Optimization on Distributed User Management in Wireless Sensor Networks

MingJian Tang and Jinli Cao
Department of Computer Science and Computer Engineering
La Trobe University
Melbourne, Australia-3086
mj2tang@students.latrobe.edu.au and j.cao@latrobe.edu.au

Abstract

In this paper, we address one of the Wireless Sensor Network (WSN) management problems – optimization on the execution of multiple commands. The objective of the paper is to provide efficient support for pre-processing a set of commands before disseminating into the sensor network. It is important that only necessary work will be assigned to the sensor network by virtue of strict energy constraint. The problem is NP-hard. We divide the problem into a series of tractable sub-problems along with their solutions. We present a novel Hierarchical Quadrant-based field Partition mechanism to virtually divide the sensor field. We also classify and model WSN management commands. We then identify merging possibilities for a given command, which results in several merging rules and constraints. Lastly, we evaluate a Simulated Annealing based search algorithm for finding optimal merge order. The results show that energy can be significantly saved within short time-delay while the overall effects of the final command set still satisfies the users' requirements.

Keywords: Wireless Sensor Networks, Multi-Command Optimization, Hierarchical Quadrant-based, Heuristics

1. Introduction

We have witnessed the recent rapid growth of the Wireless Sensor Network (WSN) applications, for instance, the information gathering, enemy detection, and environment monitoring [2]. Typically, WSN is formed by sensor nodes. Each of nodes is capable of limited computation, communication, and sensing. It can be expected that with more sensor nodes scattered around, the WSN can be more powerful in terms of providing fine-grained surrounding information. However, the growing number of sensor nodes can make the network more complex to manage. Therefore, we can envision that multiple network administrators are required for a large-scale WSN. They will work either cooperatively or separately by sending management commands to a centralized management station (base-station), which serves as the interface for communicating with the WSN. Nevertheless, this distributed management environment will potentially cause redundant administrative workloads. In addition, the unique characteristics of the WSN, extreme energy constraint, densely deployment, multi-hop, and broadcast, make managing the network itself a big challenge.

The key issue is identified as how to optimally pre-process a set of management commands so that the workload can be minimized. The major contributions of the paper are:

1) Problem decomposition: The original Multi-Command Optimization (MCO) problem is proved to be NP-hard. We decompose the problem into several tractable sub-problems along with the solutions.

2) Virtual field partition mechanism: Since information gathered from the sensor network are highly geography-correlated, a cost-effective hierarchical quadrant-based field partitioning (HQP) mechanism is proposed so that it can successfully and cost-effectively regulate the shapes of merged ranges.

3) A heuristic search algorithm: A Simulated Annealing (SA) [9, 18, 19] based optimization algorithm is proposed to one of the sub-problem. It guarantees good performance towards finding good command merging order.

The rest of the paper is organized as follows: section 2 discusses the architecture of the WSN and its management environment. The problem is then formulated along with the problem analysis. HQP scheme is presented in Section 3. Section 4 discusses the management command classification and modeling. Section 5 investigates the challenging issues of MCO along with solutions. Section 6 presents the experiment and evaluation study on the proposed algorithm. Section 7 relates the background of research. Finally, the conclusions are conducted in Section 8.

2. Problem Formulation and Analysis

We consider a flat WSN with a large number of sensor nodes, where the sensor nodes are uniformly deployed in the sensing field and they are self-organized into a connected network. From the architecture point of view, a typical WSN is two-tiered, which are in-network tier and out-network tier. The in-network tier consists of the sensor nodes. The base-station forms the out-network tier. The in-network resources are highly constrained in terms of energy, communication, and computation capabilities, whereas we assume much more abundant out-network resources.

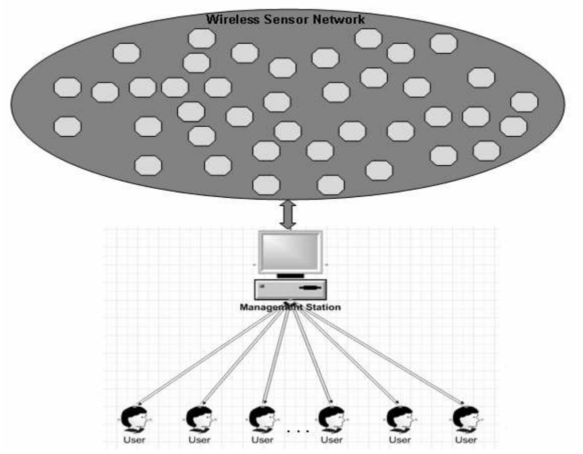

Figure 1. WSN management scenario

Figure 1 depicts a typical sensor network management scenario. There are multiple users interacting with a centralized management station through high-speed network facilities. These users can issue management commands anytime. These commands will then be transferred to the management station, where they will be queued and pre-processed before disseminating to the WSN. The management station communicates with WSN via wireless communication channel. The above WSN environment settings will be used in the rest of the paper.

