Dependable Data Aggregation on Cluster-based Wireless Sensor Networks

YUE-SHAN CHANG[1], JIUN-HUA HUANG[2], TONG-YING JUANG[1]
[1]Department of Computer Science and Information Engineering
National Taipei University
151, University Road, Sanhsia, Taipei County,
237 TAIWAN, R.O.C.
http://web.ntpu.edu.tw/~ysc/
[2]Institute of Communication Engineering
National Taipei University
151, University Road, Sanhsia, Taipei County,
237 TAIWAN, R.O.C.

Abstract: In the paper we propose a dependable and efficient data aggregation scheme based on fault map that is constructed by estimated fault probability using Bayesian Belief Network (BBN). Cluster head will forward aggregated event depend on the total dependability of collected event in which we assign each detecting node a dependence weight. The dependence weight of sensor node is mapped from its fault probability estimated by cluster head. We propose a mapping function to map node's fault probability into a dependence weight. Cluster head accumulates the dependence weight instead of number of source node and transmits the aggregated event when the threshold has reached. The simulation result shows that the approach even though take some extra delay time, it will increase the credibility and dependability of the aggregated event in the naturally unreliable wireless sensor network.

Keywords: Dependable Data Aggregation, cluster based, Wireless Sensor Networks, Fault Estimation, Fault map

1 Introduction

Data aggregation in cluster based WSN[1] is that cluster head merges sensed events from multiple sensors into a single message before forwarding to next cluster head or sink in order to greatly reduce the extra energy consumption [4, 5]. These schemes only took energy efficiency into account without considering possibly fault occurrence or unusual event reading of sensor node. In traditional data aggregation, aggregation quality defines how many number of source nodes that the cluster head need to transmit the merged message in a cluster.

In this paper we propose a data aggregation approach for effectively and efficiently aggregating event from sensor nodes. We use the fault probability information of sensor node to adjust the credibility of an event by varying some parameters. The fault probability [2] is estimated by using BBN model to verify the target event and to analyze the possible fault probabilities of nodes cumulatively. While the accumulated weight reaches a specific value, the cluster head would transmit the event to sink or gateway cluster head without the completely responses time in the cluster head.

Since the cluster head has the fault probability information of sensor nodes in the cluster, we expect to find a mapping function to map the sensor's fault probability into dependence weight. The higher fault probability maps lower weight, and the lower fault probability maps higher weight. Afterward, cluster head would accumulate the weight instead of number of source node and transmit the aggregated event till a threshold has reached. Thus, we can ensure that the transmitted event would have certain credibility.

The threshold represents the corresponsive credibility of event. When the threshold that calculated by cluster head is high, represents this aggregated event needs more number of source node or the source node with higher weight (lower fault probability) to reach the threshold. In other words, before the event in a cluster with higher threshold transmitted to sink by cluster head, it needs more weight evidence that mapped from fault probability. Thus, through the raise of threshold, the credibility of the aggregated event has been corresponsive improved.

2 Related Works

Data aggregation with cluster architecture or hierarchical schemes, such as Low Energy Adaptive Clustering Hierarchy (LEACH) Protocol [3] and [7], that combines responses from multiple sensors into single message to transmit to the sink or gateway cluster head. The LEACH randomly selects a few

sensor nodes as cluster heads, and the cluster heads response to collect the sensed data from the neighboring nodes in their own cluster. The research in the [7] enhances the Direct Diffusion scheme by a hierarchical aggregation technique to save the transmission energy.

The Greedy Aggregation [4] was an approach that adjusts aggregation points to increase the amount of path sharing to reduce energy consumption in density sensor networks. The Greedy Aggregation constructs a greedy incremental tree, that a shortest path is established for only the first source to the sink whereas each of the other sources is incrementally connected at the closest point on the existing tree.

In [6], the authors construct a game-theoretic model for reliable path-length and energy-constrained routing with data aggregation in wireless sensor networks. They define two benefit functions and show that computing optimal length-constrained paths is NP-Hard under both models for arbitrary sensor networks. They also develop sensor-centric metrics called path weakness to measure the qualitative performance of different routing schemes. They assume that the node have failure probability to evaluate the reliability of routing path from source node to the sink. They focus on the payoffs of the routing path in data aggregation.

