CLONE-BASED MOBILE AGENT ITINERARY PLANNING USING SEPARATE TREES FOR DATA FUSION IN WSNS

Soheil Javadi[1], Mohammad H. Hajiesmaili[2], Behzad Moshiri[1] and Ahmad Khonsari[2]

[1]School of ECE, College of Engineering, University of Tehran, Tehran, Iran
s.javadi@ut.ac.ir , moshiri@ut.ac.ir
[2]School of Computer Science, IPM, Tehran, Iran
hajiesmaili@ipm.ir , ak@ipm.ir

ABSTRACT

Recent studies demonstrate that Mobile Agent (MA) approach could be more effective than conventional client-server model in Wireless Sensor Network (WSN). Particularly, itinerary planning for MAs is a significant aspect of these approaches. Tree-based methods have been widely used for this purpose where many agents are dispatched simultaneously. Most of tree-based methods try to construct an optimal/suboptimal tree in terms of an objective function. In contrast, in this paper we introduce a new approach that tries to separate the MA dispatching and data fusion operations by considering two itinerary trees which results in more flexibility to regulate the itinerary plan. Based on this idea, we propose an algorithm called Two Trees Clone-based Itinerary (TTCI) which constructs two trees, one serves to distribute MA in the network and another to fuse back the sensed data.
By experimental results, we demonstrate the performance improvement of TTCI algorithm in terms of overall energy consumption in comparison with the previous schemes. Meanwhile, the TTCI keeps the delay low.

KEYWORDS

WSN, Data Fusion, Mobile Agent (MA), clone-based itinerary planning, Energy Consumption, Delay

1. INTRODUCTION

Recent advances in wireless communication has led to rapid proliferation in development of wireless sensor network (WSN) applications [1], [2], including target tracking, battlefield monitoring, and intrusion detection [3], [4].
Each node in WSN has limited resources in terms of processing power, memory, bandwidth, and energy. In addition, a sensor node generates information that needs to be transmitted to a processing element usually called sink.
Regarding these issues, there always exists a trade-off between the accuracy of received information by the sink and amount of dispatched information [5], so we must find a way to effectively utilize precious resources. Towards this, several resource allocation schemes are proposed in the literature considering various aspects of WSNs.
To minimize the energy consumption in WSNs when employing complicated applications that required large amount of information to be transferred, a large body of research activities has been devoted to the in-network processing activities like data compression, data aggregation and data fusion as a state-of-the-art research area. In [6]-[8], several models and techniques for data fusion in the WSNs have been studied.
Data gathering in traditional DSNs usually adherences the client-server approaches, in which nodes' raw data are being sent to the processing element [9]. This approach suffers from some

drawbacks: In some applications, different types of data might be produced while only homogeneous data can be fused together. In these cases, precise routing protocols are required so as homogeneous sensed data meet each other, in a right place and at a right time. Furthermore, gathering and transmission of high amount of data can result in high energy consumption and inefficient bandwidth usage.

As time went by, the concept of *Mobile Agent (MA)* arose. The idea behind the MA approach is that it might be better to send processing codes toward the nodes, instead of sending nodes' raw data to the processing element [10].

This mobile code is usually called *Mobile Agent (MA)*. MAs are software components, means code and/or execution state that can migrate in the system at run-time thus we have some kind of *code mobility*. Moreover, the program itself can make decisions about migration autonomously.

Several good reasons for using MA have been mentioned in [11], [12], such as reducing the network load, overcoming network latency, dynamic adaption. A comparison between the client-server paradigm and MA approach is presented in [13]. One of the most important reasons to use MA approach is the high flexibility in the face of requirements changing, because it is almost impossible or too difficult to reprogram sensor nodes which have already distributed in environment, while in MA approach only the processing codes require to change. On the other hand, there are still some disputes on this paradigm. One of the basic criticisms is that more resources are required to form a node, because it must be capable of executing the MA codes and the underlying framework [14].

One of the most important issues about MA approach is that each MA migrates through the network. So, the question is what is the best itinerary for MA to traverse?

This is the main focus of this work and several other research activities. Sometimes a single MA is used [5], [9], [15] and sometimes several MAs are used simultaneously [14], [16]-[19]. There is another scheme as well in the basis of cloning capability [20]. Long end-to-end delay, high energy consumption, fast energy depletion of some critical nodes, and high complexity are some of the main drawbacks that must be addressed in itinerary planning of MAs in WNS. Herein, our endeavour is devoted to propose a solution that reduces the energy consumption and keeps the delay low, while it has a special attention to reduce the fast energy depletion of the closest nodes to sink.

In this paper, we introduce a new approach towards itinerary of MAs in which two separated trees are used for dispatching MAs and data fusion, in contrast to the previous work that construct an itinerary tree for both.

The tree serves to distribute MA within network is called *Forward Tree*, and the other one employs for data fusion is called *Fusion Tree*.

Using separated trees help us to achieve more flexibility and tuning capability with regard to MA dissemination and data gathering activities.

According to the aforesaid approach, our contribution in this paper is twofold. First, we propose two heuristic algorithms for constructing the *Forward Tree*, the centralized one called *Maximum Degree Heuristic* (MDH) and its distributed version DMDH. In construction of *Forward Tree*, we try to reduce the energy consumption by minimizing the number of MA transmissions. Second, another algorithm is proposed to create the *Fusion Tree*, which is developed to balance the load of closest nodes to the sink by building some sets and appending the unprocessed nodes to the sets with fewer members.

By putting the algorithms for constructing *Forward* and *Fusion Trees* together and in the basis of cloning capabilities of MA, we introduce an algorithm called *Two Trees Clone-based Itinerary* (TTCI).

Experimental results clearly demonstrate the performance improvement of the proposed algorithms in terms of less energy consumption and lower delay in comparison to the similar approaches.

The rest of the paper is organized as follows. Section 2 is devoted to related work. Section 3 will introduce the problem statement and proposed algorithms. Section 4 contains experimental results and finally, we conclude the paper and outline some future directions in Section 5.

