ENERGY EFFICIENT SENSOR NETWORKS - ASSIGNMENT PROBLEM

Parijatham Santosh Kumar Vodela
pv8395@rit.edu

ABSTRACT
In people-centric sensor monitoring, smartphones act as mobile nodes in gathering sensor data. Usually a sensing application distributes different tasks to various mobile devices on what sensing information to collect and when to report back to the application. To collect these sensing information, mobile devices use sensors available on their devices such as GPS, cameras, etc. During this process of continuous sensor monitoring the application will drain out mobile node battery which leads to less number of people offering their mobile nodes for such sensor applications.

DEAMON [3] is an energy efficient sensor monitoring algorithm which uses a CNF equation and distributes it sub-clauses efficiently among different mobile nodes avoiding redundancy and an algorithm to monitor these sensor conditions with limited communication and sensing among nodes. In this project, we are going to discuss the assignment part and the different algorithms used and results obtained.

1. INTRODUCTION
A sensing application generates tasks in the form of a CNF equation consisting of sub-clauses. Each sub-clause signifies a sensing task. These tasks are distributed to various mobile nodes which use onboard sensors or sensors from neighboring devices to collect information and report back to the sensing application when the necessary condition is true. Here each mobile node which offers its sensors is called as a helper.

Example:
CNF equation(F) = (Wind Speed > 10) \(\cap\) (luminance = 5) \(\cap\) (humidity > 90) \(\cup\) (temperature > 100) \(\cup\) (temperature < 0)
Represented: \(x_{11}\) \(\cap\) \(x_{21}\) \(\cap\) \((x_{31} \cup x_{32} \cup x_{33})\)
Helper1 sensors: Wind speed, humidity, temperature
Helper2 sensors: Wind speed, luminance, temperature
Assignment: \(H_1\) : \(\{x_{11}, x_{31}\}\) \text{and} \(H_2\) : \(\{x_{21}, x_{32} \cup x_{33}\}\)

This assignment problem is a variant of weighted set cover problem and is NP complete. We try to work with different heuristics and compare their results.

In a task-based sensing model, the application can be present on a centralized system in which the application submits various tasks to a central task service which then distributes it to different mobile nodes. In a peer to peer architecture, the application may run on one of the mobile nodes and execute sensing tasks locally or distribute to different neighboring mobile nodes.

A Task is represented as T = (S, F, rR, tB, tE), here S is the total type of sensors that are needed to report. F is the CNF expression, rR is the reporting rate and how often to report, tB is the time to start recording and tE is when to end the recording.

2. PRIOR WORK
DEAMON-ASSIGN is the algorithm discuss in [1], basically it finds a helper with the smallest average per atom cost in every step until all the atoms are covered. This is a greedy heuristic which is asymptotically closer to the best possible solution. DEAMON-ASSIGN also has a collapse algorithm; it works by converting the newly covered atoms into sub-clauses every iteration. For example, if the newly covered atoms are \(x_{12}, x_{32}, x_{33}\) then the equation would be \(\{x_{12}, (x_{32} \cup x_{33})\}\).

This reduction reduces overheads of communication between helpers and masters.

3. PROJECT GOAL
Experiment with different heuristic algorithms in solving the assignment problem similar to set cover problem and comparing their cost achieved and time taken.

4. IMPLEMENTATION
4.1 Greedy Heuristic
Greedy heuristic makes a locally best choice at every step, hoping it would find the global optimum solution. In greedy for the set cover problem, at every step, it calculates the average cost of every helper (cost of the helper/ newly covered atoms) and chooses the helper with the lowest one and adds to the solution set. It does this process until all the atoms are covered.

