12th International Conference on Information Fusion
Seattle, WA, USA, July 6-9, 2009

Decentralised Data Fusion: A Graphical Model Approach

Alexei Makarenko, Alex Brooks, Tobias Kaupp, Hugh Durrant-Whyte
ARC Centre of Excellence in Autonomous Systems (CAS)
The Univesity of Sydney, Australia
{a.makarenko,a.brooks,t.kaupp,hugh}@cas.edu.au
Frank Dellaert
College of Computing
Georgia Institute of Technology, Atlanta, GA, USA
dellaert@cc.gatech.edu

Abstract – *This paper proposes the use of graphical models to describe decentralised data fusion systems. The task of decentralised data fusion is considered as a specific instance of the general distributed inference problem in which there is a single common state of interest which is (partially) observed by a number of sensor platforms. Our objective is to model and solve this problem using standard graphical model techniques. Two options for modeling the problem are considered. The model based on distributed variable cliques is found superior to a graphical model with cloned variables. The model and the messages arising through inference are compared with the well-known Channel Filter algorithm. Our approach to inference is to apply a distributed version of the Junction Tree algorithm developed by Paskin and Guestrin. The algorithms were validated in a series of simulated tracking problems.*

Keywords: Decentralised data fusion, graphical models.

1 Introduction

This paper proposes the use of graphical models to describe decentralised data fusion systems. Decentralised data fusion (DDF) systems comprise a network of sensor platforms together with probabilistic data fusion algorithms which are decentralised within this network without any single node acting as central fusion site [2, 3]. Graphical models are a series of techniques, founded on graph-theoretic representations, for describing the relationship between states in large probabilistic models and for encoding efficient operations for inference, prediction and fusion [4]. This paper demonstrates that graphical models are a compelling and insightful description of DDF systems which lead to a deeper and clearer understanding of DDF operations.

Decentralised data fusion may be considered as a specific instance of the general distributed inference problem in which there is a single common state of interest which is (partially) observed by a number of sensors, each physically distributed on a number of platforms. The use of a heterogeneous network of distributed sensors has many potential advantages including improved estimation performance, visibility and coverage. DDF systems have also been shown to offer significant advantages in terms of modularity, scalability and robustness [5, 6]. A major issue with DDF algorithms is to understand and formulate the process through which local estimates are formed, communicated and assimilated at remote sites in a manner that ensures efficient and consistent operation. Graphical models provide a compelling approach to this problem by enforcing a uniform network-like model of state and observation relations and the development of communication and assimilation algorithms focused on distributed message passing and local inference.

The objective of this paper is to apply graphical models techniques to both the problem formulation *and* the solution of the DDF problem. This paper begins by describing a decentralised graphical model architecture introduced by Paskin and Guestrin [1] and applying it to the problem of data fusion. The problem statement differs from that in the original work in focusing on a single common underlying state but with the significant extension that the state of interest is driven by a dynamic model.

The paper then provides a graphical model formulation of the well known Channel Filter (CF) algorithm. As part of this, the messages of standard graphical model methods are compared to those employed by CF. This comparison grounds the two approaches in a common model and allows potential exploitation of more general graphical model techniques for decentralised data fusion. In the case of a static phenomenon, the CF message-passing protocol is identical to its graphical model equivalent. In the case of a dynamic phenomenon, the graphical model implementation is an improvement over CF in that it naturally provides for the delays due to multi-hop and burst communication.

The paper is organized as follows. Section 2 defines the problem of decentralised data fusion and states all assumptions. Section 3 describes related work. The following three sections describe three approaches to distribute the model and the inference. The existing CF algorithm is described in detail in Section 4. In Section 5, we show that graphical models based on cloned variables present difficulties with performing inference in the case of Gaussian distribution.

Section 6 presents a better alternative which distribute variable cliques. Simulation results, comparisons, and discussion are presented in Section 7.

2 The Decentralised Data Fusion Problem

This paper considers a system consisting of multiple platforms, each capable of making observations, processing them, and communicating with others. The platforms can be mobile. Platforms may be heterogenous in their sensing, computing, mobility, and communication capabilities.

Physical communication links between platforms form a connectivity graph. In general, the connectivity graph may change over the lifetime of the system. In this paper, however, the attention is restricted to the special case where the communication graph is of general topology but static. Communication links between the platforms may be unreliable, meaning that only the receiver knows when a message is received and neither the sender nor the receiver know when a message is lost.

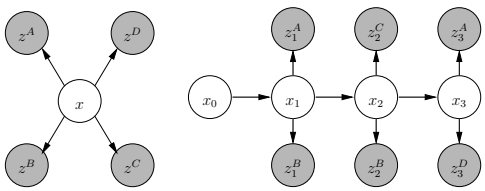
There is a state of interest which all platforms are attempting to estimate which is represented as a set of independent (uncorrelated) features. Attention is limited to individual independent features or tracks. Each feature is described by a fully-correlated state vector. The feature model may have dynamics and may be stochastic.

Distributed tracking of multiple moving targets is an important application of data fusion. The tracking task consists of several subtasks, some of which can be posed as (distributed) inference problems [7]. In this paper we set aside the complex issues of data association, track maintenance, sensor modeling and management, and only consider the task of fusion.

Our objective is to model *and* solve the decentralised data fusion problem using standard graphical model techniques. For an introduction to graphical models see [4]. There are several varieties of graphical models, including directed (Bayesian Networks), undirected (Markov Random Fields), and factor graphs. Their common feature is the use of graph-theoretic concepts to encode a certain factorization of the joint probability distribution over a set of random variables. This factorization is exploited in the design of efficient inference algorithms.