2. RELATED WORK

MA-based solutions are a new trend in distributed computing, which provides more flexibility and scalability than conventional client-server models. In [21] mobile agent is represented as a middleware for automatic data fusion in WSNs and several algorithms in the context of itinerary planning have been surveyed. Although, the problem is NP-Complete [5], some previous work tried to use heuristics that yield suboptimal solutions. Note that, we only mention homogeneous WSNs (unlike [22]). Generally, one can imagine two major categories for MA itinerary planning approaches: single MA, and multiple MA approaches.

Single MA approach: In this approach, a single MA tries to traverse among all nodes and collect the data. Qi et al. in [9] proposed two heuristics called LCF and GCF. In LCF, the next node is the closest node to the current node. In GCF, the closest node to the center of environment will be the next node, so it will relocate around the center.

In LCF and GCF, only the physical distance between nodes is considered in construction of MA's itinerary plan, while one can consider many other useful criteria.

Two algorithms named IEMF and IEMA have been introduced in [15], which consider communication cost as well. IEMF operates entirely the same as LCF, but only the first node will be chosen according to the estimation of minimum communication cost. IEMA is iterative version of IEMF, where first k nodes are chosen according to the communication cost and then behaves like IEMF. It will be a trade-off between computational complexity and energy consumption.

In [5], by reducing the MA itinerary problem to a 3D traveling salesman problem, it has been shown that the problem is NP-Complete. Authors use genetic algorithm which results in a suboptimal itinerary plan.

Multiple MA approach: To address the scalability problem in single MA approach, this approach uses more than one MA, simultaneously. In [16], a genetic-based algorithm (GA-IMP) has been introduced that uses many agents simultaneously which leads to better performance than [5]. But, the problem of GA approaches mostly is related to their slow convergence rate, so they are not proper for time critical applications.

Chen et al. in [17], find the center of the dense regions (VCL), one after another. All nodes within a circle around VCL will be covered by a single MA (by methods like LCF, etc). Choosing the proper grouping parameters or dynamic radius, is not easy and highly depends on the topology. Furthermore, using a single agent in a dense area can arise the same deficiencies of single agent itinerary planning. Moreover, considering one hop communication is not a realistic assumption.

There are some approaches based on minimum spanning trees. Inspired by constrained minimum spanning tree, Gavalas et al. in [14] proposed NOID. It finds a near optimal solution where the results are satisfactory, but the computational time is high. Konstantopoulos et al. [18] improved NOID and introduced TBID, which is a greedy suboptimal algorithm. Chen et al. in [19] proposed MST-MIP. They assumed that all nodes are connected together, and weights of the edges are estimated in terms of number of hops. Then, an agent is distributed for each branch of the constructed tree. In a dense area, edges will be light weight so these edges will be selected and in this case, one agent won't be effective. Because of that, by using the hop distance to the sink, BST-MIP tries to make a trade-off between energy consumption and delay. But, if we have low density around the sink, a few agents will be dispatched and the same problem will rise again.

CBID (Mpitziopoulos et al. [20]) uses the cloning capability and tries to build a minimal tree node by node. In order to connect a new node to the nodes of the in progress tree, CBID chooses

the node which leads to minimum expense. Gathering the nodes data by a single MA according to the tree, and in a depth first manner, defines the expense.
The problem is, this tree will results in some unnecessary dispatch of MAs, so increase the energy consumption. Also, it prefers deeper tree that increases the delay.

3. TWO TREES SCHEME

In this section we propose a new approach for itinerary planning of mobile agents in WSNs. As mentioned in Section 2, most of previous studies rely on building a unique tree for both dispatching the MAs and fusing the information. As a main contribution of this paper, we introduce a new approach, where we build two separate trees, one for dispatching the MAs and the second for information fusion. In this section, at first we will describe proposed algorithms to create *Forward* and *Fusion Tree*. Then by putting them together, we introduce TTCI algorithm.

3.1. Forward Tree

Transmission in a wireless channel has a broadcast-form nature, so every node in range will get the sent packet and obviously radio module will consume energy for receiving the packet which may or may not be delivered to the upper layers. Considering this property, our main goal is to distribute MA in the network with minimum number of transmissions which yields the lower energy consumption.
The nodes that are being distributed in a real topology can be divided into some levels with respect to their minimum hop distance from the sink [23]. The nodes that their minimum distance to the sink are i, construct the i th level nodes. The nodes in i th level are connected to some nodes in level i-1, i and i+1 . Moreover, nodes in level i can have common neighbors in level i+1. Some of the level i nodes will send (clone) MA to some level i+1 nodes and try to cover all of them.
Our goal is to cover whole level i+1 nodes with minimum number of transmissions. Towards this, at first, we formally present this problem which we refer to as *covering problem* and then will propose some optimal and suboptimal solutions to it.
Covering problem: Suppose we have a bipartite graph consists of two sets of vertices, V (level i nodes) and U (level i+1 nodes). Each $v_k \in$ V covers (is connected to) a non-empty subset $U_k \subseteq$ U. Our goal is to find a subset of V like M, such that M={ v_α | $\bigcup_{v_\alpha \in V} U_\alpha$ =U } and |M| is minimal. In the other words, there won't exists $M' \subseteq$ V that can cover U and |M|>| M' | .
In what follows, we present some algorithms for the *covering problem* to construct the *Forward tree*.

3.1.1. Naïve Algorithm

One obvious algorithm to find M, is to search through all subsets of V. It means to examine every $v_k \subseteq$ V to find out whether it covers all U or not, and finally choose the minimum one. Although this algorithm yields the optimal solution, but its complexity is exponential in the |V| ($O(2^{|V|})$). It is clear that this is too costly for big valued of |V|, so it is not a practical solution; but it can be used as a benchmark.

3.1.2. MDH Algorithm

In spite of the fact that employing the Naïve algorithm achieves us the optimal solution, it could be impractical due to its high complexity. So, we introduce *Maximum Degree Heuristic* (MDH) algorithm which leads to a near optimal solution with lower complexity.

