (23) contains the sum of the conditional probabilities of the data association hypotheses among all the targets, the measurements, and clutter, given that measurement j is associated with target t. In the case of the probabilistic data association filter (PDAF) [15], where P_j^t differs from the *a posteriori* probability, β_j^t, by a normalization constant, $F(\mathcal{A}\backslash t,\mathcal{B}\backslash j)$ may be considered to be due to the interference from other targets on β_j^t if (23) is compared with its counterpart in the PDAF.

Similarly, β_0^t may be decomposed as

$$\beta_0^t = P_0 F(\mathcal{A}\backslash t,\mathcal{B}), \qquad \text{for} \quad t = 1,2,\ldots,n \tag{28}$$

where $F(\mathcal{A}\backslash t,\mathcal{B})$ contains the sum of the conditional probabilities of the data association hypotheses among all the targets, the measurements, and clutter, given that target t is not associated with any measurement. In (28), $F(\mathcal{A}\backslash t,\mathcal{B})$ may also be considered as the interference on β_0^t from other targets. The computation of β_j^t after reformulation can now be summarized in three steps as given below.

1) Construct a validation matrix Ω and compute P_j^t using (10) and (11).
2) Compute $F(\mathcal{A}\backslash t,\mathcal{B}\backslash j)$ and $F(\mathcal{A}\backslash t,\mathcal{B})$.
3) Compute β_j^t using (23) and (28).

To compute β_0^t for $t = 1,\ldots,n$, using (14) instead of (28), the β_j^ts need to be normalized. The β_j^ts computed using (23) and (28) are not normalized. The normalizing constant c is determined by the following relationship.

$$\sum_{j=0}^{m}\beta_j^t(\mathcal{A},\mathcal{B}) = c, \qquad \text{for} \quad t = 1,2,\ldots,n. \tag{29}$$

Suppose that c is computed in the $t = 1$ case as

$$\sum_{j=0}^{m}\beta_j^1(\mathcal{A},\mathcal{B}) = c. \tag{30}$$

The same constant also applies, in all related cases where $t \neq 1$. Then, β_j^1 for $j = 0,1,\ldots,m$, are normalized as below

$$\beta_j^1 \leftarrow \frac{1}{c}\beta_j^1. \tag{31}$$

Using c, β_j^t, for $t = 2,\ldots,n$, and $j = 1,\ldots,m$, can also be normalized as the way the β_j^1s do. Now, β_0^t for $t = 2,\ldots,n$, are ready to be computed using (14). In the next section, an algorithm is developed to compute $F(\mathcal{A}\backslash t,\mathcal{B}\backslash j)$ for $t = 1,\ldots,n$, and $j = 1,\ldots,m$, and $F(\mathcal{A}\backslash 1,\mathcal{B})$.

IV. ALGORITHM DEVELOPMENT

From the notation introduced in the last section, $F(\mathcal{A},\mathcal{B})$ denotes the sum of the conditional probabilities of the data association hypotheses among the targets in $\mathcal{A}$ and the measurements in $\mathcal{B}$. Using (22) and (27), we have

$$\begin{aligned} F(\mathcal{A},\mathcal{B}) &= \sum_{\mathcal{E}(\hat{\Omega})} P(\mathcal{E}(\hat{\Omega}) \mid \mathcal{Z}) \\ &= \sum_{j=0}^{m}\beta_j^t(\mathcal{A},\mathcal{B}). \end{aligned} \tag{32}$$

After substituting (23) and (28) into (32), a recurrence relation for $F(\mathcal{A},\mathcal{B})$ can be obtained. For $t \in \mathcal{A}$,

$$F(\mathcal{A},\mathcal{B}) = P_0F(\mathcal{A}\backslash t,\mathcal{B}) + \sum_{j\in\mathcal{B}\backslash 0} P_j^tF(\mathcal{A}\backslash t,\mathcal{B}\backslash j). \tag{33}$$

The algorithm for computing $F(\mathcal{A},\mathcal{B})$ is obtained by recursive implementation of (33). The initial conditions associated with (33) are given below.

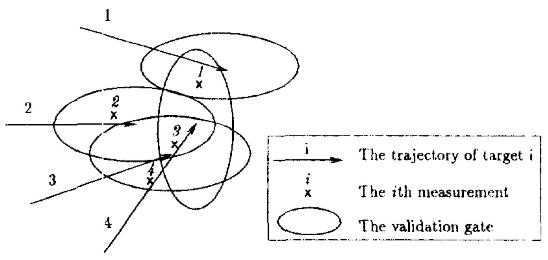


Fig. 1. Typical layout of 4 targets and 4 measurements.

1) If t is the only element in $\mathcal{A}$, then,

$$F(\mathcal{A},\mathcal{B}) = \sum_{j\in\mathcal{B}} P_j^t. \tag{34}$$

Note that $P_0^t = P_0$ in the above equation according to the notation introduced in (11).

2) If $j = 0$ is the only element in $\mathcal{B}$, then,

$$F(\mathcal{A},\mathcal{B}) = (P_0)^{|\mathcal{A}|} \tag{35}$$

where $|\mathcal{A}|$ denotes the number of elements in $\mathcal{A}$.