4.2 Better Than Greedy Heuristic
Main Idea: The greedy algorithm starts with an empty solution and in each iteration it calculates the ratio of each
Algorithm 1 Greedy algorithm
A(F) = the atom set of F
Xh = set of atoms that sensors in Helper h can determine
C = null, Hc = H
1: while A(F) is not a subset of C do
   2:     h = min \{cost of Helper i / Number of newly added elements\}
   3:     Ch = Xh / C
   4:     Fh = Collapse(Ch, F)
   5:     Hc = Hc / \{h\}
   6:     C = C \cup Ch
7: end while
8: Output A = \{ Fh : h \in H \}

subset as the cost / weight (number of elements newly covered), the minimum ratio subset is added to the solution, this is done till all the elements are covered. In the Better than Greedy \[2\], it is a modification of the greedy algorithm where in each iteration we are allowed to withdraw a subset which was already covered. If we decide to withdraw any set, we pick new subsets that cover all items that were previously covered in the set removed.

- Example- H1(1,2,3) Cost - 10 , H2(4,5) Cost - 12, H3(1,2,3,4,5) Cost - 20
  - Greedy Algorithm picks up H1 and H2 in the solution, total cost 22
  - Better than Greedy Initially picks H1 and then regrets that step by replacing H1,H2 by H3, total cost 20

4.3 Tabu Search Heuristic
Tabu search is an optimization algorithm with an embedded heuristic to constrain from returning to previously visited states in a search space \[1\]. It basically keeps track of its previous states for a certain time. It also moves to the next state even though the step worsens, basically avoiding falling in a pit like the local maxima. There are three variants for Tabu search which we tried for this project
  - Remember only fixed number of states
  - Remember all the states visited
  - Increase the memory if cost is improving else decrease memory.

5. RESULTS
The following experiments are done with multiple helpers and sensor combination. Each helper has about 3-8 sensors chosen at random.

5.1 Tables and Graphs
The following table shows results obtained for 10 to 100 helpers

<table>
<thead>
<tr>
<th># of helpers</th>
<th># of Sensors</th>
<th>Greedy</th>
<th>Better Than Greedy</th>
<th>Tabu Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>0.003094</td>
<td>0.0089</td>
<td>0.076392</td>
</tr>
<tr>
<td>15</td>
<td>30</td>
<td>0.012182</td>
<td>0.056452</td>
<td>0.176624</td>
</tr>
<tr>
<td>20</td>
<td>40</td>
<td>0.02627</td>
<td>0.109509</td>
<td>0.564301</td>
</tr>
<tr>
<td>25</td>
<td>50</td>
<td>0.102746</td>
<td>0.209238</td>
<td>1.551199</td>
</tr>
<tr>
<td>30</td>
<td>60</td>
<td>0.083881</td>
<td>0.540215</td>
<td>3.346555</td>
</tr>
<tr>
<td>35</td>
<td>70</td>
<td>0.097802</td>
<td>0.862129</td>
<td>4.985493</td>
</tr>
<tr>
<td>40</td>
<td>80</td>
<td>0.192683</td>
<td>1.157675</td>
<td>12.148563</td>
</tr>
</tbody>
</table>

6. ANALYSIS

<table>
<thead>
<tr>
<th># of helpers</th>
<th># of Sensors</th>
<th>Greedy</th>
<th>Better Than Greedy</th>
<th>Tabu Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>90</td>
<td>0.25385</td>
<td>1.427775</td>
<td>15.142473</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>0.505873</td>
<td>2.77112</td>
<td>26.748212</td>
</tr>
<tr>
<td>55</td>
<td>110</td>
<td>0.340083</td>
<td>2.68536</td>
<td>37.11562</td>
</tr>
<tr>
<td>60</td>
<td>120</td>
<td>0.552657</td>
<td>4.143856</td>
<td>50.757865</td>
</tr>
<tr>
<td>70</td>
<td>140</td>
<td>0.965081</td>
<td>7.988677</td>
<td>100.405862</td>
</tr>
<tr>
<td>80</td>
<td>160</td>
<td>1.976784</td>
<td>15.022233</td>
<td>167.600386</td>
</tr>
<tr>
<td>90</td>
<td>180</td>
<td>3.51943</td>
<td>27.459389</td>
<td>379.991118</td>
</tr>
<tr>
<td>100</td>
<td>200</td>
<td>6.079703</td>
<td>44.685262</td>
<td>636.820624</td>
</tr>
</tbody>
</table>
Algorithm 2: Tabu Search Heuristic

```
init = Initial Solution
tBest = init
tabuList = []

1: while not stopping Condition() do
2:   neighbourList = []
3:   bestCandidate = null
4:   tNeighbor = randomly chosen neighbors (covers all sensors)
5:   for sCandidate in tNeighbor do
6:     if (!tabuList.contains(sCandidate)) and (cost(sCandidate) < cost(bestCandidate)) then
7:       bestCandidate = sCandidate
8:     end if
9:   end for
10:  init = bestCandidate
11:  if (cost(bestCandidate) > cost(tBest)) then
12:    tBest = bestCandidate
13:  end if
14:  tabuList.push(bestCandidate);
15:  if (tabuList.size > maxSize) then
16:    tabuList.removehighestCostCandidate();
17:  end if
18: end while
19: return tBest
```

