Decentralised Data Fusion in 2-Tree Sensor Networks

Paul R. Thompson
The Australian Centre for Field Robotics
The University of Sydney
Australia.
p.thompson@acfr.usyd.edu.au

Hugh Durrant-Whyte
The Australian Centre for Field Robotics
The University of Sydney
Australia.
hugh@acfr.usyd.edu.au

Abstract – *This paper describes an algorithm for decentralised estimation and data fusion in '2-tree' networks. In earlier approaches, exact decentralised estimation which avoids double-counting information has generally only been possible in singly-connected or 1-tree networks, or by using conservative fusion. The problem with tree networks is that they are very fragile; a single node failure causes separation of the network. In contrast, a 2-tree network is a denser network in which two nodes must fail for the network to become separated, but which are still much sparser, and therefore more scalable, than fully-connected networks. This paper describes an algorithm for decentralised estimation in 2-tree networks that correctly accounts for common information in communication, allowing consistent and scalable operation. The method presented is scalable, i.e.: storage and communcation sizes do not grow with the size of the network. Results show the correct operation of the algorithm on complete 2-tree, 1-tree and mixed 1&2-tree networks, and show robustness of the 2-tree network against node and link failures.*

Keywords: Decentralised data fusion, global estimates, consistent fusion, scalability, survivability, network topologies, message passing algorithms, graphical models,k-trees, treewidth

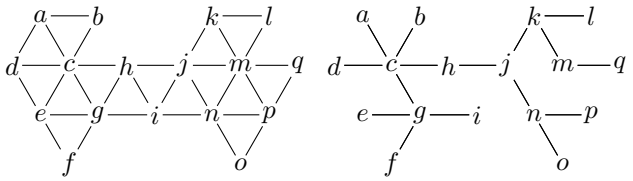
1 Introduction

The central problem in sensor networks is to provide a scalable method for consistent global data fusion. Decentralised data fusion (DDF) algorithms aim to provide a global estimate of a state observed by the nodes of an arbitrary sized sensor network. DDF methods seek a solution in which this estimate is obtained locally, in a decentralised manner, from local observations and from information communicated by neighbouring nodes. DDF methods impose constraints including no knowledge of the global network topology, no central fusion site, and no common communication medium. In return DDF methods provide scalability, modularity and survivability of the network.

In previous approaches to DDF [5, 10, 11, 4], exact decentralised estimation has generally only been possible in singly-connected or 1-tree networks. In these networks there is only one path between any two nodes and so local fusion is guaranteed to be consistent. Conversely, arbitrarily connected networks raise the possibility of information messages coming through multiple routes and, without data-tagging or extending message size, conservative fusion methods must be employed to avoid double-counting issues [6, 1, 4]. The problem with 1-tree networks is that they are vulnerable to the failure of any non-leaf node since this would leave the network in two separated components. 1-tree networks include both 'star' and chain topologies as well as conventional bi-directional trees.

This paper proposes an extension beyond single 1-trees for the network communication topology, into so called 2-tree network topologies. A 2-tree graph is made up of cliques of 3 nodes. Each adjacent pair of cliques overlaps at 2 nodes (a *junction* or *separator*). The overall graph of connections between the cliques is a tree. A 2-tree graph has *treewidth* of 2, so called because the separators are made of 2 nodes. The algorithm in this paper operates on mixed 1 and 2-tree networks. When given a 1-tree network to operate on, the algorithm gives identical results to prior 1-tree DDF methods. Example 2-tree and 1-tree topologies are shown in Figure 1.

The treewidth of a graph is well known for its role in



(a) Example 2-tree topology **(b)** An example 1-tree topology over the same nodes

Figure 1: Example 2-tree (a) and 1-tree (b) topologies

limiting the complexity of algorithms in graph theory [2, 7, 8], graphical models [3] and sparse linear algebra [12]. Given the strong effect of the treewidth on the complexity of the algorithms and network, we considered generalisations of 1-tree topologies into k-tree topologies, especially the next-highest k; $k = 2$. This k-tree approach is a computationally more feasible approach than more general topologies such as looped networks, meshes or broad unstructured topologies and, as this paper shows, is amenable to problems in decentralised estimation. The 2-tree topologies presented in this paper are more suited to the intent of a decentralised system, since they remain connected and correctly function despite single node or link failures.