2.1. Problem Formulation

The MCO problem is formally formulated as follows:

- Let $U = \{U_i \mid 1\leq i\leq n,\ i \in N\}$ be a set of management commands.
- Let *Cost:* $U\rightarrow R$ be a cost function, which assigns a real number to each management command to reflect their incurred energy cost.
- Let $W = \{T_i \mid 1\leq j \leq n,\ i \in N\}$ be a workload cover that contains a set of tasks T.
- Let $T_i \rightarrow D^m$ be the task that reflects the effect of user defined command. E.g. D^4 = *{<Attribute, Range, Interval, TimeWindow>|Attribute = NodeCount, Range = <(100, 50), (200, 100)>, Interval = 10, TimeWindow = 200}.*
- Let $Cost(U) = \sum_{i=1}^{n} Cost(Ui)$ be the overall cost of set U.
- Let $W(U) = \sum_{i=1}^{n} W(Ui)$ be the overall workload cover of set U.
- Let S be the collection of all possible command sets derived from U that cover $W(U)$.

In order to minimize *Cost(U)*, we have to rewrite command set U into a new set C, where $C = \{C_j \mid 1\leq j\leq m\leq n,\ j\in N\}$. The new set C should also keep the following constraints:

- $W(U) = W(C)$
- $W(C_i) \cap W(C_j) = \varnothing \quad 1\leq i\leq j\leq m\ \ \&\ C_i, C_i \in C$
- $Cost(C)\leq Cost(U)$, *if* $\exists C'$ *such that* $Cost(C') \leq Cost(U) \Rightarrow \forall C$ *that* $Cost(C) < Cost(C')$
- $C \subseteq S$

To find an optimal transferring strategy (with the minimum cost) for U in general is equivalent to Minimum Set Cover (MSC). A formulation of MSC is described by [19] as:

- Instance: Given a universe U and a collection S of subsets of U, a set cover is a sub-collection $C \subseteq S$ of sets whose union is U. In the set cover optimization problem, the input is a pair *(U, S)*.
- Find: A set cover $C \subseteq S$ which contains the fewest sets to cover U.

MSC is proved to be NP-hard [8]. That is, we have to use heuristics to find an approximate optimal solution.

2.2. Problem Analysis

One of the challenging problems is that management commands in WSN are mostly geography-correlated. Commands issued by multiple users are most likely in an ad hoc style, which means that the overlapped areas can have arbitrary shapes. Merging multiple commands with these shapes becomes very computation-intensive. Therefore, identifying the overlapping in geographical ranges plays a vital role in the optimization evaluation. We propose a cost-efficient field division mechanism for easing the problem. Details will be provided in section 3.

The MCO also implies a Combinatorial Optimization Problem (COP), which is finding a good command merging order. The COP formulation is given by [1]:

"Given a finite configuration space (solution space or space of configurations) $S = \{x \mid x = (x_1, x_2 \ldots x_m)\}$, *where m is called the dimension of the space, and a cost function* $C: S\rightarrow R$, *which assigns a real number to each configuration, we want to find an optimum configuration* $x^* \in S$, *such that all* $y\in S$, $C(x^*) \leq C(y)$*."*

We can map the set of all sets of command merging orders to the solution space S and cost model to the proposed cost function. The objective of MCO is to find an optimal merging order that minimizes total cost. It is similar to the goal of COP. Generally speaking, COP is also considered as NP-hard. Section 4 and 5 will solve this problem.

3. Hierarchical Quadrant-based Field Partition (HQP)

In order to efficiently support the multiple range-based commands in WSN, we propose a HQP mechanism, which iteratively and uniformly divides the sensor field.