Even though a distributed solution is the objective of this work, it is instructive to consider the centralized case first. Figure 1 shows Bayesian networks representing data fusion in the cases of static and dynamic phenomena.

The static network in Figure 1a includes the state of interest x and observations made on different platforms, *e.g.* z^A is observed by platform A. The observation model for each sensor is represented by a conditional probability in the form of $p(z^i \mid x)$. The dynamic model in Figure 1b includes a set of variables x_k representing the state of interest at time t_k. This *Dynamic Bayesian Network* is a simple Markov chain. The state prediction model between any two consecutive time slices is represented by a conditional probability



(a) static model (b) dynamic model

Figure 1: Bayesian networks for data fusion in the cases of static and dynamic models. Four sensing platforms contribute observations. Circles represent random variables. Shaded variables are observed, unshaded ones are hidden and need to be estimated.

in the form of $p(x_{k+1} \mid x_k)$.

The objective of this work is to compute the *exact* distributed solution of the problems in Figure 1. A distributed solution is considered to be exact if it produces the result equivalent to that of a centralized solution when it is given access to same set of observations. There are several caveats related to the exactness of all distributed fusion algorithms. All distributed solutions display a certain latency when compared to the centralized solution due to the time taken to propagate messages between platforms. In applications such as multi-target tracking where data association is necessary, the results of a distributed algorithm may differ from the centralized solution due to variations in data association based on incomplete information. Similarly, in cases where the system is non-linear, a distributed linearized solution may differ from the centralized one due to a different linearization point calculated based on incomplete information. We do not discuss this further because all approaches considered in this paper are affected equally by these problems.

Furthermore, the focus is on *scalable* solutions. A distributed solution is considered to be scalable if computational and communication costs scale well with the number of participating platforms. As a benchmark we use the naive solution of broadcasting or routing all system-wide observations to all platforms and implementing centralized fusion on each individual platform. One way to improve on this scheme is to combine information contained in the observations from several platforms. In order to allow this operation, we impose an additional constraint on the problem statement: all observations happen at discrete intervals, so that observations z_1^A and z_1^B in Figure 1 happen at the same time.[1]

3 Related Work

Related work can be classified along the following dimensions: (i) model type of the phenomenon of interest (static *vs.* dynamic) and (ii) solution properties (exact, conservative, or approximate).

[1] In practice, this may require time synchronization across different platforms. Approximations when the process noise is low are discussed in [8].

3.1 Distributed Estimation and Tracking

The estimation and tracking community has long studied decentralised data fusion techniques. The fundamental problem is preventing double-counting of information which can happen when two local (correlated) posteriors are fused.

The Information Graph [8] is an exact algorithm which prevents double-counting of information contained in sensor observations. This is achieved by storing and transmitting pedigree information which includes all *sensor reports* which have contributed to a certain posterior. One of the advantages of the algorithm is an intuitive graphical representation of the information flow in a distributed system (which is unrelated to the graphical model framework.) The algorithm has not been applied in the case of a stochastic process model. The algorithm can handle arbitrary topologies but is not considered to be scalable due to the requirement to carry long pedigree information for decorrelation [9].

In tracking, a distinction is commonly made between measurement fusion and track-to-track fusion, the latter referring to the fusion of local solutions. Each local solution may include information from the prior, local and remote observations, and the model. Track-to-track fusion is more challenging due to the fact that while measurement errors are commonly assumed to be independent conditioned on the state of interest, the posteriors of local trackers are generally correlated with one another [10]. Exact solutions are possible if full-rate communication is used, *i.e.* information between *all* platforms is exchanged after every observation.

The Channel Filter algorithm is a solution to the DDF problem. CF is exact for static problems or for dynamic problems when full-rate communications are used. It prevents double-counting by adhering to a tree communication topology and by keeping a record of the information transmitted over a communication channel in the so-called *channel filter*. It was originally formulated for Gaussian representations [2] and later extended to the general Bayesian case [3]. A notable application is a system of multiple UAV platforms localizing stationary features and tracking moving objects on the ground [5]. This paper demonstrates a close relationship between CF and graphical model inference algorithms.

When dealing with Gaussian distributions, an alternative approach to prevent double-counting is to always use conservative data fusion using Covariance Intersect [11]. The advantage of this approach is that it applies to arbitrary communication topologies and dynamic feature models. The results however are always approximate and may be overly conservative.

A more recent approach to distributed fusion which can also be applied to general topologies is based on consensus filters. One particular application combines the notion of consensus filters with the information filter [12]. This algorithm requires convergence and applies to quasi-static phenomena only.

Most approaches to distributed tracking rely on filtering. For out-of-sequence observations, finite memory must be retained [13]. The graphical model approach which we will consider extends naturally to the smoothing case.

Graphical models have been suggested in the context of decentralised data fusion. For example, in [14] a combination of Information Graph and Bayesian networks is used. Information Graph is used to identify common information due to past communication so that double-counting can be avoided. Bayesian network theory is used to identify the minimal state satisfying the conditional independence condition. Graphical models are not used to represent the relationships between the information maintained on different platforms, and graphical model algorithms are not used to perform the actual inference.

3.2 Distributed Inference with Graphical Models

The bulk of research in graphical models has focussed on centralized systems. The majority of distributed applications use approximate methods [7].

A notable exception is the theoretical and experimental work of Paskin applied to a large sensor network of micro sensors [1]. The architecture, which we refer to as the Distributed Junction Tree (D-JT), is designed specifically for distributed inference and consists of three interacting algorithms arranged into layers and running concurrently. The bottom layer builds a spanning tree. The spanning tree overlays the physical connectivity network which in physical sensor networks typically has a mesh topology. The middle layer operates on a set of variable cliques defined as part of algorithm initialization. A spanning tree produced by the bottom layer connects the cliques into a clique tree. The middle layer converts it into a junction tree by enforcing the running intersection property. Finally, the top layer performs inference on the junction tree.