This paper is part of ongoing research applying graphical models to the representation and solution of decentralised inference architectures and algorithms. In earlier work we considered junction-tree algorithms embedded on dynamic spanning trees [9]. The approach in this paper operates on 2-tree networks rather than spanning trees, and with the representation of each clique of 3 distributed across 3 nodes. These approaches both adapt the junction tree algorithm for use in decentralised data fusion.

Section 2 describes the proposed decentralised communications algorithm, section 2.1 describes the 2-tree properties which are exploited, and section 2.2 describes the decentralised algorithm. Section 3 shows results for the algorithm running, comparing a 2-tree to a comparable 1-tree topology.

2 Proposed Approach

The main contribution of this paper is in the decentralised topologies and algorithms for message passing. The algorithm allows each node to obtain the network global fused information using only local storage and communication. The communications algorithm actively limits the data sizes communicated and stored, leading to the scalable performance of the system. The goal of the topology and message passing is to produce a set of terms $p_i(x)$, such that the fusion of these is a consistent estimate for the state x:

$$p_{\cup}(x) = \frac{1}{c}\prod_i p_i(x) \tag{1}$$

These $p_i(x)$ are probabilities which are conditionally independent of each other given x, or equivalently, that they have independent errors.

Bayesian fusion of any variety on any state representation could be used. This paper assumes that the application is interested in obtaining estimates or totals derived from exact fusion of terms which are guaranteed to have independent errors. The actual underlying state and fusion process is not the focus of this paper.

The approach proposed in this paper guarantees against double-counting of information by using explicit "data-tagging" sets, but ensures scalability by summarising the data-tagging sets into a minimal size. This summarisation process exploits the global k-tree property and uses the local topology around the sending and receiving nodes. The necessary local topology properties are guaranteed by designing the global network topology to have a bounded treewidth of 2.

A data-tagging set is a set of separate probabilities, $p_i(x)$, each with a unique identifier. In this paper, any data-tagging set stores only conditionally independent terms, so equation 1 can be used on any data-tagging set to recover a fused estimate. Fusion of two or more data-tagging is performed as a *set union* followed by Bayesian fusion (equation 1). The set union step identifies any terms with matching labels and ensures that these are counted only once in the Bayesian fusion. Thus the data-tagging avoids double counting of information.

The approach proposed in this paper uses a very efficient, minimal form of data-tagging. This is in contrast to the inefficient *full data-tagging* appproach. In the full data-tagging method, each node maintains a set of independent information terms (conditionally independent of each other, given the true state), including its own sensor observations. In communicating out to any neighbour, the full set of information terms is sent. In receiving communication from a neighbour, the received set is merged (unioned) into the local set. This guarantees avoidance of double counting in arbitary network topology, but is expensive for large scale networks. Eventually every node's storage and every communicated set has the full list of independent information terms arising from every other node. For this simple but inefficient approach, the node storage and communication size is $O(n)$ for n nodes in the whole network. This increasing storage and communication size limits the scalability of the network for large n. Therefore this paper proposes an alternative scalable method.

The approach proposed in this paper is obtained by optimising the data-tagging approach to exploit the tree nature of the communications network. The communcations and storage scheme proposed in this paper achieves correct operation in 2-tree networks without using full data-tagging, thus obtaining a decentralised algorithm which is scalable in the number of nodes.

The key properties of the proposed approach are correct fusion, scalability and robustness against node & link loss. These properties are all closely related to the bounded treewidth network topology. The 2-tree topology guarantees robustness against a single node or link failure; The rest of the network will still achieve a global consistent estimate (but excluding any information contributed by the failed node). Under

other patterns of multiple node or link failures, the rest of the network will still function correctly as long as the network remains connected. This is a consequence of the 2-tree network topology together with the local data-tagging set based communication.

The key to the performance of the algorithm is in the choice of data-tag set communicated to the neighbours. If too many terms are included explicitly in the communicated set, this results in excess storage and communication size. If too few terms are included, the result is loss of global agreement. If too many terms are fused together inappropriately, this can lead to data looping ("gossiping"). The key aspect is to use the tree properties of the network to derive a procedure for minimally reducing the data-tag set, without compromising performance or data size.

To explain the communications algorithm, we first consider the k-tree network properties which are used in the algorithm.

2.1 k-Tree Network Properties

This section notes some important properties of the designed k-tree communications topology used for the decentralised data fusion algorithm proposed in this paper.