We consider that the WSN is deployed in a *2D* plane. The plane is partitioned by decomposing the region into four equal quads, sub-quads, and so on. We use four directional symbols for representing the partitioned quads, which are *N-north*, *S-south*, *W-west*, and *E-east*. We can combine any two of them to describe a particular quad, such as *NW*, which is the upper left quad of the deployment field and stands for Northwest quad. The dot expression is adopted for representing the hierarchy among quads and sub-quads. For example, *NE.SW* represents the Southwest quad that is relative to the Northeast quad of the field. The level of the quad structure can be determined according to the user requirement. The partition mechanism is applied at the base station. The base station works as an interpreter between the management command processor and user posed commands. It converts the directional symbols into particular ranges. For example, the sensor field is *400m×400m*, and a user issues a command asking for all the sensor nodes residual energy levels in the *NW.SE* region. The base station will process the command and convert *NW.SE* into pair of *XY* coordinates as *<(100,200),(200, 300)>* indicating that the range of the *X* coordinate is from *100m* to *200m* and the range of the *Y* coordinate is from *200m* to *300m*. The converted commands are consistent with the cost model we presented later, in which *XY* coordinates are used for range representation.

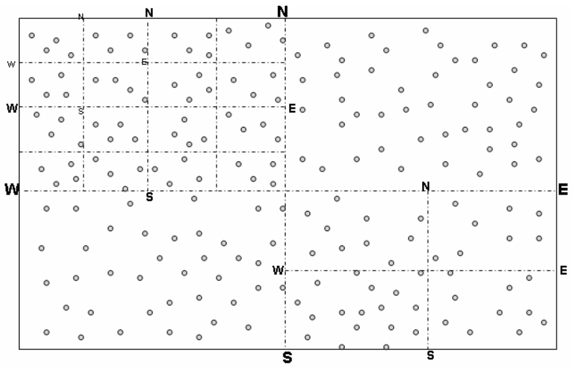

Figure 2. A sample sensor network deployment

The proposed mechanism is very cost-efficient since the field is only virtually divided and all division related works are done at the base station, there will be no in-network managerial overhead. It is also very flexible and scalable because the level of the hierarchy is defined as a user input. In addition, any change will only affect the out-network tier rather than the in-network tier. Figure 2 shows that the sensor field is divided according to HQP. The plane is equally divided into four quads. The *NW* quad is further divided into *16* sub-quads, whereas the *SE* quad is further decomposed into *4* sub-quads. To simplify the presentation, we only show an equal field partition. However, our mechanism can be easily extended to cope with inequi-partitions.

4. Management Command Taxonomy and Modeling

We discuss the classification and modeling of management commands in this section. The features of each category are extracted, and cost models are proposed to quantify different commands.

4.1. Monitoring-Oriented Command (Mono)

The Monos are issued for collecting management information, such as the node's residual energy, network connectivity, node count, and so on. It is similar to database queries which demanded management information is pulled by these commands. The structure of Mono is as follows:

<Attribute[i]> <Range> <Time Window> <Interval>

The *Attribute[i]* defines a number of management interests expressed by the user, such as the residual energy level of each node. The *Range* attribute is used for defining the size of the queried area and it also indicates the spatial granularity of the monitoring. For example, the possible *Range* structure can be < $(X_l, Y_l), (X_u, Y_u)$>, where X_l and X_u pair defines the horizontal range and Y_l and Y_u pair defines the vertical range. These two pairs together identify a rectangle-shaped monitoring area. *Time Window* attribute is in minutes, which rules how long a Mono will be valid for. For example, if *Time Window* is equal to *100*, then the monitoring actions will last for *100* minutes. There will be some propagation delays due to different factors, but we assume that the delays will be small enough to be negligible compared with the values of *Time Window*. The last element is the *Interval*, which is represented by number of minutes. It sets the temporal granularity of the monitoring. For example, with value *2*, then every two minutes monitored nodes will be activated for reporting the required management information back to the management station. An example of a Mono is shown below:

<N.Count, N.Energy><(10, 20), (100, 30)> <200><20>

The above example shows the user is interested in investigating node count and node residual energy level within the range of *X: 10 → 100m* and *Y: 20 → 30m*, and the command will be valid for *200* minutes and the reporting intensity is *20* minutes per report.

4.2. Management-Oriented Command (Mano)

The Mano differs from Mono in the way that it carries certain user actions instead of user requests. The typical structure of a Mano is shown as follows:

<*Condition*> <*Plan*> <*Range*> <*Time Window*>

The Mano consists of four parts, which are *Condition, Plan*, *Range* and *Time Window*. The *Condition* defines the trigger for executing the management plan. The *Plan* carries actual management plan, such as reboot sensor node or change the working mode of sensor node. The *Range* and *Time Window* are essentially the same as the ones in Mono model, which define the spatial and temporal requirements of a Mano. The following example shows a typical Mano.