3) If $|\mathcal{B}| = 2$ and $|\mathcal{A}| \geq |\mathcal{B}|$, then,

$$F(\mathcal{A},\mathcal{B}) = (P_0)^{|\mathcal{A}|-1}\left(P_0 + \sum_{\substack{t\in\mathcal{A}\\ j\in\mathcal{B}\backslash 0}} P_j^t\right). \tag{36}$$

In order to demonstrate how the recursive algorithm works, consider an example.

EXAMPLE Suppose that there are 4 closely spaced targets under surveillance. On one radar scan, four measurements are received as shown in Fig. 1. Then,

$$\mathcal{A} = \{1,2,3,4\} \tag{37}$$

$$\mathcal{B} = \{0,1,2,3,4\}. \tag{38}$$

The conventional validation matrix occurring in the JPDAF for this example is given in (39)

$$\Omega = \begin{array}{c} \overbrace{\begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array}}^{t} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \left.\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array}\right\} j. \tag{39}$$

For target 1, there is only one measurement which falls inside its validation gate. Therefore, only β_0^1 and β_1^1 are none zero. Using (23) and (28), β_0^1 and β_1^1 may be computed as

$$\beta_0^1(\mathcal{A},\mathcal{B}) = P_0F(\mathcal{A}\backslash 1,\mathcal{B}) \tag{40}$$

$$\beta_1^1(\mathcal{A},\mathcal{B}) = P_1^1F(\mathcal{A}\backslash 1,\mathcal{B}\backslash 1) \tag{41}$$

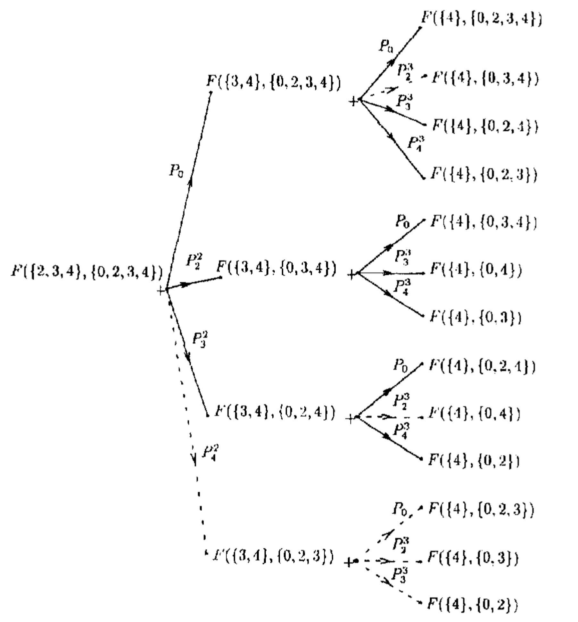


Fig. 2. Illustration of example using recursive algorithm.

where

$$\mathcal{A}\backslash 1 = \{2,3,4\} \tag{42}$$

$$\mathcal{B}\backslash 1 = \{0,2,3,4\}. \tag{43}$$

Equations (40) and (41) reduce the computations of β_0^1 and β_1^1 to those of $F(\mathcal{A}\backslash 1,\mathcal{B})$ and $F(\mathcal{A}\backslash 1,\mathcal{B}\backslash 1)$.

Since, in this example, $\mathcal{A}\backslash 1 = \{2,3,4\}$ and $\mathcal{B}\backslash 1 = \{0,2,3,4\}$, the computation of $F(\mathcal{A}\backslash 1,\mathcal{B}\backslash 1)$ may be performed step by step using (33) as shown through the directed tree in Fig. 2. In Fig. 2, the recursive procedure for computing $F(\{2,3,4\}, \{0,2,3,4\})$ is described by a directed tree. In this tree, the P_j^ts are the weights associated with the edges. The plus sign, $+$, at a node represents the summation operation. The F function at a node labelled with a plus sign is obtained from the weighted summation of the F functions at the sons (defined to be nodes which have edges incident from the node under consideration) of the node, as seen from (33). Let a leaf denote a node which does not have sons. Each F at a leaf of the tree is directly computed using the initial condition given in (34). In the computation of $F(\{2,3,4\},\{0,2,3,4\})$, in this example, the nodes in the tree shown in Fig. 2 are visited via DFS. A node is not visited if the only branch incident on it has zero weight. As a result, the sons of this node are also not visited. In Fig. 2, the nodes having only incident dashed branches are not visited. In other words, the dashed branches in the tree are pruned because their weights are zero.