It works as follows. In each round we choose vertex v_k among V which has the maximum neighbors among unlabeled nodes in U (unselected nodes). Then, we label unselected nodes in U that are neighbor of v_k by its index k. This process continues until all vertices in U have been labeled. In Algorithm 1 (Appendix I), you can see the procedure in more detail.

MDH is a *greedy* algorithm which will not guarantee the optimal (minimal) answers. Two examples of optimal and non-optimal solution of MDH algorithm are demonstrated in Figure 1 and Figure 2, respectively. We will see in Section 4 how effective are the obtained results from MDH, in comparison with Naïve algorithm. Due to its low complexity, we used MDH algorithm in TTCI (see Algorithm 2 in Appendix I).

Discussion on the Complexity of MDH:

At first, we assume that MDH will iterate n times to label all the nodes in U. We set v = |V| and u = |U|. Let us denote by v' and u' as average of nodes' degree in V (at the beginning) and average of nodes' degree in U, respectively. Furthermore, we suppose that each vertex in V (or U), knows its neighbors in U (or V).

The number of iterations n in the best case is $O(1)$, and in the worse case is $O(\mathrm{v})$. In average case, we can imagine that each vertex of V in average has v' neighbours (without any in common with other vertices). So n is equal to $O(\frac{u}{v'})$. The complexity of search and update depends on implementation. If we use a MaxHeap structure, we can find the node with largest degree in $O(1)$ and update the degrees in $O(u'v'\log v)$. Then the whole complexity (based on this consideration) is $O(\frac{u}{v'}v'u'\log v)$=$O(uu'\log v)$=$O(vv'\log v)$.

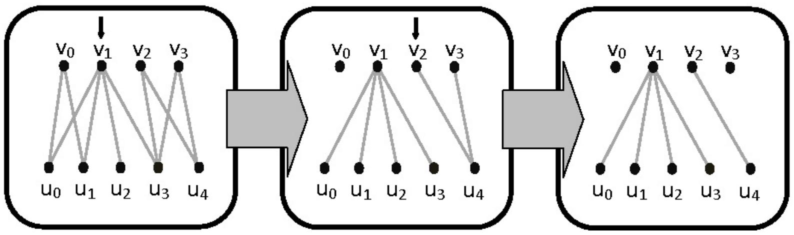

Figure 1. Optimal answer (MDH algorithm example)

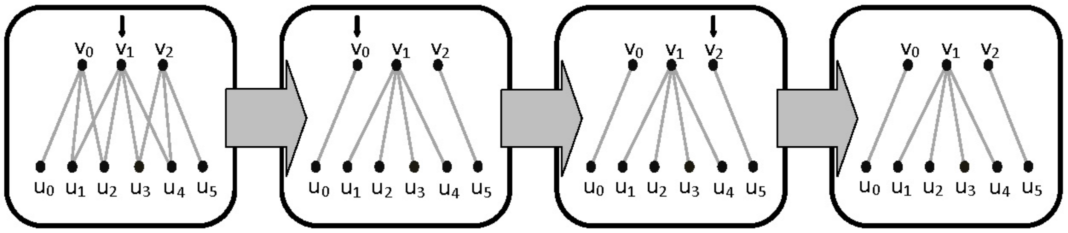

Figure 2. Optimal answer (MDH algorithm example)

3.1.3. DMDH Algorithm

In this subsection we are going to extend MDH algorithm into a distributed one named *Distributed MDH* (DMDH). At first, we assume that every node knows its level. It is not an unreasonable assumption. Consider a packet that has an additional field which is dedicated to keep the level information of the sender. The sink sends a packet which its level indicator field is initialized by zero. If a node receives a packet from a sender which is closer to the sink (in terms of hop distance) compared to the previous senders, it must update its level. Then the node must advertise its new level to its neighbors and so on. We omit further discussions about these issues because those are out of the scope of this paper. In what follows, we are going to explain the main procedure of the algorithm:

1. At the first step, every node in i th level will send a request message to its neighbor nodes.
2. In the second step, only nodes in i+1 th level introduce themselves by sending back a response message. After this phase each level i node can find out its degree (number of level i+1 neighbors).
3. Then each level i node, announces its degree. After this phase, each level i+1 node will recognize its level i neighbors' degree.
4. Finally, every level i+1 node chooses the level i node with higher degree, and announces that it has decided to connect to it. This let nodes of i level find out that to which nodes they must send the MA.

This algorithm is not optimal too but can find answers which are close to optimal. We will compare it with the optimal Naïve algorithm, in Section 4.

3.2. Fusion Tree

Construction of *Fusion Tree* can be optimized for different purposes, such as energy consumption, delay, or combination of them. There are many studies in the literature trying to build an optimal fusion (or aggregation) tree based on various criteria (even non MA-based approaches such as [24]). Note that in this work, we suppose that the construction of *Fusion Tree* is centralized and sink knows the coordination of all nodes.

Fast energy depletion of sink neighbors, especially the nodes in 1*st* level, has a significant negative impact on the network's lifetime. Thus, in our approach, our main aim is to balance the load of first level nodes, because they are the communication bridges to the sink. Fast energy depletion of some of them can lead to loss of many nodes data. One trivial solution towards this, could be to connect a new node at lower levels, to a higher level node with lower degree. But, this solution would be ineffective in some situations, because data received by a node are gathered from its entire subtree. So, a node with low degree may have large number of descendants.

Here, we are going to propose a new algorithm for building the *Fusion Tree*. Towards this, we consider each of the first level sink neighbors as a new group, i.e., the number of groups is equal to the number of direct sink neighbors. Then, at the first phase, starting from the second level, we assign each node of level i, to one of its level i-1 neighbors which its corresponding group has fewer number of members, till, all nodes are allocated to a particular group. Now, as the second phase, we try to reallocate the nodes' group to improve the overall load balancing. For this purpose, while a better allocation is found, we keep searching. Again, starting from level 2, if appending a node of level i to a node possessed in level i-1 in its neighborhood (which belongs to deferent group) leads to a lower standard deviation of groups sizes, we will rearrange the tree. Algorithm 3 (in Appendix I) shows this procedure. Obviously we can recompute the algorithm for each one of sink neighbors as a new tree root. This can balance the other nodes load also. Figure 3 is an example of the *Fusion Tree* for a sample WSN's topology, where the nodes belong to a group are demonstrated in the same color.