<*N.Energy* ≤ *20%*><*N.Mode.Sleep*><*(30, 10), (70, 20)*> <*300*>

The example shows that within the range of $X{:}\ 30 \longrightarrow 70m$ and $Y{:}\ 10 \longrightarrow 20m$, if any node residual energy level is found to be less than or equal to *20%* of their initial values, then these nodes will be put into sleep mode. The command will be valid for *300* minutes.

4.3. Cost Models

The cost model of Mono is described by the following formula:

$$Cost(Mono) = \frac{\sqrt{(X_u - X_l)^2 + (Y_u - Y_l)^2} * |Attributes| * TimeWindow}{Interval} \quad (1)$$

where square root in formula *(1)* is the length of the diagonal of coverage size of the Mono. The longer the diagonal length, the larger the coverage is. |*Attributes*| is the number of monitored attributes involved in a Mono. A Larger number of attributes involved in a command indicates more costs are needed to fulfill the requirement. For sake of simplicity, we use a uniform cost value for modeling all types of attribute collections. They can easily be modeled differently by assigning a ratio to each attribute, and then multiply with a pre-defined base value. We divide the time window by the interval, which is used to quantify number of information units that needs to be transferred back to the management station. For example, if the result of the division is *4*, it means that *4* units of information are required. These factors are multiplied together to form the cost model for Mono in the end.

Similarly, the cost model for Mano can be formulated as follows:

$$Cost(Mano) = \sqrt{(X_u - X_l)^2 + (Y_u - Y_l)^2} * |Plan| * TimeWindow * |Condition| \quad (2)$$

The cost of Mano is formulated by multiplying number of plans, number of conditions, value of Time Window, and the length of diagonal together. One may argue that the condition attribute is like selectivity in traditional DB, so more conditions applied will add more constraints on selection and results in less energy consumption. However, we argue that more conditions will make sensor nodes busier monitoring themselves so that any cross boundary condition can be detected. Therefore, more conditions will still lead to higher energy consumption. Similar reason is applied for using number of plans in the cost model.

In this paper, we limit the optimization on multiple Monos only, but it can be easily extended to cope with multiple Manos as well.

5. Multi-Command Optimization (MCO)

In this section, we concentrate on optimizing multiple management commands. The MCO is used to pre-process the original management set *U* before disseminating into the WSN. Therefore, redundant works can be eliminated, and at the same time the new management set *C* also covers *U*. The problem is NP-hard. In order to find an approximate solution within polynomial time, we first exploit Divide-and-Conquer approach to break the MCO into two phases. In the first phase, a series of rules and constraints will be defined to identify all potential commands that can be merged with a given command U_{ran}. At this phase, we don't specify merging order, so there might appear multiple solutions for merging U_{ran}. The result from this phase will be a number of general solutions for merging a given command. In phase two, the focus is on how to find a close to optimal solution among those solutions. The problem is tackled by using heuristics. A Simulated Annealing based search algorithm is proposed for finding the new management set *C* that minimizes the cost.

5.1. Merging Rules and Constraints

We define a series of merging rules and constraints for examining the merging possibility between two management commands. The heuristic algorithm exploits them to identify possible adjacent merging orders. These rules and constraints are only applicable to two commands with range overlapping. Otherwise, we consider them to be disjunct and assume they don't have merging possibility.

The main goal of command merging is that it can lead to cost reduction. We can divide command merging into two types namely Cost-Addition (CA) merge and Non-Cost-Addition (NCA) merge based on the cost changes. For example, there are three commands along with their costs below:

Q_1: *Select count, energy from sensors where location = SE every 10 mins for100 mins*

$$Cost(Q_1) = 2 * 4 * 10 = 80$$

Q_2: *Select count from sensors where location = SE.NW every 20 mins for 100 mins*

$$Cost(Q_2) = 1 * 1 * 5 = 5$$

Q_3: *Select energy from sensors where location = SE.NE every 5 mins for 100 mins*

$$Cost(Q_3) = 1 * 1 * 20 = 20$$

where Q_1 and Q_2 have merging possibility, and Q_1 and Q_3 have merging possibility. If they are merged accordingly with each other, two new commands Q_{12} and Q_{13} will be generated as follows:

