New Techniques for Incremental Data Fusion in Distributed Sensor Networks

Damianos Gavalas[1], Charalampos Konstantopoulos[2], Basilis Mamalis[3], Grammati Pantziou[4]

[1]Department of Cultural Technology and Communication, University of the Aegean, Mytilini, Lesvos Island, Greece
dgavalas@aegean.gr

[2]Research Academic Computer Technology Institute, Patras, Greece
konstant@cti.gr

[3]Department of Informatics, Technological Educational Institution of Athens, Athens, Greece
{pantziou, vmamalis}@teiath.gr

Abstract

The use of mobile agents for data fusion in wireless sensor networks has been recently proposed in the literature to answer the scalability problem of client/server model. In this article, we consider the problem of calculating a near-optimal route for a mobile agent that incrementally fuses the data as it visits the nodes in a distributed sensor network. The order of visited nodes affects not only the quality but also the overall cost of data fusion. We propose two heuristic algorithms that adapt methods usually applied in network design problems in the specific requirements of sensor networks. They suggest the proper number of MAs that minimizes the overall data fusion cost and construct near-optimal itineraries for each of them.

Keywords: Sensor networks, mobile agents, Constrained Minimum Spanning Trees

1. Introduction

Multiple sensor data fusion is an evolving technology, concerning the problem of how to fuse data from multiple sensors in order to make a more accurate estimation of the environment [Qi et. Al. (2001)]. It improves reliability while offering the opportunity to minimize the data retained. Applications of data fusion cross a wide spectrum, including environment monitoring, automatic target detection and tracking, battlefield surveillance, remote sensing, global awareness, etc [Akyildiz et. Al. (2002)]. They are usually time-critical, cover a large geographical area, and require reliable delivery of accurate information for their completion. Most energy-efficient proposals are based on the traditional client/server computing model to handle multisensor data fusion in

Distributed Sensor Networks (DSNs); in that model, each sensor sends its sensory data to a back-end processing element (PE) or sink. However, as advances in sensor technology and computer net-working allow the deployment of large amount of smaller and cheaper sensors, huge volumes of data need to be processed in real-time. In this paper, we propose the usage of mobile agents in DSNs for data fusion tasks as an alternative to the traditional client/server model.

The remainder of the paper is organized as follows: Section 2 reviews works related to our research. Section 3 discusses the design and functionality of the first of our heuristic algorithms for designing near-optimal itineraries for mobile agents performing data fusion tasks in DSNs. In Section 4, we present our second approach for constructing the itineraries that will be followed by mobile agents. Finally, Section 5 concludes the paper and presents future directions of our work.

2. Related Work

Mobile agent (MA) technology has been proposed as an answer to the scalability problems of centralized models. The term MA refers to an autonomous program with the ability to move from host to host and act on behalf of users towards the completion of a given task [Milojicic et. Al. (1999)]. DSN environments form a promising application area for MAs; yet, they pose new challenges as the link bandwidth is typically much lower than that of a wired network and sensory data traffic may even exceed the network capacity.

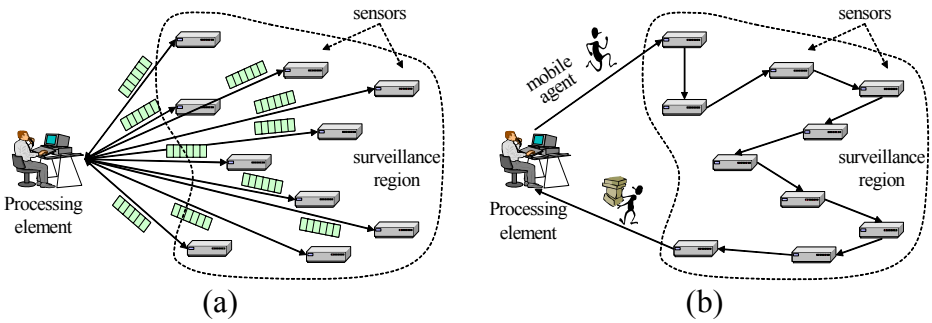


Figure 1. *Centralized vs. Mobile Agents-based data fusion in Distributed Sensor Networks*

To solve the problem of the overwhelming data traffic, [Qi et. Al. (2001)] and [Qi et. Al. (2001)] proposed the use of MAs for scalable and energy-efficient data aggregation. By transmitting the software code (MA) to sensor nodes, a large amount of sensory data may be filtered at the source by eliminating the redundancy. MAs may visit a number of sensors and progressively fuse retrieved sensory data, prior to returning to the PE to deliver the data. This scheme proves more efficient than

traditional client/server model, wherein row sensory data are transmitted to the PE where data fusion takes place (see Fig. 1).