Shin et. al. [8] proposed a reliable data aggregation scheme named RDAP, that performs the routing and query inserting process at the same time in order to accomplish the minimum power consumption and packet loss in sensor networks. RDAP eliminates the unnecessary routing process which is generated by the periodic routing and immediately adapts to network changes between routing intervals.

3 Fault Map based Data Aggregation

3.1 Assumption

A few of assumptions must be made as follows:

(a) We construct the model in immobile wireless sensor network.
(b) Sink has the global information of fault probability.
(c) The cluster head knows the information of other cluster heads via the periodic information exchange and knows the direction of the sink via the first initial message.
(d) In the cluster, the normal sensor node knows the location of the cluster head and would not receive packet from other cluster heads.
(e) Cluster head can vary transmission direction via the triangle antenna.

In cluster, the sensor node senses data when event happened, after that, the sensing node would transmit the sensed data to the cluster head. Cluster head would have some behaviors, as follows: We would depict the algorithm amply in follows.

(a) Calculates the fault probability via these data, such as temperature or noise reading.
(b) Records the fault probability in the fault rate table.
(c) Waits for some time to accumulate the accuracy of the event by the different source nodes with the same event.
(d) Transmits the event to next cluster head.

3.2 Mapping function

Cluster head already has the fault probability of each node in the cluster. Cluster head can judge the correctness of the event from the fault probabilities of source node and bypass nodes. However the greater volume of node's fault probability represents the lower correctness of the node, directly accumulate the value of the node's fault probability cannot be done with the fault map based data aggregation. The function *f* has looking for:

$$W = f(P)$$

where the W is the weight, the P is the fault probability of node, and the function f is expected. The mapping function has been separated in two parts. First, node's fault probability is mapped into accuracy. Then, the node's accuracy is mapped into weight.

3.2.1 Linear mapping function

The linear mapping function is formulated as follows.

Definition 1**:** *C_j : the accuracy of node j*

$$C_j = -((L \times P_j) - \frac{L}{2}) \tag{1}$$

where L : the total range that the fault probability going to map
P_j : the fault probability of node j

The linear mapping maps fault probability into accuracy of node, that the higher fault probability, the lower accuracy. The mapped accuracy is symmetric to zero point and linearly expends the probability into greater value, which means if the accuracy value is negative, the fault probability is bigger than 0.5, if the accuracy value is positive, the fault probability is smaller than 0.5.

3.2.2 Weighted mapping function

The linear mapping does not fit the human's experience in probability. For example, to a human, it does not make differences if an event with fault probability 0.9 or 0.95. Human just feels the event could have much possibility to be wrong. On the other hand, human feels the event might be right no matter the fault probability of the event is 0.1, 0.15 or 0.2. The node's accuracy, that after the linear mapping function, can show if the node's fault probability larger than 0.5 via the negative or positive sign, but cannot preserve the human's experiences in probability. Consequently, mapping the accuracy of a

node C_j into the weight using sigmoid function is necessary, as follows:

Definition 2*: W_{ij} *is the weight of node j that calculates by cluster head i. This value is between 0 and 1.

$$W_{ij} = \frac{1}{1+\exp(-1\times a'\times C_j)} \quad (2)$$

where a' is a factor to vary the decreasing or increasing rate of sigmoid function
C_j is accuracy, which is linear mapping from fault probability

3.3 Node model

In order to easily analyze the data aggregation algorithms, separating the behaviors of the node in the summation part and activation part and discuss them in cluster head and sensor node aspects are helpful.

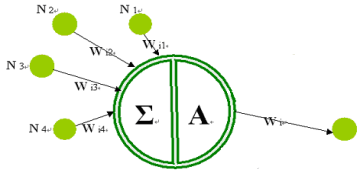


Fig. 2 node model

3.3.1 Node is cluster head

The *Ni* in the Fig. 2 represents the reading value from node *i*. Cluster head receives the reading value, and calculates the weight. W_{ij} is the weight of node j. W_i is the cluster head's weight.