Separator Property

An important k-tree property is the existence of tree separators, as shown in Figure 2. In a k-tree any k-clique is a separator. Each separator divides the network into distinct parts. Within each part, the effect of all other parts can be marginalised (fused) onto the separator. Separators enable efficient summarisation of entire branches of the k-tree network. Separators use the k-tree separator property: in a k-tree, if any path between any two nodes i, k passes through the separator, then all paths between nodes i, k pass through the separator. These separators are used at the *borders* of the local neighbourhood $\mathcal{L}$ to summarise the fused total of the rest of the network beyond the local neighbourhood.

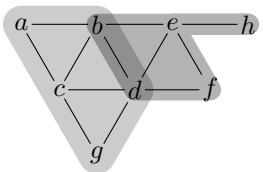


Figure 2: Illustration of the separator property. In a k-tree any k-clique is a separator. Each separator divides the network into two parts. In this figure, $b - d$ is the separator. The two parts and the intersection are shown. Within each part, the effect of the other part can be marginalised (fused) onto the separator. Separators enable efficient summarisation of entire branches of the k-tree network.

Local Neighbourhood Property

In k-tree networks, the local neighbourhood around each vertex is an efficient local summary of the relevant parts of the global topology.

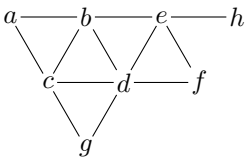
The k-tree networks allow an efficient decentralised & local neighbourhood representation to serve as the only required topology awareness at the nodes. This is important for scalability, allowing the representation of a global network with only small local neighbourhood representations. The local neighbourhood is therefore an important data structure used in the algorithm proposed in this paper.

At any node, $\mathcal{V}_i$, the *local neighbourhood subgraph* consists of $\mathcal{V}_i$, the neighbours of $\mathcal{V}_i$ and the links and cliques between them, as shown in Figure 3.

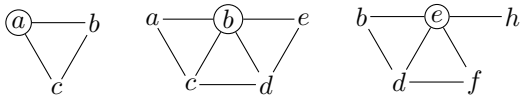
The local neighbourhood representation is motivated by the k-tree "junction path covering property": in a k-tree, if any path between any two nodes i, k passes through the local neighbourhood of a node j, then all paths between nodes i, k pass through the local neighbourhood of j.

This junction path covering property means that the local neighbourhood around a node j has control over how any messages can pass from one side to the other. The local neighbourhood encodes which neighbours to communicate with, which information terms must be maintained separately in data-tag sets (for correctness) and which terms can be fused into others (for scalability).

For 1-tree topologies used in prior works, the local-neighbourhood representation is simply the list of neighbouring vertices and list of the corresponding edges to those neighbours.



(a) Global network topology



(b) Local neighbourhood representations, $\mathcal{L}$, at a, b, e respectively

Figure 3: Illustration of the local neighbourhood representation, $\mathcal{L}$. In k-tree networks, the local neighbourhood $\mathcal{L}$ is an efficient local summary of the relevant parts of the global topology

In the next section, these k-tree properties are applied to the decentralised communications algorithm.

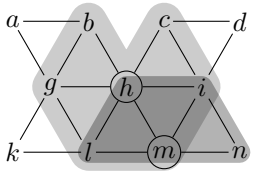
2.2 Communications Algorithm

Given the above k-tree properties, this section explains the decentralised communications algorithm. The algorithm will be described by referring to the sending node, $\mathcal{V}_\mathrm{t}$ ("this vertex") and the receiving node $\mathcal{V}_\mathrm{d}$ ("destination vertex"). The sending node has access to its own local neighbourhood graph, $\mathcal{L}$, and its own local data-tag set. The communications algorithm works by customising a *reduced* data-tag set to send to the destination, $\mathcal{V}_\mathrm{d}$.

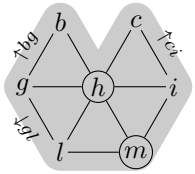
The communications algorithm is simply stated as follows: The data-tag set is *summarised* (marginalised) into the *intersection* of the local and destination neighbourhoods. This is illustrated in Figure 4.

The local neighbourhood graph is used to determine the *intersection* data-tags to send to the destination, and also to summarise terms *outside the intersection* into separators. On transmission of the data-tag set from $\mathcal{V}_\mathrm{t}$ to $\mathcal{V}_\mathrm{d}$, only terms in the *intersection* of their neighbourhoods need to be individually data-tagged. Terms in the local neighbourhood at the sending node but *outside the intersection* can be fused into summary terms, hence maintaining locality and efficiency.