To the best of our knowledge, only [Qi et. Al. (2001)] and [Wu et. Al. (2004)] deal with the problem of designing optimal MA itineraries in the context of DSNs. In [Qi et. Al. (2001)], Qi and Wang proposed two heuristic algorithms to optimize the itinerary of MAs performing data fusion tasks. In Local Closest First (LCF) algorithm, each MA starts its route from the PE and searches for the next destination with the shortest distance to its current location. In Global Closest First (GCF) algorithm, MAs also start their itinerary from the PE node and select the node closest to the center of the surveillance region as the next-hop destination.

The output of LCF-like algorithms highly depends on the MAs original location, while the nodes left to be visited last are associated with high migration cost [Kershenbaum (1993)]; the reason for this is that they search for the next destination among the nodes adjacent to the MA's current location, instead of looking at the 'global' network distance matrix. On the other hand, GCF produces in most cases messier routes than LCF and repetitive MA oscillations around the region center, resulting in long route paths and undesirable performance [Qi et. Al. (2001)][Wu et. Al. (2004)].

Wu et al proposed a genetic algorithm-based solution for computing routes for an MA that incrementally fuses the data as it visits the nodes in a DSN [Wu et. Al. (2004)]. Although providing superior performance (lower cost) than LCF and GCF algorithms, this approach implies a time-expensive optimal itinerary calculation (genetic algorithms typically start their execution with a random solution 'vector' which is improved as the execution progresses), which is unacceptable for time-critical applications, e.g. in target location and tracking.

Most importantly, both the approaches proposed in [Qi et. Al. (2001)] and [Wu et. Al. (2004)] involve the use of a single MA object launched from the PE station that sequentially visits all sensors, regardless of their physical location on the plane. Their performance is satisfactory for small DSNs; however, it deteriorates as the network size grows and the sensor distributions become more complicated. This is because the MA's round-trip delay increases linearly with network size, while the overall migration cost increases exponentially as the traveling MA accumulates into its state data from visited sensors [Fuggeta et. Al. (1998)]. The growing MA's state size not only results in increased consumption of the limited wireless bandwidth, but also consumes the limited energy supplies of sensor nodes.

Our algorithms have been designed on the basis of thee objectives: (a) MA itineraries should be derived as fast as possible and adapt quickly to changing networking

conditions (hence, efficient heuristics are needed), (b) MA itineraries should include only sensors with sufficient energy availability and exclude those with low energy level, (c) The number of MAs involved in the data fusion process should depend on the number and the physical location of the sensors to be visited; the order an MA visits its assigned nodes should be computed in such a way as to minimize the overall migration cost.

2. The First Near–Optimal Itinerary Design (FNOID) Algorithm

The problem of designing optimal itineraries is similar to traditional network design problems such as the Constrained Minimum Spanning Trees (CMST) problems [Kershenbaum (1993)]. Our First NOID (Near–Optimal Itinerary Design) algorithm adapts some basic ideas of Esau-Williams (E-W) algorithm [Esau et. Al. (1966)] in the requirements of itinerary planning problem.