$F(\mathcal{A}\backslash 1,\mathcal{B})$ may be computed in a similar way as discussed above. Generally speaking, the computation in the recursive algorithm developed here is performed from top to bottom (or from left to right as shown in Fig. 2). In the average case scenario, a lot of branches in a directed tree have zero weight, such as the one shown in Fig. 2. The computational cost can be saved if a directed tree is visited from top to bottom since the nodes which have branches with zero weights incident onto them are not visited. In the worst case scenario, however, no branches have zero weight. The overall computation may be reduced if the nodes having the same F are merged so that they are visited only once. As shown in Fig. 2, almost every F appears twice at the leaves of the tree if the F functions at the nodes which have branches with zero weights incident on to them are computed. If the computation is performed from bottom to top, it is obvious that the computational cost can be reduced in the worst case scenario. However, in the average case scenario, the majority of the Fs at the leaves of a tree appears only once after the dashed branches are pruned, as shown in Fig. 2. Therefore, a lot of the computation might be wasted if the computation is performed from bottom to top since many nodes which are connected with dashed branches do not contribute to the F at the root of a tree. It is shown in the next section that the computational cost is lower if the computation is perform from top to bottom in the average case. The computation in the fast JPDA algorithm [12] may be interpreted as being performed from bottom to top if the procedure is organized into the same tree structure as discussed above. Detailed comparison between the recursive algorithm developed here and the fast JPDA algorithm will be given below.

V. COMPUTATIONAL COMPLEXITY ANALYSIS

The computational complexity measure here is defined in terms of the numbers of multiplications, $M(n,m)$, and additions, $A(n,m)$, used for calculating the β_j^ts in the JPDAF, where n is the number of targets and m is the number of measurements. Since the computational cost for normalizing β_j^t is negligible when compared with that of $F(\mathcal{A}\backslash t,\mathcal{B}\backslash j)$, multiplications and additions for normalizing β_j^t are not included in $M(n,m)$ and $A(n,m)$. In the worst case, the m measurements received fall inside the intersection of the validation gates of the n targets. Therefore, each measurement may be associated with either any one of the n targets or clutter. In the average case, however, it is assumed that there are on average two to three measurements which fall inside the validation gate of each target. In the following, the worst case analysis is given first. Then, the average case analysis is discussed.

A. Worst Case Analysis

In the worst case, no branch is pruned if the computation process is represented in a tree structure as shown in Fig. 2, without any dashed edges. As discussed in Section III, to compute all β_j^t, only $F(\mathcal{A}\backslash t,\mathcal{B}\backslash j)$ for $t = 1,\ldots,n$, and $j = 1,\ldots,m$, and $F(\mathcal{A}\backslash 1,\mathcal{B})$ need to be computed. Therefore, $M(n,m)$ and $A(n,m)$ are, respectively, the number of multiplications and additions which suffice for computing nm β_j^ts ($j \neq 0$) and one β_0^t. In the recursive algorithm proposed in the previous section, the computation of the β_j^ts is based on (23) and (28). As shown there, the only operation required is one multiplication for computing each β_j^t after the corresponding F is computed. Therefore,

$$M(n,m) = nmM'(n-1,m) + M'(n-1,m+1) + nm + 1 \quad (44)$$

and

$$A(n,m) = nmA'(n-1,m) + A'(n-1,m+1) \quad (45)$$

where $M'(p,q)$ and $A'(p,q)$ denote, respectively, the numbers of multiplications and additions which suffice for computing the function $F(\mathcal{A},\mathcal{B})$ with $p = |\mathcal{A}|$ and $q = |\mathcal{B}|$.

In (33), there are q multiplications and $q-1$ additions used in the computation of $F(\mathcal{A},\mathcal{B})$ after $(q-1)$ $F(\mathcal{A}\backslash t,\mathcal{B}\backslash j)$s ($j \neq 0$) and one $F(\mathcal{A}\backslash t,\mathcal{B})$ are computed. Therefore,

$$M'(p,q) = (q-1)M'(p-1,q-1) + M'(p-1,q) + q \quad (46)$$

$$A'(p,q) = (q-1)A'(p-1,q-1) + A'(p-1,q) + q - 1 \quad (47)$$

with boundary conditions,

$$M'(1,q) = 0 \quad (48)$$

$$M'(p,2) = p-1, \qquad \text{for} \quad p \geq 2 \quad (49)$$

$$M'(p,1) = p-1 \quad (50)$$

$$A'(1,q) = q-1 \quad (51)$$

$$A'(p,2) = p, \qquad \text{for} \quad p \geq 2 \quad (52)$$

$$A'(p,1) = 0. \quad (53)$$

The boundary conditions for (46) and (47) are obtained from (34), (35), and (36).

In Table I, some sample values of $M(n,m)$ and $A(n,m)$ are provided. Comparing the listed values in Table I with the corresponding entries in Table II, it is not surprising to find that the recursive algorithm requires more multiplications and additions to compute the β_j^ts than the fast JPDA algorithm [12] in the worst case. This is because some Fs are computed more than once in the recursive algorithm as discussed in the previous section. However, the recursive algorithm is expected to perform much better than the fast JPDA algorithm in the average case. More discussion on the average case analysis will be given in the sequel.