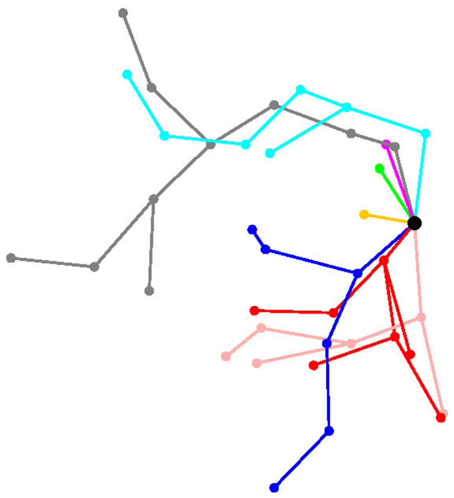
Figure 3. Optimal answer (MDH algorithm example)

3.3. TTCI Algorithm

So far, we proposed some algorithms for building *Forward* and *Fusion Trees*. In this subsection, we are going to introduce the *Two Trees Clone based Itinerary* (TTCI) algorithm by putting them together. This approach can provide us more flexibility and possibility of adjustment between different design criteria. Generally speaking, the structure used in the most tree based itinerary planning approaches is as follows:

1. Building *Fusion Tree*
2. Distributing MAs according to the constructed tree
3. Fusing the information based on the constructed tree

On the other hand, our approach contains one more phase:

1. Building *Forward Tree*
2. Building *Fusion Tree*
3. Distribution of MA according to *Forward Tree*
4. Fusing the information back to the sink, according to *Fusion Tree*

The algorithm works as follows: All nodes within the sink transmission range will constitute the first level nodes. For the sake of simplicity, we assume the sink and ordinary nodes have the same transmission range.

Then, unvisited nodes that are directly reachable from the level i nodes, will constitute the level i+1 nodes. This will continue until we visit all nodes and construct all level sets. Fortunately, this can be done concurrently with building the trees, so, there is no need for extra processing. We used MDH algorithm to create *Forward Tree* level by level (Algorithm 2 in Appendix I). Also, we have introduced Algorithm 3 (in Appendix I) for building *Fusion Tree*. After the phases of tree construction, MA can start its itinerary. Each MA cloner node, clones itself (the MA), and waits for arrival of data produced by its children through *Fusion Tree*. These cloned MAs will be annihilated after delivering data to their parent.

Obviously MA needs to carry the information about its itinerary. Simply, each node has to know, to whom it should send and from who it should receive. We use two trees instead of one, so, we must send extra information compare to single tree approach. Employing the hierarchical scheme, allows us to prune the itinerary information effectively. When a level i cloner node receives the MA, it can prune the itinerary information corresponding to nodes with level≤i-1 . Thus, in an ordinary tree based approach, every node has to send the itinerary information of size N (number of the sensor nodes). But in our approach, each node (v_i) has to send the itinerary information of size $\sum_{j=v_i \cdot level}^{j \le \max Depth} N_{l_j}$ in which N_{l_j} is the number of j th level sensor nodes.

In WSNs, communication is not completely reliable. Also, nodes may face some problems which will not let SN to operate properly. It means that when the delay of receiving data from a certain node exceeds a particular threshold, we should proceed without that. Here, we do not mention the issues about this timer settings and robustness. We suppose that we are not facing the failure and mobility of the nodes. Many researchers consider limited amount of energy for nodes, to study the network's lifetime. Also, some works consider sensor malfunction which leads to elimination of sensors, to study the robustness of their algorithms in deal of sensor death. But, these are not our concerns in this study. We didn't consider the collision condition as well.

4. EXPERIMENTAL RESULTS

In the following section, we will present experimental results to show the effectiveness of the algorithms proposed in this paper. All implementation codes were written in Java. Moreover, results have been reported by 95% confidence level. At first, we will show the experimental results about algorithms introduced to build *Forward* and *Fusion Tree*. Then we benchmark TTCI with another clone-based algorithm called CBID [20].

Energy model used in the simulation is based on the fact that the energy consumed by radio module has 3 states: Tx, Rx, and sleep [25]. Also, there are another models in which energy consumption is proportional to the number of bits [24], [26], so we reported the number of sent/received bytes. Some have used a simple linear model (in terms of the number of bits) for processing and receiving [26]. Also, sending energy can be considered as it is proportional to the square of the distance. It means $E_S \propto d^2$ [24].

Table 1. Simulation parameters

Parameter	Value
Area	500×500 (m^2)
Transmission range	45 (m)
Transmission rate	250 (kbps)
MA code size	1 (KByte)
Local data	2 (KByte)
Fusion factor (output/input)	0.7
Tx energy consumption	57.42 (mW)
Rx energy consumption	62 (mW)
Periodic energy consumption	6 (mW)

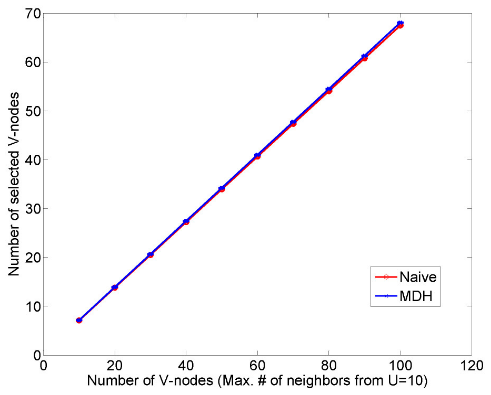

Figure 4(a)

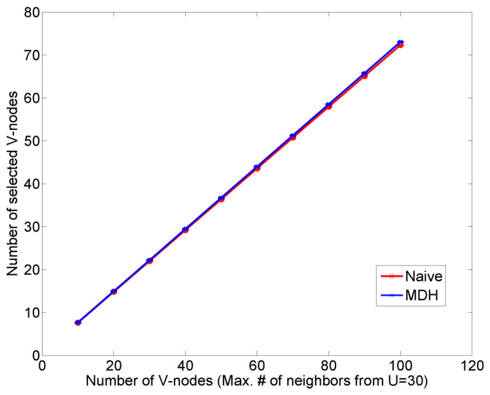

Figure 4(b)