The cost function used in E-W algorithm considers selected links cost as the only contributing factor to the total itinerary cost. This is certainly not adequate metric to evaluate the cost of agents itineraries c_{total}. A key factor also affecting c_{total} is the agent size; more importantly, the agent size increment rate [Fuggeta et. Al. (1998)], which depends on the amount of data collected by the MA on every sensor. Let us assume that a set of itineraries $I = \{I_0, I_1, \ldots, I_{k-1}\}$ is constructed, each assigned to an individual MA object *i*. Each itinerary I_m includes a set of sensors to be sequentially visited by a single MA: $I_m = \{S_0, S_1, \ldots, S_n, S_0\}$. Note that all itineraries originate and terminate at the PE node S_0. The total cost per polling interval over all itineraries $|I|$ becomes:

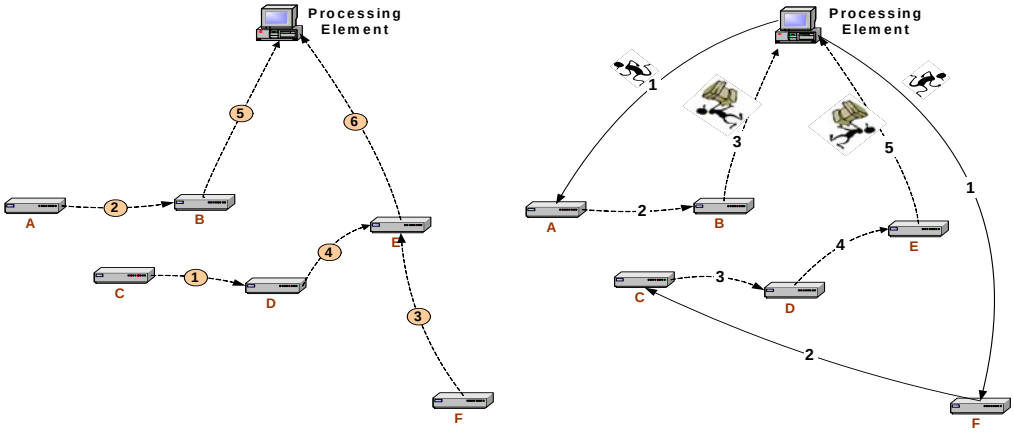
$$c_{total} = \sum_{i=0}^{|I|-1} \sum_{j=0}^{|I_i|-1} \left(d_{ij} + s_i \right) \cdot c_{ij} \qquad \textbf{(1)}$$

where d_{ij} is the amount of data collected by the i^{th} MA on the first j visited sensors, s_i the MA initial size and c_{ij} the cost of utilizing the link traversed by the MA i on its j^{th} hop, i.e. the wireless link connecting sensors S_j and S_{j+1} (c_{ij} is given by the network cost matrix). In principle, the FNOID algorithm aims at constructing a set of itineraries I minimizing the cost function of equation (1).

An output of the FNOID algorithm is illustrated in Fig. 2. The algorithm's output for the particular DSN configuration of Fig. 2 is based on the cost matrix presented in Table 1. In our prototype implementation, the calculation of the DSN cost matrix entries is only based on the spatial distance between sensors. This decision approach has been taken because the transmission power (hence, energy) required to transmit data between pairs of sensors increases linearly with their physical distance [Akyildiz

et. Al. (2002)][Wu et. Al. (2004)]. However, as a future extension, we intend to incorporate sensor energy availability metric in the calculation of cost matrix values.

The itinerary design algorithm is executed at the PE node; this is a reasonable choice since an MA always starts its data collection journey from the PE node, which can usually be equipped with more powerful computing resources than regular sensor nodes. The PE node has the predetermined knowledge necessary for performing the global optimization, such as the geographical locations (through GPS interfaces) and transmitting/receiving parameters of sensor nodes.



(a) (b)

Figure 2. *(a) Output of FNOID (the sequence numbers indicate the order in which the corresponding MA migrations are accepted, i.e. the algorithm's iteration sequence numbers) (b) MA itineraries derived from the FNOID algorithm's output.*

Unlike LCF and GCF algorithms, FNOID takes into account the amount of data accumulated by MAs at each visited sensor (without loss of generality, we shall assume this is a constant d). Namely, it recognizes that travelling MAs become 'heavier' while visiting sensors without returning back to the PE to 'unload' their collected data [Fuggeta et. Al. (1998)]. Therefore, FNOID promotes small itineraries enabling the parallel employment of multiple cooperating MAs, each visiting a subset of sensors.