TABLE I
Sample Values of $M(n,m)/A(n,m)$ in the Recursive Algorithm

n	m	$M(n,m)/A(n,m)$	n	m	$M(n,m)/A(n,m)$
3	6	134/582	5	10	43644/283900
4	8	2195/11952	6	12	1034045/7807860

TABLE II
Sample Values of $M(n,m)/A(n,m)$ in Fast JPDA Algorithm

n	m	$M(n,m)/A(n,m)$	n	m	$M(n,m)/A(n,m)$
3	6	150/396	5	10	10570/22100
4	8	1404/3232	6	12	70044/136584

TABLE III
Sample Values of $M(n,m)/A(n,m)$ in DFS Algorithm

n	m	$M(n,m)/A(n,m)$	n	m	$M(n,m)/A(n,m)$
3	6	318/787	5	10	96890/339041
4	8	5072/14849	6	12	2218992/9081085

In the worst case, the comparison between the computational complexity of the recursive algorithm and the DFS algorithm is also based on $M(n,m)$ and $A(n,m)$. In Table III, some sample values of $M(n,m)$ and $A(n,m)$ required for implementing the DFS algorithm are listed. Comparing Table III with Table I, it can be inferred that the $M(n,m)$s in the DFS algorithm are more than twice as large as those in the recursive algorithm and that the $A(n,m)$s in the DFS algorithm are always larger than those in the recursive algorithm. Considering the fact that the recursive algorithm is suitable for implementation in a multiprocessor system, the computational cost of the recursive algorithm could be much less than that of the DFS algorithm in the worst case.

The computational complexity given above is analyzed in terms of operational counts. It can also be given in "big O" notation. If the algorithm developed here is implemented in a system with a floating point unit, multiplication and addition operations would take about the same amount of time. Let $G(n,m) = M(n,m) + A(n,m)$. According to the definition given in [16, p. 2]AVA, the computational complexity of our algorithm in the worst case scenario is $O(G(n,m))$.

B. Average Case Analysis

In order to simplify the discussion, it is assumed that there are at most three measurements inside the validation gate of each target. This number was also selected in the examples given in [2, 8]. As a result, there are at most $3n$ β_j^ts ($j \neq 0$) and one β_0^t which need be computed. After the interference part of each β_j^t is obtained, $3n+1$ multiplications are sufficient for computing the β_j^ts as evident from (23) and (28).

TABLE IV
Upper Bounds of $M(n,m)$ and $A(n,m)$ in the Recursive Algorithm

n	m	$M(n,m)/A(n,m)$	n	m	$M(n,m)/A(n,m)$
3	-	50/150	5	-	1360/4080
4	-	273/819	6	-	6479/19437

Therefore, (44) and (45) lead to the inequalities,

$$M(n,m) \leq 3nM'(n-1,m) + M'(n-1,m+1) + 3n + 1 \quad (54)$$

$$A(n,m) \leq 3nA'(n-1,m) + A'(n-1,m+1). \quad (55)$$

In the average case, there are at most 4 multiplications and 3 additions required in the computation of $F(\mathcal{A},\mathcal{B})$ after 3 or less $F(\mathcal{A}\backslash t,\mathcal{B}\backslash j)$'s ($j \neq 0$) and one $F(\mathcal{A}\backslash t,\mathcal{B})$ are computed in (33). Again, from (46) and (47), we infer that

$$M'(p,q) \leq 3M'(p-1,q-1) + M'(p-1,q) + 4 \quad (56)$$

$$A'(p,q) \leq 3A'(p-1,q-1) + A'(p-1,q) + 3. \quad (57)$$

Since, in a cluttered environment, m is likely to be larger than n in the average case, therefore by using (48) and (51), the initial conditions for (56) and (57) may be simplified to

$$M'(1,q) = 0 \quad (58)$$

$$A'(1,q) \leq 3. \quad (59)$$

With the above initial conditions, it is not difficult to show that both $M'(p,q)$ and $A'(p,q)$ are independent of q and that those are bounded from above by

$$M'(p,q) \leq \tfrac{4}{3}(4^{p-1} - 1) \quad (60)$$

$$A'(p,q) \leq 4^{p} - 1. \quad (61)$$

Substituting (60) into (54) and (61) into (55), the upper bounds for $M(n,m)$ and $A(n,m)$ in the average case can be obtained after routine manipulations.

$$M(n,m) \leq \frac{3n+1}{3}(4^{n-1} - 1) \quad (62)$$

$$A(n,m) \leq (3n+1)(4^{n-1} - 1). \quad (63)$$

Since the design of the fast JPDA algorithm [12] is based on the worst case scenario, there is no computational cost reduction in the average case. As a result, the recursive algorithm developed here is much more efficient than the fast JPDA algorithm, as shown in Table II and IV. If we consider the fact that the entries in Table IV are loose upper bounds, the actual computational cost of the recursive algorithm would be much less.

The upper bounds of $M(n,m)$ and $A(n,m)$ in the DFS algorithm, given in [13, 14], are reproduced below for ready reference

$$M(n,m) < 2*4^{n} - 2 - 3n - 3^{n} \quad (64)$$

$$A(n,m) < 4^{n} + 3n*4^{n-1}. \quad (65)$$

TABLE V
Upper Bounds of $M(n,m)/A(n,m)$ in DFS Algorithm

n	m	$M(n,m)/A(n,m)$	n	m	$M(n,m)/A(n,m)$
3	-	90/208	5	-	1788/4864
4	-	417/1024	6	-	7443/22528

Comparing the upper bounds of $M(n,m)$ in (64) and (62), it is apparent that the recursive algorithm requires less multiplications than the DFS algorithm for moderate value of n. A similar conclusion may be reached by comparing the upper bounds of $A(n,m)$ in (63) and (65). Furthermore, we infer that the recursive algorithm always requires less additions than the DFS algorithm. The computational complexity of the recursive algorithm and the DFS algorithm in the average case may be assessed by comparing sample values of the upper bounds of $M(n,m)$ and $A(n,m)$ given in Table IV for the recursive algorithm and in Table V for the DFS algorithm.