Q_{12}: *Select count, energy from sensors where location = SE every 10 mins for 100 mins*

$$Cost(Q_{12}) = 2 * 4 * 10 = 80$$

Q_{13}: *Select count, energy from sensors where location = SE every 5 mins for 100 mins*

$$Cost(Q13) = 2 * 4 * 20 = 160$$

Clearly the merging possibility between Q_1 and Q_2 belongs to the NCA merge because $Cost(Q_{12}) < Cost(Q_1) + Cost(Q_2)$, whereas merging Q_1 and Q_3 is a CA merge because $Cost(Q_{13}) > Cost(Q_1) + Cost(Q_3)$. The NCA merges not only reduce their workloads but also reduce the number of commands, so they can guarantee beneficial gains. On the other hand, CA merges are not prohibited because they can reduce the number of commands. Reducing the total number of commands is important due to several reasons. Firstly, more commands disseminations mean more in-network cost consumption. Secondly, more commands will also cause higher network resource contention, such as bandwidth. Lastly, more commands will also make the network less reliable. Thus it is uncertain whether a CA merge is beneficial or not. Here we import a user-defined parameter θ which serves as a constraint for determining whether the merge is beneficial or not. Hence, instead of only merging NCA commands, we will consider any merge that incurs less than θ CA to be beneficial. The formal definition for the Beneficial Merging Rule (BMR) is as follows:

***BMR**: Given two commands and a user-defined parameter θ, merging is beneficial iif the cost addition incurred from the merged commands is less than θ.*

The merged commands should not lose any workload that the original commands cover. It is another important merging evaluation factor. It requires that the merged commands still meet the users' management requirements. Formally we define the Workload Cover Rule (WCR) as below:

***WCR**: Given two commands U_x and U_y, and their workload covers $W(U_x)$ and $W(U_y)$ respectively. They are merge-able iif the workload of the merged command/commands $W(M(U_x, U_y)) = W(U_x) \oplus W(U_y)$.*

where $\oplus$ is the exclusive addition that only considers redundant parts once.

The merged commands should be structurally consistent and syntactically correct. A typical management command consists of four parts namely *Attributes*, *Range*, *Time Window*, and *Interval*. In terms of the structural integrity, a management command must contain all the aforementioned attributes. In terms of the syntactic correctness, four dimensional attributes should be consistent. The consistency of a management command can be defined by Consistency Rule (CR) as follows:

***CR:** A management command is said to be consistent iff it contains one and only one Interval value and Time window value, which are applied on one or multiple Ranges and Attributes.*

Eventually we can derive a merging constraint from combining the above three merging rules and the constraint definition is given below:

***Merging Constraint**: Two commands are considered to be merge-able iff the merged command/commands obey the aforementioned three merging rules.*

5.2. Simulated Annealing Based Optimization Algorithm

SA is a widely used stochastic search algorithm that is applicable to COP. It is also a local search algorithm, but it differs from others by not only accepting cost-decrease transitions but also cost-increase ones. Therefore, it has the ability to escape from local optima. The ability is due to the utilization of an acceptance function, which calculates a probability to determine whether to take cost-increase transition or not. The common acceptance function is $exp(-\Delta G/T)$, where the ΔG is the gain difference between two transitions and T stands for temperature used as a control parameter. T will initially be assigned with a big value so that more freedoms are given to move from one transition to the other, and then it will be decreased gradually until it reaches zero or small enough. Therefore, the choices for choosing a transition will be narrowed down and only good moves will be selected. For detailed discussions on SA concepts, we refer readers to [9] and [19].

The modified SA algorithm [18] is detailed as follows:

SA based Optimization Algorithm

```
INPUT: $U = \{U_1 \dots U_i\}$, $T = 100$
OUTPUT: $C$
START:
1: WHILE $U$ is not empty DO
2:    Select a random command $U_{ran}$ from $U$
3:    Find all the merging solutions $S$ for $U_{ran}$
4:    $U = U - U_{ran}$
5:    IF $|S| > 1$ THEN
6:    Select a random solution $s$ from $S$
7:    $S = S - s$
8:    Compute $G(s)$
9:    Set $S^* = s$ and Set $G^* = G(s)$
10:   WHILE TRUE or $S$ is not empty DO
11:      Select a random solution $s'$ from $S$
12:      Compute $G(s')$
13:      IF $G(s') \leq G(s)$ THEN
```

```
14:         Set s' to be the current solution s
15:         IF $G(s) \le G(s^*)$ THEN
16:             Replace $S^*$ with s and $G^*$ with G(s)
17:     ELSE
18:         Generate a random number r in [0, 1]
19:         $\Delta G = G(s') - G(s)$
20:         IF $r < exp(-\Delta G/T)$ THEN
21:             Set s' to be the current solution s
22:             $T = T * 0.8$
23:     IF G is improved by less than 0.1% for 10 consecutive series of L steps THEN
24:             FALSE
25:   END WHILE
26: ELSE
27:         Insert S* into C
38: END WHILE
END
```