The local neighbourhood representation, $\mathcal{L}$, is very important for this process for several reasons: (1): $\mathcal{L}$ is used to calculate the intersection and separator sets, (2): the local data-tag set stored at each node is guaranteed to fit into $\mathcal{L}$, the local neighbourhood graph (meaning that the set of node and separator data-tags is at most the same as in $\mathcal{L}$). (3): the marginalisation of $\mathcal{L}$ is used to summarise out the irrelevant parts.



(a) The full network, highlighting neighbourhoods of $\mathcal{V}_\mathrm{h}$ and $\mathcal{V}_\mathrm{m}$ and their intersection



(b) The network beyond the immediate neighbours can be summarised within the neighbourhood

(c) To prepare a communication set, $\mathcal{V}_\mathrm{h}$ can marginalise its local neighbourhood network into the intersection with neighbour $\mathcal{V}_\mathrm{m}$ (resulting in the left hand set). This set is communicated to $\mathcal{V}_m$. $\mathcal{V}_m$ takes the *union* of the received set (left) with its local set (right).

Figure 4: Summary of the communications algorithm. The local neighbourhood is marginalised i.e.: *summarised* down into the *intersection* set for the destination. This summarised intersection set is sent to the destination and merged into the local set.

Algorithm 1: Compute the communication output

Input: $\mathcal{L}$
:a copy of the local neighbourhood graph
Input: $\mathcal{V}_t$: this node in $\mathcal{L}$
Input: $\mathcal{V}_d$: the destination neighbour in $\mathcal{L}$
Input: `localTags`: the local data-tag set
Result: `destTags`: the output data-tag set to send

1. Initial case: No summarisation:
   Copy `destTags` ← `localTags`
2. Avoid redundancy, delete terms involving $\mathcal{V}_d$:
   Erase term $\mathcal{V}_d$ from `destTags`
   Erase any terms for $\mathcal{V}_d$ separators from `destTags`
3. Summarise away parts not local to $\mathcal{V}_d$:
   Determine the region to summarise out, $\mathcal{S}$:
   $\mathcal{S}$ is all vertices in $\mathcal{L}$ except $\mathcal{V}_d$ and its neighbours
   while $\mathcal{S}$ is not empty **do**
      Find a leaf vertex $\mathcal{V}_l$ of $\mathcal{L}$ in $\mathcal{S}$
      Marginalise out $\mathcal{V}_l$, updating `destTags`
      Erase $\mathcal{V}_l$ from $\mathcal{S}$

The communications algorithm is summarised in algorithm 1. In step 1, the algorithm initally copies the local data-tag set to the output data-tag set. This corresponds to a full data-tagging solution. The subsequent steps erase and or summarise some of the entries, thus ensuring scalability. For step 2: data-tag terms involving the destination vertex are redundant and can be explicitly deleted. Step 3 marginalises out any data-tag terms which are not neighbours of the destination vertex. This is explained in the following section.

Marginalisation Process

This section explains the marginalisation process which summarises non-local information, in algorithm 1.

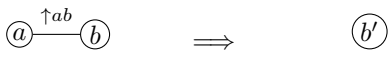
Marginalisation proceeds from a so called "leaf vertex", each marginalisation step reduces the size of the data-tag set, effectively eliminating the leaf vertex. For this paper on 2-tree networks, a leaf vertex is either a "1-tree leaf" or a "2-tree leaf". Any vertex with exactly one edge is a 1-tree leaf. Any vertex which is part of exactly one hyperedge (clique of 3) is a 2-tree leaf.

Marginalisation of a 2-tree leaf:

$$\uparrow bc + a + \uparrow ab + \uparrow ac \Longrightarrow \uparrow bc' \qquad (2)$$

In the marginalisation of a 2-tree leaf above, vertex a is deleted, along with separator terms $\uparrow ab$ and $\uparrow ac$. This marginalises the 2-tree leaf into an edge, leaving a new 1-tree leaf. The separator is updated by the marginalisation via: $\uparrow bc' = \uparrow bc + a + \uparrow ab + \uparrow ac$.