The computational complexity of our algorithm in the average case can also be given in terms of the "big O" notation. From (62) and (63), one can conclude that the computational complexity of our algorithm is $O(n4^{n})$.

VI. CONCLUDING REMARKS

In this paper, a recursive algorithm, which is suitable for implementation in a multiprocessor system, is developed. In this algorithm, the computation of the *a posteriori* probabilities, β_j^ts, is not based on the generation of the data association hypotheses like in the DFS algorithm [13]. Each β_j^t in this algorithm is decomposed into two parts. The computation of one part is trivial and the recursive algorithm, developed here, is used to compute the other part (due to interference from other targets).

In the new algorithm, the computation of the interference part of β_j^t is implemented recursively in a top-to-bottom mode. In the worst case, this recursive algorithm requires more multiplications and additions than the fast JPDA algorithm [12]. However, in the average case, the recursive algorithm is expected to outperform the fast JPDA algorithm. In comparison with the DFS algorithm developed in [13, 14] the recursive algorithm requires less multiplications and additions than the DFS algorithm. The most important feature of the recursive algorithm is that it can be implemented on a multiprocessor system. Some suggestions for the implementation of the new algorithm on a multiprocessor system are given in Appendix A.

The task of computing $F(\mathcal{A}\backslash t,\mathcal{B}\backslash j)$ is related to that of evaluating the permanents, defined in [17], of an $m \times (n-1)$ matrix and its submatrices having P_l^τ for entries, where $\tau = 1,\ldots,n$, $\tau \neq t$, $l = 0,1,\ldots,m$, and $l \neq j$. Justifications are provided in Appendix B, where illustrative examples serve to verify the results obtained by exploiting this relationship.

APPENDIX A: SUGGESTIONS FOR IMPLEMENTATION

In order to explore the inherited parallelism in our efficient algorithm, the computation of the *a posteriori* probabilities $\beta_j^t(\mathcal{A},\mathcal{B})$ in our algorithm may be summarized as below.

1) For $j = 1,2,\ldots,m$, $t = 1,2,\ldots,n$, and $P_j^t \neq 0$,

$$\beta_j^t(\mathcal{A},\mathcal{B}) = P_j^t F(\mathcal{A}\backslash t,\mathcal{B}\backslash j) \tag{66}$$

and for $t = 1,2,\ldots,n$,

$$\beta_0^t = P_0 F(\mathcal{A}\backslash t,\mathcal{B}). \tag{67}$$

2) For any $\mathcal{A}$ and $\mathcal{B}$,

$$F(\mathcal{A},\mathcal{B}) = P_0 F(\mathcal{A}\backslash t,\mathcal{B}) + \sum_{j\in\mathcal{B}\backslash 0} P_j^t F(\mathcal{A}\backslash t,\mathcal{B}\backslash j). \tag{68}$$

3) The constant

$$c = \sum_{j=0}^{m} \beta_j^t(\mathcal{A},\mathcal{B}) \tag{69}$$

may be computed for any value of t as done in (29) for the $t = 1$ case.

4) For $j = 0,1,\ldots,m$, $t = 1,2,\ldots,n$, and $\beta_j^t \neq 0$,

$$\beta_j^t \leftarrow \frac{1}{c}\beta_j^t. \tag{70}$$

To implement our efficient algorithm, memory space has to be allocated for β_j^t, P_j^t, $\mathcal{A}$, and $\mathcal{B}$. Let BETA$(0:m,1:n)$ and $P(0:m,1:n)$ be two 2-D arrays each of size $(m+1)\times n$ for storing β_j^t and P_j^t. Since $\mathcal{A}$ and $\mathcal{B}$ are ordered sets of integers, some special data structures may be required to stored these two sets. As shown in Fig. 2, two types of operations are required on $\mathcal{A}$ and $\mathcal{B}$ to compute of $F(\mathcal{A},\mathcal{B})$. One operator deletes an element from either $\mathcal{A}$ or $\mathcal{B}$ when the computation advances from a node at the higher level of the tree to a node at the lower level. The other operator inserts an element in either $\mathcal{A}$ or $\mathcal{B}$ when the computation backtracks from a node at the lower level of the tree to a node at the higher level. Thus each operation updates either $\mathcal{A}$ or $\mathcal{B}$. In order to implement the two operations efficiently, data structures can be found in [16] for storing $\mathcal{A}$ and $\mathcal{B}$. Let the two functions, Delete$(\mathcal{X},j)$ and Insert$(\mathcal{X},j)$, denote, respectively, the operations of deletion of element j from set $\mathcal{X}$ and insertion of element j into set $\mathcal{X}$ when $j \neq 0$. In the case when $j = 0$, Delete and Insert do nothing. To compute $F(\mathcal{A},\mathcal{B})$, a function can be defined using (A.3). For the sake of simplicity, let $F(\mathcal{A},\mathcal{B})$ be that function which returns the value on the right-hand side of (68).