This paper applies the D-JT algorithm to the problem of decentralised data fusion. The original work was applied to estimating a distributed field property (temperature). On the one hand, we narrow down the problem definition to the case where there are multiple platforms but only one variable of interest. On the other hand, we extend the original problem statement to include dynamics.

Another application is in distributed sensor localization based on the observations of a single moving target [15]. The solution focusses on ameliorating the dense correlations which arise from marginalizing out past states of the moving target. The current application is special in that the state of interest is assumed to be densely correlated from the outset, and hence cannot become more tightly coupled.

4 Decentralised Data Fusion

The detailed discussion of inference algorithms begins by summarizing the approach and the results obtained by Channel Filter (CF). Each platform performs inference on the same state x and the local beliefs are kept *synchronized* by exchanging messages defined by the CF protocol [3]. It

is assumed that the communication topology between platforms is constrained to a tree.

4.1 Channel Filter, Static Model

The point of departure is the centralized static model in Figure 1a. CF distributes this model by running local inference on each platform, using local observations. Messages are passed between the platforms to make the information contained in local observations available to other platforms.

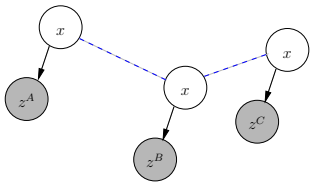


Figure 2: Distributed static model with local beliefs on each platform synchronized with CF links (dashed lines).

Figure 2 shows three local beliefs, one for each platform. The models for local observations are represented with standard Bayesian network relationships. The dashed lines represent CF links and are not part of the graphical model framework. The topology of the CF links forms a spanning tree.[2]

For each pair of platforms i and j linked by the spanning tree, CF defines two *channel filters* ϕ_{ij} and ϕ_{ji}, one stored on each communicating platform. Each channel filter stores the *common information* which has been communicated through the communication channel between the corresponding pair of platforms. To simplify the presentation, we assume an uninformative prior. In this case the channel filters are initialized to unity, $\phi_{ij} = 1$. Before the start of communication, all three platforms make local observations ($z^i = \bar{z}^i$) resulting in local posteriors $\psi_i = p(x \mid \bar{z}^i)$.

Platform A sends a message to platform B, containing the current local posterior $\psi_A = p(x \mid \bar{z}^A)$. The two channel filters ϕ_{AB} and ϕ_{BA} are set to the value of the message. On arrival of the message at platform B, the local belief is updated as follows

$$\psi_B^* = \frac{\phi_{BA}^*}{\phi_{BA}}\psi_B = \frac{\psi_A}{1}\psi_B = p(x \mid \bar{z}^A, \bar{z}^B). \quad (1)$$

Asynchronously, platform C sends its own message to platform B and updates both channel filters. On arrival of the message at platform B, the local belief is updated again.

$$\psi_B^{**} = \frac{\phi_{BC}^*}{\phi_{BC}}\psi_B^* = \frac{\psi_C}{1}\psi_B^* = p(x \mid \bar{z}^A, \bar{z}^B, \bar{z}^C). \quad (2)$$

At this point, the belief at platform B $p_B^{CF}(x) = \psi_B^{**}$ incorporates evidence from all three platforms. Now platform B sends a message to platform A, containing ψ_B^{**}. After updating the channel filters, the local belief at platform A is updated as follows

$$\psi_A^* = \frac{\phi_{AB}^{**}}{\phi_{AB}^*}\psi_A = \frac{\psi_B^{**}}{\psi_A}\psi_A = p(x \mid \bar{z}^A, \bar{z}^B, \bar{z}^C). \quad (3)$$

Now the belief at platform A $p_A^{CF}(x) = \psi_A^*$ incorporates evidence from all three platforms and is the same as that of platform B. The update of platform C happens similarly, so that after all messages are passed the belief of all three platforms is the same and is equivalent to the centralized solution.

We can summarize the CF message passing algorithm for the static case as follows.

$$\phi_{ij}^* = \psi_i, \quad \phi_{ji}^* = \psi_i \quad (4)$$

$$\psi_j^* = \frac{\phi_{ji}^*}{\phi_{ji}}\psi_j \quad (5)$$

At any point in the process, the probability distribution of the state of interest at any platform is equal to the local posterior

$$p_i(x) = \psi_i. \quad (6)$$

After all the messages have been passed, the probabilities $p_i(x)$ on all platforms are the same and are equal to the centralized solution.

4.2 Channel Filter, Dynamic Model

The same distribution procedure can be applied to the dynamic problem in Figure 1b. The model, shown in Figure 3, represents "Channel Smoother" even though it has not been described as such in the literature. The common, filtering, version can be obtained by iteratively marginalizing all past states.

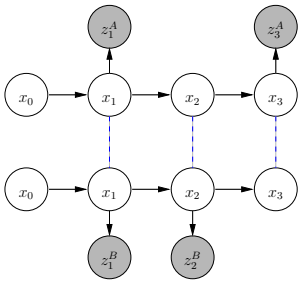


Figure 3: Distributed dynamic model with local beliefs on each platform synchronized with CF links (dashed lines).

In summary, the CF solution for a static model is exact and scalable, regardless of the actual communication schedule. When the process model is dynamic, the (filtered) CF solution produces exact results only when full-rate communication is employed. The delayed-state version of CF with finite memory of past states is possible [13] but we do not consider it here because the equivalent solution arises more naturally in the context of graphical models to which we turn our attention next.

[2]Because this is not a graphical model, we are not constrained by the requirement that all variables must be unique.