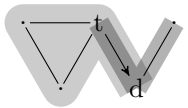
Marginalisation of a 1-tree leaf:



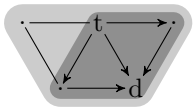
$$b + a + \uparrow ab \Longrightarrow b' \tag{3}$$

In the marginalisation of a 1-tree leaf above, vertex a is deleted, along with separator terms $\uparrow ab$. This marginalises the 1-tree leaf into a single vertex. Vertex b is updated by the marginalisation via: $b' = b + a + \uparrow ab$

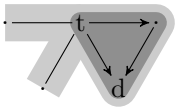
Some example cases of the above communications algorithm are discussed below.



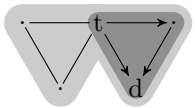
(a) $\mathcal{V}_\mathrm{t}$ to $\mathcal{V}_\mathrm{d}$ is a 1-tree link (edge)



(b) $\mathcal{V}_\mathrm{t}$ to $\mathcal{V}_\mathrm{d}$ is a 2-tree link



(c) $\mathcal{V}_\mathrm{t}$ to $\mathcal{V}_\mathrm{d}$ is a 2-tree link with some 1-tree branches on $\mathcal{V}_\mathrm{t}$



(d) $\mathcal{V}_\mathrm{t}$ to $\mathcal{V}_\mathrm{d}$ is a 2-tree link, with another 2-tree clique on $\mathcal{V}_\mathrm{t}$ not involving $\mathcal{V}_\mathrm{d}$

Figure 5: Illustration of various topologies from $\mathcal{V}_\mathrm{t}$ to $\mathcal{V}_\mathrm{d}$. These are examples from the same underlying neighbourhood marginalisation algorithm. Shadings show the neighbourhoods of $\mathcal{V}_\mathrm{t}$, $\mathcal{V}_\mathrm{d}$ and their intersection. The possible data flow paths, and hence the necessary data-tagging follow from these neighbourhood intersections

1-Tree Neighbour

A 1-tree link (an edge) between $\mathcal{V}_t$ and $\mathcal{V}_d$ is the *unique* path from $\mathcal{V}_t$ to $\mathcal{V}_d$, as shown in Figure 5a and therefore the algorithm (1) can marginalise out all other data-tags and send only one fused term to the destination. When given a 1-tree network to operate on, this leads to identical results as for 1-tree DDF methods.

2-Tree Neighbour

For cases involving 2-tree cliques, the direct path between $\mathcal{V}_t$ and $\mathcal{V}_d$ is not the unique path, there are many possible paths. However, the k-tree property of the network guarantees that all possible paths are contained within the local neighbourhood subgraph $\mathcal{L}$ at $\mathcal{V}_t$. The main contribution of this paper is in this stage, in the computation of the correct data-tag set to send to the neighbouring node.

There are 3 components added into the output data-tag set by algorithm 1:

1. The local information of nodes in the common neighbourhoods of $\mathcal{V}_t$ and $\mathcal{V}_d$. The terms for the common neighbours are included in the communicated data-tag set because this allows duplicated forwarding of information via other nodes in the k-tree network, allowing the network to survive communications link failures.

   These terms must be included separately, explicitly in the data-tag set (i.e.: not fused together), since the k-tree network has many loops within the local neighbourhood.

2. Marginalised branches off $\mathcal{V}_t$ (such as in Figure 5 (a), (c) and (d)), which are marginalised into additional terms on $\mathcal{V}_t$. No looped data flows can be caused by these branches, since all such data paths flow through $\mathcal{V}_t$, and therefore it is safe and efficient to fuse these into the same output term as for $\mathcal{V}_t$.

3. Marginalised *separator* terms for the fusion of the rest of the network "upstream" past each clique separator. The values of nodes and separators *outside* the intersection with the destination must be fused into the intersection set to be sent to the destination. This result of this process is shown in Figure 4 as the transition from (b) to (c).

3 Results

This paper contributes an algorithm for decentralised data fusion in 2-tree (and mixed 1 and 2-tree) networks. The algorithm has been implemented in c++ and demonstrated on a variety of topologies. Results show the correct operation of the algorithm on 2-tree networks, mixed 1 and 2-tree networks, and operation with missing nodes and links. The results are compared against 1-tree networks.

An example topology is shown in Figure 6, together with tables showing the information terms stored and communicated in the network. The topology in Figure 6 contains a mixture of 2-tree and 1-tree connections. The algorithm works correctly on this topology, thus showing that the algorithm correctly generalises the 1-tree case. Table 6(b) shows how the nodes store the neighbouring independent information, together with fused terms for the fusion of the "rest of network" past their bordering interfaces. Storage data are all local to the neighbourhood of the node, which is thus highly scalable.