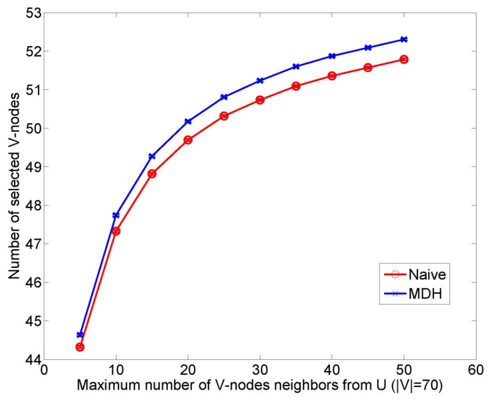

Figure 4(c)

Figure 4. Naïve and MDH comparison

4.1. Comparison of Naïve, MDH, and DMDH

To compare Naïve and MDH, we examined the results over different topologies. Figures 4(a) and 4(b) show the effect of increasing the number of level i nodes, in the situation of fix average number of neighbors in level i+1. On the other hand, in Figure 4(c) the number of level i nodes has been fixed, while average number of neighbors (in level i+1) is variable. Results from Figure 4(a), Figure 4(b), and Figure 4(c) clearly demonstrate that results of two algorithms are similar and very close. Therefore, due to lower complexity, employing MDH instead of optimal Naïve can make sense. Thus, in TTCI, we used MDH to build *Forward Tree*.

You can see the phrase "*Maximum number of neighbours*" in some figures, which needs further explanation. We choose a number Max≤|U| to build a random topology which means that each node in V will have at most Max neighbours in U. Then a random number $Rand_{v_k}$ will be chosen for node v_k in V such that $Rand_{v_k} \leq Max$ is equal to the number of v_k 's neighbours in U. For example, in Figure 4(c), Max=50 shows that each node in level i has 25 neighbours in average (because the distribution is uniform). It is obvious that by increasing the Max, the topology becomes denser. Also, we compare the results of Naïve and DMDH in Figure 5. We can see that the results are close and can lead to proper selection of V-nodes for the *covering problem*.

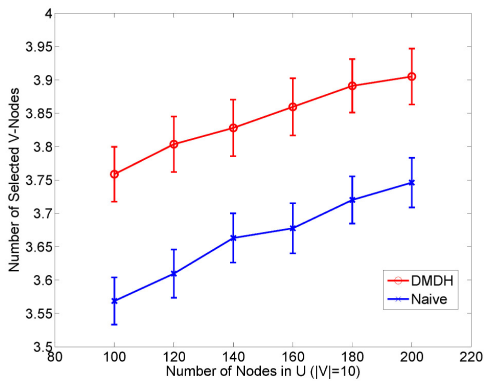

Figure 5(a). Effect of increasing the U-nodes

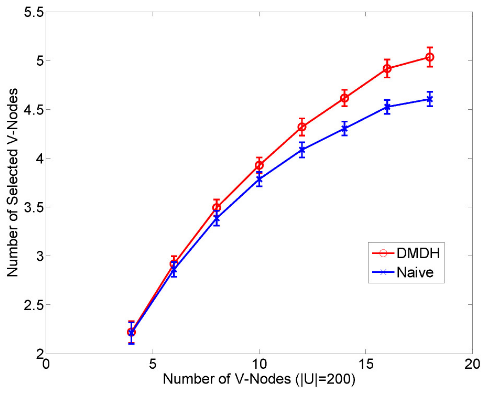

Figure 5(b). Effect of increasing the V-nodes

Figure 5. Naïve and DMDH comparison

4.2. On the Performance of Fusion Tree Algorithm

Figure 6(a) shows the effectiveness of our *Fusion Tree* algorithm on balancing the number of nodes attached to each group. You can see that even after first phase in our algorithm, we have a good balance compared to the tree constructed by CBID. Analyzing the improvement phase of our algorithm is not straightforward because of the randomness in the topology. Thus, for the sake of better illustration of the complexity, we report the mean number of changes in Figure 6(b), which means the maximum number of the *while* loop (line 17 of Algorithm 3 in Appendix I) iteration.

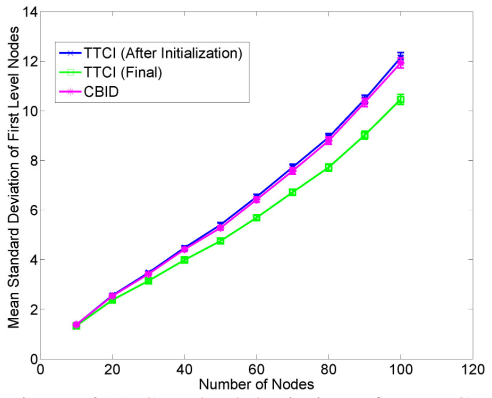

Figure 6(a). Standard deviation of groupCount

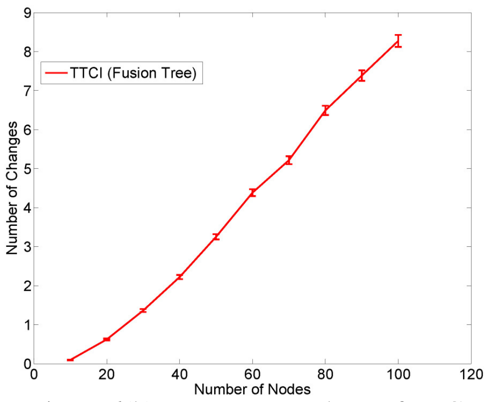

Figure 6(b). Improvement loop of TTCI: Number of changes

Figure 6. Performance of fusion tree algorithm

4.3. Simulation and Implementations outputs

Now we present the result of simulation that compares TTCI with CBID [20]. We used Castalia 3.1 framework [25] (built upon OMNeT++ simulator [27]). Table 1 summarizes the simulation conditions. Also, we mentioned the overhead of itinerary plan carried by MA in both algorithms. Delay reported for 3 rounds of data gathering.