5 Graphical Model with Cloned Variables

In this section we attempt to distribute the inference by manipulating the graphical model on the level of random variables. Since every variable in a graphical model is unique we have to create *clones* – new variables, one for each platform, which are distinct from the original state yet are understood to hold the same value. The resultant graphical model can then be solved with any of a number of inference algorithms. We will show that this modeling approach is not attractive due to the need to represent deterministic relationships between the clones.

5.1 Cloned Variables, Static Model

We start again with the centralized static model in Figure 1a. Since we have multiple platforms we create multiple clone variables, one for each platform, and define deterministic equality relationships between the clones as shown in Figure 4. A spanning tree of links is sufficient to connect all clones together, because redundant links do not add any new information.

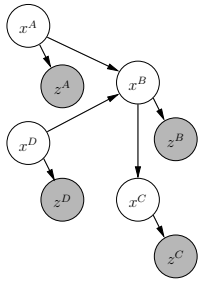


Figure 4: Distributed static model with cloned variables represented as a Bayesian network with poly-tree topology. All links between cloned variables represent deterministic equality relationships.

This model represents the joint probability distribution over all random variables.[3] Before the evidence on the observed variables is entered, the joint distribution is

$$p(\mathbf{x}, \mathbf{z}) = \prod_{i \in P} p(x^i \mid \pi(x^i)) p(z^i \mid x^i), \tag{7}$$

where $\mathbf{x}$ is the set of all cloned state variables, $\mathbf{z}$ is the set of all observation variables, P is the set of platforms, and $\pi(x^i)$ are the parents of variable x^i.

Let us examine the mechanics of specifying a deterministic equality link. To indicate the deterministic equality $x^A = x^B$ with discrete representations, the conditional probability table for node x^B would be simply the identity matrix, $p(x^B \mid x^A) = I$. In the case of continuous variables, the linear conditional Gaussian [16] is entered as a *conditional form*

$$p(x^B \mid x^A) = CG(\mu_0 + Ix^A, \Sigma_0), \tag{8}$$

where I is the identity matrix and μ_0 and Σ_0 are the appropriately-sized zero vector and matrix respectively. The standard inference procedure is to convert the conditional form to a *canonical form*[4] which involves the problematic step of inversion of the Σ_0 matrix [16, 17].

A pragmatic solution is to make the equality relationship nearly deterministic by adding a small amount of uncertainty. This leads to an approximate result and presents numerical difficulties due to poor conditioning. We do not consider this option.

An alternative procedure described in [18] avoids conversion to canonical form and performs inference with conditional form, at the expense of added complexity.

5.2 Cloned Variables, Dynamic Model

In the case of a dynamic model we follow the same approach, by cloning the state variable at each time slice and distributing a clone to each platform in the system. This results in the Bayesian network shown in Figure 5a. The process model is applied by one of the platforms and the information contained in the model is distributed through the system by the messages passed during the inference process.

The single-model approach is simple to implement and the concerns regarding the robust operation in the face of platform and communication failure may be partially alleviated by a decentralised *leader election* algorithm. Numerous such algorithms exist and, in fact, one of them is already used elsewhere in our system (as part of the decentralised spanning tree algorithm). Nevertheless, leader re-election adds latency and careful algorithm design is needed during this time to avoid model double-counting.

The system in Figure 5b attempts to increase system robustness by placing model information on every platform. It can easily be shown that the result produced by the network is incorrect due to double-counting of the model information.

The system in Figure 5c also places a model on each platform but the monolithic Bayesian network is split up into several fragments with restricted information flow between them. We now have k *distributed networks*, one for each of the time slices t_k, each spanning all clones of x_k (e.g. x_1^A and x_1^B). Each of the distributed networks operates on a single time slice and is identical to the static network in Figure 4. The k distributed networks act as *likelihood accumulators*, combining all observation likelihoods about the state x_k.

In addition, there are n *local networks*, one for each of the platforms, connecting the consecutive instances of the state. The n local networks act as *model predictors*, propagating

[3] The directed links between cloned variables can easily be replaced with undirected links.

[4] Canonical (or information) form of a Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ is defined by the information matrix Σ^{-1}, the information vector $\Sigma^{-1}\mu$, and a normalizing constant.

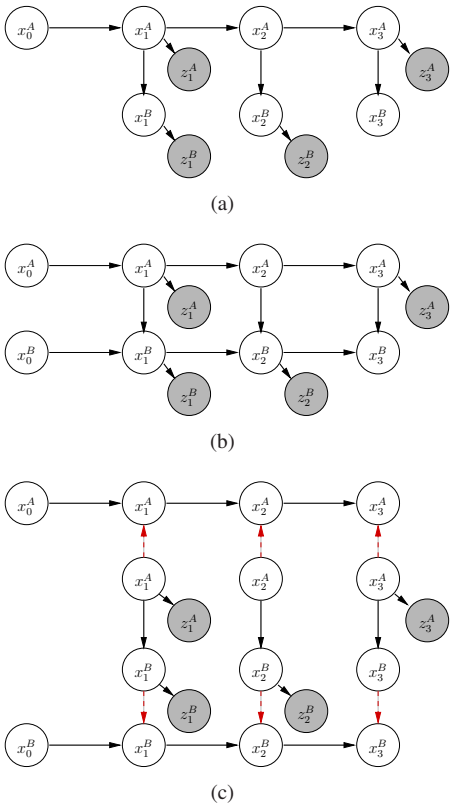


Figure 5: Distributed dynamic model with cloned variables using a Bayesian network: a) a single platform in the system applies the model, b) each platform applies the model – the result is incorrect, c) the result is correct if the local model is decoupled from the network-wide likelihood aggregation.

information from one time slice to the next. Each local network is confined to a single platform allowing a centralized inference solution.