Table 6(c) shows some examples of the data which nodes communicate. Communicated terms are always in the intersection of the neighbourhoods of the nodes, thus ensuring the scalability of the communications data size for larger networks.

The results of the algorithm operating are shown in Figures 8 (for normal operation), 9 (operation with a missing node) & 10 (operation with a missing link). The topologies used for these are shown in Figure 7. In this example, the nodes are initialised each with 1 unit of

independent observation information. This example uses a simple scalar accumulation, which serves as a proxy for summation of information for fusion of Gaussian probability distributions, summation of information for log-likelihood discrete distributions or similar data fusion algorithms. The global total information is 17.

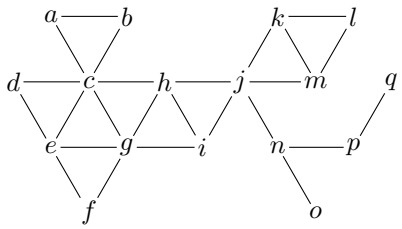
In the normal operation, Figure 8, both the 2-tree topology and the 1-tree topology operate comparably, achieving the global total on all nodes. The strength of the 2-tree topology is shown in Figure 9. In the scenario in Figure 9, the node c is disabled. Figure 9 shows how the 2-tree topology continues as normal, achieving the correct total global information from the network. By comparison, Figure 9 shows that the nodes in the 1-tree topology are only able to achieve the totals for their connected component, thus failing to reach the required global total.

Similarly, Figure 10 shows the correct operation of the 2-tree where a link $h \leftrightarrow j$ is disabled. In both cases the algorithm expects the link to be present, but it is disabled. For the 1-tree topology, the network is split in two by the failed link, and again the nodes fail to achieve the global total. But for the 2-tree topology, there is redundancy in the network via links $h \leftrightarrow i$ and $i \leftrightarrow j$, and thus the 2-tree topology is able to pass the correct information and achieve the required global total information.

In the proposed algorithm, no explicit checks or case handling is required for missing nodes or links; The 2-tree algorithm continually sends information on all the redundant links. This leads to circulation of messages, but the algorithm handles the tagging of information in a correct and efficient manner.

4 Conclusion

This paper presented an algorithm for scalable decentralised data fusion, using exact, direct methods to maintain account of independent information terms, avoiding double-counting information and/or conservative fusion. Instead of earlier methods based on tree topologies, this paper presented a method based on more general 2-trees. These 2-tree topologies are more connected than 1-trees, but much sparser than fully-connected networks. These 2-tree network topologies remain connected despite loss of a node or communications link. This paper contributed an algorithm for the decentralised communication and the accounting for common-information necessary to allow correct and scalable operation under 2-tree networks. The method presented is scalable, i.e.: node storage and communcation data sizes do not increase with the size of the network. Results showed the correct operation of the algorithm on complete 2-tree, 1-tree and mixed 1&2-tree networks, with robustness against node and link failures.



(a) Topology with mixed 1 & 2-tree connections, used for the tables below.

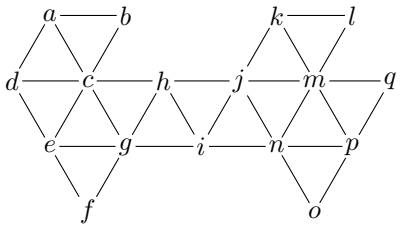
N:	Stored (term,value) data	$\sum$
a	$(a,1)$ $(b,1)$ $(c,15)$	17
c	$(a,b,c,d,e,g,h,\uparrow ge,1)$ $(\uparrow hg,9)$	17
e	$(d,e,f,g,1)$ $(c,3)$ $(\uparrow gc,10)$	17
g	$(e,f,g,h,i,\uparrow ec,1)$ $(c,3)$ $(\uparrow ih,8)$	17
j	$(h,i,j,k,m,\uparrow mk,1)$ $(n,4)$ $(\uparrow ih,7)$	17
n	$(n,o,1)$ $(p,2)$ $(j,13)$	17

(b) Examples of the information stored at various nodes in the topology above, at completion of the algorithm. For example, node c stores independent individual information for its neighbours (a, b, d, e, g, h & itself c, each with value 1), and stores fused information terms for the "rest of network" ($\uparrow ge$ value 1 & $\uparrow hg$ value 9). Storage data are all local to the neighbourhood of the node, thus highly scalable.