The algorithm starts by picking a random management command U_{ran} from U, and then all the adjacent merging orders for U_{ran} will be derived based on the merging rules and constraints. If there exists one or no merging order for U_{ran}, then the SA process is skipped so that processing time can be saved and that merging order or just U_{ran} will be included in the final solution. If there are multiple merging orders (>*1*), then those merging orders will go through the SA process inside which a good merging order will be chosen and executed. When the algorithm converges, an optimal solution will be generated with a new set of management command that minimizes the workload. We map the set of adjacent merging orders to the set of neighbor states, and the gain function to the objective function.

The initial temperature *T* is set to *100*, and it decreases by *20%* every time the temperature updating function is called. There are some other control parameters used here, such as number of consecutive steps for detecting insignificant improvement on transition gain and the number of merging orders for U_{ran}.

6. Performance Evaluation

We examine the performance of the SA algorithm in this section. We have implemented an application program for this evaluation. The program mimics the procedure of collecting management commands from different users. Normal Distribution is used as the probability distribution model for command set generation. Then, these commands will be optimized by the SA algorithm. The evaluation statistics are collected during the command processing procedure, and they are examined using different metrics.

6.1. Command Generation and Experimental Setup

The command generation acts virtually on behalf of users providing sample commands to the system. Since data in WSN environment are highly geography-correlated, hot spots are inevitably formed where users are more likely to send their management commands. It is an important and unique factor that should be considered while we generating sample commands. Hence, we exploit a Normal Distribution model for sample command generation. Table 1 shows the command settings that are used to form sample commands.

Parameters	Range of Values
No. of Attributes	*1,2,3*
Attributes	*A,B,C,D,E,F,G,H,I,J*
Hierarchy Level	*1, 2,3*
Interval	*5,10,20*
Time Window	*100, 200, 300*

Table 1: Command attributes settings

In order to simplify the process of the simulation and at the same time also keep the generality of testing sets, we regulate a command can have up to *3* attributes from *"A"* to *"J"*. They are used for symbolically representing the real sensor attributes. Theoretically interval values and time window values can be any integer. Due to the limited space we choose some sample values for the simulation study, which are adequate for evaluating the optimization algorithm. Hierarchy level relates to the size of the command location. The sensor field in this case is divided into *64* uniform grids, which is equivalent to three-level hierarchy. Each command can be assigned with a location level valued from *1* to *3*, where the smaller the number is and the bigger the grid will be. For example, a grid of *level-1* is equal to *16* grids of *level-3*. Sample commands are synthetically generated from combining the above parameters, and the probability distribution model used is Normal Distribution.

We have implemented the application program in C#. The hardware available for the experiments was a desktop computer with Pentium IV (*3.2GHz* CPU with *2* MB on-chip cache) and *2* GB ram. This is a typical computer nowadays with moderate hardware configuration that most likely a management station will be equipped with. All tests were run on this machine. We used Microsoft Visual Studio 2005 on Windows XP as the application run-time environment.

Parameters	Values
θ	*0*
T (Temperature)	*100*
T's updating rate	*0.8*

Table 2: Algorithm attributes settings

Sample commands were synthetically generated for providing inputs to compare the optimization algorithm under a range of application scenarios. We ran each scenario repeatedly for *20* times, and the final evaluation result will be derived from averaging *20* results obtained each time.

Despite the command settings, additional settings are needed for the algorithm, which are shown in Table 2.

θ is the user-defined tolerance value for considering a merging as beneficial. It will influence the command number reduction and the cost savings. Due to the limited space, we use value *0* so that negative cost savings are not taken as potential solution. *T* (temperature) and *T*'s updating rate are both used in the SA algorithm. *T* controls the freedom of the SA algorithm in terms of accepting a merge order, whereas *T's* updating rate controls the converging rate of the algorithm.

6.2. Reduction Ratio

Reduction ratio further consists of two metrics, namely cost reduction and command number reduction. The percentage of the cost reduction is expressed as the ratio of the cost before optimization on command set and after optimization on command set. The percentage of command number reduction is expressed as the ratio of the number of commands before optimization and after optimization. We also observe the impact of varying number of commands on them.