The information flow between the networks is one-directional, *i.e.* the information flows from each of the distributed networks into each one of the local networks. The information in the opposite direction is restricted to prevent double-counting of the model information.[5]

Figure 5 represents three options for model distribution, two of which are suitable for exact inference. The same dilemma for model distribution applies in the static case, a fact which we have overlooked until now. The model in the static case consists of the prior on the state. When the prior is informative and the information rate from observations is low, distributing it to every platform as is done in Figure 5b may lead to significant overconfidence.

In a dynamical system, marginalization is required to maintain constant memory requirements over time. Due to the simple model structure, marginalization of past states does not create any fill-in links between the variables.

[5] One possible implementation of this information valve is by setting up an equivalent measurement which contributes the accumulated likelihood to the local model.

In summary, inference on static or dynamic models with cloned variables requires to represent deterministic equality links which is problematic for Gaussian representation. Fortunately, there is an alternative representation which avoids this problem altogether.

6 Distributed Junction Model

An alternative to distributing the graphical model itself is to distribute the potentials[6] over the cliques of random variables of the original centralized model.

6.1 D-JT, Static Model

The centralized static model is reproduced in Figure 6a where, for clarity, we do not initially display the observation variables z^i as evidence variables.

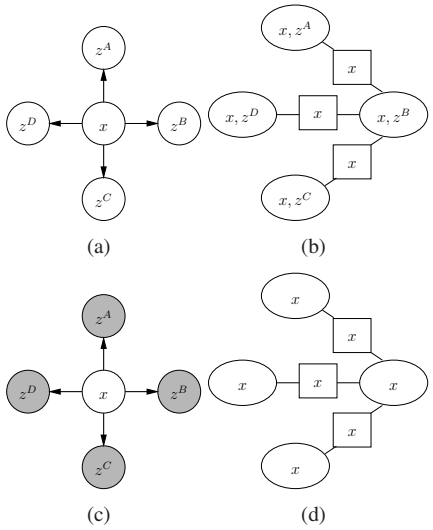


Figure 6: Distributed static model with junctions: a) graphical model with no evidence and b) a possible junction tree, with junctions shown as ovals; c) the same graphical model with evidence and d) the updated junction tree. Separation sets are also shown as squares.

By examining the moralized graph of the model we note that it is already triangulated regardless of the number of platforms in the system. Our example with four platforms contains four maximal cliques, one per platform observation, shown as ovals in Figure 6b. Once the cliques are identified, the only remaining step before conducting inference is to connect the cliques into a junction tree. A junction tree is a special clique tree which has the running intersection property. We observe that in this special case any spanning clique tree is a junction tree. The running intersection property is assured by the fact that the variable x is shared by all cliques. One of several possible spanning clique tree configurations is shown in Figure 6b.

[6] *Potentials* are functions which are real-valued and non-negative, but otherwise arbitrary. By normalization to 1, otentials can be converted to probability distributions.

In general, a junction tree data structure defined over a set of variables Y represents the joint probability distribution in the following factorized form

$$p(Y) \propto \frac{\prod_C \psi_C(Y_C)}{\prod_S \phi_S(Y_S)}, \tag{9}$$

where ψ_C are the potentials over the set of cliques C and ϕ_S are the potentials over the set of separators S. For the model of static data fusion with any number of platforms, this expression becomes

$$p(x, \mathbf{z}) \propto \frac{\prod_{i \in P} \psi_i(x, z^i)}{\prod_{e \in E} \phi_e(x)}, \tag{10}$$

where $\mathbf{z}$ is the set of all observation variables, P is the set of platforms, and E is the set of edges in the junction tree.

In Figure 6c we return to our problem statement where the observation variables are in fact observed. Entering evidence essentially eliminates these variables from the inference problem and we are left with a very special junction tree where every clique set and every separator set contains one and the same variable x.

$$p(x) \propto \frac{\prod_{i \in P} \psi_i(x)}{\prod_{e \in E} \phi_e(x)}. \tag{11}$$

This expression can be compared to the joint on all cloned variables in (7). As a result we do not have to represent the deterministic links.

We are now ready to perform inference on our junction model. We describe two standard junction tree algorithms, namely Hugin and Shafer-Shenoy, and apply them to our model. One of them, Hugin, results in messages which are identical to those passed by CF. Consistent with the CF presentation, we initially assume an uninformative prior.

The Hugin Algorithm

The data structures defined by the Hugin algorithm [19] are shown in Figure 7 for two arbitrary cliques in a junction tree. Each clique C_i has a potential ψ_{C_i} associated with it. Similarly, each separator S_{ij} contains a potential $\phi_{S_{ij}}$.

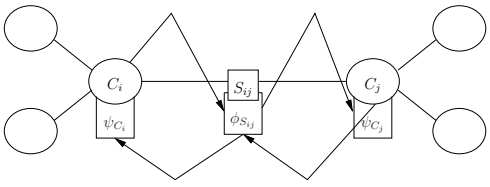


Figure 7: Data structures for the Hugin algorithm. Two cliques C_i and C_j have a separator set $S_{ij} = C_i \cap C_j$.

The inference proceeds by passing messages between the neighboring cliques and separators and by updating the potentials according to the following update equations

$$\phi^*_{S_{ij}} = \sum_{C_i \setminus S_{ij}} \psi_{C_i} \tag{12}$$

$$\psi^*_{C_j} = \frac{\phi^*_{S_{ij}}}{\phi_{S_{ij}}} \psi_{C_j}, \tag{13}$$

where $C_i \setminus S_{ij}$ denotes the set of variables in C_i which do not appear in S_{ij}. The summation symbol in Equation 12 refers to marginalisation, and is performed by integration for continuous variables.