As it was expected, TTCI has less delay, because CBID may has deeper subtrees (Figure 7(a)). Figure 7(b) shows the average of energy consumption. It shows the superiority of proposed algorithm. Some packets fail because the radio module of receiver is not in Rx mode. Figure 8(a) shows the aforesaid packet breakdown. The difference between received packet in TTCI and CBID shows that, CBID tends to send more packets compare to TTCI. Finally, Figure 8(b) shows the average number of bytes sent/received in each algorithm. TTCI has superiority because it uses the minimum number of cloners. Moreover, CBID cannot prune its itinerary data, so it has to send more packets.

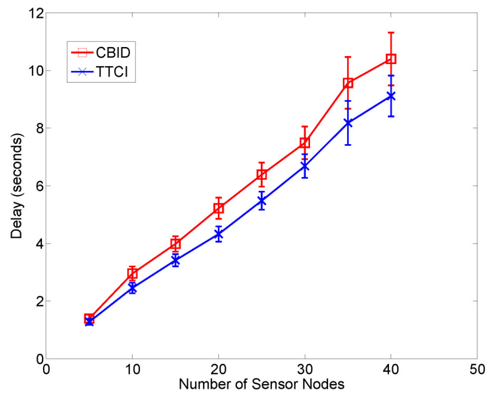

Figure 7(a). Delay comparison

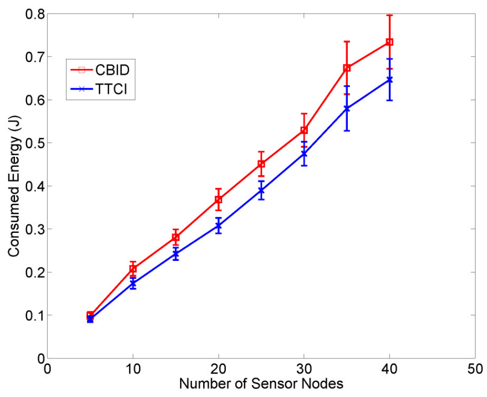

Figure 7(b). Mean energy consumption comparison

Figure 7. Comparison of TTCI and CBID

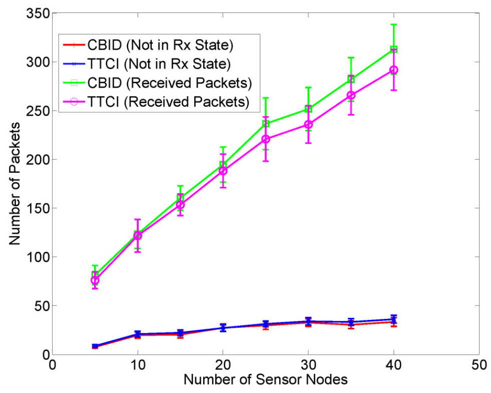

Figure 8(a). Received packets

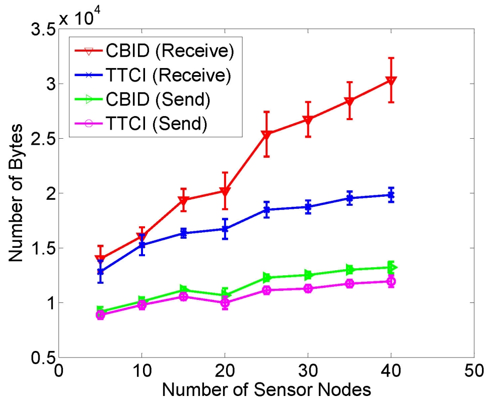

Figure 8(b). Packet exchange comparison

Figure 8. Comparison of TTCI and CBID

5. Conclusion

Due to importance of in-network data processing such as data fusion and data aggregation, several studies have focused on its efficiency. MA approaches can reduce energy consumption. They also increase flexibility in the face of requirements changing.

Lots of researches have been conducted to find the optimal itinerary plan for MA. In this paper, we introduced a tree-based approach according to the cloning capability of MAs, which uses separate trees for distributing the MAs and performing data fusion. Based on this scheme, a new algorithm named TTCI is presented. Different algorithms were studied in order to build *Forward* and *Fusion Trees*. Simulation results revealed the effectiveness of our approach in reduction of energy consumption and keeping the delay low. There are several future directions to extend this work.

One future work will be studying this method in environment along with collision conditions. Robustness is another aspect that could be addressed in the future work. Finally, different algorithms can be used to build the trees for further improvement based on different criteria.

REFERENCES

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “A Survey on Sensor Networks,” *IEEE Communications Magazine*, vol. 40, pp. 102-114, 2002.

[2] N. Xu, “A survey of Sensor Network Applications,” *IEEE Communications Magazine*, vol.40, pp. 102-114, 2002.

[3] P. Santi, *Topology Control in Wireless Ad Hoc and Sensor Networks*, John Wiley & Sons Ltd, 2005.

[4] J. Yick, B. Mukherjee, and D. Ghosal, “Wireless Sensor Network Survey,” *Computer Networks*, vol. 52, pp. 2292-2330, 2008.

[5] Q. Wu, N.S.V. Rao, J. Barhen, S.S. Iyengar, V.K. Vaishnavi, H. Qi, and K. Chakrabarty, “On Computing Mobile Agent Routes for Data Fusion in Distributed Sensor Networks,” *IEEE Trans. Knowledge and Data Engineering*, vol. 16, no. 6, pp. 740-753, Jun. 2004.

[6] E.F. Nakamura, A.A.F. Loureiro, and A.C. Frery, “Information Fusion for Wireless Sensor Networks: Methods, Models, and Classifications,” *ACM Computing Surveys (CSUR)*, vol. 39, no. 3, 2007.

[7] S.K. Das, *High-Level Data Fusion*, Artech House Publishers, 2008.

[8] H.B. Mitchell, *Multi-Sensor Data Fusion: An Introduction*, Springer, 2007.