In the summation part of the cluster head, it collects the data from the neighboring sensor node in the cluster, calculates the weight of the event via the received data, and goes to activation part if the threshold of weight has been reached. In case, if the cluster head receive the event from *Ni* in the Fig. 2, the accumulated weight as follow:

$$\sum_{j=1}^{4} W_{1j} = W_{11} + W_{12} + W_{13} + W_{14} \quad (3)$$

3.3.2 Node is sensor node

The *Ni* represent the reading value from node *i*. They might be the temperature or noise reading. In the case, the sensor node only receives reading value from other neighboring sensor node in the same cluster, which means the sensor node is the bypass node of other source node. Due to the bypass node might have high fault probability, the bypass node do not calculate the weight. Both source node and bypass node could affect the correctness of the event or reading value, the bypass node puts the sensor ID itself into the set of sensor ID in the original packet. Sensor node leaves the weight-computing task to the cluster head.

In the summation part of other sensor node, if the node is the source node of an event, goes to activation part directly. What if the node is the bypass node of the event, it puts the ID itself on the sensor ID in the packet and goes to the activation part.

The activation part of the sensor node is exact the sending task but with the next sensor node election. The next hop node election mechanism does not discuss in this research.

3.4 Fault map based data aggregation algorithm

3.4.1 Summation part

The algorithms are shown in Table I:

TABLE I: ALGORITHM OF SUMMATION PART

```
Summation(SensorID,EventID){ // act when a node receive or sense an
event
while (node itself is a cluster-head)
do{
  if (event is not from cluster-heads) // aggregation in cluster
  {    if (EventID is first received )
       {     sum.EventIDj = W; // calculate weighting
             time.EventIDj start to count down; j+1; // index to identify
       }                                              //different event
       else if (EventID==EventIDj)// which have been received
       {     sum.EventIDj += Wi;// accumulate weight for the
             // same event but from different nodes(differed by index i)
             while (sum.EventIDj>threshold)
                  Cluster_Act(ClusterID,EventIDj);// send information
                                        //form cluster head to cluster head
       }
       while (time.EventIDj > waiting_time)
       call Adaptive_threshold(sum.EventIDj,EventID) / drop ;
  }
  else // if event is from cluster-head
  {    while (EventID=S_EventID[ ] which has been sent)
       {    if ( the ClusterID =S_ClusterID[ ] the same source cluster head)
                 drop;
            else
            {    call Cluster_Act(ClusterID,EventID)
                 add ClusterID into S_ClusterID[ ]; j+1; //record a new
            }                                    // source cluster head
       }
       call Cluster_Act (ClusterID,EventID);// sent event to next
       add ClusterID into S_ClusterID[ ];     // cluster-head
       add EventID into S_EventID[ ]; j+1;// record a new event
  }
}
while (node itself is not a cluster-head)
  do{   if (the source Id of the event have not been sent)
        {   add the SensorID itself into the SensorID array(SensorID[ ]);
            call Sensor_Act(SensorID[ ],EventID);
        }
  }
}
```

Summation part:

For cluster head: The cluster head would calculate the threshold with the recorded fault probabilities of the sensor nodes in the cluster at present. Once the first event received, the waiting time starts to count down. If there is no exactly the same event has been received from other source nodes through the waiting time in cluster head, this event could be dropped or calculate a adaptive threshold compare with the accumulated weight so far again to avoid losing event

with special case. On the other hand, if the threshold has been reached among the waiting time, the cluster head goes to the activation part and the fault probability of the source node of new event would be recorded in the cluster head.

For other sensor node: If the sensor node is first sensing the event, it will go into the activation part. Otherwise the sensor node (bypass node) will add its own ID in the sensor ID of the packet and then goes into the activation part.

3.4.2 Activation part

The activation algorithms are shown in Table II as follows:

TABLE II ALGORITHM OF ACTIVATION PART

```
ClusterH_Act(ClusterID,EventID) { //transmissions between cluster heads
        N_ClusterID = Lowest _error_of_nei_CH();
        // or N_ClusterID = Shortest_path_of_nei _CH();
        Forward_to_Sink(SensorID, EventID, ClusterID,N_ClusterID);
    }

Sensor_Act(SensorID[ ],EventID) { // transmissions in cluster
        DesID = Lowest _error_of_neighbors();
        // or DesID = Shortest_path_of_neighbors();
          Forward_to_localCH(SensorID[ ],DesID,EventID);
        }
```

Activation part:

For cluster head (ClusterH_Act**):** If only the cluster head goes into the activation part, the cluster would transmit the event to next cluster head, which might chosen via the shortest path oriental or the lowest fault probability oriental, even the cross consideration of these factors, including the energy factor. While the cluster head transmit the event to next one, the packet must have the IDs of source nodes, event, ID of the source cluster head, and the information of the cluster head, which including the bias that used to estimate fault and remaining energy capacity. This way, the sink would know the geographic location of the event, update the fault map of this cluster in the sink, and estimate if the source cluster head could available to continuously execute the task. If it could not, the sink may request the cluster to elect new one. The cluster head election issue has been accomplished in many literatures. The election issues are not discussed in this paper.