After all messages have been exchanged, the marginal probability for any clique is simply the normalized potential for that clique.

$$p(C_i) \propto \psi_{C_i} \tag{14}$$

We apply this general algorithm to the model in Figure 6d by setting $C_i = C_j = S_{ij} = x$. The simplified update equations become

$$\phi^*_{S_{ij}} = \psi_{C_i} \tag{15}$$

$$\psi^*_{C_j} = \frac{\phi^*_{S_{ij}}}{\phi_{S_{ij}}} \psi_{C_j}. \tag{16}$$

By comparing these with the CF update equations (4-5), we recognize that, in the case of static data fusion, Hugin and CF send the same messages and use the same update equations.[7]

The Shafer-Shenoy Algorithm

The data structures defined by the Shafer-Shenoy algorithm [20] are shown in Figure 8 for two arbitrary cliques in a junction tree. This algorithm differs from Hugin in two respects. Firstly, the separator potentials are not defined explicitly. Secondly, the messages μ_{ij} received from neighbors are not multiplied into the local potential thereby maintaining a factored representation of the clique marginal.

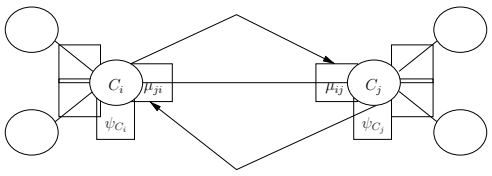


Figure 8: Data structures for the Shafer-Shenoy algorithm.

The inference proceeds by passing the following messages between the cliques.

$$\mu_{ij} = \sum_{C_i \setminus S_{ij}} \psi_{C_i} \prod_{k \neq i} \mu_{ki} \tag{17}$$

[7]The fact that CF traditionally maintains two channel filters, one on each communicating platform, is simply an implementation detail.

After all messages have been exchanged, the marginal probability for any clique is calculated as follows.

$$p(C_i) \propto \psi_{C_i} \prod_k \mu_{ki} \tag{18}$$

It can easily be shown that the clique marginal distributions obtained by both Hugin and Shafer-Shenoy are the same in the case of centralized inference [20]. When specialized to our problem, the Shafer-Shenoy messages becomes simply

$$\mu_{ij} = \psi_{C_i} \prod_{k \neq i} \mu_{ki}. \tag{19}$$

6.2 D-JT, Dynamic Model

Similar to the static case, the moralized graph of the dynamic model is triangulated for any number of platforms and any number of time slices. The available options for distributing this model are similar to those discussed in Section 5 but there are differences due to the specifics of operating on the level of junctions.

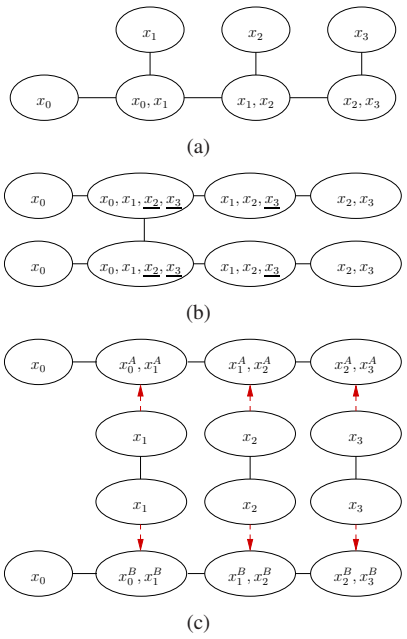


Figure 9: Distributed dynamic model with junctions: a) a single platform in the system applies the model, b) each platform defines the model, c) the locally-applied model is decoupled from the network-wide likelihood accumulation. Variables which were "pushed" into the cliques to enforce the running intersection property are shown underlined.

One option is to entrust the application of the model to a single platform. The model information will then propagate to other platforms via messages. This situation is depicted in Figure 9a. The disadvantage of this scheme is that it produces delays in applying information contained in the prediction model. This is due to the fact that all observation likelihoods must travel first to the root where the model is applied and the back to the leaves.

Figure 9b shows an option whereby each platform defines the same cliques. A strong disadvantage of this option is that enforcing the running intersection property leads to large cliques. The example in Figure 9b shows that the two cliques which carry the cross-platform link were forced to include all the variables in the system. The underlined variables were "pushed" into the clique by the algorithm which enforces the running intersection property [1]. The width of the junction tree has a strong effect on the computational complexity of inference. We observe that the width of the junction tree when using the cross-platform connection is always equal to the size of the temporal history, regardless of which clique carries the cross-platform link. This property makes this option very unattractive.

Finally, we may choose to follow the approach of using an external dynamic model as illustrated in Figure 9c. Each distributed network is implemented using either the Hugin or Shafer-Shenoy algorithm. The accumulated likelihoods for a given time slice are added to the local network in the same way as if the evidence was entered locally.

In practice we may want to marginalize out old variables to avoid the requirement of infinite memory. The general case of variable marginalization as part of junction tree filtering has been considered in the context of Simultaneous Localization and Mapping [21]. The procedure is described in a centralized setting and is used in this paper when applied to the marginalization of the past states of the local model in Figure 9c, which is confined to a single platform.

In summary, both the Shafer-Shenoy and the Hugin algorithms produce exact results for static and dynamic models. For the case of a static model, it was demonstrated that the messages passed by the Hugin and the CF algorithms are identical. In the dynamic case, the approach in Figure 9c is preferred because it avoids the extra latencis of Figure 9a and the wider junction tree of Figure 9b.