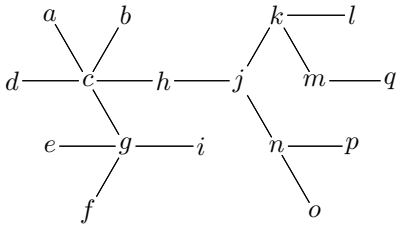
Node pair	Communicated (term,value) data:
$a \rightarrow b$	$(a,1)$ $(c,15)$
$b \rightarrow a$	$(b,1)$ $(c,15)$
$c \rightarrow d$	$(e,1)$ $(c,3)$ $(\uparrow ec,12)$
$c \rightarrow h$	$(c,3)$ $(g,1)$ $(\uparrow gc,3)$
$c \rightarrow g$	$(e,h,1)$ $(c,3)$ $(\uparrow ec,1)$
$d \rightarrow c$	$(d,1)$ $(e,1)$
$h \rightarrow c$	$(g,h,1)$ $(\uparrow hg,9)$
$g \rightarrow c$	$(e,g,h,\uparrow ge,1)$ $(\uparrow hg,9)$
$j \rightarrow m$	$(k,1)$ $(j,14)$
$j \rightarrow n$	$(j,13)$
$j \rightarrow i$	$(h,1)$ $(j,8)$

(c) Examples of the information communicated between node pairs. For example, h sends to c independent individual information for g & h and fused information for the "rest of network beyond h & g", $\uparrow hg$. Nodes with 1-tree connections, such as $j \rightarrow n$ send only a single fused information term. Communicated data are limited to the intersections of the neighbourhoods of the nodes, thus highly scalable.

Figure 6: A topology with mixed 1 & 2-tree connections, and the node storage and communication sets generated by the algorithm. This topology has mixed 1-tree regions and 2-tree regions, thus showing the generality of the algorithm for both 1 and 2-tree topologies.



(a) 2-tree test topology



(b) A 1-Tree topology over the same nodes

Figure 7: Topologies compared for the results. (a) shows a 2-tree topology. (b) is a comparable 1-tree topology over the same nodes, with approximately the same span. Results are shown in Figures 8, 9 & 10

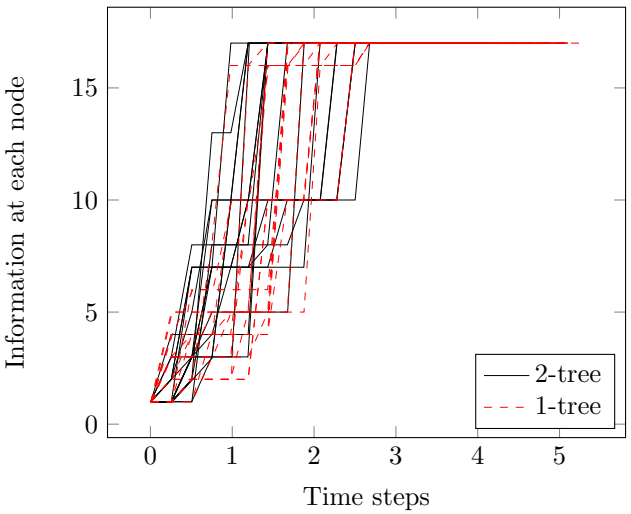


Figure 8: Normal Operation. Each node's information versus time. The topologies are from Figure 7. Each node starts with its own local information (1 unit at $t = 0$), increasing as they acquire information through the decentralised communication. At completion, all nodes have the global total information (17 units).

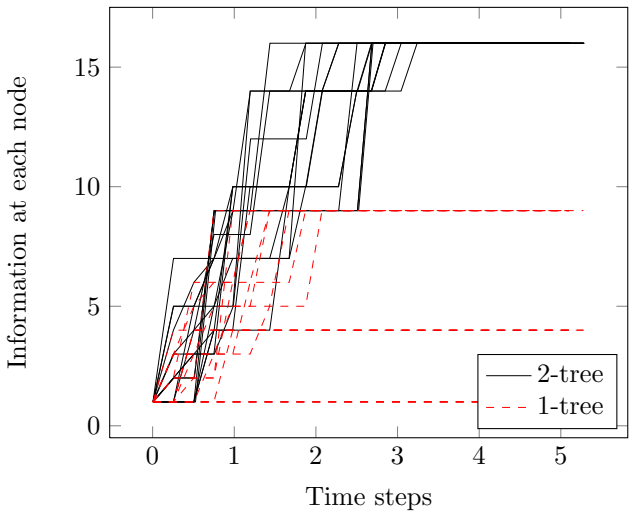


Figure 9: 2-tree versus 1-tree topologies from Figure 7, each with a missing node, c. In the 1-tree case the missing node breaks the network into several components, and therefore none reach the required global fusion. In the 2-tree case, the missing node does not separate the network, and the global fused estimate is obtained.