Table 3: Number of helpers and cost recorded for each algorithm with 10 to 30 helpers

<table>
<thead>
<tr>
<th># of helpers</th>
<th># of sensors</th>
<th>Greedy</th>
<th>Better Than Greedy</th>
<th>Tabu Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>10634</td>
<td>9634</td>
<td>8921</td>
</tr>
<tr>
<td>15</td>
<td>30</td>
<td>21837</td>
<td>20837</td>
<td>17829</td>
</tr>
<tr>
<td>20</td>
<td>40</td>
<td>26505</td>
<td>26505</td>
<td>23505</td>
</tr>
<tr>
<td>25</td>
<td>50</td>
<td>45210</td>
<td>45210</td>
<td>43770</td>
</tr>
<tr>
<td>30</td>
<td>60</td>
<td>50142</td>
<td>48142</td>
<td>45775</td>
</tr>
</tbody>
</table>

Table 4: Number of helpers and cost recorded for each algorithm with 35 to 100 helpers

<table>
<thead>
<tr>
<th># of helpers</th>
<th># of sensors</th>
<th>Greedy</th>
<th>Better Than Greedy</th>
<th>Tabu Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>70</td>
<td>65864</td>
<td>65864</td>
<td>63460</td>
</tr>
<tr>
<td>40</td>
<td>80</td>
<td>89581</td>
<td>84180</td>
<td>84180</td>
</tr>
<tr>
<td>45</td>
<td>90</td>
<td>131624</td>
<td>117703</td>
<td>117703</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>167728</td>
<td>167728</td>
<td>163473</td>
</tr>
<tr>
<td>55</td>
<td>110</td>
<td>129695</td>
<td>129695</td>
<td>129695</td>
</tr>
<tr>
<td>60</td>
<td>120</td>
<td>159857</td>
<td>159857</td>
<td>159857</td>
</tr>
<tr>
<td>70</td>
<td>130</td>
<td>229194</td>
<td>229194</td>
<td>227627</td>
</tr>
<tr>
<td>80</td>
<td>140</td>
<td>348460</td>
<td>348460</td>
<td>348460</td>
</tr>
<tr>
<td>90</td>
<td>150</td>
<td>418059</td>
<td>418059</td>
<td>399867</td>
</tr>
<tr>
<td>100</td>
<td>200</td>
<td>535004</td>
<td>535004</td>
<td>530004</td>
</tr>
</tbody>
</table>

Figure 3: Tabu Search Algorithm

- Greedy has a good time performance as it just gives up the best helper at that point of time and is the clear winner compared to better than greedy and Tabu search.
- It terms of lowest cost, Tabu search has good results compared to greedy and better than greedy algorithms.

7. CHALLENGES

- Finding cost for large numbers of helpers
- Finding the best possible solution via brute force

8. FUTURE WORK

Based on the results of the assignment problem, integrating its solution with the monitoring problem.

Figure 4: Graph for 45 to 100 helpers-Time(sec).
9. REFERENCES