7 Simulation

The algorithms were validated in a series of simulated tracking problems. In each case, a number of platforms and a single target are distributed randomly throughout an environment, as shown in Figure 10. Platforms can make noisy observations of the target when within sensor range, and can communicate with each other when within communications range. In order to ensure stable connectivity graph, the platforms are static. We examine the following cases:

- **Static *vs* Dynamic Phenomenon**: In the dynamic case the target moves.
- **Shafer-Shenoy *vs* Hugin**: On the inference layer we use D-JT, using either Hugin or Shafer-Shenoy as described in Section 6.

In all cases, the network consists of 200 nodes, and the simulation is run for 120 iterations. On the 100^{th} iteration (t_s), we begin a "settling time", during which observations

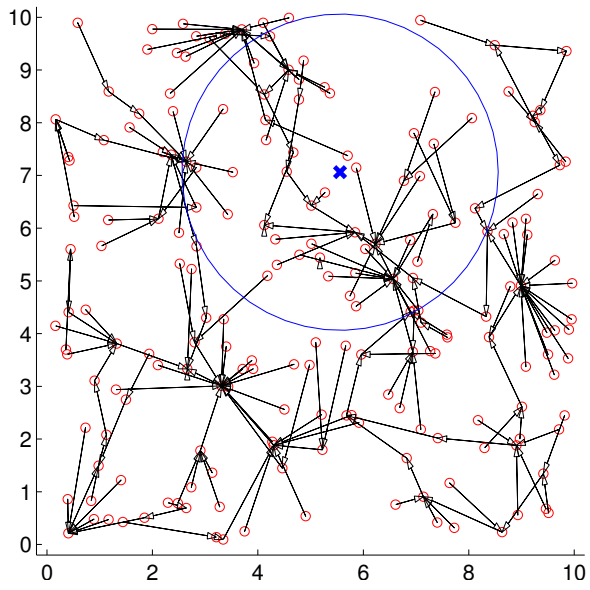


Figure 10: Simulation of 200 nodes constructing a decentralised spanning tree. The target is indicated by the cross, and can be observed by any platform within the blue circle centred on it.

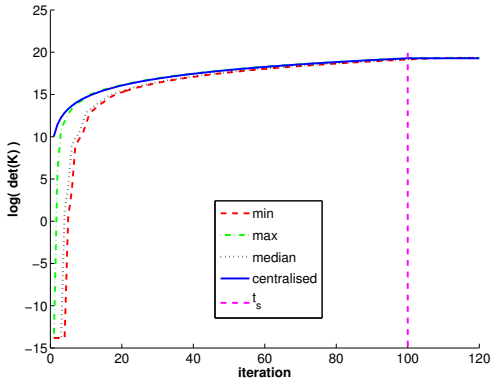


Figure 11: Information (the volume of the information ellipse) collected by sensor network about a static phenomenon. The results produced by Hugin and Shafer-Shenoy are identical.

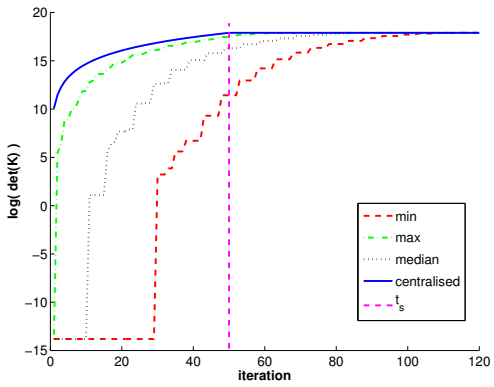


Figure 12: Hugin/Shafer-Shenoy static phenomenon and burst communications. Settling time begins earlier as the algorithm requires more time to converge.

are not made. This settling time lasts for 20 iterations. In all cases, information content is measured using the log of the determinant of the information matrix, proportional to the log of the inverse of the volume of the covariance ellipse. The information contained in each platform's local belief is compared against the information content of the centralised solution.

7.1 Static Phenomenon

Both the Hugin and Shafer-Shenoy algorithms produce identical estimates when the phenomenon of interest are static (Figure 11). The figure also plots the minimum, median, and maximum of the information content available to all the platforms in the system, showing that there is an "information gradient" across the network as information is constantly flowing outwards from the platforms closest to the target.

We also consider the case of burst communications. This is implemented by having each platform increment a counter (initialised randomly), and only transmit on every n^{th} iteration. Figure 12 shows results for $n = 5$, demonstrating that the algorithms are able to cope with delayed observations.

The differences are that (1) information flows through the network more slowly (responsible for a greater lag behind the centralised solution), and (2) the information on the platforms grows in steps as messages arrive in bursts.

7.2 Dynamic Phenomenon

Figure 13 shows that the Hugin and Shafer-Shenoy algorithms also give identical exact results when the model is dynamic. In this case, the information content of the centralised solution changes as the target moves, and hence the number of platforms within sensor range of the target changes. With the start of the settling time, observations cease, the centralised precision decreases due to the process noise, and the distributed estimates catch up to the centralised estimate.

8 Conclusions

The main focus of this paper is in casting the DDF problem in terms of general distributed inference problem and applying standard inference algorithms.

Two options for modeling the DDF problem were investigated. The one utilizing cloned variables was found unattractive due to the difficulties with representing deterministic links in the case of Gaussian distributions. The distributed junction model was shown to be more suitable for this problem.

Two standard junction tree algorithms were considered.