For other sensor node (Sensor_Act): If only the sensor node goes into the activation part, it would send the event to next sensor node, which could be the cluster head or normal sensor node. The next node might be chosen via different mechanisms.

3.5 Aggregation threshold

3.5.1 Threshold formulation

The weight, calculated by mapping function, can be treated like a correct probability of a sensor node, and the accumulated weight value can be treated like the correctness of the event. The definition and formulation of the threshold are followed.

Definition 3*: *Weight Threshold, Th, is a value that compared with the accumulated weight calculated by cluster head. If the accumulated weight greater than the threshold, cluster head transmits the event to sink. Since the threshold is compared with the weight of an event in cluster, formulating the average threshold via the average weight in cluster, average number of node that would sense the event, and the average hop counts in a cluster, as follows.

$$Th = \frac{N_s}{N_c} \times W_a^h \tag{4}$$

where : W_a is the average weight in the cluster
N_c would be the number of sensor nodes that would sense the event, in average
N_s : there are N_s nodes sensed event when an event happened in average.
h is the average hop counts in the cluster

3.5.2 Adaptive threshold

In real case, both the cluster size and the size of event region would affect the number of sensor node that senses the event in the cluster. Due to the uncertainty, the cluster head might lose some events that only sensed by few sensor nodes that are with low fault probability (high weight). Consequently we need an adaptive threshold to deal with these special cases. The equation, N_s/N_c, represents the average number of sensor node that would sense the event in a cluster. In the adaptive threshold mechanism, the cluster head evaluate this number. Before the cluster head drop the event that the accumulated weight is still not equal or greater than the threshold, the cluster head would exchange the average number, N_s/N_c, into the number of source node that the cluster head received so far to lower the threshold.

3.5.3 Threshold variation

The cluster head uses the error probability of all sensor nodes in the cluster to evaluate the *100%* threshold, which means the average threshold in the cluster. This threshold is easier to reach but it still has some kind of dependence. If the threshold is advanced, represent the event that sent by the cluster head is more dependable. If the *50%* threshold is used, represent the threshold is calculated via the front half number of sensor node that with the lower error probability, which means the higher threshold would be generated. Thus, the cluster head would need either the source node that with the lower error probability or more source node to increase the volume of weight to reach the *50%* threshold. The 50% threshold could probably arise the delay time, but the dependability of the event would increase oppositely.

3.6 Waiting time

To avoid the extra memory storage in the cluster head due to the fault information, setting up a waiting time is necessary. Since the factor *a* discussed in the mapping function has the attribute to vary the volume of the weight in the same accuracy (input), the factor could control the waiting time of the event.

Definition 4: *The Waiting time,* T_w, is just a bound in cluster to avoid the extra storage. The waiting time for one single node that sense event is formulated simply via the maximum hop counts in cluster, average computing time, average sensing time, and average transmission time, as follows.

$$T_w = T_s + h_c \times (T_t + T_c) + T\_CH_c$$

where T_s is the sensing time of the sensor node
h_c is the maximum hop counts in the cluster
T_t is the average transmission time including the idle and collision time
T_c is the average computing time
T_CH_c is the computing time of the cluster head

4 Simulation Results

4.1 Delay time vary average hop counts and threshold

In this simulation, the delay time caused by the average hop counts and varied threshold is shown in Fig. 3(a). In this model, factor *a =1*, one cluster head handles *10* nodes, because the average hop counts can be easily controlled. We can see that as the average hop counts larger, the delay time is longer. From the threshold aspect, as the percentage of threshold lower, the delay time is longer. As discuss about, the 100% threshold represents the average threshold in a whole cluster. The 50% threshold represents the front half number of sensor node with the lower error probability that means when the percentage is lower, the calculated threshold is higher. It is more difficult to reach the lower percentage threshold, thus the delay time is getting longer. The threshold can be varying by the cluster head, what if the 100% threshold value in the cluster head is below *0.5*, which means the whole cluster could possibly in an abominable situation, such as the humidity is high, represents the error probabilities of the nodes in the cluster are high.