With the notations introduced above, the computation of *a posteriori* probabilities $\beta_j^t(\mathcal{A},\mathcal{B})$ in our efficient algorithm for implementation on a multiprocessor system is summarized below.

1) For $j = 0,1,2,\ldots,m$, $t = 1,2,\ldots,n$, and $P(j,t) \neq 0$,

$$\text{BETA}(j,t) = P(j,t)F(\text{Delete}(\mathcal{A},t),\text{Delete}(\mathcal{B},j)).$$

2) The normalization constant for any target t is

$$c = \sum_{j=0}^{m} \text{BETA}(j,t).$$

3) For $j = 0,1,2,\ldots,m$, $t = 1,2,\ldots,n$, and BETA$(j,t) \neq 0$,

$$\text{BETA}(j,t) \leftarrow \frac{1}{c}\text{BETA}(j,t).$$

Each BETA(j,t) in steps 1 and 3 above can be computed in parallel in a multiprocessor system such as CM-200 from Connection Machine Inc.. A programmer may view the CM-200 as a set of *virtual processors*, one for each data element. The corresponding code and data are passed to each processor. On a CM-200, steps 1 and 3 may be implemented using a conditional FORALL statement in CM FORTRAN [18]. The effect of a conditional FORALL statement is similar to the embedding of an IF statement in a DO loop using FORTRAN 77. With a conditional FORALL statement, all data elements are operated on simultaneously. In general, our efficient algorithm is ready to be implemented in any parallel computer with single-instruction and multiple-data (SIMD) or multiple-instruction and multiple-data (MIMD) architecture.

APPENDIX B: RELATIONSHIP BETWEEN $F(\mathcal{A},\mathcal{B})$ AND PERMANENT

The definition for a permanent is given below [17].

DEFINITION Let $A = (a_j^t)$ be an $m \times n$ matrix over any commutative ring, $m \leq n$. The *permanent* of A, written Per(A) is defined by

$$\text{Per}(A) = \sum_{\sigma} a_1^{\sigma(1)} a_2^{\sigma(2)} \cdots a_m^{\sigma(m)} \tag{71}$$

where the summation extends over each of the $n!/(n-m)!$ one-to-one maps, denoted by σ, of the set $\{1,2,\ldots,m\}$ to the set $\{1,2,\ldots,n\}$.

In our case, let $\mathbf{P} = (P_j^t)$ denote the matrix having $(m+1)$ rows and n columns. $\mathcal{A}$ and $\mathcal{B}$, defined before, contain, respectively, the column and row indices of $\mathbf{P}$, i.e., $\mathcal{A} = \{1,2,\ldots,n\}$ and $\mathcal{B} = \{0,1,\ldots,m\}$. For notational convenience, in later usage, we denote the permanent of $\mathbf{P}$, Per($\mathbf{P}$), by Per($\mathcal{A},\mathcal{B}$). This causes no confusion or ambiguity because the 2-tuple (j,t) formed from an element, t, of $\mathcal{A}$ and an element, j, of $\mathcal{B}$ uniquely defines P_j^t. Therefore, applying the definition for permanent to a matrix or its transpose, we get

$$\mathrm{Per}(\mathcal{A},\mathcal{B}) = \sum_{\sigma} P_{j_1}^{\sigma(j_1)} P_{j_2}^{\sigma(j_2)} \cdots P_{j_{|\mathcal{B}|}}^{\sigma(j_{|\mathcal{B}|})}, \qquad |\mathcal{B}| \le |\mathcal{A}| \tag{72}$$

$$\mathrm{Per}(\mathcal{A},\mathcal{B}) = \sum_{\sigma} P_{\sigma(t_1)}^{t_1} P_{\sigma(t_2)}^{t_2} \cdots P_{\sigma(t_{|\mathcal{A}|})}^{t_{|\mathcal{A}|}}, \qquad |\mathcal{B}| > |\mathcal{A}|. \tag{73}$$

It is not difficult to show that $F(\mathcal{A},\mathcal{B})$ in (33) is expandable as

$$\begin{aligned} F(\mathcal{A},\mathcal{B}) &= (P_0)^{|\mathcal{A}|} + (P_0)^{|\mathcal{A}|-1}\left(\sum_{t\in\mathcal{A}} \mathrm{Per}(\{t\},\mathcal{B}\backslash 0)\right) \\ &+ (P_0)^{|\mathcal{A}|-2}\left(\sum_{t_1,t_2\in\mathcal{A}} \mathrm{Per}(\{t_1,t_2\},\mathcal{B}\backslash 0)\right) + \cdots \\ &+ (P_0)^{|\mathcal{A}|-|\mathcal{B}\backslash 0|}\left(\sum_{t_1,\ldots,t_{|\mathcal{B}\backslash 0|}\in\mathcal{A}} \mathrm{Per}(\{t_1,\ldots,t_{|\mathcal{B}\backslash 0|}\},\mathcal{B}\backslash 0)\right), \\ & \qquad |\mathcal{B}\backslash 0| \le |\mathcal{A}|, \end{aligned} \tag{74}$$