Specifically, the aim of FNOID algorithm is, given a set of sensors $S = \{S_0, S_1, \ldots, S_{n-1}\}$, the PE node S_0 and the cost matrix C, to return a set of near-optimal itineraries $I = \{I_0, .., I_k\}$, all originated and terminated at the PE. Initially, we assume $|S|$ $(= n)$ itineraries $I_0, .., I_{n-1}$, as many as the network nodes, each containing a single host (S_0,

S_1, …, S_{n-1}, respectively). On each algorithm step, two nodes *i* and *j* are 'connected' and, as a result, the itineraries $I(i)$ and $I(j)$ including these hosts respectively are merged into a single itinerary.

As mentioned in Section 2, LCF and GCF algorithms usually fail as they tend to leave hosts located far from the center stranded since they prioritize the inclusion of hosts closed to last selected host or the center. As a result, relatively expensive links are left last to be included in the solution, significantly increasing the overall cost. A way of dealing with this problem is to pay more attention to nodes far from the center, giving preference to links incident upon them. FNOID algorithm accomplishes this by using the concept of 'tradeoff function' $t_{i,j}$ associated with each link (i, j), defined by:

$$t_{i,j} = c_{i,j} + \sum_{k=1}^{|I(i)|+|I(j)|} d - C_{i,S_0} \qquad (2)$$

where c_{ij} is the cost of link connecting nodes *i* and *j*.

Table 1. *Cost matrix of the DSN shown in Fig. 2*

	S_0	A	B	C	D	E	F
S_0	-	50	40	62	56	42	88
A		-	22	24	58	73	177
B			-	22	21	27	130
C				-	19	39	131
D					-	18	80
E						-	73
F							-

The concept of the tradeoff function is introduced in E-W algorithm, defined as follows: $t_{i,j} = c_{i,j} - C_{i,S_0}$. Equation (2) extends and adapts this function in the specific requirements of agent itinerary planning problem. In particular, the inclusion of a parameter representing the amount of data collected from each host (d) and also the number of hosts already included in the itineraries considered for merging, i.e. $|I(i)|$ and $|I(j)|$, obstructs the construction of large itineraries, thereby promoting the formation of multiple itineraries, assigned to separate MAs. Equation (2) implies that the more nodes an itinerary already includes the more difficult for a new host to become part of that itinerary, especially when d is large.

In equation (2), C_{i,S_0} is the cost of connecting $I(i)$ to the PE S_0. Initially, this is simply the cost of connecting node *i* directly to the PE. As *i* becomes part of an itinerary containing other sensors, however, this changes to:

$$C_{i,S_0} = \min_{k \in I(i)} c_{k,S_0} \quad (3)$$

On each algorithm's step, trade-off function values $t_{i,j}$ are evaluated for all pairs (i,j), except of those where nodes i and j are already part of the same itinerary; the 'itineraries' including the nodes that produce the minimum $t_{i,j}$ value are merged. For instance, if the tradeoff function is minimized for the pair of nodes m and n, then $I(m)$ and $I(n)$ are merged into one itinerary. When FNOID's execution finishes, one or more 'sub-trees' (groups of nodes) rooted at the PE node have been constructed; this is shown on Fig. 2a, where the sequence numbers enclosed within circles indicate the order in which individual links (or migrations) become accepted in the corresponding algorithm steps. It is then a trivial task to produce the itineraries (started and terminated at the PE node) for traversing the nodes of each sub-tree; these itineraries correspond to a post-order traversal of the sub-trees (shown in Fig. 2b).

3. The Second Near–Optimal Itinerary Design (SNOID) Algorithm

Now, we present the Second Near – Optimal Itinerary Design (SNOID) algorithm for determining the number of Mobile Agents (MA) that should be used and the itineraries these MA should follow. The main idea is to partition the area around the Processing Element (PE) into concentric zones (Fig. 3) and start building the MA paths with direction from the inner (close to PE) zones to the outer ones. The radius of the first zone which includes the PE is equal to αr_{max} where α is an input parameter in the range (0,1] and r_{max} is the maximum transmission range of any sensor node. All sensor nodes inside this region are connected directly to the PE and these nodes will be the starting points of the itineraries of the MAs. This also implies that the PE will create as many MAs as the number k of these nodes (k=3 in our example). With the inclusion of parameter α, we can control the amount of energy needed for the nodes of the first zone to communicate with the PE.