The Fig. 3(b) represents the same situation as Fig 3(a) but factor *a =2*. Compare the Fig. 3(a) and Fig. 3(b), the delay time can be different via the adjustment of factor *a*. While the factor *a* is greater, the delay is shorter. The Fig. 3(c) with factor *a =3* has the same trend as Fig. 3(a) and Fig. 3(b). It proves that the factor *a*, in range *1~3*, can decrease the delay efficiently. The factor *a* can be varied by the cluster head, under some emergent case.

The delay time is small due to the simulated model. Since the cluster head handles *10* nodes, the distribution of the fault probability could not expend between value *0~1*. Consequently the difference of the *50%, 60%, 70%, 80%, 90%* and *100%* is small.

4.2 Delay time vary cluster size and threshold

In the section, we discuss the delay time caused by the cluster size and varied threshold when factor *a =1* is shown in Fig. 4(a). The lower percentage threshold accompanies the longer delay time especially in the larger number of cluster size. This could possibly cause by the distribution of the error probability. The distribution is more normal when in large cluster size, so the 50%, 60% even 70% could make the threshold more differ. Thus the delay time would be increased in the large size of cluster.

The Fig. 4(b) shows the same situation when factor *a = 2*. Although the cluster size increase, the delay time still decrease comparing with the Fig 4(a). The factor *a =3* in Fig. 4(c). Delay time could still decrease when factor *a* increase.

4.3 Delay time comparing with fully aggregation

Fully aggregation represents the all of the sensor nodes in a cluster, transmit the sensed event when an event happened. This simulation result is shown in Fig. 5:

We simply compare the fully aggregation delay time with the fault map based data aggregation delay time. In the fault map based data aggregation, the factor *a =1*, and varying the cluster size when the threshold are *100%* and *50%*. We can observe that the huge difference between them. The result represent that the fault map based data aggregation can transmit the aggregated event before the unbalance response time of all the sensor nodes in cluster. The huge delay time is due to the unbalance response time of sensor node in cluster.

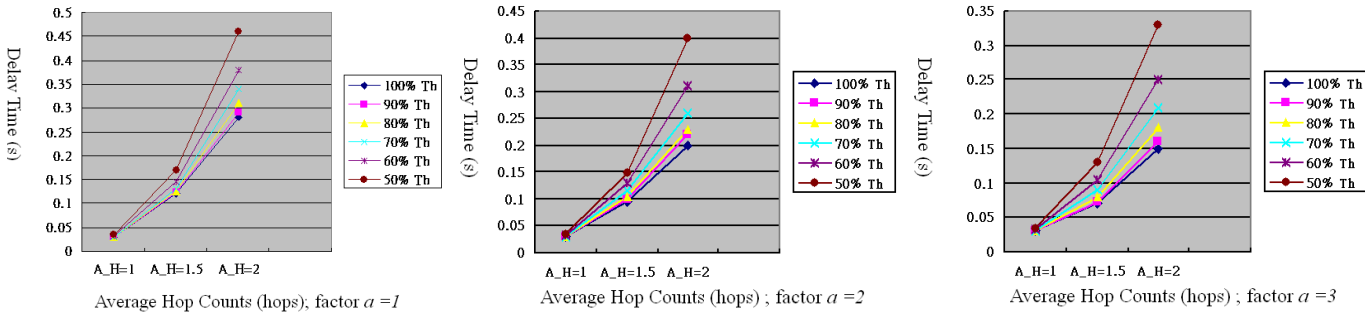
The delay time of fault map based data aggregation scheme greatly decrease comparing with the full aggregation time. Due to the uncertain response time of sensor nodes, it would affect delay time of the fully aggregation. This data aggregation scheme can reach comparative credibility via adjusting the threshold value, but without the fully aggregation response time.

5 Conclusions

We have shown the performance of the fault map based data aggregation on cluster based wireless sensor networks. By modeling the cluster head and other sensor nodes with the algorithms, we can enhance the comparative credibility of the aggregated event via varying the threshold.