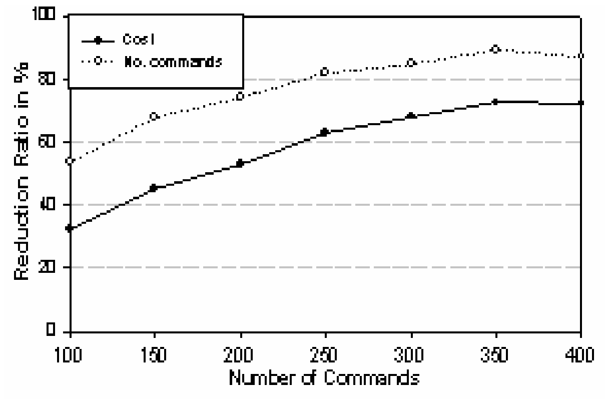

Figure 3. Reduction ratio (cost and command number)

Figure 3 shows two reduction ratios, and it proves SA is very effectively in terms of eliminating command redundancies. The averaged cost reduction is about *58%*, whereas the averaged command number reduction is about *76%*.

6.3. Execution Time

We use execution time to evaluate the temporal efficiencies of the algorithm. We examine the algorithm by providing it with varying number of commands, and execution times are derived from these experimental settings. We assume that user acceptable delay will be less than *20* seconds, and we will stop experimenting after this limit is reached by the algorithm. The end result can be used to demonstrate the processing capacity of the algorithm in terms of number of commands.

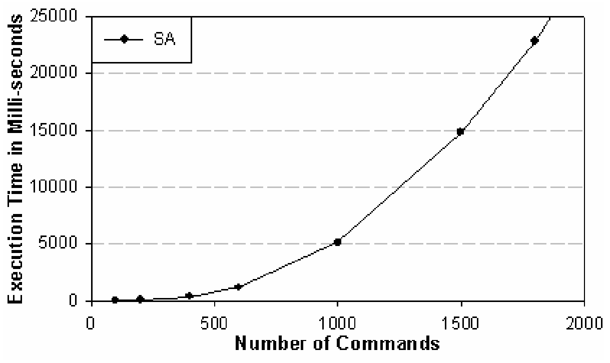

Figure 4. Execution time

Figure 4 shows the results of SA execution times that are observed through a number of experiments. It can be seen that the algorithm is extremely fast while processing up to *500* commands at a time. With the increasing number of commands, the algorithm execution time does grow asymptotically, and eventually reaches its limit which is close to *1800* commands.

7. Background and Related Work

Our problem is essentially related to Multiple-Query Optimization (MQO) problems in traditional database system. Many researches in [3, 4, 5, 6, 13, 14] were to find the common expressions or sub-expressions among a group of concurrent queries; and then those commonalities are exploited so that the execution times can be reduced by running repeated query parts only once (result sharing). This kind of problem is proved to be NP-hard, and some heuristics were proposed for providing an approximate solution [12].

Recently, with the growth of the emerging technologies, the MQO problems are also found in new domains, such as in multicast environment [7], in mobile database domain [11], in data stream systems [15, 16], and eventually WSN environment [10, 17, 21].

In [17], result sharing and result encoding techniques are proposed for processing multiple queries. The main mechanism behind their techniques is the use of linear reduction so that only basis is sent instead of full range of values. Their research work focus on reducing the in-network communications. A Fjords architecture [10] is another example of managing multiple queries, and the focus is also on in-network information processing and control.

In [21], a two-tier multiple-query optimization scheme is proposed for processing multiple queries in WSN. The base-station tier and in-network tier are the two tiers of the scheme. The base-station tier pre-processes queries sequentially identifying and removing query redundancies, and the in-network tier again helps to reduce the in-network energy cost by utilizing a better Directed Acyclic Graph. Though their base-station tier works well for multiple sequential queries processing, it might not work for multiple concurrent queries optimization.

By comparison, the contribution of our work is to efficiently pre-process multiple management commands for WSN. It is a base-station approach, and all the processing works are done on the base station only so there will incur no extra in-network cost. Our work differs from [21] in the way that emphasis is given on the actual processing of multiple concurrent commands, where a solid algorithm is provided. In addition, our work can be applied on top of those previously mentioned in-network techniques. Lastly, our work is deployable for those query processing based systems as well.