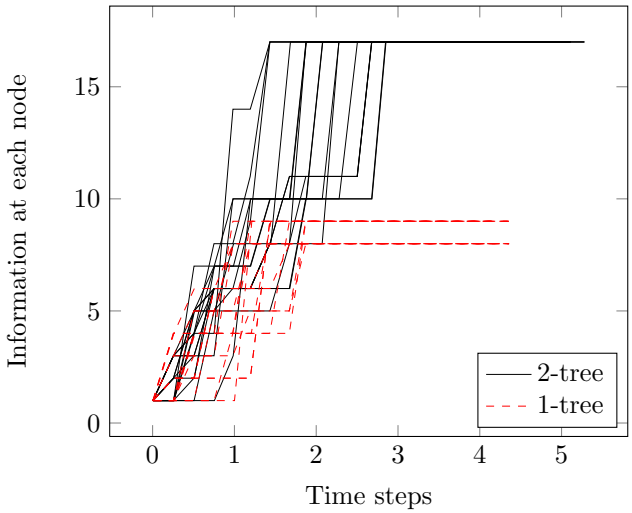


Figure 10: 2-tree versus 1-tree topologies from Figure 7, each with a missing link ($h \leftrightarrow j$). In the 1-tree case the missing link breaks the network into two components, and therefore none reach the required global fusion. In the 2-tree case, the missing link does not separate the network, and the global fused estimate is obtained.

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] A. R. Benaskeur, "Consistent fusion of correlated data sources," *IECON Proceedings (Industrial Electronics Conference)*, vol. 4, pp. 2652–2656, 2002. [Online]. Available: http://dx.doi.org/10.1109/IECON.2002.1182812

[2] B. Bollobás, *Modern Graph Theory*. Springer, 1998.

[3] V. Chandrasekaran, N. Srebro, and P. Harsha, "Complexity of inference in graphical models," *24th Conference on Uncertainty in Artificial Intelligence and Statistics*, 2008.

[4] K. Chang, C. Chong, and S. Mori, "On scalable distributed sensor fusion," Cologne, Germany, 2008. [Online]. Available: http://dx.doi.org/10.1109/ICIF.2008.4632322

[5] H. Durrant-Whyte and Mike Stevens, "Data fusion in decentralised sensing networks." in *4th International Conference on Information Fusion*, Montreal, Canada, 2001.

[6] S. J. Julier and J. K. Uhlmann, "A non-divergent estimation algorithm in the presence of unknown correlations," *Proceedings of the American Control Conference*, vol. 4, pp. 2369–2373, 1997. [Online]. Available: http://dx.doi.org/10.1109/ACC.1997.609105

[7] T. Kloks, *Treewidth, Computations and Approximations*, ser. Lecture Notes in Computer Science, 1994.

[8] E. Korach and N. Solel, "Tree-width, path-width, and cutwidth," *Discrete Applied Mathematics*, vol. 43, no. 1, pp. 97–101, 1993.

[9] A. Makarenko, A. Brooks, T. Kaupp, H. Durrant-Whyte, and F. Dellaert, "Decentralised data fusion: A graphical model approach," Seattle, WA, United states, 2009, pp. 545–554.

[10] E. Nettleton, "Decentralised architectures for tracking and navigation with multiple flight vehicles," Ph.D. dissertation, Australian Centre for Field Robotics, Department of Aerospace, Mechanical and Mechatronic Engineering, The University of Sydney, 2003.

[11] D. Nicholson, C. Lloyd, S. Julier, and J. Uhlmann, "Scalable distributed data fusion," in *Information Fusion, 2002. Proceedings of the Fifth International Conference on*, vol. 1, 2002, pp. 630–635 vol.1.

[12] M. A. Paskin and G. D. Lawrence, "Junction tree algorithms for solving sparse linear systems," University of California, Berkeley., Tech. Rep. UCB/CSD-03-1271, 2003.