So, compared with the FNOID algorithm, the SNOID algorithm determines the number of MAs differently by taking only into account the cost of communication between the PE and the first nodes of the itineraries. The basic assumption here is that we should use as much MAs as possible, ideally one for each sensor node, and the only restriction on doing this is the energy cost required for direct communication of the PE with the first nodes of the itineraries.

With regard to the remaining zones, these have a constant width equal to $r_{max}/2$. So, each sensor node in a zone can only communicate with the nodes of the same zone as well as the nodes of the two adjacent zones. The end result of our technique will be k trees each having as a root one of the k nodes of the first inner zone. Then, the

itineraries of the MAs are derived easily, by the depth-first traversal of these trees, short-cutting the route whenever possible.

The whole processing consists of two phases. In the first phase, we start from the inner zones and proceed to the outer zones. When visiting a zone, we try to connect each of the nodes with a node of the previous as well as the current zone which already has a connecting path back to the PE. During this process, we pay attention to the latency of the formed trees up to that point, and we do not connect a node to a tree if the total walk time after this insertion on tree will exceed a certain threshold *W*. In the following, we give more details of our technique.

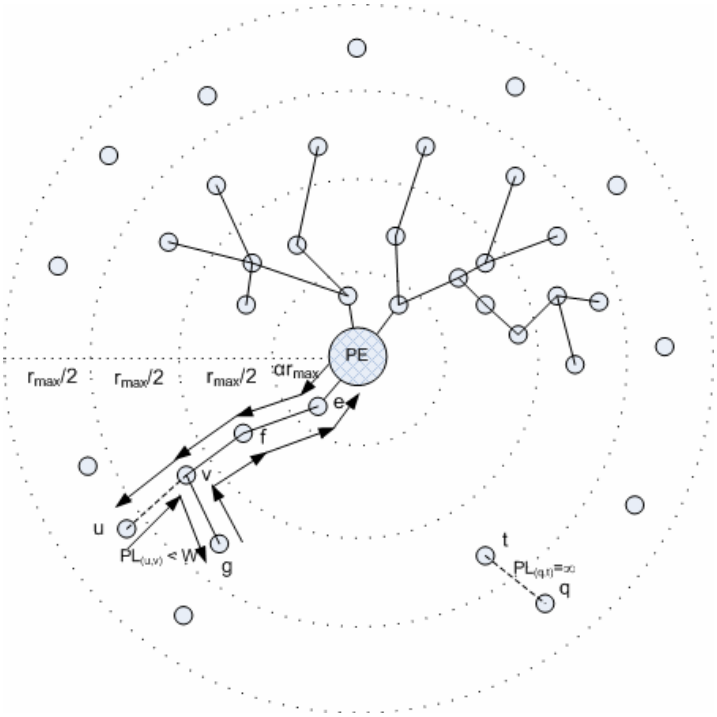


Figure 3. *Partitioning the area around PE into a number of zones*

Assume that we have reached the zone *i* after have visited the first *i-1* zones. In our example, zone *i* is the outer zone. We consider now all the edges (u,v) that connect nodes *u* of the current zone with nodes *v* of either the same zone or the previous zone *i-1*. We also define the Potential Latency $PL_{(u,v)}$ of edge (u,v) as the ensuing latency after connecting the node *u* to the tree of node *v*. If node *v* has not been already connected to a tree, we assume that $PL_{(u,v)}=\infty$. The same value is also assumed when $PL_{(u,v)} > W$, e.g. when the inclusion of the node *u* to the tree of *v* increases the latency of the walk along the tree beyond the bound *W*. In the example of Fig. 3, the *PL* value

of the edge (u,v) is finite and under the bound W. We can also see the walk that will be followed by an agent if the node u is attached to the tree of node v. The PL value of the edge (u,v) is exactly the latency of this walk, that is $c_{PE,e} + d \cdot c_{e,f} + 2d \cdot c_{f,v} + 3d \cdot c_{v,u} + 4d \cdot (c_{u,v} + c_{v,g}) + 5d \cdot (c_{g,v} + c_{v,f} + c_{f,e} + c_{e,PE})$.

Note that in the derivation of this expression we have not considered short-cutting the MA route. Consider also the case of edge (q,t) which has an infinite PL value because node t in zone $i-1$ has not been connected to a tree so far.