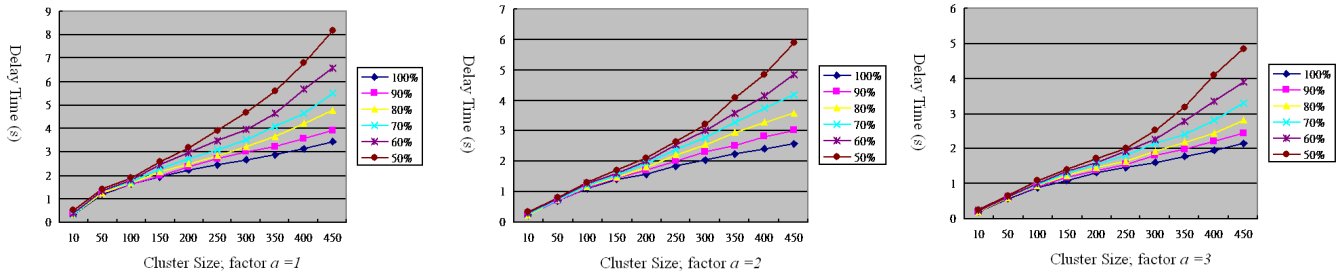
Although there could be some extra delay time comparing with the cluster architecture wireless sensor networks without aggregation mechanism, it is worth to increase the credibility of the aggregated

event in the unreliable and resource constraint wireless sensor networks. Besides, the parameter, factor *a*, could decrease the delay time. When an emergent event happened, we can increase the factor *a* to decrease delay time.



(a) (b) (c)

Fig. 3. Delay time varying average hop counts. (a) *a* =1 (b) *a* =2 (c) *a* =3



(a) (b) (c)

Fig. 4 delay time varying cluster size (a) *a* =1 (b) *a* =2 (c) *a* =3

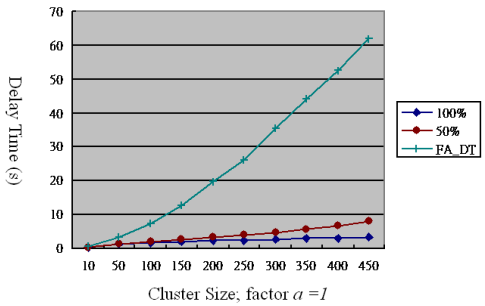


Fig. 5: Delay time comparing with fully aggregation

References:

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, "Wireless sensor networks: a survey," Elsevier Computer Networks 38 (2002) 393-422.

[2] Y.S. Chang, T.Y. Juang, C. J. Lo, M.T.Hsu, and J. H. Huang, "Fault Estimation and Fault Map Construction on Cluster-Based Wireless Sensor Networks," in Proc. of IEEE International Workshop on AHUC 2006, Taichung, Taiwan, Jun. 2006, pp. 14-19.

[3] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks," HICSS 2000, pp. 4-7, January 2000.

[4] C. Intannagonwiwat, D. Estrin, R. Govindan, and J. Heidemann, "Impact of Network Density on Data aggregation in Wireless Sensor Networks." IEEE International Conference on Distributed Computing System, 2002

[5] Fei Hu, Xiaojun Cao, and Carter May, "Optimized Scheduling for Data Aggregation in Wireless Sensor Networks," IEEE International Conference on Networking, Sensing and Control, April 2005.

[6] Rajgopal Kannan and S. Sitharama Iyengar, "Game-Theoretic Models for Reliable Path-Length and Energy-Constrained Routing With Data Aggregation in Wireless Sensor Networks," IEEE Journal on Selected Areas in Communication," vol. 22, no 6, pp. 1141-1150, August, 2004.

[7] Bin Zhou, Lek Heng NGOH, Bu Sung Lee, Cheng Peng Fu, "A Hierarchical Scheme for Data Aggregation in Sensor Network." 2004. (ICON 2004). IEEE International Conference. pp.525 – 529.

[8] Sang-ryul Shin, Jong-il Lee, Jang-woon Baek, Dae-wha Seo, "Reliable Data Aggregation Protocol for Ad-hoc Sensor Network Environments," Proc. on ICACT 2006, Feb 20-22 2006, pp.531-534.