[9] H. Qi, and F. Wang, “Optimal itinerary analysis for mobile agents in ad hoc wireless sensor networks,” *Proc. Int'l Conf. on Wireless Communications*, pp.147-153, 2001.

[10] H. Qi, S.S. Iyengar, and K. Chakrabarty, “Multi-Resolution Data Integration Using Mobile Agents in Distributed Sensor Networks,” *IEEE Trans. Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 31, no. 3, pp. 383-391, 2001.

[11] D.B. Lange, and M. Oshima, “Seven Good Reasons for Mobile Agents,” *Communications of the ACM*, vol. 42, no. 3, pp. 88-89, 1999.

[12] C.G. Harrison, D.M. Chess, and A. Kershenbaum, “Mobile Agents: Are they a good idea?,” *Mobile Object Systems Towards the Programmable Internet*, vol. 1222, pp. 25-45, 1997.

[13] Y. Xu, H. Qi, and P.T. Kuruganti, “Distributed Computing Paradigms for Collaborative Processing in Sensor Networks,” *IEEE Global Telecommunications Conference, GLOBECOM '03*, vol. 6, pp. 3531 - 3535, Dec. 2003.

[14] D. Gavalas, A. Mpitziopoulos, G. Pantziou, and C. Konstantopoulos, “An approach for near-optimal distributed data fusion in wireless sensor networks,” *Wireless Networks*, vol. 16, no. 5, pp. 1407-1425, 2010.

[15] M. Chen, V. Leung, S. Mao, T. Kwon, and M. Li, “Energy-Efficient Itinerary Planning for Mobile Agents in Wireless Sensor Networks,” *IEEE Int'l Conf. Communications, 2009. ICC'09.*, 2009.

[16] W. Cai, M. Chen, T. Hara, L. Shu, and T. Kwon, “A Genetic Algorithm Approach to Multi-Agent Itinerary Planning in Wireless Sensor Networks,” *Mobile Networks and Applications*, pp. 1-12, DOI:10.1007/s11036-010-0269-z, 2010.

[17] M. Chen, S. Gonzlez, Y. Zhang, and V.C.M. Leung, “Multi-Agent Itinerary Planning for Sensor Networks,” *Proceedings of the IEEE 2009 International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine 2009), Las Palmas de Gran Canaria, Spain*, 2009.

[18] C. Konstantopoulos, A. Mpitziopoulos, D. Gavalas, G. Pantziou, "Effective Determination of Mobile Agent Itineraries for Data Aggregation on Sensor Networks," *IEEE Trans. on Knowledge and Data Engineering*, vol. 22, no. 12, pp. 1041-4347, Dec. 2009.

[19] M. Chen, W. Cai, S. Gonzalez, and V.C.M. Leung, "Balanced Itinerary Planning for Multiple Mobile Agents in Wireless Sensor Networks," *Ad Hoc Networks*, vol. 49, no. 7, pp. 416-428, 2010.

[20] A. Mpitziopoulos, D. Gavalas, C. Konstantopoulos, and G. Pantziou, "CBID: A Scalable Method for Distributed Data Aggregation in WSNs," *International Journal of Distributed Sensor Networks*, DOI:10.1155/2010/206517, Hindawi Publishing Corporation, 2010.

[21] A. Mpitziopoulos, D. Gavalas, C. Konstantopoulos, and G. Pantziou, "Mobile Agent Middleware for Autonomic Data Fusion in Wireless Sensor Networks," In M.K. Denko, L.T. Yang, and Y. Zhang, *Autonomic Computing and Networking, chapter 3 (pp. 57-81)*, USA:Springer, 2009.

[22] G. Wu, H. Li, and L. Yao, "A Group-based Mobile Agent Routing protocol for multitype Wireless Sensor Networks," *IEEE/ACM International Conference on Green Computing and Communications & IEEE/ACM International Conference on Cyber, Physical and Social*, pp. 42-49, 2010.

[23] S. Kulkarni, A. Iyer, and C. Rosenberg, "An Address-Light, Integrated MAC and Routing Protocol for Wireless Sensor Networks," *IEEE/ACM Transactions on Networking*, vol. 14, pp. 793-806, 2006.

[24] H.O. Tan, I. Korpeoglu, and I. Stojmenovic, "Computing Localized Power-Efficient Data Aggregation Trees for Sensor Networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 22, no. 3, pp. 489-500, Mar. 2011.

[25] A. Boulis. (2010). Castalia A Simulator for Wireless Sensor Networks and Body Area Networks. version 3.1 User's Manual, NICTA. [online]. Available: castalia.npc.nicta.com.au/pdfs/Castalia\%20-\%20User\%20Manual.pdf.

[26] S. Eswaran, M. Johnson, A. Misra, and T. La Porta, "Adaptive In-Network Processing for Bandwidth and Energy Constrained Mission-Oriented Multi-hop Wireless Networks," *In Proceedings of the 5th IEEE International Conference on Distributed Computing in Sensor Systems*, pp. 87-102, DOI:10.1007/978-3-642-02085-8_7, 2009.

[27] A. Varga. (2004). OMNeT++ Discrete Event Simulation System. version 3.0 User Manual. [online]. Available: www.omnetpp.org/doc/omnetpp/Manual.pdf.

APPENDIX I

The Appendix I contains the pseudo code of the presented algorithms. These may further clarify the algorithms' purpose.