Now, we sort the edges according to their PL values and start examining the edges with the order of increasing PL. If (u,v) is the edge currently under examination and provided that $PL_{(u,v)} < \infty$, we connect the node u to the tree of node v, with node v being the parent of u in the tree. This also means that node u after connecting to the tree of v has a path leading to the PE.

Now, all the edges (u,w) of node u with neighboring node w already being connected to a tree are taken out of consideration and we continue with the remaining edges. Notice also that for each edge which connects a node in the zone i with some node on the tree of node u, we should recalculate the PL value since this latency surely changes after the attachment of node u to the tree. We can also easily see that the computational overhead for the PL re-evaluation is small, since the new values can be derived incrementally from the previous PL values. Moreover, note that after the connection of the node to a tree, all the edges pointing to this node may change their PL from infinite to a finite value and so they should be reconsidered again.

After adjusting the PL values of edges and possibly changing the node visiting order accordingly, we continue examining the remaining edges till we find that all PL values are equal to ∞. The start vertices of these edges are nodes of zone i which cannot connect to a tree spanning the first i zones either because all candidate trees have high latency or because their mounting points in zone i or $i-1$ have not been connected to any tree so far.

After zone i, we proceed to zone $i+1$ and we apply the same method as above. We continue till visiting all the zones around the PE. At this point, we have finished the first phase of our technique.

In the second phase, we will try to connect all the nodes that could not be connected to a tree in the first phase. For achieving this, we now allow more possibilities for each node to connect to a tree. Specifically, for each node u belonging to a zone i, we consider all the edges connecting this node with nodes of the same zone as well as nodes of the zone $i-1$ and $i+1$. Similarly to the first phase, we estimate the PL values of these edges but now we globally sort these values in an increasing order

irrespectively of zones they belong to. Then, we visit the edges starting from the edge with the lowest *PL* value and proceeding to higher values. Now, each time a node is connected to a tree, we calculate again the new *PL* values of the affected edges. Also again, an edge is removed from the list of the edges to be visited only when it connects nodes already connected to trees.

Generally, the second phase is executed till we find that all remaining edges have infinite values. This indicates that there are nodes that cannot be connected with the current value of threshold *W*. In order to connect these nodes, we double the value of *W* and we execute again the second phase. If necessary, we continue doubling the *W* value and then executing the second phase till all nodes connect to some tree.

4. Conclusions and Future Work

In this article we presented two efficient heuristic algorithms that derive near-optimal itineraries for MAs performing incremental data fusion in DSN environments. Our algorithms consider spatial distance among sensor nodes for constructing MA itineraries and are shown to outperform alternative existing approaches.

References

Akyildiz F., Su, W., Sankarasubramaniam, Y., Cayirci, E. (2002), *A survey on sensor networks*, IEEE Communications Magazine, August, pp. 102-114.

Esau, L.R., Williams, K.C. (1966), *On teleprocessing system design. Part II - A method for approximating the optimal network*, IBM Systems Journal, vol. 5, pp. 142–147.

Fuggeta, A., Picco, G.P., Vigna, G. (1998), *Understanding Code Mobility*, IEEE Transactions on Software Engineering, vol. 24, no. 5, pp. 346–361.

Kershenbaum, A. (1993), *Telecommunications Network Design Algorithms*, McGraw-Hill.

Milojicic, D. (1999), *Mobile agent applications*, IEEE Concurrency, vol. 7, no. 3.

Qi, H., Iyengar, S.S., Chakrabarty, K. (2001), *Multi-Resolution Data Integration Using Mobile Agents in Distributed Sensor Networks*, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Rev., vol. 31, no. 3, pp. 383-391.

Qi, H., Wang, F. (2001), *Optimal itinerary analysis for mobile agents in ad hoc wireless sensor networks*, in Proc.: International Conference on Wireless Communications.

Wu, Q., Rao, N., Barhen J., Iyengar, S., Vaishnavi, V., Qi, H., Chakrabarty, K. (2004), *On Computing Mobile Agent Routes for Data Fusion in Distributed Sensor Networks*, IEEE Transactions on Knowledge and Data Engineering, vol. 16, no. 6, pp. 740-753.