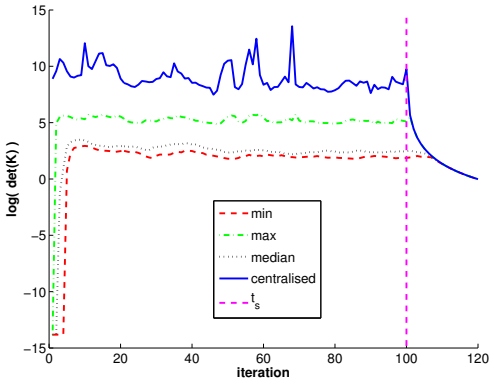


Figure 13: Hugin/Shafer-Shenoy with a dynamic model.

Both the Shafer-Shenoy and the Hugin algorithms produce exact results for static and dynamic models. For the case of a static model, it was demonstrated that the messages passed by the Hugin and the CF algorithms are identical. This identifies the link between the existing work and the graphical model approach.

The results of this paper are limited to static communication topology. We are currently comparing the solution properties of the two version of junction tree algorithm in the case of topology changes.

The algorithms have been implemented based on the ideas of D-JT architecture. Communication links in the current implementation are simulated but the true distributed system with wireless communications is under development.

In some respects our implementation based on the graphical model techniques has gone beyond the previous DDF solutions based on the Channel Filter. Our implementation assembles the network using the decentralised spanning tree algorithm of D-JT. We have also implemented the equivalent of delayed-state version of CF which allows delayed and burst communication.

Graphical model descriptions of DDF systems provide guidance on improved fusion algorithms for large networks subject to rapidly varying topology changes and to issues of conservative fusion in the face of data delay. Another interesting direction is to enlarge the types of models which are possible to hybrid distributions, sparse feature descriptions, and non-linear relationships. Finally, we would like to apply the ideas and implementation to non-tracking applications like SLAM.

Acknowledgements

This work is partly supported by the ARC Centre of Excellence programme, funded by the Australian Research Council (ARC) and the New South Wales State Government, and by the U.S.Army Research Laboratory under the Micro Autonomous Systems and Technology program.

References

[1] M. Paskin and C. Guestrin. A robust architecture for distributed inference in sensor networks. Report IRB-TR-03-039, Intel, 2004.

[2] S. Grime and H. Durrant-Whyte. Data fusion in decentralized sensor networks. *Control Engineering Practice*, 2(5):849–863, 1994.

[3] A. Makarenko and H. Durrant-Whyte. Decentralized Bayesian algorithms for active sensor networks. *International Journal of Information Fusion*, 7(4):418–433, 2006.

[4] R.G. Cowell. *Probabilistic networks and expert systems*. Springer, NY, 1999.

[5] S. Sukkarieh, E. Nettleton, J.-H. Kim, M. Ridley, A. Goktogan, and H. Durrant-Whyte. The ANSER project: Data fusion across multiple uninhabited air vehicles. *Int. J. of Robotics Research*, 22(7/8):505–540, 2003.

[6] B. Upcroft, M.F. Ridley, L. Ong, B. Douillard, T. Kaupp, S. Kumar, T.A. Bailey, F.T. Ramos, A. Makarenko, A. Brooks, S. Sukkarieh, and H.F. Durrant-Whyte. Multi-level state estimation in an outdoor decentralised sensor network. In *ISER'06*, 2006.

[7] M. Uney and M. Cetin. Graphical model-based approaches to target tracking in sensor networks: An overview of some recent work and challenges. *ISPA*, pages 492–497, 2007.

[8] C.Y. Chong, S. Mori, and K. Chang. Distributed multitarget multisensor tracking. In Y. Bar-Shalom, editor, *Multitarget-Multisensor Tracking: Adv. Applications*, pages 247–295. Artech House, 1990.

[9] K.C. Chang, Chee-Yee Chong, and S. Mori. On scalable distributed sensor fusion. *Int. Conf. on Information Fusion*, pages 1–8, 2008.

[10] Shozo Mori, W.H. Barker, Chee-Yee Chong, and Kuo-Chu Chang. Track association and track fusion with nondeterministic target dynamics. *IEEE Trans. on Aerospace and Electronic Systems*, 38(2):659–668, April 2002.

[11] D. Nicholson, C. Lloyd, S. Julier, and J. Uhlmann. Scalable distributed data fusion. In *Conf. Information Fusion*, volume 1, pages 630–635, Sunnyvale, CA, USA, 2002.

[12] Kevin M. Lynch, Ira B. Schwartz, Peng Yang, and Randy A. Freeman. Decentralized environmental modeling by mobile sensor networks. *IEEE Trans. on Robotics*, 24(3):710–724, 2008.

[13] EW Nettleton and H.F. Durrent-Whyte. Delayed and asequent data in decentralised sensing networks. In *SPIE Photonics*, pages 1–9, 2001.

[14] Chee-Yee Chong and Shozo Mori. Graphical models for nonlinear distributed estimation. In *Conf. Information Fusion*, volume I, pages 614–621, 2004.

[15] S. Funiak, C. Guestrin, M. Paskin, and R. Sukthankar. Distributed localization of networked cameras. In *IPSN'06*, pages 34–42, 2006.

[16] S. Lauritzen. Propagation of probabilities, means and variances in mixed graphical association models. *J. of the Amer. Statistical Assoc.*, 87:1098–1108, 1992.

[17] K. Murphy. Inference and learning in hybrid bayesian networks. Technical report, U.C. Berkeley, 1998.

[18] S. Lauritzen and F. Jensen. Stable local computation with conditional gaussian distributions. *Statistics and Computing*, 11:191–203, 1999.

[19] F. Jensen, S. Lauritzen, and K. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quaterly*, 4:269–282, 1990.

[20] G. Shafer and P. P. Shenoy. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2:327–52, 1990.

[21] M. Paskin. Thin junction tree filters for simultaneous localization and mapping. In *IJCAI'03*, pages 1157–1164, 2003.