$$\begin{aligned} F(\mathcal{A},\mathcal{B}) &= (P_0)^{|\mathcal{A}|} + (P_0)^{|\mathcal{A}|-1}\left(\sum_{t\in\mathcal{A}} \mathrm{Per}(\{t\},\mathcal{B}\backslash 0)\right) \\ &+ (P_0)^{|\mathcal{A}|-2}\left(\sum_{t_1,t_2\in\mathcal{A}} \mathrm{Per}(\{t_1,t_2\},\mathcal{B}\backslash 0)\right) + \cdots \\ &+ \mathrm{Per}(\mathcal{A},\mathcal{B}\backslash 0) \qquad |\mathcal{B}\backslash 0| > |\mathcal{A}|. \end{aligned} \tag{75}$$

The validities of (74) and (75) are illustrated by the consideration of some special cases below.

EXAMPLE 1 Let $\mathcal{A} = \{t\}$, so that $|\mathcal{A}| = 1$. In this case, the use of (74) or (75), as the case demands, yields

$$\begin{aligned} F(\mathcal{A},\mathcal{B}) &= P_0 + \mathrm{Per}(\mathcal{A},\mathcal{B}\backslash 0) \\ &= P_0 + \sum_{j\in\mathcal{B}\backslash 0} P_j^t = \sum_{j\in\mathcal{B}} P_j^t. \end{aligned} \tag{76}$$

The above equation agrees with (34).

EXAMPLE 2 Let $\mathcal{B} = \{0\}$, so that $|\mathcal{B}| = 1$ and $|\mathcal{B}\backslash 0| = 0$. In this case, $|\mathcal{A}| \ge 1$ and (74) applies, yielding

$$F(\mathcal{A},\mathcal{B}) = (P_0)^{|\mathcal{A}|}. \tag{77}$$

The above equation agrees with (35).

EXAMPLE 3 Let $\mathcal{B} = \{0,j\}$, where $j \ne 0$, and $|\mathcal{A}| \ge 2$, so that $|\mathcal{B}| = 2$ and $|\mathcal{A}| > |\mathcal{B}\backslash 0|$. Then (74) applies in this case, to yield equation (36).

EXAMPLE 4 Let $\mathcal{A} = \{1,3,4\}$ and $\mathcal{B} = \{0,2,4,5,7\}$. The matrix (P_j^t), where the indices t and j are drawn from the sets $\mathcal{A}$ and $\mathcal{B}$, is (note that $P_0^t \overset{\Delta}{=} P_0$)

$$\begin{pmatrix} P_0 & P_0 & P_0 \\ P_2^1 & P_2^3 & P_2^4 \\ P_4^1 & P_4^3 & P_4^4 \\ P_5^1 & P_5^3 & P_5^4 \\ P_7^1 & P_7^3 & P_7^4 \end{pmatrix}.$$

In this case, $|\mathcal{A}| < |\mathcal{B}\backslash 0|$. Then (75) applies, yielding

$$\begin{aligned} F(\mathcal{A},\mathcal{B}) &= (P_0)^3 + (P_0)^2 \sum_{t\in\mathcal{A}} \sum_{j\in\mathcal{B}\backslash 0} P_j^t \\ &+ P_0 \sum_{\substack{t_1,t_2\in\mathcal{A} \\ t_1\ne t_2}} \sum_{\substack{j_1,j_2\in\mathcal{B}\backslash 0 \\ j_1\ne j_2}} P_{j_1}^{t_1} P_{j_2}^{t_2} \\ &+ \sum_{\substack{j_1,j_2,j_3\in\mathcal{B}\backslash 0 \\ j_1\ne j_2\ne j_3}} P_{j_1}^1 P_{j_2}^3 P_{j_3}^4. \end{aligned} \tag{78}$$

$F(\mathcal{A},\mathcal{B})$ can also be calculated as below, using (33)

$$\begin{aligned} F(\mathcal{A},\mathcal{B}) &= (P_0)^2 \sum_{j\in\mathcal{B}} P_j^4 + P_0 \sum_{j_1\in\mathcal{B}\backslash 0} P_{j_1}^3 \sum_{j_2\in\mathcal{B}\backslash j_1} P_{j_2}^4 \\ &+ P_0 \sum_{j_1\in\mathcal{B}\backslash 0} P_{j_1}^1 \sum_{j_2\in\mathcal{B}\backslash j_1} P_{j_2}^4 \\ &+ \sum_{j_1\in\mathcal{B}\backslash 0} P_{j_1}^1 \sum_{j_2\in\mathcal{B}\backslash\{0,j_1\}} P_{j_2}^3 \sum_{j_3\in\mathcal{B}\backslash\{j_1,j_2\}} P_{j_3}^4. \end{aligned} \tag{79}$$

It is not difficult to show that (78) and (79) are equivalent.

ACKNOWLEDGMENT

The authors thank Jongtae Chun for his careful reading of the manuscript.