8. Conclusion

We identified the challenging problems in distributed user management in WSN environment, which were essentially to optimize multiple commands and remove redundant administrative management overheads. The problem was formulated and modeled as MSC. We divided and conquered the problem gradually by applying a series of solutions. Firstly, we presented a novel HQP mechanism so that the resultant geographical shapes could be controlled and arbitrary shapes could be avoided. This essentially eased the computation burdens on processing these complex shapes. Secondly, we classified WSN management commands into two types along with the respective cost models. Thirdly, we decomposed MCO into two phases. The first phase, we proposed a series of merging rules and constraints. The second phase, we proposed a SA based optimization algorithm for finding good merge orders. Finally, our experiments proved that the algorithm has good performance and can efficiently reduce the redundant workloads among multiple commands. The energy reduction was enormous.

References

[1] E. Aarts and J. K. Lenstra, Local Search in Combinatorial Optimization: Chichester[England]; New York: Wiley, 1997.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on sensor networks," IEEE Communications Magazine, vol. 40(8), pp. 102-114, 2002.

[3] W. G. Aref and H. Samet, "Optimization strategies for spatial query processing," presented at Proceedings of the 17th International Conference on Very Large Databases (VLDB), Barcelona, Spain, 1991.

[4] S. Chakravarthy, "Divide and conquer: A basis for augmenting a conventional query optimizer with multiple query-processing capabilities," presented at Proceedings of the Seventh International Conference on Data Engineering, Kobe, Japan, 1991.

[5] U. S. Chakravarthy and J. Minker, "Multiple Query Processing in Deductive Databases using Query Graphs," presented at Proceedings of the 12th International Conference on Very Large Data Bases, 1986.

[6] F.-C. F. Chen and M. H. Dunham, "Common Sub-expression Processing in Multiple-Query Processing," IEEE Transactions on Knowledge and Data Engineering, vol. 10, pp. 493 - 499, 1998.

[7] A. Crespo, O. Buyukkokten, and H. Garcia-Molina, "Query merging: improving query subscription processing in a multicast environment," IEEE Transactions on Knowledge and Data Engineering, vol. 15, pp. 174- 191, 2003.

[8] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Hardness: W H Freeman & Co, 1990.

[9] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," Science, vol. 220, pp. 671-680, 1983.

[10] S. Madden and M. Franklin, "Fjording the Stream: An Architecture for Queries over Streaming Sensor Data," presented at the18th International Conference on Data Engineering (ICDE), San Jose, California, USA, 2002.

[11] R. Malladi and K. C. Davis, "Applying Multiple Query Optimization in Mobile Databases," presented at Proceedings of the 36th Annual Hawaii International Conference on System Sciences, 2003.

[12] B. Nam, H. Andrade, and A. Sussman, "Multiple Range Query Optimization with Distributed Cache Indexing," presented at Supercomputing, 2006. SC '06. Proceedings of the ACM/IEEE SC 2006 Conference, Tampa, FL, USA, 2006.

[13] T. Sellis and S. Ghosh, "On the Multiple-Query Optimization Problem," IEEE Transactions on Knowledge and Data Engineering, vol. 2, pp. 262-266, 1990.

[14] T. K. Sellis, "Multiple-Query Optimization," ACM Transactions on Database Systems (TODS), vol. 13, pp. 23 - 52, 1988.

[15] S. Seshadri, V. Kumar, and B. F. Cooper, "Optimizing Multiple Queries in Distributed Data Stream Systems," presented at Proceedings. 22nd International Conference on Data Engineering Workshops, 2006.

[16] Y.-K. Suh, J. H. Son, and M. H. Kim, "GAGPC: Optimization of Multiple Continuous Queries on Data Streams," presented at Proceedings of the 24th IASTED International Conference on Database and Applications, Innsbruck, Austria, 2006.

[17] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman, "Multi-query Optimization for Sensor Networks," in International Conference on Distributed Computing in Sensor Systems (DCOSS), 2005.

[18] D. Turgut, B. Turgut, R. Elmasri, and T. V. Le, "Optimizing clustering algorithm in mobile ad hoc networks using simulated annealing," presented at Wireless Communications and Networking (WCNC 2003). IEEE, 2003.

[19] R. V. V. Vidal, Applied simulated annealing. Berlin: New York: Springer-Verlag, 1993.

[20] Wikipedia, "Set cover problem." http://en.wikipedia.org/wiki/Set_cover_problem

[21] S. l. Xiang, H. B. Lim, K. L. Tan, and Y. Zhou, "Two-Tier Multiple Query Optimization for Sensor Networks," presented at the 27th International Conference on Distributed Computing Systems (ICDCS'07), Toronto, Canada, 2007.