Algorithm1: MDH Algorithm

```
Definitions
1: V and U are two array of nodes (in the bipartite graph) which we want to cover whole U by some of the
   nodes in V
2: int[] deg keeps unprocessed degree of V-nodes (deg[k]=unprocessed degree of $v_k$ which is initialized
   by its degree)
3: index is an integer for tracing the next selected node in V
4: M is the resulting suboptimal set
5: findMax(deg) returns the index of element with the maximum value in deg
6: N($v_k$) returns all neighbors of $v_k$
7: $u_k \cdot parent$ is the node in V which is selected to cover $u_k$
Main
```

```
1: while $\exists u_k \in U$ such that $u_k \cdot parent$ is null do
2:     index ← findMax(deg)
3:     M · add($u_{index}$)
4:     for each $u_{k'} \in$ N($v_{index}$) such that $u_{k'} \cdot parent$ is null do
5:         $u_{k'} \cdot parent \leftarrow v_{index}$
6:         for each $v_{k''} \in$ V such that $u_{k'} \in$ N($v_{k''}$) do
7:             deg[$k''$]--
8:         end for
9:     end for
10: end while
```

Algorithm 1. MDH Algorithm for the Covering Problem

Algorithm 2: Forward Tree Algorithm

Definitions

1: V is the array of all nodes of size n+1 (do not getting confused by V and U in Alg. 1)

2: int[] **deg** for unprocessed degree of each node initialized by 0 (deg[k]=unprocessed degree of v_k , k=1, $\cdots$,n)

3: v_0 is the sink

4: **processed**, **candidates** and **children** are set data structures (also, they can be implemented by ordinary arrays)

5: v_{index} is used to point to a selected node (temporary variable)

6: N(v_k) returns the neighbors of v_k

7: parent of a node, is the node which is selected to cover it

8: findMax(deg,candidates) returns the node with maximum degree in candidates set

9: $v_k \cdot$ addChild() adds a node to the children list of v_k

Main

```
1:  processed ← $v_0$
2:  candidates ← N( $v_0$ )
3:  for each $v_j \in$ candidates do
4:      $v_j \cdot parent \leftarrow v_0$
5:      $v_0 \cdot$ addChild( $v_j$ )
6:  end for
7:  while processed · size()<(n+1) do
8:      for each $v_{j'} \in$ candidates do
9:          for each $v_{j''} \in$ N( $v_{j'}$ ) do
10:             if $v_{j''} \notin$ (candidates $\bigcup$ processed) then
11:                 children ← children $\bigcup$ { $v_{j''}$ }
12:                 deg[ $j'$ ]++
13:             end if
```

```
14:             end for
15:         end for
16:         if children · size()>0 then
17:             while $\exists v_{k'} \in$ children such that $v_{k'} \cdot parent$ is null do
18:                 $v_{index} \leftarrow$ findMax(deg,candidates)
19:                 for each $v_{k''} \in$ N($v_{index}$) such that (($v_{k''} \in$ children) &&
                                                ($v_{k''} \cdot parent$ is null )) do
20:                     $v_{k''} \cdot parent \leftarrow v_{index}$
21:                     $v_{index} \cdot$ addChild($v_{k''}$)
22:                     for each $v_{\alpha} \in$ N($v_{k''}$) such that $v_{\alpha} \in$ candidates do
23:                         deg[$\alpha$]--
24:                     end for
25:                 end for
26:             end while
27:             processed $\leftarrow$ processed $\bigcup$ candidates
28:             candidates $\leftarrow$ children
29:             children · clear()
30:         end if
31: end while
```

Algorithm 2. Forward Tree Algorithm used in TTCI (based on MDH)

Algorithm3: Fusion Tree Algorithm

Definitions

1: **maxDepth** shows the maximum depth of the *Forward Tree*

2: V is the set of all nodes as in Alg. 2

3: sink (v_0) level is 0 and we assume that by execution of Alg. 2, each node level is determined

4: **FuParent** of a node (v_k), is the node which is selected to receive its (v_k) data for data fusion (and the same for addFuChild())

5: v_k and $v_{k'}$ are temporary variables

6: **groupCount** is an array of integers of size |N(v_0)|

7: group is the groupID assigned to a node, initialized by -1

8: N(v_k) returns all neighbors of v_k

9: FuRemove($v_{k''}$) removes $v_{k''}$ from the fusion children list, of a node

Main

```
1:  for each node like $v_k \in$ N($v_0$) do
2:      assign a new groupID (from [0 .. |N($v_0$)|-1]) to $v_k \cdot group$
3:      $v_k \cdot FuParent \leftarrow v_0$
4:      $v_0 \cdot$ addFuChild($v_k$)
5:      groupCount[$v_k \cdot group$]++
6:  end for
7:  for i from 2 to maxDepth do
```

```
8:      for each node like $v_k$ which $v_k \cdot$ level==i do
9:          find the best $v_{k'}$ which ( $v_{k'} \in$ N( $v_k$ ) ) && ( $v_{k'} \cdot$ level==i-1) &&
                                        (groupCount[ $v_{k'} \cdot group$ ] is minimum)
10:         $v_k \cdot FuParent \leftarrow v_{k'}$
11:         $v_{k'} \cdot$ addFuChild( $v_k$ )
12:         $v_k \cdot group \leftarrow v_{k'} \cdot group$
13:         groupCount[ $v_{k'} \cdot group$ ]++
14:     end for
15: end for
16: boolean ctnu $\leftarrow$ true
17: while ctnu==true do
18:     ctnu $\leftarrow$ false
19:     for i from 2 to maxDepth do
20:         for each node like $v_k$ which $v_k \cdot$ level==i do
21:             find the best $v_{k'}$ which ( $v_{k'} \in$ N( $v_k$ )) && ( $v_{k'}$ level==i-1) &&
        (appending $v_k$ to $v_{k'}$ leads to minimum standard deviation of groupCount)
22:             if $v_{k'} \neq v_k \cdot FuParent$ do
23:                 $v_k \cdot FuParent \cdot$ FuRemove( $v_k$ )
24:                 $v_k \cdot FuParent \leftarrow v_{k'}$
25:                 $v_{k'} \cdot$ addFuChild( $v_k$ )
26:                 ctnu $\leftarrow$ true
27:                 decrease groupCount[ $v_k \cdot group$ ] by the number of $v_k$
                                                                  descendants +1
28:                 increase groupCount[ $v_{k'} \cdot group$ ] by the number of $v_k$
                                                                  descendants +1
29:                 update $v_k \cdot group$ and its descendants with $v_{k'} \cdot group$
30:             end if
31:         end for
32:     end for
33: end while
```

Algorithm 3. Fusion Tree Creation Algorithm used in TTCI