REFERENCES

[1] Bar-Shalom, Y., and Fortmann, T. E. (1988)
Tracking and Data Association.
Orlando, FL: Academic Press, 1988.

[2] Fortmann, T. E., Bar-Shalom, Y., and Scheffe, M. (1983)
Sonar tracking of multiple targets using joint probabilistic data association.
IEEE Journal of Oceanic Engineering, **OE-8** (July 1983), 173–184.

[3] Reid, D. B. (1979)
An algorithm for tracking multiple targets.
IEEE Transactions on Automatic Control, **AC-24** (Dec. 1979), 843–854.

[4] Kurien, T., and Washburn, R. B., Jr. (1985)
Multiobject tracking using passive sensors.
In *Proceedings of the American Controls Conference*, Boston, MA, June 1985, 1032–1038.

[5] Kurien, T., and Liggins, M. E., II (1988)
Report-to-track assignment in multisensor multitarget tracking.
In *Proceedings of the 27th Conference on Decision and Control*, Austin, TX, Dec. 1988, 2484–2488.

[6] Bar-Shalom, Y. (1990)
Multitarget-Multisensors Tracking: Advanced Applications.
Norwood, MA: Artech House, 1990.

[7] Fitzgerald, R. J. (1986)
Development of practical PDA logic for multitarget tracking by microprocessor.
In *Proceedings of the American Controls Conference*, Seattle, WA, June 1986, 889–898.

[8] Sengupta, D., and Iltis, R. A. (1989)
Neural solution to the multiple target tracking data association problem.
IEEE Transactions on Aerospace and Electronic Systems, **25** (Jan. 1989), 96–108.

[9] Sengupta, D., and Iltis, R. A. (1990)
Multiple maneuvering target tracking using neural networks.
Technical Report 90-32, Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, Dec. 1990.

[10] Zhou, B., and Bose, N. K. (1993)
A comprehensive analysis of "neural solution to the multitarget tracking data association problem".
IEEE Transactions on Aerospace and Electronic Systems, **29** (Jan. 1993), 260–263.

[11] Nagarajan, V., Chidambara, M. R., and Sharma, R. N. (1987)
Combinatorial problems in multitarget tracking—A comprehensive solution.
IEE Proceedings, Pt. F, *Communications, Radar and Signal Processing*, **134** (Feb. 1987), 113–118.

[12] Fisher, J. L., and Casasent, D. P. (1989)
Fast JPDA multitarget tracking algorithm.
Applied Optics, **28** (Jan. 1989), 371–376.

[13] Zhou, B., and Bose, N. K. (1993)
Multitarget tracking in clutter: Fast algorithms for data association.
IEEE Transactions on Aerospace and Electronic Systems, **29** (Apr. 1993), 352–363.

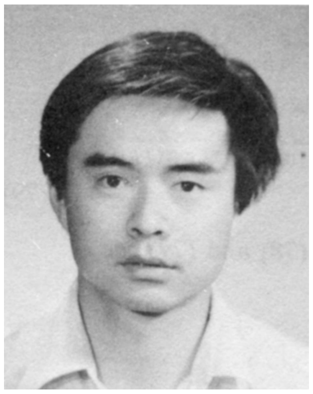
[14] Zhou, B. (1992)
Multitarget tracking in clutter: Algorithms for data association and state estimation.
Ph.D. dissertation, Pennsylvania State University, Department of Electrical and Computer Engineering, University Park, PA 16802, May 1992.

[15] Bar-Shalom, Y., and Tse, E. (1975)
Tracking in a cluttered environment with probabilistic data association.
Automatica, **11** (Sept. 1975), 451–460.

[16] Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1974)
The Design and Analysis of Computer Algorithms.
Reading, MA: Addison-Wesley, 1974.

[17] Minc, H. (1978)
Permanents.
Reading, MA: Addison-Wesley, 1978.

[18] Thinking Machines Corp. (1991)
Getting Started in CM Fortran.
Cambridge, MA: Thinking Machines Corp., Nov. 1991.



Bin Zhou (S'86—M'93) received the B.S. degree in electrical engineering from Fudan University, Shanghai, China, in 1982, the M.S. degree in computer engineering from Shanghai Institute of Technical Physics, China, in 1985, and the Ph.D. degree in electrical engineering from Pennsylvania State University, University Park, in 1992.

Dr. Zhou was a postdoctoral scholar at the Center for Spatial and Temporal Signal Processing and the Center for Multivariate Analysis, Pennsylvania State University. He is now with the Johnson School of Management at Cornell University. His research interests are in the areas of multitarget tracking, neural networks, and signal processing.

N. K. Bose (S'66—M'67—SM'74—F'81) received his B.Tech (with honors), M.S., and Ph.D. degrees, all in electrical engineering, at Indian Institute of Technology, Kharagpur, India, Cornell University, Ithaca, NY, and Syracuse University, Syracuse, NY, respectively.

He is currently the HRB-Systems Professor and Director of the Spatial and Temporal Signal Processing Center at Pennsylvania State University, University Park, PA.

Dr. Bose is the author or editor of several books and numerous papers. He is currently Editor-in-Chief of *Multidimensional Systems and Signal Processing*.